CS270 - Lecture

28 Aug 1990
Tuesday

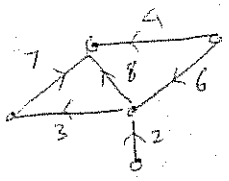
- INTRO. TO COMBINATORIAL COMPUTING
  FOUNDATIONS (TARJAN CH1)
    RANDOM ACCESS MACHINES
    TIME BOUNDS
    GROWTH RATES OF FUNCTIONS
- DFS & BFS
    MIN SPANNING TREE
    DISJOINT SETS (TARJAN CH2)
    HEAPS (TARJAN CH3)

Combinatorial Computing Problem
Selection of a configuration from a finite, but very large, set of possible configurations.

Optimization: Min^m Cost Config.
Feasibility: configuration that satisfies constraints
Enumeration: How many different configurations

Will need to develop data structures

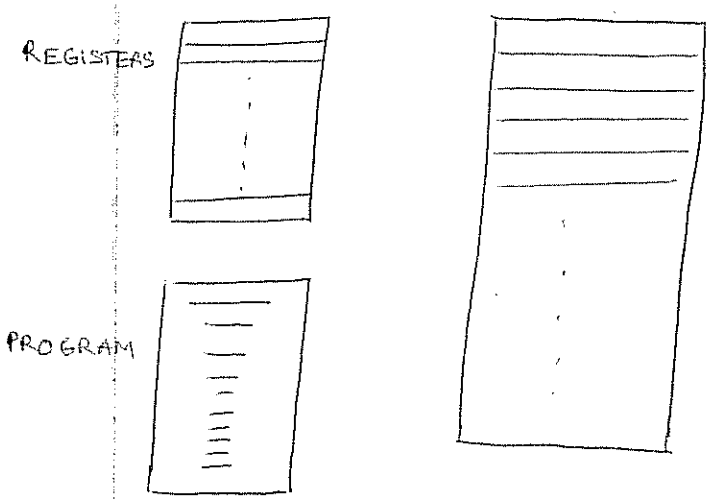
GRAPHS AND NETWORKS



- Min^m Spanning Tree
- Shortest Path
- Max Flow
- Min Cost
- Matching

TARJAN - Elegant monograph

Turing M/C - not suitable
∴ Random Access Machine



X_i contents of i^{th} location

INTEGERS stored in memory

$R_i := R_j + R_k$

$R_{R_i} := X_{R_j}$; indirect Addressing (allows ptr chasing)

$X_{R_j} := R_{R_i}$

$R_i := 17$

TRA m IF $R_j > 0$

Unit time comp on arb large integers leads to inconsistencies. (∵ sum is vector operations)

either ① bd integer sizes

② change move for large integers

Usually the problem instance has size n.

Require: There is a constant c s.t. on instance of size n all computed integers are $\leq n^c$ in abs. value

i.e. can be written in $c \log n$ bits;

$f: N \longrightarrow N$ ($N = \{$non neg integers$\}$)

ALGOR RUNS IN TIME f IF FOR ALL n AND ALL INSTANCES OF SIZE n THE NO OF STEPS is $\leq f(n)$ /* Not order i $3n^2 \neq 17n^2$ */

↳ This dfn leads us to worst case analysis.

Our model of comp is sequential rather than $//^{el}$ and deterministic not nondet.

Growth rates of fns

$f: N^k \longrightarrow N$ $\quad f(n_1, n_2, \ldots n_k)$

$f = O(g)$ if $\exists \; c_1 > 0, c_2 > 0$ s.t.
$\forall (n_1, \ldots n_k) \quad f(n_1, \ldots n_k) \leq c_1 \, g(n_1, \ldots n_k) + c_2$
↳ grows no faster than

$f = \Omega(g)$ if $g = O(f)$
↳ grows at least as fast as (upto constants)

$f = \Theta(g)$ if $f = O(g)$ and $f = \Omega(g)$
↳ grow at same rate

We ignore consts because by just changing m/c instruction set we can change things.

For fns of one variable:

$f(n) = o(g(n))$ if $f(n)/g(n) \to 0$ as $n \to \infty$
↳ grows more slowly than

$f(n) \sim g(n)$ if $f(n)/g(n) \xrightarrow[n \to \infty]{} 1$; Stirling's Formula $n! \sim \sqrt{2\pi n}\left(\frac{n}{e}\right)^n$

$G = [V, E]$ V is a finite set of vertices
E is a finite set of edges

Undirected $\{v, w\}$
Directed $[v, w]$
Ambiguous (v, w) ; when talking of dir & undir in same breath

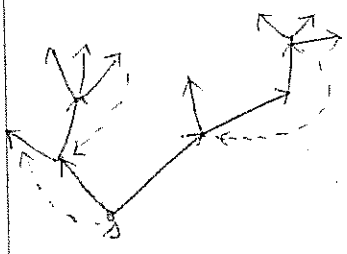
Wont be fussy about 

Bajan Ch 1: Defn of Paths, Reachability, etc
Linked lists circular etc
High Level Prog Notation

DIRECTED GRAPH

Search a directed graph $G = (V, E)$ from a start vertex s

① Determine set of vertices reachable from s.
② Determine a directed tree rooted at s and reaching all these vertices.
③ Traverse Find each edge $[v, w]$ s.t. v is reachable from s.
↳ If the tree was connected we might find all edges in tree right off the bat so we've never touched those edges.



Generic Search Method

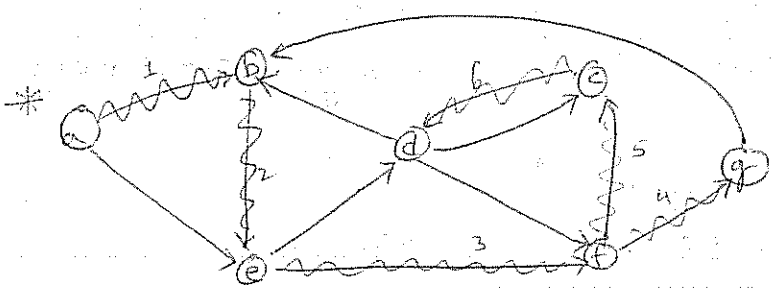
do as long as possible
choose an edge $[v, w]$ such that $[v, w]$ has not been chosen before and v is reachable from s.
if w is unvisited then

mark w as visited
place [v, w] in rooted tree
#

There are two ways of doing this:

DFS: stack of unexplored edges (considers most recently encountered edges first)

BFS: considers last recently encountered edges first
uses Queue data structure



; assume edge ; list representation

Need: Array to keep track of visited vertices

VISITED

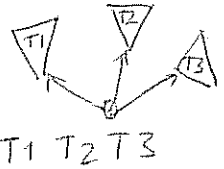
a	b	c	d	e	f	g
1	1	1	1	1	1	1

STACK

a b ae
ae
be ae
ae
cf ed ae
fg fc ed ae
fc ed ae
(been visited)
gb fc ed ae
ed ae
cd ed ae
ed ae
df db ed ae

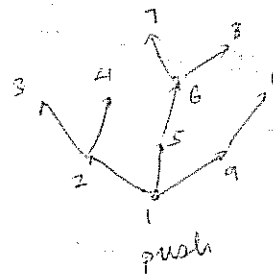
Properties of DFS tree

PREORDER POSTORDER numbering

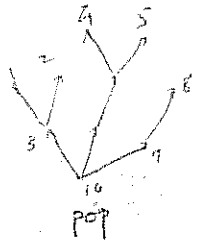


preorder: Root T1 T2 T3

postorder: T1 T2 T3 Root



push



pop

Running Time for BFS & DFS: $O(m+n)$ m = # of edges n = # of vertices

TOPOLOGICAL SORTING

$G = [V, E]$ digraph, Acyclic (No directed cycles)

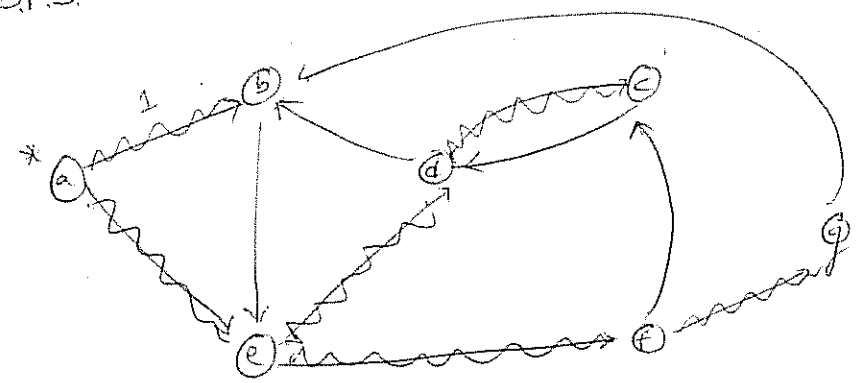
assign each vertex v a number $num(v)$ s.t. for every edge $[v,w]$ $num(v) < num(w)$

DFS on dir graph: every edge is either
(i) Forward (ancestor to descendant)
(ii) Backward edge (desc to anc)
(iii) Cross edge (from higher postorder to lower)

Charac of acyclic dir graph: no back-edges!
in DAG the postordering has prop opp. to what we need. So do negative

∴ DFS delivers topological ordering of acyclic digraph.

B.F.S.



a	b	c	d	e	f	g
1	1			1		

QUEUE

ae be .....

Sp. Property: Generates shortest Paths
Label each vtx v level $*$: ans is distance from root.

EECS244

28 Aug '90
Tuesday

TA : Paul Stephan
11:00 - 12:30
Rick McGeer
463 Cory mcgeer@ic.Berkeley.EDU ; 550 G Cory 1st 2 weeks
3:30-500 office hours
Reader : COPYMAT

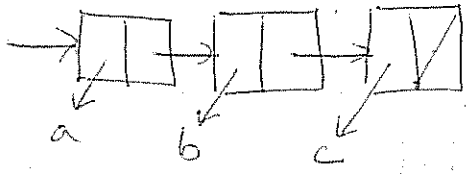
CS60C <Lecture> 28 Aug '90
Tuesday

Copyright SouthSide GCA

- Algorithms
- Large Programming Project ⇒(various levels)
  - dBX
  - Make

Learn LISP

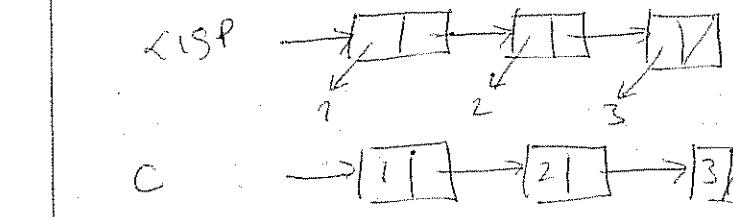
(a b c)



cons: "construct" take two things and return a ptr to each of the two things.

cons
car → first element
cdr → everything else

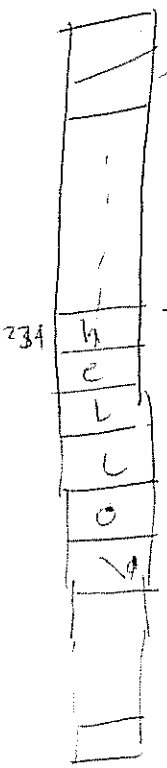
struct - type issue




```
struct pair {
        int pair_car;
        struct pair *pair_cdr;
    };
```

implicit size!

#define NIL ((struct pair*) 0)

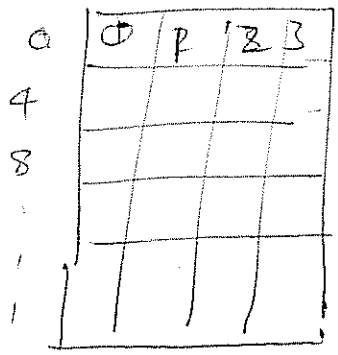


foo(~)
{ char word[6] = "hello";
char *wp = &word[0]; * not legal math
* but legal decl */
& "address of"

wp [1234]

pointer width – 32 bit 4 Bytes
(∵ Address Space)

mode of memory



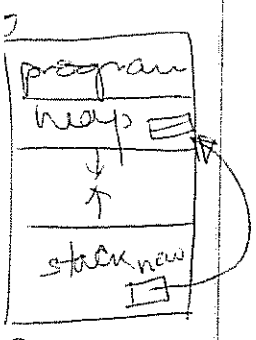
8bytes total [int : 4bytes (ie INT MAX = $2^{31}-1$ ~ 2Billion)
struct pair *pair_cdr; 4bytes

old C style

relationship between local (stack) global (heap)

struct pair *cons (num, list)
int num
struct pair * list;
{
struct pair *new = (struct pair *) malloc (sizeof (struct pair));
4bytes
3bytes
new → pair_car = num; need to specify so can do ptr arithmetic
new → pair_cdr = list;
return (new);
}

sizeof not procedure! just from arg its a type not available.



comment (1) Can't initialize a local array
(2) check malloc() returns valid
(3) use typedef for struct pair

easy way to write CAR :-

```
#define car(x)  ((x)->pair_car)
```

"number defines does char subs"

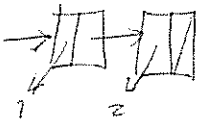
() have to do c operator precedence

```
struct thing {
        int thing_type;   <- 0,1,2 one of these!
        union {
                int thing_int;
                char * thing_symbol;
                struct { struct thing *thing_car,
                         *thing_cdr; } thing_pair
        } thing_stuff;
};
```

alternatively

```
struct pair {
        struct thing *pair_car, *pair_cdr;
};
struct thing {
        int thing_type;
        union {
                int thing_int;
                char *thing_symbol;
                struct pair thing_pair;
        } thing_stuff;
};
```

#define car(x) ((x)->thing_stuff.thing_pair.thispair)

(1 2) in Lisp 

#define ~~INT~~ ∅
#define SYM 1
#define PAIR 2

integer with manifest type ?)

#define get_int(x) ((x)->thing_stuff.thing_int)

CS270 Lecture

30 Aug '90
Thursday

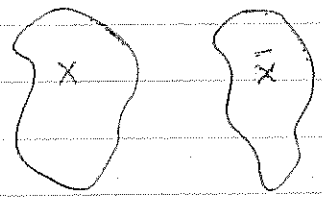
MIN SPANNING TREE (Jayan Ch 6)
DISJOINT SET DATA STRUCTURE (Jayan Ch 2)

CLARE BOLFING 569 Evans (Secretary)

Minimum Spanning Tree Problem –
Connected Graph (V, E)
Each edge e has cost $c(e)$
Assume for convenience all costs distinct

Tree: connected graph $\not\ni$ cycles
Goal: spanning tree of min^m total cost.

defn A CUT is a partition of the vertex set V into two (disjoint) sets $X, \bar{X}$



An edge crosses $(X, \bar{X})$ if it has one endpt in X and the other in $\bar{X}$.

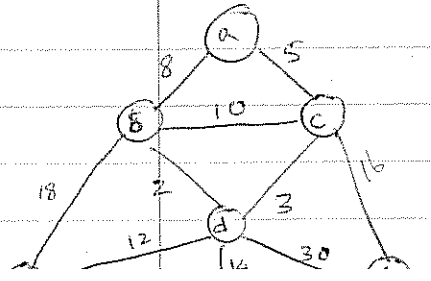
Thm (Recall ASSMP all costs are distinct) The MST is unique. For each edge 'e' the following are equivalent

(i) $e \in T$

$\exists (X, \bar{X})$ s.t. e is most costly crossing $(X, \bar{X})$ $\equiv$ (ii) e is the cheapest edge crossing some cut

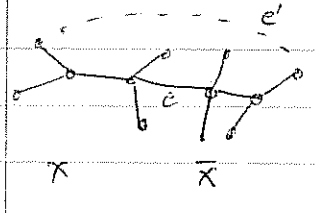
$\exists$ cycles C e is the most costly edge in C $\equiv$ (iii) e is not the most costly edge of any cycle

look at cycle $abc \Rightarrow 5, 8, 10$ $\therefore$ 10 $\notin$ Tree (iii)
look at partion $\{a,b,c\}, \{d,e,f,g\}$ 18, 2, 3, 16



Proof (i) ⇒ (ii) ⇒ (iii) ⇒ (i) circle of implications!

(i) ⇒ (ii) e is in some min^m S.T. ⇒ e is cheapest in some cut

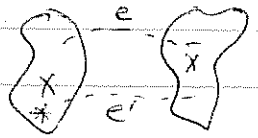


Remove e ; if there were an e' in cut then e couldn't be in MST

In KARPS words: "Consider cut $(X,\bar{X})$ determined when e deleted from T Suppose that for contrad. that this cut contains e' $(c(e') < c(e)$ then $T+e'-e$ is less costly than T

⇒⇐

ii) ⇒ iii) e is cheapest in some cut
⇒ e is not most costly in any cycle



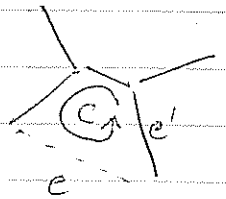
cycle must start someplace w.o.l.g. in X!

Any cycle C containing e also includes some second edge e' crossing cut $(X, \bar{X})$

$cost(e) < cost(e')$
∴ e not most costly

(iii) ⇒ (i) e not most costly in any cycle
⇒ e is in MST

Suppose for contrad that e not in T

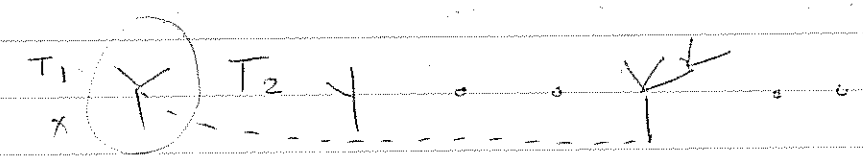


T+e contains exactly one cycle
Supposed that e not most costly.
then T-e+e' cheaper than T

⇒⇐

Uniqueness ∵ (i) & (iii) give nec & suff cond for e to be

FOREST OF BLUE TREES



EDGE e IS EMERGENT FROM TREE T_i IF IT HAS EXACTLY ONE END POINT IN T.

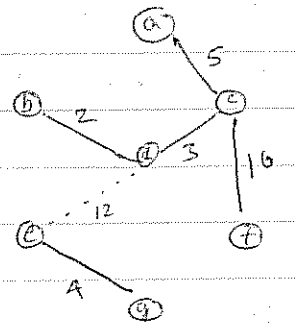
The cheapest edge emergent from T_i can be colored blue since its the cheapest in a cut.

This will reduce # of trees in forest by 1.

1. BORUVKAS ALGORITHM

INITIALIZE each vertex is a trivial blue tree

ITERATION STEP in par each blue tree T_i find cheapest edge emergent from T_i and color it blue.



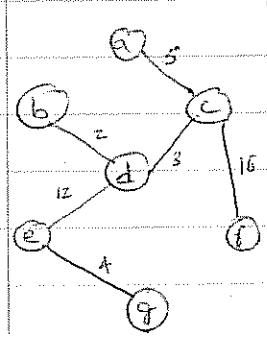
iteration 1 iteration 2

2. KRUSKALS ALGORITHM

SORT edges in increasing order of cost

color each edge in turn blue iff it doesnt form a cycle w previous blue edges

2,3,4,5,8,10,12,14,16,18,26,30

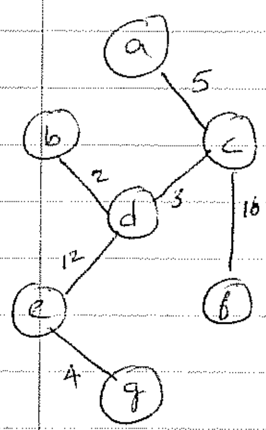


proof: edge not blue ⇒ not in MST (∵ max in cycle)

Rest as an exercise

PRIMS ALGORITHM

Grow single tree from arb start vertex at each step choosing the cheapest emergent edge.



Proof: Every edge selected is cheapest in some cut.

ROUND-ROBIN ALGORITHM

Its easier to look at fan out edges from ~~cheapest~~ smallest trees (As opposed to Prim)

So keep blue trees in a Queue

Initialization: Each vertex constitutes a trivial tree.

Repeat n-1 times

T_1 := tree at head of Queue

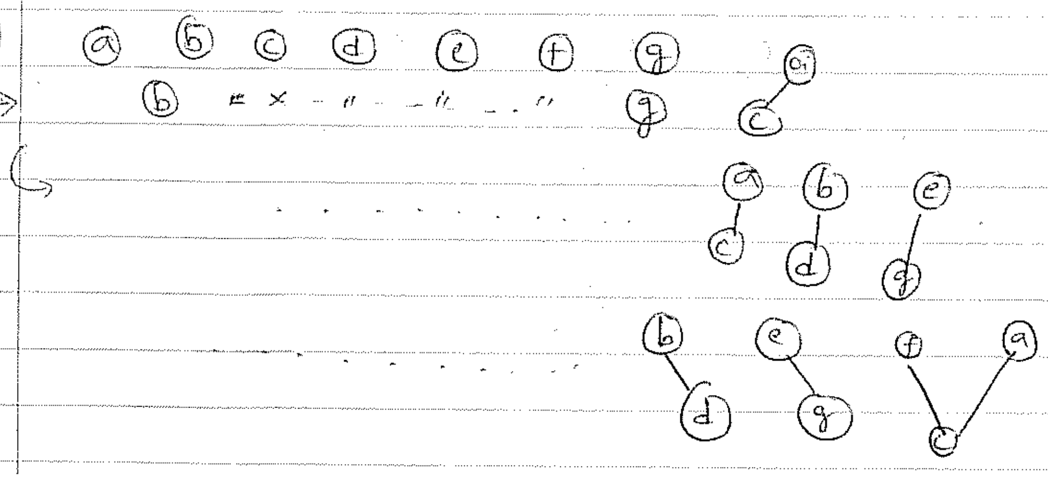
e := cheapest edge emergent from T_1

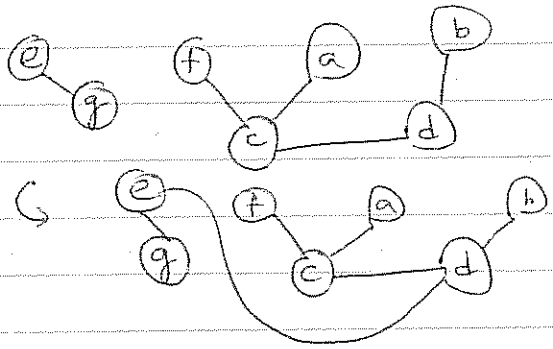
T_2 := other tree incident with T_1

Delete T_1 and T_2

Insert T_1+T_2+e at rear of Queue

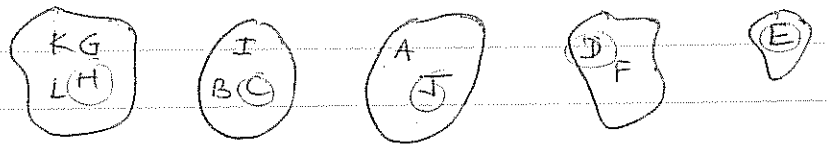
Q initial





ALGORITHM has to keep track of blue trees.
For each tree the algorithm must delete the cheapest emergent edge. (HEAPS!)

Keep track of partition of V into vertex sets of blue trees



; We need names
; for the sets!
; Represent names
; by Canonical Elem

Operations needed: link(x, y) x & y are Canonical Elem and $x \neq y$

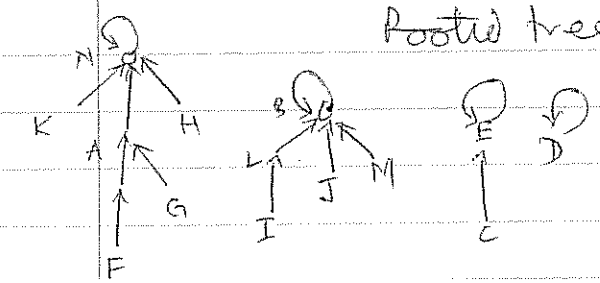
↳ combine sets rep. by x and y destroy original sets
Select a canonical elt for new set.

make set(x): to construct a new set consisting of x alone where x isn't in any previous set!

find(x): returns canonical element of x's set.

DISJOINT SET DATA STRUCTURE

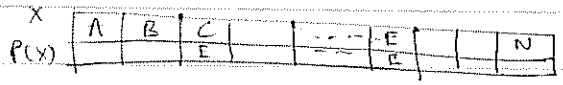
Rooted tree ↄ all edges pointing



find: go up pointers
link: choice

Is there an easier rep. than pointers.

- Array (when size of Universe predetermined)



etc. * we will use these datasets

COSTS

Makeset (x) : 1

Link (x, y) : 1

find (x): # of nodes on path to canonical elem. (root)

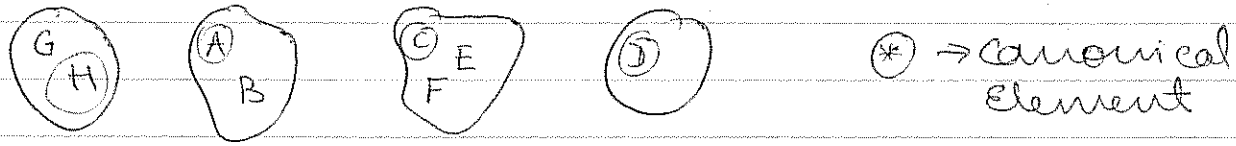
short bushy trees better

CS270 Lecture 4 Sept

No office hour today
Homework – not being graded fully
– Hand in best solution

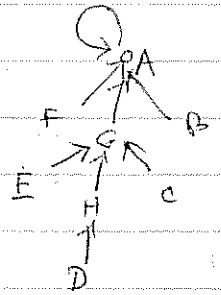
169 1600 9th Str. Cedar 525-8797

Data Structure for storing DISJOINT SETS



makeset(x)
find(x) ; returns CanonicalRep of x's set
link(x,y) ; x≠y x,y canonical elements
⇒ merge sets named x,y and naming can. elt.
→ could extend to any elements (by going through find)

Rep. each element by rooted tree
Node for each elt
Can Elt at root
Parent Pointers



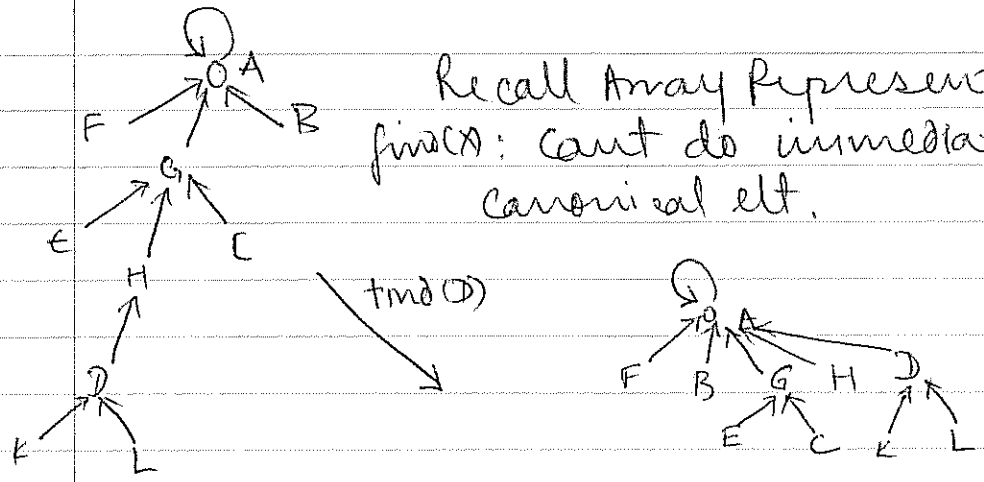
1 makeset(·) O(1) time
1 link(·,·) O(1) time
FIND
COST = # of pointers followed

Arose in context of MST
Now we study in own right!
want short busy trees. ∴ introduce

Refinements

(1) Path Compression
when doing find(x), cause all nodes along the path followed to point directly to Canonical Elt.

emiris@calliope
adnan@bryce



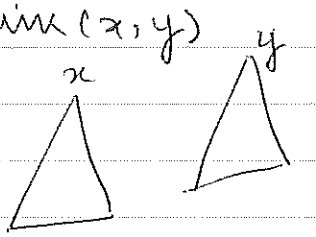
Recall Array Representation!
find(x): can't do immediately ∴ must find canonical elt.

Other rooted trees untouched.
A sort of defensive measure
"collapsing find"

UNION BY RANK: Associate to each element x a non neg integer rank(x)
[Dynamic variable]

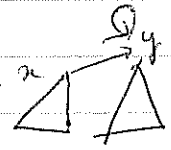
makeset(x) ; rank(x) := 0
find(x) ; leaves ranks unaffected

link(x,y)

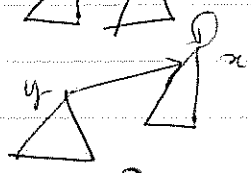


in general smaller should get to larger!

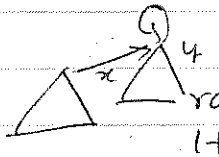
So (i) rank(x) < rank(y)



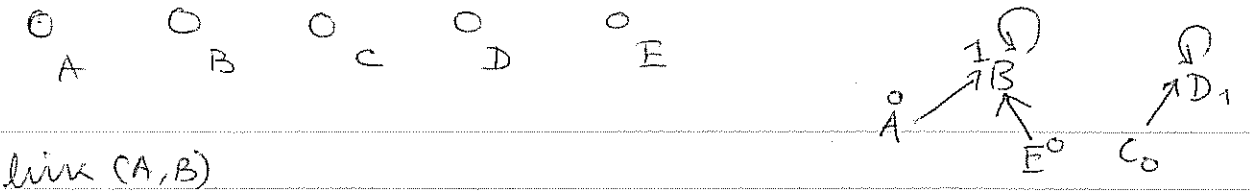
(ii) rank(x) > rank(y)



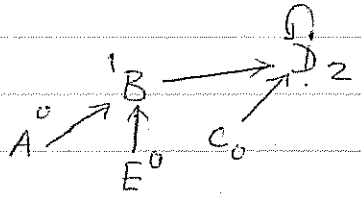
(iii) rank(x) = rank(y)



rank(y) := 1 + rank(y)



link (A, B)
link (B, E)
link (C, D)
link (B, D)



1 rank(x) initially 0
2 rank(x) can't decrease
3 rank (p(x)) ≥ rank(x) ; equality ⇔ x is canon. Elt
proof by Ind on # of steps
4 rank (p(x)) can't decrease ; part of x not fixed!
; (consider collapse)
5 rank(x) stops changing once x is no longer canonical
6 ∴ rank(x) has a final value

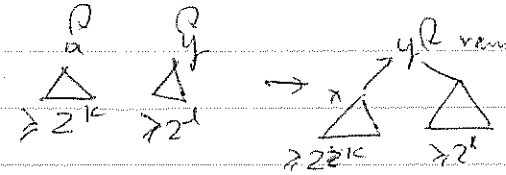
Call it $\overline{\text{rank}}(x)$ = Final value of rank(x)

canonical Elt !

A tree with root x has at least $2^{\text{rank}(x)}$ # of nodes

Pf: induction on steps.

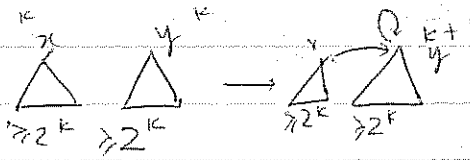
Case (i) rank(x) = k
rank(y) = l
Before step by ind hyp



Case (ii) symm.

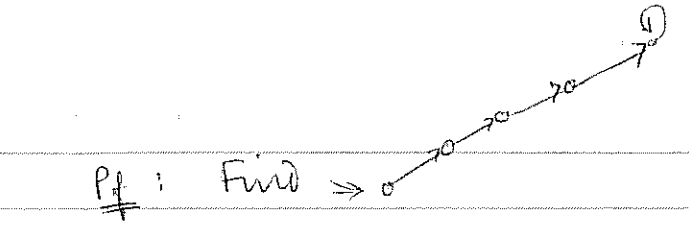
trivially true!

case (iii) rank(x) = rank(y) = k



Evident : $\geq 2^k + 2^k = 2^{k+1}$

NO. OF NODES 'x' WITH FINAL $\overline{\text{RANK}}(x) = k \leq n/2^k$
where n is the number of nodes (total)
(INDUCTION ON STEPS)
no element has $\overline{\text{rank}}$ greater than $\log_2 n$ [lg n]
∴ cost of find operation can't exceed lg n + 1



ranks ↑ strictly

$\leq \log n$

≥ 0

∴ only $(\lg n + 1)$ operations

Pf: Find ⇒

~~Another theorem~~

n elements in a universe

n makeset operations

$(n-1)$ links (at most)

m operations altogether $m \geq n$

cost of m operations $\leq m(\lg n + 1)$?

This is true in absence of path compression

with path compression + Union by rank

cost of m operations $= O(m\,\alpha(m,n))$

most slowly growing fn you've ever seen

Accounting Scheme

n elements

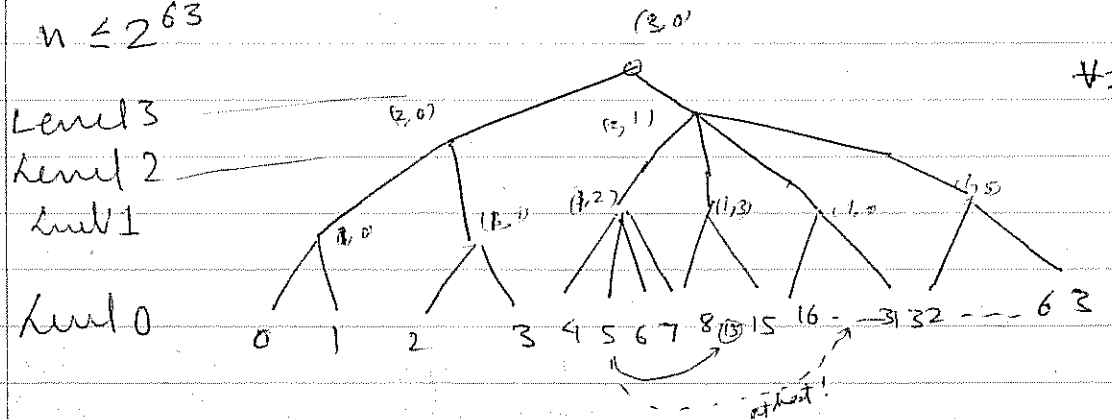
m operations

When we perform an operation, charge some of the cost of the operation and the rest to elts.

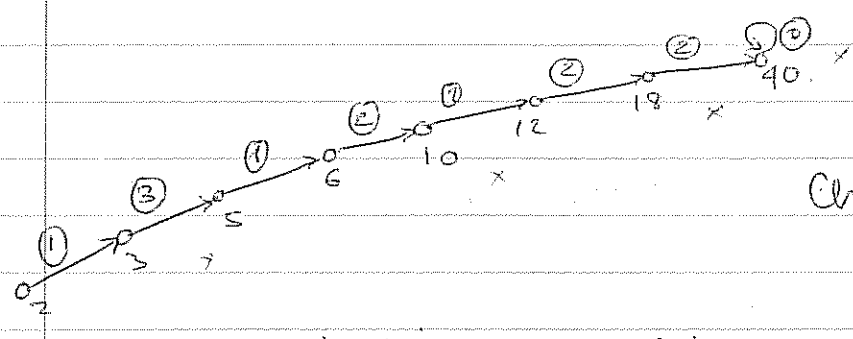
GOAL: $\sum_{\text{ops}}$ (cost charged to op) + $\sum_{\text{elts}}$ (cost charged to elts) $\leq O(m\alpha(m,n))$

$n \leq 2^{63}$

$\forall x$ rank $(x) \leq 63$

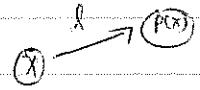


Least common ancestor - lowest node in common on search paths



Charge of 1 to elts of Rank 2,5,6,12

Charge to find operation = No. of distinct levels of edges



not the last edge of level l
Charge 1 to the element of rank x.

h = highest level number ($h=3$)

total charge to an element $x \leq$ if tree fixed

if $\overline{rank}(x) = 0, 1, or\ 2$ then $h\lfloor \frac{m}{n} \rfloor$

if $\overline{rank}(x) = k \geq 3$ then $h\lfloor \frac{m}{n} \rfloor k$

Total Charge to Elements:

$\overline{Rank}$		
0	n	$h\lfloor \frac{m}{n} \rfloor \leq hm$
1	$n/2$	$h\lfloor \frac{m}{n} \rfloor \leq h\, m/2$
2	$n/4$	$h\lfloor m/n \rfloor \leq h\, m/4$

# of elts st $\overline{rank}\ k \geq 3 \leq n/2^k$

so cost $\leq \frac{n}{2^k}(h\lfloor \frac{m}{n} \rfloor k) \leq \frac{k}{2^k} hm$

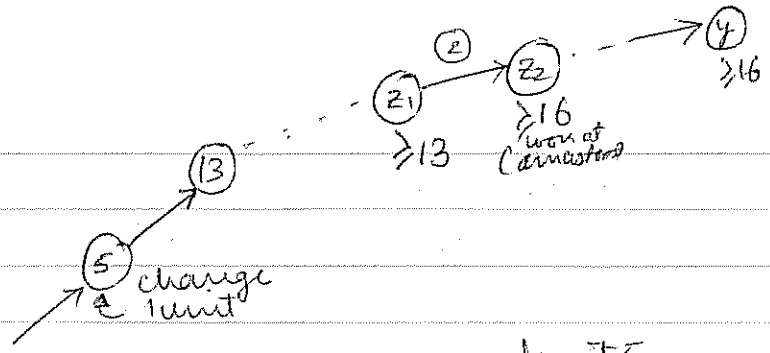
Sum of charges to all elts

$$\leq hm\left(1 + 1/2 + 1/4 + \sum_{k=3}^{\infty(\log n)} k/2^k\right)$$

$$= O(hm)$$

low. sum.

Q: How should the tree look? This is crucial if we are to attain these bounds



∴ node at 5 can be ~~change~~ at most changed at most 3 times (Perhaps 2 by ad hoc arg!)

Levels $0, 1, \ldots, h$

Nodes at level i be $(i,0)\ (i,1) \cdots (i,j) \cdots$

L_{bij} = # of children of node (i,j)

$b_{21} = 4$

$b_{15} = 32$

IF rank k has level i ancestor (i,j) then a node x with $\text{rank}(x) = k$ incurs a total change of $\leq b_{ij} - 1$ because of level i edges

CS270 (343) Sept 90 Thursday

DISJOINT SET DATA STRUCTURE

Ackermann's Function

$m\alpha(m,n)$ time bound

7b Line 2 Replace x By x_1

Line 7 Replace s by t

DISJOINT SET DATA STRUCTURE

using Path Compression

Union by Rank

Thm (TARJAN) Time to perform m ops on an n element universe ($m \geq n$) is $O(m\alpha(m,n))$

Ackermann Function $A(i,j)$ positive integer arguments

$A(1,j) = 2^j$

$A(i,1) = A(i-1,2)$

$A(i,j) = A(i-1, A(i,j-1))$ $j \geq 2$

$2^{2^{(2^2)}}$ $2^2 = 4$; $2^{2^2} = 2^4 = 16$; $2^{2^{2^2}} = 2^{16}$; $2^{2^{2^{2^2}}} = 2^{(2^{16})}$

$2^{2^{2^{2^{(2^{16})}}}}$

$A(i,j)$

i \ j	1	2	3	4	5
1	2^1	2^2	2^3	2^4	2^5
2	2^2	2^{2^2}	2^{2^3}	2^{2^4}	2^{2^5}
3	2^{2^2}	$2^{2^{2^2}}$			
4	$2^{2^{2^2}}$				

$A(2,2) = A(1, A(2,1)) = A(1,2^2) = 2^{2^2}$

$A(3,2) = A(2, A(3,1)) = A(2,16)$

invented in 20's

Logicians - Computability fns

primitive recursive fn.

↑ equiv. as nested loops

$$\alpha(m,n) = \min\{ i \geq 1 \mid A(i, \lfloor \tfrac{m}{n} \rfloor) > \lg n \}$$

expn elem

upper bound $\alpha(2^{63}, 2^{63})$ $\lfloor \frac{m}{n} \rfloor = 1$ $\lg n = 63$

Look at table $\Rightarrow 4$

$\alpha(\underset{m}{5 \cdot 2^{63}}, \underset{n}{2^{63}})$ $\lfloor \frac{m}{n} \rfloor$ $\lg n = 63$

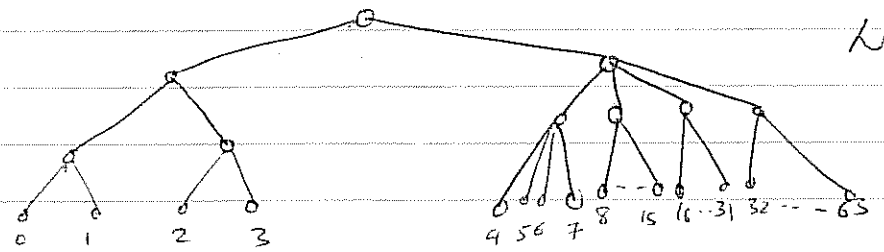
$\Rightarrow \alpha(5 \cdot 2^{63}, 2^{63}) = 2$

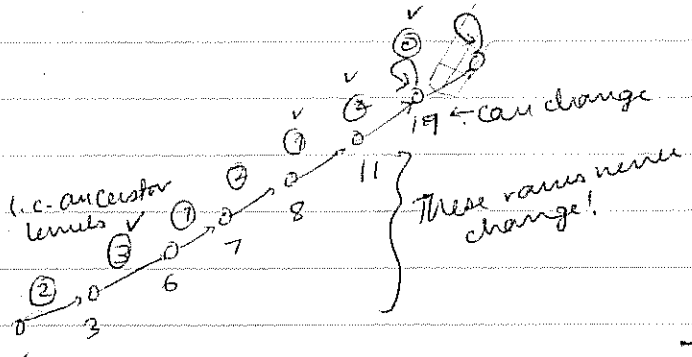
$\alpha(\cdot) \leq 4$ $\forall$ all situs!

Back to Union Find

Use ACCOUNTING trick

Charges each step in a FIND either to an operation or to an element.





FIND cost = 7 (# pts chosi

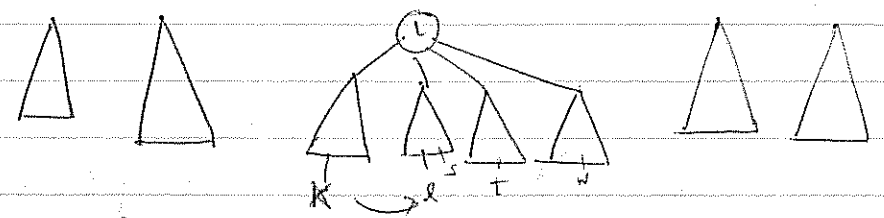
Cost allocated to operation: no. of levels occupied = 4

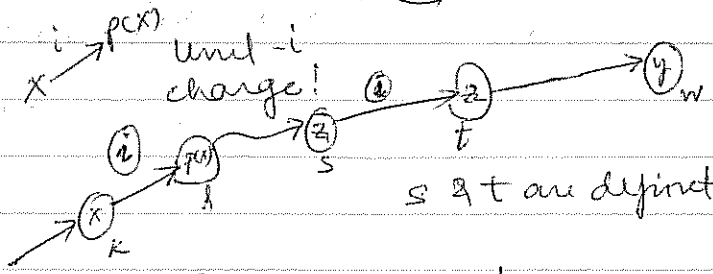
Charge of 1 to elts of rank 1, 6, and 7

Idea: construct tree so that sum of charges to elts is small

Charge to element of rk $x \leq$
no. of right-siblings of leaf x and its ancestors

eg elt 8 has 9 charged to it.





Fewer elements are charged high!

Levels $0, 1, \ldots h$

Charge to operations $\leq h+1$

$\sum$ charges to ops $\leq m(h+1) = O(mh)$

Sum of charges to any elt of rank 0, 1, 2, or 3 $\leq h \lfloor \frac{m}{n} \rfloor$

For $k > 3$ sum of charges to elt of rank k
$\leq h \lfloor \frac{m}{n} \rfloor \frac{k}{k}$

Sum of charges to elts

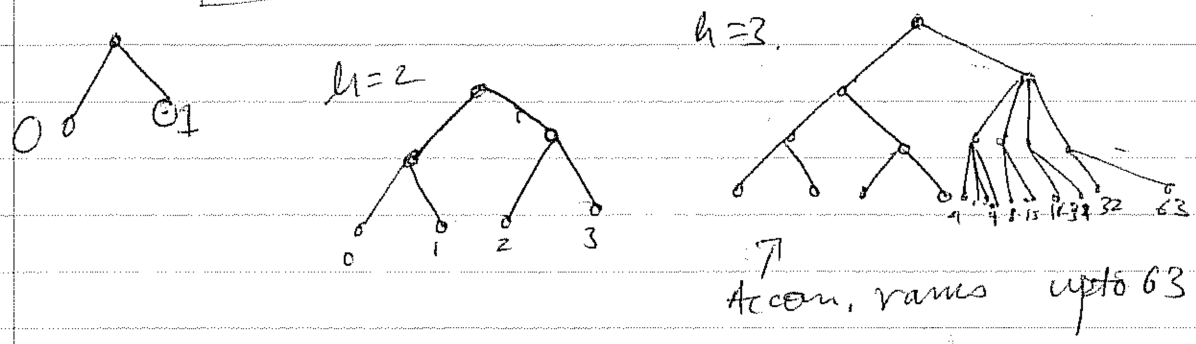
RANK	NO. OF ELTS	TOTAL CHRG TO ELT	TOTAL CHRG
0	n	$h \lfloor \frac{m}{n} \rfloor$	hm
1	$n/2$		$hm/2$
2	$n/4$		$hm/4$
3	$n/8$		
k	$n/2^k$	$h \lfloor m/n \rfloor k$	$hmk/2^k$

sum is $O(hm)$

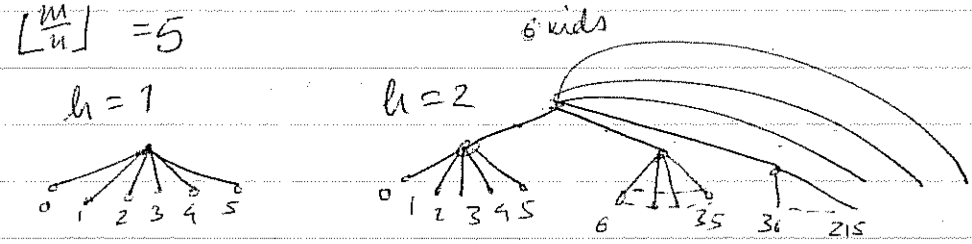
Sufficient to require that any ancestor of rank 0, 1, 2, 3 has $\leq 1 + \lfloor m/n \rfloor$ children and any ancestor of rank k has

Let design a tree subj to this (we want as few levels as possible!)

$h=1$ $\lfloor \frac{m}{n} \rfloor = 1$



$\lfloor \frac{m}{n} \rfloor = 5$



Write down constr. in rigorous way, obtain recursion rule ⇒ Ackermann fn!

Brain Teaser: need to go back after reaching root in find. Q: Find something which still obey $O(A(\cdot,\cdot))$ complexity. A single "pass" alternative

⟨MST⟩

$G=(V,E)$ $V=\{1,2,\ldots,n\}$ $|E|=m$

KRUSKALS ALGORITHM

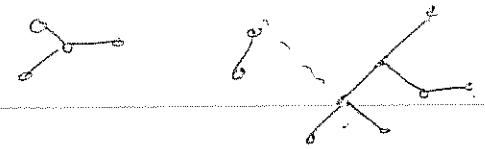
SORT THE EDGES IN ORDER OF INCREASING COST

$m \log m$ $e_1, e_2, \ldots e_m$

$E_T := \emptyset$

$m\alpha(m,n)$ for $i=1,2,\ldots m$ do // $e_i=\{u,v\}$ //

if u and v are in different components of present forest then $E_T := E_T \cup \{e\}$



$\{u,v\}$ $x := find\ u$

$y := find\ v$

if $x \neq y$ then $E_T := E_T \cup \{e\}$; $link(x,y)$

∴ could have done in $O(n \log)$ even w/o collapsing find.

PRIMS Algorithm

$X := \{s\}$ X nodes in component constructed so far

$E_T := \emptyset$ E_T edges in tree so far

terminating operation while $|E_T| < n-1$ do

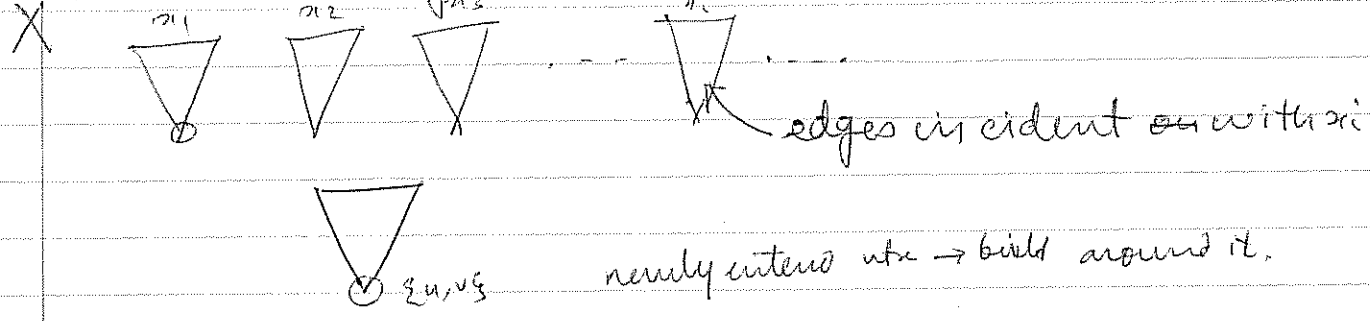
$e :=$ MIN COST EDGE CROSSING CUT $(X, \bar{X})$

$\|e = \{u,v\}\|$

$e_T := e_T \cup \{e\}$ (* don't need fancy data structure *)

$X := X \cup \{u,v\}$

How do we organise this?



CS 270 — Combinatorial Algorithms — 11 Sept 1990 Tuesday

3108 Etchremy from Thursday

PRIMS MST algo Sec 6.2

Heaps (Ch 3)

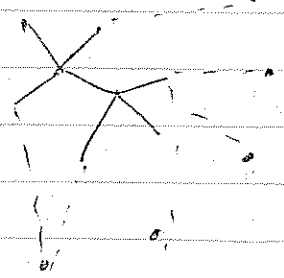
d-Heaps SPP Algo Ch 7

cont.

CS270
Combinatorial Algorithms

(12-1) Sep '91
Tuesday

PRIMS Algorithm for MST
- Grow the tree from arbitrary start vertex s
- At each iteration, adjoin to the tree the min^m cost edge emergent from the present tree.



v borders T IF $v \notin T$ and $\exists$ edge from T to v

Data Structures: A bit vector indicating which vertices border T. (border)

For each vertex v that borders T

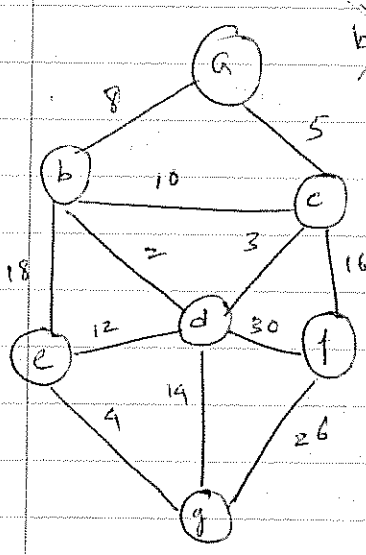
key(v): the cost of the cheapest edge from T to v

blue(v): The cheapest edge from T to v

weak tie! arbitrarily!

Select vertex v for which key(v) is min^m

adjoin edge blue(v) to T.



border	v	key(v)	blue(v)	in T
	a			1
~~1~~	b	~~82~~ 5	ab db	
1	c	5	ac	1
	d	3	cd	
	e	12	de	
	f	16	cf	
	g	14	dg	

. . . . etc

- What data structure needed?

HEAP

HEAP collection of items each w a key

- insert (i, h) : insert new item i into heap h
- deletemin (h) : return and delete min^m wt item
  (in case of tie : any)
  (in case of empty : NULL)
- makeheap (s) : create a heap w/ set of items s.
  (in time ∝ to # of items)

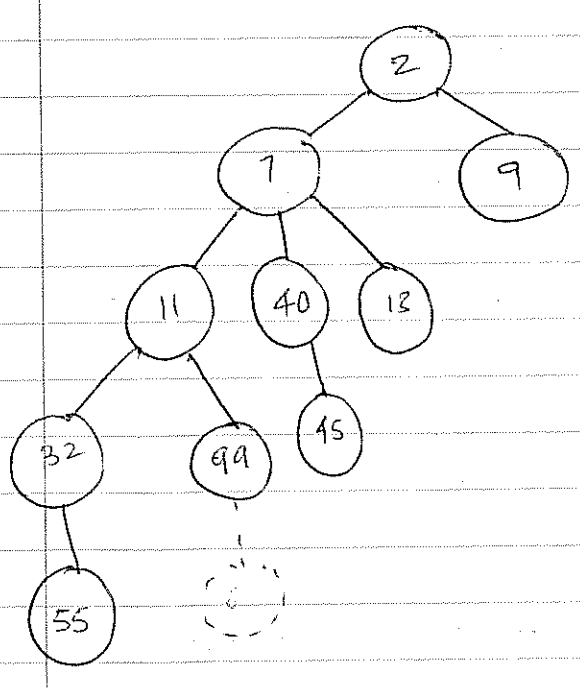
findmin(h) : find but don't delete item of min^m key.
(Returns NULL if empty)

delete (i, h) : delete i from h given pointer to i

meld (h_1, h_2) : take two disjoint heaps & meld into a single heap

(sift up) decreasekey (D, i, h) decrease key of item i in heap h by positive amount D.

HEAP-ORDERED TREE



prop : key (p(x)) ≤ key (x)

insertion:
adjoin new leaf node (vacancy)
then sift up. (vacancy moves up to root)

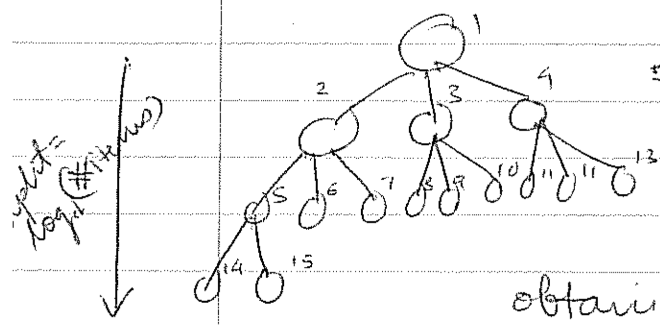
deletion (i, h). make vacancy. pick an element & slip something into it.
take a leaf, move up to vacancy
Then sift down the vacancy
(look @ min^m key of children & swap)
↳ Requires either sift up or sift down

delete : always sift down

findmin: trivial (maintain ptr to root)

d-ARY HEAP (d=3 example)

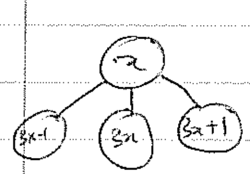
a complete tree in which every internal node has 3 children



Ternary ē 15 items

could be an implicit Data Struc (ie could calc child's address using arithmetic rather than prts)

obtain addresses by #ing level by level, left to right.



∴ parent(x) = $\lceil \frac{x-1}{3} \rceil$

General d: address of parent = $\lceil \frac{x-1}{d} \rceil$

addresses of kids (dx+2-d ... dx+1)

Now in delete, must use (15) when delete: to preserve complete

COST of sift up = O(log$_d$ n)

COST of sift down = O(d log$_d$ n)
↑ each child

which ops need siftup & siftdown

siftUp: insert, decrease key

SiftDown: delete, deletemin

How do you choose d?

Look at tradeoff.

n VERTICES

m EDGES

COSTS • deletemin: (n-1) times d log$_d$ n

expensive ↑*cheap*

organization triplet $(v, key(v), blue(v))$

COST • insert $(n-1)$ times } $\log_d n$

COST • decrease key $(m-n+1)$

Total Work $= m \log_d n + nd \log_d n$] worst case analysis. Think about this in context of avg. case.

min^m when ?

$m = nd$ i.e. $d = m/n$

or to be precise $d = \lceil 2 + m/n \rceil$ $\neq \max(2, m/n)$

WORK $O(m \log_{m/n} n)$

Dense Graph $m = n^{1+\epsilon}$; $\epsilon > 0$

WORK $= m \log_{n^\epsilon} n = m/\epsilon$

For dense graphs this algo (for appr. choice of d) is optimal

SHORTEST PATH PROBLEM (Heaps Reqd)

Meld operations will be impt. (Ch7)

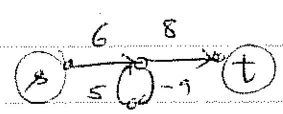
shortest paths →

digraph G

for each edge $[u,v]$ length (u,v)

length path $= \Sigma$ length of edges

shortest path from s to t.

(i) SP may not exist 

There is a path from s to t and

Thm $\exists$ SP from s to t $\Leftrightarrow$ no path from s to t includes a negative cycle. If there is a S.P. from s to t then there is a a S P which is simple.

Suppose some s-t path contains a negative cycle

$\nexists$ a S.P. (Evident)

Suppose that no s-t path contains a negative cycle
Clearly there is a shortest simple path from s to t.
Let its length be L.
Could there be a non simple path which is cheaper?
NO. (∵ repetition, cut it out)
∴ Each non simple path also costs at least L.

Single Source S.P.P.

Find shortest paths from s to all other vertices (or else find a neg cycle)

Key obs. ∃ a S.P. tree i.e. the set of SP from S to all other vertices is a tree.

CS270 — Combinatorial Algorithms — 13 Sept '90 Thursday

S.P. Ch 7 Tarjan Reordered
MT 1 10/11 Open book in class
Given a directed graph G w/ vtx set V, edge set E.
$l(u,v)$ = length of (u,v)

Thm: Let s and t be st. A $s \to t$ exists
$\exists$ a shortest $s \to t$ path $\Leftrightarrow$ $\nexists$ $s \to t$ path containing a negative cycle.
If $\exists$ a shortest $s \to t$ path then $\exists$ a shortest $s \to t$ path that is simple.

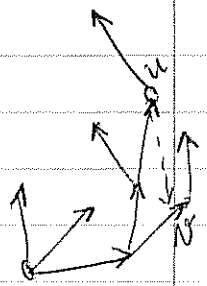
Find S.P.'s from s to all vertices reachable from s, or find a neg. cycle reachable from s.

S.P. from s can be represented by a directed rooted tree.

Given tree how do you verify that it is S.P.
Given tree T

$D_T(u) =$ if $u \in T$ then $s \to u$ distance in tree
else ∞

Thm T is a SP tree $\Leftrightarrow$ for every edge (u,v)
(*) $\text{Distance}(u) + l(u,v) \geq \text{Distance}(v)$



pf Suppose $D_T(u) + l(u,v) < D_T(v)$

Then $\exists$ a shorter path to v

Suppose that $\forall (u,v)$
$D_T(u) + l(u,v) \geq D_T(v)$
wish to conclude T is a S.P. tree

pf: $x_0 \to x_1 \to x_2 \cdots \to x_t$
x_0 st. vtx $x_0 = s$

length of this path $\geq D_T(x_t)$
pf by induction on t.

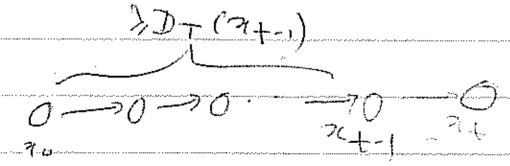
CS270 cont

13 Sept 90
Thursday

Trivially true for base case $t=0$.

Induction Step:

Assume result for $t-1$



$\therefore$ length of path $\geq D_T(x_{t-1}) + l(x_{t-1}, x_t) \geq D_T(x_t)$ (*)

QED

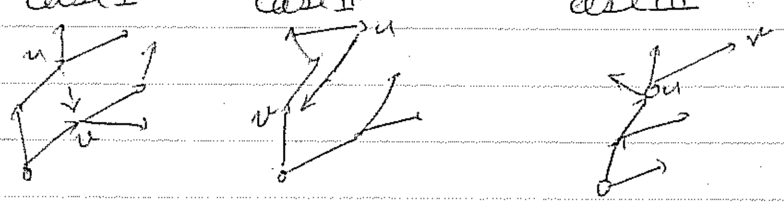
Keep track of parent: suff to describe tree!
ie $p(v)$ for v.

LABELING

At each step have directed rooted tree at s, associated with T, $D_T(u)$

Find edge (u,v) violating (*)

case I — case II — case III



set $p(v)=u$ & change distances updated for v descendants
pref algo have v s.t. v has no kids

neg cycle

set $p(v)=u$

v desc of u → no problem

Scan Based Algorithms

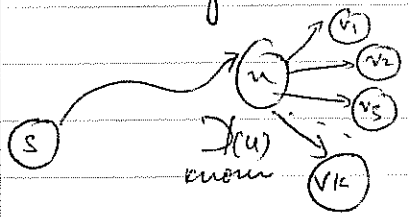
Status of a vertex

Unlabeled $D_T(v)=\infty$

vertices in tree { Labeled
{ Scanned

What does it mean to scan a vtx?

scanning a labeled vtx u



process (u,v_1)
(u,v_2)
(u,v_k)

(v_i)

if $D(v_i)$ gets changed, mark v_i as labeled

mark u as scanned

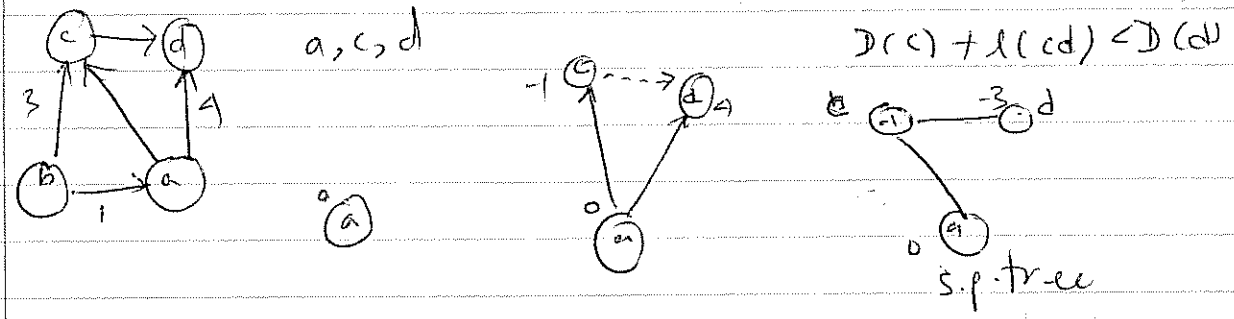
Digraph is acyclic
(No directed cycles)

Arrange the vtxs reachable from s in a sequence s.t. every edge (u,v) runs from earlier vtx to a later one.

Scan vertices in this order

once u is scanned it wont become labeled again.

Ex Algorithms S.P. tree



$O(m)$ algorithm : (each edge only once (2 times?)

Can also comp. longest paths in DAG (replace by neg)

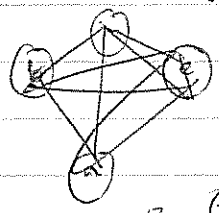
Termination condition: no labelled vertices remain

Neg cycle: ∞ loop (unless you have smart for stopping)

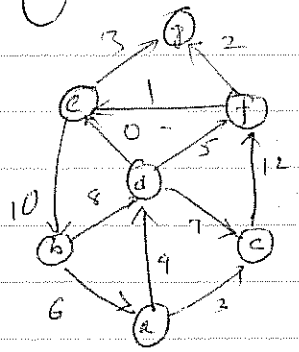
All lengths non negative ≥ 0

Dijkstra's Algorithm

Scan a labeled vertex whose distance is least



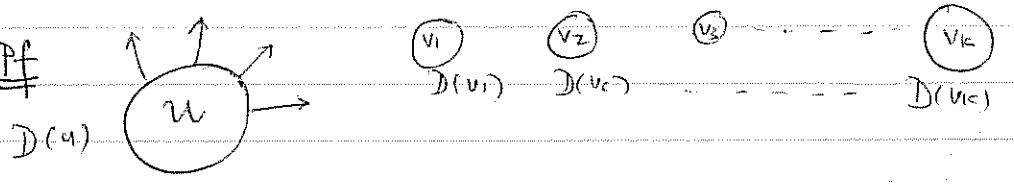
Example (in book)



never scan vtx twice (but only because non negative edges)

Key concept: Once a vtx get scanned it never gets labeled again.

Pf



Know $D(u) \leq D(v_i)$ $\forall i$

∴ every vtx that gets labeled at or after the scanning of u receives a label $\geq D(u)$

(Formally: ind on steps.)

Thingo completely on non neg. lengths.

Similar to Prims

Data Structure: Heap!

Would still work but exp. bad possibility

Implementation

d-HEAP

for every vtx which is labeled, will maintain

(v, D(v) p(v)) ← key on D(v)

Time: $(m \log_d n + nd \log_d n)$

$d = \lceil 2 + m/n \rceil$

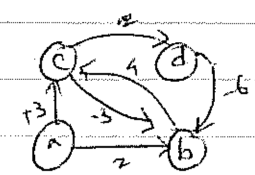
Time $m \log_{\lceil 2 + \frac{m}{n} \rceil} n$ (m = # edges, n = # vertices)

Neg edges :?

Keep labeled vertices on a Queue

Scan the vtx at the head of the queue

When a scanned or unlabeled vertex becomes labeled, place at rear of queue.



u	D(u)	p(u)
a	0	–
b	~~2~~ ~~0~~ -1	
c	3	
d	5	

Q

a b c b d b' [illegible]

A vertex can get scanned repeatedly but not more than (n-1) times

Claim: In absence of negative cycles, a vtx gets scanned at most (n-1) times.

CS270 18 Sept '90

SINGLE SOURCE SHORTEST PATHS
ALL PAIRS SHORTEST PATHS
DYNAMIC PROGRAMMING

Midterm Oct 11 Guest Lecturers Sanjoi Irani 9/25 9/27
Abhijit Sahay 10/2

Digraph $G = (V, E)$
$l(u,v)$ length of Edge $[u,v]$
(=weight)

Single Source –
(a) Negative Cycle
(b) Tree of shortest paths from "s" to all reachable vertices

SCAN algorithms: Each vertex is unlabeled, labeled or scanned
$D(u)$ tentative distance from s to u.

$D(u)$ tentative distance from s to u.
$P(u)$ predecessor of u on path to u.

While there is a labeled vertex,
choose a labeled vertex u and scan it.

SCAN "u" → for each edge $[u,v]$ do
if $D(u) + l(u,v) < D(v)$
then $D(v) := D(u) + l(u,v)$
$P(v) := u$
mark v as labeled
mark "u as "scanned"

Think about how we initialize and start.

Terminate when there are no labeled vertices

In absence of negative cycles, termination occurs

(1) ACYCLIC DIGRAPH

Scan vertices in topological order

Single pass (Each vtx is scanned only once)

Time $O(m)$ ← # of edges

(2) When All wts nonnegative:

(DIJKSTRA)

Scan labeled vertex of least tent. distance

Each vertex gets scanned once

(Data structure overhead) dheap: Time Bound $O\left[m \log_{\lceil 2+\frac{m}{n} \rceil} n\right] = O(n^2)$

(3) GENERAL CASE

Keep labeled vertices on Queue

Scan vertex at head of Queue

Put newly labeled vertices at rear of Queue.

Start with s on Queue

$D(s) = 0 \quad D(v) = \infty \quad v \neq s$

Pass 0: scan s, initially on queue

Pass j: Scan vertices on queue at end of pass j.

Claim At beginning of pass j the following holds for all vertices u.

$D(u) \leq$ min^m length of an $s \to u$ path

with at most j edges

Proof (induction on j)

(Suppose no neg cycles viz
⇒ should be done by pass $n-1$
(∵ all simple paths have
at most $(n-1)$ edges)
By beginning of pass $n-1$
$D(u)$ = length of min $s \to u$ path
∴ negative cycle ⇔ ud^n in
some $D(u)$ during pass $n-1$.

Time Bound

$O(m)$ per pass

Time Bound for whole algorithm $O(mn)$ best so far (in presence of neg. wts)

ALL PAIRS S.P.P.

Find either a negative cycle or between all pairs of vertices, the S.P. i $\forall u, \forall v$ shortest $u \to v$ path.

Approach 1: Single Source run n times.

All costs ≥ 0

$n\, m \log_{\lceil 2 + \frac{m}{n} \rceil} n = O(n^3)$ "Dijkstras"

General Case $n(mn) = n^2 m$

Idea (General Case) Transformation that yields non negative edge costs

Then execute n Dijkstras

Given $G = (V, E)$ $l(u,v)$

Introduce "s" an additional new Dummy vertex
Want to make sure everything is reachable from "s".
add directed edge $[s,u]$ of length 0 from s to each vertex.

call the new graph G^*.

In G^* we solve the single source problem from s.

$\Rightarrow O(nm)$

2 possibilities

(a) negative cycle in $G^* \Rightarrow$ neg. cycle in G

OR

(b) obtain a shortest path tree

$D(u)$ = $s \to u$ distance

Define $l'(u,v) = D(u) + l(u,v) - D(v)$

Must have

(I) $l'(u,v) \geq 0$

(II) preserve shortest paths.

(I) because else could get path to v which is cheaper than $D(v)$

(II) 

general path

ORIGINAL COST $l(x_0,x_1) + l(x_1,x_2) + \cdots l(x_{t-1},x_t)$

TRANSFORMED COST $l'(x_0,x_1) + l'(x_1,x_2) + \cdots l'(x_{t-1},x_t)$

$= [D(x_0) + l(x_0,x_1) - D(x_1)] + [D(x_1) + l(x_1,x_2) - D(x_2)] + \cdots$

$\cdots + [D(x_{t-1}) + l(x_{t-1},x_t) - D(x_t)]$

$= D(x_0) + l(x_0,x_1) + \cdots l(x_{t-1},x_t) - D(x_t)$

$\longleftarrow$ original length $\longrightarrow$

$D(x_0) - D(x_t)$ is constant for any fixed pair of pts.

$\Rightarrow$ cheapest paths preserved Q.E.D

Using this approach, the ~~cost~~ time bound of the all pairs problem is $O(mn)$ + $O(mn \log_{\lceil 2+m/n \rceil} n) = O(n^3)$
(preprocessing step)

HW#3
Problem 1: Graph is undirected | on sheet

DYNAMIC PROGRAMMING
no defn, pt of view

"To solve a problem, imbed it in a collection of related problems, and solve them all together." (informally)

Eg1. SP in Acyclic Digraph G
Vertex Set $\{1, 2, \ldots n\}$ $l(i,j)$ length of edge $[i,j]$
(ass. already top. ordered so $i<j$)
shortest path from 1 to n.
Method: compute S.P. from all vertices to n.

$D(i)$ = shortest distance from i to n.

$D(n) = 0$ (i)

$D(i) = l(i,j) + D(j)$ if via j

$\therefore D(i) = \min_j [l(i,j) + D(j)]$ (ii)

What order?
decreasing order!

Eg2 Longest Common Subsequence (allow gaps)
sequence 1 A B A C B A B
sequence 2 B A B A B C C

Strings $a_1 a_2 \ldots a_m$, $b_1 b_2 \ldots b_n$ || LCS

$L(i,j)$ = length of the longest common subsequence of $a_i a_{i+1} \ldots a_m$ and $b_j b_{j+1} \ldots b_n$

we want $L(1,1)$

$L(i,j)$ = if $a_i \neq b_j$ then $\max [L(i,j+1), L(i+1,j)]$ ← max of adj

else $1 + \max(i+1, j+1)$

$a_i \ldots a_m$
$b_j \ldots b_n$

$L(i,n+1) = 0$ B.C.

$L(m+1,j) = 0$

"Solving by working backwards"

Practise calculation -

	1	2	3	4	5	6	7	
	B	A	B	A	B	C	C	
1 A		x		x				0
2 B	x		x		x			0
3 A		x		x				0
4 C						x	x	0
5 B	x		x		x			0
6 A	3	3	2	2	1	0	0	0
7 B	3	2	2	1	1	0	0	0
	0	0	0	0	0	0	0	0

1	2	3	4	5	6	7
A	B	A	C	B	A	B

order of filling in: ← ← ← ← ← ← ←

CS270 (19th) Sept 90
Thursday

DYNAMIC PROGRAMMING

TSP
MIS in a tree
Δ evaluation
Binomial Queues

To solve a problem, imbed it in a family of interrelated problems & solve them all together

Understand by examples
1 S.f. in an acyclic digraph
2 Longest common subsequence

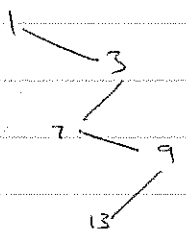
3. TRAVELLING SALESMAN PROBLEM

TSP in plane
We will deal in more general form: non Euclidean, non plane

d_{ij} : distance from city i to city j
start at 1 visit each city exactly once and then return to city 1.
Obj: minimize total distance travelled.

Approach 1: try all tours $(n-1)!$ too high!

Generic Problem:
Present city i
$S \subseteq \{2, 3, \ldots n\} - \{i\}$
set of cities not yet visited.



$C(i, S)$ Min^m cost of visiting all cities in S and then returning to city 1, starting at i $(i \notin S)$

$C(i, \phi) = d_{i1}$

Goal - $C(1, \{2,3,\ldots n\})$

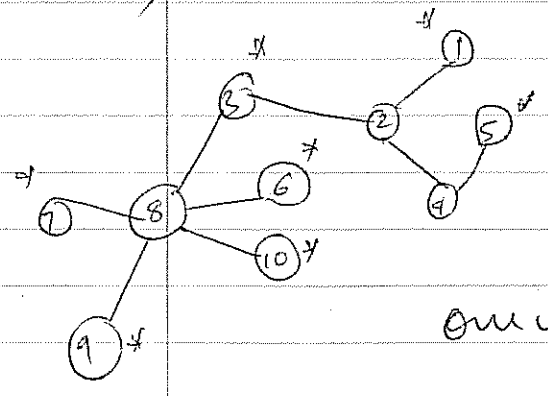
$$C(i, S) = \min_{j \in S} \left[d_{ij} + C(j, S - \{j\}) \right]$$

Go through sets in increasing order of size.

# of subproblems	$\sim n \cdot 2^{n-1}$	← real killer: its the size of the table (storage requirement)
# of arithmetic steps	$\sim n^2 2^{n-2}$	

Practical for $n \leq 15, 16$

4. Maximum wt independent set in a tree



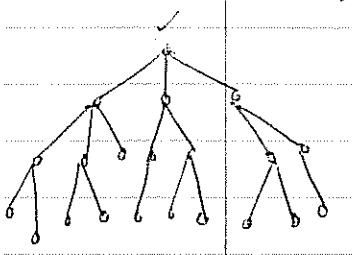
A set of vertices is indep if no two are adjacent.

One way - "Greedy" start from highest

General DP on trees -

Root tree, use recursion on subtrees

Solution arrived at by sequence of decisions.



(1) use root ⇒ skip one level
(2) don't use root ⇒ max wt ind set of children

For each node x,

$i(x)$ = max wt of an independent set including x in the subtree rooted at x,

$e(x)$ = max wt of an independent set excluding x in the subtree rooted at x,

$w(x)$ = wt of node x

$\Gamma(x)$ = set of children of x

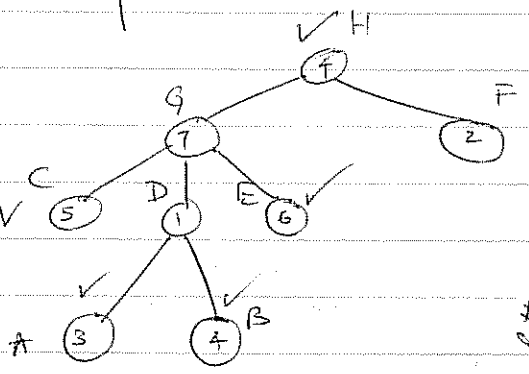
$$i(x) = w(x) + \sum_{y \in \Gamma(x)} e(y)$$

$$e(x) = \sum_{y \in \Gamma(x)} \max(e(y), i(y))$$

How shall we do the tabulation-?

Bottom Up, starting at leaves.

Example



x	e(x)	i(x)
A	0	3
B	0	4
C	0	5
D	7	1
E	0	6
F	0	2
G	18	14
H	20	22

Bottom Up ordering

$O(4)$ Constant of work per edge

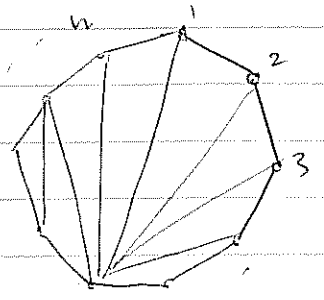
How do you compute actual soln?

incl H

incl C, E

incl A, B

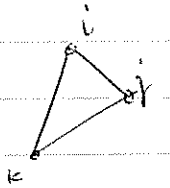
5. Triangulation of a Convex Polygon



Subdivision into Δs using non crossing lines.

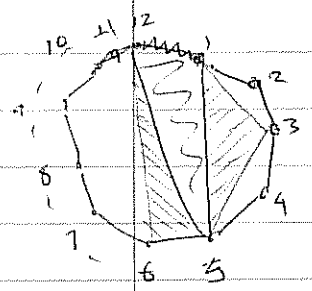
- min^m length
- as near to equilateral

|| possible applic



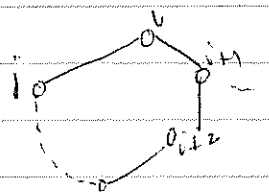
$\Delta(i,j,k)$ is cost of $\Delta(ijk)$

Min sum of costs (Exp # of possibilities!)



$\Delta(12,1,5)$

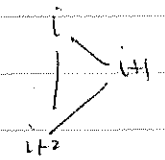
$t(i,j)$ = min^m wt of triangulation of polygon $(i, i+1, \ldots, j, i)$



$t(i,i+2)$

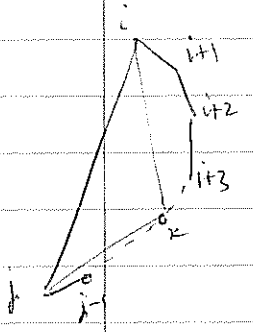
only one way!

cost = $\Delta(i, i+1, i+2)$



$t(i,i+1) = 0$ natural choice!

$$t(i,j) = \min_{i+1 \le k \le j-1} \left[\Delta(i,k,j) + t(i,k) + t(k,j) \right]$$



we want $t(1,n)$

Tabulate in increasing order of $(j-i)$

# of steps = # of distinct (i,k,j) triples $= {}^nC_3$

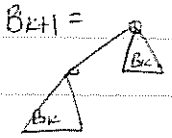
$\therefore$ cost is $\Theta(n^3)$

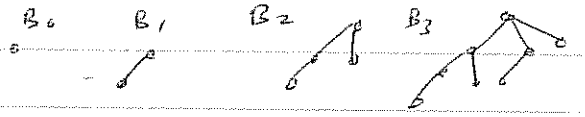
MELDABLE HEAPS items, keys

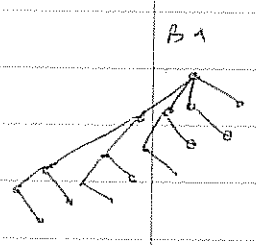
Makeheap(S) put set of itts into a heap
insert(i, h)
deletemin(h)
findmin(h)
meld(h1, h2) combine two disjoint heaps (may have identical keys but not itts)

Binomial Queue (Vuillemin)
Brown

Based on concept of Binomial Tree

B_0 B_{k+1} = 

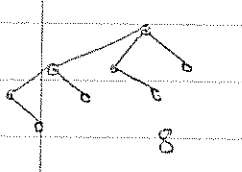




induction

- B_k has 2^k elements
- # of nodes at distance l from root = ${}^kC_l$

Binomial Forest = Sequence of Binomial Trees of strictly decreasing sizes



only possible ~~tree~~ forest w/ 11 nodes

$2^3 + 2^2 + 2^1 = (1011)_2$ ← ones of Binary rep. of 11

Binomial Queue: Heap ordered binomial forest
item in each node

$$key(x) \geq key(p(x))$$

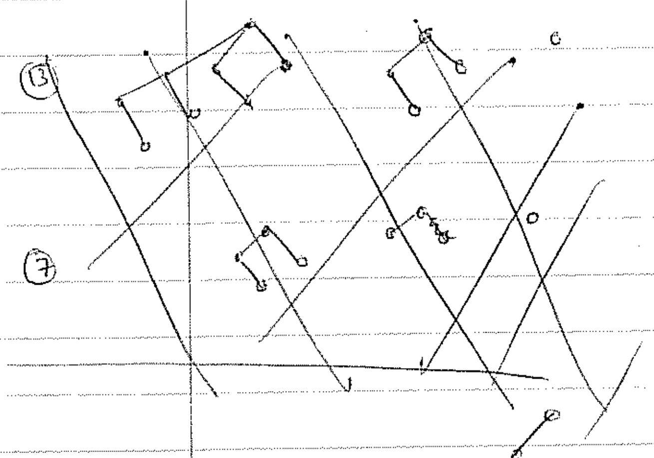
let n = # of items in Queue

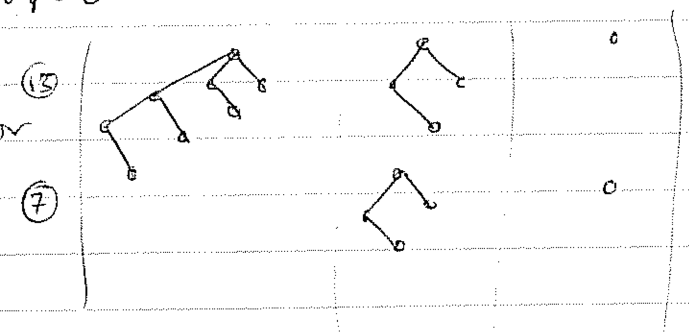
findmin (h) : march through roots - ⌈log n⌉ (as many bits as there are in binary rep of log n
$O(\log n)$

meld (h_1, h_2) $|h_1| = 13$ $|h_2| = 7$

(13) 1101
+ (7) 111
10100

} similar to meld

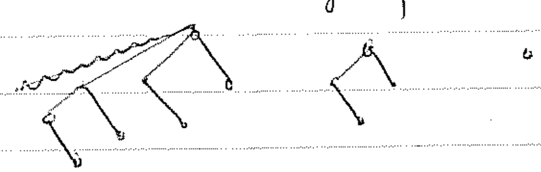




root of the two trees being combined is lesser of the two roots
$\underline{O(\log n)}$

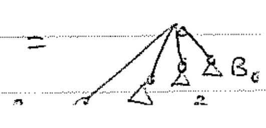
insert (i, h) ← like adding 1 (11)
so like meld special case
$\underline{O(\log n)}$

deletemin ↳ going to be a root of one of the trees.



start c̄ B_k delete root ⇒ have k different trees
∴ combine Binomial forests.
Meld → $O(\log n)$

B_k =



CS270 (Section)

21 Sept '90
Friday

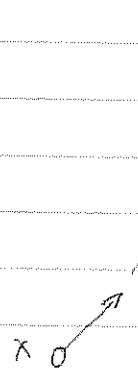
Q2. Revision Union Find Algorithm
Q4. Using Rank Find to maintain a depth determination abstraction
Q8. Finding the k^{th} smallest elt in a binary heap
Q1. One $\alpha(N,N) = O(\log^* N)$

binomial queue (implementation of a heap)
binomial forest (bin trees $\leq$ 1 of each size)

lazy heap: can assume it's implicit.

meld of two heaps:

Q2. Impl. find(x) s.t. we traverse path x to root once



two pass (as we saw)

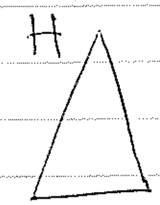
↳ one pass?

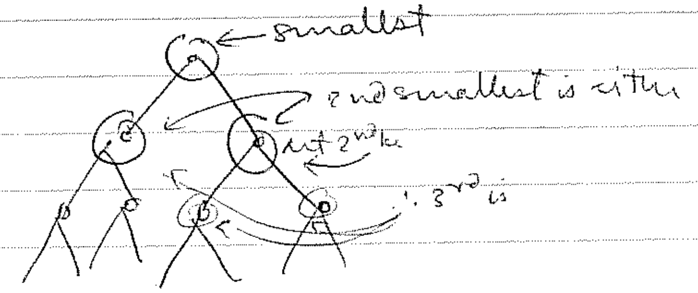
```
find(x)
for i = 0 to h
    store[i] = null
while p(x) ≠ x do
    {level = level(x, p(x))
    if store[level] ≠ null p(store[level]) = p(x)
        store[level] = x
        x = p(x)}
∀ i, p(store(i)) = x
return x
```

Claim: if $(x, p(x))$ is not the last i^{th} level ptr level $(x, p(x)) = i$ then $p^+(x)$ = parent of rightmost i^{th} level ptr

⇒ Proof that cost allocated to a node of RANKS $\leq$ # of right siblings of R & its ancestors

Q8./ Finding the k^{th} smallest elt in a heap.
(Dave Cohen)





Use auxiliary heap A

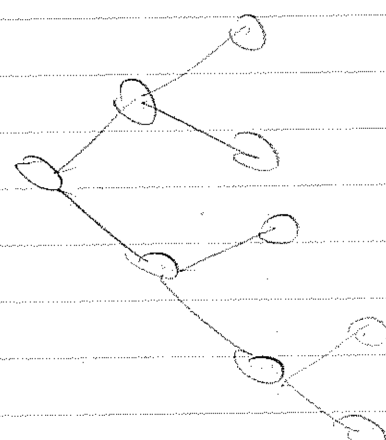
A = make heap (findmin H)
for i = 1 to k
$O(\log_2 |A|)$ {
ism [i] = deletemin (A)
insert (child1[ism [i]], A)
" (child2 (" , . . .)
}

k^{th} smallest is ism [k]

$\sum_{i=1}^{k} \log_2(i) \approx k \log_2(k)$ time

lower bound — $\Omega(k)$ lower bound
children @ k^{th} level (?)

k (k) —
k+1 k-1+2 = k+1
k^2



Algorithms and Data Structures

Daniel Sleator Editor

Relaxed Heaps: An Alternative to Fibonacci Heaps with Applications to Parallel Computation

JAMES R. DRISCOLL, HAROLD N. GABOW, RUTH SHRAIRMAN, and ROBERT E. TARJAN

ABSTRACT: The relaxed heap is a priority queue data structure that achieves the same amortized time bounds as the Fibonacci heap—a sequence of m decrease_key and n delete_min operations takes time O(m + n log n). A variant of relaxed heaps achieves similar bounds in the worst case—O(1) time for decrease_key and O(log n) for delete_min. Relaxed heaps give a processor-efficient parallel implementation of Dijkstra's shortest path algorithm, and hence other algorithms in network optimization. A relaxed heap is a type of binomial queue that allows heap order to be violated.

1. INTRODUCTION

The Fibonacci heap data structure of Fredman and Tarjan allows an optimal implementation of Dijkstra's shortest path algorithm [3]. It is central to the best-known algorithm for minimum spanning trees [5] and many other algorithms. These applications are based on the fact that, with a Fibonacci heap, a sequence of m *decrease_key* and n *delete_min* operations takes time $O(m + n \log n)$. Equivalently, Fibonacci heaps achieve an amortized time of $O(1)$ for *decrease_key* and $O(\log n)$ for *delete_min*.

The research effort of Gabow was supported in part by NSF Grant No. MCS-8302648 and AT&T Bell Laboratories. The research effort of Tarjan was supported in part by NSF Grant No. DCR-8605962 and ONR Contract No. N00014-87-K-0467.

This article presents a new data structure called the relaxed heap. Two implementations of relaxed heaps are given. The first achieves the same amortized bounds as Fibonacci heaps, but maintains more structure. This structure may make relaxed heaps faster in practice. The second implementation of relaxed heaps gives a theoretical improvement over Fibonacci heaps: it achieves the above time bounds for *decrease_key* and *delete_min* in the worst case, rather than in the amortized case. The article concludes with some applications. The main result is a processor-efficient parallel algorithm for shortest paths on a directed graph with nonnegative edge lengths. Specifically, for an EREW PRAM parallel machine with p processors and a graph with n vertices and $m \geq n \log n$ edges, an optimal running time of $O(m/p)$ is achieved for $p \leq m/(n \log n)$. This result can be achieved using either implementation of relaxed heaps. We do not know whether the same result can be achieved using Fibonacci heaps. The parallel shortest path algorithm can be used as a subroutine to get processor-efficient algorithms for other problems in network optimization, such as the minimum spanning tree problem, assignment problem, transportation problem, and others.

Relaxed heaps are based on a more structured family of trees than Fibonacci heaps, namely the binomial trees. The height of a binomial tree of n nodes is a factor $\log_2 \phi = 0.69+$ times the height of a Fibonacci tree. This improves the constant in asymptotic estimates. Whether these savings are actually realized in

practice is a matter for experimental verification, which we have not done. (Brown [2] shows that the closely related binomial queues are efficient in practice.) Relaxed heaps give a large family of alternatives to Fibonacci heaps. These alternatives provide both theoretical insight and possibly the flexibility needed for efficient practical implementation.

Concurrent with this work, Peterson [12] has proposed an elegant data structure that achieves the same amortized bounds as Fibonacci heaps. It has a number of interesting properties, such as working with binary trees. Comparing Peterson's work with the concerns of this article, [12] does not achieve the bounds in the worst case, and does not seem to support the applications to parallel computation given here.

In this article log n denotes logarithm to the base two. A *priority queue* is a data structure for storing a set of *items* x, each having a numerical *key* denoted $k(x)$. The main operations are:

make_heap—initialize a heap to store the empty set;
insert(x)—make x a new item in the heap;
decrease_key(x, v)—decrease key $k(x)$ to a smaller value v;
delete_min(x)—delete an item of minimum key from the heap and return it as x.

Two useful additional operations are:

find_min(x)—return an item of minimum key as x;
delete(x)—delete item x from the heap.

Other operations will be introduced as needed.

Binomial queues were introduced by Vuillemin [14]. The *binomial trees* B_r are defined recursively as follows: B_0 is one node; B_{r+1} consists of two B_r trees, the root of one being a child of the root of the other. See Figure 1(a). In all figures, a triangle labeled r represents the binomial tree B_r. Figure 1(b) shows an equivalent description of B_{r+1}: For any k, $0 \le k \le r$, B_{r+1} consists of a B_k tree with additional children of the root that are themselves roots of $B_k, B_{k+1}, \ldots, B_r$ trees. For any node x in a binomial tree, *rank*(x) is the index r of the maximal subtree B_r rooted at x. A binomial tree is an ordered tree, with the children of a node ordered by increasing rank. The *last child* of a node is the child of highest rank. Note that B_r has 2^r nodes and height r.

In a *tree data structure*, each node c stores one item (including its key, denoted $k(c)$). Node c is *good* if it is the root, or if its parent p satisfies $k(p) \le k(c)$; otherwise ($k(p) > k(c)$) c is *bad*. In a *heap-ordered tree*, all nodes are good. A *binomial queue* for 2^r items is a heap-ordered tree B_r. A binomial queue for n items, n arbitrary, consists of at most $\lfloor \log n \rfloor + 1$ heap-ordered binomial trees, a tree corresponding to each one bit in the binary expansion of n. The *link* operation for binomial queues takes two root nodes of equal rank r and creates a heap-ordered tree B_{r+1} by making the node with larger key a child of the smaller.

It seems difficult to process *decrease_key* in $O(1)$ time and maintain heap order. The philosophy of relaxed heaps is to avoid this difficulty by permitting violations of heap order (whence the name). The rank relaxed heaps of Section 2 allow just one bad child per rank. The run relaxed heaps of Section 3 are less stringent and allow runs of bad children.

2. RANK RELAXED HEAPS

A *relaxed tree* is a tree data structure where some nodes are distinguished as *active* and any bad node is active. The terms *relaxed binomial tree* and *relaxed binomial queue* are interpreted according to this definition. A *rank relaxed heap* is a relaxed binomial queue that satisfies two conditions:

(a) For any r there is at most one active node of rank r.
(b) Any active node is a last child.

Condition (a) implies there are at most $\lfloor \log n \rfloor$ active nodes. Condition (b) is not crucial. It determines various programming details; we shall return to this point. In the rest of this section, "relaxed heap" means "rank relaxed heap."

The crux of the data structure is processing m *decrease_key* operations in time $O(m)$. The next paragraph

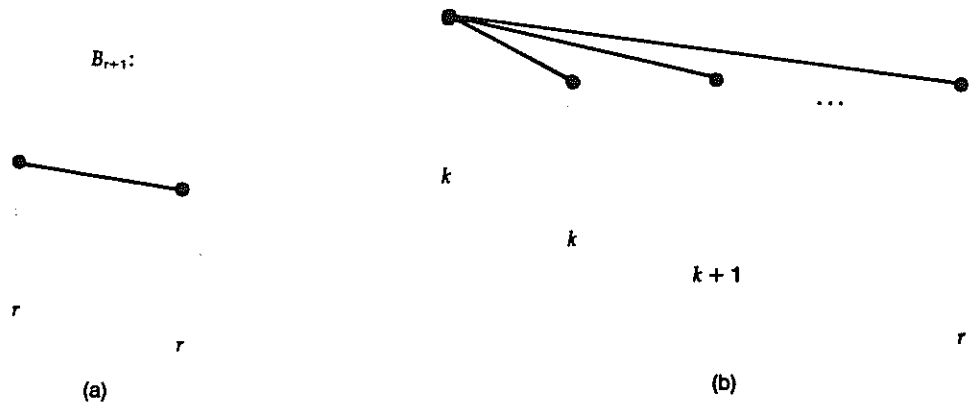


FIGURE 1. Recursive Definition of Binomial Tree B_{r+1}

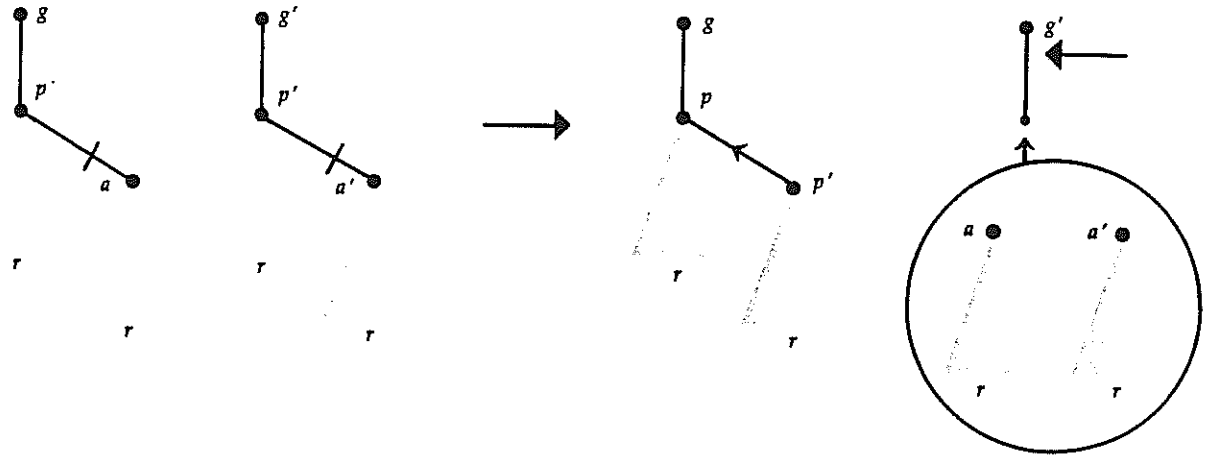


FIGURE 2. Pair Transformation

gives a plan for accomplishing this. We then fill in the details of the plan and give the algorithm for *delete_min*.

The *decrease_key* algorithm rearranges nodes to keep the heap relaxed. It does this with three transformations. More precisely *decrease_key*(x, v) resets $k(x)$ to v, after which it may stop or execute a transformation; a transformation does $O(1)$ work, after which it may stop or execute another transformation. This gives rise to a sequence of transformations. To achieve the time bound, let α denote the number of active nodes at any point in the algorithm. Initially there are no active nodes, so α is zero. If resetting $k(x)$ in *decrease_key* makes x bad, x is designated active. This increases α by one. Each transformation either

(i) decreases α, or
(ii) does not change α and does not execute another transformation.

Observe that (i) and (ii) imply that any sequence of m *decrease_keys* uses time $O(m)$: There are at most m type (ii) transformations since each is last in its sequence. There are also at most m type (i) transformations, since α is nonnegative and only *decrease_key* increases α. This argument will not be affected by *delete_min* operations (see the proof of Theorem 1).

To fill in the details of the plan we describe the transformations, at first informally. They are illustrated in Figures 2–5. In these figures an edge joining a child c to its parent p is labeled in one of four ways: An arrow from c to p indicates that c is good; a cross mark on the edge indicates c is active (and so can be good or bad); no mark indicates the status of c is unknown; an arrow pointing to the edge indicates c may be a new active node. The transformations use an operation "make node x the rank r child of node y." This means that the entire B_r tree rooted at x becomes the B_r subtree of y. For each transformation, assume *decrease_key* (or a transformation) has created an active node a of rank r with parent p and grandparent g (a is actually a bad node, although this fact is not used).

The main idea is embodied in the *pair transformation* shown in Figure 2. It applies when a is the last child of p, and further, the relaxed heap already contains an active node a' of rank r, with parent p' and grandparent g'. Note that by condition (b), node a' is the last child of node p'. The transform removes the active nodes from their parents, so nodes a, a', p, p' all have rank r. Without loss of generality assume $k(p) \leq k(p')$. The transform makes p' the rank r child of p (hence p remains a rank $r+1$ node). Then it links a and a' to form a B_{r+1} tree with root c (so c is a or a'). It makes c the rank $r+1$ child of g'. If c is now a bad child it is active, and a transformation is done for it. Note that c need not be the last child of g'.

The last detail concerns linking a and a'. In general, as in Figure 3, suppose nodes q and q' of rank $s+1$ have just been linked, making q the new root. If x, the rank s child of q, is active, it now violates condition (b) of the relaxed heap structure. Figure 3 compensates for this by performing a *cleaning operation*. It uses the fact that x', the rank s child of q', is good if x is active. This follows from the definition of a relaxed heap. (Care should be taken here, since the transformations are applied to heaps where the relaxed heap structure has been violated; however, we will use the clean operation only when this deduction is valid.) The operation repairs the damage by interchanging x and x'. In the following, to *clean* node q means to apply the operation of Figure 3 if x is active; otherwise do nothing. To *combine* two nodes means to link them and then clean the new root. Thus, in Figure 2, the *pair* transformation *combines* a and a'.

The *pair* transformation achieves property (i), since two initially active nodes are replaced by at most one. Also note that the reason this transformation is used only when a and a' are last children is to achieve the $O(1)$ time bound: If a' has rank r but is not the last child, its siblings must not be moved along with p'. Assuming a reasonable data structure (e.g., a parent pointer for each node) this can consume more than constant time. Note that this objection does not apply if

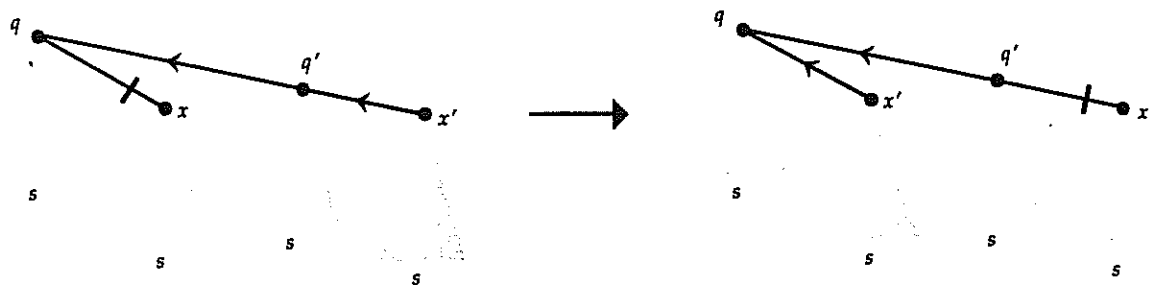


FIGURE 3. Cleaning

a' is a last child but a is not. This is the case in the good sibling transformation described below, which does a pair transform of this kind.

The remaining transformations are "sibling transformations." For these assume that a has rank $r + 1$ sibling s. The *active sibling transformation* applies when s is active, as shown in Figure 4. The definition of a relaxed heap implies that s is a last child, so $rank(p) = r + 2$. The transform removes the two active children from their parent p. It *combines* p and a, making a a rank $r + 1$ node. Then it *combines* a and s into a tree whose root c becomes the rank $r + 2$ child of g. If c is now bad, a transformation is done for it. The active sibling transformation achieves property (i), since two active nodes are replaced by at most one.

The *good sibling transformation* applies when s is good. Let c be the last child of s; hence $rank(c) = r$. There are two cases. If c is active the algorithm does a pair transform for a and c (as depicted in Figure 5). As noted previously, the transformation works correctly even though a is not a last child; the only delicate point is to make sure that if $k(p) = k(s)$ the algorithm makes s the child of p, not vice versa. This case achieves property (i), as above. The second case occurs if c is a good child. In this situation the cleaning operation of Figure 3 is applicable. The transform *cleans* p, making a a bad last child of s. It then processes a as a new active last child: If there is an active node of rank r a *pair* transform achieves property (i); otherwise the sequence of transformations stops, achieving (ii).

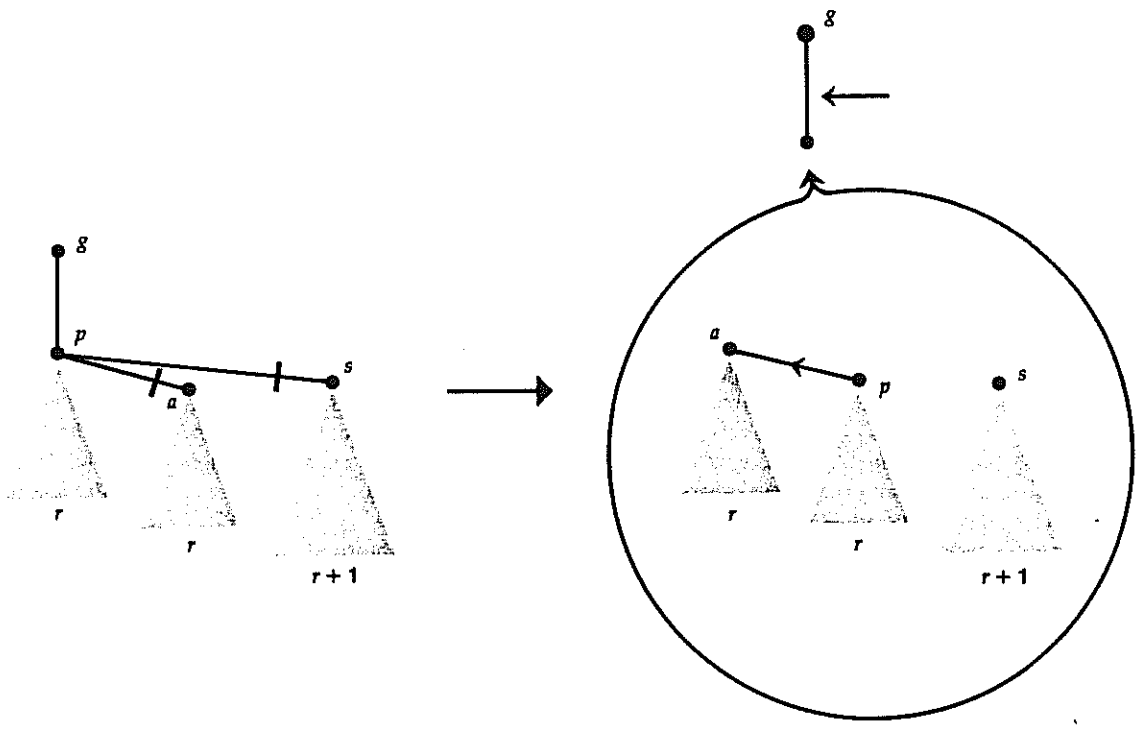


FIGURE 4. Active Sibling Transformation

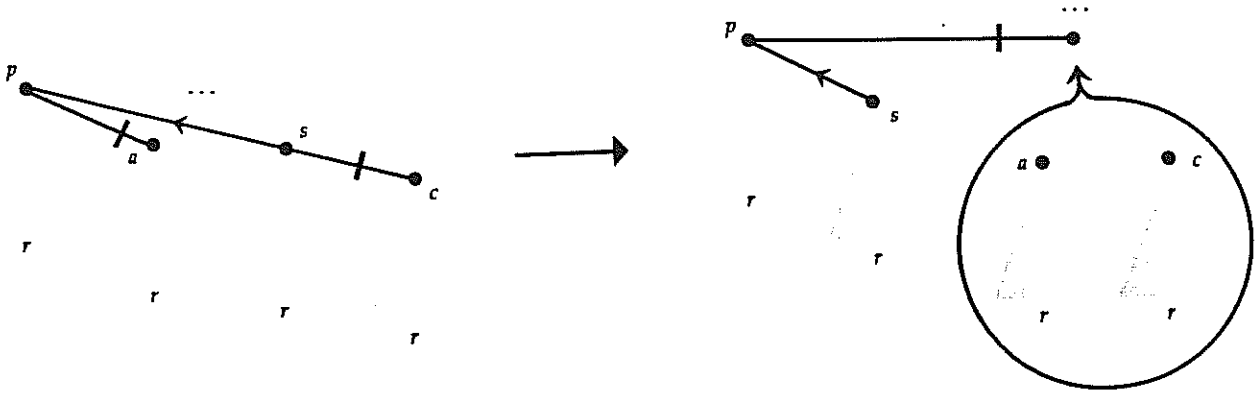


FIGURE 5. Good Sibling Transformation: *c* Active

Now we describe the *delete_min* operation. Note that in most applications (e.g., all those in [3, 5]) it is unnecessary to reclaim the storage used by a deleted node. We give two implementations of *delete_min*, the simpler of which does not reclaim storage. Both implementations start by finding the smallest node x. Since x is either active or the root of a tree in the queue, it can be found in $O(\log n)$ time.

The nonreclaiming algorithm sets $k(x)$ to ∞ and changes $rank(x)$ from r to 0. Then it merges x and its former children into a new rank r node, by repeatedly *combining* the two nodes of smallest previous rank. The new rank r node replaces x in the tree. Note that x becomes a leaf and will not participate in any future transformation.

The reclaiming algorithm is similar. It deletes x and removes the root node of smallest rank y from its tree; this makes the previous children of y into roots with the smallest ranks in the queue. Then it processes y (not x) and the former children of x as in the nonreclaiming algorithm.

Figure 6 gives a more detailed description of the algorithm in pseudo-Algol. The *delete_min* implementation uses the reclaiming algorithm. The following data structure is assumed. Each node x has a record containing $k(x)$, $rank(x)$, and pointers to its last child, its two neighboring siblings, and its parent. (A sibling pointer is needed in the sibling transformations. Hence pointers to both siblings are needed to allow nodes to be moved. A last child pointer is needed in the good sibling transformation.) There is a dummy node ρ treated as the root of the entire queue: each root of a binomial tree has parent ρ, so the roots can be treated as siblings and are not special cases. Further, $k(\rho) = -\infty$ and ρ is its own parent. In addition, there is an array $A[0 \ldots \lfloor \log n \rfloor - 1]$; each $A(r)$ is a pointer to the active child of rank r, if it exists. The A array is used to check if a node is active, e.g., *promote* tests if s is active by checking the condition $A(r + 1) = s$. It should be clear that this data structure supports the desired operations and can be maintained in time $O(1)$ per transformation.

It should be noted that the data structure can be initialized in $O(n)$ time (assuming, as is often the case, that the number of items n is known in advance). One way is to construct a binomial queue on n items, with each key equal to ∞. The operation *insert*(x) is done by executing *decrease_key*$(x, k(x))$.

THEOREM 1. *Rank relaxed heaps correctly process a sequence of m decrease_key and $k \leq n$ delete_min operations in time $O(m + k \log n)$.*

PROOF. It is easy to check that the algorithm maintains the following invariant: At the start of each call to *promote*, making the edge between a and its parent good gives a valid relaxed heap structure. This implies correctness. (Note that an active node can be good or bad: An entry in the A array starts out as a bad child; it may become good without being processed in a transformation, if the key of its parent is sufficiently decreased.)

For the timing, observe the *decrease_key* routine uses $O(1)$ time and the *delete_min* routine uses $O(\log n)$ time. The time for transformations is bounded by the argument given as the plan for the algorithm (*delete_min* decreases α by one or zero, and so only improves the bound). □

The algorithm of Section 4 uses several other properties of relaxed heaps. We start with an extension of the theorem.

COROLLARY 1. *Any subsequence of k consecutive decrease_key operations is processed in time $O(k + \log n)$.*

PROOF. Let the subsequence start with α_0 active nodes and end with α_k active nodes. Each *decrease_key* increases α by at most one. Hence the number of type (i) transformations (that decrease α) is at most $\alpha_0 + k - \alpha_k$. The number of type (ii) transformations (that are last in their sequence) is at most k. Since $\alpha_0, \alpha_k \leq \log n$ the number of transformations, and hence the total time, is $O(k + \log n)$. □

As noted in the proof of the theorem, a single *delete_min* operation in a sequence uses time $O(\log n)$. Section 4 uses two other priority queue operations: The operation *find_min* (defined in Section 1) is implemented exactly like the first part of *delete_min*. Hence it uses time $O(\log n)$. The operation *delete*(x) can be implemented by

CS 270
Combinatorial Algo

22 Sept (Tuesday)

NETWORK FLOWS — TARJAN

1. Max Flow
2. Aug Paths
3. Residual Graphs
4. Cuts
5. MaxFlow - MinCut Thm
6. Aug Path Algo

$G = (V, E)$ a directed graph
- $s, t \in V$ distinct nodes, source & sink
- a capacity $c(v,w)$ for $(v,w) \in E$

Problem: To find a flow of max value

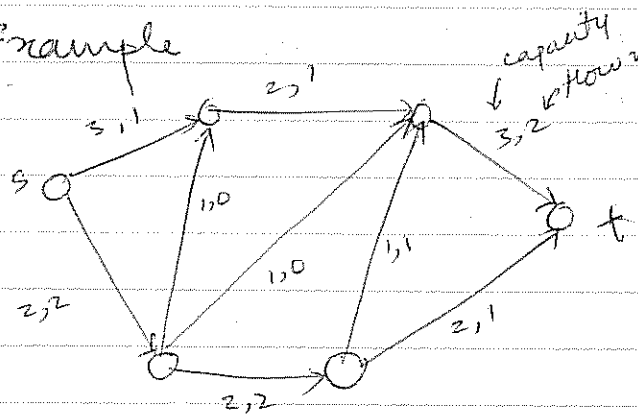
when $f: V \times V \to \mathbb{R}$

f has (i) Skew symmetry $f(w,v) = -f(v,w) \quad \forall (v,w)$

(ii) Capacity constraints $f(v,w) \leq c(v,w) \quad \forall v,w \in V \times V$

(iii) Flow conservation $\forall v \in V - \{s,t\} \quad \sum_w f(v,w) = 0$

Example



← a feasible flow

$|f| = \sum_w f(s,w)$

Def: Augmented Path

P a path from s to t (undirected!)
(set of edges)
→ such that
(i) If any edge $(v,w) \in P$ is directed from s to t (a forward edge) $f(v,w) < c(v,w)$
(ii) If any edge $(v,w) \in P$ is directed from t to s (a backward edge) $f(v,w) > 0$.

∃ aug path ⇒ not optimal flow

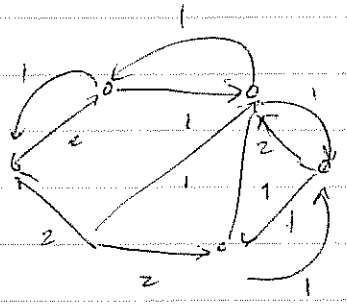
Residual Graph (for f)

(i) $\forall (v,w) \in V \times V$

$$res(v,w) = c(v,w) - f(v,w)$$

(ii) $R_f = (V, E)$

original ↪ $E = \{(v,w) : res(v,w) > 0\}$



How do you come up w/ R.G ?

(i) $f(u,v) = c(v,w)$

(saturated) ⇒ R has the edge (w,v) with

$res(w,v) = f(v,w)$

(ii) If $f(v,w) = 0$

R G has the edge (v,w) with $res(v,w) = c(v,w)$

(iii) If $0 < f(v,w) < c(v,w)$

R has (v,w) with $res(v,w) = c(v,w) - f(v,w)$

R also has edge (w,v) with $res(w,v) = f(v,w)$

AUGMENTING PATH in G ≡ (s,t) path in R

CUT: A cut is a partition of V into disjoint sets X and $\bar{X}$ such that $s \in X$, $t \in \bar{X}$.

capacity of a cut $(X, \bar{X})$

$$= \sum_{\substack{v \in X \\ w \in \bar{X}}} c(v,w)$$

Given f, $f(X, \bar{X}) \triangleq$ Flow across a cut

$$= \sum_{\substack{v \in X \\ w \in \bar{X}}} f(v, w)$$

PROPOSITION

For any flow X, for any cut $(X, \bar{X})$

$|f| = f(X, \bar{X})$

Pf $f(X, \bar{X}) \triangleq \sum_{\substack{v \in X \\ w \in \bar{X}}} f(v, w)$

$$= \sum_{\substack{v \in V \\ w}} f(v, w) - \sum_{\substack{v \in X \\ w \in X}} f(v, w)$$

$= |f|$ $\qquad \neq 0$ ← claim

$$\sum_{\substack{v \in X \\ w}} f(v, w) = |f|$$

$$\because \sum_{w} f(u, w) = 0 \quad \forall u \in V - \{s, t\}$$

For any cut $(X, \bar{X})$, for any flow

$f(X, \bar{X}) \leq$ capacity $(X, \bar{X})$ → $\text{cap}(X, \bar{X})$ = $\sum$ fwd edges

(Weak) Max Flow Min cut Thm:

MaxFlow ≤ Min cut capacity

Max Flow Min Cut Thm

The following are equivalent

(i) f is a max flow

(ii) $\nexists$ an any path

(iii) $\exists$ a cut with capacity $|f|$

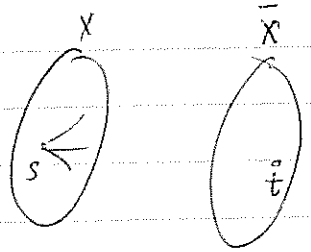
(i) ⇒ (ii) ✓ (proved by negation)

(ii) ⇒ (iii) Proof overleaf

$X = \{ v : v \text{ is reachable from } s \text{ in } R_f \}$

$\bar{X} = V - X$

Observe: $t \in \bar{X}$ (else $\exists$ aug path!)



$$f(X, \bar{X}) = |f| = \sum_{\substack{v \in X \\ w \in \bar{X}}} f(v, w)$$

$\therefore c(v,w) - f(v,w)$ must be ≥ 0

(i) $\forall$ all edges (v, w) $v \in X, w \in \bar{X}$

$f(v, w) = c(v, w)$ ($\because$ Edge $(v, w) \notin R$)

(ii) $\forall$ all edges (v, w) with $v \in \bar{X}$, $w \in X$

$f(v, w) = 0$ ($\because$ Edge $(w, v) \notin R$)

$\therefore$ (ii) $\Rightarrow$ (i)

(iii) $\Rightarrow$ (i) ✓ (Trivial

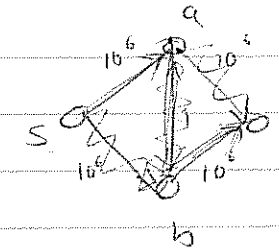
($\because |f| \leq cap(X, \bar{X})$ & we have $|f| \Rightarrow$ max flow)

ALGORITHM FOR MAX FLOW

1. Start with zero flow
2. Repeat till impossible
   FIND Augmenting Path for the current flow and augment as much as possible along the path

(1) If all capacities are integers then the algo terminates w/ max flow
($\because$ increase by 1 unit each time)

Bad Case:



$|f^*| = 2 \times 10^6$

How do you look for good aug paths quickly

(i) Always pick aug path with maximum residual capacity (along which largest increment will be achieved)

(ii) use shortest aug path

CS270
Combinatorial Algos

27 Sept '95
Thursday

f on G $\forall v,w \in V \times V$ $res(v,w) = c(v,w) - f(v,w)$

Residual Graph $R_f = (V, E_f)$
$(v,w) \in E_f \Leftrightarrow res(v,w) > 0$

Example:

G f^* $\|f^*\|$ ← max flow on G
given f
R_f has a max^m flow of $\|f^*\| - \|f\|$

Pf: Let f' be any flow on R_f
$f + f'$ is defined as $f + f'$ on edges

$(f+f')(u,w)$ ← $f(v,u) + f'(v,w)$

$f'(v,w) \le res(v,w) = cap(v,w) - f(v,w)$

$\therefore (f' + f)(v,w) \le cap(v,w)$

$\therefore \|f'\| \le \|f^*\| - \|f\|$
(we exhibit a flow of this amt.)

$f' = f^* - f$ $f'(v,w) = f^*(v,w) - f(v,w)$

claim: f' is a valid flow of R_f

tpt $f'(v,w) \le res(v,w)$

we have $f'(v,w) = f^*(v,w) - f(v,w) \le cap(v,w) - f(v,w)$

Start w/ a zero flow f

until no more aug. path → Find an augmenting path on the residual graph R_f. augment the path as much as is possible.

How do you find an augmenting path?

Find max capacity path

capacity of a path $\triangleq$ capacity of min edge in path.

Theorem (Edmonds & Karp)

Augmenting along max capacity path requires $O(m \log c)$ augmentation steps where capacities are integers & c = max capacity of any edge.

$n = |V|$

$m = |E|$

Lemma Can find a max flow in $\leq m$ steps, where each step augments along one path

Let f^* = max flow

$G^* = (V, E^*)$ $(v,w) \in E^*$ if $f^*(v,w) > 0$

$i \leftarrow 1$

Repeat until G^* has no path from s to t

- Find path p_i from s to t. (max capacity?)
- Let Δ_i be capacity of p_i in G^*
- $\forall (v,w) \in p_i$ $f^*(v,w) \leftarrow f^*(v,w) - \Delta_i$
- remove (v,w) from G^* if $f^*(v,w) = 0$
- increment i

w.? can't loop more than m times (remove one edge w/ each iteration)

$P_1, \ldots, P_m$

$\Delta_1, \ldots \Delta_m$ where are there only m paths

flow f

R_f has a flow of $\|f^*\| - \|f\|$ ← avg

So what? ∃ path in R_f that capacity $\geq [\|f^*\| - \|f\|]/2m$ O.K.!

∴ there has to be a way of getting to max flow in ≤ m steps.

augment along max capacity path 2m times
↳ i.e. Do 2m times: find max cap path (Dijkstra) & augment

max cap path $\leq \|f^*\| - \|f\| / 2m$ ∴ at some point in your 2m iterations this must hold (else start having -ve values!)

∴ if we do 2m iterations, log [value of max capacity path in G] times, we're done.

⇒ $O(m \log c)$

Can find max capacity path in $O(m \log n)$ <using S.P. concepts>

∴ $O(m^2 \log n \log c)$ algo for maxflow.

Polynomial time algo in bits to encode.

But what about poly in (m, n) i.e. (n)

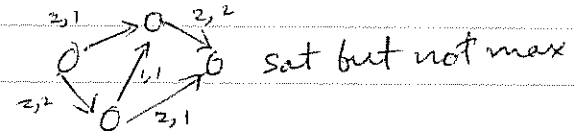
Another idea: augment on S.P. as opposed to minimum capacity path

↖ maximum

DINIC

f is a blocking flow if any path from s to t contains a saturated edge.

Blocking $\not\Rightarrow$ Max



sat but not max

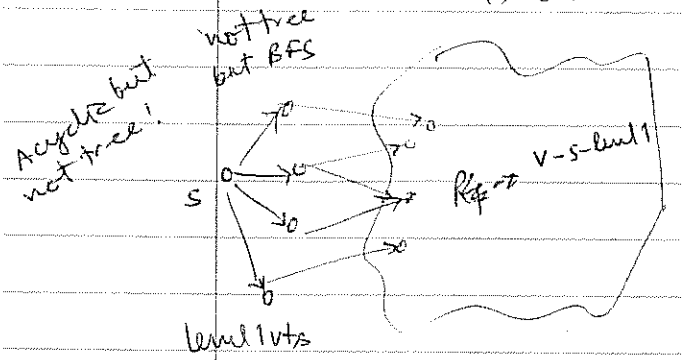
level of a vertex = length of S.P. from s to v

level of (s) = 0

f R_f L_f

L_f (1) contains all vertices reachable from s in R_f

(2) $(v,w) \in E(L_f) \iff level(v)+1 = level(w)$



Properties: level (t) = length of S.P. from s to t.

L_f contains all paths of length level (t) from s to t.

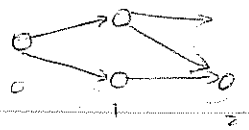
Repeat until no path from s to t in R_f

↑ Find blocking flow in L_f

Augment along blocking flow

Update f

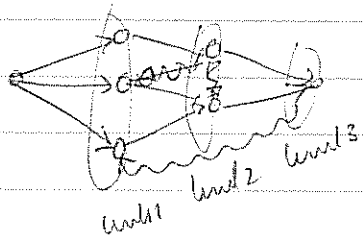
Theorem: DINIC's algo terminates after at most $\leq n-1$ iterations



Before	iteration	After
f		f' = flow in G after iteration
R_f	$\Rightarrow$	$R_{f'}$
L_f		$L_{f'}$
$dist(s,t)=l$		$dist(s,t)=l'$
b in R_f		b in $R_{f'}$

Claim $l' \geq l^*$

Proof: R_f, L_f



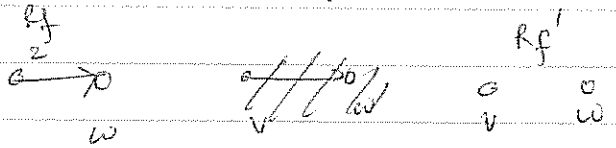
$\longrightarrow \Rightarrow$ edge in L_f

$\rightsquigarrow \Rightarrow$ edge in R_f not in L_f

find blocking flow in L_f

for every [path from s to t in R_f of length c / path from s to t in L_f]

some edge in the path becomes saturated



- every saturated edge dosn't appear in $R_{f'}$
- every edge in $R_{f'}$ is either an edge of R_f or the reverse of an edge in L_f ?

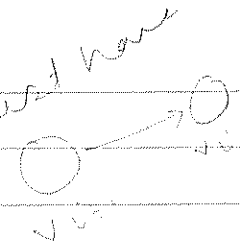
Let $s=v_0, v_1, v_2, \ldots, v_{l'} = t$

Let be a S.P. from s to t in $R_{f'}$

$(v_0,v_1), (v_1,v_2), \ldots (v_{l'-1}, v_{l'-1})$

For some i $(v_i, v_{i+1}) \notin L_f$ (else not blocking in L_f)

Suppose $(v_i, v_{i+1}) \notin L_f$ $\in R_{f'}$



$level(v_{i+1}) \leq level(v_i) + 1$

for every (v_k, v_{k+1})

$level(v_{k+1}) \leq level(v_k) + 1$ (inequality $\Leftarrow$ case 2)

$\Rightarrow l' > l$

QED

CS270 Section 28 Sep '90 Friday

Technique 1

max capacity augmenting path has capacity $\geq \frac{\|f^*\| - \|f\|}{m}$

Claim: augment $2m$ times then max capacity augmenting path has capacity $\leq \frac{\|f^*\| - \|f\|}{2m}$

$O(m \log c)$

↑ yields non poly time bound in N

Dinics Algorithm:

(I) $f = 0$

(II) find R_f

(III) find L_f

(IV) find blocking flow L_f, b_f

(V) $f = f + b_f$

until $t \notin L_f$

$\Rightarrow$ Poly in N

CLAIM: requires $\leq N$ iterations

Q7 max asc. sequence

Given N pts $\in \mathbb{R}^2$

$\{(x,y)\}$ $1 \le i \le N$

find maximal sequence (x_k, y_k) s.t. $\forall i > j$

$\Rightarrow x_k \geq x_j$, $y_k \geq y_j$

(i) Sort on X coordinates

(ii) create table ~~using y~~ s.t.

min j^{th} elt of an increasing seq of length j.

ms[ ] of size N

$\forall i$ set ms[i] = ∞

Maintain invariant

$\forall i$ $\forall k$ if ms[k] $\neq \infty$ at step i of our loop

ms[k] = min k^{th} elt. of an increasing subsequence of the length k embedded in $y_1 \ldots y_i$

for i = 1 to N

find k s.t. ms[k] > y_i

$\forall j < k$ ms[j] $\le y_i$

if we can find, set ms[k] = y_i

end for

<u>Claim</u>: length of longest increasing subsequence

= MAX k s.t. ms[k] $\neq \infty$

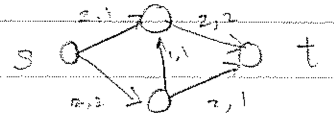
CS-270 — 20 Oct '90 Tuesday

Finding Blocking Paths in Acyclic N/Ws

- DINIC O(nm)
- KARZANOV O(n^2)

f is a blocking flow if

- f is a flow
- any path from f contains a saturated edge

eg. 

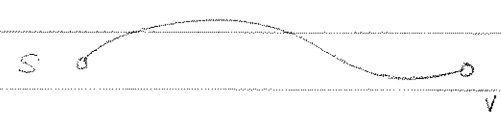
have flow f, R_f

$res(v,w) = capacity(v,w) - f(v,w)$

$(v,w) \in E(R_f) \iff res(v,w) > 0$

level of a vertex : level(v) = length of the S.P. from s to v

$L_f \Rightarrow \begin{cases} V = \text{the set of all vertices reachable from } s \text{ in } R_f \\ E = \text{the set of all } (v,w) \text{ s.t. } level(v)+1 = level(w) \end{cases}$



L_f is acyclic

DINIC Algorithm

start with 0 (zero) flow

Key → Repeat until f is max
find a blocking flow f' on L_f
$f \leftarrow f + f'$

Last time : DINICS algo takes atmost (i.e ≤) n-1 iters

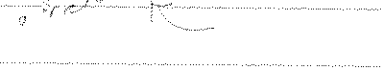
Problem: given G, capacities
G is acyclic
Find a blocking flow on G

Dinic algo for blocking flow

Dumbest way – $O(nm^2)$ (n iterations, each time $O(m)$ time to get a path from s to t.)
m: edges

Smart – $O(mn)$

- **Init** edge $p = [s]$ $v = s$ goto Advance
- **Advance** if there is no edge out of v goto Retreat
  else let (v,w) be an edge out of v
  $p \leftarrow p \oplus [w]$ /* concat path */
  $v \leftarrow w$
  if $v = t$ then goto augment
  if $v \neq t$ then goto advance
- **Augment** let $\Delta = \min_{(v,w)\in p} cap(v,w) - f(v,w)$
  $\forall (v,w) \in p$ $f(v,w) \leftarrow f(v,w) + \Delta$
  delete saturated edges
  /* guaranteed to delete an edge */
  Goto initialize
- **Retreat** if $v = s$ then halt /* done! */
  otherwise let (u,v) be the last edge in p. delete v from p
  delete (u,v) from G
  $v \leftarrow u$



Complexity: Retreat, Adv, Init $\Rightarrow O(1)$ time
Augment $\Rightarrow O(n)$

≤ n advance steps per retreat or augment
≤ m augment + retreat steps (∵ each deletes an edge)
≤ O(mn) advance steps total
≤ O(m) augment or retreat steps

⇒ O(mn) time

DINICS algo finds max flow in O(mn²) time
bad in dense graphs!

KARZANOV finds blocking flow in O(n²) time

f is a __preflow__ f: V×V ⟶ ℝ
(i) f(v,w) ≤ c(v,w)
(ii) f(v,w) = −f(w,v)
(iii) Δf(u) = $\sum_v f(v,u)$ ≥ 0

↳ flow going in is @least (≥) flow going out

Start w/ __blocking preflow__ ⇒ blocking flow

initialize f(s,v) = c(s,v)
f(v,w) = 0 ∀ v≠s

v is unbalanced if Δ(f(v)) > 0 (v ≠ s,t)
v is balanced if v ≠ s,t Δf(v) = 0

__vertex__ v is blocked or unblocked

⇒ if v is unblocked, can be blocked
(but not vice versa.)

Increase Flow (v)

If v is unblocked

Repeat until $\Delta f(v) = 0$

OR $\nexists$ edge (v,w) s.t. w is unblocked

Let (v,w) be unsaturated w/ w unblocked

$f(v,w) \leftarrow f(v,w) + \min\{\Delta f(v), cap(v,w) - f(v,w)\}$

Decrease Flow (v)

If v is blocked

Repeat until $\Delta f(v) = 0$

Let (u,v) be an edge w/ true flow

$f(u,v) \leftarrow f(u,v) - \min\{f(u,v), \Delta f(u)\}$ ← always succeeds in balancing a vtx (can always push it back)

$f(s,v) \leftarrow c(s,v)$

$f(v,w) \leftarrow 0 \quad \forall v \neq s$ (is that vts?) } start w/ s blocked, $v \in V - \{s\}$ unblocked

Repeat INCR, DECR until no unbalanced vtcs

INCR scan vertices in topological order from s to t

{ If v is unblocked & unbalanced increaseflow(v)

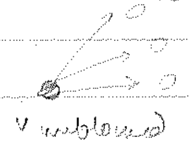
{ If v still unbalanced, make v blocked

DECR Scan vertices in rev. top. order from t to s

Decrease flow (v)

Every blocked vertex v

every path from v to t contains a saturated edge



At end of an INCR pass, there are no unbalanced, unblocked vertices.

At end of decrease phase, every blocked vtx is balanced.

$\overset{0}{\checkmark}$ balanced, blocked

~~After~~ Each INCR phase except last blocks at least one vtx.

$\leq n-1$ INCR phases

$\leq n-2$ balancings per INCR & DECR phase.

$\Rightarrow$ $\leq (n-1)(n-2)$ total balancings

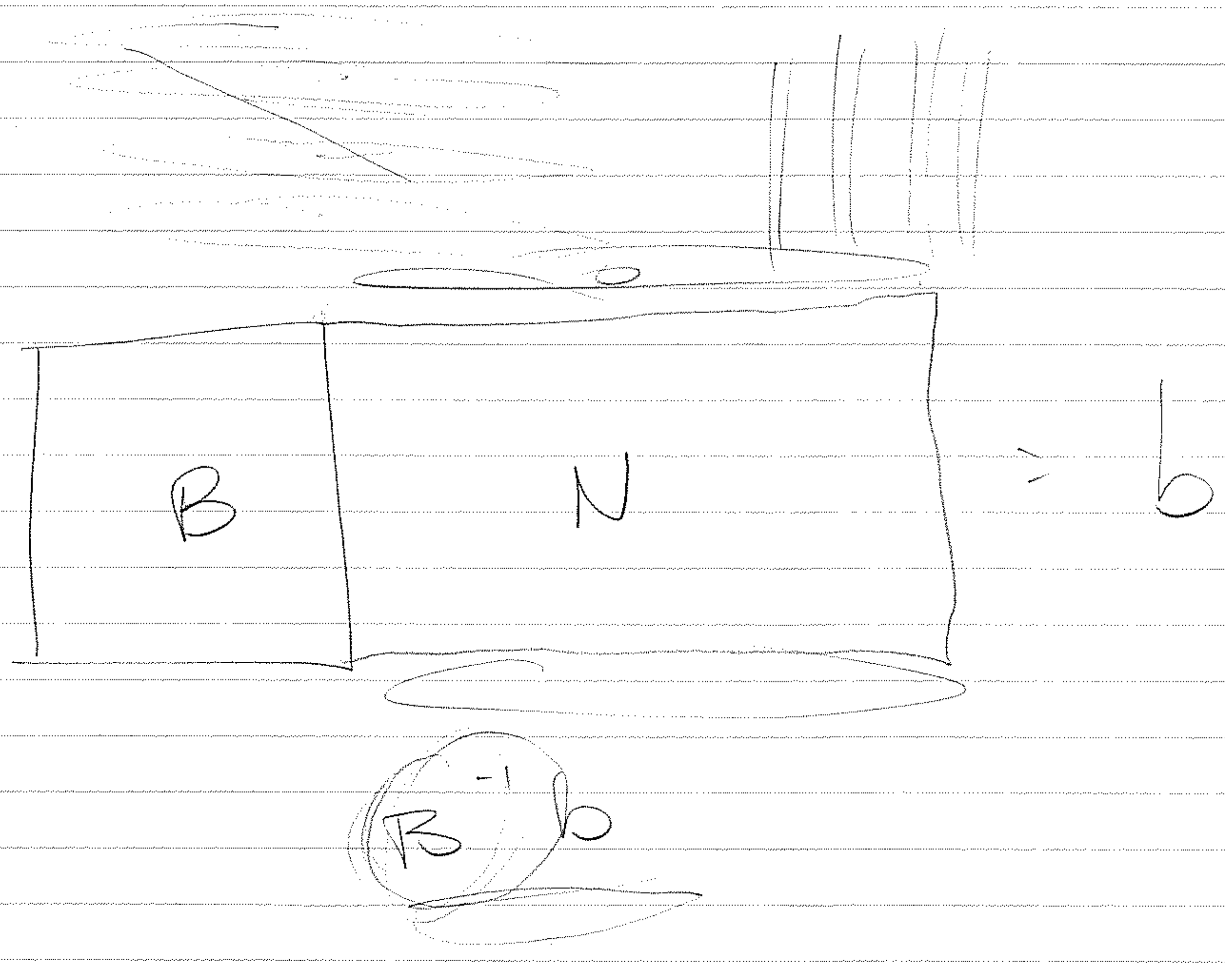
Each push operation either saturates an edge → at most m times

reduces flow to zero (only m times)

or balances a vertex $\leq (n-1)(n-2)$

$m = O(n^2)$

∴ total # of push operations is $O(n^2)$



CS270 4 Oct 1991 Thursday

Max Flow Problem

Review

Push Relabel

Goldberg & Tarjan

JACM 1:35 1988

Flow f

Residual N/W R

$r(u,v) = cap(u,v) - f(u,v)$

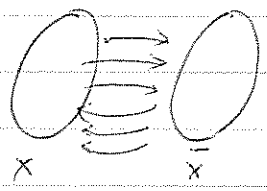
Augmenting Path - directed path in R from source to sink.

Keep augmenting till no augmenting path exists

Thm f is max $\Leftrightarrow$ $\nexists$ augmenting path wrt f.

Thm Cor : max flow min cut theorem

$$\max_{f} |f| = \min_{X, \bar{X}} cap(X, \bar{X})$$



Thm : If all capacities are integral, then $\exists$ an integer max flow i.e. $\exists$ max flow f s.t. $\forall u, v$ $f(u,v)$ is integer

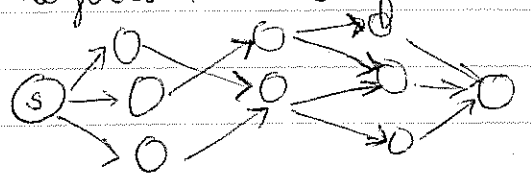
always choose a shortest aug. path.

The distance length of shortest augmenting path never decreased.

Can prove $\leq n$ (augmentations) is $O(nm)$

each $O(m)$ $\Rightarrow$ total algo is $O(nm^2)$

"layered new idea of DINIC" very impt.



Seek a blocking flow

. Can do computation of blocking flow in time $O(mn)$

$\therefore$ Overall time is $O(mn^2)$

Wave Algorithm: Computes blocking flow in an Acyclic directed graph in time $O(n^2)$ rather than $O(nm)$

$\therefore$ can compute max flow in $O(n^3)$ steps

Preflow $f(u,v)$

skew symmetric $f(u,v) = -f(v,u)$ mathematical notation - just a convenience

$f(u,v) \leq c(u,v)$

if $u \neq s,t$ $\leq f(v,u) \geq 0$

"math preflow then bring it down to a flow" idea of this algo.

$\Delta f(u) = \sum_v f(v,u)$

Push-Relabel Algorithm

Maintains two functions

Preflow $f(u,v)$

Distance $d(u)$ non negative integer

If $(u,v) \in R$ then

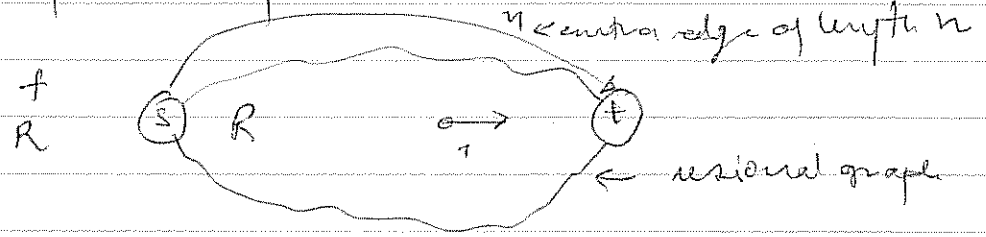
$d(u) \leq 1 + d(v)$

$d(s) = n$ (number of nodes)

$d(t) = 0$

$d(\cdot)$ is a "sort" of distance from source.

More precisely look at invariants



$d(u)$ ≤ length of a shortest path from u to t in this subgraph

Vertex u is "active" if $\Delta f(u) > 0$

Edge $(u,v) \in R$ is "admissable" if $d(u) = 1 + d(v)$

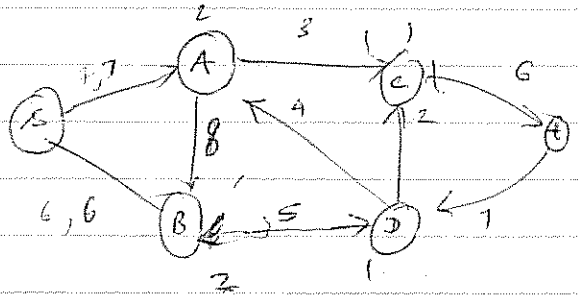
"PUSH"

If ∃ an admissable edge (u,v) then increase $f(u,v)$ by amt $\min(\Delta f(u), res(u,v))$

else $d(u) = 1 + \min_{(u,v) \in R} d(v)$ ("RELABEL")

Initial preflow: saturate all edges out of s, all other edges have flow 0.

initlabels $d(u)$ = min length of $u \to t$ path in R ($O(n)$: BFS)



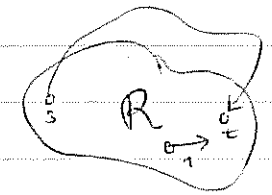
initial values

A	7
B	7
c	0
y	0

A	7 4 ∅ 4	A is dead changes to 3.
B	6 ∅ 5 1	
C	∅ 3	:worm through this
D	1 ∅ 5	

lemma: If f is a preflow and u is a vertex with positive excess i.e. $e(u) > 0$ then R, the residual graph wrt f includes a directed path from u to the source.

Pf: ~~distance~~ $d(u) \leq 2n-1$



∴ distance labels don't get too big

can obtain upper bounds on # of times we do things in this algorithm

- # of relabel steps $\leq (n) \times (2n-1)$
  - ↑ # vtcs
- # of pushes
  - saturating pushes
  - non saturating pushes

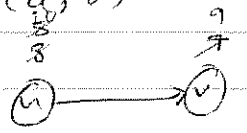
if edge admissable $\min\{e(u), res(u,v)\}$ (?)

Separately count up sat & non sat pushes

Number of saturating pushes on (u, v)

$\leq 2n$

∴ total # of sat pushes $\leq 2mn$



Number of nonsaturating pushes ?

# of saturating pushes $\leq 2nm$

Φ : potential function

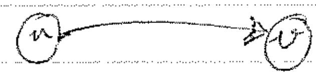
f, d

$$\Phi = \sum_{u \text{ is active}} d(u)$$

Relabel Step: $d(u) := d(u) + k \Rightarrow \Phi$ increases by k ($\uparrow k$)

Increases in Φ due to relabelling sum to $\leq 2n^2$

Saturating push $\uparrow \Phi$ by at most $2n-2$



$\therefore$ total increase in Φ due to sat pushes is $O(n^2 m)$

Non saturating push decreases Φ by at least 1.

Put together $\Rightarrow$ # of non sat pushes $= O(n^2 m)$

$\therefore$ troublemakers are non sat pushes.

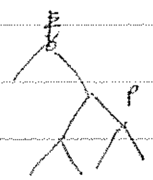
CS270 9 Oct 1990

Review Session

Comparison of Prim & Dijkstra Algorithms

MST

T



follows from cycle / cut property

P is such that P has min max $w(e)$ paths $e \in$ path

always refer to charac!

Prims Algo $X = \{s\}$ set of vertices in the comp that we have constructed so far

$E = \emptyset$ = set of edges in the component that we have constructed so far.

$v = s \quad h = \emptyset$

while $|E_T| < N-1$ do

$\forall w \in \Gamma(v)$

if $w \notin h$ insert(h, w, weight(v, w))

$p(w) = v$

... weight(v, w) < weight(($p(w)$, w))

decrease-key (h, w, weight(v, w))

$p(w) = v$

v = deletemin (h)

$X = X \cup \{v\}$

$E_T = E_T \cup \{(p(v), v)\}$

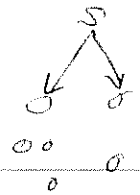
until which

at most (m) deletemins

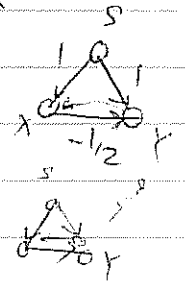
(m) decrease keys

$\therefore \left[m \log_{\lceil m/n+2 \rceil} N \right]$ time

Dijkstra's algorithm



Single source shortest path problem



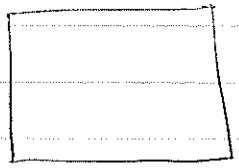
If we take $G = (V, E)$

and for every vertex $v \neq s$,

draw the shortest path

Question: Dynamic Programming

Given n matrices



$(\quad)(\quad)(\quad)$

CS270

4 Oct 90
Tuesday

- PUSH RELABEL ALGORITHM
- EXCESS SCALING
- BIPARTITE MATCHING (Tarjan Ch9)

Review session 3 Evans 8pm
Binomial Queues not in course

Push Relabel Algos

- Maintain preflow $f(u,v)$
- Distance fn $d(u)$
- Initial f: saturate all arcs out of s.
  all other flows are set to zero.

$d(s) = n \quad d(t) = 0$

$d(u) =$ min'm # of arcs in a path from $u \to t$.
{easy to compute, eg BFS}

Invariant Properties

- R doesn't contain a path from $s \to t$
- If $\Delta f(v) > 0$ then R contains a path from v to s.
- if $(u,v) \in R$, then $d(u) \leq 1 + d(v)$
  ↳ physical interpretation. (heights, flow under gravity)

v is active if $\Delta f(v) > 0$

$(u,v) \in R$ is admissable if $d(u) > d(v)$
(& its a prop of algo that its never ignored, so $d(u) = 1 + d(v)$ is equiv. charac.)

Choose active vertex v

If there is an admissable arc $\overset{(v,w)}{\vec{}}$ directed out of (v) in R
then $f(v,w) := f(v,w) + \min(\Delta f(v), res(v,w))$
"push step"

else $d(v) := 1 + \min_{(v,w) \in R} d(w)$
"relabel step"

Went through a little proof to show well defined.

All distance labels $\leq 2n-1$

$\therefore$ No. of relabeling steps $\leq 2n^2$ (∵ each vtx can be relabeled only $2n$ times)

After every non sat. push, the vtx is not active!

the number of saturating pushes $\leq 2nm$

Number of non saturating pushes $= O(n^2 m)$

Proof: Potential function Φ

$\Phi \uparrow$ by labelling

sat pushes

$\Phi \downarrow$ by non saturating pushes

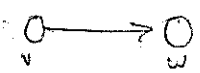
$\Phi = \sum_{\{v \mid v \text{ active}\}} d(v)$

- Initial potential is non negative
- Final potential $= 0$

Relabelling $d(v) := d(v) + k \Rightarrow \Phi \uparrow k$

Accounts for $\uparrow$ in Φ by $\leq 2n^2$

Saturating push



worst case: v remains active, w becomes active.

$\Rightarrow \Phi \uparrow d(w)$; but $d(w) \leq 2n-2$

$\uparrow$ in potential due to saturating pushes

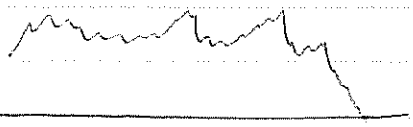
$\leq 2nm(2n-2) = O(n^2 m)$

Non saturating Push 

$d(v) = 1 + d(w)$

~~Reduces $d(v)$ (?)~~

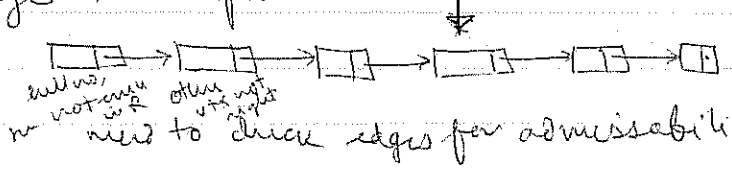
$\uparrow \Phi = d(w) - d(v) = -1$, in the worst case

Φ 

The # of times it goes down
(i.e # of non sat pushes)
$\leq \Sigma$ increases $\leq 2n^2 + O(n^2 m) = O(n^2 m)$
$+ \Phi_{init}$
$\leq O(n^2)$

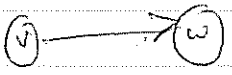
The non saturating edges are crucial.

DISCHARGING a vertex
- active vertex v
- Edge list for v



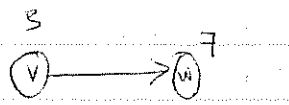
Key Obs: when we ~~reach~~ reach the end of the list, relabelling is necessary.

Proof:

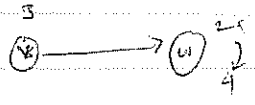


either $res(v,w) = 0$
or $d(v) \leq d(w)$

$d(w) \uparrow$



$d(w) \uparrow$
non admissable



non admissable either

change this ~~to~~ to relabelling step. (n)

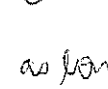
change to each relabelling step $= O(n)$
# of label steps is $O(n^2)$
$\therefore$ upper bound on total amt of nodes work in $O(n^3)$

Time Bound: $O(n^2 m)$
$\therefore$ same as Dinic. more than work

Choose active vertex w/ maximum distance label
Process until it becomes inactive or relabelling occurs
* of non sat pushes $O(n^3)$
∴ Time bound is $O(n^3)$
Actually $O(n^2\sqrt{m})$ good when not dense

PHASE Maximal period during which largest label of any active vertex is fixed.
We will show:
(1) * of phases $\in O(n^2)$
(2) * of non saturating pushes per phase $= O(n)$

Suppose max label of active vertex is b



as long as pushing, only will lay off all excessive flow.
∴ 2 kinds of phase
- phases in which relabelling occurs
- phases in which no relabelling occurs

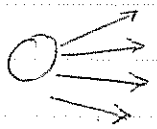
* of phases in which relabelling occurs $= O(n^2)$

to analyse * phases in which no relabelling occurs:
Use potential function

$$\Phi = \max_{\{v \mid v \text{ active}\}} d(v)$$

$d(v) := d(v) + k$ $\Phi \uparrow k$
Sum of increases $\leq 2n^2$
Each phase w/o relabelling $\downarrow \Phi$ by 1
$\Phi_{final} = 0$ $\Phi_{init} \leq n-1$
∴ * of phases w/o relabelling $= O(n^2)$ ∴ host ind source

# of non saturating pushes per phase $= O(n)$



- One non saturated push per vertex per phase
- $\therefore$ result is $O(n^3)$ non sat. pushes

- Same time bound as wave algo.
- But has several points in its favour
  - Actually $O(n^2\sqrt{m})$ so better for non dense
  - Conceptually simpler (no layered n/w needed to be set up)
  - Lends itself to parallelism
  - Faster in real life

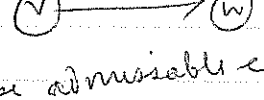
Heuristic: prefer to push a large amount of flow.

Leads to "Scaling the Excess"

- Works only when capacities are integers
- But U = maximum capacity. Then the time bound for this algo is $O(nm + n^2 \log U)$ (vs $O(n^3)$)
- Unless U is exp in n, this is superior!

Δ Power of 2.

- Maintained as an upper bound on maximum excess of any active vertex
- Proceed by cutting Δ in half.
- Maintains integral pseudo flow
- Stop when $\Delta = 1/2$ $\therefore$ by integrality all excesses are zero



Suppose admissible edge

$f(v,w) := f(v,w) + \min(\Delta f(v), r_f(v,w), \Delta - \Delta f(w))$ (same amount)

- Same time bound – again a bound on # of non sat pushes

- Heuristic choose for large excess, small label choose active vertex v of smallest (opp. of prev) label s.t. $\Delta_f(v) > \Delta/2$.
  Existence guarantee.

Non sat push: $(\Delta f(v), res(v,w), \Delta - \Delta f(w))$

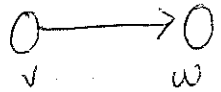
$\geq \frac{\Delta}{2}$ (↑) $\quad \geq \frac{\Delta}{2}$ (↑ this is why choose smallest label!)

∴ nonsat pushes at least $\Delta/2$
(not $res(v,w)$ ∵ its Non sat push)

Potential fn used in time bound -

$$\Phi = \sum_{\{v \mid v \text{ active}\}} d(v) \cdot \left\{\frac{\Delta f(v)}{\Delta}\right\}$$

Non saturating push ↓ Φ by @ least $\frac{1}{2}$



Excess ↓ $\geq \frac{\Delta}{2}$ — Excess ↑ by same amount

Conclusion : # of non sat pushes $= O(n^2 \log v)$
∴ Time Bound $O(mn + n^2 \log V)$

CS270 Oct 16, 199

Matching Theory
Taijan chq

A matching in a graph G is a set of edges, no two of which meet at a common vertex.

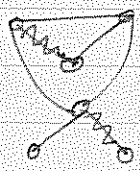


fig 1

← a maximum matching!

Perfect Matching: covers all vertices

M ——— matched edge

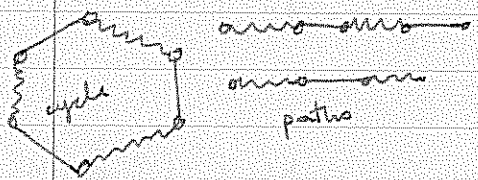
——— free edge

matched vertex covered by M

free vertex not covered by M

When is a matching maximum?

An alternating path (cycle)
is a simple path (cycle) whose edges are alternately matched and free.



Defn: An augmenting path is an alternating path between two distinct free vertices

fig 1 doesn't exhibit an augmenting path

simple augmenting path ⇒ can increase matching size.

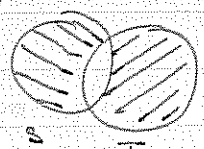
Obs 1: ∃ an augmenting path for a matching M, then M isn't a maximum matching. (Pf: can augment)

How about converse?

Let M be a non-maximum matching
Let $\bar{M}$ be a maximum matching
Let $k = |\bar{M}| - |M|$
$=$ amount by which M fails to be a max matching

then there are k vertex disjoint augmenting paths relative to M.

Proof: Symmetric Difference Notion
$= (S - T) + (T - S)$



$S \oplus T = (S \cap \bar{T}) \cup (\bar{S} \cap T)$

Consider $M \oplus \bar{M}$ (call other edges of graph as thrown away)

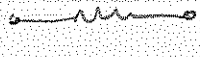
Look at an arb. vtx

There are 4 possibilities:

$M \oplus \bar{M}$ ← what are the components of this?

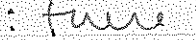
- isolated vertices
- alternating cycle / alternating paths

alt paths: 3 kinds

1. 
2.  none possible ($\because |\bar{M}| \geq |M|$)
3. X 

$\therefore$ each of 2. is an augmenting path

must have k of these

($\because$ there are k more  (3 edges) than  (squiggles))

(each of type 2. adds one edge than squiggly)

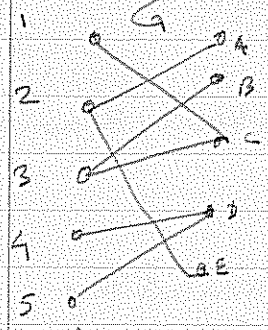
Claim: about the maximum size of alternating paths (at most $n/k - 1$)

Cor: M is a max. matching ⇔ ∄ augmenting path for M

Suggests a relation between matching theory & flow theory.

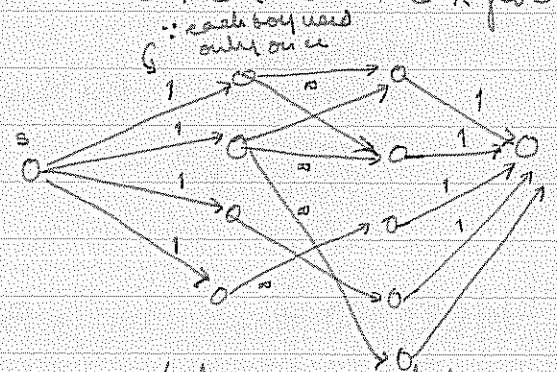
Only if G is bipartite

G is bipartite ⇔ vertex set can be partitioned into A, B such that each edge joins a vertex in A with a vertex in B.



Ex Graph is bipartite ⇔ G has no odd length cycle

Formulation as max flow problem -

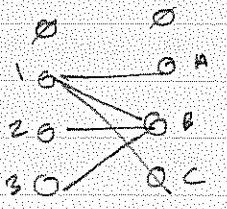


∃ a 1-1 correspondence between matchings in G and integral s-t flows in N. The size of the flow matching = value of flow

Recall: Integrality Thm. Integer capacities ⇒ max flow of all integers

Value of max flow in N = Size of max matching in G.

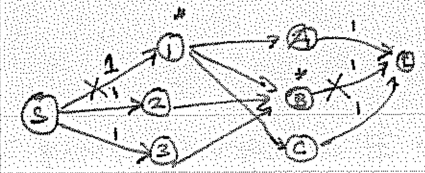
A vertex cover in G is a set of vertices that touches all the edges.



{1, B} is a vertex cover

∃ a 1-1 correspondence between vtx covers in G and s-t cuts of finite capacity in N. The size of the vertex cover = capacity of the corresponding cut.

{1, B} = vertex cover



cut of finite capacity

{1, B} form vertex cover so (s,1) & (B,t) cut removes all paths from s to t.

The actual partition is $X = \{s, 2, 3, B\}$ $\bar{X} = \{1, A, C, t\}$ (;-))

$X = \{s\} \cup$ BOYS \ COVER $\cup$ GIRLS $\cap$ COVER
$\bar{X} = \{t\} \cup$ BOYS $\cap$ COVER $\cup$ GIRLS \ COVER

have to define in this way so as to get right capacity (it will be finite no matter what you do as long as {s,t} are ... {t,B} ...)

Putting this together, can get a thm about Bipartite Graphs

G		N
Size of max^m matching	=	value of Maxflow
	‖ by Maxflow-Mincut Thm	
Size of min^m vertex cover	=	capacity of min^m cut

∴ KÖNIG-EGERVARY thm in a Bipartite graph

Size of max^m matching = Size of min^m vtx cover

isolated vtx → no loss of generality by incl. in matching *)

Can represent a Bipartite Graph as a matrix!

	A	B	C
1	(x)	x	x
2		(x)	
3		x	

Boys ↔ Rows
Girls ↔ columns
In this context, matching = set of ind. x's (i.e. no two in same line (row(col))

A vertex cover in this representation is a set of lines that covers all the x's.

Max^m # of independent x's = min no. of lines to cover all the x's

Application of KETHM: "Assignment Problem" Bipartite graph of n boys n girls each edge has a weight. Desire perfect matching of min wt. Assume complete WLOG (large wts).

	A	B	C	D
1	5	2	2	1
2	1	6	7	4
3	1	6	5	3
4	1	8	9	7

must be independent!
∴ n indep cells of min^m total cost.

*

Small

. Bipartite Matching
  . computation of maxmatching, min vertex cover
  . assignment problem
  . max ind set = min edge cover
  . applications
    . VLSI
    . Dilworth's thm

. Asgn 6 corrections
  (c) replace "minm" by "maxm"
  . put X in 4-C position

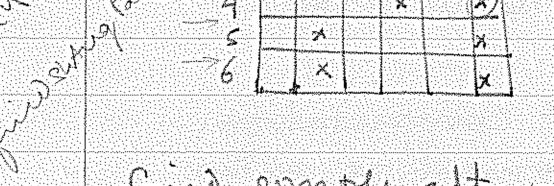
K.E. Thm : Size of maxm matching = Size of minm vertex cover

. Do directly on Bipartite graph
. Do on assoc. flow n/w.

	A	B	C	D	E	F
1	x	(x)	x		x	
2	x	x		(x)		x
3		x		x		
4				x		(x)
5		x				x
6		x				x

← could do by aug path method on flow network
We choose to work on this directly.

Find simple alt. path between free vertices.
start by getting a maximal matching



A is free
C is free
E is free

← level graph

3 — B ~ 1 — A

each girl has only one wiggly edge!

⊙ shortest aug paths

Can do blocking flow calculation in O(m) (∵ can erase all edges of a path)

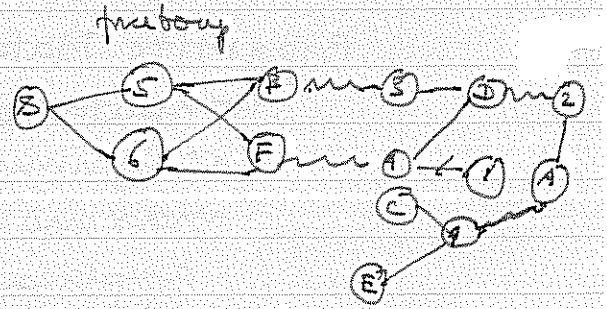
Ester said: Start w/ unmatched boys; get level graph starting from them will guaranteed to get aug path (if any) in O(m)
⇒ how to find starting paths

⊙ when you form the BFS tree you've got all the shortest aug paths

something about blocking flows.

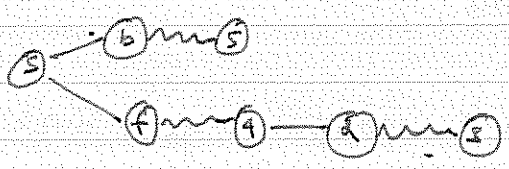
Becomes

	A	B
1	✗	✓
2	✓	✓
3		✗
etc		



Add (5,B) (3,D) (2,A) (1,C)
Add (3,B) (2,D) (1,A)

∴ new matrix . .
can't have any more augmentations (? (How can actually because not max^m length yet.)



; search for aug path fails
∴ we have max^m matching

max^m vtx cover: reached girls + unreached boys
(KE Thm) {B, D, F, 1, 2}

Is this a vertex cover? (just checking)
; Think about girl who is not reachable from any boy & whether she will be in vtx cover.

Phase 1: Maximal vertex cover computation
Set of shortest augmenting paths

After each phase, length of shortest aug path ↑,
length of aug path is always odd.
So it increases by at least 2!

Each phase can be executed in time $O(m)$
(∵ can erase all edges in path)
How many phases?
$O(\sqrt{n})$
Time Bound $O(m\sqrt{n})$

cs270

Recall Thm: Matching M $\quad$ n: # of vertices

Max Matching $\bar{M}$

$K = |\bar{M}| - |M|$

$\exists$ k vertex disjoint augmenting paths relative to M

$\therefore$ one of them has to contain atmost n/k vertices.

$\exists$ an augmenting path of length $\leq n/k - 1$

$\therefore$ there will be @ least one short augmenting path.

1, 3, 5, . . .

How many phases will take place before length of the shortest augmenting path $\geq \sqrt{n}$

Ans: At most $\frac{1}{2}\sqrt{n} + 1$

Assume length of shortest aug path $\geq \sqrt{n} - 1$

$\frac{n}{k} - 1 \geq$ length of shortest aug path $\geq \sqrt{n} - 1$ (after $\sqrt{n}$ iterations)

$\therefore$ $\boxed{k \leq \sqrt{n}}$

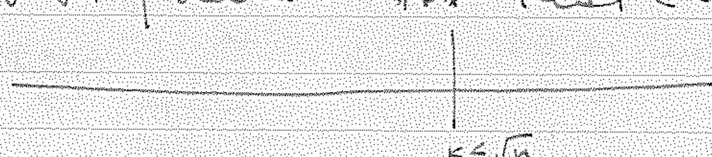
How many more aug. would you do now $\rightarrow O(\sqrt{n})$ ($\because$ k aug at most and $k = \sqrt{n}$)

At most $\sqrt{n}$ further phases

$\therefore$ $O(\sqrt{n})$ total # of phases

How does k vary as algo progresses?

first $\sqrt{n}$ phases $\quad$ Rest ($\leq \sqrt{n}$ phases)



Assignment Problem:

$n \times n$ matrix c_{ij}

find n indep cells in the matrix of min^m total cost.

eg. $\begin{bmatrix} 5 & 2 & 2 & 1 \\ 1 & 6 & 7 & 4 \\ 1 & 6 & 5 & 3 \\ 1 & 8 & 9 & 7 \end{bmatrix}$

problem doesn't change if we add a constant (+/-) to a row.

(∵ have to pay exactly one elt of a row. So all costs change by same amount)

ditto column

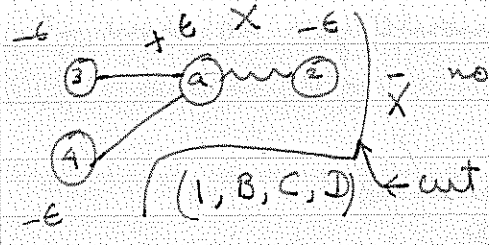
Transform to non negative matrix w/ n indep. zeros.

How can we use matching theory to solve this?

. subtract row minima from each row

$$\therefore \begin{bmatrix} 4 & 1 & 1 & 0 \\ 0 & 5 & 6 & 3 \\ 6 & 5 & 4 & 2 \\ 0 & 7 & 8 & 6 \end{bmatrix} \xrightarrow{\text{col. zeros}} \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} a & b & c & d \end{array} \\ \begin{bmatrix} 4 & 0 & 0 & 0 \\ 0 & 4 & 5 & 3 \\ 0 & 4 & 3 & 2 \\ 0 & 6 & 7 & 6 \end{bmatrix} \end{array}$$

How would we know max^m # of ind zeros → max matching



no pre-gils ∴ maximal

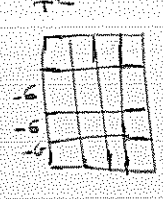
wish to progress across the cut.

Goal: create solid edge of cost 0 from X to $\bar{X}$

reduces cost of forward edges (increases those coming back)

choose ϵ just large enough to get a zero edge across cut!

∴ $\epsilon = \min \begin{matrix} 4 & 5 & 3 \\ 4 & 3 & 2 \\ 6 & 7 & 6 \end{matrix}$ i.e. 2

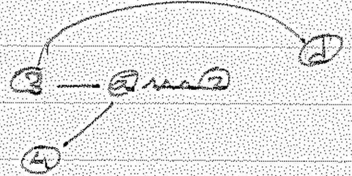


new matrix →

	a	b	c	d
1	6	0	0	0
2	0	2	3	1
3	0	2	1	0
4	0	4	5	4

p3
CS270
0018

look at only any path



; d is a free girl ⇒ have succeeded in augmenting.

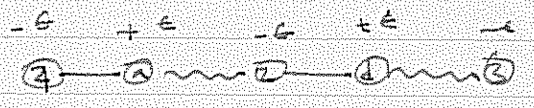
now have 3 ind zeroes

	+ε a	b	c	d
1	6	0	0	0
2 −ε	0	2	3	1
3	0	2	1	0
4 −ε	0	4	5	4

→

	a	b	c	d
1	7	0	0	0
2	0	1	2	0
3	1	2	[illegible]	0
4	0	3	4	3

greatest augmenting or something?

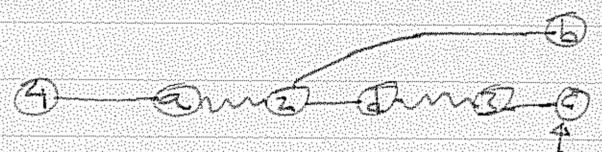


obtain

	+ε a	b	c	+ε d
1	7	0	0	0
−ε 2	0	1	2	0
−ε 3	1	2	1	0
−ε 4	0	3	4	3

→

	a	b	c	d
1	8	0	0	1
2	0	0	1	0
3	1	1	0	0
4	0	2	3	3



add 4a 2d 3c
unmark 2a 3d

→

8	0	0	1
0	0	1	0
1	1	0	0
0	2	3	3

Hungarian algo for the assignment problem

can do in $O(n^3)$

for Bipartite Graphs | not general

K-E Thm: size of max^m matching = size of min vertex cover

↕

Size of max ind set = size of min^m edge cover

ind. set = set of vertices, no two adjacent
Edge cover = set of edges that touch all vertices.
n = No of vertices in the graph

S = set of vertices
$\bar{S}$ = complement of S

| S is independent ⇔ $\bar{S}$ is a vertex cover

Proof: S ind ⇔ no edge touches 2 elts of S. ⇔ Each edge touches at least one elt of $\bar{S}$ ⇔ $\bar{S}$ is a vertex cover

(Graph assumed to be connected i.e. no isolated vertex (else no edge cover))

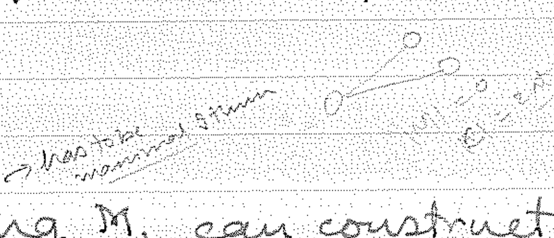
from which it follows

size of Minimum Vtx cover
+ size of Maximum ind set = n } any connected graph

Show size of max^m matching + size of a minimum edge cover = n

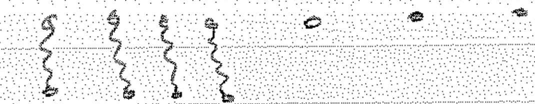
(n is total # of vertices in graph)

then done



Given a matching M, can construct an edge cover C s.t. |M| + |C| = n } any graph

Ex: do the rest (?)



each edge of M covers 2 vtxs
∴ n − 2|M| remain.

for each edge of uncovered vertex, take an edge & add this to the matching edges → there are n − 2|M| ?? of this

|C| = n − |M|

(any graph) ∴ given a matching M of size |M| we can associate an edge cover of size n − |M|. Therefore ∃ an edge cover of size n − |Mmax|. So size of min^m edge cover ≤ n − |Mmax|
Now consider any edge cover C. It can be decomposed into a forest of trees, the trees being connected components (if there were cycles, remove an edge (will still remain edge cover)) so now we pick an edge from each tree to get M a matching
# of vertices in a tree = 1 + # of edges. So, since every vertex is in some tree, can conclude |C| + |M| = n (n = total # vertices in graph)
⇒ given C can find M s.t. |M| = |n| − |C| so |Mmax| ≥ n − |Cmin|
QED

But size of Max Matching = size of min^m vtx cover
only if Bipartite so can say |MIS| = |min edge cover|

Oct 23, '90

/* EECS244 */
CAD for ICs
CS270

Adnan Aziz

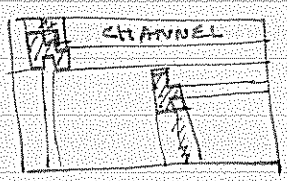
Applications of Bipartite Matching
- Decompose into rectangles
- Dilworth's Theorem
- Tactical Problem
- Nonbipartite Matching

Bipartite Graph G w/ no isolated vertices:
Size of maximal set of vertices =
Size of min edge cover

MIS = complement of Min Vertex Cover

Minimum Edge Cover can be derived from maximum matching

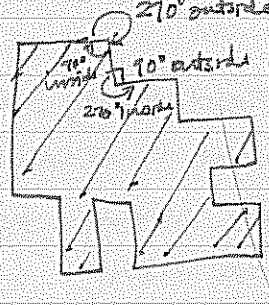
(VLSI design)
chip



~~rectangular~~ modules: rectilinear polygon
in side are // or ⊥ to sides of chip.

Problem: Divide open area into min # of rectangles
(Preparation for wire routing)

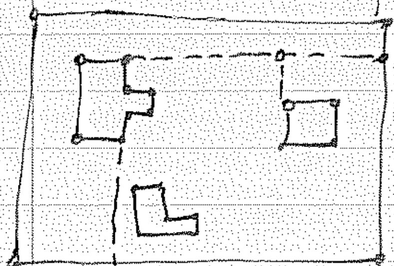
Given a module:



Concave vtx: must draw a horizontal line or a vertical line segment from each concave vertex.

Theorem: Minimizing the # of rectangles is equivalent to minimizing the # of segments drawn.

Proof ↓



$c = 4$; $f = 4$

- - - - later $c = 2$; $f = 5$

can view as a graph
c = # of connected components
f = # of faces (regions) into which the rectangle is divided

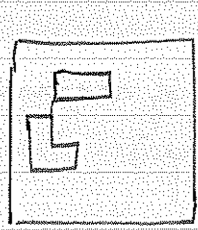
. decrease # of components, ~~increase~~ faces same
: faces increase components unchanged

each segment increases $f - c$ by 1 || $c\downarrow 1$ // $f\uparrow 1$

segments = ~~f(x)~~ $(f_{FINAL} - c_{FINAL}) - (f_{INITIAL} - c_{INITIAL})$;
(c_{FINAL} = 1 known; $f_{INITIAL}$, $c_{INITIAL}$ known)

final # of ~~ch~~ faces = # channels + # modules
∴ minimize # of channels by minimizing # ~~ch~~ faces

∃ 2 kinds of segments → kill two birds w/ one stone



minimize # segments = max #

A segment is good if it joins two concave vertices.

Good Segments: Good Horz Segments
Good Vertical Segments

~~Good segments may conflict (i.e. cross)~~

→ Problem: a good horz segment will cross a good vertical segment.

p2 | 270
oct 23

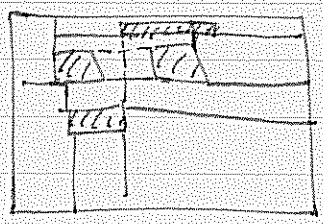
- We draw only one segment from each concavity

Bipartite graph:

Boys are good horiz segments
Girls are good vertical segments

Adjacency ⇔ crossing

Look for an MIS; throw away then finish off any bad seg.

example 

DILWORTHS Theorem:

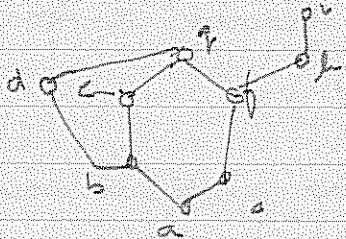
A partial order is a binary relation denoted $\prec$ that is transitive & and antisymmetric.

Transitive: if $a \prec b$ & $b \prec c$ then $a \prec c$.

Antisymmetric ⇒ if $a \prec b$ then $b \not\prec a$
(in particular, $a \not\prec a$)

Finite only

use Hasse diagram to decide



; all edges are directed upward.

a and b are comparable if $a \prec b$ or $b \prec a$, incomparable otherwise

A chain is a set of mutually ~~x~~ comparable elts

An anti chain is a set of mutually incomp. elts

max chain is {a, c, f, h, i}

chain may skip elts!

max antichain (min) {d, e, f}

Min cover by disjoint chains

Any antichain gives lower bound on # of chains needed to ~~cover~~ cover

{a, b, d, g}

{e}

{c, f, h, i}

Min cover by antichains

need @ least 5 (∵ ∃ a chain of length 5)

{a} {d, e, f}

{b, c} {i}

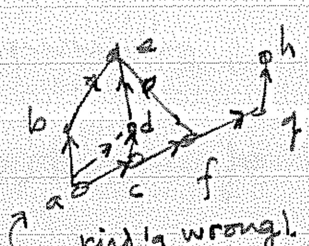
{g, h}

Min # of (disjoint) antichains to cover all elts
= max size of chain

Easy - on H.W.

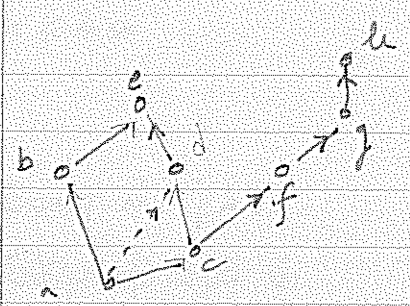
<u>Dilworth</u> : Min^m # of (disjoint) chains to cover all elts
= Max size of antichain

connect to a matching problem somehow.



kind'a wrong!

p3 CS270
Oct 23, 1990

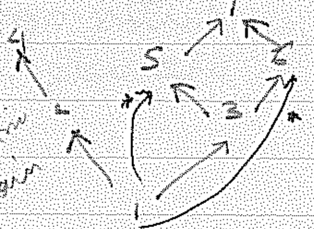


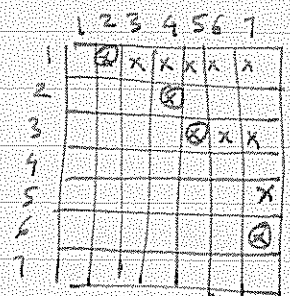
for each elt i

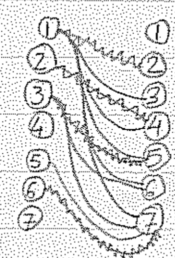
u_i i as predecessor   (?)

v_i i as successor

Edge $\{u_i, v_j\}$ if $i < j$ → transitive closure







$1 \to 2 \to 4$ — 4 has no successor in matching

$3 \to 5$ — 5 has no successor in matching

$6 \to 7$ — 7 has no successor in matching

. how to generate matrix
. how to generate chain

∴ the decomposition into chains is $\{1,2,4\}$ $\{6,7\}$ $\{3,5\}$

↑ elts who have no successor in the matching (?)

# of chains = # of such elts

n = No. of elts in the partial order

From any matching M, can get decomposed into $n - |M|$ chains. (see the matching matrix)

(∴ $n - |M|$ boxes (goals) not matched! ... every chain ... in this fashion ends at an unmatched box (is that goal?))

(1) ⇒ wish to be able to say, from any decomp into t chains can get a matching of size $n-t$.

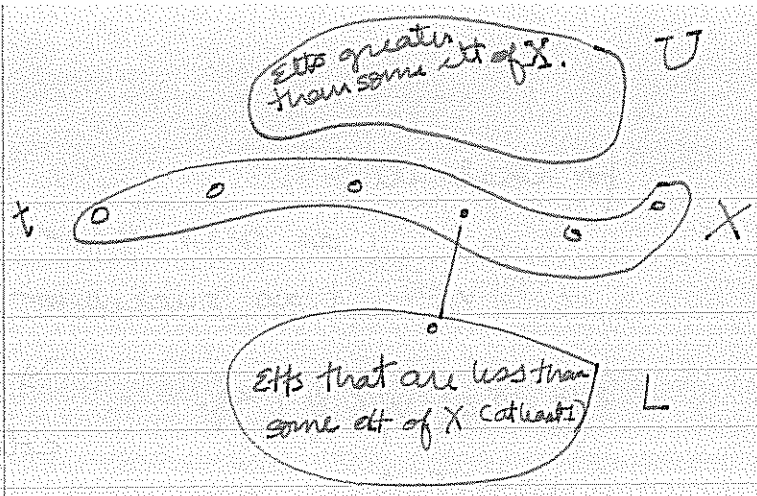
Putting together, min^m # of chains needed to cover all elts = n − SIZE OF MAX MATCHING

(2) ⇒ would like to prove (n − size of min vertex cover) = max size of an antichain. (Then done by KE thm) ↖ in a bipartite graph of size $2n$ $\{u_1, u_2, \ldots u_n\} \cup \{v_1, \ldots v_n\}$

Given a maximal antichain X of size t, can get a vertex cover of size $n-t$. (for the bipartite matrix corresp to transitive closure!)

(not really nec.)

i.e. every elt outside X is comparable to some elt in X



$a < x_i$
$\Rightarrow a \not> x_k$ all k!

$i \lessdot j$ in partial order
in Bip Graph,
$\{u_i, v_j\}$
Vertex cover must
include u_i or v_j.

Vertex cover will be
$\{u_i \mid i \notin L\} \cup \{v_j \mid j \in U\}$
cardinality $= n - t$ correct!

Must check. if $i \lessdot j$ (so $\{u_i, v_j\}$ is an edge)
then either $i \in L$ or $j \in U$.

Ex. Given a minimum vertex cover of size t, we can construct an antichain of size $n-t$.

p1 25 Oct CS270

EECS244
IC CAD - Logic Synthesis

25 Oct '90
Thursday

a fn is UNATE positive in X iff $\bar{x}$ doesn't appear in any prime of F.

(but may appear in cover. Think about this.)

i.e. F unate ⇒ cover unate

but Cover Unate ⇒ F unate (clear)

→ view as symbolic

f unate in x ⇒ Every essential prime unate in X

$uv + \bar{u}w = 1 \;\forall u \Leftrightarrow v = w = 1$

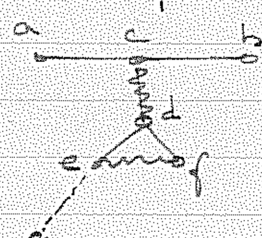
CS270 - Comb Algo

25 Oct '90
Thursday

Non Bipartite Graph Matching
Tarjan Ch 9

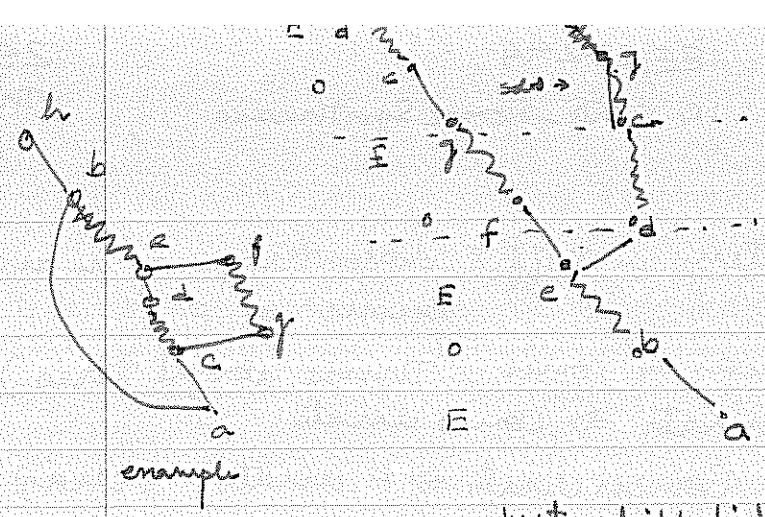
M is a maximum matching
⇔ There is no M-augmenting path
(General Fact - has nothing to do w/ bipartite)

How to find any path.
Complication: odd cycles



how can we search for augmenting paths?

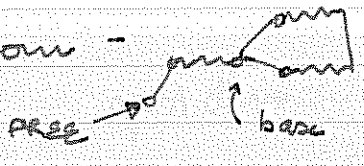
dont want to look at all paths.
compromise on # of times allow a node in our search structure twice only once on odd level once on even.
↳ problem: could get non simple paths
so revised strategy: once even once odd & simple paths only
(by prevent a vtx from being ancestor of itself



but still didn't work! missed any path

1965 Edmonds "Paths, Trees, & Flowers"

Blossom -



blossoms are cause of trouble
algos shrink blossoms down to a single vertex.

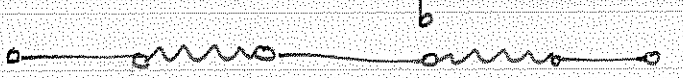
when a blossom occurs
shrink it!

Lemma: Let G', M' be obtained from G, M by shrinking some blossom b.

G' has an M' augmenting path | nontrivial
iff G has an M-aug path

we prove if G' has an M' aug path then G' has an M aug path

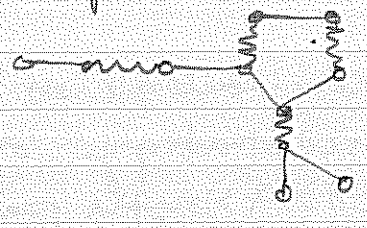
G'



I b doesn't occur in f : trivial - same aug path
II b occurs internally to path
III b is free

very similar

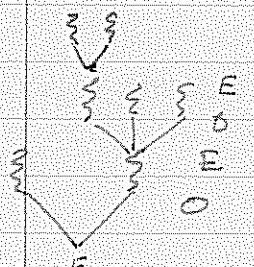
II. Try to show an aug path



insert into the path the even length section of t
this is an augmenting path also in the original graph

Build Alternating Trees from all free vertices.

initially o o o o o o o o



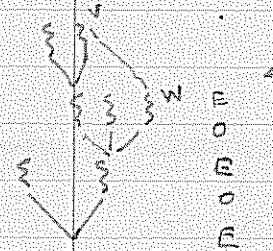
While there is an unexamined edge, $\{v,w\}$ incident with an even vertex v do:

1) w is an even vertex, v and w are in different trees.

Augment (success!)

(v,w) ← But might be blossoms so would have to expand recursively

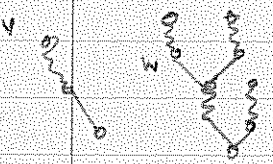
2) w is an even vertex, v and w are in the same tree



∴ have formed a blossom

Shrink it

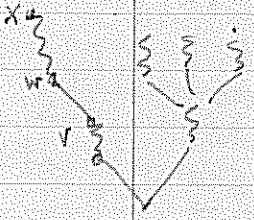
3) w is odd - do nothing



4) w is neither even nor odd

∴ can't be not

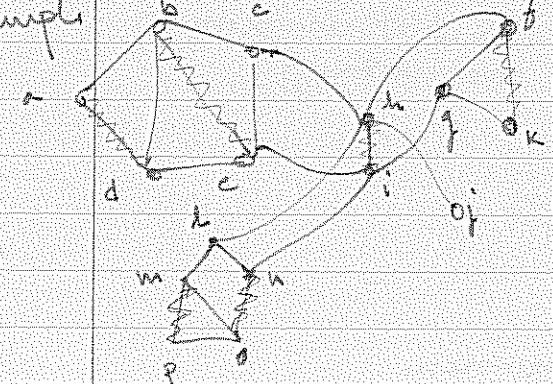
w is matched to some vertex x



Add (v,w) & (w,x) to the tree

(x couldn't already be in the tree - small argument to this effect)

Example

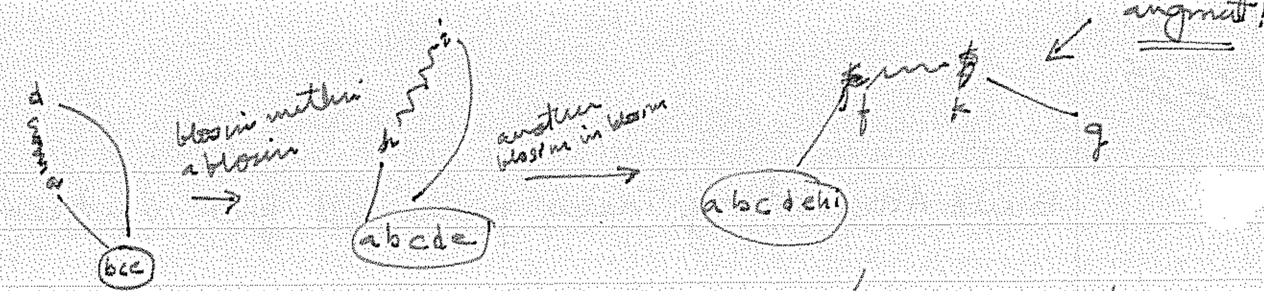


Put down an initial matching by inspection

∴ blossom

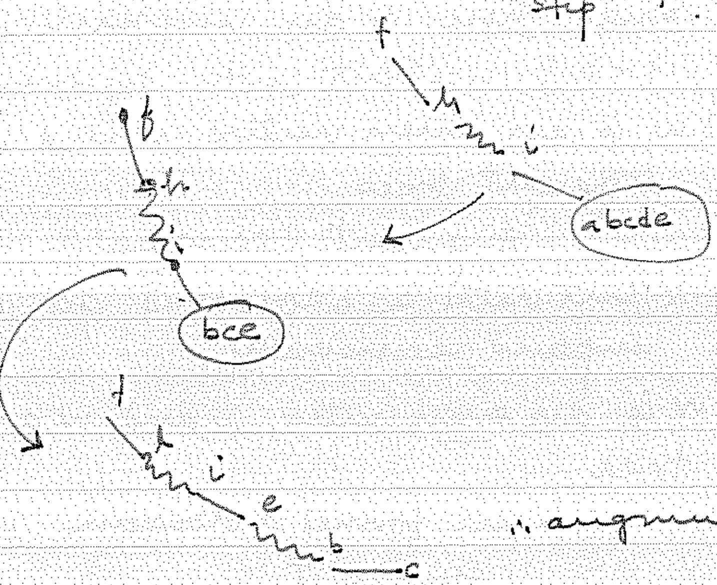
⇒ quit

augment!



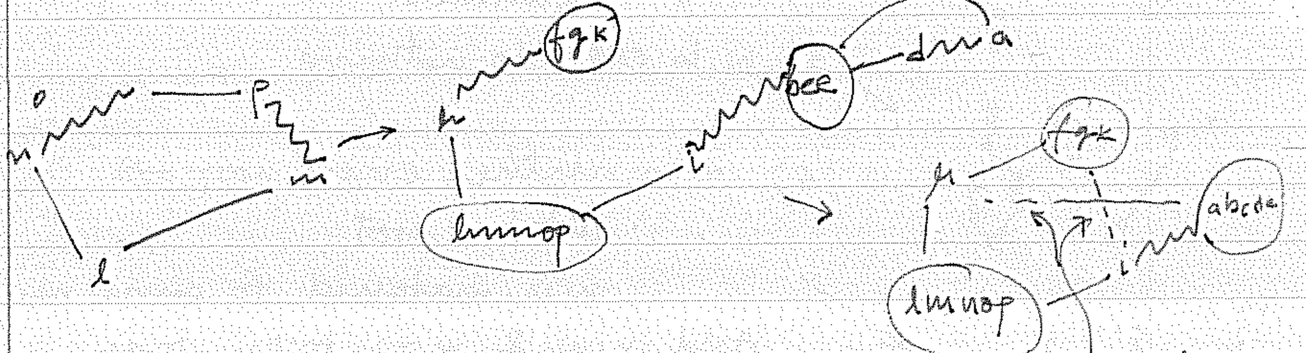
must unshrink!

unshrinking step . note which one we unfold

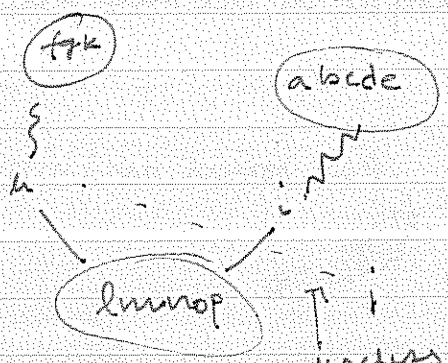


∴ augment along this in graph

now



useless edges
∴ don't need in search structure.
Hide it.



← start at remaining free vertex (j)

↑ i useless

search terminates

∴ matching is maximum
(Thm at start of class.)

p3
cs270
oct25

Thm: Since no aug in this structure, matching is max.

Stuck when the algorithm fails to augment the matching M in original graph is maximum

---

Odd set cover: A collection of vertices $v_1, v_2, \ldots v_r$ and odd cardinality sets of vertices $S_1, S_2, \ldots S_t$ s.t.

each edge of the graph is either incident with some v_i or has both end points in some S_j.

$\{l, m, n, o, p\}$	2
$\{f, g, k\}$	1
$\{a, b, c, d, e\}$	2

odd set cover

Capacity (v) = 1

$Cap(S) = \lfloor \frac{|S|}{2} \rfloor$

Cap (cover) = sum of capacities

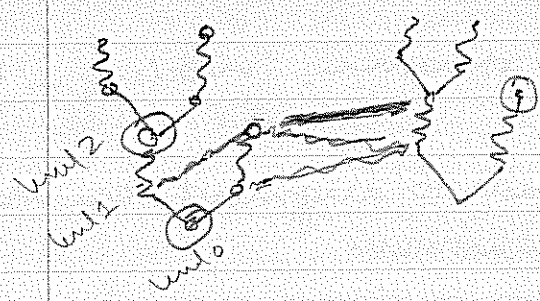
$\therefore$ Cap (cover) = 7 = 2 + (2 + 2 + 1)
(vertices, sets)

~~Thm: Size of any matching M~~

Let M be a matching and C an odd set-cover then $|M| \leq cap(C)$

Examine graph he gave
easily the argument is general.

Clearly if $|M| = cap(C) \Rightarrow M$ is maximum & C is min cap cover!

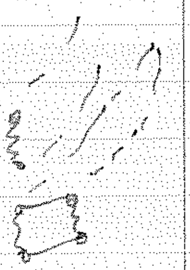


how do we pull out. odd set cover?

vertices - odd vertices

odd nts - maximal blossoms

Easy to check size of matching
= capacity of cover



, one "lil" detail: this may not be whole graph.
there may be a perf. matched component.
↳ perf ⇒ even # of vertices
so pick one singleton vtx and the
rest to be the odd set.

CS270 30 Oct 90

Comb. Algorithms

Linear Programming

Simonnard

Dantzig

Chvátal

NonBipartite matching: conclusion

"oddSet cover Thm" - For any graph

$$\max_M |M| = \min_C \text{cap}(C)$$

M ranges over matching

C ranges over oddset covers

- V covers incident edges

  cap(v) = 1

S

- addset S covers edges with both endpoints in S

cap $\lfloor |S|/2 \rfloor$

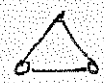
K-E Theorem:

For any bipartite graph

$$\max_M |M| = \min_K |K|$$

M: matchings

K: vertex covers



$\max_M |M| = 1$

smallest non bipartite graph

$\min_K |K| = 2$

See the need for odd set covers

Data Structures: See Jayan $O(n^3)$ max matching algo

maintain blossom } disjoint set data structure is
maximum } at the heart

Bipartite: $m\sqrt{n}$ (Blocking flow algo)

LINEAR PROGRAMMING

Recall $\max_f |f| = \min_{X,\bar{X}} cap(X,\bar{X})$

$\max_M |M| = \min_K |K|$

odd set cover

Dilworth Thm.

} all instances of the duality theorem of L.P.

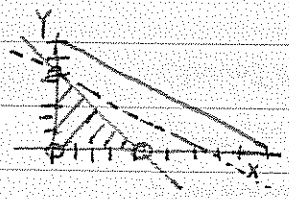
L.P.

maximize or minimize a linear fn subject to linear inequalities

$\max\ 2x+3y$

s.t. $x \geq 0 \quad y \geq 0$

$x + y \leq 4$



Feasible region is hatched

Plot locus of $2x+3y = 18$, go down till

Corners of feasible region: extreme pt.

Optimal value occurs at an extreme point

CS270
Oct 30 1990

in high dimensional spaces,
polytope & hyperplane

R^n : n dimensional real space
A set of points is called convex $\Leftrightarrow$ if it contains x and y then it contains line segment $\overline{xy}$

Hyperplane: $\{x \mid a_1x_1 + a_2x_2 + \cdots a_nx_n = b\}$

closed half-space $= \{x \mid a_1x_1 + a_2x_2 + \cdots a_nx_n \geq b\}$

POLYTOPE - intersection of a finite number of half spaces

CONVEX POLYHEDRON = HD polytope

clearly region of feasible solns of an LP is a polytope

Review of linear algebra

primal LP: max $c \cdot x$ subject to $Ax \leq b$

dual LP: min $y^T b$ subject to $y \geq 0$ $y^T A = c^T$

let $\bar{x}$ be feasible primal
and $\bar{y}$ be dual feasible
then $c^T \cdot \bar{x} \leq \bar{y} \cdot b$

Proof: $c^T x \le \bar{y}^T b$

Pf look at $y^T(A\bar{x})$; $A\bar{x} \le b$

$y \geq 0 \rightarrow$ ...

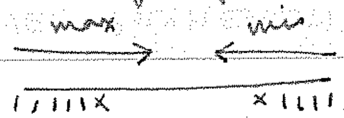
" $\le y^T b$

also $(\bar{y}^T A)\bar{x}$

$= c^T \bar{X}$ ≤

$\therefore c\bar{x} \le \bar{y}b$

values of objective fns:



Duality thm: they meet.

Classification of LP

. INFEASIBLE max x $x \geq 2$, $x \le 1$

. FEASIBLE, nonoptimal (UNBDD obj fn) max x $x \geq \emptyset$

. Optimal Solution: ~~max~~ min x, $x \geq 0$

Duality Theorem: Exactly one of 3 cases holds

① Primal has Optimal solution $\bar{X}$, Dual has optimal soln $\bar{Y}$ and $c\bar{X} = \bar{Y} \cdot b$

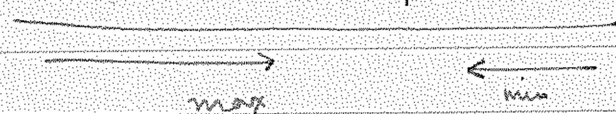
② Both infeasible

③ One is unbdd other is infeasible
(has ∞ obj fn value)

P3
CS270
Oct 30, 1990

If primal has unbdd obj fn then dual is infeasible

$c\bar{x} \le \bar{y}b$



dual can't have feasible soln for dual as primal can be made arb large (& vice versa)

Suppose primal has optimal feasible solution then dual has optimal feasible soln $\bar{y}$, and the obj fn values agree.

Extreme Point of a Pointset: Point which doesn't lie on a lin seg. which lies in set.

Optimal value doesn't change when slack constraints are removed.

$\alpha_1 \cdot x \le b_1$
$\cdot \quad \cdot \quad \cdot$
$\alpha_k \cdot x \le b_k$

} in LP. hold tight at opt. extreme point

$\Rightarrow c \cdot x \le \delta$

Need to invoke FARKAS Lemma: If

$\alpha_1 x \le b_1$
$\cdot \; \cdot \; \cdot$
$\alpha_k x \le b_k$ $\Rightarrow c \cdot x \le \delta$ then $\exists\; y_1 \ge 0 \ldots y_k \ge 0$ s.t. $c = \sum y_i \alpha_i$ || $\delta \ge \sum y_i b_i$

$\hookrightarrow \therefore (\sum y_i \alpha_i) \cdot x \le \delta$

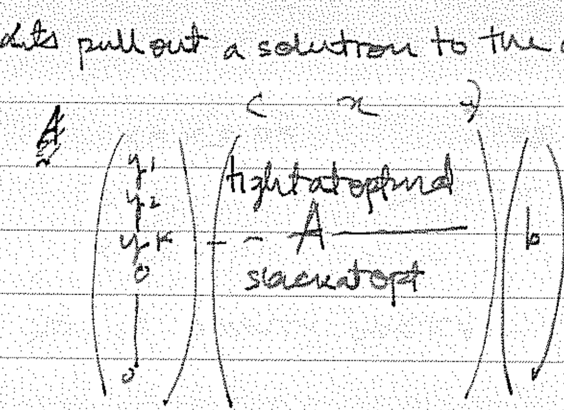
$\Rightarrow \sum y_i (\alpha_i \cdot x) \le \delta$ (position b_i)

$\Rightarrow \sum y_i b_i \le \delta$

y is dual feasible

gives us we construct a dual feasible soln (i.e. $y \ge 0$, $y^T A = c$) that $y^T b \le \delta = c\bar{x}$ for dual $\Rightarrow$ y is opt (feasibility)

lets pull out a solution to the dual



$y \geq 0$ ✓

$y^T A = c^T$ ✓

$\therefore$ dual : $y^T b \leq \delta$ ← optimal value for primal.

$\therefore$ must be $y^T b = \delta$

$= c^T x$

CS270 Nov 1, 1990 Thursday

DUALITY THM & APPLICATIONS

Prog. - Plan / Resource Utilization

MidTerm 2: Nov 20

final: Dec 17

Primal Problem

$\max c \cdot x$

$Ax \le b$

Dual Problem

$\min y \cdot b$

$y^T A = c^T;\ y \geq 0$

$\bar{x}$ feasible soln to primal

$\bar{y}$ feasible soln to dual

Scalar $\bar{y}^T A \bar{x} \le \bar{y}^T b$, $= c^T\bar{x}$

natural question - when will equality hold? i.e.

$\bar{y}^T A \bar{x} = c^T \bar{x}$

so $\bar{y}^T A \bar{x} \Rightarrow \le \bar{y}^T b$

$\bar{y}^T (b - A\bar{x}) = 0$ (nonneg, nonneg)

equality $\Leftrightarrow \bar{y}^T b = c^T \bar{x} = \bar{y}^T A \bar{x}$

$\Leftrightarrow \bar{y}^T (b - A\bar{x}) = 0$

Scalar prod = 0 $\Leftrightarrow$ must have $\bar{y}_i > 0$ only if $(b - A\bar{x})_i = 0$

positive dual variable corresponds to tight primal constraints.

"complementary slackness" condition.

$\bar{x}$ primal feasible

$\bar{y}$ dual feasible

$\bar{y} \cdot (b - A\bar{x}) = 0$

CSC ⇒ optimal

but we still have to show optimal ⇒ CSC

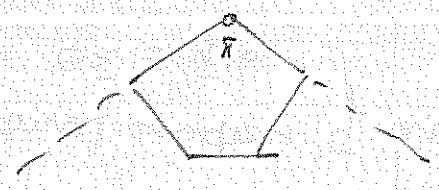
$\therefore \bar{x}, \bar{y}$ must be optimal for their respective problems

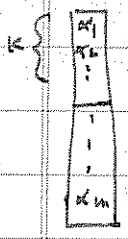
We want to show if primal program has optimal solution $\bar{x}$ then dual has optimal solution $\bar{y}$ and the objective functions agree i.e. $c \cdot \bar{x} = \bar{y} \cdot b$

what we did so far did not show that if primal has opt $\bar{x}$ then dual has optimal soln $\bar{y}$. All we showed was $\bar{y}^T b \ge c^T\bar{x}$ given $\bar{x}, \bar{y}$ opt.

// in a nutshell to show ($\exists \bar{y}$ optimal for dual we invoke Farkas lemma on optimal structure of primal to get dual feasible $\bar{y}$ s.t. $c\bar{y} \le b\bar{x}$ ($\Rightarrow c\bar{y} = b\bar{x}$ too)

$\begin{pmatrix} c \\ b \end{pmatrix}$ primal variables $\quad \begin{pmatrix} x \\ y \end{pmatrix}$ dual variables





opt values determined by tight constraints

$a_1 x \le b_1$

$\vdots$ opt soln at $\bar{x}$

$a_K x \le b_K$

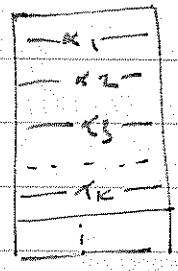
then c is a positive l.c. of $a_1, a_2, \ldots a_K$

iff $\exists\ \bar{y}_1, \bar{y}_2, \ldots \bar{y}_K \ge 0$

s.t. $c = \sum_{i=1}^{K} \bar{y}_i a_i$ — formal way

$y \ge 0$ (Farkas)

$y^T A = c$; constructive

A 

dual feasible by inspection

$\bar{y} \ge 0$ ✓

$\bar{y}^T A = \sum_{i=1}^{K} \bar{y}_i a_i = c$

invoke complementary slackness here.

CSC is sat $\Rightarrow$ $\bar{y}$ is dual optimal

TO show c is true l.c. . .

invoke Farkas lemma.

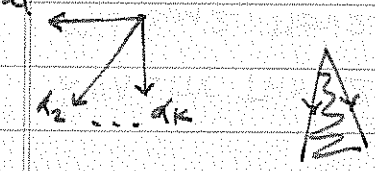
Farkas lemma:

Given Polyhedral cone determined through $a_1, \ldots a_K$

then either

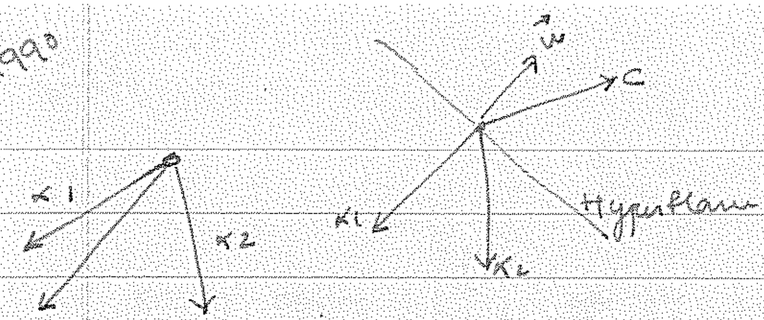
(1) c is in the cone determined by $a_1, \ldots a_K$

$\equiv$ (2) $\exists$ a hyperplane through origin that



cone is $\sum_{i=1}^{K} y_i a_i$; $y_i \ge 0$

P.2
CS270
Nov 1 1990



If $\vec{c}$ outside cone generated by $\vec{\alpha_1}, \vec{\alpha_2}$ ... then can find a hyperplane which has all points in cone on opposite side to that of c so use dot product rule

1 possibility

Given $\alpha_1, \alpha_2, \ldots \alpha_k, c$

either $\exists y_1, \ldots y_k \geq 0$ s.t.

$c = \sum y_i \alpha_i$

or (Analytic Geometry give HP with normal $\vec{u}$ recall c on one side or other of HP based on sign of dot product $u \cdot c$)

other possibility

← u is outside all the hyperplanes

iff $\exists u \quad \alpha_i u \leq 0 \quad i = 1, 2, \ldots k$; and $c \cdot u > 0$

max $c \cdot x$ ← forget at $\bar{x} \Rightarrow$ replace $\alpha_i \cdot \bar{x}$ by b_i

constraints $\alpha_1 x \leq \alpha_1 \cdot \bar{x}$

⋮

$\alpha_k x \leq \alpha_k \cdot \bar{x}$

Assumption: Maximum for this LP occurs at $\bar{x}$.

desired

Conclusion: c is a positive linear combination of $\alpha_1, \ldots, \alpha_k$

Suppose for contradiction that the conclusion is false

then $\Rightarrow$ $\exists u$ s.t. $\alpha_i u \leq 0 \quad i = 1, \ldots k$

and $c \cdot u > 0$

Proof: define x^* by $u = x^* - \bar{x}$

$\alpha_i \cdot (x^* - \bar{x}) \leq 0 \quad i = 1, \ldots k$

but $c\,(x^* - \bar{x}) > 0$

$\Rightarrow \alpha_i x^* \leq \alpha_i \bar{x}$; $i = 1 \ldots k$

but $c x^* > c \cdot \bar{x}$

max $2x_1 + 3x_2$

$x_1 + x_2 \leq 4$

$x_1 \geq 0$

$x_2 \geq 0$

massage $\Rightarrow$

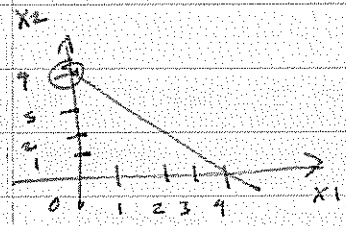
Primal

max $2x_1 + 3x_2$

$x_1 + x_2 \leq 4$

$-x_1 \leq 0$

$-x_2 \leq 0$



$x_1 = 0$

$x_2 = 4$

$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} \quad A = \begin{bmatrix} x_1 & x_2 \\ -1 & 0 \\ 0 & -1 \end{bmatrix} \quad \begin{bmatrix} 4 \\ 0 \\ 0 \end{bmatrix} b \quad \checkmark \quad \checkmark$$

$(2 \;\; 3)$ ↑ c

Dual

min $4y_1$

subj. to. $y_1 \geq 0 \quad y_2 \geq 0$

$y_1 - y_2 = 2$

$y_1 - y_3 = 3$

the constraints that were tight were true ($\checkmark$)

$y_1 (1, 1) + y_2 (-1, 0) = (2, 3)$

solution is $y_1 = 3 \quad y_2 = 1$ (the constraints were tight)

$y_3 = 0$ (non tight $\Rightarrow$ comp slack $= 0$)

dual soln $\begin{pmatrix} 3 \\ 1 \\ 0 \end{pmatrix}$ obj fn value $= (3\;1\;0)\begin{pmatrix} 4 \\ 0 \\ 0 \end{pmatrix} = 12$ $\checkmark\checkmark$

$x_2 = -\frac{2}{3}x_1 + m$

p3
CS 210
Nov 1 1990

$\max cx$ $\quad$ $\min y \cdot b$

$Ax \le b$ $\quad$ $y \ge 0$ $\quad$ asymmetric

$y^T A = c$

In general can mix in following sense

Primal var x_j

$x_j \ge 0$ → sign constraint variable

else → not sign constraint

$\sum a_{ij} x_j \le b_i$ inequality

$\sum a_{ij} x_j = b_i$ equal

Primal variable	$\Leftrightarrow$	Dual Constraint
$x_j \ge 0$		$\sum_i y_i a_{ij} \ge c_j$
Primal constraint		Dual Variable
$\sum_j a_{ij} x_j \le b_i$		$y_i \ge 0$
$=$		y_i ±

Rule: sign constraint variable ⇒ inequality

unconstrained variable ⇒ equality

objective function always the same

primal: min always $\le$

dual: max always $\ge$

theorems go through

dual of dual is primal

c.s.c go through (albeit more complicated)

$\bar{x}$ primal feasible

$\bar{y}$ dual feasible

C.S.C. for every sign constrainted primal variable x_j

$$\bar{x}_j \left(c_j - \sum \bar{y}_i a_{ij}\right) = 0$$

for every sign constrained dual variable y_i

$$\bar{y}_i \left(b_i - \sum a_{ij} \bar{x}_j\right) = 0$$

conclusion of feasibility & slackness:

$\bar{x}, \bar{y}$ are optimal $\Rightarrow c \cdot \bar{x} = \bar{y} \cdot b$

Diet Problem

nutrients $1 \ldots m$ — Vit A, B, thyamin

feeds $1 \ldots n$ — Hay, oats

b_i daily req of nutrient i

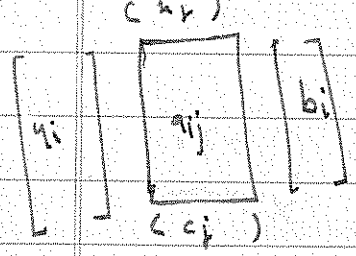
c_j cost per unit of feed j

a_{ij} amount of nutrient i in one unit of feed j

Let x_j = amount of feed j ; wish to minimize $\sum c_j x_j$

$x_j \geq 0 \ \forall j$; $\forall i \ \sum a_{ij} x_j \geq b_i$.

→ there always something instructive to learn from dual



dual: max $\sum y_i b_i$

$0 \leq y_i \ \forall i$

$\forall j, \ \sum y_i a_{ij} \leq c_j$

p4
CS270
Nov 1, 1990

Interpretation of dual:

What if sell only nutrients?

How do you set prices

y_i → price per unit of nutrient i

manufacturer wishes to price things so that farmer gives no feed only nutrients.

$\sum y_i b_i$ ← profit

$\sum y_i a_{ij} \le c_j$ ← must sell pills more cheaply than from feed i.e. could get value of hay from pill for less $ cost of hay

↓

farmer can simulate each feed no more exp than cost of feed

BIPARTITE

Graph $G = (V, E)$ formulate maximum matching problem

x_e ← variable for each edge

$= 0$ if $e \notin M$

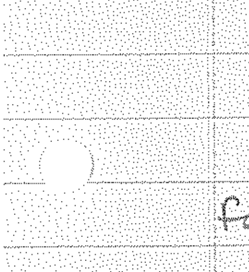
$= 1$ if $e \in M$

fractional possibility?

Obj fn: max $\sum x_e$

Constraints (1) $x_e \ge 0$

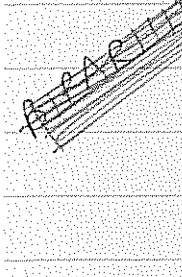
(2) for each vertex v, $\sum_{e \text{ incident with } v} x_e \le 1$



Dual: exactly 2 1's per column! min $\sum y_i$

$y_v \ge 0$

for each edge (u,v), $y_u + y_v \ge 1$ (∵ primal was [illegible] constrained)



→ Yields KE thm! (* not so fast - only in Bipartite are we guaranteed results)

HW: in bipartite case both primal & dual have optimal solutions in which each comp is 0 or 1

EECS 244: IC-CAD 6 Nov '90 Tuesday

y: Boolean N/W

x_i Primary inputs n "satisfiability" don't care
y_i "internal" variables m don't cut = $\leq$
f_i whose output is y_i
F_i Primary outputs

Q: How many points are there in the don't care set?

→ a const. dep. only on n and m.

• How many divisors can there be of a Boolean fn?

CS 270 6 Nov 1990 Tuesday

Applications of Duality Theory to Weighted Matching
Papadimitriou & Steiglitz (Ch 11)

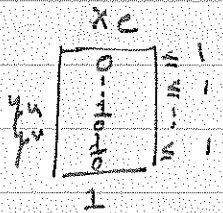
Construct a maximum matching.

$$x_e = \begin{cases} 1 & \text{if } e \in M \\ 0 & \text{if } e \notin M \end{cases}$$

Primal $\max \sum x_e$ fractional solutions?

$x_e \geq 0$

$\sum_{e \text{ inc. w/ } u} x_e \leq 1$



Dual $y_u \geq 0$

dual constraints: for each edge (u,v)

$y_u + y_v \geq 1$

$\min \sum_u y_u$

$\Longleftrightarrow$

0-1 solutions of primal correspond to matchings.

looking at dual: (1) no motivation for $y_u > 1$ ever.

0-1 solns of dual $\Leftrightarrow$ vertex covers!

If we don't have to worry about fractional parts we get K.-E.-thm for bipartite graphs.

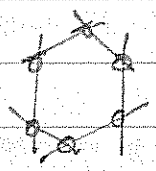
HW1. In Birkhoff we proved that primal and dual have optimal solns in which every component is zero or one.

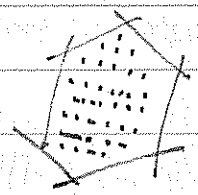
size of max matching = opt value for Primal = opt value for dual = size of min vertex cover

L.P.
max $c \cdot x$
$\exists\ Ax \le b$

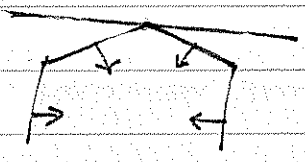
I.P
max $c \cdot x$
$Ax \le b$; x integer

} much more difficult; generally hard.



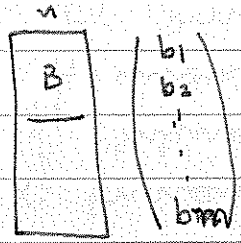


Once in a while, extreme points of a polyhedron are integral.



Fact: extreme point determined by intersection of n Hyperplanes whose coeff are linearly indep.

vertex determined by solving system of linear equations



$Bx = \begin{pmatrix} b_1 \\ b_2 \\ \vdots \\ b_n \end{pmatrix}$

$x = B^{-1} \begin{pmatrix} b_1 \\ \vdots \\ b_n \end{pmatrix}$

To prove that:
x is integer

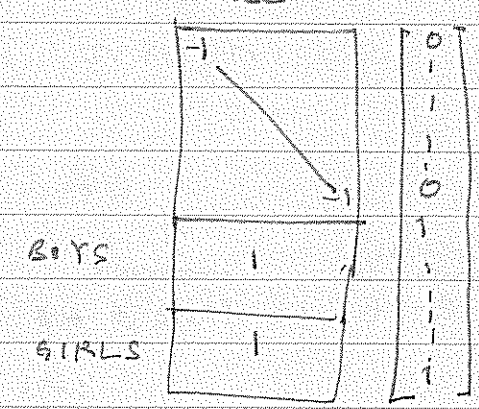
P2
CS270
Nov 5, 1991

Bipartite Matching

$x_e \geq 0 \Leftrightarrow -x_e \leq 0$

$\sum x_e \leq 1$

one inc. with



matrix

∴ if we can show B^{-1} is an integer then done.

integer

Cramers Rule:

$$(B)^{-1}_{ij} = (-1)^{i+j} \frac{\det B_{ji}}{\det B}$$

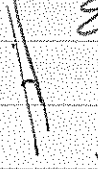
integer

∴ sufficient to show $\det(B) = \pm 1$ (easy)

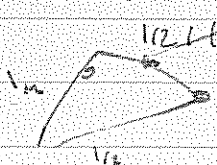
Every Square SubMatrix of const Matrix has det 1, or 0, or -1.

↳ total unimodular

Lets try and form non bipartite matching as an LP!



← w/ integers



← w/ fractions

∴ we must do something.

⇒ dream up additional inequalities

Try to connect to blossoms, odd sets, etc. . .

For every odd set S of vertices,

$$\sum_{e \in S} x_e \le (|S|-1)/2$$ ← Exp. # of constraints!!

can't do w/ cycles ∴ must use odd sets.

max $\sum x_e$; $x_e \ge 0$

$$\sum_{e \text{ at } u} x_e \le 1$$

~~$\sum$~~ $$\sum_{e \text{ in } S} x_e \le \frac{|S|-1}{2}$$

Hope: integer solutions at the ~~...~~ each extreme pt.

Dual (♡ more about now)

$\min \left[\sum_u y_u + \sum_{\text{all } S} Z(S) \frac{|S|-1}{2} \right]$

subject to

for each edge

$y_u + y_v + \sum_{S \text{ contains } e} Z(S) \ge 1$

~~$\forall S \quad Z(S) \ge 0 \quad (?)$~~

the other constraint is same (for edge each edge e, $y_u + y_v + \sum \ge ($...$)$

$y_u \ge 0$, $Z(S) \ge 0$

(?)

$y_u \ge 0$ $\quad Z(S) \ge 0$

For each edge $e = (v, w)$

$$y_w + y_v + \sum_{S \text{ contains } e} Z(S) \ge 1$$

$y_u \ge 0$ $\quad Z(S) \ge 0$

Reasoning as before, all variables ≤ 1.

0-1 solns of primals ⟺ matching

0-1 soln of dual ⟺ odd set cover.

p3 CS270
~~[illegible]~~
6 Nov 90

∴ exactly right obj fn, same

~~opt value of primal =~~

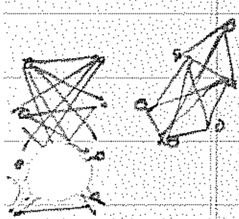
Thm: The primal and dual have optimal 0-1 ~~[illegible]~~ solution.

Pf: Duality + oddset cover thm.

WEIGHTED MATCHING

Each edge has a weight w_e

Goal: Perfect matching of max wt.



Bipartite Case: (this is the assignment problem!)

L.P. max $\sum w_e x_e$; $x_e \geq 0$

$\sum_{e \ni u} x_e = 1$ (doesn't matter because if $\sum_{e \ni u} x_e \leq 1$ ⇒ optimal will always be equality! (no unmatched pair will [illegible]))

dual → y_u not sign constrained ;

for each edge (u,v)

$y_u + y_v \geq w_e$;

min $\sum y_u$

primal & dual have optimal solutions in integers.
(from previous arguments based on determinants.

$\bar{c}_e = y_u + y_v - w_e$

"modified cost" of edge e.

primal feasibility : perfect matching

dual feasibility : non negative modified cost

C.S.C. : means $x_e \neq 0 \Rightarrow e \in M \Rightarrow \bar{c}_e = 0$

This is exactly what the Assign algo was doing!

NON BIPARTITE WEIGHTED MATCHING -

what constraints do we add to primal to get rid of fractional problems

↳ need odd set constraints!

primal

$$\max \sum_e w_e x_e$$

$$x_e \geq 0$$

$$\sum_{e \text{ at } u} x_e = 1$$

$$\sum_{e \text{ in } S} x_e \leq \frac{|S|-1}{2}$$ $Z(S)$ does have integer B.F.S.

dual

$$Z(S) \geq 0$$

$$\min \sum_u y_u + \sum Z(S) \frac{|S|-1}{2}$$

for every edge $e = (u,v)$

$$y_u + y_v + \sum_{S \ni e} Z(S) \geq w_e$$

$$\bar{c}_e = y_u + y_v + \sum_{S \ni e} Z(S) - w_e$$

/* Will get two kinds
* of CSC. Need one
* is if $Z(S) > 0 \Rightarrow$
* S contains $\frac{|S|-1}{2}$ edge
* of M
*/

Weighted Matching

Simplex Method for L.P.

w_e weight of edge e

Perfect matching of max wt

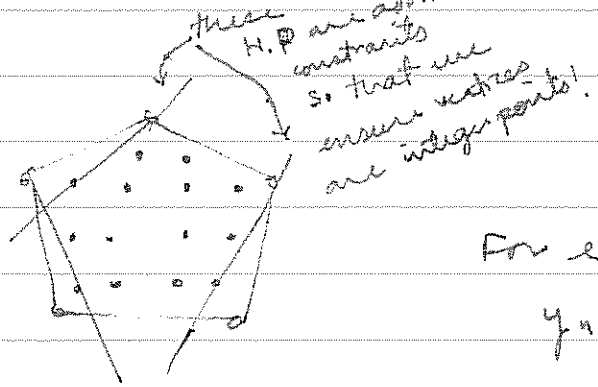
$$x_e = \begin{cases} 1 & \text{if } e \in M \\ 0 & \text{if } e \notin M \end{cases}$$

Primal

$$\max \sum w_e x_e \quad \ni x_e \geq 0$$

$$\sum_{e \text{ at } v} x_e = 1$$

$$\sum_{e \in S} x_e \leq \frac{|S|-1}{2} \quad \therefore \text{to ensure its integral!}$$



Dual

$$\min \sum y_u + \sum z(S)\left(\frac{|S|-1}{2}\right)$$

For edge (u,v)

$$y_u + y_v + \sum_{S \ni e} z(S) \geq w_e$$

$$z(S) \geq 0$$

$$\bar{c}_e = y_u + y_v + \sum_{S \supset \{u,v\}} z(S) - w_e \geq 0$$

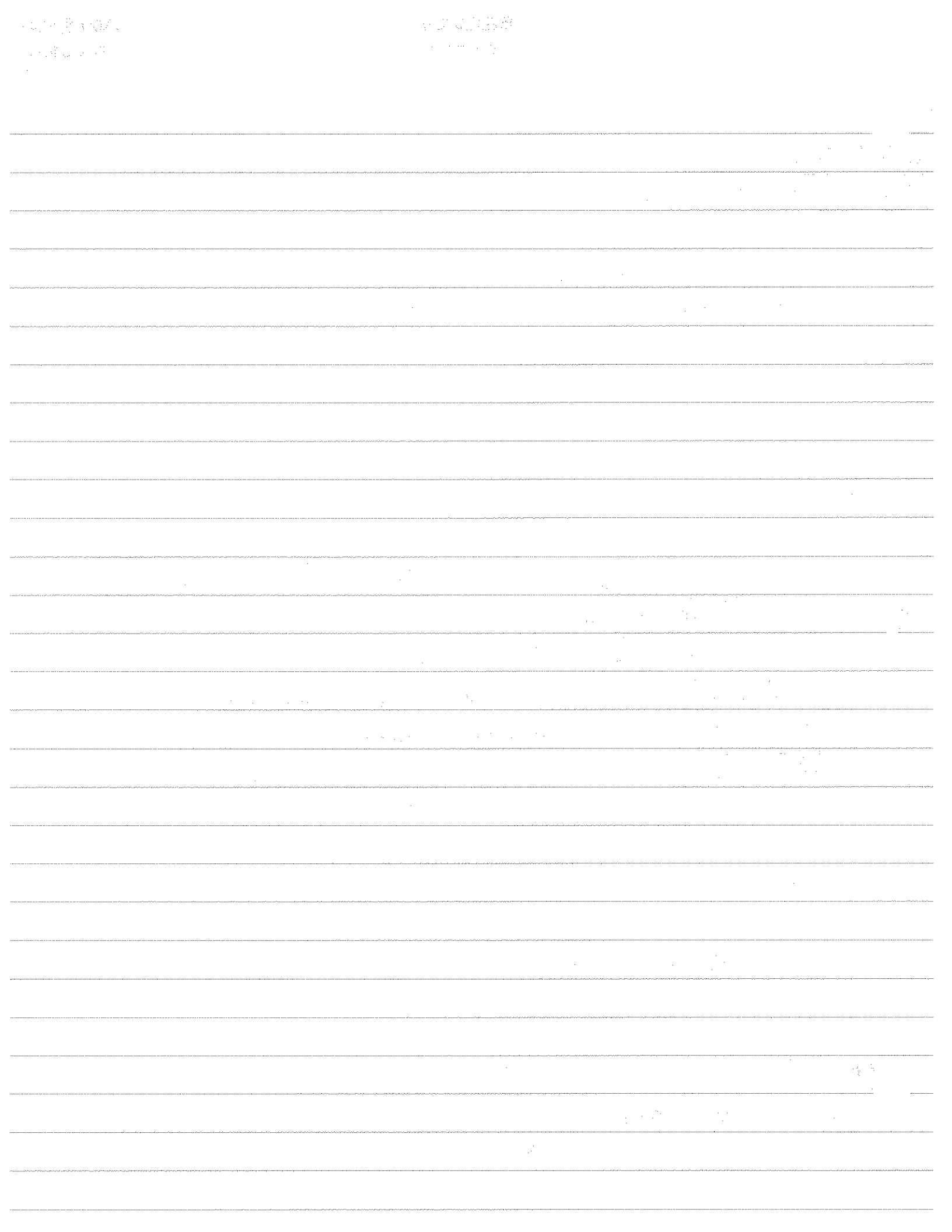
mod cost of edge e.

Dual feasibility $\bar{c}_e \geq 0$

Primal perfect matching.

Dual $y_u, z(S)$ $\bar{c}_e \geq 0$

$z(S) \geq 0$



PB
CS270

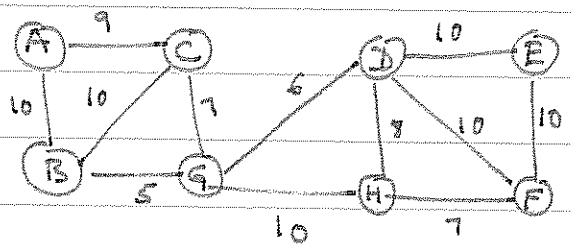
C.S.C. $e \in M \Rightarrow \bar{c}_e = 0$

$z(S) > 0 \Rightarrow S$ is saturated

Primal Dual Approach to solve

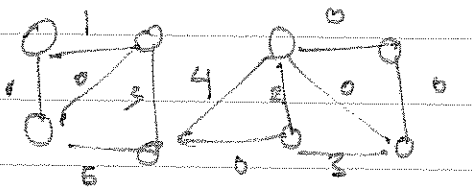
↳ to relax perfectness requirement; then iterate later
↳ while retaining constraints.

Examine example:

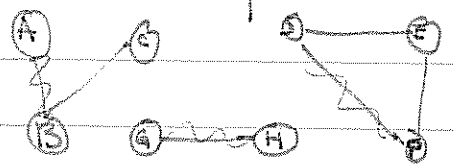


$y_u + y_v + \sum_{S \supseteq e} z(S) - w_e \Rightarrow$ get graph as above but all neg edges

y, z are at our disposal
$\Rightarrow$ add s^{th} to get all nonnegative



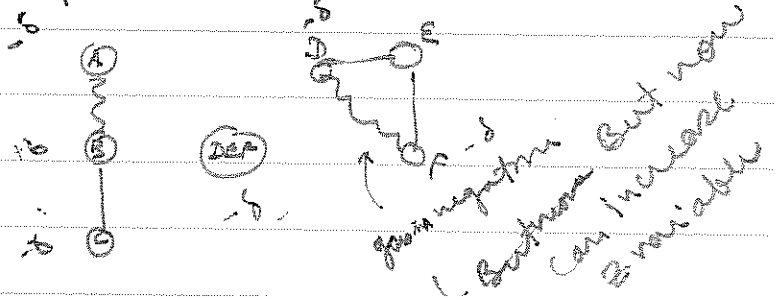
Form restricted primal

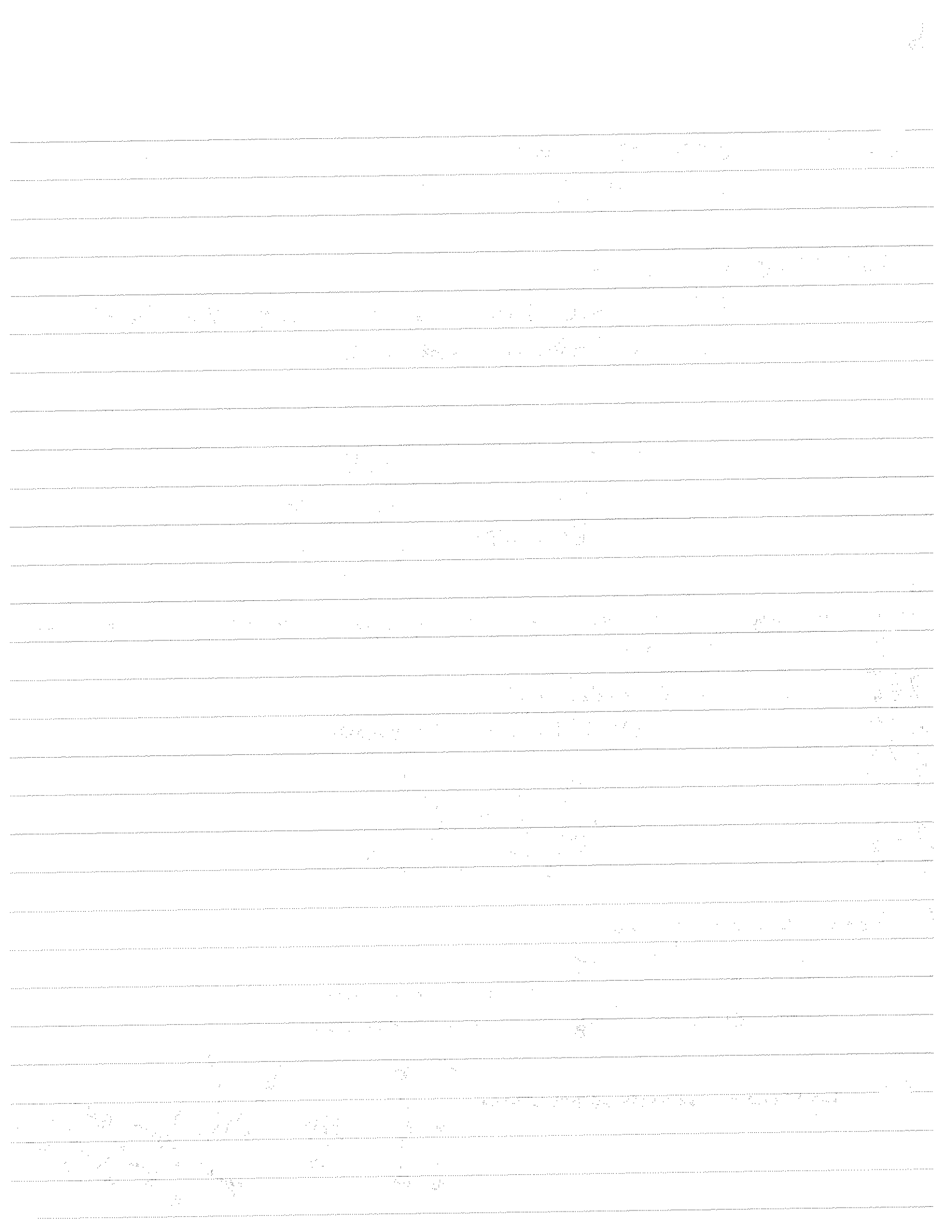


$\Rightarrow$ get max size matching

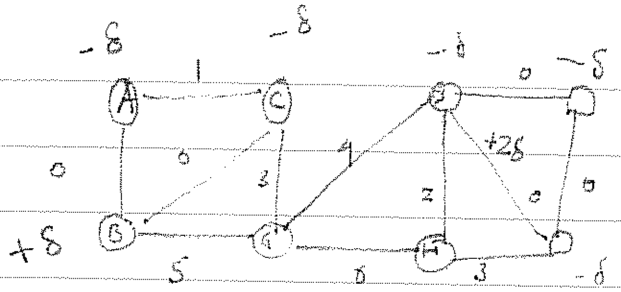
(eg Hungarian)

↳ view as search structure



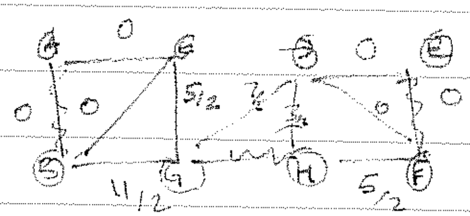


p3
CS270
Nov 8, 1990



- Reduce y_u by δ for each u which is an even vertex contained in an even macrovertex.
- Incr. y_u by δ for each u which is odd vtx or in odd macrovtx.
- Increment $z(S)$ by 2δ if S is even macrovtx
- Reduce $z(S)$ by 2δ if S is odd macrovertex

∴ Choose δ_{max} by keeping $\bar{c}_e \geq 0$
$z(S) \geq 0$



/* vtx. not can
have frac. wt. ??
*/

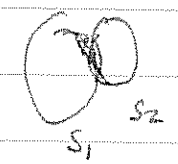
$z(\{D, E, F\}) = 1$

S; $z(S) > 0 \Rightarrow$ S is not

/* shrink each odd set S w/ $[z(S)] > 0$ to a macrovertex
*/

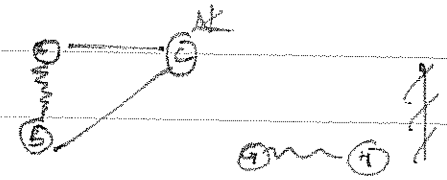
have to ensure:

Imp Point.



oddsets w/ $z > 0$ are disjoint or nested.

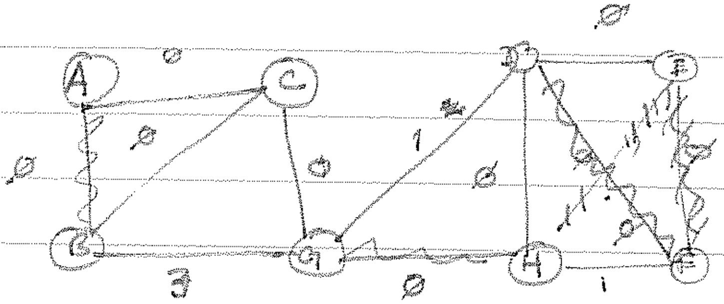
p4
CS270
Nov 8 1990



restricted primal

* ⇒ ~~Kekkule~~ free vertices

After some further iterations, get

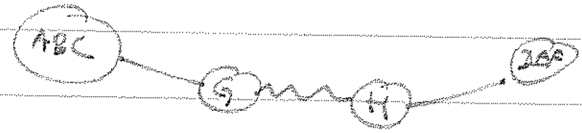


$z(\{D,E,F\}) = 4$

$z(\{A,B,C\}) = 3$

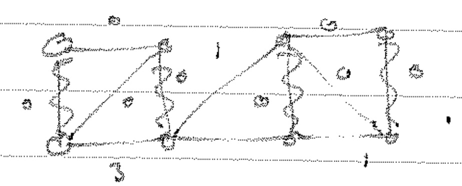
~~Restricted~~ ~~Kekkaluf~~

Restricted primal :



~~Aug Path in Graph~~ after shrinking
∴ unshrink & get aug. path

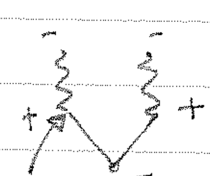
∴ end up



Time Bound (Please tell Adnan about this)

Each iteration (Search in restricted primal end in

- augmentation
- bring new vertex into search structure
- blossom forms
- unshrink macrovertex at odd level
  ↳ can't have a loop as unshrink only when z value drops to zero which can take place only in odd stages (even it increases) then always get to an



merits by edge between two even vertices and then having blossom to ...

more vertex

Linear Programming Algorithms

. Simplex Method

. Interior Point Method

Min cost Circulation

$\min c \cdot x$

$Ax = b$

$x \geq 0$

$(\quad x \quad)$

$m \quad A \quad n \quad \quad (b)$

$(\quad c \quad)$

Assume A is of full rank (rank(A) = m)

Basic Solutions $\Leftrightarrow$ Extreme Points

Determined by m l.i. cols of A

$(x_B)(\quad x_N \quad)$

Partition A $\rightarrow$ $[\, B \mid N \,] = (b)$ — (x_B) basic variables, (x_N) nonbasic variables

↑ non singular

$(c_B)(\quad c_N \quad)$

Basic Solution: set x_N to zero

$\min c_B^T x_B + c_N^T x_N = z$

$B x_B + N x_N = b$

$x_B \geq 0$

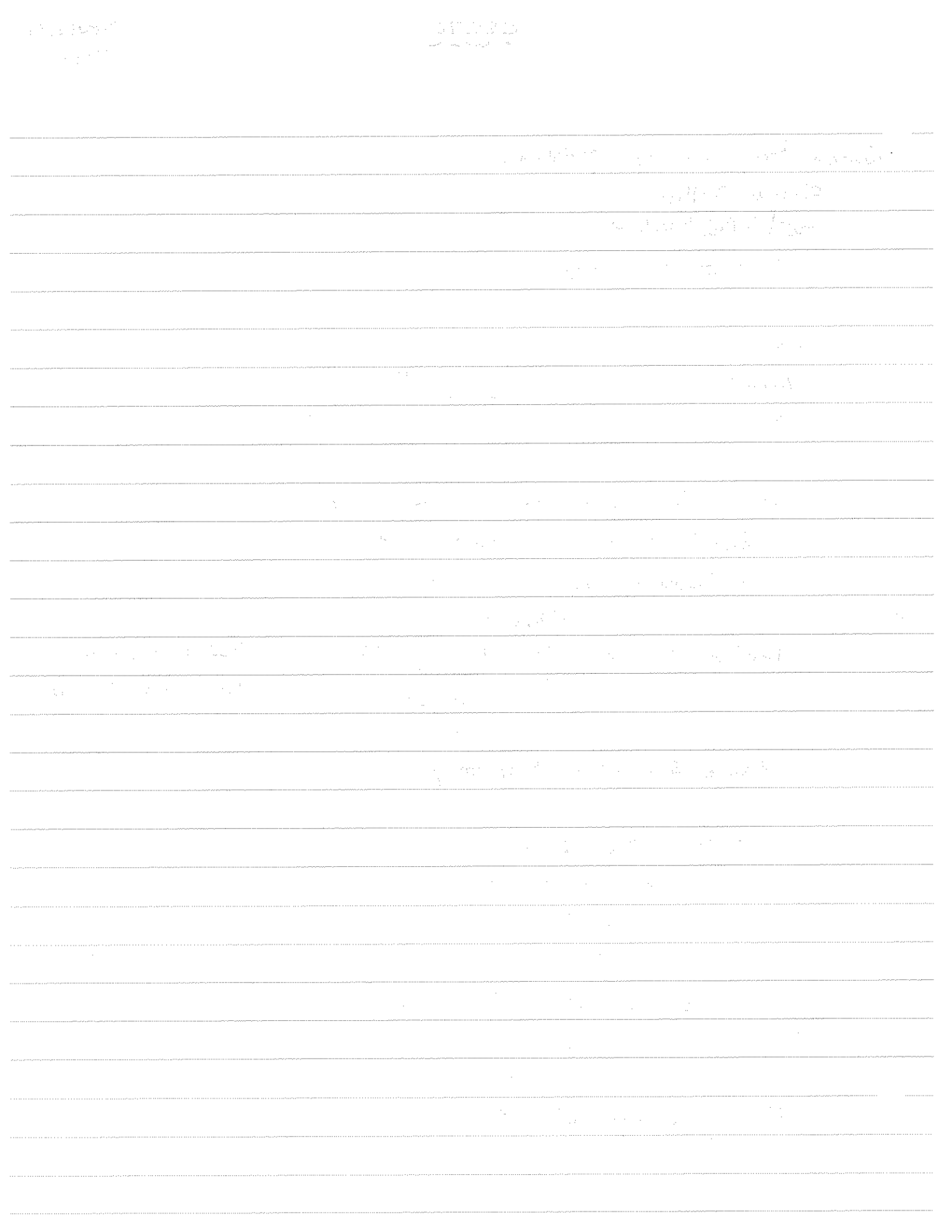
$x_N \geq 0$

$x_B + B^{-1} N x_N = B^{-1} b$; can now read off basic soln.

Basic solution is $x_B = B^{-1} b$

$x_N = 0$

Feasible if $B^{-1} b \geq 0$



$$x_B = B^{-1}b - B^{-1}N x_N$$

$$\begin{aligned} z &= c_B^T x_B + c_N^T x_N \\ &= c_B^T (B^{-1}b - B^{-1}N x_N) + c_N^T x_N \\ &= c_B^T (B^{-1}b) + (c_N^T - c_B^T B^{-1}N) x_N \end{aligned}$$

f $\qquad$ g

$$\begin{bmatrix} x_B & x_N \\ I & E \end{bmatrix} = (d) \quad ; \; d = B^{-1}b$$

$$z = f + g \cdot x_N$$

when is soln optimal?

Suppose $\forall t \; g_t \geq 0$

$\Rightarrow$ any tampering $\bar{\text{o}}$ non basic variables increases obj fn

The present B.F.S. is optimal

Suppose $g_t < 0$; wish to decrease obj fn by increasing $(x_B)_t$ while keeping the other non basic variables zero.

$$(x_B)_i + e_{it}(x_N)_t = d_i$$

$$(x_B)_i = d_i - e_{it}(x_N)_t \geq 0 \qquad \forall i, e_{it} \leq 0$$

suppose $e_{it} \leq 0 \Rightarrow$ no restriction $\Rightarrow$ unbd) value

$e_{it} > 0 \Rightarrow (x_N)_t \leq (d_i / e_{it})$

$$\text{set } (x_N)_t = \min \{ d_i/e_{it} \mid e_{it} > 0 \}$$

which basic variable becomes zero?

the ith one; i corresponds to this

$(z_N)_t$ becomes basic

$(z_N)_i$ becomes zero (nonbasic)

Objective fn $\leftarrow f + g_t \left(d_i / e_{it}\right)$

$d_i = 0 \quad e_{it} > 0 \quad \Rightarrow$ (eqn) degeneracy

Small example:

$$\min \; -x_1 - x_2 = z$$
$$3x_1 + 5x_2 \leq 8$$
$$2x_1 - x_2 \leq 1$$
$$x_1, x_2 \geq 0$$

obs. 1. not in standard form (ineq not strictly equal)
so introduce slack variables x_3, x_4

$$\min \; -x_1 - x_2 \quad (= z)$$
$$3x_1 + 5x_2 + x_3 = 8$$
$$2x_1 - x_2 + x_4 = 1$$
$$x_1, x_2, x_3, x_4 \geq 0$$

Can start easily (∵ x_3, x_4 can be basic as 8, 1 automatically) so start

Simplex Tableau — Nonbasic / Basic variables

x_1	x_2	x_3	x_4	$-z$	
3	5	1	0	0	$= 8$
2	-1	0	1	0	$= 1$
-1	-1	0	0	1	$= 0$

x_1	x_2	x_3	x_4	$-z$	
3	5	1	0	0	= 8
2	-1	0	1	0	= 1
-1	-1	0	0	1	= 0

which one do we increase? x_2 made basic.

$3x_1 + x_3 = 8$ $\quad x_3 = 8 - 3x_1$

$2x_1 + x_4 = 1$ $\quad \boxed{x_4 = 1 - 2x_1}$ limiting factor

NO

bring x_1 into the basis, take x_4 out of the basis

x_1 basic

x_4 non basic

YES

what if made x_2 made basic

$x_3 = 8 - 5x_2$

$x_4 = 1 + x_2$

~~x_3 becomes non basic~~

x_2 becomes basic

x_1	x_2	x_3	x_4	$-z$	
3/5	1	1/5	0	0	= 8/5
13/5	0	1/5	1	0	= 13/5
-2/5	0	1/5	0	1	= 8/5

bring x_1 into the basis

STOP → . unbdd soln (bring variable into basis and it can grow out)

. no more -ves in last in line

convergence:

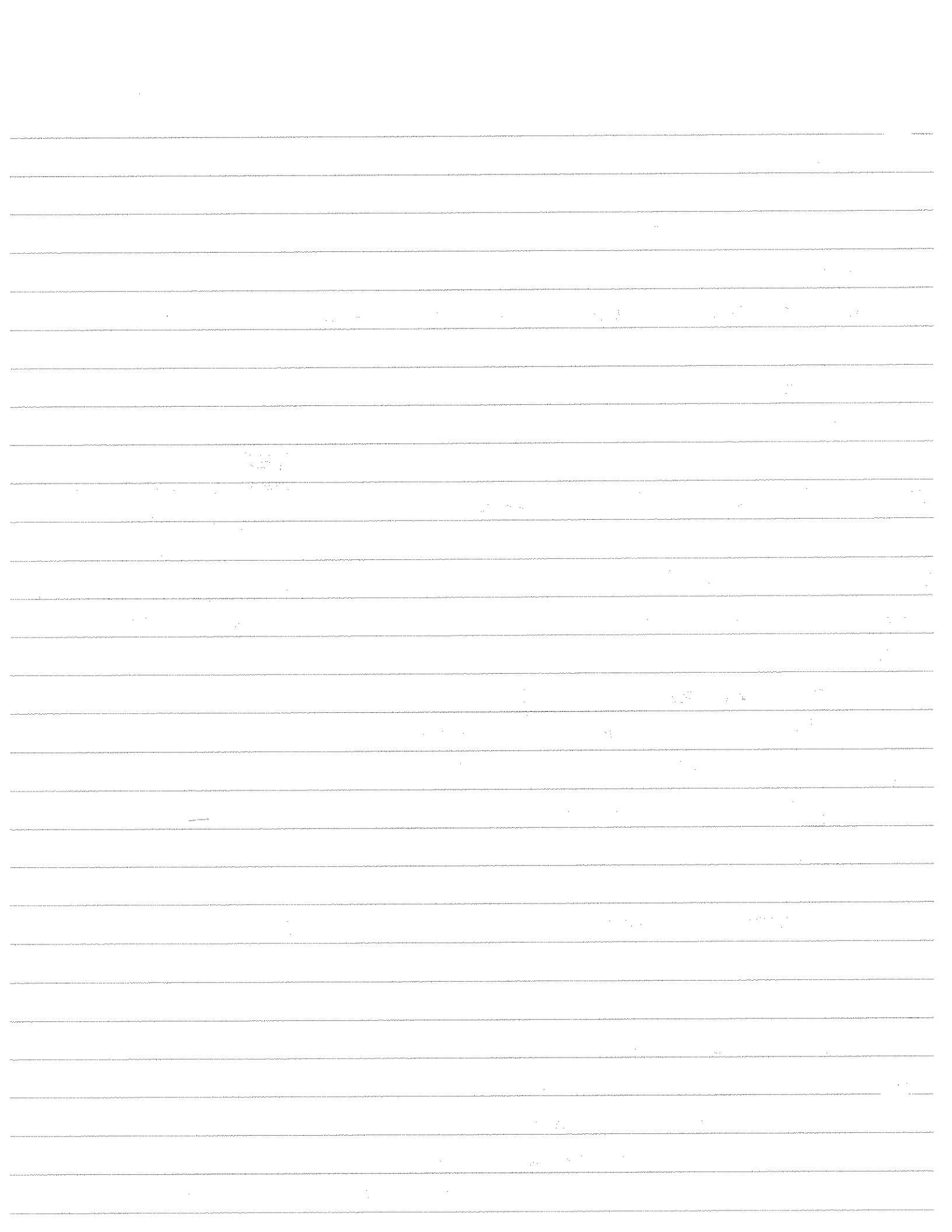
# of basic solutions is finite

∴ if wont repeat BFS we will converge.

Can we prove will never repeat?

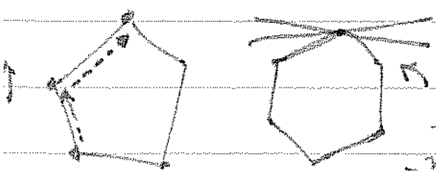
z decreases always (except degeneracy)

so in absence of degeneracy, we are done (z dec each step)



Presence of degeneracy: Method might cycle.

Degeneracy corresponds to having many DLPs

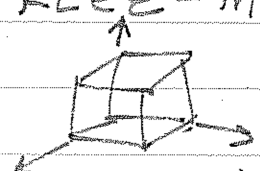


can describe this vertex in many ways!
which edge will get you out of that nest of descriptions?

Why need we not worry?
- (Engineer) Never in practice
- Perturbation techniques eliminate it

Degeneracy occurs in practice
but cycling never occurs in practice

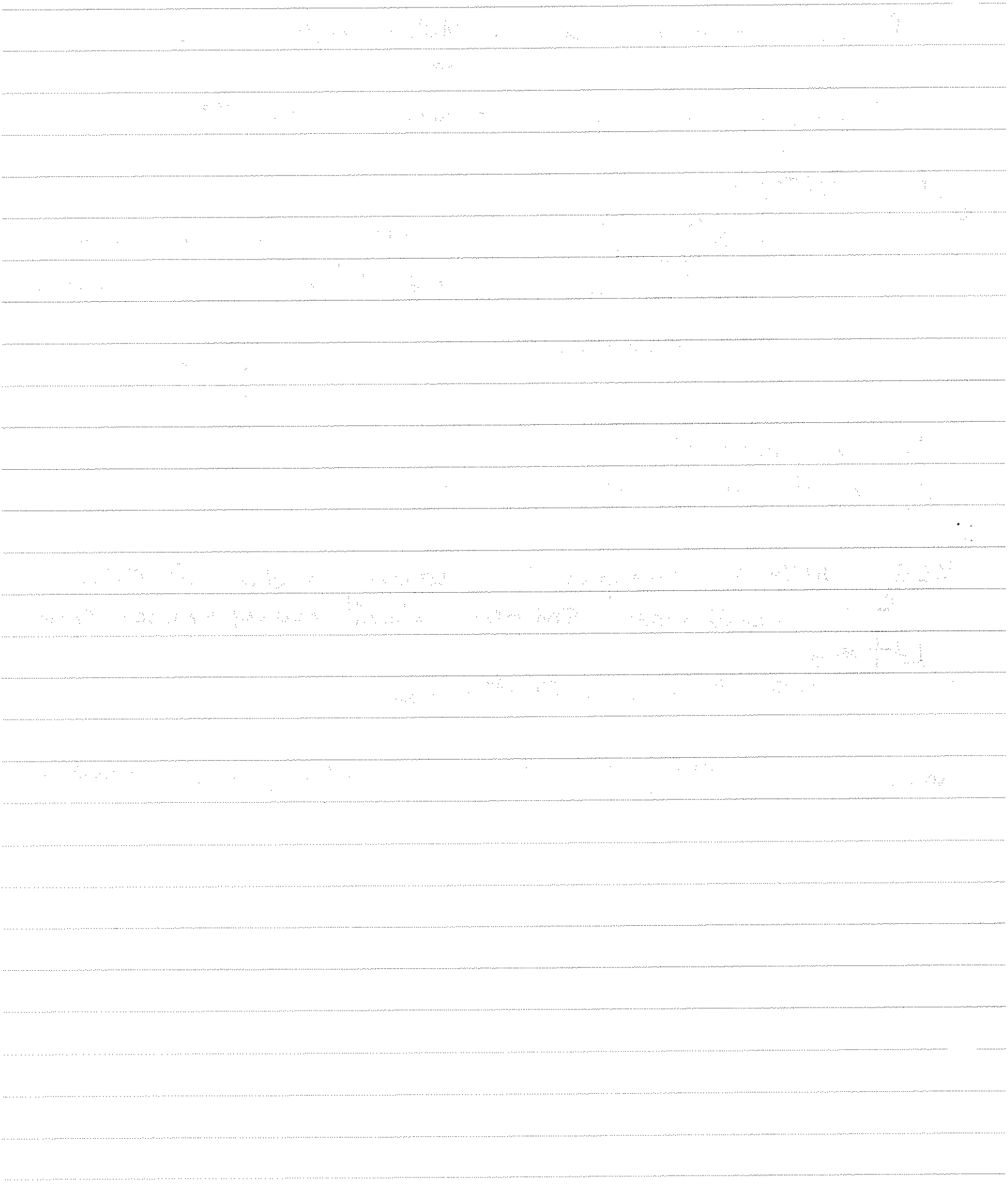
KLEE-MINTY Examples: Slightly perturbed H.C.



could make Simplex visit every vertex of H.C.

→ Practice 3m iterations suffice.

various rules: • most neg coeff | all fail w/ Klee-Minty Example
• comp for each obj fn



min Cost Circulation

8:30 60 Exams

SIMPLEX: How to get initial BFS? (Know how to iterate after this)

"Artificial Variables"

$\min c \cdot x$

subject to $Ax = b$, $x \geq 0$ w.l.o.g. assume $b \geq 0$ (else change the signs)

$\min \sum y_i$ /* y_i are artificial variables */

$AX + Iy = b$

$x \geq 0 \; ; \; y \geq 0$ ($\sum a_{ij} x_j + y_i = b_i$)

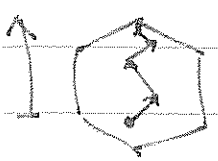
can extract initial BFS easy: just take $y_i = b_i$

Solve this for the optimal BFS. ($y_i = 0 \; \forall i$)

Now go to original; this gets us a starting BFS

1984: Karmarkar

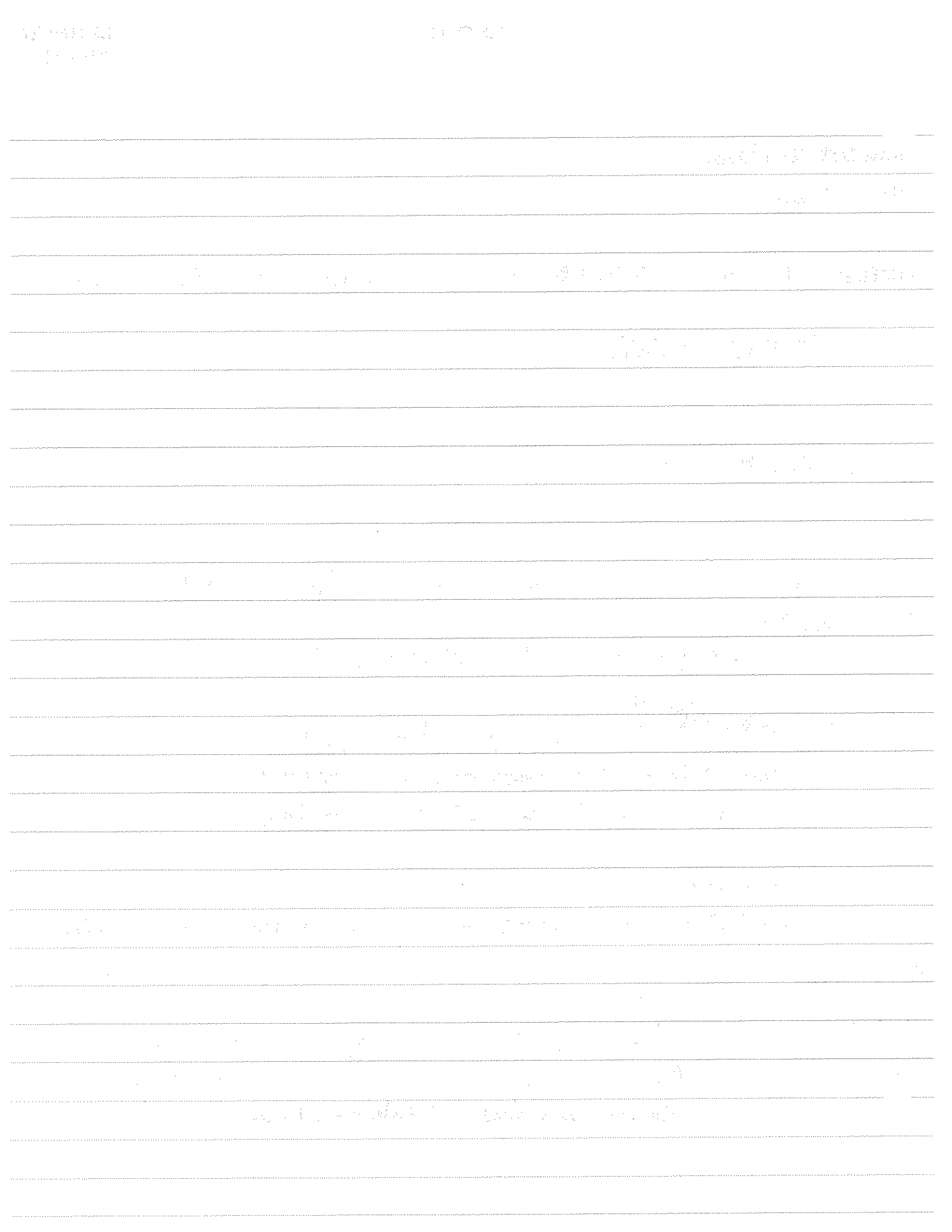
interior point methods shoot through the middle rather than go around

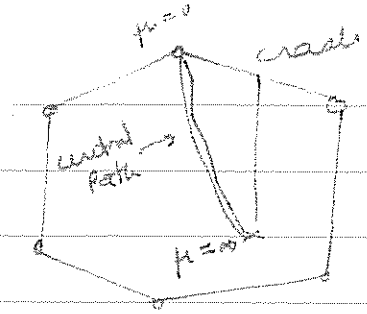


1. Poly time
2. Very small # of iterations (empirical observation)

But each iter. took a long time for most people

Karmarkar uses good data structures





find direction of steepest descent increase
. might crash into boundary
. curved approach into optimum

(cx) "Centrality" of x : how far from boundaries x is.

$H_1, H_2, \ldots, H_k$

$\sum_H \log_\varepsilon^{(\text{dist}}(x, H))$ on boundary $\to (-\infty)$ centrality

$\max c \cdot x + \mu C(x) \quad (\mu > 0)$ /* family of problems */
$Ax \le b$

$\mu > 0 \Rightarrow$ opt. at centre
trace optimum soln as fn of μ
get central path.

MINIMUM COST CIRCULATION

Given a digraph

circulation $f : E \to R$

$f(u,v)$

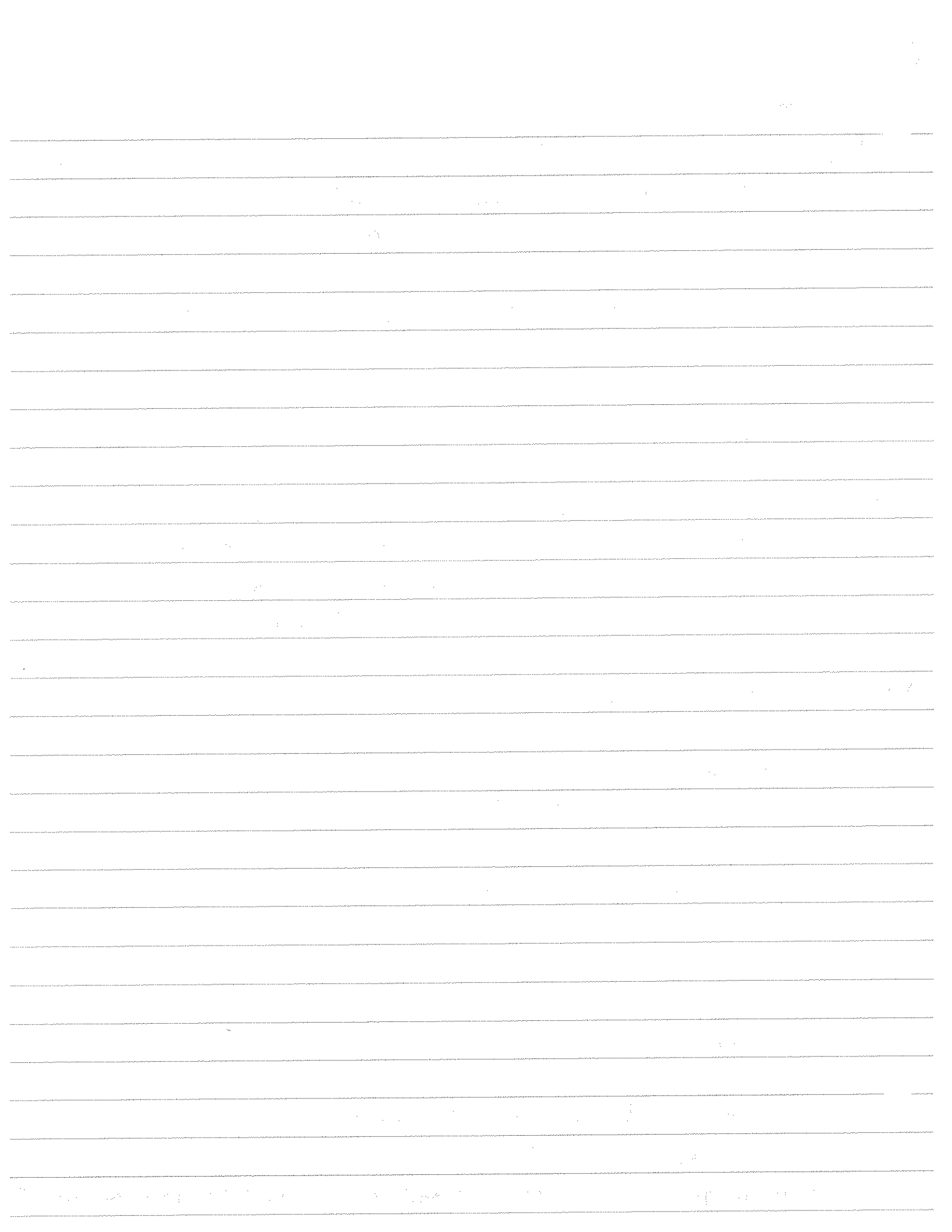
foreach u $\sum_w f(w,u) = \sum f(u,v)$

$l(u,v)$ lower bound
$c(u,v)$ upper bound
$d(u,v)$ cost

feasible $\forall (u,v) \quad l(u,v) \le f(u,v) \le c(u,v)$

cost of circulation $= \sum d(u,v) f(u,v)$

/* very general : covers most of all we've done so far */



Shortest Path | HW
Max Flow
Assignment .

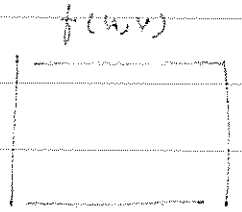
No problem of integrality as we allow fractional flows.

$$\min \sum_{u,v} d(u,v) f(u,v)$$

$$\sum_w f(w,u) = \sum_v f(u,v) \iff \sum_w f(w,u) - \sum_v f(u,v) = 0 \qquad \pi(u) \text{ not sign constr}$$

$$f(u,v) \geq d(u,v) \Rightarrow \alpha(u,v) \geq 0$$
$$-f(u,v) \geq -c(u,v) \Rightarrow \beta(u,v) \geq 0$$

Dual: $\max \sum \alpha(u,v) d(u,v) - \sum \beta(u,v) c(u,v)$



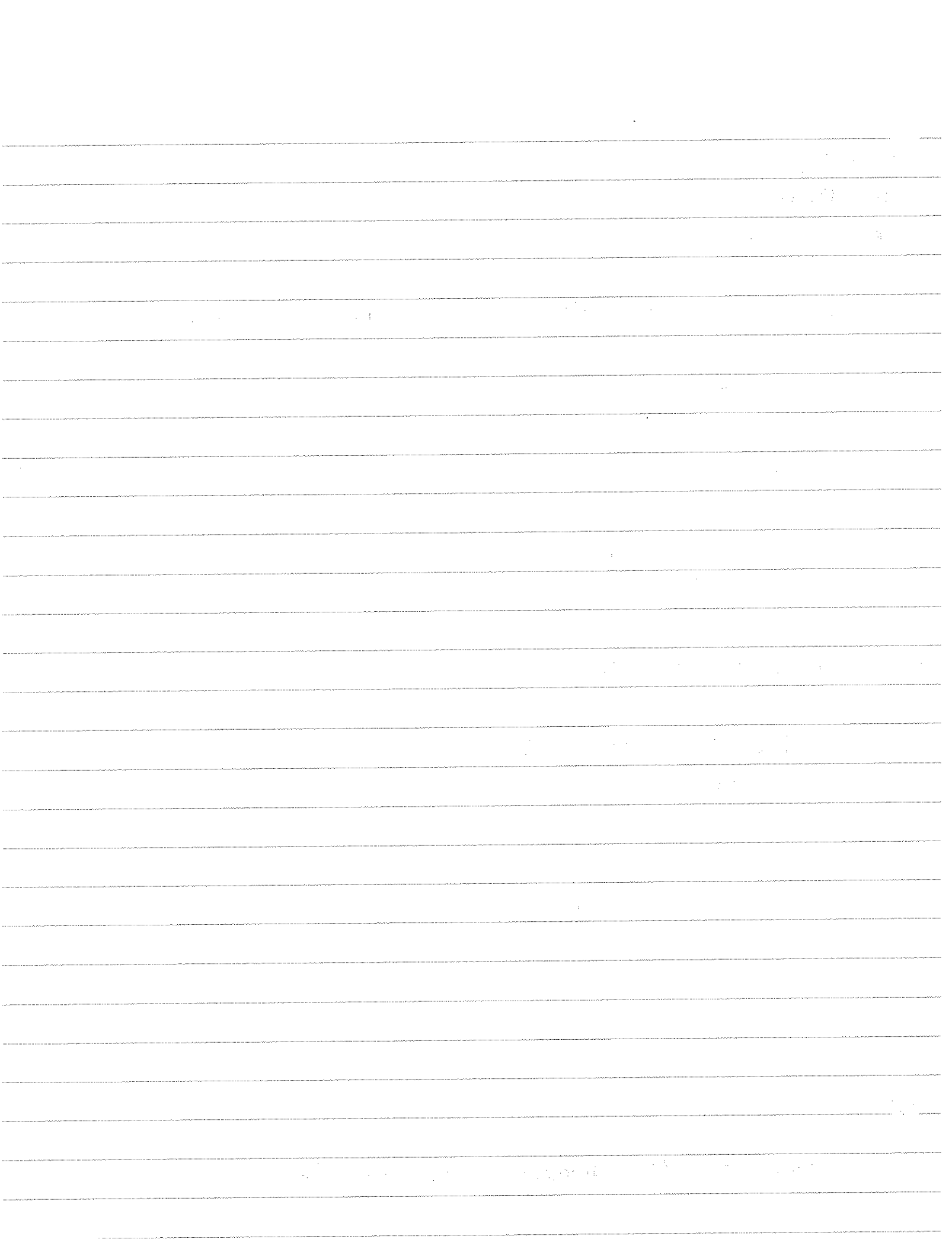
$$-\pi(u) + \pi(v) + \alpha(u,v) - \beta(u,v) = d(u,v)$$
$$\alpha(u,v) \geq 0$$
$$\beta(u,v) \geq 0$$

$\bar{d}(u,v) = \pi(u) + d(u,v) - \pi(v)$ /* modified cost of edge (u,v) */

$\alpha(u,v) - \beta(u,v) = \bar{d}(u,v)$ (* dont need to worry about f *)

Having chosen π-variables determining $\bar{d}(u,v)$ how shall we choose $\alpha(u,v)$ and $\beta(u,v)$?

Fixing π variables decouples the equation.



Becomes ~~an array observation~~ easy optimization problem

maximize $\alpha(u,v) d(u,v) - \beta(u,v) c(u,v) = \alpha d - \beta c$

such that $\alpha - \beta = \bar{d}$

$\alpha \geq 0$

$\beta \geq 0$

two variable L.P. problem!

optimal solu will not have both $\alpha, \beta > 0$.
(else could reduce by same small amt $\Rightarrow$ increases obj. fn !)

If $\bar{d}(u,v) \geq 0$ then $\alpha(u,v) = \bar{d}(u,v)$

$\beta(u,v) = 0$

If $\bar{d}(u,v) \leq 0$ then $\alpha(u,v) = 0$

$\beta(u,v) = -\bar{d}(u,v)$

Examine CSC: (only dual has sign constrained variables)

$\alpha d(u,v) > 0 \Rightarrow f(u,v) = d(u,v) \iff \bar{d}(u,v) > 0$

$\beta(u,v) > 0 \Rightarrow f(u,v) = c(u,v) \iff \bar{d}(u,v) < 0$

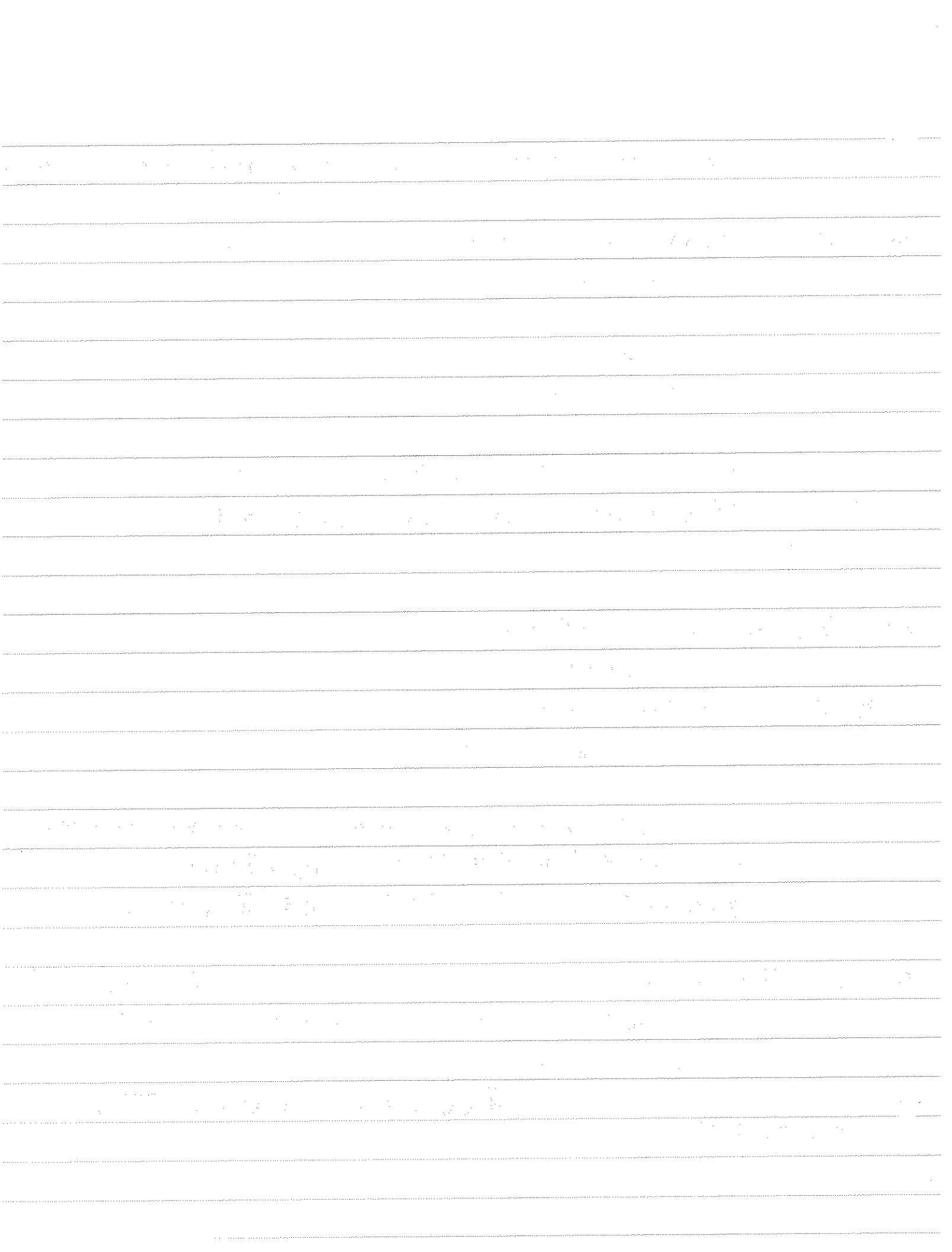
Feasible Circulation: Need dual feasibility

CSC $\bar{d}(u,v) > 0 \Rightarrow f(u,v) = 0$ = $l(u,v)$ is various!

$\bar{d}(u,v) < 0 \Rightarrow f(u,v) = c(u,v)$ */

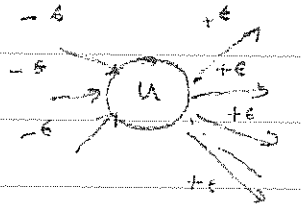
$\bar{d}(u,v) = \pi(u) + d(u,v) - \pi(v)$

Choose f, π



$\min \sum d(u,v) f(u,v)$

$\min \sum \bar{d}(u,v) f(u,v)$



$\Pi(u) \uparrow \epsilon$

look at intuition : effect of π variables cancels out !

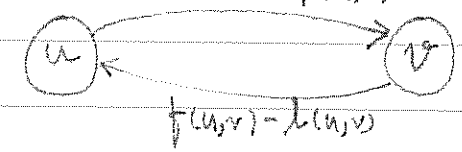
Also csc arise from simple reasoning (obj fn!

(necessity ns sufficiency ?)

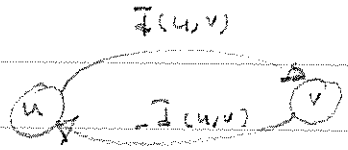
one more way of looking at this

RESIDUAL N/W :

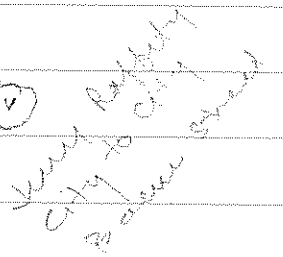
Rel to f $\quad l(u,v) \le f(u,v) \le c(u,v)$



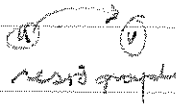
Residual capacities



costs

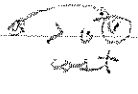


csc : $\bar{d}(u,v) > 0 \Rightarrow f(u,v) = l(u,v)$ so only edge



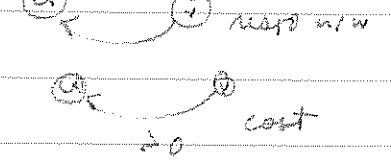
resid graph

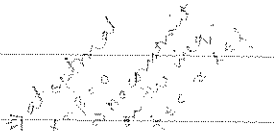
$\bar{d}(u,v) < 0 \Rightarrow f(u,v) = c(u,v)$



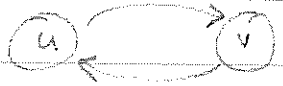
cost

$= 0$





$\bar{d}_{uv} = 0 \Rightarrow$



0

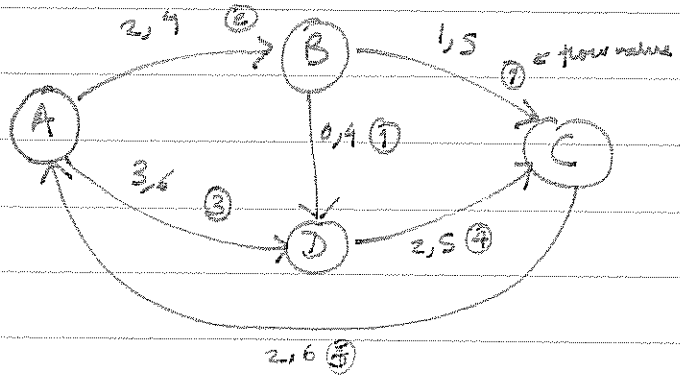
flow can be strictly between upper & lower bounds

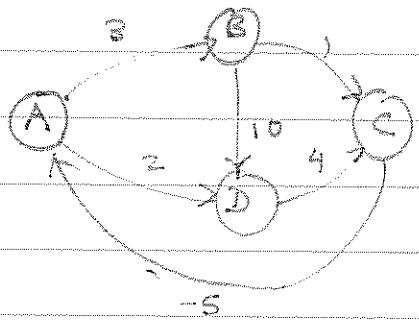
$\therefore$ Equiv formulation of csc is that each edge of the residual n/w has non negative (modified) cost

∴ another formulation is

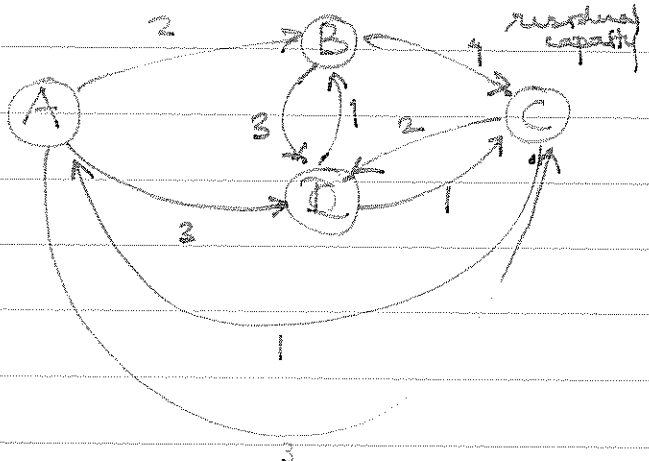
find feasible circulation of node numbers π determining $\bar{d}$ s.t.

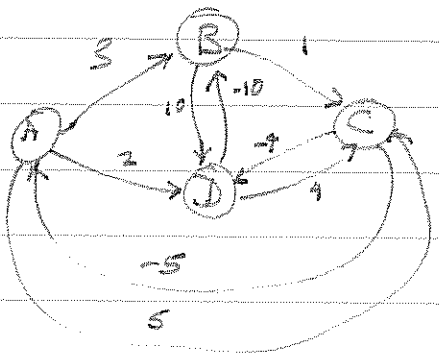
each edge of residual n/w has non neg (modified) cost.





are opt. cond. satisfied?





Satisfy optimality conditions? NO ∵ ∃ cycle of -ve weight. C which we could send flow around cycle!

Transshipment: S set of sources

$a(s)$ supply at s

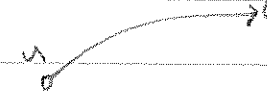
T set of destinations

$b(t)$ demand at t

capac $c(u,v)$ cost $d(u,v)$

require $\sum_{s \in S} a(s) \geq \sum_{t \in T} b(t)$

mincost circulation:



$l(u,v) = 0$ (no lower bnd)

$c(u,v)$, $d(u,v)$

amt to dest^n = amt from source

$\therefore$ create fictitious w for each s

$l(w,s) = 0$

$c(w,s) = a(s)$

$d(w,s) = 0$

and, for each $t \in T$

(t,w) $\quad l(t,w) = b(t)$

$c(t,w) = b(t)$

$d(t,w) = 0$

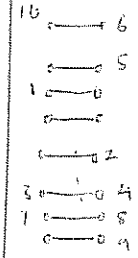
$\therefore$ Have formulated this as a minimum cost circulation problem.

Min Cost Circulation

MidTerm

Graph G w/ positive edge costs
Polytime algo for min cost edge cover
(HINT: use min cost perfect matching)

How does edge cover differ from perf matching?



edge cover — assume each edge in an optimal edge cover
(∵ no free edges)

observe: any edge which singly covers a vertex must be min cost among edges adjacent to vertex.
∴ introduce dummy vertices to match these.
left over dummy vertices → match to each other w/ zero wt edges.

$G = (V, E)$ edge e has cost $c(e) > 0$

$w(v) = \min\{ c(e) \mid e \text{ inc with } v \}$

G' = vertex set $V \cup V'$ each vertex $v \in V$ has mate $v' \in V'$

E' = edge set = E with original costs
for each vertex $v \in V$, edge $\{v, v'\}$ with wt $w(v)$ (min edge wt inc. to v)
for each pair $\{u', v'\} \subset V'$ edge $\{u', v'\}$ weight zero.

Min cost Perfect matching in G' yields min-cost edge cover in G

Matching
- $v - w$
- $v - v'$
- $u' - v'$

Edge Cover
- $v - w$
- v's cheapest edge
- (disregard)

Min Cost circulation Problem:

$$\min \sum_{(u,v)\in E} d(u,v) f(u,v)$$

$$l(u,v) \le f(u,v) \le c(u,v)$$

for each node u $\sum_{(u,v)\in E} = \sum_{(w,u)} f(w,u)$

Dual

Node Number Assignment

$\pi(u)$

modified edge cost of edge (u,v)

$$d_\pi(u,v) = \pi(u) + d(u,v) - \pi(v)$$

CSC

Feasible circulation f is optimal

$\Leftrightarrow \exists \pi$ s.t.

if $d\pi(u,v) < 0$ then $f(u,v) = c(u,v)$

if $d\pi(u,v) > 0$ then $f(u,v) = l(u,v)$

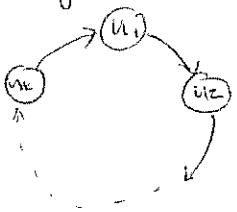
recall natural intuitive way of arriving at these

Exercise: For any circulation f,

$$\sum d(u,v) f(u,v) = \sum d_\pi(u,v) f(u,v) \quad \blacksquare$$

Exercise:

given any cycle



$$\sum_{(u,v)\in C} d(u,v) = \sum_{(u,v)\in C} d_\pi(u,v)$$

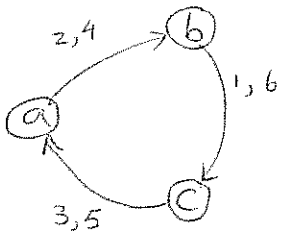
- Finding a feasible circulation (recall HW problem)

Pseudo Flow:

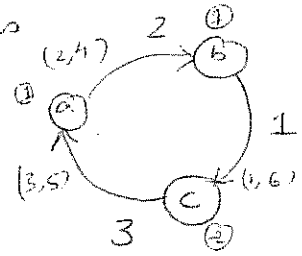
$$l(u,v) \le f(u,v) \le c(u,v)$$

start with pseudo flow (which doesn't necc. satisfy conservation), and convert to circulation

Example:



⇒ Pseudo Flow is



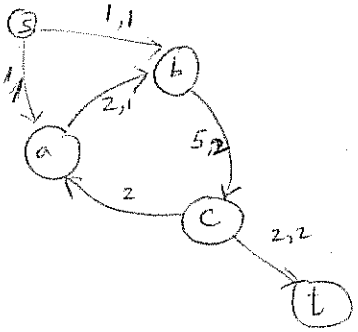
* not a flow
* ∵ no conns.
*/

(excess of u): $e(u) = \sum f(v,u) - \sum f(u,w)$

Sum of excesses is zero
So pump flow from nodes having positive excess to those having negative excess

S = Set of nodes having +ve excess
T = Set of nodes having -ve excess

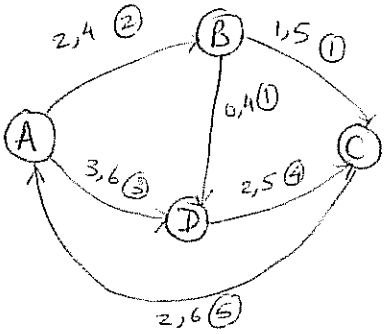
Goal: Send excess from S to T.

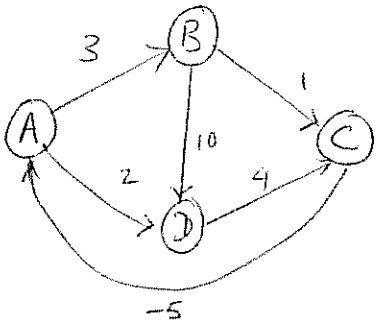


For each edge (u,v) of original n/w (u,v) cap (c(u,v) - f(u,v))
for each $u \in S$,
(s,u) CAP e(u)
for each $v \in T$,
(v,t) CAP ~~e(v)~~ - e(v)

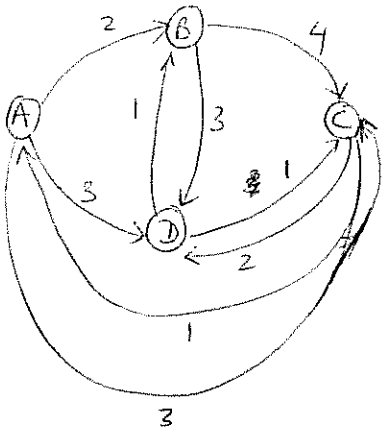
Thm (HW) ∃ a feasible solution ⇔ The new n/w has a max flow in which all edges at source and sink are saturated.

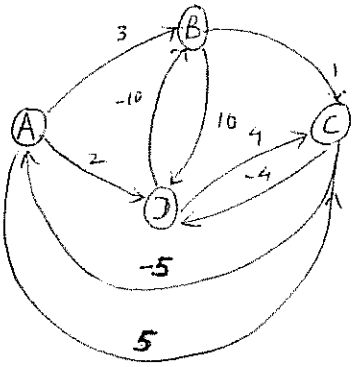
costs





↳ Residual Cap network





$R_{f,\pi}$ Residual NW for feasible circulation f w.r.t. modified costs d_π

---

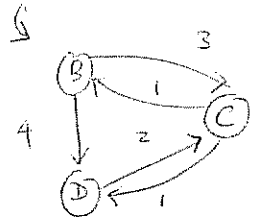
f is optimal $\Leftrightarrow \exists\ \pi$ s.t. every edge of residual network has non negative cost.

f as it stands has a negative cost cycle; so it is suboptimal (∵ a unit flow around cycle is feasible, and can decrease the ~~cost~~ cost by reducing flow (???) i.e. can augment around the negative cost cycle always)

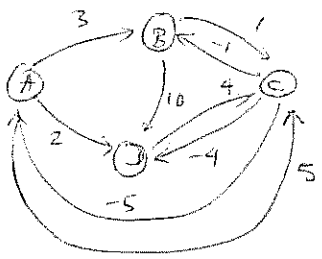
Negative cost cycle allows costs to be reduced.

S.R.P.
Sep 18, 1990
Tarjan's book

Send 1 around cycle

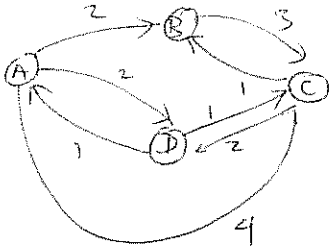


(no cap)

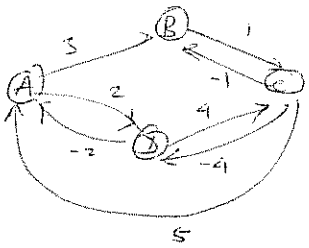


costs

iterate and get



no cap



cost $+ d_\pi$

If all-edge wts ≥ 0 $\Rightarrow$ optimal ($\because$ no neg cycles)

$O(nm)$

Given digraph w/ edge wts $w(u,v)$ either

(1) $\exists$ neg wt cycle

OR

(2) $\exists$ node-number asst p s.t.
for every edge (u,v) $p(u) + w(u,v) - p(v) \geq 0$

* we are
* proving
* that absence
* of neg cycles
* $\Rightarrow$ optimality

Claim: $\ell_{f,\pi+p}$ has all edge costs non negative
$\therefore f$ is optimal

Thm: If $\ell_{f,\pi}$ has no ~~neg~~ neg wt cycles then f is optimal.

Gives us a family of algs:
choose neg cycle
aug
repeat until no more neg cycles.

Qs: Which neg cycles to choose, and how to get it.

Which neg cycle to choose?

$\epsilon \geq 0$ f feasible circulation
π node number assignment

f is ϵ optimal wrt π if every edge of the residual n/w $R_{f,\pi}$ has cost $\geq -\epsilon$

If we could get 0 optimal → good

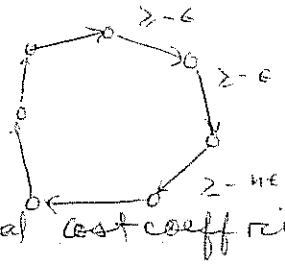
f is <u>ϵ-optimal</u> if $\exists\ \pi$ such that <u>f is ϵ optimal</u> wrt π

ϵ ↓ better. But don't need to go all the way to zero.

<u>Thm</u>: Assume all cost coeff are integers
Let n = no. of vertices.
Let $\epsilon < 1/n$
Then any ϵ optimal soln is optimal

<u>Proof</u>: $\epsilon < 1/n$ f is ϵ optimal wrt π
Residual n/w $R_{f,\pi}$  

any cycle ⇒ length $\geq n$



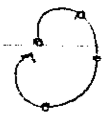
wt of cycle $\geq n(-\epsilon) \geq -1$
∴ cost of cycle ≥ 0 (∵ integral cost coefficients)

$\mu - \frac{\mu}{k} = \frac{\mu}{k}$ ... $= -\varepsilon(1-\frac{1}{k}) \geq -\varepsilon(1-\frac{1}{n})$

start w/ $E(f) = \varepsilon$

claim end w/ feas circ f' st $E(f') \leq (1-\frac{1}{n})\varepsilon$

pf $R_{f\pi}$ $\geq -\varepsilon$ After loops $R_{f'\pi}$



$\forall e$ wt $\geq -\varepsilon$
$\exists e$: wt ≥ 0
$\mu \geq -(1-\frac{1}{n})\varepsilon$

Hence $\forall$phase decreases ε by $\geq (1-\frac{1}{n})$ factor ie $\varepsilon' \leq (1-\frac{1}{n})\varepsilon$

Assume $\forall (uv)$ $|d(uv)| \leq C$, f_0 init circ, f_i circ after i-th iteration

$E(f_0) \leq C$, $E(f_{i+1}) \leq (1-\frac{1}{n})E(f_i) \leq (1-\frac{1}{n})^2 E(f_{i-1}) \leq (1-\frac{1}{n})^{i+1} E(f_0) \leq (1-\frac{1}{n})^{i+1} C$

stop when $E(f_t) < \frac{1}{n}$ ie. $(1-\frac{1}{n})^t C < \frac{1}{n}$ suffices $e^{-\frac{1}{n}t} C < \frac{1}{n} \Leftrightarrow nC < e^{t/n} \Leftrightarrow \ln(nC) <$

$e^{-x} > 1-x$ $\Leftrightarrow n \cdot \ln(nC) < t$ $\therefore t \leq 1 + n\ln(nC)$

$\therefore$ #phases $\leq 1 + n\ln(nC)$

phase in $T \sim m^2 = |E|^2$ $\Big\}$ tot. $T \sim m^2(1 + n \cdot \ln(nC))$

- Poly-T alg iff $T \sim$ poly(size of input); size of input = #bits of input $\geq m, n, \log C$
  $\therefore$ this is poly-T alg.
- Strongly poly if input-size is #e's, vert's, not bit-size $\Big\}$ $\Rightarrow$ this is not strongly-
  ie #arith op's = poly(m)
  $\exists$ strongly poly [Tardos '84]

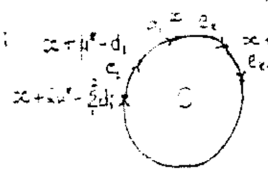
<u>e-mail:</u> $\Theta(\min\{nm \log(nC), m^2\})$ iterations of min-mean cycle cancel. alg of Goldberg + Tarjan

max-mean cut cancel. alg - dual of min-mean C cancel. alg

⇨ <u>Thm.</u> f not opt $\Rightarrow$ $E(f) = -\mu(R_{f_0})$

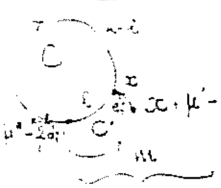
<u>Pf.</u> Let $\mu^* = \min\{\mu | C\}$; I'll find π: $d_\pi(e) \geq \mu^*$ $\forall e$, $\Rightarrow E(f) \leq -\mu^*$ since $\forall C$ $\mu(C)$ unaltered by any π

Start w/ C: $\mu(C) = \mu^*$, π: , $x = x + k\mu^* - \sum_1^k d_i$ works



$\forall e \in C$, $d_\pi(e) = \mu^*$

Take any C', st $C' \cap C \neq \emptyset$, π:
let $\mu(C') = \mu'$, $|C'| = \ell + m$



follow. C' [illegible]

$x + m\mu' - \sum_1^m d_i' \leq x + (k-\ell)\mu^* - \sum_1^{k-\ell} d_i$ $\Leftarrow$

By this, this vert have lower [illegible] (in C' viewpoint) it doesn't hurt us; still $d_\pi(e) \geq \mu^*$ $\forall e$. Same effect by ind.

$\Leftarrow m\mu' - [(\ell+m)\mu' - \sum_1^{\ell+m} d_i] \leq (k-\ell)\mu^* - [k\mu^* - \sum_1^{k} d_i]$

$\forall e \in C'$ $d_\pi(e) \geq \mu' \geq \mu^*$ $\Leftarrow$ $\ell\mu^* \leq \ell\mu' \Leftarrow \mu^* \leq \mu'$

CS270

29 Nov '90
Thursday

Review Thurs 13th 60 Evans @ 8:30

MIN COST CIRCULATION -

Goal: Find f, π s.t.
(π = price fn (Economics))

every edge of the residual nw $R_{f,\pi}$ has non neg modified cost

Equiv Goal: Find f s.t. $R_{f,0}$ has no cycle of neg wt.

Approach: Keep sending flow around negative cycles.

$\epsilon \geq 0$

f is ϵ-optimal wrt π if every edge of the residual network has modified cost $\geq -\epsilon$

ϵ = a measure of how well we are doing

Assume all costs are integers
Warning: modified costs will not generally be integers.

$\pm (\pi(u) + d(u,v) - \pi(v))$

f is ϵ-optimal if $\exists \pi$ s.t. f is ϵ-optimal wrt π

$E(f) = \min \{\epsilon \mid f$ is ϵ-optimal$\}$

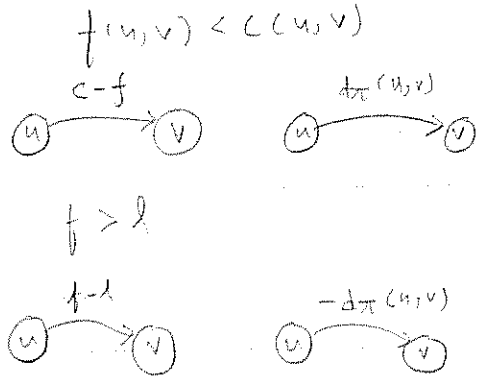
Strategy: compute $\{f_i\}$ $E(f_i) \downarrow$
method: to augment around negative cycles

/* Review: Residual N/w Construction */

f

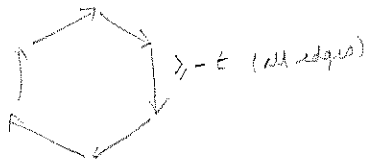
π

$d_\pi(u,v) = \pi(u) + d(u,v) - \pi(v)$



If f is ϵ-optimal, $\epsilon < 1/n$ then f is optimal.

Pf (3rd time round) f is ϵ-optimal wrt π
consider cycle in $R_{f,\pi}$



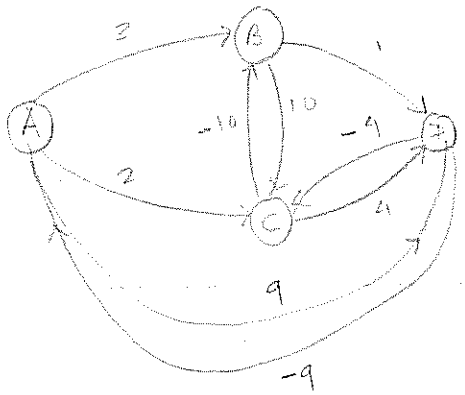
$\therefore$ cost of cycle $\geq n(-\epsilon) > -1$

but cost of cycle integer!
so cost of cycle is non negative.

$E(f) = \min\{\epsilon \mid f \text{ is } \epsilon\text{-optimal}\}$

Let G be a directed graph with numerical weights on the edges. For any cycle C, mean wt of C = avg wt of edges in C

$\mu(G)$ = min mean cycle weight.



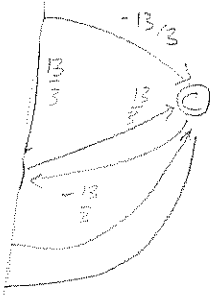
$\mu(G) = -13/3$

Algo 3 R-5 $O(mn)$ algo to compute min mean cycle wt.

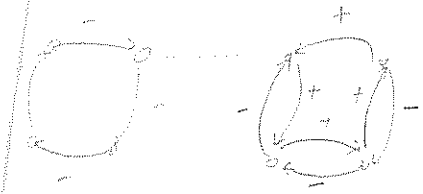
f

$R_{f,\pi}$

↖ modified costs



cle w/ all negative wts



, will at least one neg edge
, don't make any more +/ve edges
, any +ve edges introduced are of no consequence

no of ① → ② loop ≤ no. of negative edges

w/ $E(+) = \epsilon$

no feasible circulation f'

$1/n)\ \epsilon$; n = # of vertices

$\geq -\epsilon$

$R_{f',\pi}$

$\geq -\epsilon$

also somewhere around the cycle there is an edge whose wt is ≥ 0

∴ cycle mean $\geq -(1 - 1/n)\,\epsilon$

f is Fourier Transform

$E(f) \geq$

Theorem:

E

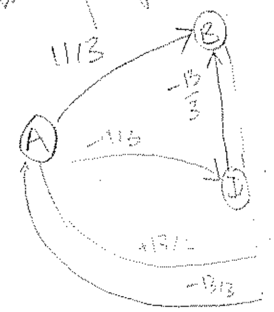
(Proof simple

Given a fea
compu
① find π
phase → ② form
③ keep
whose
until E(

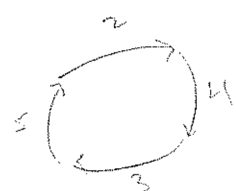
Min cycle mea
Add 13/3 to ev
min cycle mean
No negative c
∴ can find by s
numbers wh
of G' a non
i.e. p(u)

node numbering

We finally get



look for a cy



# of itera

Start Ph
end with
E(f')

$R_{f,\pi}$

To summarize what a phase achieves is to bring down ϵ a little bit.

Each phase decreases ϵ to at most $(1-1/n)\epsilon$

↳ precisely $\epsilon_{FINAL} \leq (1-1/n)\epsilon_{INIT.}$

Suppose $\forall (u,v)\ |d(u,v)| \leq C$

f_0 initial circulation
f_i circulation after iteration i.

$E(f_0) \leq C$
$E(f_{i+1}) \leq (1-1/n) E(f_i)$

stop when $E(f_t) < 1/n$

$\boxed{e^{-x} > 1-x}$ from ① to end (apply it)

$\therefore$ an upperbound on the number of phases
$\leq 1 + n \ln(nC)$

Exec time of a phase $= O(m^2)$
m = no. of edges
↳ wt time is $O(m^2 n \log(nC))$

so this a polynomial time algorithm?
* of steps bdd above by polynomial in size of i/p.

Size of input $\geq m, n$
$\geq \log C$
$\therefore$ polynomial. 1971

So there a strongly polynomial algorithm?
(Not of arithmetic operations is poly in m and n)
1989 Eva Tardos

CS 270

Tuesday

Concepts from complexity theory:
{P, NP, co-NP, reducability, NP-complete, NP-hard}
Coping w/ NP-hard problems

"A problem is tractable if it can be solved by an algorithm in number of steps bounded by a polynomial in the size of input."

defn of tractable ok
how about "problem", "solved", "algorithm", "number of steps", "size of i/p"

bitstring = universal rep. for input
let $\{0,1\}^*$ set of all finite strings of 0's and 1's

Problem: a function $f : \{0,1\}^* \to \{0,1\}^*$

Size of input = length

Algorithm: program for some universal abstract computer.
Turing m/c, RAM, ...

FP : set of functions computable in polynomial time.

Decision Problems: $f : \{0,1\}^* \to \{0,1\}$
x is accepted $\Leftrightarrow f(x) = 1$

P: class of decision problems solvable in poly time

MST
SPT
MaxFlow
Matching
Min Cost Flow
} FP

4 Nov 1999
Tuesday

Decision Problem $f: \{0,1\}^* \rightarrow \{0,1\}$

f lies in NP IFF $f(x) = \bigcup_{\{y \mid |y| \le p(|x|)\}} h(x,y)$

where $|y|$ = length of string y
p is a polynomial
h is a 2 variable fn computable in polynomial time.
$\cup$ is logic 'OR'

If $h(x,y) = 1$ then y is a witness for x.

SAT:

input: Boolean formula in CNF (easily encoded in bits)
(product of sums)

Ex: $(\bar{A} \cup B \cup \bar{C})(A \cup B)(\bar{B} \cup C \cup \bar{D})$ / product of clauses
clause

Property: There is a truth variable assignment that makes all clauses true.
$A=T$, $B=F$, $C=X$, $D=X$

Verify membership in NP
x CNF FORMULA
y TRUTH VALUE assignment
$p(|x|) = |x|$
$h(x,y) = 1 \Leftrightarrow y$ satisfies x

co-NP: $f: \{0,1\}^* \rightarrow \{0,1\}$

$\bar{f}$ complement of f $\bar{f} = 1 - f$
defn: $f \in$ co-NP $\Leftrightarrow \bar{f} \in$ NP.

f : matching problem

Input : graph G
Property : G has perfect matching

matching ∈ NP (∵ witness is matching itself)
matching ∈ co NP (∵ witness is oddset cover)

∴ matching ∈ NP ∩ co NP.

Satisfiability ∈ co NP?

$(A \vee B)(\bar{A} \vee \bar{B} \vee \bar{C})(B \vee \bar{D})$

∃ short witnesses for unsatisfiability?
Truth table, has not small

① $P \stackrel{?}{=} NP$
Every problem that has easy to check is easy to solve

② $NP \stackrel{?}{=} coNP$
If a prop. has short witnesses then its compl has short witnesses.

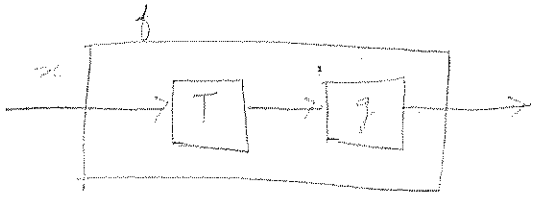
③ $NP \cap coNP \stackrel{?}{=} P$
less unlikely than the other 2.
Primality ∈ NP ∩ coNP

Poly-Time Reducibility

$f : \{0,1\}^* \rightarrow \{0,1\}^*$

$g : \{0,1\}^* \rightarrow \{0,1\}^*$

$f \leq_p g \iff \exists T : \{0,1\}^* \rightarrow \{0,1\}^*$
such that
- $T \in FP$
- $\forall x \; f(x) = g(T(x))$

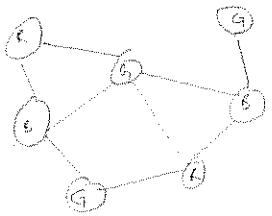


if $f \leq_p q$ and $q \in FP$ then $f \in FP$

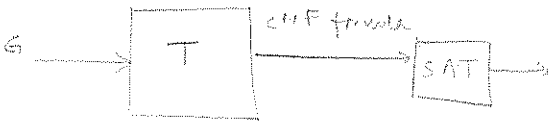
3-coloring

input: Graph G

property: Each vertex can be colored R, B, or G so that no two adjacent vertices have the same color



3-col $\leq_p$ SAT



$R_i = \begin{cases} T & \text{if vertex } i \text{ colored red} \\ F & \text{otherwise} \end{cases}$

B_i, G_i

Clauses: $\forall i \; (R_i \cup B_i \cup G_i)$

$\bar{R}_i \cup \bar{B}_i$ not both R and B

$\bar{R}_i \cup \bar{G}_i$

$\bar{B}_i \cup \bar{G}_i$

for each edge (i, j)

$R_i \cup \bar{R}_j \; ; \; \bar{B}_i \cup \bar{B}_j \; ; \; \bar{G}_i \cup \bar{G}_j$

obviously polynomial time reduction

$f : \{0,1\}^* \to \{0,1\}$ $\quad g : \{0,1\}^* \to \{0,1\}$

If $f \leq_p g$ and $g \in P$ then $f \in P$

Informally "g is at least as hard as f"

a decision problem $f : \{0,1\}^* \to \{0,1\}$ is NP-complete

if (i) $f \in NP$ (ii) every problem in NP is reducible to f

Cook-Theorem: SAT is NP-complete

Boolean Alg. is very general to express ... :
problems in NP