Celebrity Problem (Reducing Problem Size)

A knows B ?
- Yes: A is not a celebrity
- No: B is not a celebrity

$\Rightarrow$ gets rid of 1 person

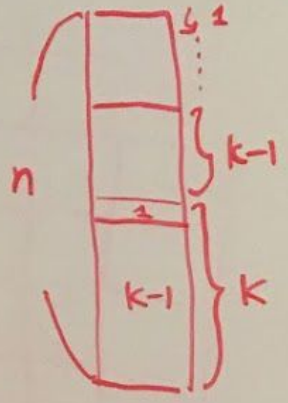
Repeat the process $n-1$ times and left with one candidate

Ask $n-1$ people if they know this person

Ask this person if he/she knows $n-1$ people

$\Rightarrow$ We know if this person is a celebrity or not

Two-Egg Problem (Distributing Worst Case)

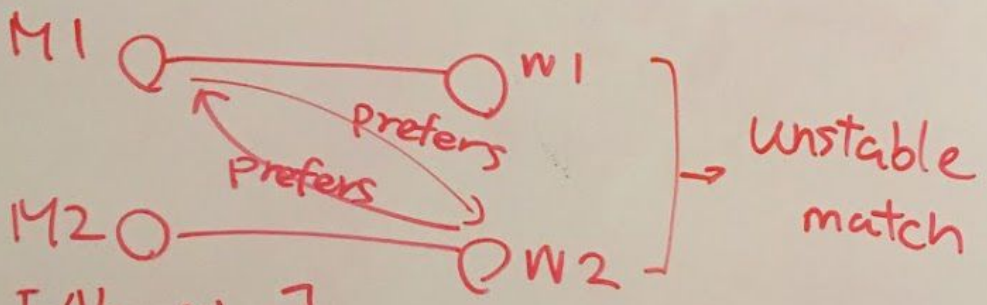


$$\frac{1}{2}k(k+1) = n$$

$$k^2+k-2n = 0$$

$$k = \frac{-1 \pm \sqrt{1+8n}}{2}$$

Stable Matching Problem



[Algorithm]

While there's a man who is free and has not proposed to all the women in his list:

Arbitrary pick such a man m

Check the woman w next in his list

m propose to w

w accepts the proposal if m is higher in her preference list of men than her current engaged man m'

Stable Match Problem Property

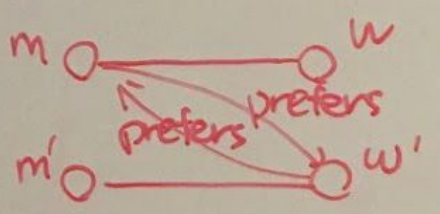
1. w remains engaged from the point she receives the first proposal
2. The time complexity of this algorithm is $O(n^2)$. Every man consumes all his preference list at the worst case.
3. If m is free at some point of the algorithm, then there's a woman to whom he has not yet proposed to.

   [proof] Suppose m is free but he's already proposed to every woman. They by (1), all women are engaged.

   Then n men are engaged. This contradicts that m is free. □

4. The termination of algorithm results in stable matching

(proof) WTS unstable matching can't happen



Suppose on the contrary,
m matches w
m' matches w'
m prefers w'
w' prefers m

If m propose to w' before w:

Then m was rejected by w'

i.e. $\exists m''$ s.t. $m'' > m$ in w''s preference list

Then either $m'' = m'$ or $m' > m''$

Either way, this contradicts that $m > m'$

Else:

$w > w'$ for m's preference list

which contradicts that $w' > w$

Interval Scheduling (Greedy Algorithm)

[Algorithm] Pick intervals that end first

Throw away overlapping intervals when we pick an interval

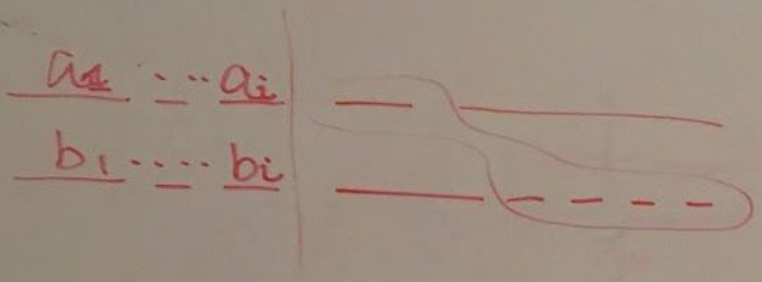
If intervals are not sorted, then it takes $O(n \log n)$

Otherwise, if sorted, $O(n)$

[Proof] Greedy algorithm "stays ahead" of any other solution.

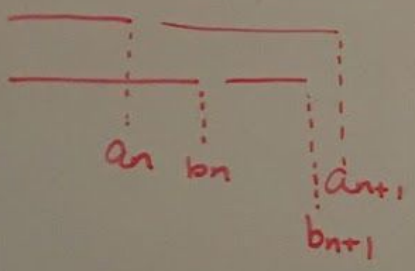
Claim: a_n always ends same or before than b_n.

Base Case: Clearly $a_1 \leq b_1$ because we pick interval that ends first.



Suppose $a_n \leq b_n$ is true

WTS $a_{n+1} \leq b_{n+1}$



Suppose on the contrary

$a_{n+1} > b_{n+1}$

Then our algorithm would have picked b_{n+1} because $a_n \leq b_n$ and b_{n+1} started after b_n

Thus $a_{n+1} \leq b_{n+1}$

e pick

Majority

[Algorithm]

Find two different candidates

Get rid of them

Left with one candidate

Check if this candidate is indeed a majority

[proof]

Majority element occupies at least $\frac{n}{2}+1$ seats

When we get rid of two different people, either none or one of them is majority element. Now, there's at least $\frac{n}{2}$ majority elements out of $n-2$ seats. Since $\frac{n}{2} \geq \frac{n-2}{2}+1 = \frac{n}{2}$, this element is still a majority.

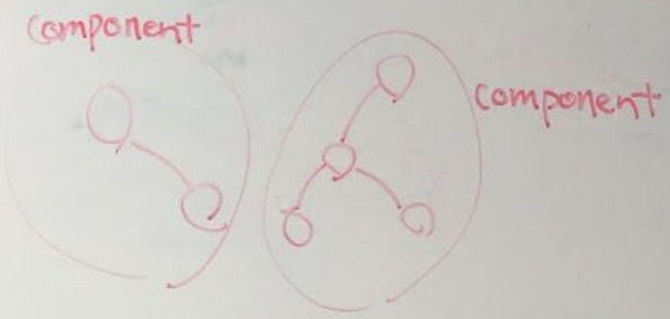
BFS : traverses vertices using queue
(FIFO)

DFS: traverses vertices using stack
(FILO)

Connected graph: You can reach all vertices
from any vertex

Unconnected graph: You can not reach all vertices
from a vertex

Component : A set of vertices reachable from one another



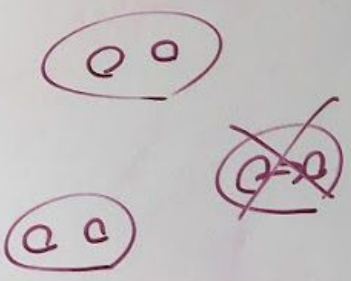
We need at least $n-1$ edges to have only 1 component

Run Time of BFS/DFS is $O(V+E)$

We need to process all vertices and edges

Finding all shortest paths can take exponetial Time

How to partition a graph s.t.
there's no edge within every group

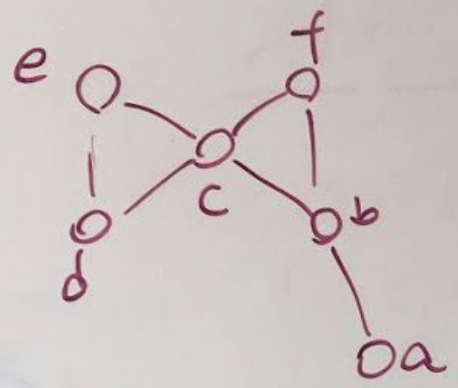


[Algorithm]

1. Add all sources (i.e. vertices w/o incoming edges)
   to a group
2. Delete sources
3. Repeat 1 and 2

Given undirected graph.

find a vertex whose removal will not disconnect the graph



e.g. Removal of c will disconnect the graph

[Algorithm]

1. Perform BFS from arbitrary vertex
2. Arbitrary choose a leaf node

[Idea]

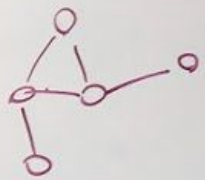
To keep a graph connected, every vertex is reachable to any vertex. Removing a leaf node does not break the paths from source to any vertices. Since the edges are undirected, every vertex is still reachable to any vertex.

Let d_i denote degrees (# of incoming/outgoing edges) of ith node. Prove $\sum d_i = 2(n-1)$

Base Case:

o degree = 0 ✓

Inductive Case: Suppose we have $2(n-1)$ degrees in total with n nodes. Now if we add a node to this graph, this new node will have degree of 1. The other node connected by this new node will increment its degree. Thus the total degree of this new graph with $n+1$ nodes is $2(n-1)+2 = 2n$ □



Two vertices in a graph are strongly connected

Then there are paths $a \to b$ and $b \to a$.

(Property)
If $u \leftrightarrow v$ and $v \leftrightarrow x$, then $u \leftrightarrow x$.

Given a directed graph, how do we know if it is strongly connected?

[Idea]
WTS v can reach every node in the graph and any node x can reach v.

[Algorithm]
Perform DFS from v
If not all nodes are visited: Return False
Perform DFS from v on Reversed Graph
If not all nodes are visited: Return False
Return True

2 colorable

Adjacent nodes must have different color

A graph is 2 colorable if it does not have a cycle.

If it has a cycle, it's still 2 colorable only if it is even cycle. Odd cycle is not 2 colorable.

Topological Sort

Assuming DAG → n-1 edges

Sources: vertices w/o incoming edges

indegree of vertex: # of incoming edges

outdegree of vertex: # of outgoing edges

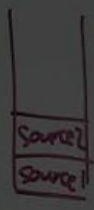
Q: How to find indegree and outdegree of all vertices?

Vertex Centric: $\underbrace{n}_{\#\text{ of } V}\ \underbrace{(n-1)}_{\text{Max \# of } E}$

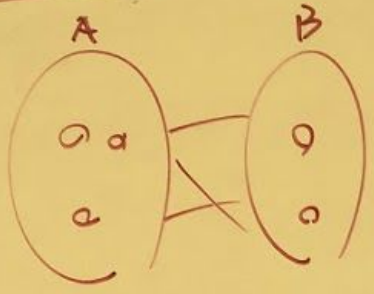
Edge Centric: $O(E)$ (loop through every edge and update vertex degree count)

[Algorithm]

1. Find indegree and outdegree of all vertices $O(E)$
2. Find sources by checking indegree of all vertices $O(n)$
3. Output a source $O(1)$
4. Update indegree count of removed source $O(E)$ throughout the algorithm
5. Repeat 3 and 4



Bipartile graph



No edges in groups

Edges are only between A and B

$\Leftrightarrow$ 2 colorable Problem

because ajacent nodes have different color (i.e. group)

Dykstra's Algorithm

Shortest paths from a source to every other vertex

[Algorithm]

1 Add source to a group

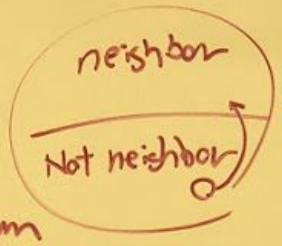
2. Pick neighbor vertex of the group with least weight and finalize the shortest path to the vertex

3. Add the vertex to the group

4. Repeat 2 and 3

[Time Complexity]

Vertex Centric :



Move a vertex from bottom to top takes at least $O(n)$

$O(n^2)$ in total

Edge Centr

Use n
neighbo

Findin

Add/U

Note

Thus

⇒ O

Edge Centric

Use min heap to keep track of neighbors with weights

Findins neighbor with min weight $O(1)$

Add/Update neighbors $O(e \log n)$ throughout the algorithm

- e: # of add/update in total
- $\log n$: add/update

Note $e = n^2$ at most

Thus $\ln e = 2 \ln n \Leftrightarrow O(\ln e) = O(\ln n)$

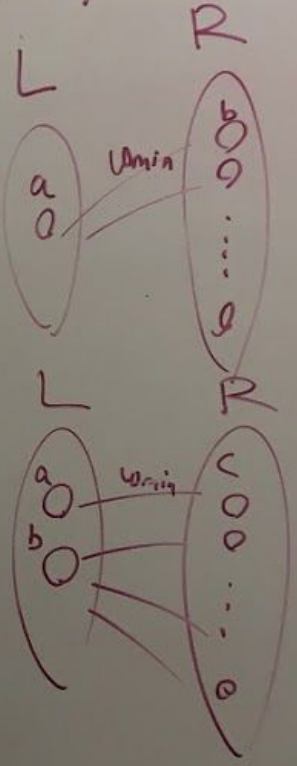
$\Rightarrow O(e \log e)$

Minimum Spanning Tree

- Minimum total weight
- Has exactly $n-1$ edges

Prim's Algorithm (Vertex Centric)

1. Arbitrary pick a vertex a.
2. Check all neighbors of a
3. Choose Wmin
4. Mark vertex b to L
5. Check all edges from L to R
6. Choose next Wmin
7. Mark vertex c to L
8. Repeat from 5 to 7



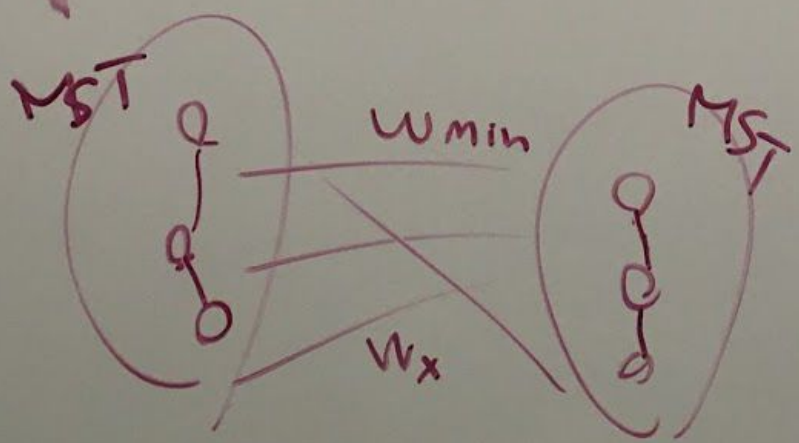
Time complexity is $O(n^2)$
(For each vertex, we need to check the vertices it is connected to)

Or if we use heap to keep track of min weight, it takes

$O(e \log e)$

[proof of Optimality]

Consider partition of graph G



Claim: $W_{min} \in MST$

Proof of Claim: Suppose $w_{min} \notin MST$

Then $\exists w_x$ s.t. $w_x \in MST$

Notice $w_x > w_{min}$

Thus MST w/ w_x > MST w/ w_{min}

$\Rightarrow$ MST w/ w_x can't be a MST

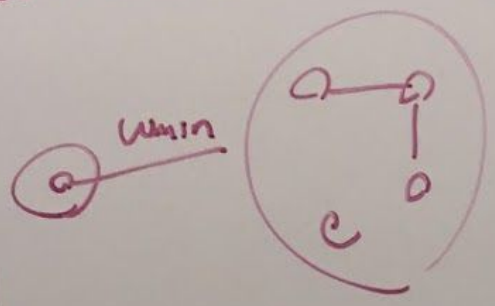
Difference between Dykstra and MST

Dyskstra $\longrightarrow$ nearest to given source

MST $\longrightarrow$ nearest to given partition

Kruskal's Algorithm

[Idea] Pick a node and put that into a partition. Put all other nodes into another partition



[Algorithm]

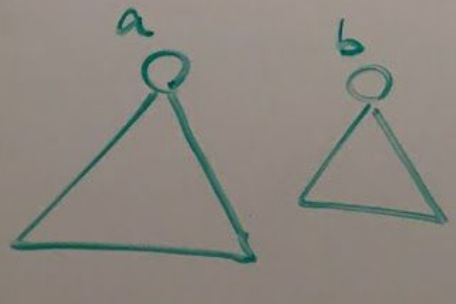
1. Sort edges in ascending order by weight
2. Pick edge and add it to MST if it does not create a cycle.
3. Repeat 2 until we find n-1 edges for MST

Union – Find Problem

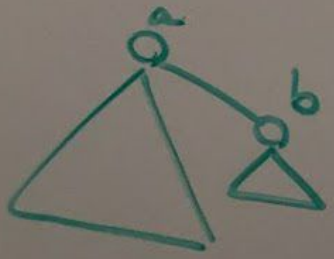
Objective1: We want to know if two elements are in the same group

Objective2: We want to union two groups into one group

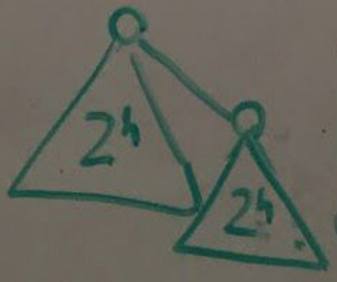
[Idea] If root is same, then they are in the same group.



Assume
$h \approx \log(\#\text{ nodes})$



If size(a) > size(b)
$h \approx \log(\#\text{ nodes})$ is maintained



If size(a) = size(b)
$h \approx \log(\#\text{ nodes})$ is maintained

Time Complexity

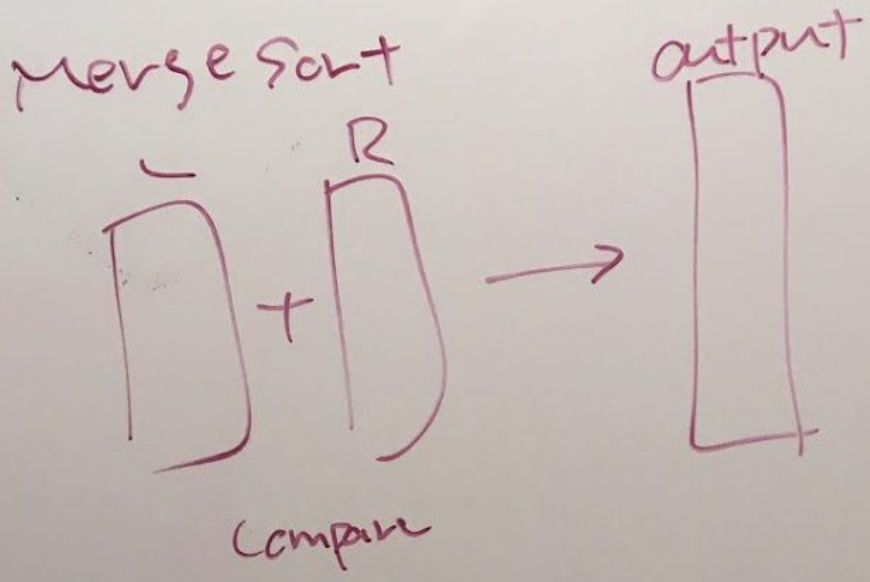
Union : $O(1)$? Find : $O(\log n)$

Kruskal's Algorithm

[Time Complexity]

Sort edges	$O(e \log e)$
e unions	$O(e)$
+ e find	$O(e \log n) = O(e \log e)$
	$O(e \log e)$

Divide and Conquer



Focus on the output

For each entry, we need one comparison

$$T(n) = T(\tfrac{n}{2}) + T(\tfrac{n}{2}) + n$$

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

Binary Search

$$T(n) = T(\tfrac{n}{2}) + C = O(\log n)$$

Clustering

Given # of clusters k,
Want to group vertices into clusters
s.t. distance between clusters is maximized
i.e. $\max_{ij} \min d_{ij}$ where d_{ij} denotes
distance between cluster i and cluster j

[Approach]
Run Kruskal's Algorithm
until it's there's k clusters

$O(e \log e)$

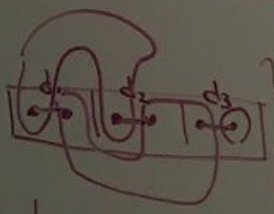
[U

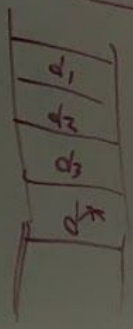
[proof]

Suppose Kruskal's Algorith gives clusters $C_1, C_2, \ldots, C_k$.

Let d^* be the minimum distance between two clusters. (i.e. $d^* = \min_{i,j} d_{ij}$)

Suppose on the contrary, clusters $C_1', C_2', \ldots, C_k'$ are the optimal solution and their minimum distance $d^{*'}$ greater than d^*. Then $\exists C_k$ s.t a vertex $v_1 \in C_k$, and $v_2 \in C_k$ are in different cluster. Note the distance d^{**} between v_1 and v_2 must be smaller than d^* (i.e $d^{**} < d^*$) and $d^{*'} \leq d^{**}$. Thus $d^{*'} < d^*$. //





[Variation]

Given MST, and egdes are sorted.

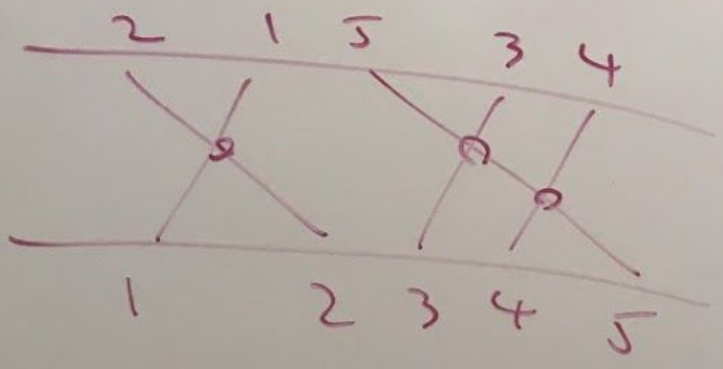
Then Remove largest edge until

k clusters. $\Rightarrow O(e)$

$< d^*$)

Inversion Counting

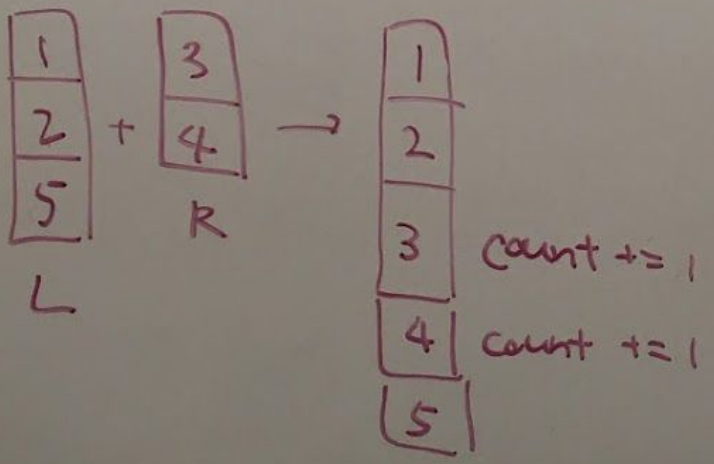
Two parallel lines



Q: Find # of crossings

[Algorithm]

Modification of Merge sort



When choosing element from R, increment # of count by # of elements left in L