25/10/17

# P- Completeness

## Parallelizability

When is a "problem" parallelizable?

## Notions of parallelizability

(1) Sequential time $T(n)$

If $p$ processors are available, the ideal speedup is $\Theta(p)$.

speedup achievable for $1 \leq p \leq \alpha(n)$ where $\alpha(n)$ is an increasing function of $n$.

If $\alpha(n)$ is "small", the problem is not parallelizable.

(2) A problem is in NC (Nick's complexity class) if

(a) P can be solved by a polylogarithmic time algorithm using a polynomial no. of processors $(n^c)$.

Example: BFS $\in$ NC

$O(\log^2 n)$ time $\parallel$ algorithm.

$O(M(n) \log n)$ work

↳ complexity of multiplying two $n \times n$ matrices

Coppersmith - Winograd matrix multiplication → $n^{2.3}$ (best known)

let's take $M(n) = n^3$

$p$ processors

Running time $= O\left(\frac{n^3 \log n}{p} + \log^2 n\right)$

↳ slower than sequential $O(|E|+|V|)$ running time for $p = O(n \log n)$

Some problems ^ do not seem to ~~have~~ be in NC in P

Examples: (1) DFS
(2) Network flow
(3) Linear programming

|| optimization problems

$$\min \sum_{i=1}^{n} c_i x_i$$

subject to $A\underline{x} \leq \underline{b}$ and $\underline{x} \geq 0$

Decision problems → binary o/p

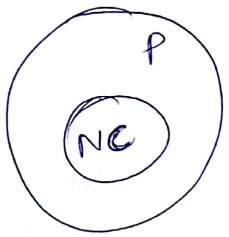
Given $(G, s, t, k)$, ~~Is~~ a network flow of amount $k$ possible?

NC consists of decision problems only.

$NC \subseteq P$ → obvious

whether $P \subseteq NC$ ? → not known

Popular belief: $\boxed{P \neq NC}$




→ maybe not true but popular belief

## NC-reduction

$P_1, P_2 \in P$

$P_1 \leq_{NC} P_2$ ($P_1$ is NC-reducible to $P_2$)

if there exists an NC algorithm that, given an i/p $I_1$ for $P_1$, generates an instance $I_2$ for $P_2$, such that $P_2(I_2) = 0 \Leftrightarrow P_1(I_1) = 0$

($P_2(I_2)$: ans of $P_2$ on $I_2$; $P_1(I_1)$: ans of $P_1$ on $I_1$)

We call a problem $Q \in P$ P-complete if every problem in P is NC-reducible to Q.

Lemma: $P_1 \leq_{NC} P_2$, $P_2 \in NC$
$\Rightarrow P_1 \in NC$

Lemma: $P_1 \leq_{NC} P_2$
$P_1 \notin NC \Rightarrow P_2 \notin NC$

Lemma: If any P-complete problem is in NC, then $P = NC$

Lemma: $P_1 \leq_{NC} P_2$ & $P_2 \leq_{NC} P_3 \Rightarrow P_1 \leq_{NC} P_3$ ⇒ transitivity holds

First example

CVP – circuit value problem

↳ combinational ckt consisting of AND, OR and NOT gates.

some inputs and one output

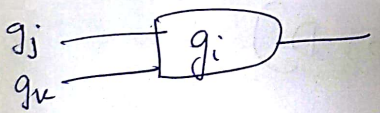
25/10/17

Thm:- The CVP is P-complete

Instance of CVP

$\langle g_1, g_2, g_3, \ldots, g_n \rangle$

Each $g_i$ is either an input or a gate




$j, k < i$

topo sorting of boolean ckt.

$g_n$ is the o/p.

Proof:- CVP $\in$ P ( evaluate in $O(n)$ time)

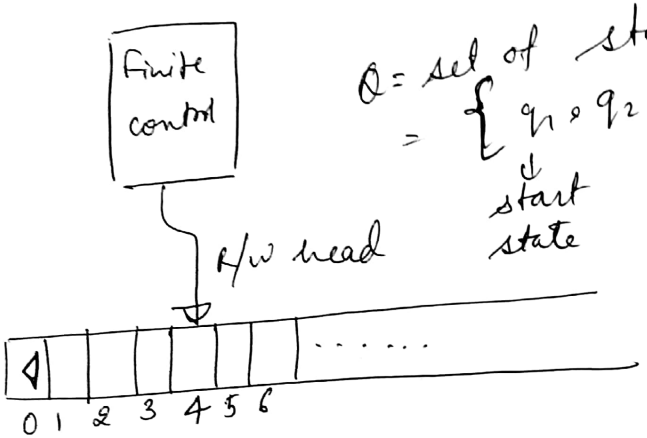
Let $Q \in P$

To show $Q \leq_{NC}$ CVP

There is a deterministic Turing m/c that accepts Q with running time $T(n)$

↑ polynomial in $n$.

# Turing Machine




$Q$ = set of states
$= \{q_1, q_2, \ldots, q_s\}$
$q_1$ ↓ start state

semi-infinite ~~tape~~ tape

Each cell contains symbols from $\Sigma$ = the tape alphabet
$= \{a_1, a_2, \ldots$

$$\delta : Q \times \Sigma \rightarrow Q \times \Sigma \times \{L, R\}$$

$$\delta(q, a) = (q', b, D)$$

## Acceptance

cell 1 contains 1

## Rejection

cell 1 contains 0

$t=0$ m/c starts working

$t=1, 2, 3, \ldots, T(n)$ → keeps on making transit

Three families of Boolean functions :-

(i) $H(i,t) = 1 \iff$ The head is scanning cell $i$ at time $t$.

$0 \le t \le T(n)$

$0 \le i \le T(n)$

(ii) $C(i,j,t) \iff$ Cell $i$ contains symbol $a_j$ at time $t$

$0 \le t \le T(n)$

$0 \le i \le T(n)$ , $1 \le j \le m$

(iii) $S(k,t) = 1 \iff$ The m/c is in state $q_k$ at time $t$.

$0 \le t \le T(n)$

$1 \le k \le s$

$s, m$ are ~~constants~~ constants.

**Init (Level 0)**

$$H(i,0) = \begin{cases} 1 & \text{if } i = 1 \\ 0 & \text{if } i \ge 2 \end{cases}$$

$$C(i,j,0) = \begin{cases} 1, & \text{if } 1 \le i \le n \text{ and } a_j \text{ is the } i\text{th symbol in the i/p.} \\ 0, & \text{otherwise} \end{cases}$$

$$S(k,0) = \begin{cases} 1, & \text{if } k = 1 \\ 0, & \text{if } k \ge 2 \end{cases}$$

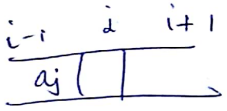
## From level $t$ to level $t+1$

$$H(i,t+1) = H(i-1,t) \sum_{j,k \in I_R} S(k,t) \cdot C(i-1,j,t)$$

$$+ H(i+1,t) \sum_{j,k \in I_L} S(k,t) \cdot C(i+1,j,t)$$

$q_k$ ↓ boolean or

↑ boolean and




$$I_L = \{(j,k) \mid \delta(q_k, a_j) = (*,*,R)\}$$

$$I_L = \{(j,k) \mid \delta(q_k, a_j) = (*,*,L)\}$$

$1 \le j \le m$, $1 \le k \le s$ → $(ms)$ Constanst time sum.

$O(T(n))$ processors, $O(1)$ time ↓ for all $i$

$$C(i,j,t+1) = \overline{H(i,t)} \cdot C(i,j,t) + \overline{H(i,t)} \sum_{(k,j')\ \delta(q_k,a_j)=(*,a_j,*)} \{C(i,j',t) \cdot S(k,t)\}$$

→ Constant fold sum.

Total effort ≡ $O(T(n)^2)$ processors for $(i,t)$ in $O(1)$ time

$$S(k, t+1) = \sum_{(i,j,k')} H(i,t) \cdot C(i,j,t) \; S(k',t)$$

s.t. $\delta(q_{k'}, q_j) = (q_k, *, *)$

$O(T(n)^2)$ processors

$O(k \log n) = O(\log n)$ running time

$j, k'$ constants

o/p - $C(1, 1, T(n))$

$(T(n)^2)$ processors → poly in $n$.

$O(\log n)$ running time

So, reduction algo is in NC

MCVP : monotone ckt value problem
only AND and OR gates

MCVP is P-complete

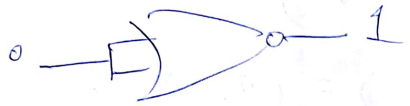
$$\overline{H(i,t)} = \sum_{i' \neq i} H(i',t)$$

01/10/17

## NORCVP

Ckt consists only of NOR gates

Inputs: 1 (All i/ps are 1's)

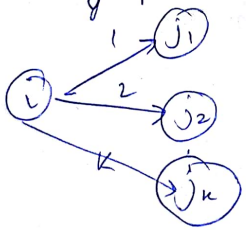



If Q is P-complete and $Q' \in P$ and $Q \leq_{NC} Q'$, then Q' is also P-complete.

**Thm:-** ORDERED_DFS is P-complete

It is assumed that there is ordering on outgoing edges.

Directed graph




first, explore $j_1$, then $j_2$ ⇒ and so on...

There is a source s & two other vertice u, v ∈ V

To decide whether dfs(u) < dfs(v)

NORCVP $\leq_{NC}$ ORDERED_DFS

$(g_1, g_2, \ldots, g_n)$ → NOR ckt

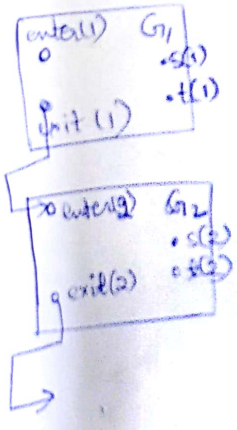
$G = (V, E)$

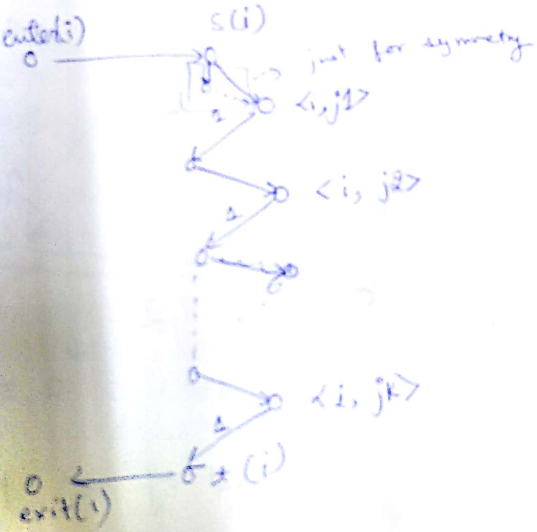
$g_n = 1 \iff dfs(u) < dfs(v)$

$g_i$ —prepare→ gadget $G_i$

Construction of $G_{ri}$

$g_i$ is an i/p

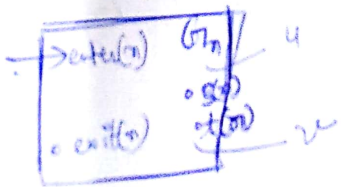


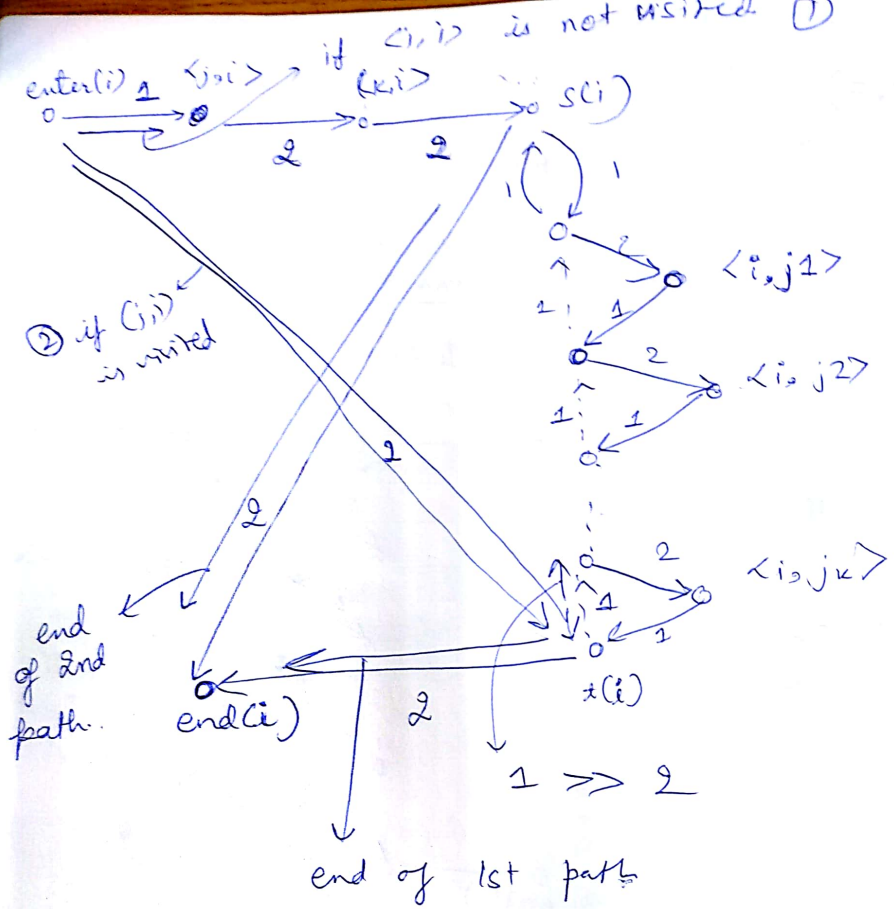




$g_i$ is fed to $g_{j1}, g_{j2}, \ldots g_{jk}$

$g_i$ is a gate:

$g_i = (g_j + g_k)$






② if (j,i) is visited

Claim :- If $g_i = 1$, then the ~~path~~ path ① is explored

If $g_i = 0$, the the path ② is explored.

Pf: [By induction on n]

$g_i$ input → obvious

$g_i = (g_j + g_k)'$

$g_i = 1$
$\Rightarrow g_j = g_k = 0$

Red path in $G_j$ and $G_k$.

$\langle j, i \rangle$ , $\langle k, i \rangle$ unvisited

if $g_i = 0$

$\Rightarrow g_j = 1$ or $g_k = 1$

$\langle j, i \rangle$ visited in $G_j$ } by induction

$\langle k, i \rangle$ visited in $G_k$

02/11/17

## Max Flow Problem

$N = (V, A)$ → not a ~~ADG~~ DAG

directed graph

$s, t \in V, \quad s \neq t$

source, sink

Edge $(u, v) \in A$ has a capacity $c(u, v)$ [non-ve integers]

$u \in V, \quad u \neq s, t$

→ Capacity cond.

flow $f: A \to \mathbb{Z}_{\geq 0} \Rightarrow \boxed{0 \leq \text{flow}(u, v) \leq c(u, v)}$

$\forall v \neq s, t$

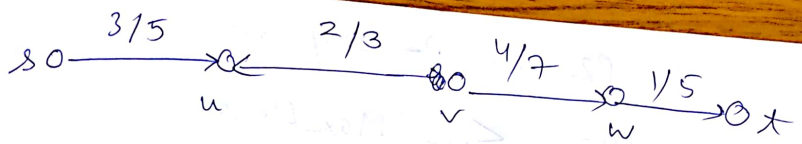
$$\sum_{(u,v) \in A} f(u, v) = \sum_{(v,w) \in A} f(v, w) \quad \to \text{conservation constraint}$$

$$|f| = \sum_{(s,v) \in A} f(s, v) - \sum_{(u,s) \in A} f(u, s)$$

$$= \sum_{(u,t) \in A} f(u, t) - \sum_{(t,v) \in A} f(t, v)$$

Task: to maximize $|f|$

Decisional question: whether the max. flow $|f|_{max}$ is odd or even




$f(u,v) / c(u,v)$

augmenting

An ~ path consists of forward and backward edges.

Ford-Fulkerson algorithm :- Repeatedly find augmenting paths and do some other work. and construct a residual network. until no further augmenting paths are found.

Every maximal flow is also maximum. → prove it.

$O(n^3)$ algos for solving the Max-flow problem.

Prove :- Max-flow $\in$ P.

MCVP 2

↓

monotone dot value problem

↳ only AND and OR gates

→ i/ps are 0s and 1s.

→ Each i/p goes to at most one gate (fan-out is 0 or 1)

→ Each gate can have fan-out at most 2.

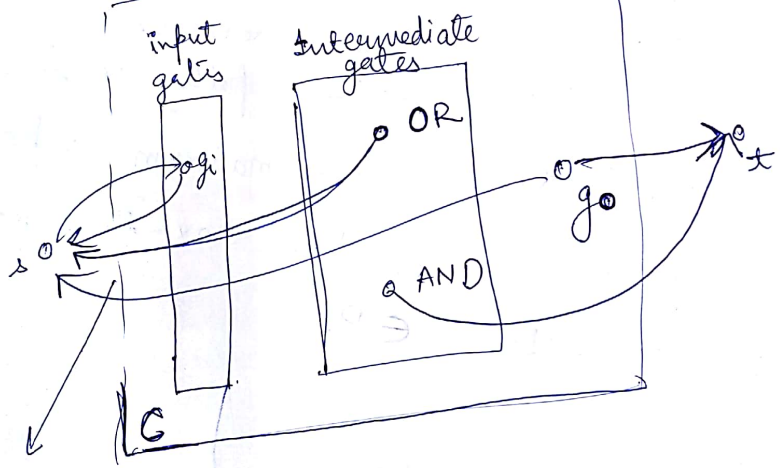
→ The o/p is an OR gate

Exercise:- MCVP2 is P-complete

MCVP2 $\leq_{NC}$ Max-flow

$C \longmapsto N = (V, A)$

$C = \langle g_n, g_{n-1}, g_{n-2}, \ldots, g_0 \rangle$ ↳ o/p OR gate




bidirectional connections

1. $g_i$ is an i/p

$c(i,s) = 2^i$

$c(s,i) = \begin{cases} 2^i & \text{if } g_i = 1 \\ 0 & \text{if } g_i = 0 \end{cases}$

2. $g_i$ is an AND gate → intermed. gates.

$g_i = g_j$ AND $g_k$

$c(j,i) = 2^j$

$c(k,i) = 2^k$

$j, k > i$

$c(i,t) = 2^j + 2^k - d$

where d is the fanout of 2.

3. $g_i$ is an OR gate

$g_i = g_j$ OR $g_k$

$c(j,i) = 2^j$

$c(k,i) = 2^k$

$c(i,s) = 2^j + 2^k - d \cdot 2^i$

where $d$ is the fan-out of $g_i$.

4. $c(o,t) = 1$

Define, $f_i$ is a flow f on $\mathbb{N}$

1. $g_i$ is an i/p

$f(s,i) = c(s,i)$

$f(i,s) = \begin{cases} f(s,i), & \text{if fan-out of } g_i \neq 0 \\ 0, & \text{otherwise} \end{cases}$

2. $g_i$ feeds $g_j$

$f(i,j) = \begin{cases} 2^i & \text{if } g_i = 1 \\ 0 & \text{if } g_i = 0 \end{cases}$

3. $g_i$ is an AND gate where $g_i = g_j$ AND $g_k$

$f(i,t) = \begin{cases} c(i,t) & \text{if } g_i = 1 \\ f(j,i) + f(k,i) & \text{if } g_i = 0 \end{cases}$

4. $g_i$ is an OR gate

$g_i = g_j$ OR $g_k$

$i \neq 0$

$$f(i,s) = \begin{cases} f(j,s) + f(k,i) - d \cdot 2^i, & \text{if } g_i = 1 \\ 0, & \text{otherwise} \end{cases}$$

5. $$f(0,s) = \begin{cases} f(j,0) + f(k,0) - 1, & \text{if } g_0 = 1 \\ 0, & \text{if } g_0 = 0 \end{cases}$$

6. $$f(0,t) = \begin{cases} 1, & \text{if } g_0 = 1 \\ 0, & \text{if } g_0 = 0 \end{cases}$$

03)

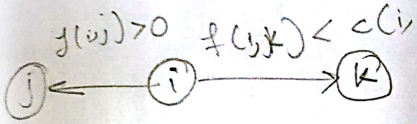
03/11/17

claim:- $f$ is a maximal flow.

Proof by contradiction:- $f$ is not maximal

$\Rightarrow \exists$ an augmenting path


$f(s,i) = c(s,i)$ $\forall e$ going out of $s$ is filled to the capacity.




$j \neq t \quad k \neq s$

Case 1: $g_i$ is an i/p gate

$\Rightarrow j = s$

$g_i = 0 \Rightarrow f(s,i) = f(i,s) = 0$

$f(i,j) > 0$ is false

$g_i = 1 \Rightarrow f(s,i) = 2^i = f(i,k) \quad k \neq t$

$\Rightarrow f(i,j) = 0$

Case 2:- $g_i$ is an AND gate

$f(i,j) > 0$

$\Rightarrow g_i = 1$

$\Rightarrow f(i,k) = c(i,k)$ { $k \neq t$ both are equal to 1, $k = t$ we have taken, $f(i,t) = c(i,t)$

Case 3:- $g_i$ is an OR gate, $i \neq 0$

$k \neq s, t$

$k$ is an internal gate.

$f(i,k) < c(i,k)$

$\Rightarrow f(i,k) = 0$

$\Rightarrow g_i = 0$

$\Rightarrow f(i,j) = 0$ { $j \neq s$, Then ij are internal gates; $j = s$, by defn $f(i,s) = 0$ if $g_i = 0$

Case 4:- $i = 0$, $g_0 = 0$ $\Rightarrow$ Both i/ps are zero

$\Rightarrow$ zero inflow

$\Rightarrow$ zero outflow

$\Rightarrow f(0,t) = 0$

$g_0 = 1$ $\Rightarrow$ at least one i/p is 1

$\Rightarrow$ inflow > 0

$\Rightarrow f(0,t) = 1$ to maximize $f$

## Linear Inequation (LI)

If To decide whether $AX \leq b$ is solvable.

Prop:- LI is P-complete

LI $\in$ P $O(n^3)$ algorithms exist

CVP $\leq_{PC}$ LI

$\langle g_1, g_2 - - - g_n \rangle$

Case 1: $g_i$ is an input

$g_i = 0$

$g_i = 1$

Introduce variables

$x_1, x_2, \ldots, x_n$

$x_i \geq 0$ $(-x_i \leq 0)$

$x_i \leq 0$

$x_i \geq 1$

$x_i \leq 1$

Case 2:- $g_i = g_j'$

$x_i = 1 - x_j$

$x_i \geq 1 - x_j$

$x_i \leq 1 - x_j$

Case 3:- $g_i = g_j$ AND $g_k$

$x_i \geq 0$

$x_i \leq x_j$

$x_i \leq x_k$

$x_i \geq x_j + x_k - 1$

if $x_j = 1$ & $x_k = 1$ $\Rightarrow x_i = 1$

Case 4:- $g_i = g_j$ OR $g_k$

$x_i \leq 1$

$x_i \geq x_j$

$x_i \geq x_k$

$x_i \leq x_j + x_k$

Case 5:- $g_n$ is the output gate $x_n \geq 1$

$x_n \leq 1$

§ Graph Algorithms

- Readability - connected components
- MST

$G = (V, E)$ simple undir. graph

→ To find out the connected components of G.

Given $u, v \in V$, decide whether u & v belong to the same component

conn_comp $\leq_{NC}$ DFS

↳ P-complete

Solve conn. comp. without using dfs.

Directed Pseudo-forests :-

$C : V \to V$

Construct a graph $F = (V, A)$ such that $A$ is the set of $A = \{ (u, C(u)) \mid u \in V \}$ (some arbitrary $C$)

$C$ partitions $V$ into $V_1, V_2, \ldots, V_k$ and $A$ into $A_1, A_2, \ldots, A_k$ such that $(V_i, A_i)$ is a tree with an extra edge.

$V = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$

$C(1) = 8$

$C(2) = 6$

$C(3) = 5$

$C(4) = 9$

$C(5) = 5$

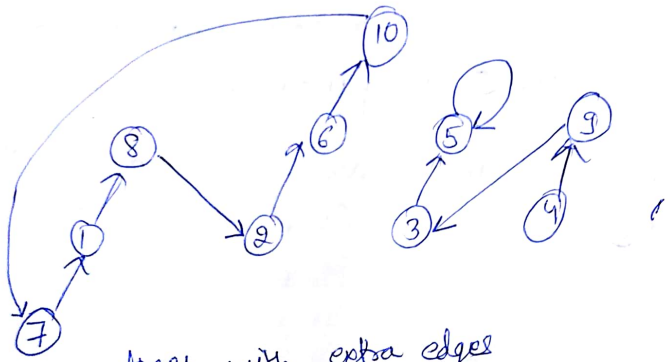
$C(6) = 10$

$C(7) = 1$

$C(8) = 2$

$C(9) = 3$

$C(10) = 7$




trees with extra edges

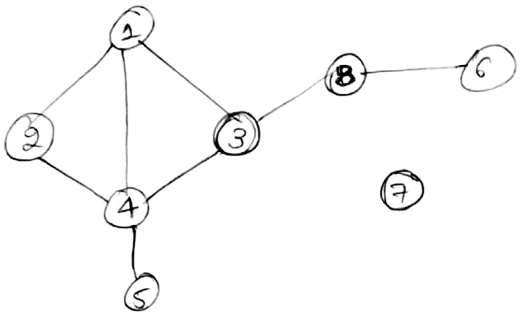
$n$ vertices and $n$ edges → tree and one extra edge

Every comp. $V_i$ is a ~~tree~~ component

star

$V = \{1, 2, \ldots, n\}$

$$C(u) = \begin{cases} u, & \text{if } u \text{ is an isolated vertex} \\ \min_{(u,v) \in E} (v) & \text{otherwise} \end{cases}$$




c(1) = 2

c(2) = 1

c(3) = 1

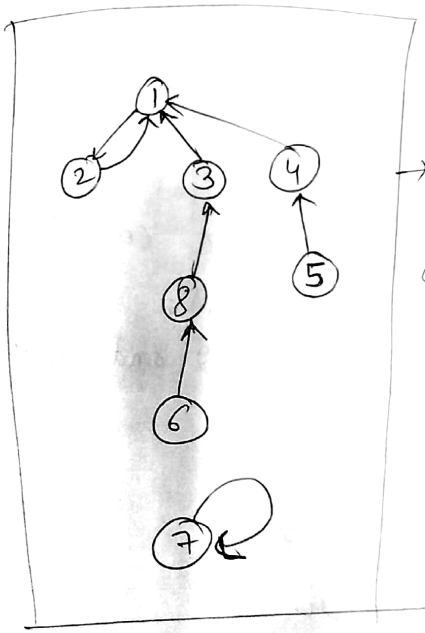
c(4) = 1

c(5) = 4

c(6) = 8

c(7) = 7

c(8) = 3




→ Already identified components.

Swap (3) & (8)

c(1) = 2

c(2) = 1

c(3) = 6

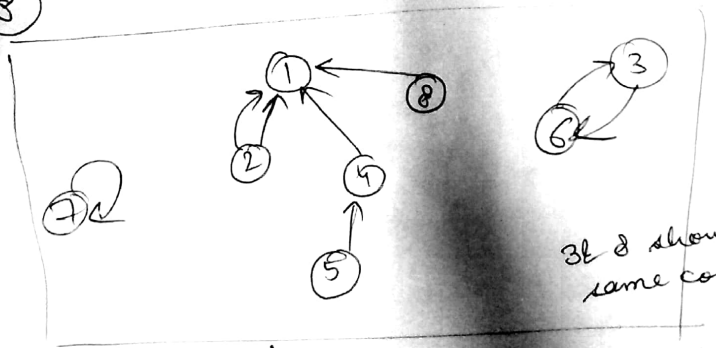
c(4) = 1

c(5) = 4

c(6) = 3

c(7) = 7

c(8) = 1




3 & 8 should be in same component

↓

We have yet not identified the components in org. graph.

Proposition:

1. Each $V_i$ ~~belongs to~~ is a subset of a connected component of G.
2. The cycle in each $T_i$ is either a loop or a 2-cycle.
3. $\min_{v \in V}(v)$ lies on this cycle.

Proof:-

1. Each directed edge of $T_i$ comes from an edge in G.

2 and 3. Let $r = \min V_i$

If $|V_i| = 1$, 2 and 3 are obvious.

$|V_i| \geq 2$

$u = C(r) \neq r$

$C(u) = r$

09/11/17

$C : V \to V$

Complexity of computing C :-

Each node min $O(\log(\#\text{ of nbrs}))$ time
$O(\#\text{ of nbrs})$ work.

$O(\log n)$ time
$O(n)$ work
↓
for matrix representation.

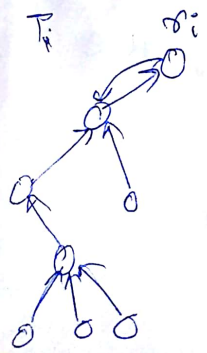
$$C(v) = \begin{cases} v, & \text{if } v \text{ is isolated} \\ \min(w), & \text{otherwise} \\ (v,w)\in E & \end{cases}$$

Total running time : $O(\log n)$ time
& $O(n^2)$ work

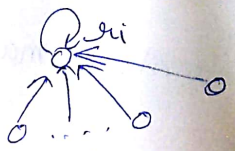
Representation for each pseudo tree

"The" root → the smaller value of the 2-cycle

Every node in ~~T~~ should be labelled by $r_i$.




Pointer Jumping reduces this tree to a star.




Bounded time with $O(\log n)$ time
$O(n\log n)$ work
or
$O(n)$ ?

After C is defined & ptr jumping is done, you have a sequence of stars & isolated vertices.

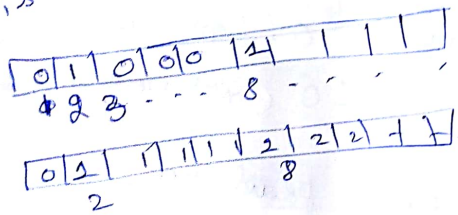
Each star is a part of a component.

$r_i$ is k/a "supervertex".

① Prepare an undirected graph on the super vertices.

② Rename them ($r_i$'s) as $1, 2, \ldots, n_k$. using prefix computation. $O(\log n$ time$)$ $O(n)$ work

$r_i$'s 2 8 13 17 21

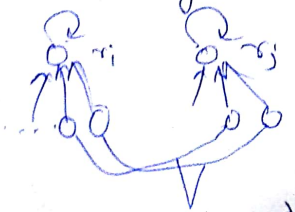



Prefix sum

$\forall (u,v) \in E$
if $u$ belongs to $T_i$
and $v$ belongs to $T_j$
for some $i, j$, $i \neq j$
then add the edge $(i,j)$ to the supervertex graph.

$O(1)$ time
$O(m)$ time
$\rightarrow m = n^2$ (dense graph)

Concurrent write may take place.




both trying to write same thing.

Repeat # of iterati

$n_0 =$

kth it

$n_{k-1}$

$n_k$

$\Rightarrow$ log

Overall Total

Repeat in the supervertex graph

# of non-trivial stars at the end of the kth iteration $= n_k$

trivial star

$n_0 = n$

kth iteration

$n_{k-1}$ ↓ $n_k$

Let $n'_{k-1}$ be the # of isolated vertices. Each star contains atleast two nodes.

$$n_k \leq \frac{1}{2}\left(n_{k-1} - n'_{k-1}\right)$$

$$n_k \leq \frac{1}{2} n_{k-1} \quad \textcircled{1}$$

$\Rightarrow \log n$ iterations are necessary.

Overall running time $= O(\log^2 n)$

Total work $= O\left(\sum_{k=1}^{\log n} n_k^2\right)$

$$= O\left(n^2 + \frac{n^2}{4} + \frac{n^2}{16} + \cdots\right)$$

Work $= O(n^2)$

↳ optimal for dense graphs.

⇒ a better algo for sparse graphs.

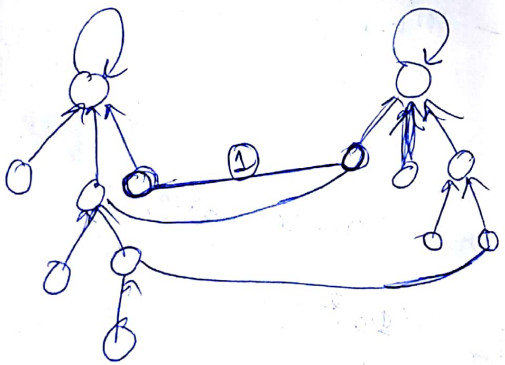
Step 3:- Reverse the contraction process.

## An algo for sparse graphs

Each iteration takes $O(\log m)$ time for previous algo.

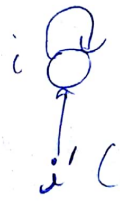
We desire each iteration to run in $O(1)$ time.

Computation of $C_i$, the supervertex graph → cannot be done still, we will be merging trees.



Each tree is a part of a connected component.

To Start with: $n$ disjoint trees on single vertices



Merging / Grafting one tree to another

~~The tree~~

Merge $(T_i, T_j)$

$u \in V(T_i)$, $v \in V(T_j)$

$(u, v) \in E(G)$

Requirement:-

$u$ ~~is not~~ must be the root of $T_i$ or a child of $T_i$.

Problem 1:- How to merge trees with edges belonging to far-away-from-root nodes?

Path compression may take $O(\log n)$ time in each iteration.

Solve this using one iteration of ptr jumping.

→ Every node will now point to its parent of parent.

Nodes become closer and closer to the root with iterations.

Problem 2:- link ① To avoid Merge $(T_i, T_j)$ & Merge $(T_j, T_i)$ simultaneously.

Check for a root or its children?

$D(D(u)) = D(u)$ → either a root or
↳ parent ptr child of a root

If $\{D(D(u)) = D(u)$ and $D(D(v)) = D(v)\}$
and $(u,v) \in E(G)$ and $D(u) \neq D(v)$
then merge the larger vertex to the small
vertex.

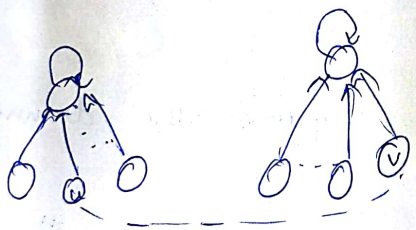
10/11/17

One iteration

1. for all $(u,v) \in E$ pardo → O(1) time, O(m) work
   if $(D(D(u)) = D(u))$ and $(D(v) < D(u))$
   then set $D(D(u)) = D(v)$
2. Preprocessing → O(1) time, O(n) work
2. for all $(u,v) \in E$ pardo
   if u belongs to a star, and $(D(u) \neq D(v))$ set $D(D(u)) = D(v)$
   ↳ O(1) time, O(m) work
3. If all pseudotrees are stars, exit.
   otherwise, for all $u \in V$ pardo
   set $D(u) = D(D(u))$
   ↳ O(1) time, O(n) work

Step 2:

$(u,v) \in E$
u and v both belong to stars.




But step 1 must have merged one of these into some other star/tree.
So, if there are two stars after step 1, they cannot share an edge in G.
⇒ exit point is justified.

How to ~~belong~~ know that u belong to a star?

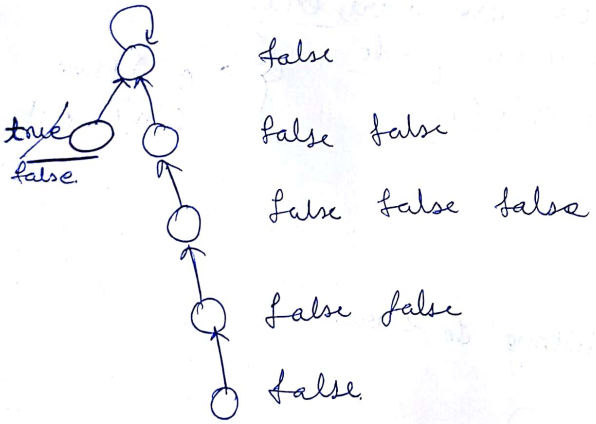
for all $u \in V$ pardo [Preprocessing for step 2]

star(u) = true

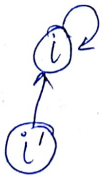
if $(D(D(u)) \neq D(u))$

~~set~~ set star(u) = star(D(u)) = star(D(D(u))) = false

set star(u) = star(D(u))




Mental exercise :- Step 2 requires the dummy node



## Running Time

Each iteration takes $O(1)$ time and does $O(m+n)$ work.

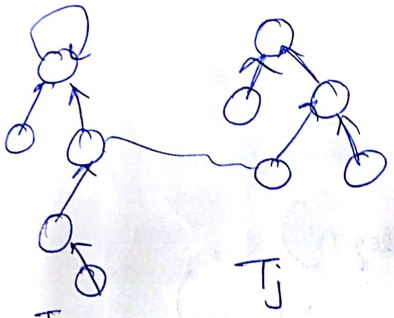
How many iterations?

Let $K$ be a component in $G_r$.

$ht_K(K)$ = sum of heights of all $u \in K$ in their respective trees after the $k$th iteration.

or

sum of all the trees containing members of $K$ after the $k$th-iteration

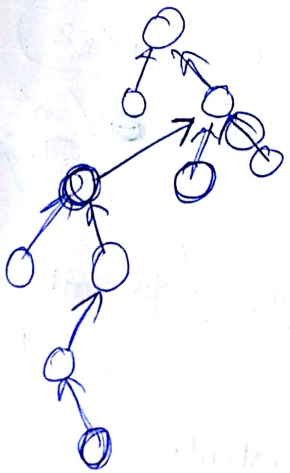



$ht(T) \leq ht(T_i) + ht(T_j)$

$ht_0(K) = |K|$

↓

these many individual trees of height 1 initially.



Step 3:- reduces the height of each non-

by a factor of 2/3.

After $l$ iterations

$$ht_l(K) \leq \left(\frac{2}{3}\right)^l |K|$$

After $\log_{3/2}(|K|)$ iterations, K is converted to a star.

$$|K| \leq n$$

$\Rightarrow O(\log n)$ iterations.

Total running time : $O(\log n)$

Total work done : $O((m+n)\log n)$

↓

better than $O(n^2)$

if $m = O\left(\frac{n^2}{\log n}\right)$

PRAM model :-

Steps 1 & 2 :- require arbitrary / priority CRCW PRAM.

compatible with preprocessing step 2.

Syllabus

Chap 1:- PRAM models, WT scheduling | Skip Network models

Chap 2:- Basic techniques only

Chap 3:- Atleast remember the results

Post-midsem

## Minimum Spanning Tree (MST)

$G = (V, E)$ connected undir graph

A spanning tree for $G$ is $T = (V, E')$ which is a tree

Weight function
$w: E \to \mathbb{R}$

$wt(T)$ = sum of the wts of the edges in $E'$

To find a spanning tree $T$ s.t. $wt(T)$ is as small as possible.

WLOG, we ~~may~~ assume that all weights are distinct.

Given $w: E \to R$, ~~distinct~~ construct

$w': E \to \mathbb{R} \times \mathbb{N}$

$e \to (w(e), $ [illegible]

⇒ The MST of G is unique (distinct weights)

$c : V \to V$

$u \mapsto v$

s.t. $v = \underset{x}{\operatorname{argmin}} \{ w(u,x) \mid (u,x) \in E \}$

$$v = \underset{x}{\operatorname{argmin}} \{ w(u,x) \mid (u,x) \in E \}$$

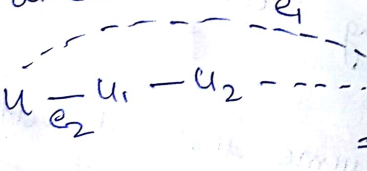
c is well-defined.

**Lemma 1:** $\forall u \in V$, $(u, c(u))$ is an edge in the MST.

**Proof:-** $T \to$ MST

$(u, c(u)) \notin E(T)$

T is connected without a direct link from u to c(u)




$w(e_1) < w(e_2)$

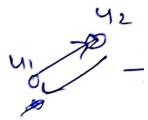
$\Rightarrow w(T + e_1 - e_2) < W(T)$

hence T is not an MST.

**Lemma 2:** pseudotre

**Proof:-**

k-cycle, k


**Lemma 3:-** L

for each
of $V_i$ wit

All $e_i$ m

Sollin's A

Init : A for

Compute c :

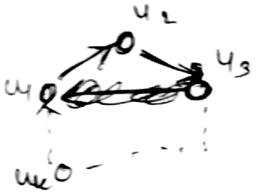
CREW P

O(log

Lemma 2 :- $C$ defines a pseudoforest on $V$. Each pseudotree contains a unique 2-cycle.

Proof :- 1-cycle $\Rightarrow$ $c(u) = u$ (not possible in this choice of $C$; $(u,u) \notin E$)

$k$-cycle, $k \geq 3$





$u_2 = c(u_1)$
$u_3 = c(u_2)$
$\vdots$
$u_k = c(u_{k-1})$
$u_1 = c(u_k)$

$\Rightarrow$ $T$ contains a cycle. Hence, a contradiction.


$\rightarrow$ same edge only.

Lemma 3 :- Let $V_1, V_2, \ldots, V_k$ be a partition of $V$. For each $i$, let $e_i$ be the min-weight edge connecting a vertex of $V_i$ with vertex of $V - V_i$. All $e_i$ must be in the MST $T$

Sollin's Algo

Init : A forest of $n$ different vertices.

Compute $c : V \rightarrow V$ $\rightarrow$ minimum finding

CREW PRAM

$O(\log n)$ time, $O(n^2)$ work

Mark each $(u, c(u))$ as an MST edge.

This gives a collection of pseudo-trees (vertex disjoint).

Use ptr-jumping to convert each pseudo tree to a rooted star. → $O(\log n)$ time, $O(n \log n)$ work

Treat each star as a supervertex and generate the supervertex graph.

Keep on iterating until only one tree is left.

$G_0$ $G_1$ $G_2$ ------- $G_k$

$n_0 = n$ $n_1$ $n_2$ $n_k = 1$

$$n_{i+1} \leq \frac{n_i}{2}$$

Max $\log n$ iterations are required

$\therefore$ total running time $= O(\log^2 n)$

Total work done $= O\left(n^2 + \frac{n^2}{4} + \cdots\right)$

$= O(n^2)$