CS60026 / CS60040

Parallel Algorithms ≠ Programming

TEXTBOOKS:

Joseph Jàjà, Introduction to Parallel Algorithms

Michael Quinn.

Grama.

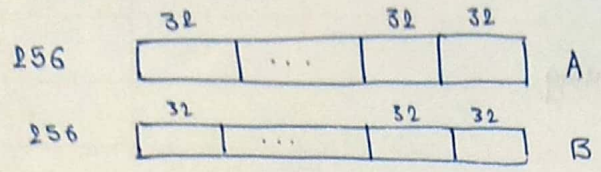
CT - 20

MS - 30

ES - 50.

→ Multicore Machines

→ SIMD



→ GPGPU

- Parallel Programming Models

P → a computational problem

$T^*(n)$ = best known / optimal sequential running time

A → a parallel algo to solve P.

$T_p(n)$ = running time of A on p processors.

$S_p(n)$ = Speedup = $\frac{T^*(n)}{T_p(n)} \leq p$

Ideal case : $S_p(n) \approx p$

Note: $T_1(n) \geq T^*(n)$

~~Problems~~

→ Concurrency
→ Dependency (some problems are "inherently" sequential)
→ Communication overhead.

} Reasons why ideal case is not always achieved.

$E_p(n)$ = Efficiency = $\frac{T_1(n)}{p\,T_p(n)}$

As p becomes larger than some value, efficiency decreases.
So, there's a limit on the no. of processors

$T_\infty(n) \leq T_p(n)$ ∀ p.

$$\frac{T_1(n)}{p\,T_p(n)} \leq \frac{T_1(n)}{p\,T_\infty(n)}$$

if $p \gg \frac{T_1(n)}{T_\infty(n)}$, efficiency is small.

"optimal" no. of processors.

- DAGs
- PRAM ✓
- Network

Chap 2 to n-1

Design and analysis of parallel algos.

Chap n

Limitations of parallelization.

Models of Parallel Computation

- Simplicity
- Implementability

Three models

Directed Acyclic Graphs (DAGs)

Architecture Independent

Nodes of in-degree zero - Inputs

Nodes of out-degree zero - Outputs.

Internal nodes - Operations

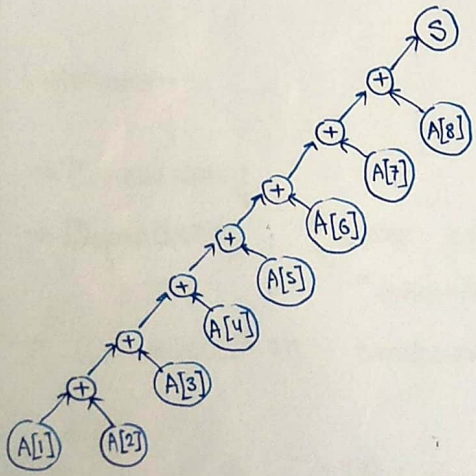
→ at most two incoming edges.

$A[1 \dots n]$ $t=3$

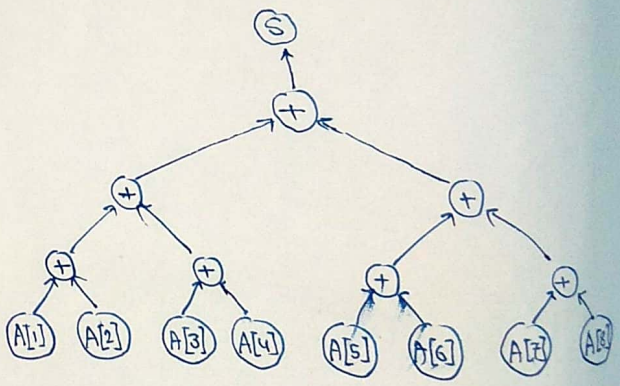
$n = 2^t$

To add eight elements :

DAG 1



DAG 2



Schedule:

p processors

An assignment of a processor and a time to each internal node.

i^{th} internal node gets a pair (p_i, t_i)

p_i ← operation mapped to p_i at a time t_i

Assumption -

(1) Each ~~of takes~~ operation takes one unit time.

(2) The output of an operation is available to all processors immediately after the operation ~~in~~ is performed.

Requirements

(1) If $t_i = t_k$ for some $i \neq k$, then $p_i \neq p_k$.

(2) If there is an edge from node i to node j then $t_j \geq t_i + 1$.

For a given schedule,

running time = $\max t_i$

$T_p(n) = \min(\max t_i)$

↑ over all possible schedules

DAG 1

$T_p(n) = n-1$ irrespective of p.

DAG 2

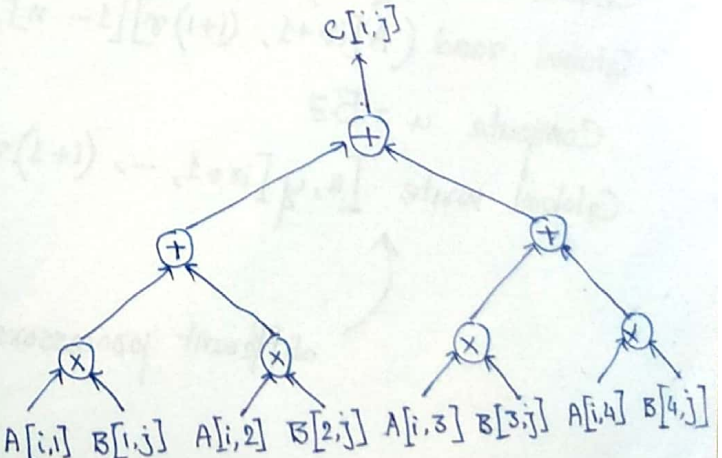
$T_{n/2}(n) = \log n$.

Matrix Multiplication

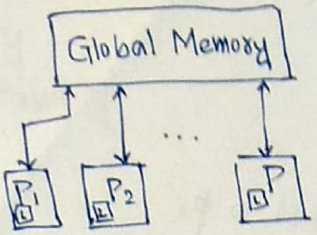
A, B nxn matrices

$C = AB$

$$C[i,j] = \sum_{l=1}^{n} A[i,l] \cdot B[l,j]$$



PRAM (Parallel Random Access Machine)



L = local memory

Input, Output → in global memory
Individual computations done on data in local memory.

Effects
- B = A → global read (A, B): fetch data A from global memory to B in local memory.
- D = C → global write (C, D): write back from local memory C to global memory D

Synchronised PRAM – same clock for all processors.
Non-synchronised PRAM – private clock for each processor.

$y_i = A_i x$ ← done at P_i

$\bar{y} = A\bar{x}$ (A: n×n, x̄: n×1)

p processors
n = pr

$$Ax = \begin{bmatrix} A_1 \\ A_2 \\ \vdots \\ A_p \end{bmatrix} X = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_p \end{bmatrix}$$

Each A_i is an r×n block

Each y_i is an r-dim sub vector

Code for P_i

Global read (x, z)
Global read (A[ir+1, (i+1)r][1 ... n], B)
Compute u = Bz
Global write [u, y[ir+1, ..., (i+1)r]]

different processors

Sum of hypercube

$d = 3$

For $i = d-1$ down to 0.

For $j = 0$ to $(2^i - 1)$

P_j gets $A_{j(i)}$ from $P_{j(i)}$

P_j sets $A_j = A_j + A_{j(i)}$

$O(\log n)$ time.

Broadcasting in a d-dimensional hypercube.

Initially, P_0 contains X.

$O(\log n)$ time where $n = 2^d$.

Step 1: P_0 sends X to P_1.

Step 2: P_0, P_1 send (in parallel) X to P_2, P_3 resp.

...

Step i: $P_0, P_1, \cdots, P_{2^{i-1}-1}$ sends (in parallel) X to $P_{2^{i-1}}, P_{2^{i-1}+1}, \cdots, P_{2^i-1}$

Matrix multiplication on a hypercube

$C = AB$ ($n \times n$ matrices)

$n = 2^t$

$p = n^3 = 2^{3t}$ ($3t$-dimensional hypercube).

processor IDs: $0, 1, \cdots, n^3 - 1$

Map these IDs to i, j, k.

$P_{i,j,k}$ $i, j, k \in \{0, \cdots, 2^t - 1\}$

$ID = i \times n^2 + j \times n + k$.

Fix i, k.

Let j vary

$\{ P_{i^*, j, k^*} \mid j \in \{0, \cdots, 2^t - 1\} \}$ form a t-dimensional subcube.

$$C(i,j) = \sum_{k=0}^{n-1} A(i,k) B(k,j)$$

Initially apply $A_{i,k}$ to $P_{i,0,k}$

and apply $B_{k,j}$ to $P_{0,j,k}$.

$\forall i,k$, $P_{i,0,k}$ broadcasts $A_{i,k}$ to all $P_{i,j,k}$. $\leftarrow t = O(\log n)$ time steps

$\forall k,j$, $P_{0,j,k}$ broadcasts $B_{k,j}$ to all $P_{i,j,k} \leftarrow t = O(\log n)$ time steps.

$C_{i,j,k}$

$\forall i,j,k$, $P_{i,j,k}$ computes $C'_{i,j,0} = A_{i,k} \times B_{k,j}$ in parallel. $\leftarrow$ time = 1

$\forall i,j$, $P_{i,j,k}$, $k = 0,1,\cdots,2^t - 1$, add $C'_{i,j,k}$ to $C(i,j)$ using the addition algo. $\leftarrow t = O(\log n)$ steps.

$C(i,j)$ are finally stored in $P_{i,j,0}$.

Running time $= O(\log n)$

DAG → too simple, no communication, scheduling to be done separately

Network model: Algos vary widely from topology to topology

PRAM :- synchronized shared memory model.

Performance measures

A → a parallel algo for solving Q.

(1) $T(n)$ time on $P(n)$ processors.

(2) $T(n)$ time, cost $= T(n)P(n)$.

A sequential algo on one processor runs in $O(T(n)P(n))$ time.

(3) $p \leq P(n)$ processors.

$O\left(\frac{T(n) \cdot P(n)}{p}\right)$.

(4) $p > P(n)$ processors are available.

Use only $P(n)$ processors and run A in $T(n)$ time.

$$O\left(\frac{C(n)}{p} + T(n)\right).$$

Date: 28th July, 2017

A - parallel algo using $P(n)$ processors and taking $T(n)$ time.

$c(n) = T(n)\,P(n)$.

↳ includes the idle time of processors.

n-fold sum $A(1) + A(2) + \cdots + A(n)$, $n = 2^t$.

Code for P_i, $i = 1, 2, \ldots, n$

Copy $A(i)$ to $B(i)$.

For ~~each~~ $h = 1, 2, \ldots, t$ {

if $(i \le n/2^h)$ set $B(i) = B(2i-1) + B(2i)$

}

if $(i = 1)$ copy $B(1)$ to S.

Work - time (WT) presentation.

for $1 \le i \le n$ pardo $B(i) = A(i)$

for $h = 1, 2, \ldots, t$ {

for $1 \le i \le \frac{n}{2^h}$ pardo $B(i) = B(2i-1) + B(2i)$

}

if $(i-1)$ set $S = B(1)$

Schedule this work to p processors.

$W(n)$ = work.

= the no. of operations.

$$= n + \left(\sum_{h=1}^{\log n} \frac{n}{2^h} \right) + 1$$

$$\le n + n + 1$$

$$= 2n + 1$$

$$= O(n).$$

$c(n) = n \log n = O(n \log n)$.

for $1 \le i \le n$ pardo something$_1$ // this gets done first
// all other processors wait

for $1 \le i \le n$ pardo something$_2$.

The WT principle

A → a parallel algorithm
with work $W(n)$ and running time $T(n)$.

Then A can be scheduled to p processors to run in

$$T_p(n) = O\left(\frac{W(n)}{p} + T(n)\right) \text{ time.}$$

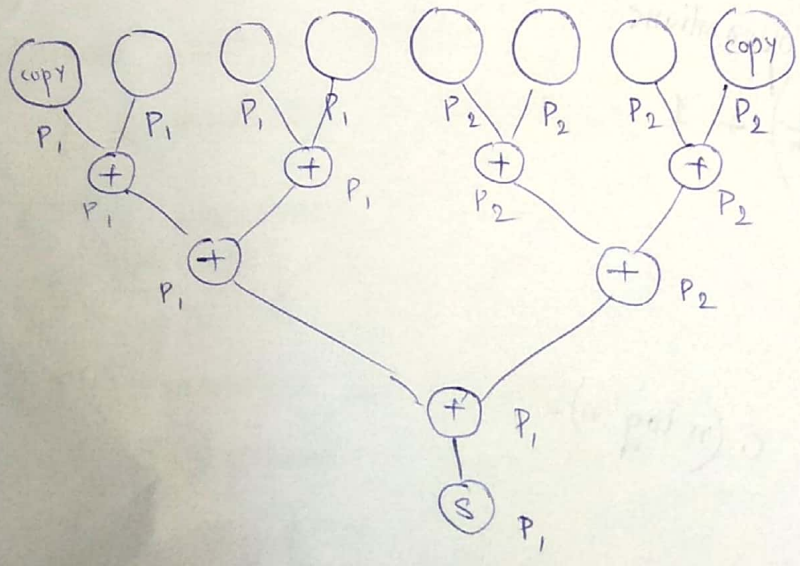
	work
Step 1	$W_1(n)$
Step 2	$W_2(n)$
step i	$W_i(n)$
Step $T(n)$	$W_{T(n)}(n)$

Equitable sharing to p processors take $\left\lceil \frac{W_i(n)}{p} \right\rceil$ time.

$$T_p(n) = \sum_{i=1}^{T(n)} \left\lceil \frac{W_i(n)}{p} \right\rceil \le \sum_{i=1}^{T(n)} \left(\frac{W_i(n)}{p} + 1\right)$$

$$= \frac{W(n)}{p} + T(n)$$

$n = 8, t = 3, p = 2.$



$T(n) = \log n$
$W(n) = O(n)$

Work-time representation (WT)

$W(n)$ = total work done

$T(n)$ = the running time

WT scheduling lemma

$$T_p(n) = O\left(\frac{W(n)}{p} + T(n)\right)$$

$$C_p(n) = pT_p(n) = O(W(n) + pT(n))$$

(cost)

If $p = O\left(\frac{W(n)}{T(n)}\right)$, then $C_p(n) = O(W(n))$.

Parallel sum

If $p = O\left(\frac{n}{\log n}\right)$, then $C_p(n) = O(W(n))$.

Optimality:

Q → a problem whose optimal sequential running time is $T^*(n)$.

A is a parallel algorithm to solve Q presented in the WT form.

A is called optimal ~~bu~~ (weakly) if $W(n) = \Theta(T^*(n))$

p processors

$$T_p(n) \leq O\left(\frac{T^*(n)}{p} + T(n)\right).$$

$$S_p(n) = \frac{T^*(n)}{T_p(n)} = \Omega\left(\frac{T^*(n)}{\frac{T^*(n)}{p} + T(n)}\right)$$

$$= \Omega\left(\frac{pT^*(n)}{T^*(n) + pT(n)}\right).$$

If $p = O\left(\frac{T^*(n)}{T(n)}\right)$, then $S_p(n) = \Theta(p)$.

A is called strongly optimal or WT-optimal if-

i. A is weakly optimal

ii. No optimal (weakly) algorithm has running time $o(T(n))$

Example:

Parallel sum (binary tree algo)

$T(n) = \log n$

On a CREW PRAM, this running time cannot be improved.

Communication Complexity

$C = AB$ $n \times n$ matrices

For all i, j, k, pardo

compute $c'(i,k,j) = A(i,k) * B(k,j)$

For all i, j pardo {

for $h = 1, 2, \ldots, \log n$ {

~~set~~ For $k = 1, 2, \ldots, n/2^h$ pardo

set $c'(i,k,j) = c(i, 2k-1, j) + c'(i, 2k, j)$

}

}

For all i, j pardo. copy $c'(i,1,j)$ to $c(i,j)$

decides complexity

$O(\log n)$

$T(n) = \log n$

$W(n) = \Theta(n^3)$

$p = n^2$ processors are available

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

$= O(n)$

by the WT lemma.

Step 1: P_i computes $C'(i,k,j)$ $\forall$ k,j.

↳ requires communication of

i) i^{th} row of A (n) } $n+n^2$

ii) entire matrix B (n^2)

iii) $C'(i,k,j)$ $\forall$ k,j (n^2) } n^2

$O(n^2)$ running time

$O(n^2)$ communication.

Step 2: P_i computes $C(i,j)$ $\forall$ j.

↳ requires $C'(i,k,j)$ $\forall$ k,j

$O(n^2)$ running time

$O(n^2)$ communication.

Step 3: P_i copies $C'(i,1,j)$ to $C(i,j)$

$O(n)$ time

$O(n)$ communication.

Matrix mul with $o(n^2)$ communication complexity

$n = \alpha^3$

$C = AB$

$n \times n$

n processors

$$A = \begin{bmatrix} A_{11} & A_{12} & \cdots & A_{1\alpha} \\ A_{21} & A_{22} & \cdots & A_{2\alpha} \\ \vdots & \vdots & & \vdots \\ A_{\alpha 1} & A_{\alpha 2} & \cdots & A_{\alpha\alpha} \end{bmatrix}$$

$B \to B_{ij}$

$C \to C_{ij}$

$A_{ij} \to \alpha^2 \times \alpha^2$ block.

$$C_{ij} = \sum_{k=1}^{\alpha} A_{ik} B_{kj}$$

Block-level multiplication.

We have $n = \alpha^3$ block-level multiplications (α values of i, α values of j, α values of k)

(1) Each processor will compute one block mul.

(2) α processors parallelly computes C_{ij} for each i, j.

→ Communication

i) Receive A_{ik} and
$O(\alpha^4) = O(n^{4/3})$

ii) α^4 α-way addition
$O(\alpha^4 \log \alpha) = O(n^{4/3} \log$

(3) Copy C' to C.

Running time

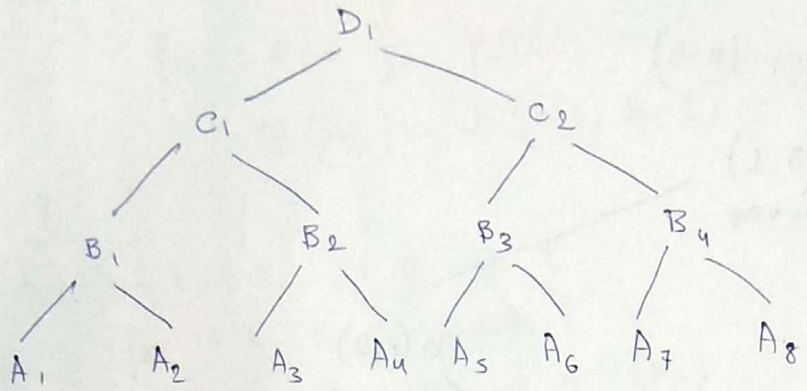
i) $O((\alpha^2)^3) = O(n^2)$

ii) $O(\alpha^4 \log \alpha)$
$= O(n^2)$

Lesson: The 10T-level presentation gives no hint on how to optimize communication complexity.

Some generic methods

1. Balanced Trees



$A = (a_1\ a_2 \cdots a_n)$ $\quad n = 2^t$

$S_i = a_1 + a_2 + \cdots + a_i$, $\quad 1 \le i \le n$.

Goal: To compute all of $S_1, S_2, \ldots, S_n$.

Recursive parallel algo

if $(n=1)$ { set $s_1 = a_1$, return }

for $i=1$ to $n/2$ pardo

set $b_i = a_{2i-1} + a_{2i}$

Recursively compute the ~~partial~~ prefix sums of $b_1, b_2, \ldots, b_{n/2}$.

and store these in $\gamma_1, \gamma_2, \ldots, \gamma_{n/2}$.

for $i=1$ to n pardo {

if $i=1$, set $s_1 = a_1$

else if i is even set $s_i = \gamma_{i/2}$.

else set $s_i = \gamma_{(i-1)/2} + a_i$

}

$n=8$

$b_1 = a_1 + a_2 \quad b_2 = a_3 + a_4 \quad b_3 = a_5 + a_6 \quad b_4 = a_7 + a_8$

$\gamma_1 = a_1 + a_2$

$\gamma_2 = a_1 + a_2 + a_3 + a_4$

$\gamma_3 = a_1 + a_2 + a_3 + a_4 + a_5 + a_6$

$\gamma_4 = a_1 + a_2 + a_3 + a_4 + a_5 + a_6 + a_7 + a_8$

$s_1 = a_1$

$s_2 = \gamma_1 = a_1 + a_2$

$s_3 = \gamma_1 + a_3 = a_1 + a_2 + a_3$

$s_4 = \gamma_2 = a_1 + a_2 + a_3 + a_4$

$s_5 = \gamma_2 + a_5$

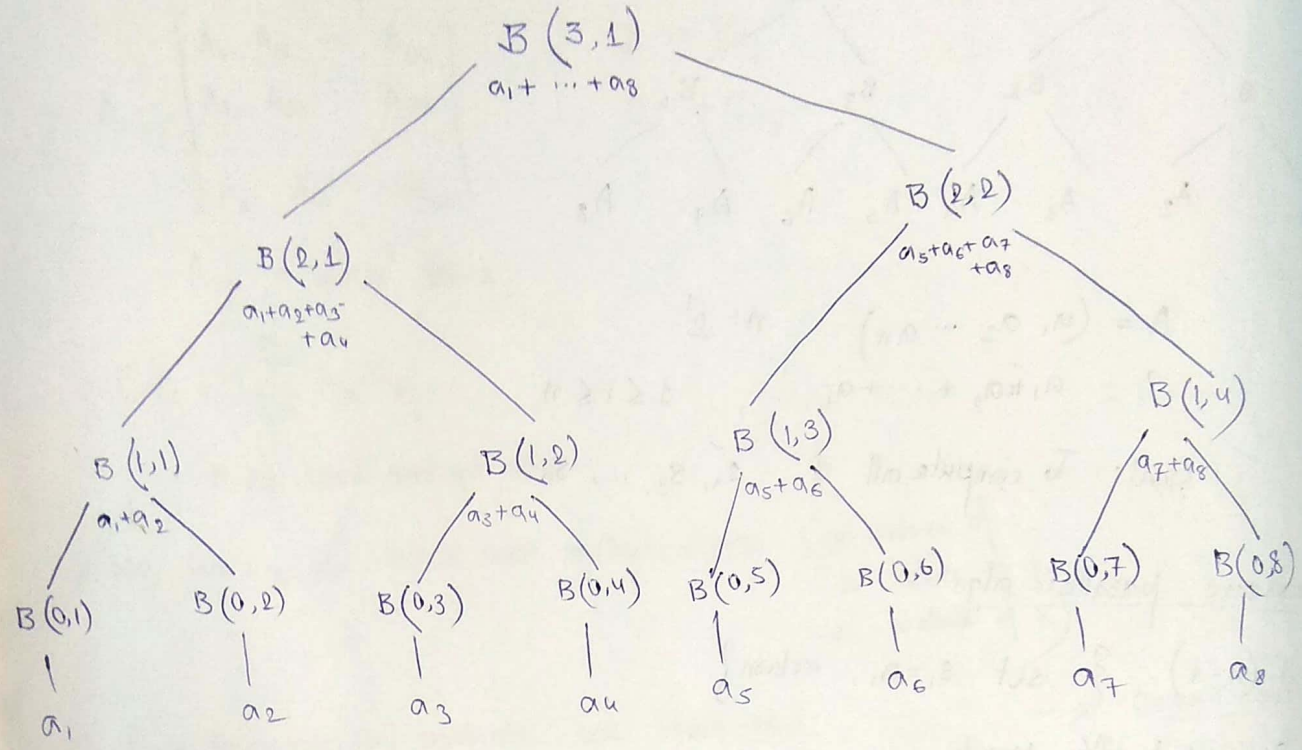
$s_6 = \gamma_3$

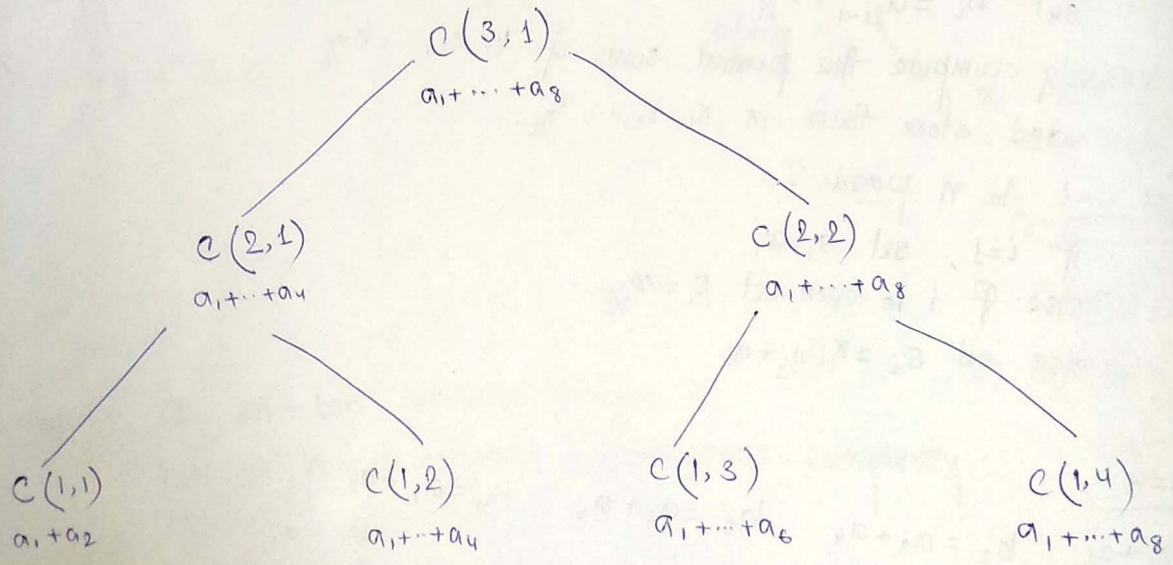
$s_7 = \gamma_3 + a_7$

$s_8 = \gamma_4$

$$W(n) = W\left(\frac{n}{2}\right) + \alpha n \quad \Big\} \quad W(n) = \Theta(n)$$
$$T(n) = T\left(\frac{n}{2}\right) + \beta \quad \Big\} \quad T(n) = \Theta(\log n)$$

Replacing recursion by iteration ($n=8$)





$n = 2^t$

for $1 \le i \le n$ pardo $B(0,i) = a_i$

for $h = 1, 2, \ldots, t$ do {

for $1 \le i \le \frac{n}{2^h}$ pardo

$B(h,i) := B(h-1, 2i-1) + B(h-1, 2i)$

}

for $h = t, t-1, \ldots, 1, 0$ do {

for $1 \le i \le \frac{n}{2^h}$ pardo {

if $(i=1)$ $C(h,i) := B(h,i)$

else if i is even $C(h,i) := C(h+1, i/2)$

else $C(h,i) := C(h+1, \frac{(i-1)}{2}) + B(h,i)$.

}

}

for $1 \le i \le n$ pardo $s_i := C(0,i)$

$B(\cdot,\cdot)$, $C(\cdot,\cdot)$ > simulates the recursion stack.

$T(n) = O(\log n)$

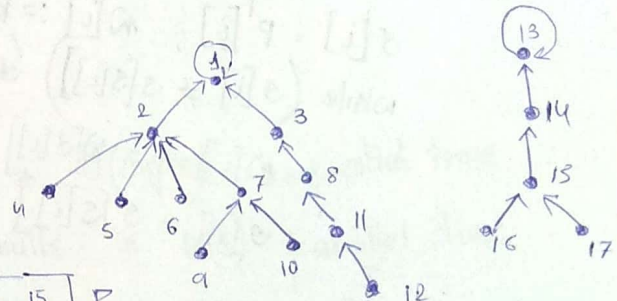
$W(n) = O(n)$

CREW PRAM

WT optimal

Tool 2 : Pointer jumping

parent presentation



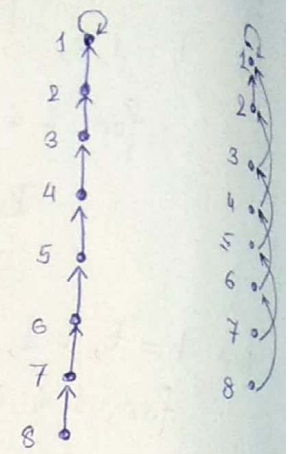
1	1	1	2	2	2	3	7	7	8	11	13	13	14	15	15	P
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17

1	1	...	1	13	13	...	13	S

n – the number of nodes in the forest

nodes are 1, 2, 3, ..., n.

S links



Init

Iter 1

for $1 \le i \le n$ pardo {

$S[i] = P[i]$

while $(S[i] \neq S[S[i]])$ do

$S[i] = S[S[i]]$.

}

$T(n) = O(\log h)$ [1 for initialization, $\log(\text{height}(F))$ for rest]. So $1 + \log h$]

$W(n) = O(n \log h)$

Iter 2

Key(i)

$W[i]$ → the sum of all key values on the unique path from i to its root.

for $1 \le i \le n$ pardo {

$S[i] = P[i]$, $W[i] := key[i]$

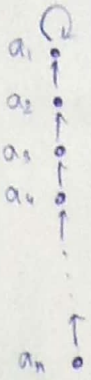
while $(S[i] \neq S[S[i]])$ do {

$W[i] += W[S[i]]$

$S[i] = S[S[i]]$

}

}



Prefix sum can be solved using this method

$O(\log n)$

$n \log n)$

3. Divide and Conquer.
4. Partitioning.

Convex hull

Merging two sorted arrays.

optimal sequential time is $O(n \log n)$ in the worst case.

DIVIDE & CONQUER

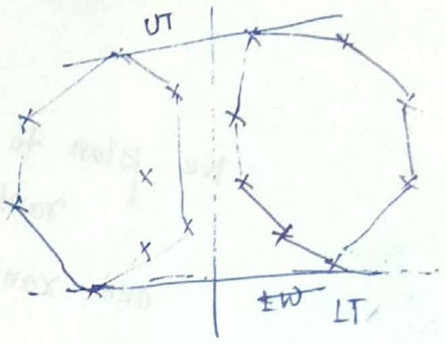
Step 1: Sort the points in S w.r.t their x-coordinates.

$O(\log n)$ parallel time.

Step 2: Divide in two halves L and R.

$O(1)$ seq. time.

Step 3: Compute the $LH = CH(L)$, $RH = CH(R)$ in parallel.



Step 4: Merge LH and RH to CH(S)

UT (upper tangent) / LT can be computed in $O(\log n)$ sequential time.

Step 5: Merge the upper and lower hulls in $O(1)$ parallel time.

$T(n) = T\left(\frac{n}{2}\right) + O(\log n)$ [From ③ and ④]

$W(n) = 2 \cdot W\left(\frac{n}{2}\right) + O(n)$ [$O(n)$ from ⑤]

$T(n) = O(\log^2 n)$

$W(n) = O(n \log n)$

CREW (PRAM type)

EREW might work as well!

PARTITIONING

$A = [a_1, a_2 \cdots a_n]$ (m)

$B = [b_1, b_2 \cdots b_n]$

No duplicates in $A \cup B$.

Output : $C = [c_1, c_2 \cdots c_{2n}]$ $(m+n)$

Can be solved in $O(n)$ (or $O(m+n)$) sequential time.

How to parallelize?

~~rank~~ (

Let X, Y be two sets (You may have $X = Y$)

$x \in X$

rank $(x : Y)$ = the number of elements of Y that are $\leq x$.

We plan to compute

rank $(a_i, A \cup B)$

and rank $(b_j ; A \cup B)$

$\forall\ i, j$

rank $(x : A \cup B)$

= rank $(x : A)$ + rank $(x : B)$

It suffices to compute rank $(A : B)$ and rank $(B : A)$

Look at a parallel algo to compute rank (A, B).

For all $a \in A$ pardo
find rank $(a : B)$ by binary search.

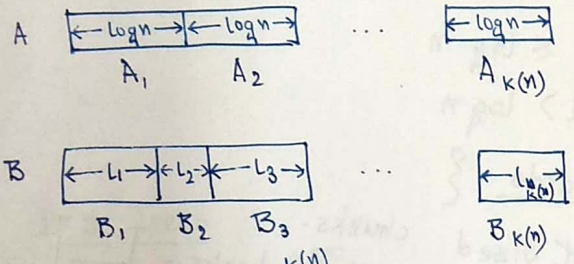
CREW PRAM.

$O(\log n)$ time but $O(n \log n)$ work
↑
not optimal.

$\log n$, $k(n) = \frac{n}{\log n}$ are all integers

~~rank $(A : B)$~~

Each element of $A_i \cup B_i$ is larger than each element of $A_{i-1} \cup B_{i-1}$



$|A_i| = \log n$, $\sum_{i=1}^{k(n)} |B_i| = n$.

Concatenate
merge (A_i, B_i)
for $i = 1, 2, \ldots, n$ to get C.
↳ can be done in $O(1)$ parallel time.
using $O(n)$ work.

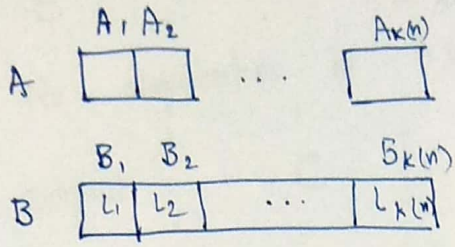
Partitioning of B:

Binary search for $a_{i \log(n)}$
in parallel for $i = 1, 2, \ldots, k(n)$

4. Partitioning

Merge two sorted arrays ~~each of size~~ A and B each of size n.

$$k(n) = \left\lceil \frac{n}{\log n} \right\rceil$$



CREW PRAM

$O(\log n)$ parallel running time

~~Dot~~ work done $= O(k(n) \log n) = O(n)$.

$B_i \rightarrow$ small block if $l_i \leq \log n$
Large block if $l_i > \log n$

for each large block B_i pardo {
Break B_i in $\log n$ - ~~st~~ sized chunks. → # of chunks $= \left\lceil \frac{l_i}{\log n} \right\rceil$

Partition A_i in parallel
}

Work $= \sum_{B_i \text{ is a large block}} \left\lceil \frac{l_i}{\log n} \right\rceil \log(\log n)$. $= \sum_{B_i \text{ large}} \left\lceil \frac{l_i}{\log n} \right\rceil - 1$ ← Increase in the number of subproblems

$\leq \frac{(\sum l_i) \log \log n}{\log n} = \frac{n \log \log n}{\log n} = O(n)$.

$\leq \sum_{B_i \text{ large}} \frac{l_i}{\log n} \leq \sum_{i=1}^{k(n)} \frac{l_i}{\log n} = \frac{n}{\log n} = k(n)$

for $i = 1, 2, \ldots, k$ pardo

merge(A_i, B_i) using the sequential linear tree algo

$|A_i| \leq \log n$ $\quad$ $O(\log n)$ running time

$|B_i| \leq \log n$ $\quad$ work done $= O(k \log n)$

$= O(n)$ (since $k \leq 2k(n)$)

5. Pipelining.

T_1 $\quad$ $t_{11}\ t_{12} \cdots t_{1n}$

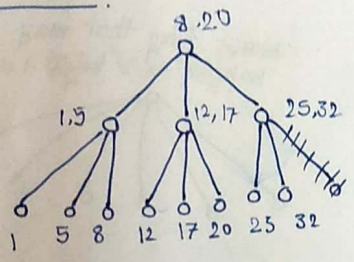
T_2 $\quad$ $t_{21}\ t_{22} \cdots t_{2n-1}\ t_{2n}$

$\vdots$

T_k

Instead of ~~$k(h)$~~ $O(kh)$ running time, we have $O(k+h)$ running time.

2-3 Trees.



$v \rightarrow$ internal node

$L[v] \rightarrow$ largest key stored in the first subtree.

$M[v] \rightarrow$ Largest key stored in the second subtree.

$n \rightarrow$ no. of keys stored in T

$2^h \leq n \leq 3^h$, $\quad h =$ height (T)

$h = O(\log n)$

search (x, T) $\quad$ → If T is a leaf, return whether $x = key[T]$.

if $(T = NULL)$ return false

if $(x \leq L[T])$ return search $(x, T \rightarrow First)$

if $(x \leq M[T])$ return search $(x, T \rightarrow second)$

return search $(x, T \rightarrow third)$

runtime – $O(\log n)$

$T \rightarrow$ a 2-3 tree with $n-2$ leaves

To insert $k+2$ elements

$$b_0 < b_1 < b_2 < \cdots < b_k < b_{k+1}$$

Sequential run time $= O(k \log n)$

insert b_0 in T
insert b_{k+1} in T
$\Big] \; O(\log n)$

The keys in T are $a_1, a_2, \ldots, a_n$

$\forall\, j = 1, 2, \ldots, k$

$\exists\, i \in 1, 2, \ldots, n-1$

s.t. $a_i < b_j < a_{i+1}$

for $i = 1, 2, \ldots, n-1$

define

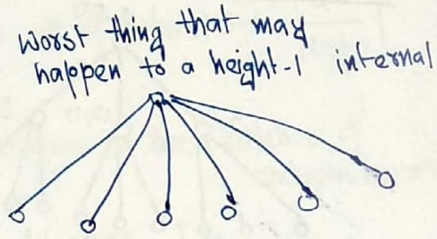
$$B_i = \{ j \mid a_i < b_j < a_{i+1} \}$$

$$\sum_{i=1}^{n-1} |B_i| = k.$$

Let $l_i = |B_i|$

Assumption. each $l_i \leq 1$

$$\boxed{k \ll n}$$



First insert all b_j values in parallel.

General case

Allow $|B_i| \geq 2$

$B_1, B_2, \cdots, B_{n-1}$

Use simple insertion procedure to insert the medians of all non-empty B_i.

$B_i = \{j, j+1, j+2, \cdots, j+k-1\}$

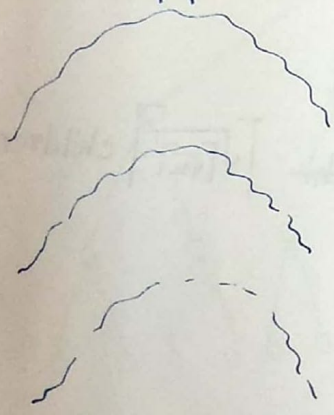
↳ broken into two subchains of size $k/2$.

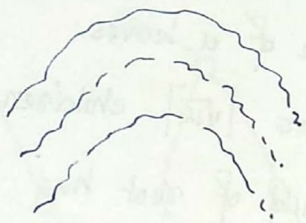
After at most $\lceil \log k \rceil$ simple insertion procedures, all b_j will be inserted.

↳ each called a wave.

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

Waves can be pipelined.





runtime $= O(\log k + \log n)$
$\simeq O(\log n)$

6. Accelerated Cascading.

A → array of n elements. (distinct)

To ~~ea~~ compute max A.

Common CRCW (if all threads try writing the same value, write succeeds) otherwise not.

CREW
balanced tree algorithm
$O(\log n)$ time
$O(n)$ work
↳ optimal for a CREW PRAM

A constant-time non-optimal algo

for all $i,j \in \{1,2,\dots,n\}$ pardo

if $a_i \geq a_j$ set $B(i,j) = 1$

else set $B(i,j) = 0$

for all $i \in \{1,\dots,n\}$ pardo $C(i) = 0$

for all $i \in \{1,\dots,n\}$ set $C(i) = B(i,1) \wedge B(i,2) \dots \wedge B(i,n)$ ← CW

for all $i \in \{1,\dots,n\}$ if $C(i) = 1$, set $max = A(i)$

Constant time ↗

$O(1)$ time

$O(n^2)$ work

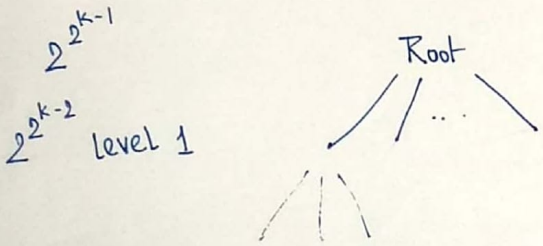
A doubly logarithmic time algo :

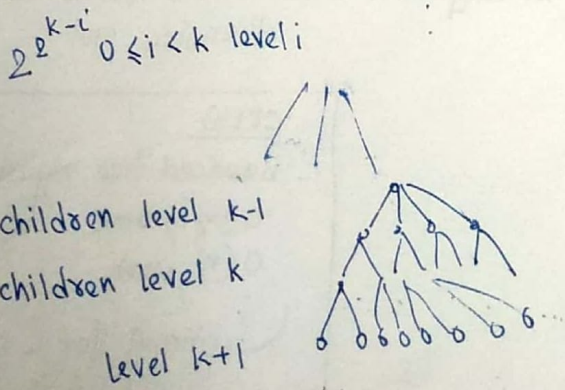
tree of u leaves.

Root has $\lceil \sqrt{u} \rceil$ children.

Each child of root has ~~$\lceil \sqrt{u} \rceil$ chi~~ $\lceil \sqrt{\lceil \sqrt{u} \rceil} \rceil$ children.

$n = 2^{2^k}$.





16.08.2017

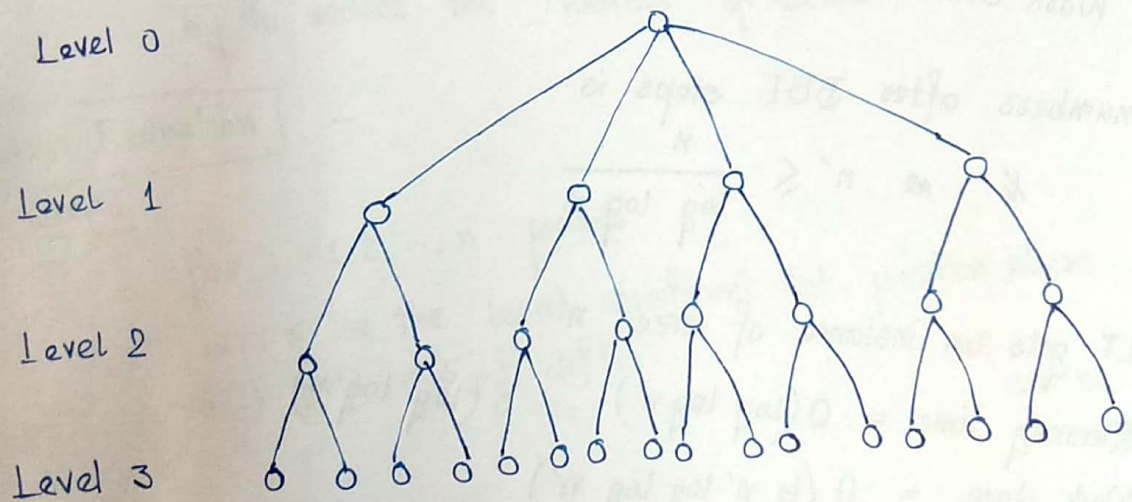
Finding maximum of $A = (a_1, a_2, \cdots, a_n)$

$O(1)$ time
$O(n^2)$ of work } algo for CREW PRAM.

Doubly logarithmic height tree.

$n = 2^{2^k}$ for some $k \geq 1$.

$k = 2$: $n = 16$.



Parallel running time = $O(\log \log n)$ [2 here].

Level i

# of nodes = $2^{2^k - 2^{k-i}}$

# of children of each node = $2^{2^{k-i-1}}$

Total work done at level i

$$= O\left(2^{2^k - 2^{k-i}} \times \left(2^{2^{k-i-1}}\right)^2\right) = O\left(2^{2^k}\right) = O(n)$$

Total work done (at all levels) $= O(nk) = O(n \log \log n)$

still not optimal.

Balanced binary

CREW PRAM ← optimal for CREW and EREW PRAMs.
EREW

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

6. Accelerated cascading.

Phase 1: Run an optimal algo until the problem instances reduce in size.

Phase 2: Run a non-optimal algo on the small instances.

Algo 1: BBT Algo 2: DLT

Phase 1: Run BBT for $\lceil \log \log \log n \rceil$ levels

Running time $= O(\log \log \log n)$

Work done $= O(n)$

# of numbers after BBT steps is

$n' \leq \frac{n}{\log \log n}$

Phase 2: DLT gets an instance of size n'.

Running time $= O(\log \log n') = O(\log \log n)$

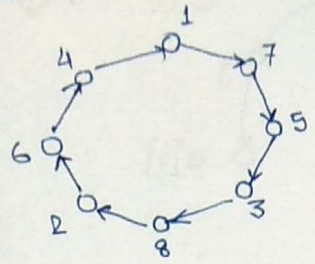
Work done $= O(n' \log \log n')$

$= O\left(\frac{n}{\log \log n} \log \log n'\right)$

$= O(n)$

7. Symmetry breaking.

Digraph G → a cycle



Edges stored in an array $S[1, \ldots, n]$ $[S[1]=7, S[2]=6, S[3]=8, S[4]=$

$S(i)$ is the ~~ss~~ successor of node i in the cycle

3. colouring always possible.

Optimal sequential algo : $O(n)$ time.

Start with $C(i) = i$

n colours for n vertices

Try to reduce the number of colours.

Colour Reduction –

Step 1 :

For $i = 1, 2, \ldots, n$ pardo {

Let k be the least significant bit position where $c(i)$ and $c(S[i])$ differ.

↑ can be done in $O(1)$ time (Later).

Give the new colour

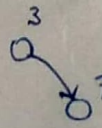
$c'(i) = 2k + (c(i))_k$ to node i.

$[c(i)_k \rightarrow k^{th}$ LSB of $c(i)]$

}

$O(1)$ parallel time

$O(n)$ work.



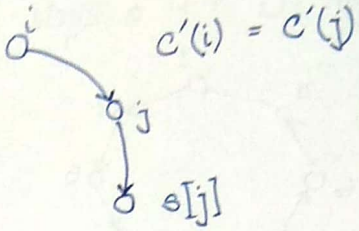
$c(3) = 3 = 011$

$c(S[3]) = c(7) = 7 = 111$

$k = 2$

$c'(3) = 2 \times 2 + 0 = 4$

Claim: c' is a proper colouring of the cycle.



Proof: Suppose not.

$$c'(i) = 2k + (c(i))_k$$
$$c'(j) = 2l + (c(j))_l$$
$$\| \quad \| \quad \|$$
$$\alpha \quad \alpha \quad \alpha$$

We must have

$k = l = \alpha$ quot 2

$c(i)_k = c(j)_l = \alpha$ rem 2

$\Big\}$ $c(i)_k = c(j)_k$.

a contradiction since $c(i)$ and $c(j)$ differ in the k^{th} bit position.

Edge Detection.

Si

Colouring of a cycle

Yesterday's algo:

$C \rightarrow q$ colours

How many colours in C'?

$$2^{t-1} \leq q < 2^t$$

$$C'(i) = 2k + (c(i))_k$$

$$\leq 2(t-1) + 1$$

$$= 2t - 1$$

$$q' \leq 2t$$

$$\boxed{q' \leq 2(\log q + 1)}$$

There is a decrease for $t > 3$.

A better algo:

Keep on calling basic colouring algo until $t \leq 3$.

Run time $= O(\log^* n)$

Work done $= O(n \log^* n)$

$$\log^{(1)} x = \log x$$

$$\log^{(i)} x = \log(\log^{(i-1)} x)$$

for $i \geq 2$.

$$\log^* x = \min\{i \mid \log^{(i)} x \leq 1\}$$

$$x = 2^{65536}$$

$$\log^* x = 5$$

Handle $t = 3$

Six possible colours = 0, 1, 2, 3, 4, 5,

$O(1)$ time, $O(n)$ work:

for $j = 3, 4, 5$, sequentially do

  for $1 \leq i \leq n$ pardo

    if (colour(i) = j)

      set colour(i) = the smallest permissible colour in $\{0, 1, 2\}$

$S[i]$ → successor

for $1 \leq i \leq n$ pardo

  set $P[S[i]] = i$ → predecessor

// j: colour number

// i: node number

An optimal 3-colouring algo

1. Start with $c(i) = i-1$ $\forall\, i \in \{1, 2, \ldots, n\}$ (par) → $O(1)$ time, $O(n)$ work

2. Run base colour-reduction algo only once → $O(1)$ time, $O(n)$ work

/# of colours = $s = O(\log n)$/

3. Sort $(i, c'(i))$ w.r.t the second component → $O(\log n)$ time, $O(n)$ work [Exercise]

(Counting sort)

4. for $j = 3, 4, \ldots, s$, sequentially do
   for all nodes i of colour j pardo
   set colour (i) = the smallest permissible colour in $\{0, 1, 2\}$.

   → constant time.

   } $O(\log n)$ iterations.

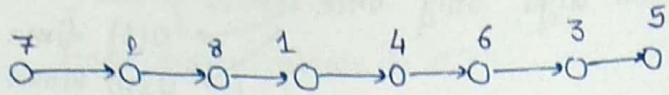
Total time = $O(\log n)$

Let n_j be the # of nodes of colour j.

Total work done = $O\left(\sum_{j=3}^{s} n_j\right) = O\left(\sum_{j=0}^{s} n_j\right) = O(n)$.

LISTS AND TREES

List Ranking



n nodes numbered $1, 2, \ldots, n$
provided as an array $S[1, \ldots, n]$
↑
Successor array.

S [Last node] $= 0$

Given S, one can compute the predecessor array $P[1, \ldots, n]$ in $O(1)$ time and with $O(n)$ work.

Assume S, P are both available.

P [first node] $= 0$

$R(i)$ = rank of i
= distance of i from the end of the list

Sequential algo:
$O(n)$ time
optimal

To compute $R(i)$ for $i = 1, \ldots, n$.

Algo 1 [Pointer Jumping]

1. for $1 \le i \le n$ pardo {
   if $S(i) \neq 0$, set $R(i) = 1$
   else set $R(i) = 0$
   }

→ $O(1)$ time, $O(n)$ work

2. for $1 \le i \le n$ pardo {
   set $Q(i) = S(i)$
   while $(Q(i) \neq 0)$ and $(Q(Q(i)) \neq 0)$ {
   update $R(i) += R(Q(i))$
   set $Q(i) = Q(Q(i))$
   }
   }

→ $O(\log n)$ iterations & running time
$O(n \log n)$ work

	7	2	8	1	4	6	3	5
Init	1	1	1	1	1	1	1	0
Iter 1	2	2	2	2	2	2	1	0
Iter 2	4	4	4	4	3	2	1	0
Iter 3	7	6	5	4	3	2	1	0

Preparation for Algorithm 2

1. Shrink the list to $\frac{n}{\log n}$ elements.
2. Run pointer jumping algo on the shrinked list.
3. Compute the ranks of the deleted elements.

HOW TO SHRINK?

7 - 2 - 8 - 1 - 4 - 6 - 3 - 5

$I \subseteq \{1, 2, \ldots, n\}$ is called an independent set. if $i \in I \Rightarrow s(i) \notin I$

Example: $I = \{2, 3, 4\}$

Do renumbering if needed.

Removal of an IS

1. Remember
2. Rank update
3. Link update.

$\forall\, i \in I$ pardo {

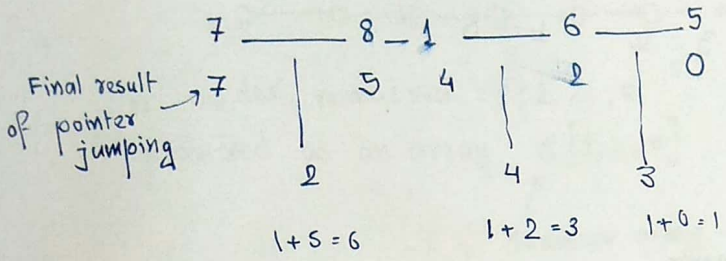
1. Store $(i, s(i), P(i))$
2. Update $R(P(i)) += R(i)$
3. Set $s(P(i)) = s(i)$
   and $P(s(i)) = P(i)$

}

7 - (2) - 8 - 1 - (4) - 6 - (3) - 5

1 1 1 1 1 1 1 0

Step 2 of removal → 2 1 2 2 0



How to find an IS?

Color the nodes with k colors.

i is called a local minima if

$$\text{colour}(i) < \text{colour}(S(i))$$

and $$\text{colour}(i) < \text{colour}(P(i))$$

Claim: The set of all local minima is an independent set of size $\Omega(n/k)$.

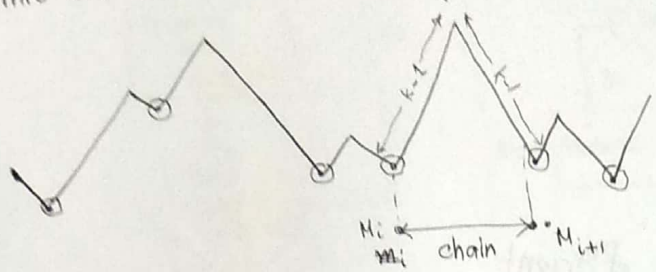
List Ranking (contd.)

k-coloring done

Set of all local minima is an IS. $\longrightarrow S$ $\quad |S| \geq \alpha n$ (α: a constant depending on k)

This set contains $\Omega(n/k)$ elements.

In particular, $k=3$, $|S| \geq \frac{1}{3}n$



Two consecutive local minima have a maximum distance of $2k-3$

Minimum possible number of chains $\simeq \frac{n}{2k-3} = \Omega\left(\frac{n}{k}\right)$

1. $n_0 = n$, $k = 0$.
   while $\left(n_k > \frac{n}{\log n}\right)$ {
   - 3-color the nodes — $O(n)$ work, $O(\log n)$ time
   - S = the set of the local minima — $O(n)$ work, $O(1)$ time
   - Remove S from the set of nodes. — ~~$O(n)$ work~~ $O(n)$ work, $O(1)$ time

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

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

   $++k$
   $n_k = n_{k-1} - |S|$
   }

2. Run the pointer jumping algo on the contracted list. — $O(n)$ work, $O(\log n)$ time.

3. Insert back all the removed nodes in the reverse sequence of deletion — $O(n)$ work, $O(\log \log n)$ ~~time~~ time

$n_{k+1} \leq \frac{n_k}{5/4}$ $\qquad \left(\frac{5}{4}\right)^k = \log n$

$n_k \leq \frac{n_0}{(5/4)^k}$ $\qquad k = \frac{\log \log n}{\log \frac{5}{4}}$

$$O(n_0 + n_1 + n_2 + \cdots + n_k)$$

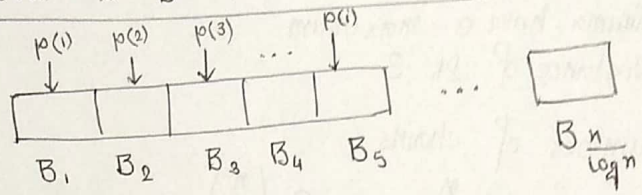
$$= O\left(n + \frac{4}{5}n + \left(\frac{4}{5}\right)^2 n + \cdots + \left(\frac{4}{5}\right)^k n\right)$$

$$= O\left(\frac{n}{1 - \frac{4}{5}}\right)$$

$$= O(n)$$

* A "forbidden" algorithm.

Removal to be made more efficient



$|B_i| = \log n$.

Init : $p(i)$ = index of the first element in B_i

$= (i-1)\log n + 1$

$N(p(i))$ → node at $p(i)$

Mark all $N(p(i))$ active

$\forall\ i = 1, 2, \cdots, \frac{n}{\log n}$

Mark all other nodes passive.

Iteration 1

isolated active node = an active node whose successor and predecessor are not active.

Remove all isolated active nodes.

Non-isolated nodes group in one or more sublists in the linked list.

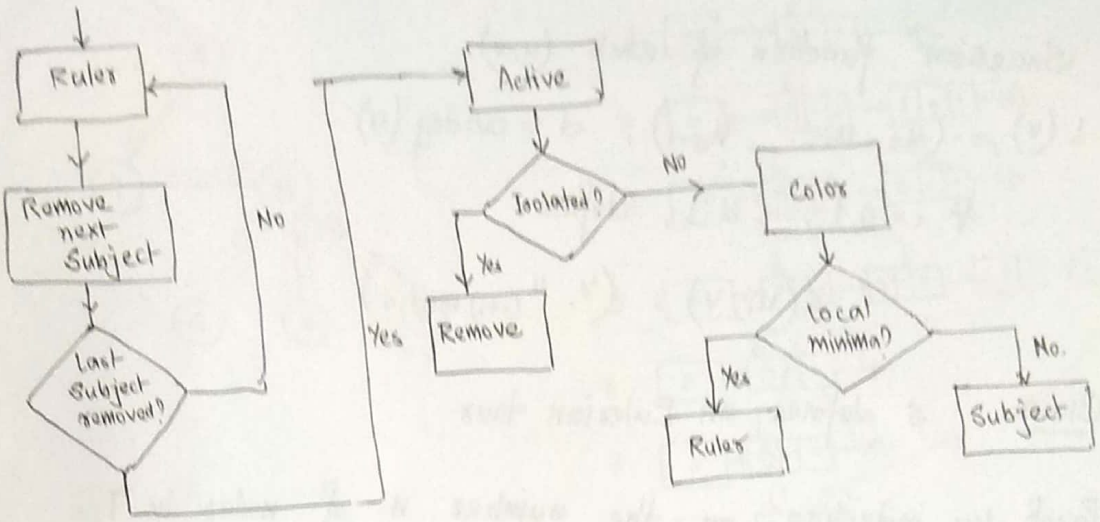
Run the basic color-reducing algo two times.

$O(\log \log n)$ colors.

Call each local minimum a ruler.

and other active nodes as subjects.

Schematic diagram of a later iteration (i^{th} iteration)



Algo 2

3 colouring $O(\log n)$ time

$$n_{k+1} \leq \frac{4}{5} n_k$$

$$= n_k - \frac{n_k}{5}$$

$\log \log n$ iterations.

Algo 3

basic color-reduction twice

$O(1)$

$$n_{k+1} \leq n_k - \frac{n}{\log n}$$

Analysis shows $O(\log n)$ iterations are necessary.

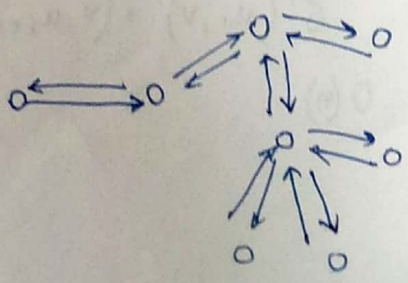
Euler tours in trees

(Directed)

Replace each edge $\{u, v\}$ by two directed edges.

(u, v) and (v, u)

~~Hon~~



How to construct an Eulerian tour in this graph?

Successor function of each (u, v)

$L(v) = (u_0, u_1, \ldots, u_{d-1})$, $d = \text{outdeg}(v)$

$\forall\ i = 0, 1, \ldots, d-1$, define

$$s(u_i, v) = (v, u_{(i+1) \bmod d})$$

Claim : s defines an Eulerian tour.

Proof by induction on the number n of nodes in T.

$n = 2$

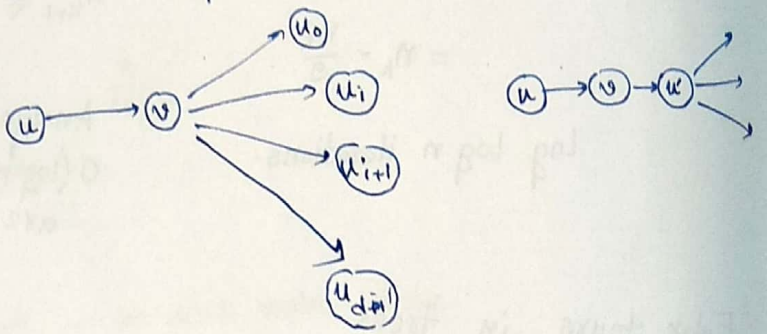
$s(u, v) = (v, u)$

$s(v, u) = (u, v)$

$\therefore$ True for the basis case

$n \geq 3$

Let u be any leaf of T

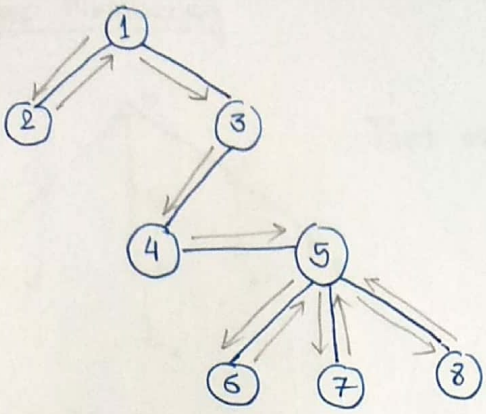


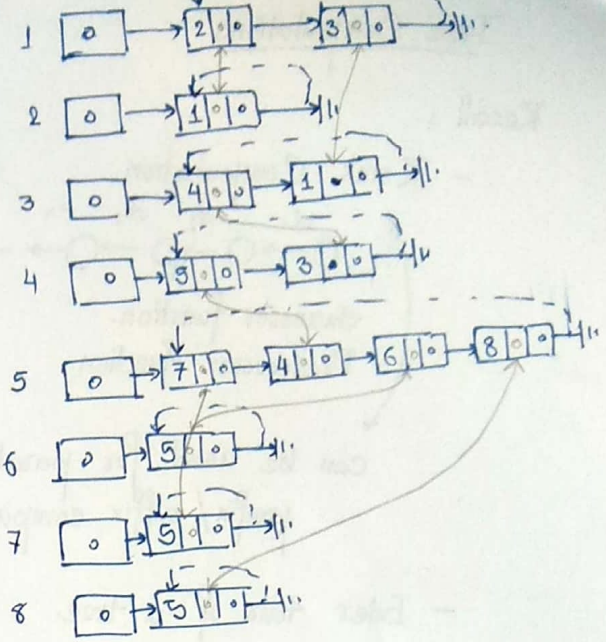
$L(v) = (u_0, u_1, \ldots, u_i, u, u_{i+1}, \cdots, u_{d_v-1})$

$L'(v) = (u_0, u_1, \ldots, u_i, u_{i+1}, \ldots, u_{d-1})$.

$$s'(u_i, v) = (v, u_{i+1})$$

Sequential complexity - $O(n)$





Two modifications :

(1) Make each list circular.

(2) $(u, v) \in E(T)$

$\Leftrightarrow$ add two edges between

u in $L[v]$

and v in $L[u]$

$1 \to 2 \to 1 \to 3 \to 4 \to 5$

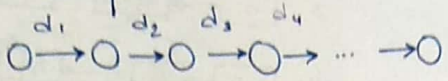
$\to 6 \to 5 \to 8 \to 5 \to 7 \to 5$.

S can be prepared in parallel in $O(1)$ time using $O(n)$ work.

Tree Computations

Recall :

- Rank Computation

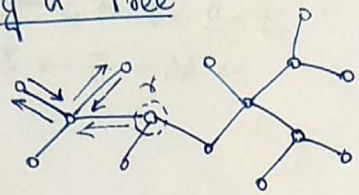


Successor function.
Predecessor function.

Can be used for parallel prefix / suffix computations.

- Euler tour of a tree.
  preparation of the successor table.

1. Rooting a Tree



→ Eulerian tour simulates DFS in T.

$s(u_d, r) = 0$.

An $r \in V$ is supplied as input.

Output : $p[V]$
such $\forall\, v \in V$, $v \neq r$
$p[v]$ stores the parent of v.

$L[r] = \langle u_1, u_2, \ldots, u_d \rangle$

Assign a weight
$w(x,y) = 1$
$\forall$ directed edge (x,y).

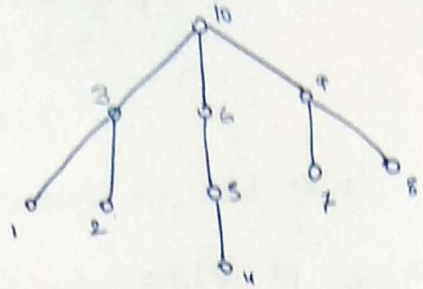
$O(\log n)$ time
$O(n)$ effort.

Do parallel prefix computation on these weights and guided by the successor function available from the ET.

After $O(\log n)$ time, we have an array $prefsum(x,y)$.

Set $p(y) = x \iff prefsum(x,y) < prefsum(y,x)$.

2. Post Order Numbering



Post order.

$w(x, p(x)) = 1$

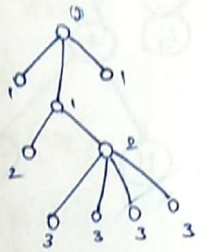
$w(p(x), x) = 0$

Compute prefix sums.

postorder (x) = prefsum $(x, p(x))$

postorder $(r) = n$

3. Level of Nodes



Level (x)

$w(p(x), x) = 1$

$w(x, p(x)) = -1$

Do prefix sum

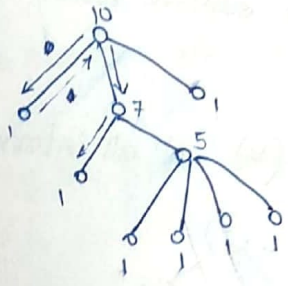
Level (x) = prefsum $(p(x), x)$

Level $(r) = 0$

4. Subtree Size

(# of descendants)

~~$w(p(x), x) =$~~



# of descendants
= subtree size
(at every node)

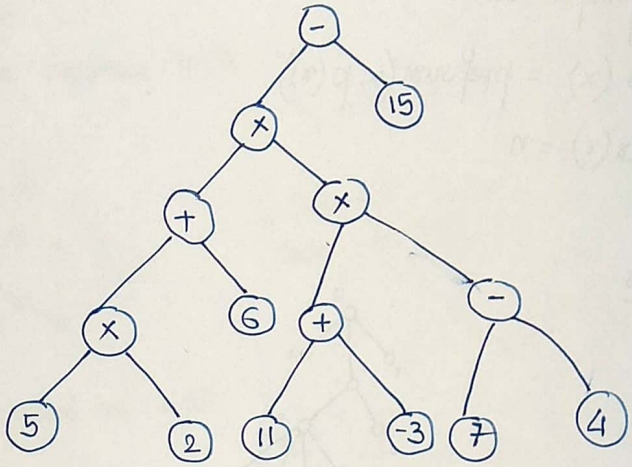
$w(x, p(x)) = 1$

$w(p(x), x) = 0$

$size(x) = prefixsum(x, p(x)) - prefixsum(p(x), x)$

5. Expression Evaluation

+, −, ×



Internal node : operator

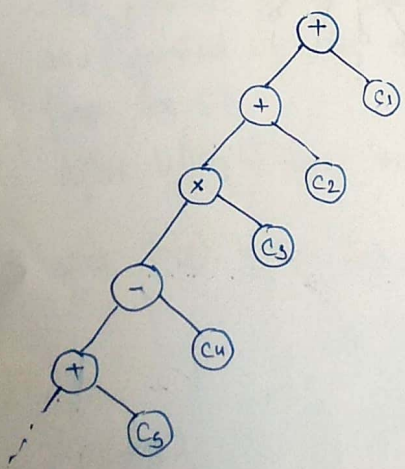
Leaf : a constant

Given such a tree of n nodes.

val(v) → value of the subexpression in the subtree rooted at v.

Goal : val(r)

or val(v) at all internal nodes v.



Sequential algo →
O(n) time
(O(n) effort)

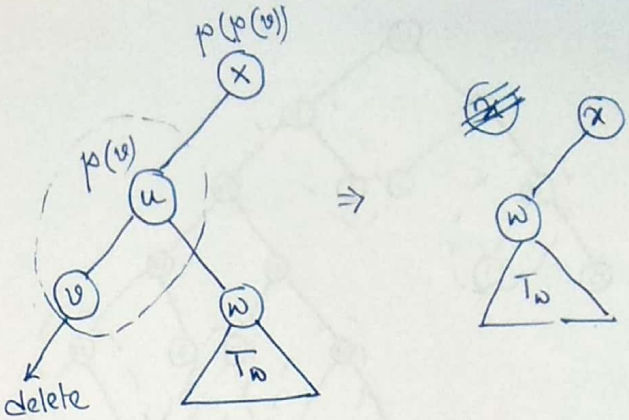
Raking

v is a leaf.

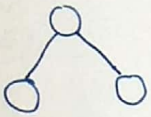
$p(v) \neq r$

$sib(v)$

- Remove v and $p(v)$
- Attach $sib(v)$ to $p(p(v))$



T - binary tree with every internal node having exactly two children.



Tree contraction

v_1 and v_2 are two leaves where raking is allowed.

Raking at these two nodes can be done in parallel if

(i) $p(v_1) \neq p(v_2)$

(ii) $p(p(v_1)) \neq p(v_2)$

(iii) $p(p(v_2)) \neq p(v_1)$

$A = (a_1, a_2, a_3, a_4, a_5, \ldots)$

$A_{odd} = (a_1, a_3, a_5 \ldots)$

$A_{even} = (a_2, a_4, a_6 \ldots)$

Use ET Technique to find all the leaves and list them from left to right.

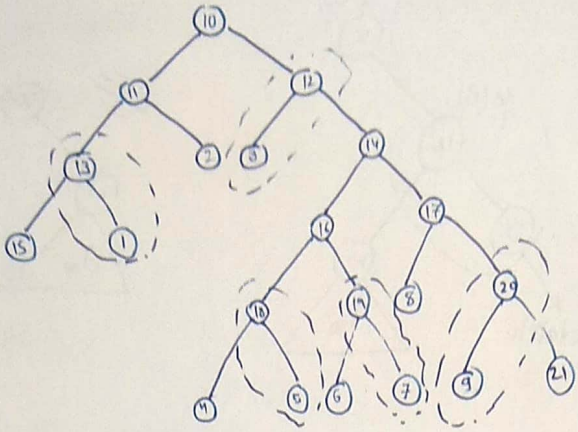
Let A be these leaves including the first and the last.

For $\lceil \log n \rceil$ iterations do {

Step 1: Apply raking in parallel on all leaves in A_{odd}, that are left children.

Step 2: Apply raking in parallel on the remaining elements in A_{odd}.

Step 3: Set $A = A_{even}$.



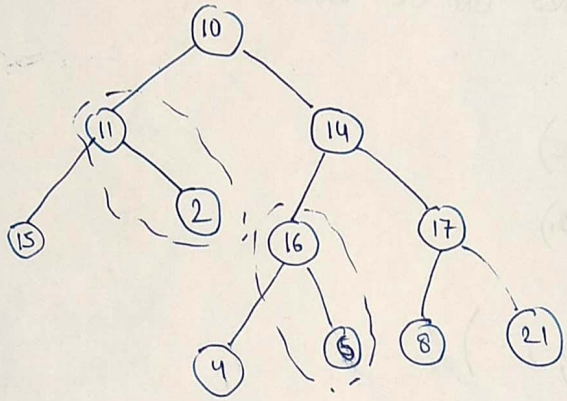
Iteration 1:

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

$A_{odd} = \{1, 3, 5, 7, 9\}$

Step 1: Rake at 3 and 9.

Step 2: Rake at 1, 5, 7.



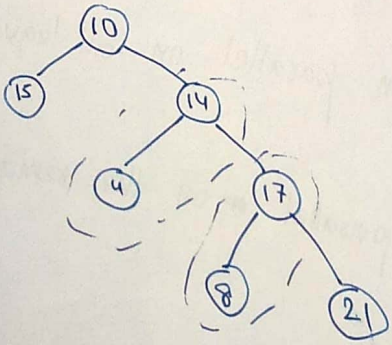
Iteration 2:

$A = \{2, 4, 6, 8\}$

$A_{odd} = \{2, 6\}$

Step 1: Nothing

Step 2: Rake at 2,6



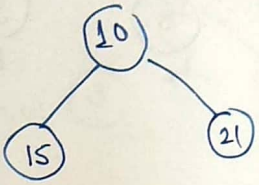
Iteration 3:

$A = \{4, 8\}$

$A_{odd} = \{4\}$

Step 1 : Rake 4 at 4

Step 2 : Rake at 8



How many iterations ?

$m \rightarrow$ # of internal nodes

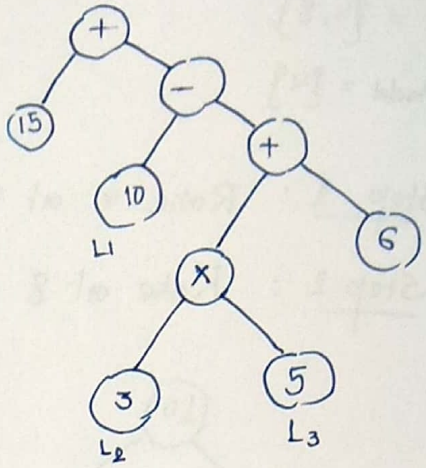
$m \rightarrow m/2 \rightarrow m/4 \rightarrow m/8 \rightarrow \dots$

After $\log m$ no. of iterations, no leaves will be left in A.

$m \leq n$.

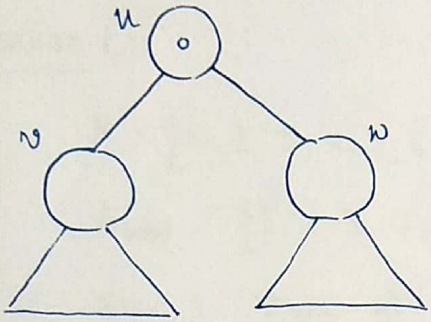
Evaluation of arithmetic expressions

+ , − , ×



At every node v, we maintain two numbers (a_v, b_v)

Invariance I

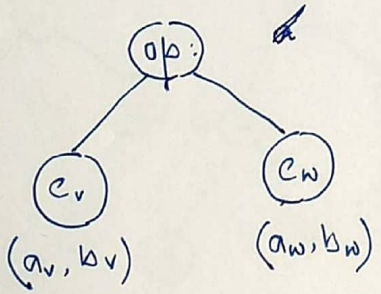


$$val(u) = (a_v\, val(v) + b_v) \circ (a_w\, val(w) + b_w)$$

Init: $(a_v, b_v) = (1, 0)$

$\forall$ nodes v.

Raking



$$a'_w \leftarrow val(w) + b'_w$$
$$= a_u\, val(u) + b_u$$
$$= a_u [\quad\quad] + b_u$$

$$val(r) = a_u\left[(a_v c_v + b_v) \cdot op\,(a_w c_w + b_w)\right] + b_u.$$

Contribution of u to its parent before raking

= contribution of w to its parent after raking.

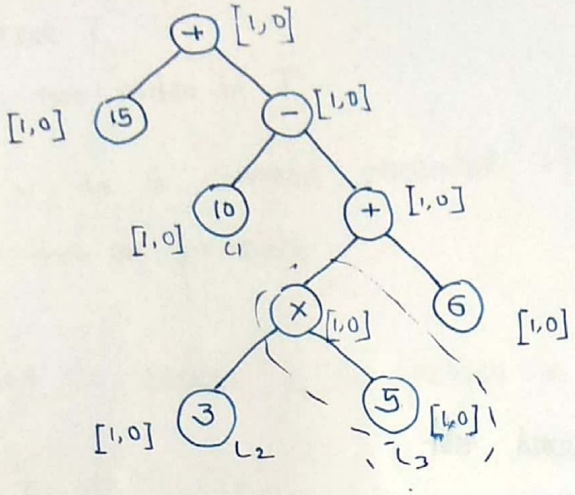
$= val(v)\ op\ val(w)$

$= (a_v c_v + b_v)\ op\ [a_w\, val(w) + b_w]$

(by invariance before raking)

$= a'_w\, val(w) + b'_w$

Example

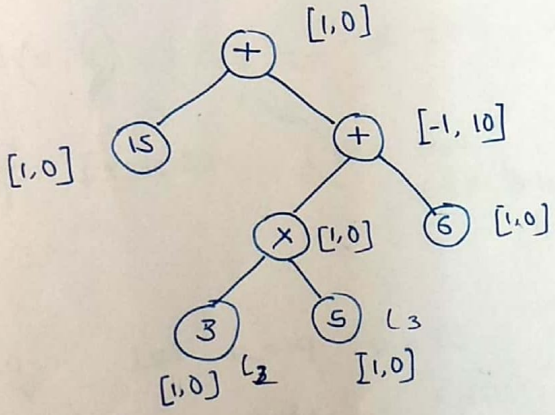


Rake at L1

$(1 \times 10 + 0) - (1 \times X + 0)$

$= 10 + (-1)\,X$

$1 \times (10 - X) + 0 = 10 + (-1)\,X$



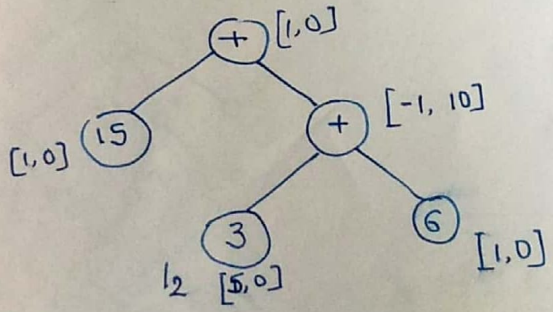
Rake at L3

$(1 \times X + 0) \times (5 \times 1 + 0)$

$= 5X + 0$

~~$-1(5X) + 10$~~

$=$

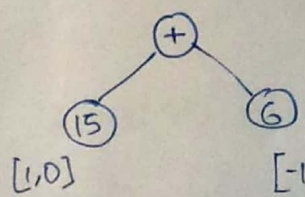


Rake at L2

$(-1) \cdot [(5 \times 3 + 0) + (X \times 1 + 0)] + 10$

~~$= +[21] + 10$~~ $= (-15 - X + 10)$

$= -X - 5$

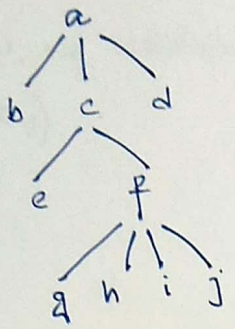


$val(v) = (1 \times 15 + 0) + (-1 \times 6 - 5)$

$= 15 - 11$

$= 4$

G - commutative semi-group $*$, identity ε.



Special case :

G → any totally ordered set

$*$ → the min operation.

ε → PLUS_INFINITY.

$O(\log n)$ time $O(n)$ work algo for min finding.

The LCA (Lowest Common Ancestor) Problem

Rooted tree T

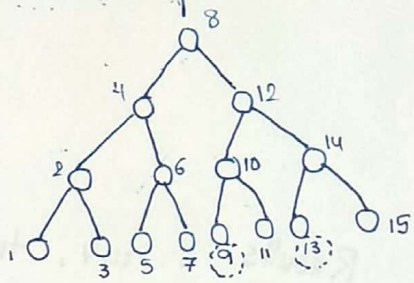
u, v are two nodes in T.

LCA (u, v) is a common ancestor of u and v and is at a level as large ~~as~~ as possible.

Ideally like to answer LCA queries in O(1) sequential time

~~Preprocess T~~

LCA in a complete binary tree



$Inorderno(u) = z_{l-1} z_{l-2} \cdots z_i 1 u_{i-2} \cdots u_0$

$Inorderno(v) = z_{l-1} z_{l-2} \cdots z_i 0 v_{i-2} \cdots v_0$

$LCA(u,v) = z_{l-1} z_{l-2} \cdots z_i 100 \cdots 0$

LCA (u, v)

~~Inorderno (u) = $z_{l-1} z$~~

~~Inorderno = z~~

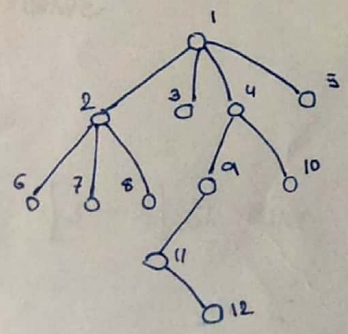
Proof: Exercise. (can use induction)

Eg. Let u = 9, v = 13

= 1001 = 1101.

LCA (u, v) = 1100 = 12.

T - a general rooted tree



Euler tour

$\langle r, u_1 \rangle \langle u_1, u_2 \rangle \langle u_2, u_3 \rangle \cdots$

$A[] = r, u_1, u_2, u_3, \cdots, u_k, r$

Euler array

A = [1, 2, 6, 2, 7, 2, 8, 2, 1, 3, 1, 4, 9, 11, 12, 11, 9, ~~4, 1, 5,~~ 4, 10, 4, 1, 5, 1]

|A| = 2(n-1) + 1 = 2n-1

B [0, 1, 2, 1, 2, 1, 2, 1, 0, 1, 0, 1, 2, 3, 4, 3, ~~2, 3, 2, 1, 2,~~ 2, 1, 2, 1, 0, 1, 0]

$b_i = \text{Level}(a_i)$

$\forall v \in V$

$l(v)$ = index of the first occurrence of v in $A[]$.

$r(v)$ = index of the last occurrence of v in $A[]$.

If v is the root,

$l(v) = 1 \quad r(v) = 2n-1.$

$v \neq$ root

$l(v) = i$ such that $b_{i-1} = b_i - 1$ and $a_j = v$

$r(v) = j$ such that $b_{j+1} = b_j - 1$ and $a_j = v$

$O(1)$ parallel time

$O(n)$ work

Results : u, v, two different nodes in T.

(1) u is an ancestor of v.

$\Leftrightarrow \quad l(u) < l(v) < r(u)$

(2) u is not an ancestor of v and v is not an ancestor of u.

$\Leftrightarrow \quad r(u) < l(v) \quad$ or $\quad r(v) < l(u)$

(3) $r(u) < l(v)$

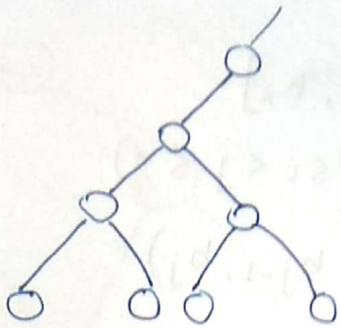
$A[r(u) \cdots l(v)]$ contains the LCA (u,v) at positions where $B[r(u) \cdots l(v)]$ is the smallest

LCA problem reduces to the

Range Minimum Problem

Given an array B and indices k, l find $\min B[k \cdots l]$

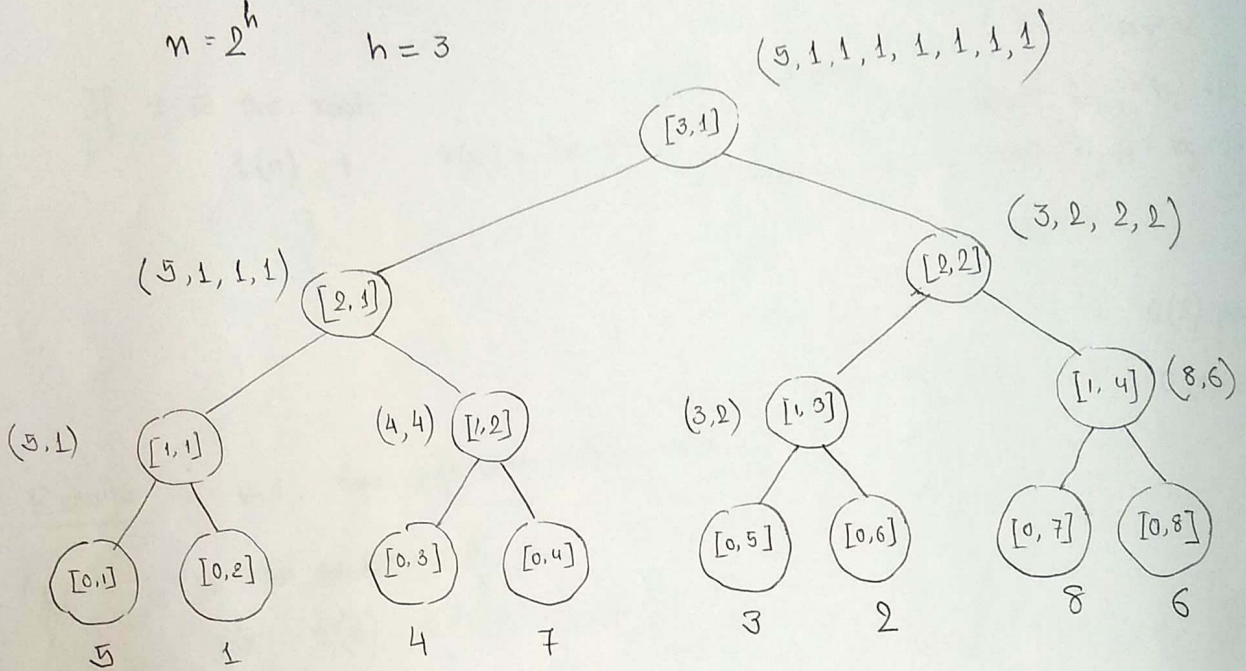
$|B| = 2^h$



Range Minima Problem

Given an array $B = [b_1, b_2, \cdots, b_n]$
and two indices i, j $(1 \le i < j \le n)$
to compute $\min(b_i, b_{i+1}, \cdots, b_{j-1}, b_j)$

$n = 2^h$ $\quad h = 3$



Node $[k, l]$

$P[1 \cdots 2^k] = (pm_1, pm_2, \cdots, pm_{2^k})$

$S[1 \cdots 2^k] = (sm_1, sm_2, \cdots, sm_{2^k})$.

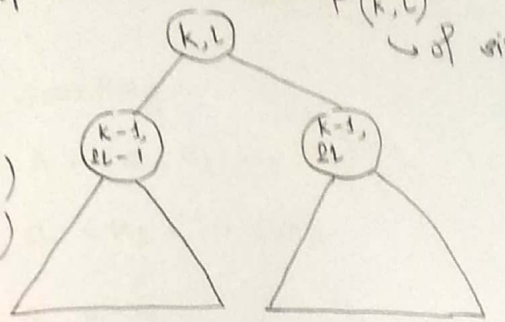
```
for 1 ≤ i ≤ 2^h pardo {
    set P[0, i] = (b_i)
    and S[0, i] = (b_i)
}
```

} O(1) time, O(n) work

```
for k = 1, 2, 3, ..., h do {          ← log n iterations
    for l = 1, 2, ..., n/2^k pardo {
        Merge P(k-1, 2l-1) and P(k-1, 2l)
              to P(k, l)
        Merge S(k-1, 2l-1) and S(k-1, 2l)
              to S(k, l)
    }
}.
```

} O(1) time, $O\left(\frac{n}{2^k} \times 2^k\right) = O(n)$ work

Merging



$P(k,l)$ ↳ of size 2^k

$P(k-1, 2l-1)$ (of size 2^{k-1})

$P(k-1, 2l)$ (of size 2^{k-1})

Copy $P(k-1, 2l-1)$ in parallel to the left half of $P(k,l)$

Let α be the last element of $P(k-1, 2l-1)$

Copy $\min(P(k-1, 2l), \alpha)$ in parallel to the right half of $P(k,l)$.

∴ Preprocessing in
$O(\log n)$ time
$O(n \log n)$ work.

Need a better algo ! $O(n \log n)$ won't do.

1. Break B into $\frac{n}{\log n}$ chunks B_i each of size $\log n$. ($O(n)$ work, $O(1)$ time)
2. Process each individual block B_i using the best sequential algorithm. ($O(n)$ work (??))
3. Compute the prefix and suffix minima for each block. → $O(n)$ work
4. Let $x_i = \min B_i$. Apply the previous algorithm on $(x_1, x_2, \ldots, x_{\frac{n}{\log n}})$ = x

↳ $O\left(\frac{n}{\log n} \log\left(\frac{n}{\log n}\right)\right) = O(n)$ work.

How to answer range minimum queries?

B, i, j

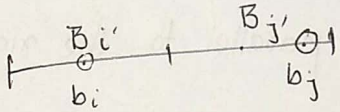
$b_i \in B_{i'}$

$b_j \in B_{j'}$

Case 1: $i' = j'$

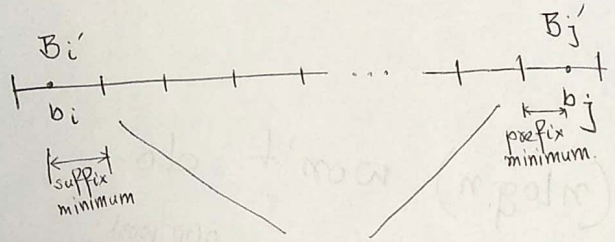
Step 2 has already handled this situation.

Case 2: $i' + 1 = j'$



$\min$ (suffix minimum of b_i in $B_{i'}$;
prefix minimum of b_j in $B_{j'}$)

Case 3: $j' \geq i' + 2$.



Make a range minimum query on X' with indices $i'+1$, $j'-1$

SEARCHING, MERGING & SORTING

1.1. Searching

$A = (a_1, a_2, \ldots, a_n)$

$a_1 < a_2 < \cdots < a_n$

$a_0 = -\infty$

$a_{n+1} = +\infty$

Given b to find the index i such that

$a_i \leq b < a_{i+1}$.

Sequential algo : binary search

↓

$O(\log n)$ time. → Optimal (Can't be improved upon)

p processors are available

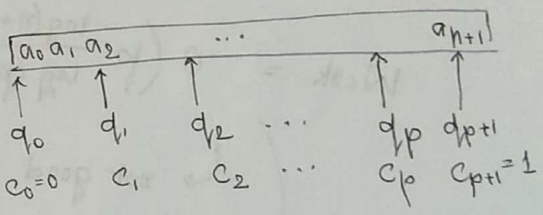
make a $(p+1)$ way search.

Algorithm for processor j, $1 \leq j \leq p$.

if $(j=1)$ set $q_0 = 0$ and $q_{p+1} = n+1$.

and $l = q_0$ and $r = n+1$.

$c_0 = 0$ and $c_{p+1} = 1$



while $(r - l > p)$ {

set $q_j = l + \left\lfloor \frac{r-l}{p+1} \right\rfloor * j$

if $(b = a_{q_j})$ { return q_j, exit }

else if $(a_{q_j} < b)$ set $c_j = 0$.

else set $c_j = 1$.

If $(c_j < c_{j+1})$ set $l = q_j$ and $r = q_{j+1} - 1$

If $(j=1)$ and $(c_0 < c_1)$

set $l = 0$ and $r = q_1 - 1$.

// Here $r - l \leq p$

f $(j \leq r - l)$ {

if $a_{l+j} = b$ {return $l+j$, exit }

else if $(b > a_{l+j})$ set $c_j = 0$

else set $c_j = 1$.

if $(c_{j-1} < c_j)$ { return $1+j-1$, exit }

}

$s_i = r_i - l_i$

$s_0 = n+1$.

$s_i \leq \frac{s_{i-1}}{p+1} + p$. (largest possible size of the last part).

$$s_i \leq \frac{n+1}{(p+1)^i} + p + 1$$

# of iterations of the while loop

$$= O\left(\frac{\log(n+1)}{\log(p+1)}\right).$$

If $p = n^{\alpha}$, $0 < \alpha < 1$

↑ constant

Then the running time is $O(1)$.

$$\text{Work} = O\left(p \frac{\log(n+1)}{\log(p+1)}\right).$$

↳ not good for $p = n^{\alpha}$.

↳ optimal, that is, $O(\log n)$, if p is a constant.

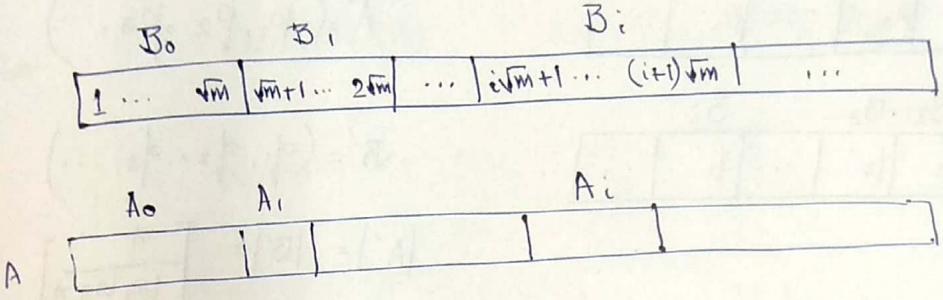
Work-Time Tradeoff

To compute rank $(B; A)$, $|B| = m$, $|A| = n$.

$m \leq n$

1. If $m \leq 3$, search with $p = n$ $\longrightarrow$ $O(1)$ time, $O(n)$ work
2. Let $Y = (b_{\sqrt{m}}, b_{2\sqrt{m}}, \ldots, b_m)$ $\leftarrow$ "small".

   Compute rank $(Y; A)$ $\rightarrow$ $O(1)$ time, $O(n)$ work.



3. $\forall\, i = 0, 1, \ldots, \sqrt{m} - 1$ pardo

   if $(j(i) = j(i+1))$ set rank $(B_i; A_i) = (0, 0, \ldots, 0)$

   else recursively compute rank $(B_i; A_i)$

4. $\forall k$ which is not a multiple of $\sqrt{m}$,

   set $i = \left\lfloor \frac{k}{\sqrt{m}} \right\rfloor$ and rank $(y_k; A) = j(i-1) + \text{rank}(y_k; A_i)$

Correctness : clear (?)

Induct

Merging

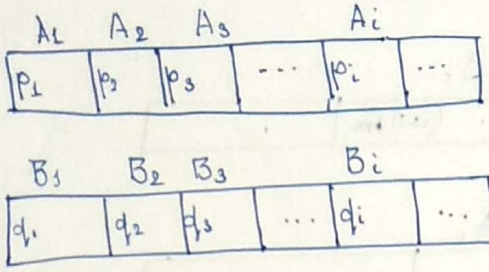
pre-midsem $O(\log\log n)$ time, $O(n \log\log n)$ work

post-midsem $O(\log\log n)$ time, $O(n)$ work.

$|A| = |B| = n$

No duplicates in $A \cup B$.

Break A and B into chunks of size $\lceil \log\log n \rceil$



$A' = (p_1, p_2, p_3, \dots)$

$B' = (q_1, q_2, q_3, \dots)$

$|A'| = |B'| = \left\lceil \frac{n}{\log\log n} \right\rceil$

Step 1: Compute Rank $(A'; B')$ and Rank (B', A') using the pre-midsem algo.

$O(\log\log n)$ time, $O(n)$ work.

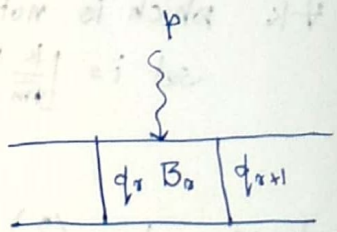
Step 2: Compute Rank $(A; B)$ and Rank $(B; A)$

* Rank $(A'; B')$

$p_i \in A'$

In Step 1, we have completed

r = rank $(p; B')$



rank (p) can be determined by Linear search (sequential) in $O(\log\log n)$ time by computing its rank in B_r

Step 3:

Rank $(A; B)$ and Rank $(B; A)$

Do the merging

$a \in A_i$, $a \neq p_i$

rank $(p_i; B) = r_i$

rank $(q_j, A) = s_j$

} available to step 2

Case 1: $r_i = r_{i+1}$ sits immediately after b_{r_i}.

Entire block A_i ~~belongs to Block B_{r_i}~~.

~~merge (A_i, B_{r_i})~~

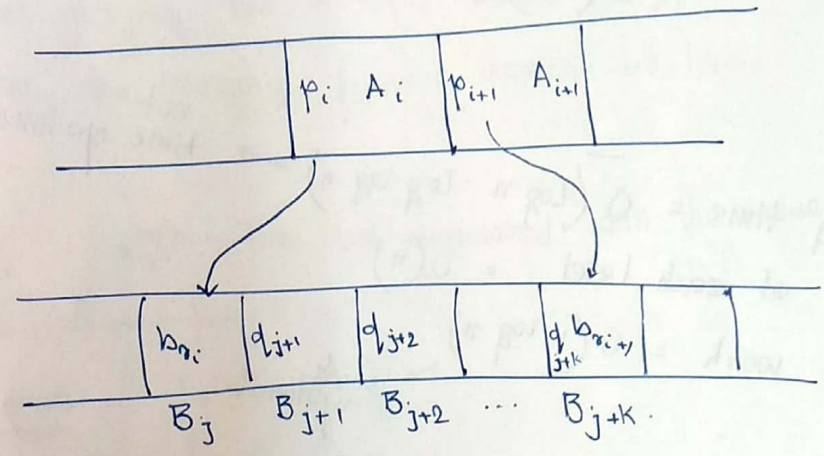
Case 2: $r_{i+1} > r_i$

subcase (a) $r_{i+1} - r_i \leq \log \log n$

merge $(A_i, B[r_i \cdots r_{i+1}-1])$ sequentially.

$O(\log \log n)$ time and work.

~~Ca~~ subcase (b) $r_{i+1} - r_i > \log \log n$



~~compute~~ rank $(b_{r_i}, q_{j+1}, q_{j+2}, \cdots, q_{j+k})$, ~~$b_{r_{i+1}}$~~ in A and in A_i.

are known from step 2. So

Compute rank (b_{r_i}), rank $(b_{r_{i+1}-1})$ } sequentially.

These ranks subdivide A_i.

Merge in parallel blocks / sub-blocks in B with the sub-blocks of A_i.

~~$O(\log \log$~~ $O\left(\frac{n}{\log \log n}\right)$ merging can run in parallel

no merging (sub) block is of size $> \log \log n$.

Parallel merge sort

$n = 2^t$.

Balanced binary search tree method.

(bottom-up)

```
for 1 ≤ i ≤ n pardo set L(0;i) = a_i
    for h = 1, 2, ..., t do {
        Merge (
        for i = 1, 2, ..., n/2^h pardo {
            merge L(h-1, 2i-1) and L(h-1, 2i)
                to L(h,i)
        }
    }
}
```

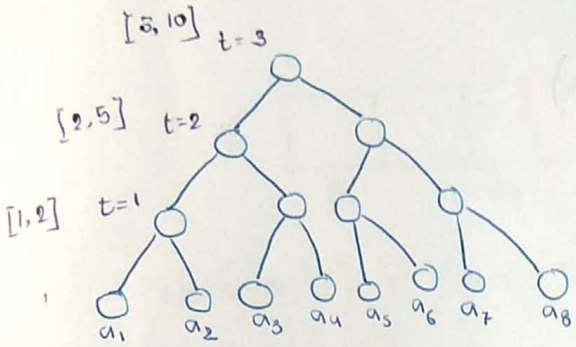
running time = $O(\log n \log\log n)$ → not time-optimal

work at each level = $O(n)$

total work = $O(n \log n)$ → work optimal

Forbidden Algorithm 2

Sorting in $O(\log n)$ time
with $O(n \log n)$ work



- To let merging run in $O(1)$ time.
- Pipeline the merging process across all levels.

Init: v leaf, $L_0(v)$ = the list associated with the value stored there.

~~$L_0(v) = ( )$~~

v internal node, $L_0(v) = ()$

After stage $s = 1, 2, 3, \ldots$

$L_s(v) \rightarrow$ the list at node v.

Eventually, v will store the complete list $L(v)$

Each $L_s(v)$ is an aproximation of $L(v)$

$X = (x_1, x_2, x_3, \ldots)$

$c \in \mathbb{N}$

$\text{Sample}_c(X) = (x_c, x_{2c}, x_{3c}, \ldots)$

Let v be a node at height h.

Let s be a stage number

$$\text{sample}(L_s(v)) = \begin{cases} \text{Sample}_4(L_s(v)) & \text{if } s \leq 3h \\ \text{sample}_2(L_s(v)) & \text{if } s = 3h+1 \\ \text{sample}_1(L_s(v)) & \text{if } s = 3h+2 \end{cases}$$

A node v at height h remains active during the time stages $h \leq s \leq 3h$.

- To let merging run in $O(1)$ time.

For $i = 1, 2, \cdots, n$ pardo
    initialize $L_0(i^{th}\ leaf) = (a_i)$

For all internal nodes v pardo
    initialize $L_0(v) = \phi$

For $s = 0, 1, 2, \cdots, 3t-1$ do {
    /* stage $s+1$ */
    For all active nodes v pardo {
        Let u and w be the two children of v.
        Get $L'_s(u) = sample\ (L_s(u))$
            $L'_s(w) = sample\ (L_s(w))$
        Merge $L'_s(u)$ and $L'_s(w)$ using $L_s(v)$
        as a cover to generate $L_{s+1}(v)$
                ↳ 4-cover!
    }
}

$X = (x_1, x_2, x_3, \cdots, x_s)$ be a sorted list

$X_\infty = (\underset{x_0}{-\infty}, x_1, x_2, \cdots, x_s, \underset{x_{s+1}}{\infty})$

$A = (a_1, a_2, \cdots, a_n)$ be another sorted sequence.

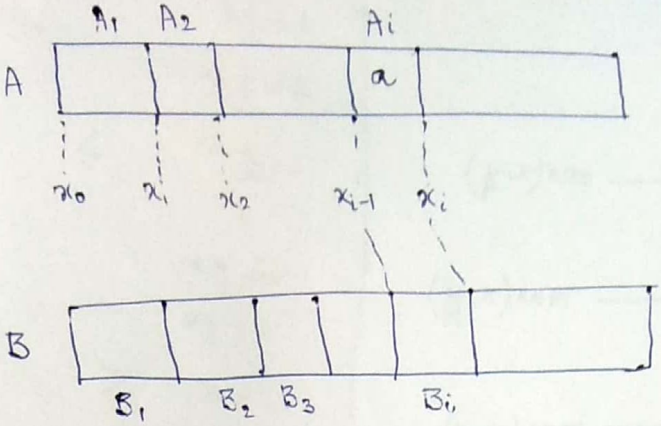
X is called a c-cover of A if

$$\left|\{a_i \mid x_j < a_i \leq x_{j+1}\}\right| \leq c$$

for all $j = 0, 1, 2, \cdots, s$

Prop: Let A, B be two sorted lists of size n. Let X be a c-cover of both A and B. For some constant c. Suppose that Rank $(X; A)$ and Rank $(X; B)$ are known. Then Rank $(A; B)$ and Rank $(B; A)$ can be ~~completed~~ computed in $O(1)$ time with $O(n)$ work. In particular, A and B can be merged in $O(1)$ time with $O(n)$ work.

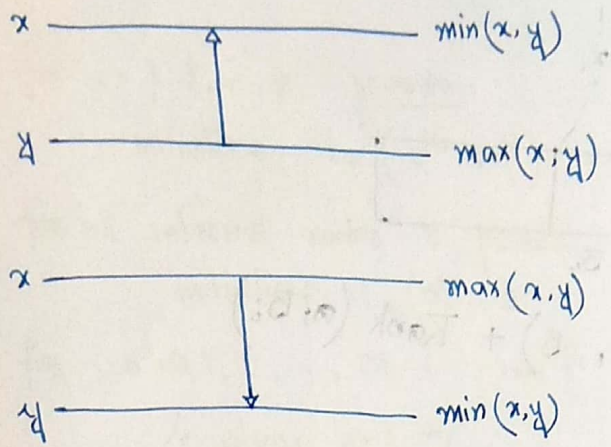
Proof:



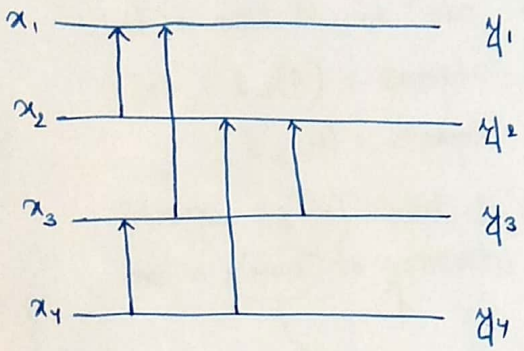
$$Rank(a; B) = Rank(x_{i-1}; B) + Rank(a; B_i)$$

Sorting networks

A comparator



Eg.



depth = maximum no. of comparators that you encounter from an input to an output.
|||
runtime

size = the no. of comparators.
|||
work

Bitonic sequences

5 6 7 8 9 4 1 0
1 2

8 9 4 1 0 5 6 7
1 2 1

Unique Crossover Property.

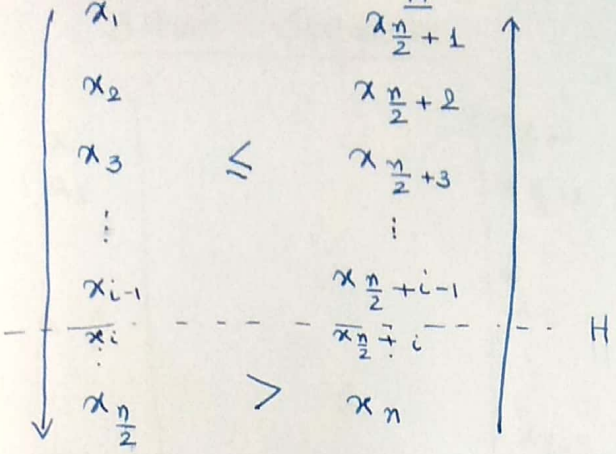
$x_1, x_2, \cdots, x_n$

$x_1 \geq x_2 \geq x_3 \geq \cdots \geq x_{n/2}$

$x_{\frac{n}{2}+1} \leq x_{\frac{n}{2}+2} \leq \cdots \leq x_n$

$x_1 \leq x_2 \leq x_3 \leq \cdots \leq x_{n/2}$

$x_{\frac{n}{2}+1} \geq x_{\frac{n}{2}+2} \geq \cdots \geq x_n$



(1) $x_i \in L$, $x_{\frac{n}{2}+i} \in R$.

If $i \in \leq$, then $x_i \leq x_{\frac{n}{2}+i}$

If $i \in >$, then $x_i > x_{\frac{n}{2}+i}$

(2) If $x_i, x_j \in L$, $i < j$, then $x_i \leq x_j$

If $x_i, x_j \in R$, $i < j$, then $x_i \geq x_j$

$x_i > x_{\frac{n}{2}+i}$ for the first i.

1 2 3 4 5 6 10 9 8 7

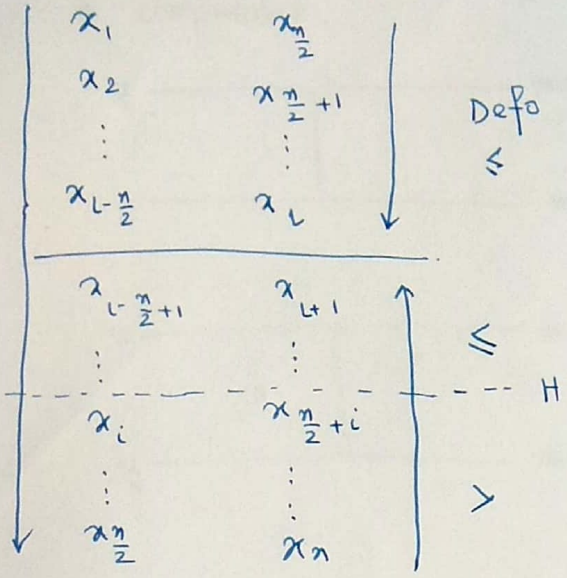
4	$\leq$	8
5		7
6		1
10	$>$	2
9		3

Proposition - Unique crossover property holds for any critical sequence.

Proof - Without loss of generality,

$x_1 \leq x_2 \leq x_3 \cdots \leq x_L$

$x_{L+1} \geqslant x_{L+2} \geqslant \cdots \geqslant x_n$.



Found i such that $x_i > x_{\frac{n}{2}+i}$

Sort a bitonic sequence

$X = (x_1, x_2, \cdots, x_n)$ $\quad n = 2^t$

$L = (y_1, y_2, \cdots, y_{\frac{n}{2}})$ $\quad y_i = \min(x_i, x_{\frac{n}{2}+i})$

$R = (z_1, z_2, \cdots z_{\frac{n}{2}})$ $\quad z_i = \max(x_i, x_{\frac{n}{2}+i})$

// different i not the crossover i

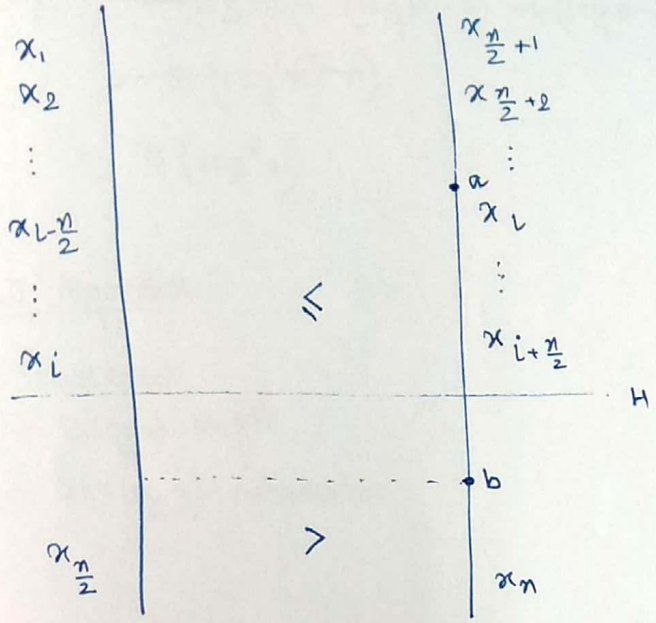
$L = (x_1, x_2, \cdots, x_{i-1}, x_{\frac{n}{2}+i}, \cdots, x_n)$

$R = (x_{\frac{n}{2}+1}, x_{\frac{n}{2}+2}, \cdots, x_{i+\frac{n}{2}-1}, x_i, \cdots, x_{\frac{n}{2}})$.

L and R are subsequences of X and so, are bitonic.

Any element of $L \leq$ Any element of R.

Bitonic Sequences



$$y_i = \min(x_i, x_{\frac{n}{2}+i}) \quad \left.\right\} \text{ for } i = 1, \ldots, \frac{n}{2}$$
$$z_i = \max(x_i, x_{\frac{n}{2}+i}) \quad \left.\right\}$$

X, Y, Z

Claim: Every element of Y is smaller than every element of Z.

Property 2

Take a, b from the same column c. such that a is in the $\leq$ region and b is in the $>$ region.

Then, $a \leq b$ if $c = L$

or, $a \geq b$ if $c = R$

$$Y = (x_1, x_2, \cdots, x_i, x_{\frac{n}{2}+i+1}, \cdots, x_n)$$

$$Z = (x_{\frac{n}{2}+1}, x_{\frac{n}{2}+2}, \cdots, x_{\frac{n}{2}+i}, x_{i+1}, \cdots, x_{\frac{n}{2}})$$

Sorting bitonic sequences

$$C(n) = \frac{n}{2} + 2C\left(\frac{n}{2}\right)$$
$$D(n) = 1 + D\left(\frac{n}{2}\right)$$

$$C(n) = O(n \log n)$$
$$D(n) = O(\log n)$$

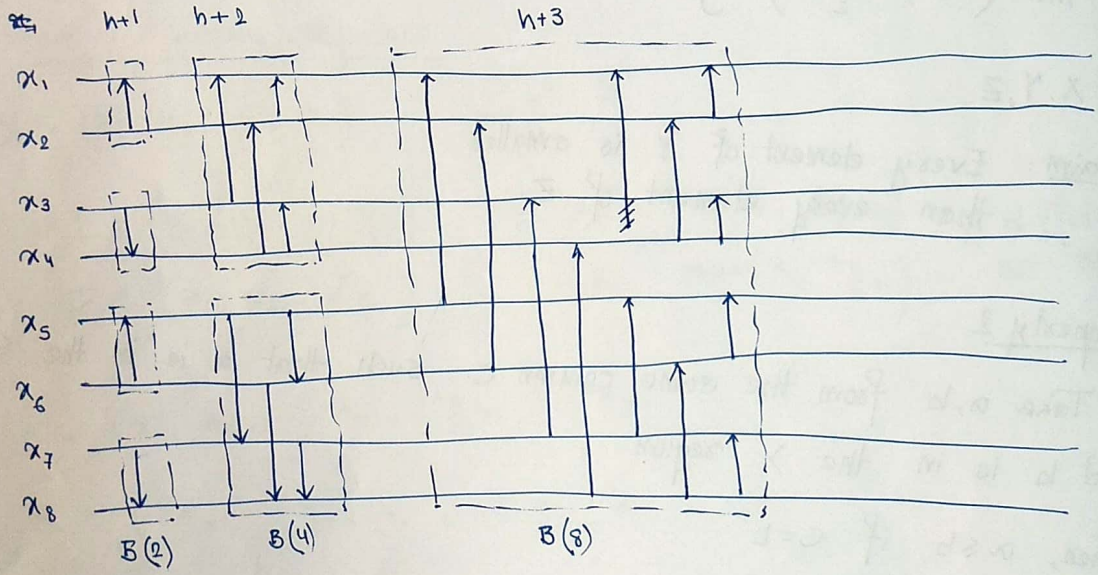
Sorting arbitrary sequences

$B(k)$ → bitonic sorting network for k inputs

height 2

height 2 $\frac{n}{4}$ $B(4)$ networks

height 1 $\frac{n}{2}$ $B(2)$ networks



$C(k)$, $D(k)$ → for bitonic sorter

$C'(k)$, $D'(k)$ → for orbit sorter

$$C'(n) = C(n) + 2C\left(\frac{n}{2}\right) + 4C\left(\frac{n}{4}\right) + \cdots + \frac{n}{2}C(2)$$

$$= O\left(n\log n + 2\cdot\frac{n}{2}\cdot\log\frac{n}{2} + \cdots\right)$$

$$= O(n\log^2 n)$$

$$D'(n) = D(n) + D\left(\frac{n}{2}\right) + D\left(\frac{n}{4}\right) + \cdots + D(2)$$

$$= (\log n) + (\log n - 1) + (\log n - 2) + \cdots + 1$$

~~$= O(\log^2 n)$~~

$$= O(\log^2 n)$$

AKS Algorithm

Szemeredi

$O(\log n)$ depth

$O(n \log n)$ comparator.

Selection

k^{th} smallest in unsorted

$$A = (a_1, a_2, \cdots, a_n)$$

An $O(n)$ - work $O(\log n \log\log n)$ algorithm.

We know an element a in A such that

$$\frac{n}{4} \leq \text{Rank}(a, A) \leq \frac{3n}{4}$$

A_1	A_2	A_3
$< a$	$= a$	$> a$
n_1	n_2	n_3

$n_1, n_3 \leq \frac{3n}{4}$

If $n_1 < k \leq n_1 + n_2$, output a

else if $k \leq n_1$, call select (A_1, k)

else call select $(A_3, k - (n_1 + n_2))$

Recursive call reduces the array size by a factor of $\frac{3}{4}$.

$\Rightarrow$ After $O(\log\log n)$ iterations, the array size becomes $\leq \frac{n}{\log n}$.

Now, sort the array. (accelerated cascading).

How to find a?

$n_0 = n$

n_1 = the size of A after 1 one reduction

n_2

$\vdots$

n_s = ... after s reductions

A contains n_s elements

Break A into $\frac{n_s}{\log n}$ chunks each of size $\log n$.

For all chunks; pardo compute the median m_j → each chunked can be processed in $O(\log n)$ time

Total work $= O(n_s)$.

sort $m_1, m_2, \cdots, m_{\frac{n_s}{\log n}}$ by FA2

Runtime $= O\left(\log\left(\frac{n_s}{\log n}\right)\right)$

$= O(\log n)$

Work $= O(n_s)$

Each m_j is $\geq \frac{\log n}{2}$ elements in its chunk

$a \geq \frac{n}{2 \log n}$ chunk medians

$a \geq \frac{n}{2 \log n} \times \frac{\log n}{2} = \frac{n}{4}$ elements of A

Total work in the reduction process

$= O(n_0 + n_1 + \cdots)$

$= O\left(n + \frac{3}{4}n + \left(\frac{3}{4}\right)^2 n + \cdots\right)$

$= O\left(\frac{n}{1 - \frac{3}{4}}\right) = O(n)$

Lower Bounds

Parallel Comparison Trees –

Each comparison has 2 or 3 outputs

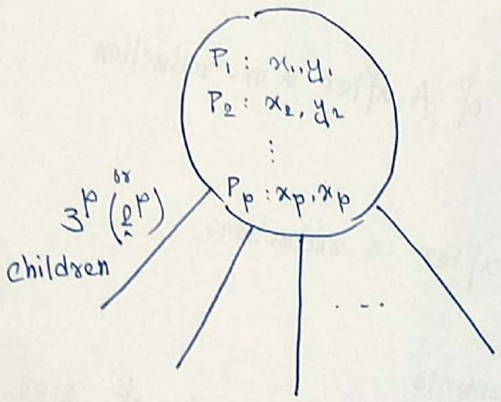
$=, \neq$

$\leq, >$

$<, =, >$

No arithmetic with the numbers

p processors



$\Omega(n)$
$\Omega\left(\frac{n}{2}\right)$
} for most problems

Adversary Strategy

~~Force the adversary~~

The adversary forces the algo to run on a computation path as long as possible.

Lower bound on Search

n distinct elements ← A → a sorted array.

x → to check whether x belongs to A

p processors.

Any parallel algorithm to solve this problem takes ~~$\Omega\left(\frac{\log n}{\log(p+1)}\right)$~~

~~$\Omega\left(\log\left(\frac{n}{p+1}\right)\right)$~~ running time.

~~$\Omega\left(\frac{\log n}{p+1}\right)$~~ $\Omega\left(\frac{\log n}{\log(p+1)}\right)$

First comparison

P_1 : a_{i1} ; a

P_2 : a_{i2} ; a

...

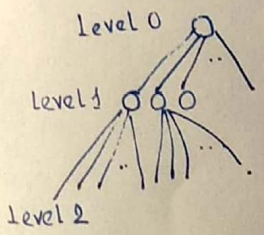
P_p : a_{ip} ; a

↓

Partition A into $\leq p+1$ parts.

At least one part has size $\geq \frac{n}{p+1}$.

To make the computation long, the adversary chooses the largest part.



$n_0 = n$

$n_1 \geq \frac{n}{p+1}$

$n_2 \geq \frac{n_1}{p+1} \geq \frac{n}{(p+1)^2}$

$n_k \geq \frac{n}{(p+1)^k}$.

k should be such that

$1 \geq \frac{n}{(p+1)^k}$

$(p+1)^k \geq n \Rightarrow k \geq \frac{\log n}{\log(p+1)}$

If we have n processors, i.e., $p = n$, how to search in ~~$O(n)$~~ $O(1)$ time?

$$n_{i+1} \geq \frac{n_i}{p+1} \quad \text{---------} \quad (*)$$

Lower bound
— best algorithm
↓
Worst case of the best algorithm.

If p processors are available, then any search algorithm, however clever, cannot make a size reduction better than $(*)$ in the presence of a "smart" adversary.

Lower bound on max

$O(\log \log n)$ time
$O(n)$ ~~tree~~ work

Property: If $p \leq n$, then this problem can be solved in $\Omega(\log \log n)$ time.

Lemma (Turin)

$G = (V, E)$

$|V| = n, \quad |E| = m$

$\Rightarrow$ G contains an independent set of size $\geq \frac{n^2}{2m+n}$.

p comparisons $\quad A = (a_1, a_2, \cdots, a_n)$

a_{i_1}, a_{j_1}
a_{i_2}, a_{j_2}
$\vdots$
a_{i_p}, a_{j_p}

We construct a graph

$G = (V, E)$
$|V| = n$
$|E| \leq p$

$\therefore$ G has an independent set of size: $\geq \frac{n^2}{2p+n} \geq \frac{n}{3}$

$$n_{i+1} \geq \frac{n_i}{3}$$

Supposed to be $\boxed{n_{i+1} \geq \frac{n_i^2}{3}}$

$\therefore \Omega(\log \log n)$

Lower bound on max

n $\quad n_0 = n$

$n/3$ $\quad n_1$

n_2

$\vdots$

$$n_i \geq \frac{n_{i-1}^2}{2p + n_{i-1}} \geq \frac{n_{i-1}^2}{3n} \geq \cancel{\left(\frac{n_{i-2}^2}{(3n)^2}\right)\frac{1}{3n}}$$

$$n_2 \geq \left(\frac{n_1^2}{3n}\right) \geq \left(\frac{(n/3)^2}{3n}\right) = \frac{n}{3^3}$$

$$n_3 \geq \left(\frac{n_2^2}{3n}\right) \geq \left(\frac{\left(\frac{n}{3^3}\right)^2}{3n}\right) = \frac{n}{3^7}$$

$$\therefore \quad n_i \geq \frac{n}{3^{2^i - 1}}$$

Lower bound on merging

(k, s) merging problem.

k instances each of size s.

$A_i, B_i \qquad i = 1, 2, \ldots, k.$

Each A_i, B_i is sorted.

$$|A_i| = |B_i| = s$$

Each element of $A_i \cup B_i$ is smaller then each element of $A_{i+1} \cup B_{i+1}$.

$$2ks \leq p \leq \frac{ks^2}{8}$$

We want to generate (k', s') merging instance.

Divide each A_i, B_i into b blocks.

$$b = \sqrt{\frac{8p}{k}} = 2\sqrt{2}\sqrt{\frac{p}{k}}$$

$$4\sqrt{3} \leq b \leq s$$

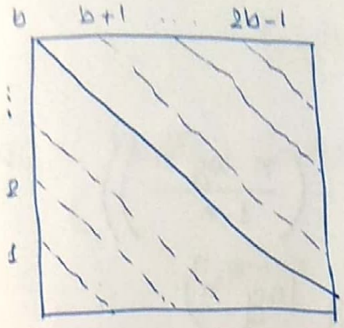
s' = size of each block $= \frac{s}{b} = s\sqrt{\frac{k}{8p}}$

(To compute k')

For one parallel computation step,

M_i $b \times b$ matrix for each $i = 1, 2, \ldots, k$

$$M_i(u,v) = \begin{cases} 1 & \text{if some element of } A_{i,u} \text{ is compared with some element of } B_{i,v}. \\ 0 & \text{otherwise.} \end{cases}$$



These ~~non~~-zero entries are (l_1, m_1), $(l_2, m_2), \ldots, (l_{d_{i,j}}, m_{d_{i,j}})$

The adversary arranges the values such that

(l_i, m_i) sub instance $< (l_{i+1}, m_{i+1})$ subinstance

D_{ij} = the j^{th} diagonal of M_i

$d_{i,j}$ = The number of 0's in $D_{i,j}$

$$\sum_{i=1}^{k} \sum_{j=1}^{2b-1} d_{ij} \geq kb^2 - p = 7p \qquad \left[\because b = \sqrt{\frac{8p}{k}} \Rightarrow b^2 k = 8p\right]$$

$$\sum_{j=1}^{2b-1} \sum_{i=1}^{k} d_{ij} \geq 7p$$

$$\sum_{i=1}^{k} d_{ij} \geq \frac{7p}{2b-1} \geq \frac{7p}{2b} = \frac{7p}{4\sqrt{2}\sqrt{\frac{p}{k}}} = \frac{7}{4\sqrt{2}} \sqrt{pk}$$

$$2sk \leq p \leq \frac{s^2}{8}$$

Running time. $T(k, s, p) \geq 1 + T(k', s', p)$

$$s' = \frac{s}{2\sqrt{2}}\sqrt{\frac{p}{k}} \geq 1 + T\left(\frac{7}{4\sqrt{2}}\sqrt{kp} \cdot \frac{s}{2\sqrt{2}}\sqrt{\frac{p}{k}}, p\right)$$

Leads to an inductive proof of

$$T(k, s, p) \geq \frac{1}{4}\log\left(\frac{\log(p/k)}{\log(p/ks)}\right)$$

In particular,

$$T(k, s, p) = \Omega\left(\log\log s - \log\log\left(\frac{p}{ks}\right)\right)$$

Special case : (Our problem)

$(1, n)$ merging.

$p = n \log^{r} n$, r constant.

$$T(1, n) = \Omega\left(\log\log n - \log\log\left(\frac{n \log^{r} n}{1 \cdot n}\right)\right).$$

$$= \Omega\left(\log\log n - \log\log\log n\right)$$

$$= \Omega\left(\log\log n\right)$$

achievable. ↗

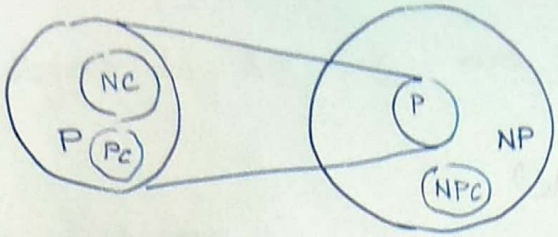
Sorting

$\Omega(n \log n)$ sequential.

If we use $p \leq n$ processors parallel running time is

$$\Omega(\log n).$$

What if $p > n$?



$P \stackrel{?}{=} NP$

Polynomial Running time.

NC - Nick's Complexity Class.

Similar to the $P \stackrel{?}{=} NP$ problem,

$NC \stackrel{?}{=} P$ problem

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 (Nicks Complexity class) if
P can be solved by a polylogarithmic-time algorithm using a polynomial no. of processors.

Example.

BFS

There is an algorithm which $O(\log^2 n)$ time
$O(M(n) \log n)$ work

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

$M(n) = n^3$ using standard algo

Coppersmith-Winnograd.

$M(n) = n^{2.3\ldots}$

p processors.

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

$\therefore$ BFS $\in$ NC

Some problems do not seem to be in NC (in P)

Examples:

~~(1) BFS~~

(1) DFS
(2) Network Flow
(3) Linear Programming.

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

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

Decision Problems

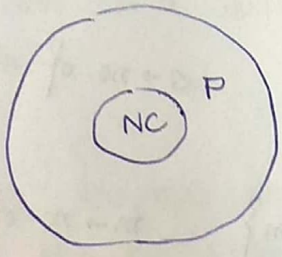
(G, s, t, k)

Is a network flow of amount k possible?

NC consists of decision problems only.

$NC \subseteq P$ ✓ (obvious) $P \stackrel{?}{\subseteq} NC$ (Not known.)

Popular belief $\rightarrow P \neq NC$.



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 some NC algorithm that given an input I_1 for P_1, generates an instance I_2 for P_2 such that

$P_2(I_2) = 0 \Leftrightarrow P_1(I_1) = 0$

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: $P_1 \leq_{NC} P_2$ and $P_2 \leq_{NC} P_3 \Rightarrow P_1 \leq_{NC} P_3$

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

First example :

CVP - Circuit Value Problem.

↳ Combinational circuit consisting of AND, OR and NOT gates.

Some inputs
One output.

Turing Machines
(Notations)

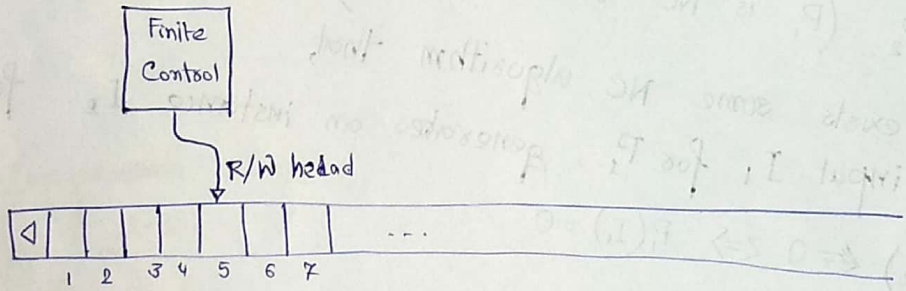
Q = set of states = $\{q_1, q_2, \cdots, q_s\}$ $\quad s \rightarrow$ no. of states

q_1 = start state.

Σ = tape alphabet = $\{a_1, a_2, \cdots, a_m\}$. $\quad m \rightarrow$ no. of symbols

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

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



Acceptance
Cell 1 contains 1

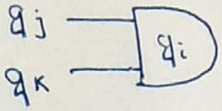
Rejection
Cell 1 contains 0

Theorem : The CVP is P-complete.

Instance of CVP

$\langle q_1, q_2, \cdots, q_n \rangle$

Each q_i is either an input or gate.



$j, k < i$

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

Let $Q \in P$

To show $Q \leq_{NC} CVP$

There is a DTM that accepts Q with running time $T(n)$.
↗ a polynomial in n

$t = 0$ the m/c starts working

$t = 1, 2, 3, \cdots, T(n)$

Three families of boolean functions –

(i) $H(i,t) = 1 \iff$ The head is scanning cell i at time t.
$0 \leq t \leq T(n)$
$0 \leq i \leq T(n)$

(ii) $C(i,j,t) = 1 \iff$ cell i contains a_j at time t.
$0 \leq t \leq T(n)$
$0 \leq i \leq T(n)$
$0 \leq j \leq m$

(iii) $S(k,t) = 1 \iff$ The m/c is in state q_k at time t.
$0 \leq t \leq T(n)$
$1 \leq k \leq s$

Init (Level 0)

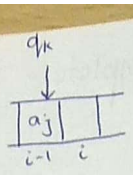
$O(1)$ time by $(T(n))^2$ processors.

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

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

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

From level t to level t+1



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

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

$$H(i, t+1) =$$

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

$O(T(n)^2)$ processors, $O(1)$ time

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

$O(T(n)^2)$ processors, $O(1)$ time

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

$O(T(n)^2)$ processors, $O(\log n)$ running time.

$$O/p = C(1, 1, T(n))$$

$T(n)^2$ processors $\rightarrow$ poly in n.

$O(\log n)$ running time

Reduction algo is in NC.

MCVP $\rightarrow$ monotone circuit value problem
Only AND & OR gates

MCVP is P-Complete

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