ADA notes week 4

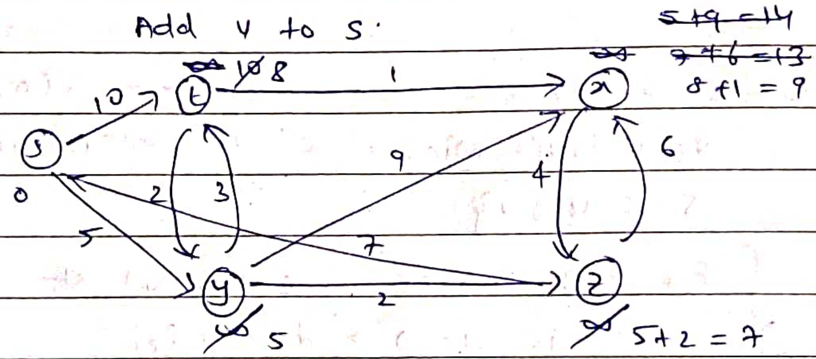
Shortest path :

Given $G = (V, E)$ with $w(u,v) \geq 0$ for every edge,
Find shortest path from $s \in V$ to $V - s$.

Dijkstra's Algorithm

Initially $S = \{s\}$ and $d(s) = 0$
For all other nodes $d(s) = \infty$

while $S \neq V$
select a node $v \notin S$ with atleast one edge
from S for which
$$d(v) = \min(d(u) + l_e)$$
Add v to S.



$S = \{s\}$
$S = \{s, y\}$
$S = \{s, y, z\}$
$S = \{s, y, z, t\}$
$S = \{s, y, z, t, x\}$

Pg4 Proof of correctness.

PAGE No
DATE / /

Dijkstra's Time complexity

Considering there are as many edges as there are nodes.

Binary heap implementation → $O(m \log n)$
Binomial heap implementation → $O(m \log n)$
Fibonacci heap implementation → $O(m + n \log n)$

Dijkstra's Algo.

S = Null

$O(n)$ Initialize priority queue Q with all nodes V where $d(v)$ is key value
(All $d(v) = \infty$, except for s $d(s) = 0$)

while $S \neq V$ ← $O(n)$
 v = Extract-Min(Q) ← n operations
 $S = S \cup \{v\}$
 For each vertex $u \in Adj(v)$ ← $O(n)$
  if $d(u) > d(v) + l_{ei}$
   Decrease-key $(Q, u, d(v) + l_e)$ ← $O(n)$

→ But here we can only call every edge once.
This gives $O(n^2)$ but if every edge is called only once $O(n^2) \Rightarrow O(m)$

Initialize priority queue: $O(n)$
Max no. of Extract-min operations: $O(n)$
Max no. of decrease key operations: $O(m)$.

(*) Any Tree that covers all nodes of a graph is called spanning Tree.

no cycles

A spanning Tree with minimum total edge cost is a minimum spanning Tree (MST).

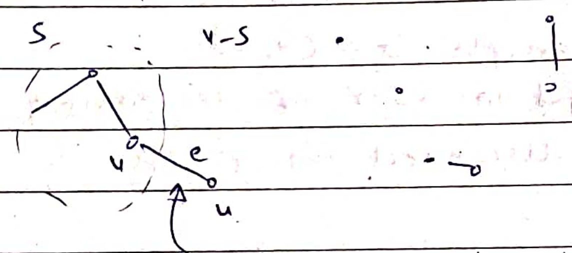
Method 1 Kruskals Algorithm.

Sort all edges in increasing order of cost.

Add edges to T as long as no cycle is formed

Fact: let S be any subset of nodes that is neither empty nor equal to V, and let edge e = (v, w) be the min cost edge with one end in S and other end in V - S. Then every MST contain the edge e.

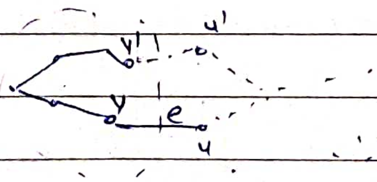
Kruskals proof of correctness:



From the above mentioned. Fact our Algo matches the Fact Hence works perfect everytime.

Hence Kruskals gives MST.

Proof of Fact



There are many edges from S to V-S 'e' is the smallest.

Proof by contradiction

∵ To avoid cycle we would remove v'u' because its cost is more than vu

∴ we would take vu instead of v'u'

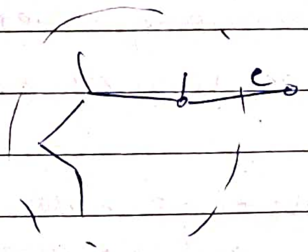
Method 2 Prims Algorithm.

Similar to Dijkstra's Algorithm, start with a node S

initially S is only root node.

At each step grow S by one node, adding the node v that minimizes the attachment cost.

Proof of correctness



Min cost between S and V-S

Fact used again

Hence prims is also optimal.

Method 3 ~~Backward Delete~~ / Reverse-Delete

Backward version of kruskals

Start with full graph $G = (V, E)$

Begin deleting edges in order of decreasing cost as long as it does not disconnect the graph.

Fact: highest cost edge in cycle cannot belong to MST.

Proof of correctness

Rev Del is removing edge i.e useless high cost edge not in MST

Hence it also finds MST.

Prims ~ Dijkstra's $\Rightarrow$ $O(m \log n)$

Kruskall

	Array implementation	pointer imple
make-set	$O(1)$	$O(1)$
Find-set	$O(1)$	$O(\log n)$
union	$O(\log n)$	$O(1)$

Find-set: Takes node and returns set it belongs to

Algo

A = NULL

For each $v \in V$
make set (v) $\Big\}$ $O(n)$

sort edges acc. to non-decreasing edge cost $\Big\}$ $O(m \log m)$

For each edge $(u, v) \in E$ in this order $\rightarrow m$
if Find-set(u) $\neq$ Find-set(v) $\rightarrow O(1)$
$A = A \cup \{(u,v)\}$
union (u, v) $\rightarrow \log n$

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

Prims vs Kruskals
$O(m \log n)$ $O(m \log m)$

worst case $m = n^2$
$\rightarrow O(m \log n^2) \Rightarrow O(m \log n)$

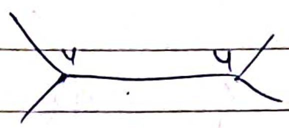
Reverse Delete

Sort edges in dec order ← $O(m \log m)$

For x in edges ← $O(m)$

$O(m+n)$ (if x does not disconnect graph.
remove(x)

$O(m^2)$



To remove edge (u, v)

we run BFS starting at v and see if we reach u.

if we reach u graph is not disconnected and we remove edge (v, u)

⊛ Clustering

Given a set of n objects

$p_1 \ldots p_n$

where $d(p_i, p_i) = 0$

$d(p_i, p_j) > 0$

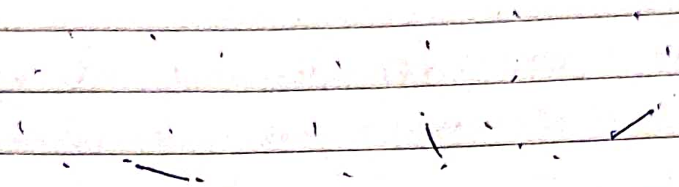
$d(p_i, p_j) = d(p_j, p_i)$

A 'k' clustering of u is partitions of u into $\underline{k}$ non-empty sets $C_1 \ldots C_k$

The spacing of a 'k' clustering is the minimum distance between any pair of points lying in different clusters.

Problem

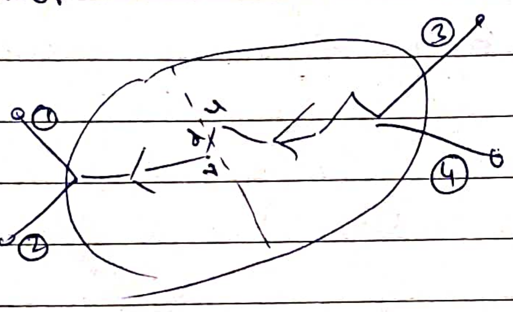
Given a set of n objects
Find k-clustering with maximum spacing.



works like prims just leave out last
'k-1' edges.

dist btwn cluster ↑
dist of points in dustes ↓

Proof that there is maximum spacing



lets say cost of edge is
'd'
our spacing is 'd*'
↑ min cost

$d < d^*$

Kruskal drops last k-1 edges ① ② ③ ④

Kruskals choose d over ① ② ③ ④ this means

$d < \underbrace{① \quad ② \quad ③ \quad ④}_{d^*}$