# Design Analysis of Algorithm

modules: → devide & conquer
→ Optimization: greedy, DP
→ Network flow
→ Interactibility - approximation (polynomial approximate of an exponent problem)
→ Advanced → Distributed, Cryptography..

## Very smaller problem can have very different Complexity:

P: class of problem solvable in $O(n^k)$ $\exists k \in R$.

NP: " " " whose solution can be checked in Polynomial time.

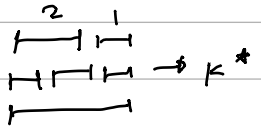
↳ E.g: given a Directed graph find a cycle which goes through each node exactly once (Hamiltonial Cycle).

NP-Complete: Problem is in NP and as hard as any problem in NP. (The hardest NP)

E.g. Hamiltonian Cycle.

## Scheduling problem:

resources: 3




Greedy Algorithm :-

1. Take 1 according to a rule / heuristics
2. Reject all incompatible ones.
3. Repeat 1 & 2 until remaining = 0.

Earliest finish time :-

Claim: given a list of intervals L, using min finish time can complete $k^*$ jobs ; claim $k^*$ is maximum possible

proof by contradiction :-

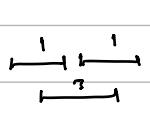

$S^* = \text{using greedy} = \{ a_1, a_2, a_3 \cdots \}$

another maximum sol$^n$ $S^{**} = \{ b_1, b_2, b_3 \cdots \}$


$S^{**}$ takes any other heuristics than $S^*$

finish time of $b_1$ > finish time of $a_1$

## weighted Scheduling Problem:



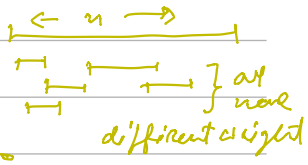

weight
1 : 1
3 : 33

DP solution:

no. of Subproblems = $n$ (amount of total time)

Complexity = no. of Subproblems × time to solve each subproblem (O(1) for repeat computation)

we have max $n$ length to fit all compatible scheduling





dp[i] = maximum weight possible till this finish time = i

$dp[i] = \max(dp[i],\ wt[x] + dp[i - x_s])$

check for all $x$ $x$ = appointment where $x_f$ finish time of $x$ <= $i$ & $x_s$ = start time of $x$ -

## Scheduling in Non Identical Machine:

(Assume weight = 1)

but some request is completed by some no. of machines.

$M_i$ can handle set of request $S_i(R)$.

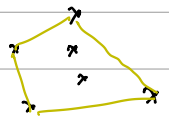
# This is NP Complete.

# Devide and Conquer

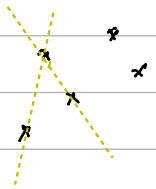
- Paradigm :- problem devided into a subproblem of size $n/b$.

- Convex hull :-

Smallest Convex Polygon



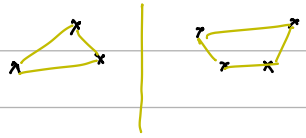
Brute force:
$O(n^3)$



Draw line & check whether rest of the points are on same side.

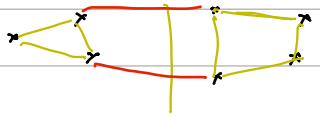
- Using Devide & Conquer :-

devide :

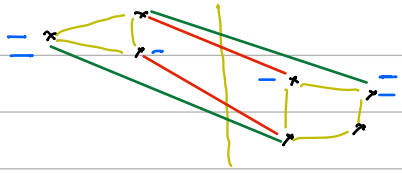


Sort on x axis
devide 1/2 on left 1/2 on right

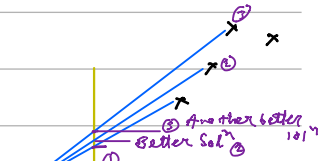
merge:

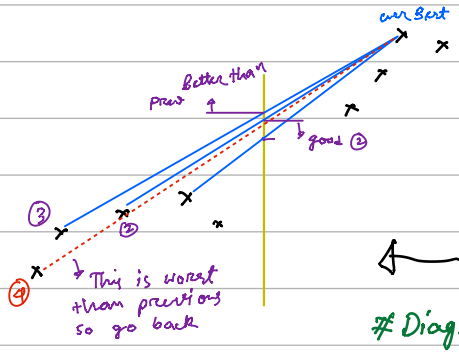


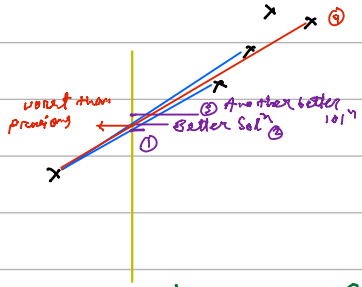
merging highest & lowest points may not work



correct way of doing this is with trying possibilities along the convex hull direction



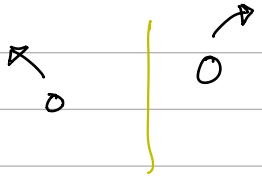



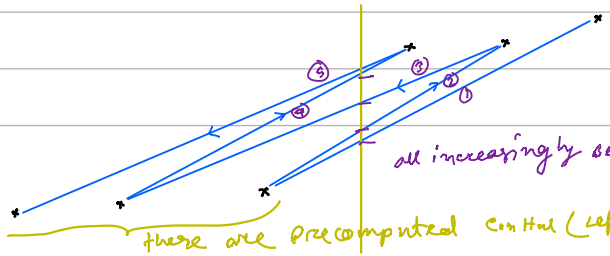

# Come back because worst than previous

# Diagram is not correct step.

algorithm is to move one step in left Part (anticlockwise) & then one step in Right part (clockwise).

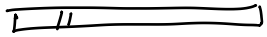
alternately check left & Right



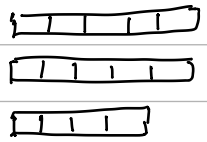


## Median of an Array :

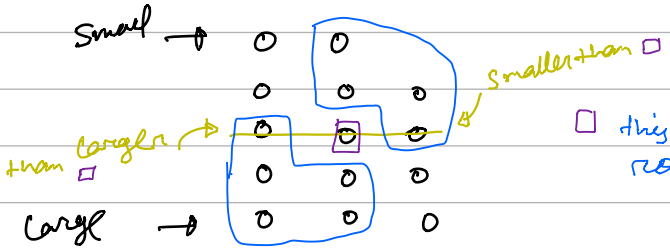
- one non deterministic way — pick a random element & find all those are smaller & those are larger.



- sort arr in chunk of 5



arr of 14 length




□ this element seperates the array roughly in 1/2 1/2.

Sorted these 3 arry according to the middle element in decreasing order

## Strassen's Algorithm for Matrix Multiplication :-

$$\begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_A \times \begin{bmatrix} \cdot & \cdot \\ \cdot & \cdot \end{bmatrix}_B = \begin{bmatrix} C_{11} & C_{12} \\ C_{13} & C_{14} \end{bmatrix}$$

$C_{11} = A_{11}B_{11} + A_{12}B_{21}$

$C_{12}$ — —

$C_{13}$ — —

$C_{14}$ — —

$8 \times \left(\frac{n}{2}\right)^3 \simeq n^3$ ✗

This is as efficient as Vanilla matrix Mul interms of complexity is concerned.

Strassen Defines $M_1, M_2, M_3, M_4, M_5, M_6, M_7$

↳ From these 7 matrices $C_{11}\ C_{12}\ C_{21}\ C_{22}$ can be calculated

## polynomials : $a_0 + a_1 x + a_2 x^2 + \cdots a_{n-1} x^{n-1}$

↳ root's , samples (for given x, find f(x))

| Algos | Representation format: coefficient | roots | samples | |
|---|---|---|---|---|
| 1) evaluation | O(n) | O(n) | $O(n^2)$ | no representation is perfect |
| 2) addition | O(n) | ∞ | O(n) | |
| 3) Multiplication | $O(n^2)$ | O(n) | O(n) | |

using FFT this transformation can be done in $n \log(n)$ time.

## FFT:

FFT is the algorithm; Discrete Fourier Transform is the math

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & & & \\ \vdots & & & & & \\ 1 & x_{n-1} & & & & \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ a_{n-1} \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ \vdots \\ y_{n-1} \end{bmatrix}$$

coefficient to sample. V.A $O(n^2)$

sample to coeff $V^{-1}Y$ $O(n^2)$

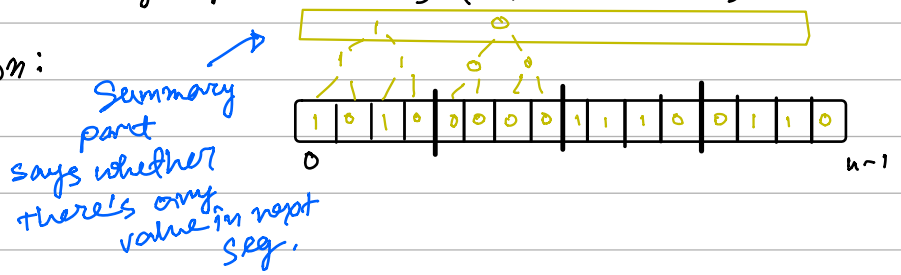
Gaussian Elimination - $O(n^3)$

↳ Transforming in Complex Coordinate (FFT $n \log n$)

## Van Emde Boas Tree :- $O(\log \log u)$ insert, Delete & Successor

$u$ : Range of $n$ elements $\{0, 1, \ldots, u-1\}$

main intuition:




Summary part says whether there's any value in next seg.

general space complexity = $(u \log \log u)$

put $n$ elements in a linked list ; space $O(u)$

# Amortization + amortized Analysis

↳ helps analyze complexity of certain Data structure.

E.g. Table doubling cost of insertion $2^0 + 2^1 + 2^2 + \cdots + 2^{\log n}$ for n elem

$= \theta(n)$

amortized cost per operation $= O(1)$

∅ aggregate method $\frac{\text{total cost of } K \text{ operation}}{K}$

= amortized cost per operation.

∅ Amortized bound: preserve the total cost.

$\sum_{\text{Total Op}}$ amortized cost $\geq \sum_{\text{Total}}$ actual cost

Eg. B-Tree (2-3 tree) have insert $\log n$ Delete $\log n$ + create $O(1)$

in Amortized — insert $\log n$ Delete $O(1)$

$c\,O(1) + i\log(n) + d\log(n)$

$\approx c\,O(1) + 2i\log(n) + d\,O(1)$

↳ Can't delete more than no. of insertion.

# For amortized Calculation Total Time spent matter.

∅ Accounting Method: Store credits & allow later operation take compensation for the credit

E.g. at every insertion stores Credit Deletion exhaus the credit.

∅ charging method: take charge from past; in past collect coin such that can afford future cost.

∅ Potential method: define potential fn ∅

→ amortized Cost = actual cost + $\Delta\phi$

$\Delta\phi = \phi(\text{after OP}) - \phi(\text{before OP})$

# $\sum$ amorti $= \sum$ actual $+ \phi(\text{end cost}) - \phi(\text{beginning cost})$

E.g. Binary Counter $+\ \begin{matrix}0\,0\,1\,0\,1\,1\,1\\ \hline 0\,0\,1\,1\,1\,1\,1\end{matrix}$ } addition 1 will remove 4 1's & add one 1.

actual cost $= O(1 + \#\text{trailing 1's})$

amortized $= 1 + t - t + 1 = 2 * c \approx O(1)$

eg. amortization,

$\phi$ in 2-3 tree no. of splits per insertion is $O(1)$ amortized.

→ actual: $\leq \log n$ worst case

$\phi$ = no. of nodes with 3 children

this stack up and all collapses and 1 3 children remains.

# ⇨ Randomization

probably Correct : (deterministically fast)
e.g. monte Carlo.

Probably Fast : e.g las vegas (deterministically incorrect)

Probably Correct + Probably fast → Atlantic City.

Matrix Multiplication :- Strassen $O(n^{2.81})$
Coppersmith $O(n^{2.73})$

checking mat Mul :- $O(n^2)$ with random checker.

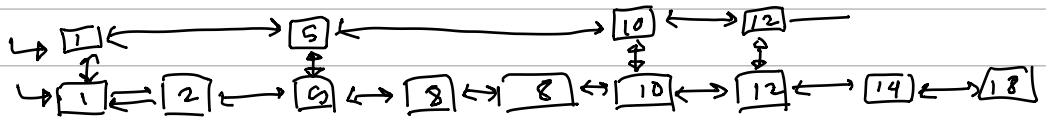
if $A \times B = C$ checker[ans = Yes] = 1

if $A \times B = C$ checker[ans = Yes] $\leq 1/2$

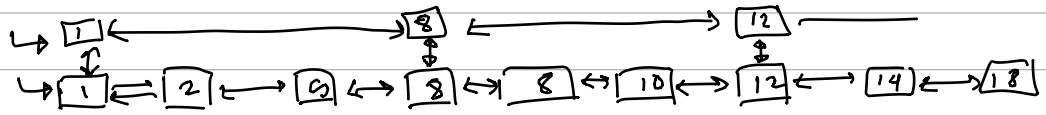
○ Randomized Data Structure : Skip List.

maintains dynamic set of n elements
in $O(\log n)$ time per operation.

normal linked list + with a finger list.



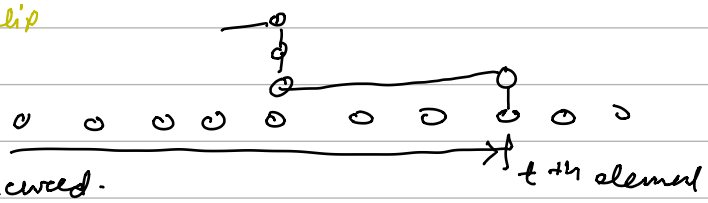

more optimally arrange the top list in $\sqrt{n}$ division




searching : $O(\sqrt{n})$


↑ flip & progress to next level until getting a tail.

while inserting

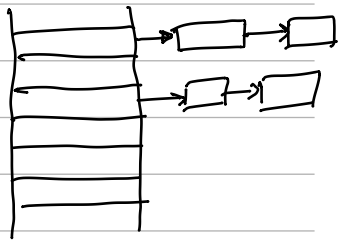



backward search:

going left : Tail has occured.
going up : Head has occured
flipping coin $t$ times  no. of head + no. of tail
$\equiv$ searching going down & going right

## Universal Hashing & Perfect Hashing:

$u$: no. of possible keys
$n$: no. of elements present in Hash
$m$: no. of slots in Table
$h: \{0,1,\dots u-1\} \to \{0,1,\dots m-1\}$



hashing with chaining; $\Theta(1+\alpha)$

Ideally Probability $(h(k_1) = h(k_2)) = 1/m$
$k_1 \neq k_2$

To guarantee perfect hashing –
assume keys are random (average case)

- Universal hashing
- Perfect hashing (guarantee no conflict, best if you exclude insert & delete; e.g. Word Dictionary Oxford)

→ Universal :
- choose hash fn $h$ randomly from the family of hash fn $H$.
- Probability $\{h(k) = h(k')\} \leq 1/m$
  $h \in H$

↳ then it can be proved (in lecture) search, delete, insert is $O(1+\alpha)$ & $\alpha$ doesn't depend on input, but on the Universal Hash family.

## ○ Dot product hashing :-

→ Key $k = \langle k_0, k_1, k_2 \cdots k_{r-1} \rangle$ $\quad 0 \leq k_i < m$

Key is of $r$ digits.

$U = m^r$ is a digit for $r$ = integer & $m$ is prime number.

for another key $a = \langle a_0, a_1, \cdots a_{r-1} \rangle$

define $h_a(k) = (a \cdot k) \bmod m$

$$= \sum_{i=0}^{r-1} a_i k_i \bmod m$$

Then $H = \{ h_a \mid a \in \{0, 1, \cdots u-1\} \}$

$h_{ab}(k) = [(a \cdot k + b) \bmod p]$ $\quad p > m$

$H = \{ h_{ab} \mid a, b \in \{0, \cdots u-1\} \}$

## ∅ Perfect hashing :-

given $n$ keys

- $O(1)$ worst case time for search
- $O(n)$ " " space
- To build this data structure Polynomial.

# Augmentation

↳ Take existing data structure & update to do extra stuff.

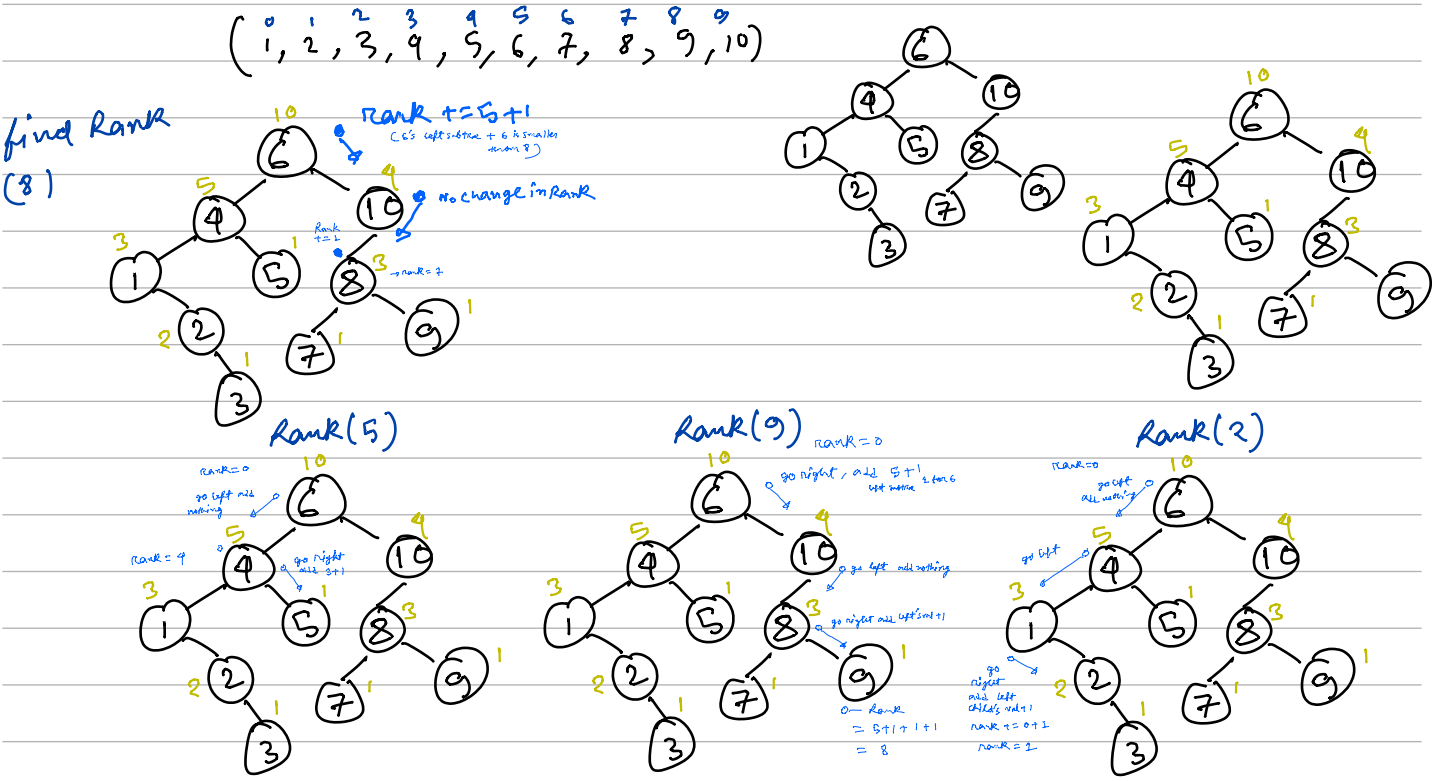
## Order statistic tree :-

insert, delete, successor, rank of element, element of specific Rank

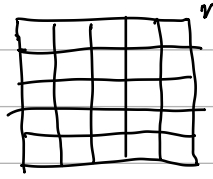
↳ Normal Tree +
Some function to easily calculate
total no. of Descendent / total no. of node in Subtree

Eg: 6, 9, 1, 2, 5, 10, 3, 8, 7, 0

(1, 2, 3, 9, 5, 6, 7, 8, 9, 10)
0 1 2 3 4 5 6 7 8 9

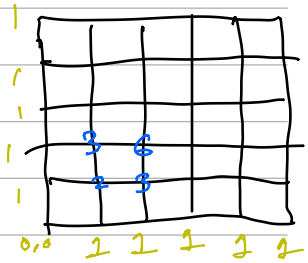


# ⇨ Dynamic Programming

Eg. 

how many distinct paths.
you can move right & top.

distinct path from 0,0 to m,n




Eg. Coin Change : $O(Nm)$ m : no. of coins
↳ same as Knapsack Problem.
# Knapsack is NP-Complete.

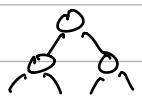
E.g Erect Blocks : n blocks $l_i, w_i, h_i$
stack on top of other s.t. $l_j < l_i$ & $w_j < w_i$ (can't rotate)

Take the tallest Greedy won't work

Similar problem to Weighted Interval Scheduling.

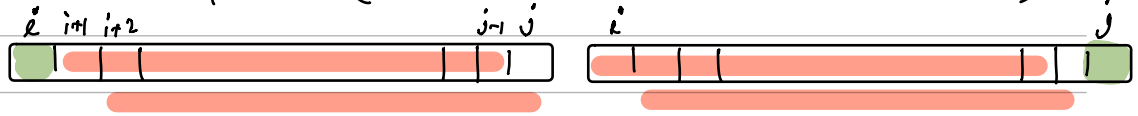
## ⓪ Longest Pallindromic Sequence

radar.

ϕ generate optimal BST :- 
node $i$ has weight $i$
minimize $\Sigma(\text{depth}_i \times w_i)$

ϕ optimal game strategy :-

$$v(i,j) = \max \left\{ \begin{array}{l} \min\{v(i+1,j-1),\ v(i+2,j)\} + v_i, \\ \min\{v(i+2,j),\ v(i+1,j-1)\} + v_j \end{array} \right\}$$




Red is enemy's zone; hope you'll get minimum from it.

# All pairs Shortest Path in a Graph :-

| graph | SSSP Algo (single source Shortest Path) | |
|---|---|---|
| unweighted | BFS $(V+E)$ | |
| non -ve edge wt. | Dijkstra $V \log V + E$ | |
| general | Bellman Ford $O(VE)$ | $V^V$, $V^3$ depending on $E \simeq V, V^2$ |
| DAG. | topological sort + 1 round B.F. $O(V+E)$ | |
| general | Johnson's | $O(V^2 \log V + VE)$ <br> $O(V^3)$ max for $E = O(V^2)$ |

DP is basically a DAG problem

① subproblems ② guessing ③ Recurrence
④ Acyclic (Topo Sort) ⑤ Solve Original problem as big Subproblem.

⇒ all pair Shortest path problem by DP I

$$d_{uv}^{(m)} = \min\left(d_{ux}^{(m-1)} + w(x,v)\right) \quad + d_{uu} = 0 \text{ if there's no path } \infty.$$

sol$^n$ for i in range $(0, m)$
  for u
    for v
      for x
} $O(V^4)$

↳ This problem can be transformed into a matrix Multipl.

$D^{(m)} = D^{(m-1)} \odot W = D^{(m)}$ ⊙ → addition
⊕ → min operation.

$D = \{d_{ij}\}$ $\{w_{ij}\}$

$D^m = (D^{m-2} \odot W) \odot W \ldots$ $O(V^4)$

⇒ All pair SP by Floyed Warshall DP II

going from u to v either take k node or don't take

$$c_{uv}^{k} = \min\{ c_{uv}^{k-1}, c_{uk}^{k-1} + c_{kv}^{k-1} \}$$

Dont take  Take k.

$O(V^3)$

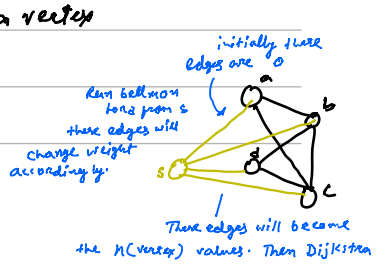
Johnson's Algo :-

① assign weight to node ; Define a f$^n$ $h : V \to \mathbb{R}$ which gives a number to a vertex

$w_h(u,v) = w_{uv} + h(u) - h(v) \geq 0$

② Run Dijkstra

③ $\delta(u,v) = \delta_h(u,v) - h(u)$

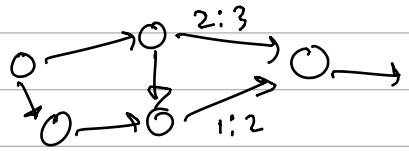


# ⇨ Greedy

Prim's & Kruskal's

---

# ⇨ Incremental Improvement :-




weight & flow

assumption : self flow not allowed.


⇒ is converted to

# +ve flow ≠ net flow. if this is allowed

Formal def$^n$ ; A flow on $G_r$ is a $f^n$ $f : V \times V \to R$

1) that satisfy $f(u,v) \leq c(u,v)$ ← capacity of edge
2) $\sum_{v \in V} f(u,v) = 0$ ← flow conservation ; ∀ $u \neq s,t$ source & sink.
3) $f(u,v) = -f(v,u)$

Properties :-

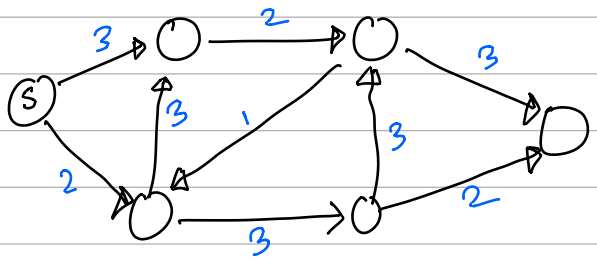
$f(x,x) = 0$ $\quad f(a,b) + f(b,a) = 0$

$f(X \cup Y, Z) = f(X,Z) + f(Y,Z)$ if $X \cap Y = \phi$

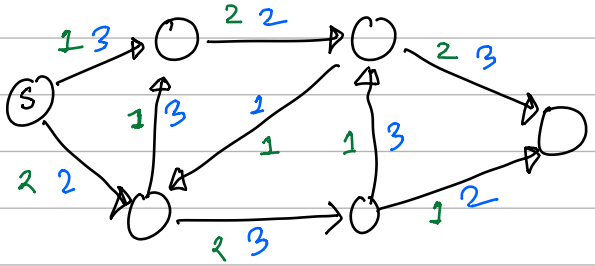
A cut $(S,T)$ of a flow network $G_r(V,E)$ is a partition of $V$ s.t. $s \in S$ & $t \in T$,

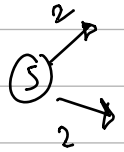
If $f$ is a flow on $G_r$, then the flow across the cut is $f(S,T)$

# Flow:




if no edge capacity = 0




max flow for this is 4.

---

Proof: $|f| = f(V, t)$ $|f| = \sum_{v \in V} f(s, v) = f(s, V)$

means flow is what is pushed in into the sink.

Impl. Summation.

$$|f| = f(s, V) = f(V, V) - f(V - s, V)$$

$$= f(V, V - s)$$

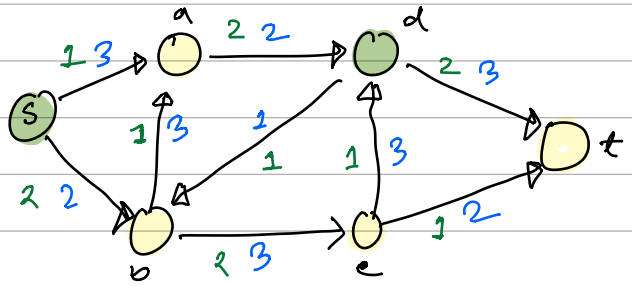
$$= f(V, t) + f(V, V - s - t)$$

$f(V, V) = 0$ coz this is all vtx except source & sink hence their total flow = 0

Cut :- $f(S, T)$

assume $S \to s, d$

$T \to a, b, c, t$




so, $f(S, T) = \underset{sa}{(2+2)} + \underset{da}{(-2} + \underset{db}{1} - \underset{dc}{1} + \underset{dt}{2})$

check every edge where vtx from S & T are interesting

✓ Capacity of a Cut : $c(S, T) = \underset{sa}{(3+2)} + \underset{ab}{(1+3)}$

$= 9$

* Capacity of any Cut, is the max flow value across the network.

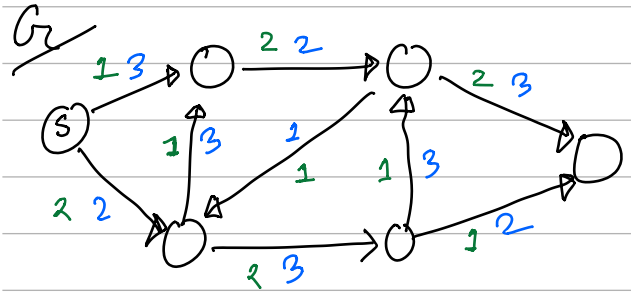
## Residual Network: $G_f(V, E_f)$

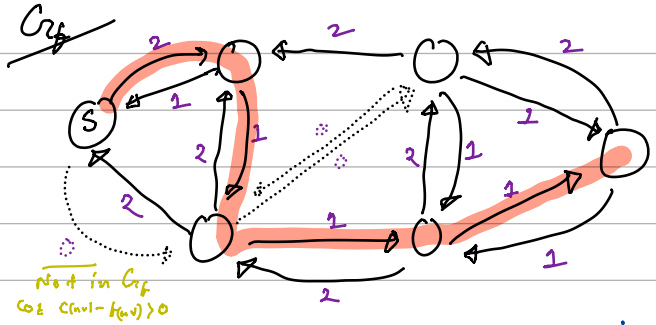
$G_f(V, E_f)$ ; Strictly +ve residual capacity.

$c_f(u,v) = c(u,v) - f(u,v) > 0$

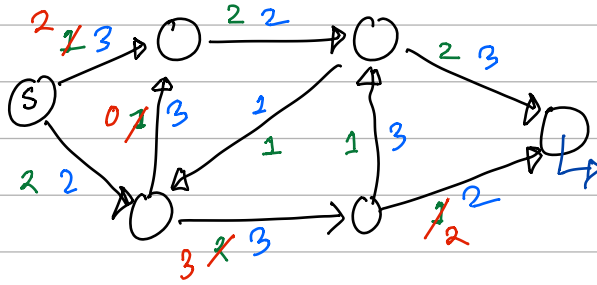
edges in $E_f$ admit more flow.

if $(v,u) \notin E$, $c(v,u) = 0$ but $f(v,u) = -f(u,v)$




According to $G_f$ this path is unutilized.




max achieved = 4

✓ augmenting path :- any path from s to t in $G_f$.

✓ residual capacity of a augmenting path is min capacity along the path

## Ford Fulkerson Algo :-

$f[u,v] \leftarrow 0 \quad \forall (u,v)$ set all u, v flow = 0

while an augmenting path in $G_f$ exist

augment f by $c_f(p)$.

## The following are equivalent ;

1. $|f| = c(S,T)$ for some cut $(S,T)$
2. f is a maximum flow
3. f admits no augmenting path.

Proof: $1 \Rightarrow 2$ since $|f| \leq c(S,T)$ for any cut $(S,T)$

hence $f = c(S,T)$ is maximum. because f can't increase.

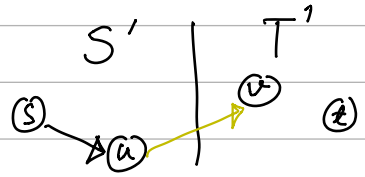
$2 \Rightarrow 3$ if $\exists$ augmenting path f can increase. contradicting f is maximal.

$3 \Rightarrow 1$ Suppose $f$ admits no augmenting path

$S' = \{u \in V : \text{there exist a path in } G_f \text{ from } s \text{ to } u\}$

$T' = V - S'$

Hence $S'$ & $T'$ is a cut




if $c_f(u,v) > 0$ than $v$ would have been in $S'$ not $T'$

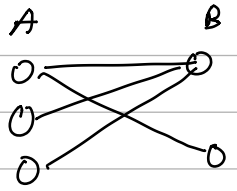
hence, $c_f(u,v) \leq 0$ ; $c_f(u,v) = 0$

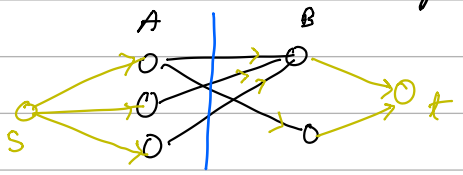
$$c_f(u,v) = c(u,v) - f(u,v) = 0$$

$$\Rightarrow \underbrace{f(u,v) = c(u,v)}_{\text{Summing over all } u \in S' \text{ \& } v \in T'}$$

$$f(S', T') = c(S', T')$$

Application/eg. ① bipartite graph : assign which A to which B.



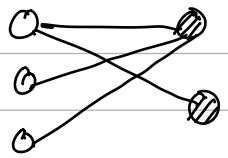




max flow cut from s to t.

② max matching or min Cover

↓

min no. of node to cover s.t. all node have covered neighbor or covered.



P – Polynomial

NP – Non deterministic Polynomial (can be checked in Polynomial time)

X is NP-Hard – every problem $Y \in NP$ reduces to X

X is NP Complete if $X \in NP$ & $X \in NP$-Hard.


Reduction: In polynomial time convert Problem A's input to equivalent Problem B's input.

$A \rightarrow B$

– if $B \in P$ then $A \in P$

– if $B \in NP$ then $A \in NP$.

# Distributed Systems

- **Distributed Algorithms:**
  Algorithm that run on networked processor or processors that share memory.

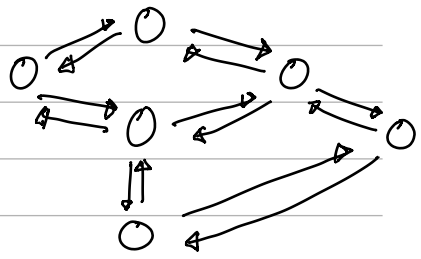
∅ They Solve many kind of Problems:
- ✓ Communication
- ✓ Data management
- ✓ Resource Management
- ✓ Synchronization etc ...

∅ They work in difficult settings:
- ✓ Concurrent activity at many processing locations.
- ✓ Uncertainity of timing, order of events, inputs.
- ✓ Failure & recovery of processors & channels.

- 2 most common distributed Algorithms:-
  1. Synchronous Distributed Algorithm
     Eg. ✓ Leader Election
     - ✓ Maximal Independent Set
     - ✓ Breadth-first Spanning Tree.
     - ✓ Shortest Path Trees.

  2. Asynchronous Distributed Algorithm.
     - ✓ Breadth-first Spanning Tree
     - ✓ Shortest path Trees.

Distributed Networks are Based on Undirected Graphs:



$n = |V|$

$\Gamma(u)$ = set of neighbors of node $u$

$deg(u) = |\Gamma(u)|$

- Associate a process with each graph vertex
  - → an infinite-state automaton.
  - → Sometimes refer to processes on vertices or nodes.
- Associate 2 directed communication channels with each edge.

## Synchronous Network Model:

- Processes at node communicate via messages. (undirected Graph)
- Each process has some input & output ports.
  - ✓ Doesn't know who its ports' chanels connect to
  - ✓ Knows the ports only by local names $1, 2, \ldots K$. $K = deg(u)$
- Processes need not be distingeishable ...
  - ✓ They need not have unique Identifiers.
  - ✓ Only Know how many ports they connect to & theirs local names.

## Execution:-

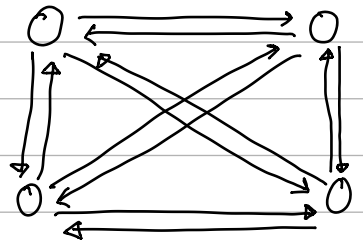
- An Algorithm executes in synchronous Rounds.
- Depending on its state, process determines which message to send on all its ports.
  - — atmost 1 messagee per port per roeund.
- message gets put into chanel & delivered to other end
- Then each process computes a new state based on its old state and Arriving Message.

* Ignore: Local computation cost (time/space complexity) just consider the no. of rounds, no. of messages / no. of Bits in message.

## Problem : Elect a Leader

given a graph chose the process which will be leader i.e. who will decide the communication / which message to send etc.

∅ simple case : Clique Network.
(all node connected to all other)



**Theorem 1** :

If Gr(V, E) is a n vertex Clique.
There exist no Algorithm consisting of Deterministic, Indistinguishible processes that is garanteed to elect a leader in Gr.

proof : 2 node clique  if you write an algorithm to select self as a leader and run on the network it will run indefinitely on both the nodes.

proof by Contradiction : lets assume an algorithm can give a leader. after each round All nodes' state will be the same & the leader node will round from 1st node to $n^{th}$ node.

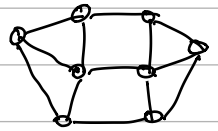
# Deterministic Non-Distinguishable processes may not always provide result.

⇒ Possible Solutions :

need to break the symmetry using some method. In a symmetrical system there is no answer to the problem.

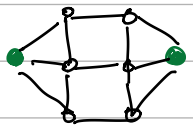
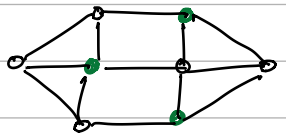
1) Give every process a UID. (largest UID is leader)
2) Randomness

## Problem : Maximal Independent Set (MIS);



select a subset of Nodes that forms maximal independent set.

independent : no. two neighbors are both in the set.
maximal : we cant add any more vtx without violating independent.

e.g. :  , 

set of green nodes are 2 different maximal independent sets.

# another Break the Symmetry problem.

application : Fruit Fly's nervous system, some cell need to declare themselves as precursor.

## Luby's Algorithm :- (2-round phase)

- ✓ initially all cell are active
- ✓ at each phase some cell decides if they are in MIS & become inactive.
- ✓ Untill all nodes are inactive.

Round 1 :
- send a random value $r \in \{1, 2, \dots n^5\}$
- receive value from neighbors ;
- if $r >$ all received value declare your node in MIS.

Round 2 :
- if you joines MIS . announce to all neighbors.
- if you receive any MIS from your neighbor send all your neighbor you are not MIS.
- Become inactive if you have decided.

## Breadth First Spanning Tree :

* Leader / root is decided $v_0$.
* All node have UID.

output : all process should return their parent node.

Algo : each round process ;
- if process i receives message & its not marked ;
  - marks itself.
  - marks the message sending neighbour as parent.
  - in next round sends message to it's neighbors.
- if already marked (Do nothing)

⇒ Complexity : 1) atmost D no. of rounds D = Diameter of Graph.
2) atmost O(E) no. of messages.

∅ Termination of Algo :
1) either go with at most D rounds.
2) all node will send a done message to its parent node (message sending neighbour)
(# converge casting).

# Shortest path Spanning Tree :-

output : each node (v) gives minimum distance from $v_0$ (root) to v

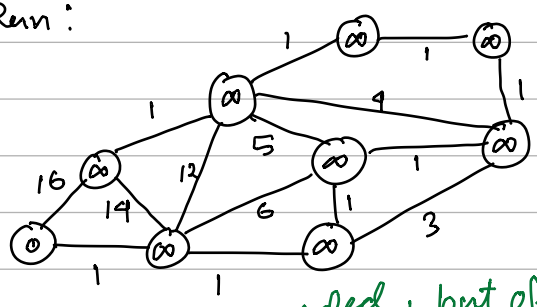
(V-1) round of Bellmon-Ford.

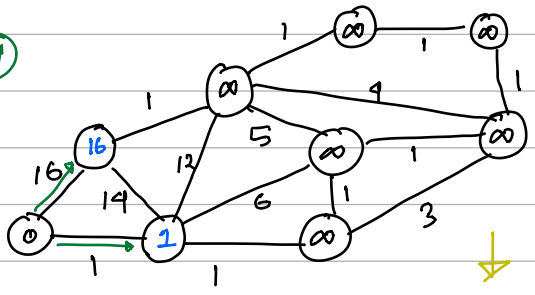
→ at each round

- Send currDist to all neighbors
- receive dj from all neighbors.

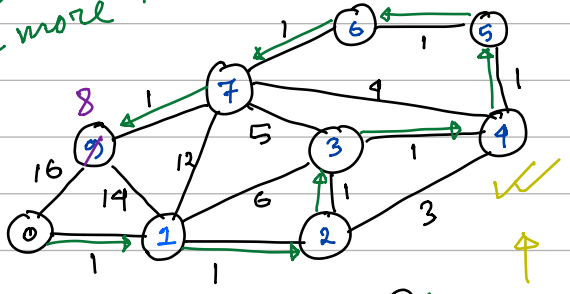
```
if ( dj + wtij < currDist)
        currDist = dj + wtij
        parent = j ;
```

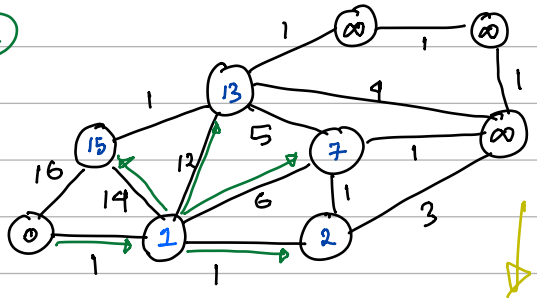
Run :



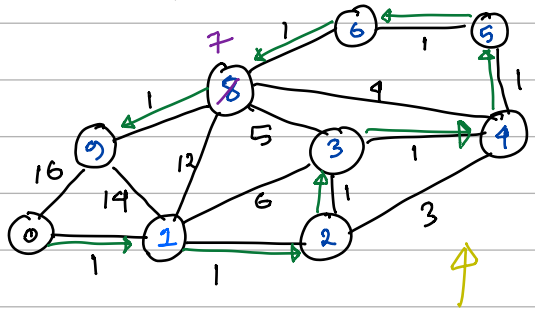



(7) #One more round needed, but okay.

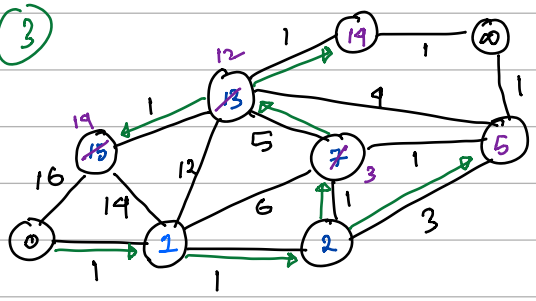




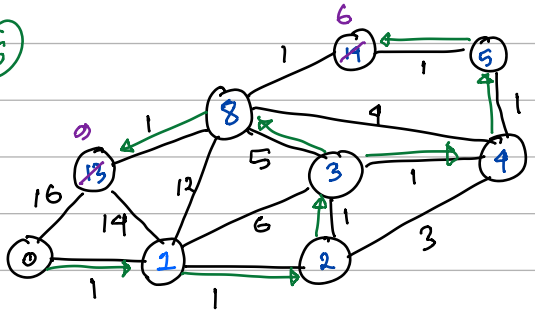

(6)



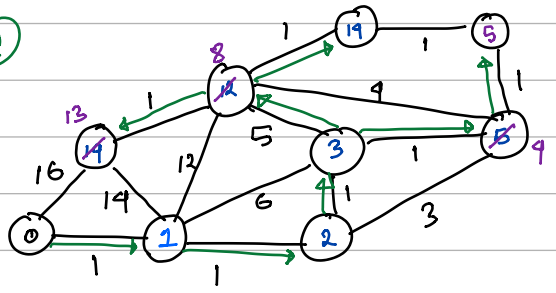
(3)



(5)



(4)



→ No rounds, process steps & message delivery happens at arbitrary times, arbitrary orders.
→ Much more non-determinism.

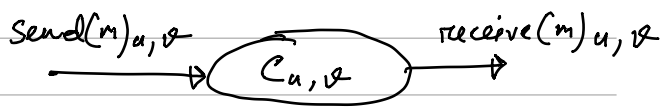
# Can't understand how they execute, must understand abstract properties of execution.

## Asynchronous Network Model

- processes are nodes of Undirected graph $G(V, E)$. communicate through messages.
- Communication Channels associate with edges
  2 chanels per edge to be specific.
  — $C_{uv}$ : chanel from $u$ to $v$.
- Each process communicate via output & input ports that that connect to its communication channels.
- Processes need not be distingeishable.

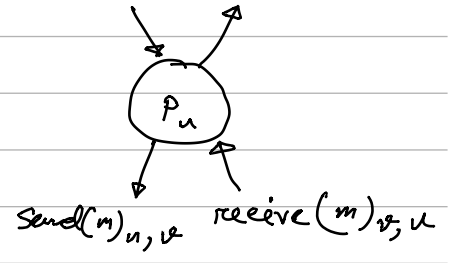
### Chanel Automaton $C_{u,v}$ :

Input action: $send(m)_{u,v}$
Output action: $receive(m)_{u,v}$




* State variable : mqueue
send(m) : add m to queue
receive(m) : m = queue.front ; queue.pop().

Overall : assume process $P_u$



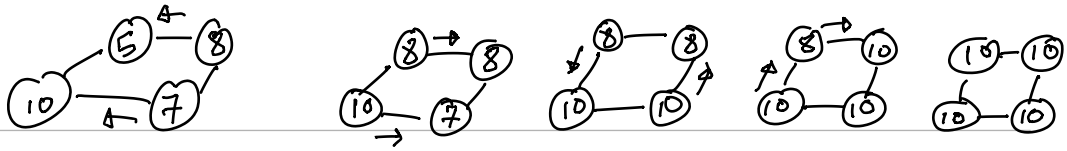

E.g. each node have send[v] = True initially


1) Receive(m)v,u
   if m > max :
     max = m
     ∀ neighbor send[v] = True.

2) if (send[v])
     send(m)u,v
     send[v] = false.

(find maximum in the graph)

Dry run



message Complexity: $O(n|E|)$

all process send to all other process.

Time Complexity: Not obvious because there's no round.

## Ø Synchronous Algo for Parent finding.

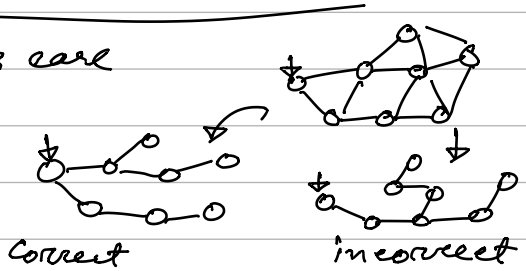
### Process Automaton $P_u$

- Input actions: $receive(search)_{v,u}$
- Output actions: $send(search)_{u,v}$; $parent(v)_u$
- State variables:
  - $parent$: $\Gamma(u) \cup \{\perp\}$, initially $\perp$
  - $reported$: Boolean, initially false
  - For every $v \in \Gamma(u)$:
    - $send(v) \in \{search, \perp\}$, initially $search$ if $u = v_0$, else $\perp$
- Transitions:
  - $receive(search)_{v,u}$
    - Effect: if $u \neq v_0$ and $parent = \perp$ then
      - $parent := v$
      - for every $w$, $send(w) := search$

### Process Automaton $P_u$

- Transitions:
  - $receive(search)_{v,u}$
    - Effect: if $u \neq v_0$ and $parent = \perp$ then
      - $parent := v$
      - for every $w$, $send(w) := search$
  - $send(search)_{u,v}$
    - Precondition: $send(v) = search$
    - Effect: $send(v) := \perp$
  - $parent(v)_u$
    - Precondition: $parent = v$ and $reported = false$
    - Effect: $reported := true$

won't work for asynchronous case




# To implement this in Asynchronous:
we can do something like shortest path in Bellman Ford.
i.e. number of hop from Root & relax it when found a better answer.

### Process Automaton $P_u$

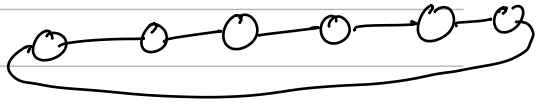
- Input actions: $receive(m)_{v,u}$, $m$ a nonnegative integer
- Output actions: $send(m)_{u,v}$, $m$ a nonnegative integer
- State variables:
  - $parent$: $\Gamma(u) \cup \{\perp\}$, initially $\perp$
  - $dist \in N \cup \{\infty\}$, initially 0 if $u = v_0$, $\infty$ otherwise
  - For every $v \in \Gamma(u)$:
    - $send(v)$, a FIFO queue of $N$, initially $(0)$ if $u = v_0$, else empty
- Transitions:
  - $receive(m)_{v,u}$
    - Effect: if $m + 1 < dist$ then
      - $dist := m + 1$
      - $parent := v$
      - for every $w$, add $dist$ to $send(w)$

### Process Automaton $P_u$

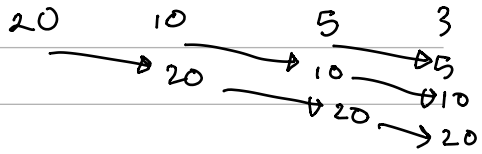
- Transitions:
  - $receive(m)_{v,u}$
    - Effect: if $m + 1 < dist$ then
      - $dist := m + 1$
      - $parent := v$
      - for every $w$, add $m + 1$ to $send(w)$
  - $send(m)_{u,v}$
    - Precondition: $m = \text{head}(send(v))$
    - Effect: remove head of $send(v)$
- No terminating actions...

# Termination: Using Done message.

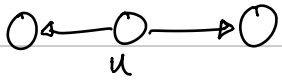
Q Problem 1 : Nodes in Circular Configuration ;
find max node.



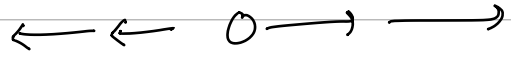
if normal Distributed algorithm applied
worst case $O(n^2)$ messages passing




optimization :

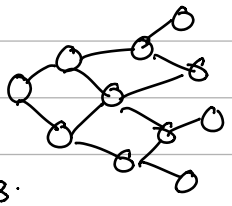



node u



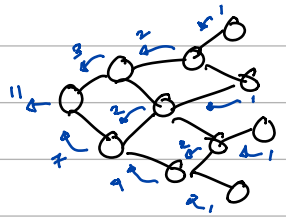
will check whether its local maxima or not if it is local maxima then it will forward message further.

Q. Count no. of Nodes



approach 1 : Root collects all unique UIDs then count no. of nodes.

approach 2 :
- ✓ Generat Spanning tree
- ✓ child report to its parent no. of nodes in subtree.



# For this question it's not needed the minimum ST / BFS ST.
any spanning Tree returning count of subtree to parent will work.

This algorithm is Asynchronous Correct.