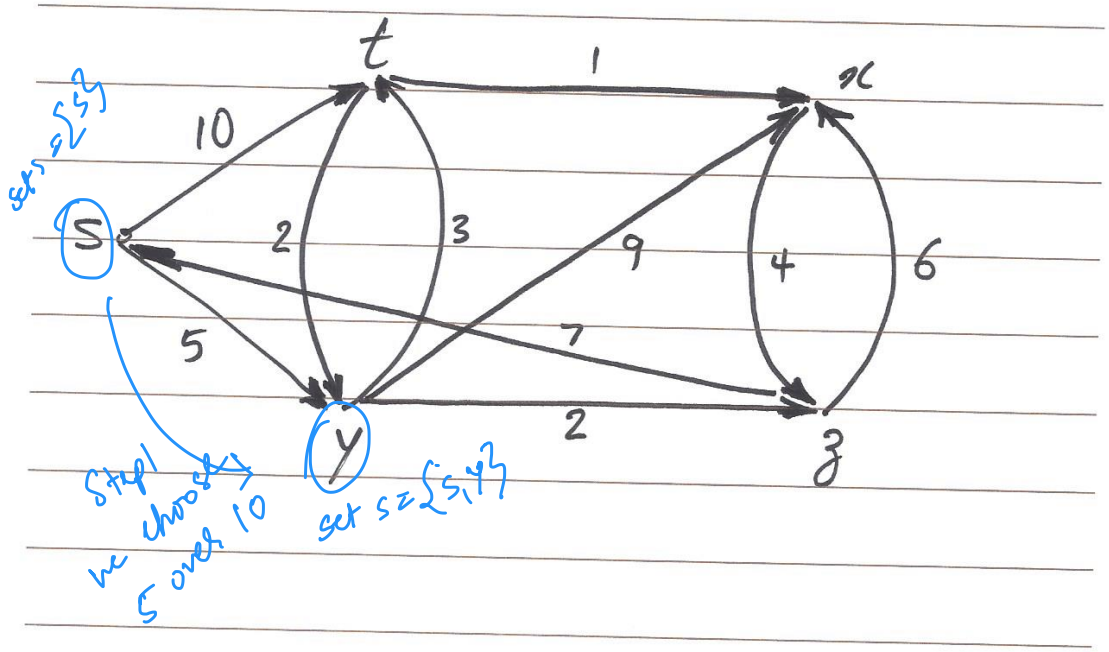
Shortest Path

Problem Statement:

Given $G = (V, E)$ with $w(u,v) \geq 0$ for each edge $(u,v) \in E$, find the shortest path from $s \in V$ to $V - \{s\}$



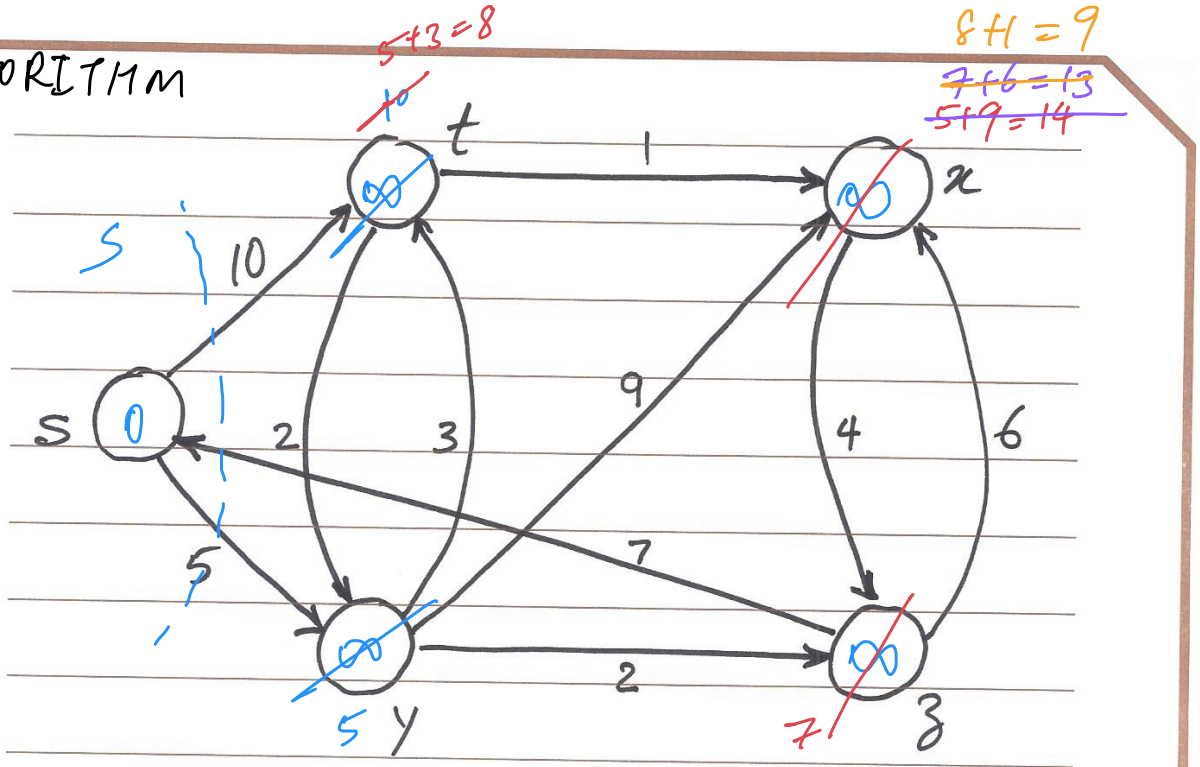
Solution

1. Start with a set S of vertices whose final shortest path we already know.

2. At each step, find a vertex $v \in V - S$ with shortest distance from s.

3. Add v to S, and repeat.

DIJKSTRA'S ALGORITHM



Initially all nodes are at ∞ distance from start

$S_1 = \{s\}$ look at neighbours of s s→y current ∞ but we can get to y from s in cost 5

$S_2 = \{s, y\}$

$S_3 = \{s, y, z\}$

$S_4 = \{s, y, z, t\}$

$S_5 = \{s, y, z, t, x\}$

By doing this we get shortest distance from s to all other nodes

To find shortest path we can keep track of parent

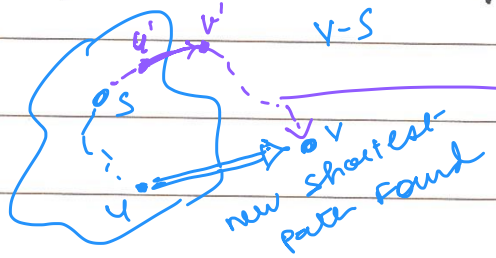
Proof of Correctness

We will prove that at each step, Dijkstra's algorithm finds the shortest path to a new node in the graph.

Proof by mathematical induction:

Base case: $|S| = 1$, $S = \{s\}$ and $d(s) = 0$

Inductive Step: Suppose the claim holds when $|S| = k$ for some $k \geq 1$. We now grow S to size $k+1$ and prove that we have found the shortest path to the new node.



parent
$p(v) = u$
consider this to be shortest path

but when we are on u'
if v would have shortest path
from u' we would have picked
up there
and there are no -ve costs
∴ the difference could not be made
up by v' to v

Implementation of Dijkstra's

Initially $S = \{s\}$ and $d(s) = 0$

for all other nodes $d(u) = \infty$

While $S \neq V$

Select a node $v \notin S$ with at least one edge from S for which

$d(v) = \min_{e(u,v): u \in S} (d(u) + l_e)$

Add v to S

endwhile

for d we will use priority queue

More Detailed Implementation of Dijkstra's

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)'s are = ∞, except for s where d(s) = 0)

While $S \neq V$ — 'n' operations

  v = Extract_Min(Q)

  S = S ∪ {v}

  for each vertex $u \in Adj(v)$ — $O(n-1)$

    if $d(u) > d(v) + l_e$:

      Decrease_key(Q, u, d(v) + l_e)

  end for

end while — $O(n)$

Decrease_key → $O(n^2)$

But we can only call every edge once

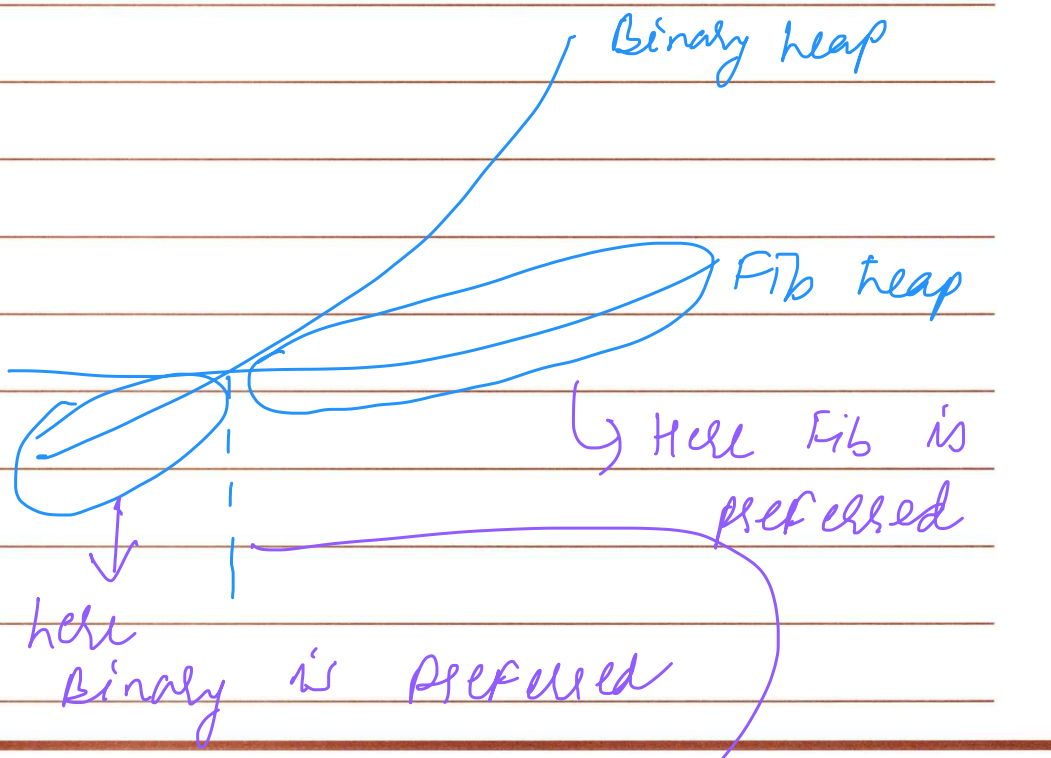
hence $O(n^2)$ can be dropped to $O(n)$

Complexity Analysis

- Initialize Priority Queue $O(n)$
- Max. no. of Extract-Min op's : $O(n)$
- Max. no. of Decrease-key op's : $O(m)$

	Binary Heap	Binomial Heap	Fibonacci Heap
n Extract mins	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
m Decrease keys	$O(m \log n)$	$O(m \log n)$	$O(m)$
assuming	there are as many edges as there are nodes		
Total	$O(m \log n)$	$O(m \log n)$	$O(m + n \log n)$
Sparse graph	$O(n \log n)$	$O(n \log n)$	$O(n \log n)$
Dense graph	$O(n^2 \log n)$	$O(n^2 \log n)$	$O(n^2)$

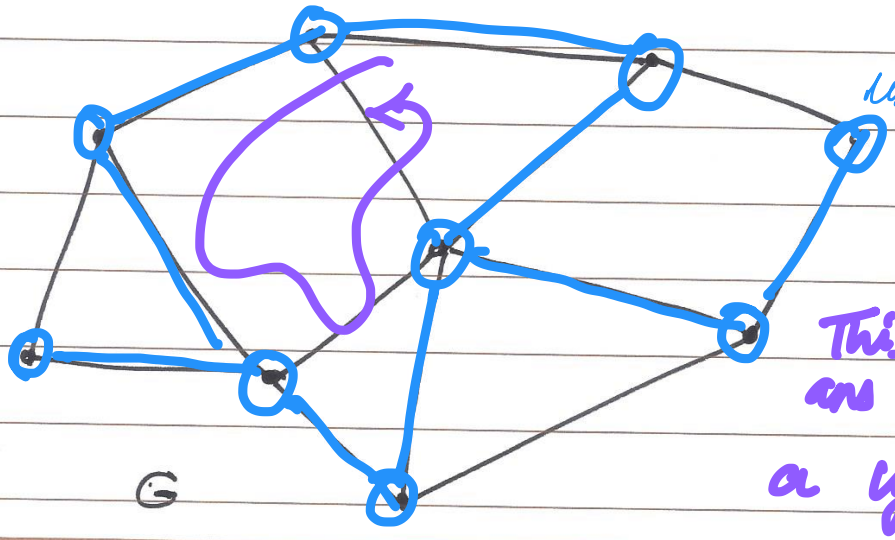
that doesn't have that many edges



But we do not know where this cut-off point is

Problem Statement

Find minimum cost network that connects all nodes in G.



G

undirected weighted graph

This cannot be one as there is a cycle

→ no cycles

Def. Any tree that covers all nodes of a graph is called a spanning tree.

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

Problem Statement

Find a MST in an undirected graph

Sol. 1: Sort all edges in increasing order of cost. Add edges to T in this order as long as it does not create a cycle. If it does, discard the edge.

Kruskal's algorithm

Sol. 2: Similar to Dijkstra's algorithm, start with a node set S (initially the root node) on which a minimum spanning tree has been constructed so far. At each step, grow S by one node, adding the node v that minimizes the attachment cost.

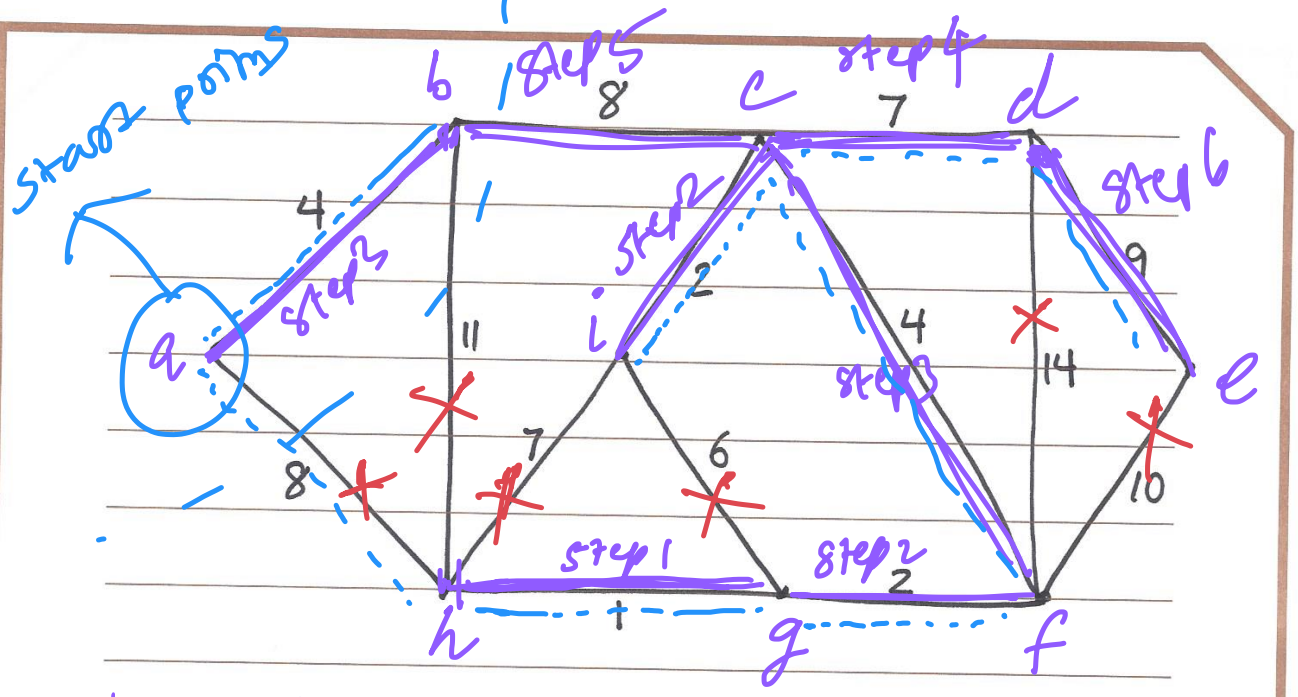
Prim's algorithm

Sol. 3: Backward version of Kruskal's.
Start with a full graph (V, E).
Begin deleting edges in order of decreasing cost as long as it does not disconnect the graph

Backward version of Kruskal

Reverse-Delete

S | V-S



Kruskals

Prims $S=\{a\}$ $S=\{a,c\}$ $S=\{a,c,h\}$

$S=\{a,b,g,h\}$ $S=\{a,c,g,h,f\}$

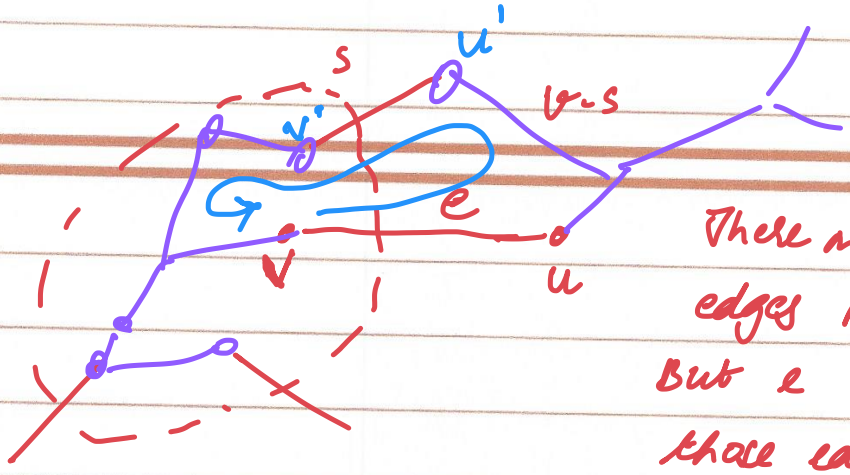
$S=\{a,b,g,h,f,c\}$ $S=\{a,c,g,h,f,c,i\}$

$S=\{a,b,g,h,f,c,i,d\}$ $S=\{a,c,g,h,f,c,i,d,e\}$

Reverse - Del

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

Considering all edge costs are unique



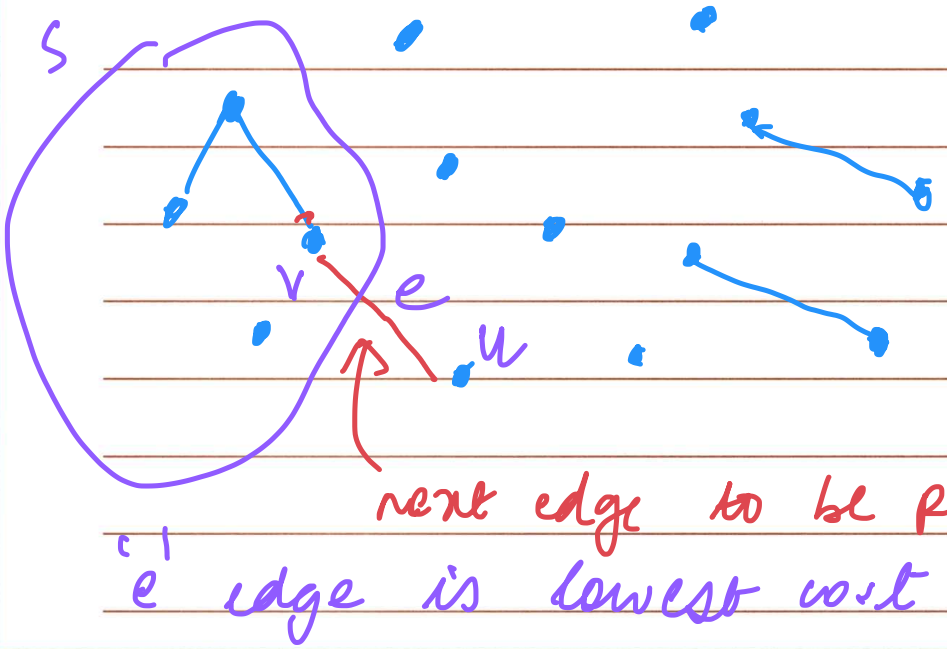
There may be many edges from s to v-s But e is smallest of those edge

PROOF OF CONTRADICTION

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

∴ we would take vu instead of v'u'

Kruskals proof of correctness



'e' edge is lowest cost edge between S to V-S

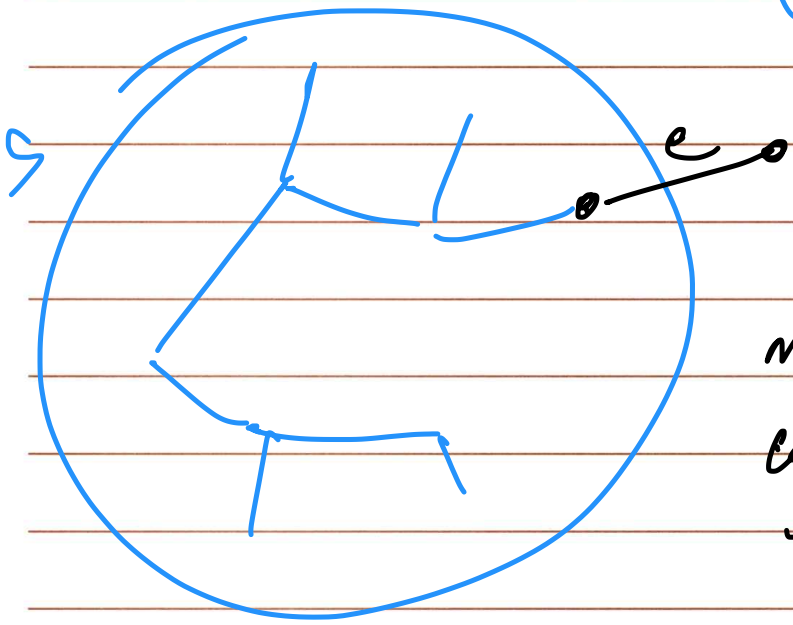
ref Page 13 Fact

works everytime

Hence Kruskal's gives min spanning Tree

Prims

V-S



min connection cost b/w S and V-S

Page 13 FACT again

Hence Prims also finds minimum spanning Tree

Rev Del

FACT: highest cost edge in cycle cannot belong to minimum spanning Tree

Rev Del is removing edge i.e useless and not in minimum spanning Tree

Hence it also finds min spanning Tree

More Detailed Implementation of ~~Dijkstra's~~ PRIMS

S = Null

Initialize priority Queue Q with all nodes V where d(v) is the key value.
(All d(v)s are $=\infty$, except for s where d(s)=0)

While $S \neq V$

v = Extract_Min (Q)

$S = S \cup \{v\}$

for each vertex $u \in Adj(v)$

if $d(u) > d(v) + le$;

Decrease_key (Q, u, $d(v) + le$)

end for

end while

Kruskal's

Create an endp set for each node
$A = \text{Null}$
Sort edges in non-decreasing order of weight
for each edge $(u,v) \in E$, taken in this order, if u & v are NOT in the same set then
$A = A \cup \{(u,v)\}$
merge the two sets
end if
endfor

Union-Find data structure

- Make set — $O(1)$ for set size = 1
- Find set — $O(1)$ or $O(\lg n)$
- Union — $O(\lg n)$ or $O(1)$

array impl. — ptr impl.

takes node and returns set it belongs to

perform union of two sets

pointer implementation for

Initially there are 'n' independent sets at end we get 1 set

1 edges have to go thru 2 diff sets and then merge 2 sets

Implementation of Kruskal's

```
A = Null
for each vertex v ∈ V
    Make-set(v)
endfor
```

$O(n)$

$O(m \log m)$

```
Sort the edges of E into non-decreasing
order of cost
```

```
for each edge (u,v) ∈ E in this order:
    if Find-set(u) ≠ Find-set(v) then
        A = A ∪ {(u,v)}
        Union(u,v)
    endif
endfor
```

$O(m \log n)$

m

1

$\log n$

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

$= O(m \log n)$

Prims

$O(m \log n)$

highest m can be n^2

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

$= O(m \log n)$

Kruskals

$O(m \log m)$

≠ every node connected to every other node

Reverse Delete

sort edges in dec order

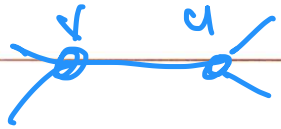
For x in edges: ← $O(m)$

if x does not disconnect graph

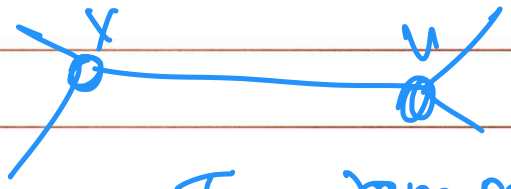
remove x

$O(m \log n)$

$O(mtr)$



$O(m^2)$



To remove v u

we run bfs starting at v and see if we reach u

if we can reach u then graph is not disconnected and we remove edge v u

Clustering

Defn Given a set U of n objects $p_1 .. p_n$ where $d(p_i, p_i) = 0$, $d(p_i, p_j) > 0$ For $p_i \neq p_j$ and $d(p_i, p_j) = d(p_j, p_i)$, a k clustering of U is a partitions of U into $\leq$ k nonempty sets $C_1 .. C_k$

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

Problem statement

Given a set of n objects as described above, find a k-clustering with maximum spacing

Kruskals

works like [illegible]

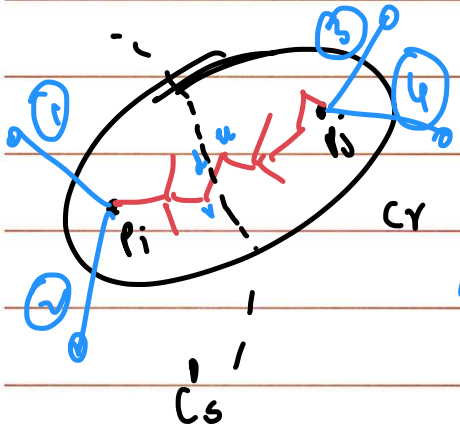
just leave out last K-1 edges

In kruskals we form sets, so these clusters are those sets. We do not take take the last 'k-1' edges hence we have k clusters and distance between them is maximum as kruskals sort the edges and selects the edges in increasing order. SO last k-1 edges are the last k-1 edges in the increasing order of the sorted edge lengths

dist btn cluster ↑

dist of points in cluster ↓

Proof that there is maximum spacing



lets say cost of edge is d

Our spacing it is d^*

minimum cost

$d < d^*$

Kruskals drops last k-1 edges those are the one's marked in blue

( )

Kruskals choose d over ①②③④

this means

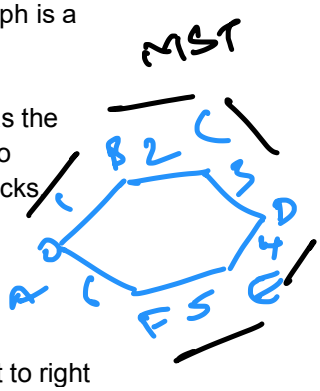
$d < ① ② ③ ④$ d^*

Discussion 4

1. Hardy decides to start running to work in San Francisco city to get in shape. He prefers a route that goes entirely uphill and then entirely downhill so that he could work up a sweat uphill and get a nice, cool breeze at the end of his run as he runs faster downhill. He starts running from his home and ends at his workplace. To guide his run, he prints out a detailed map of the roads between home and work with k intersections and m road segments (any existing road between two intersections). The length of each road segment and the elevations at every intersection are given. Assuming that every road segment is either fully uphill or fully downhill, give an efficient algorithm to find the shortest path (route) that meets Hardy's specifications. If no such path meets Hardy's specifications, your algorithm should determine this. Justify your answer.

2. You are given a graph representing the several career paths available in industry. Each node represents a position and there is an edge from node *v* to node *u* if and only if *v* is a prerequisite for *u*. Top positions are the ones which are not prerequisites for any positions. The cost of an edge *(v, u)* is the effort required to go from one position *v* to position *u*. Ivan wants to start a career and achieve a top position with minimum effort. Using the given graph can you provide an algorithm with the same run time complexity as Dijkstra's? You may assume the graph is a DAG.

3. You have a stack data type, and you need to implement a FIFO queue. The stack has the usual POP and PUSH operations, and the cost of each operation is 1. The FIFO has two operations: ENQUEUE and DEQUEUE. We can implement a FIFO queue using two stacks. What is the amortized cost of ENQUEUE and DEQUEUE operations.



4. Given a sequence of numbers: 3, 5, 2, 8, 1, 5, 2, 7

a. Draw a binomial heap by inserting the above numbers reading them from left to right
b. Show a heap that would be the result after the call to deleteMin() on this heap

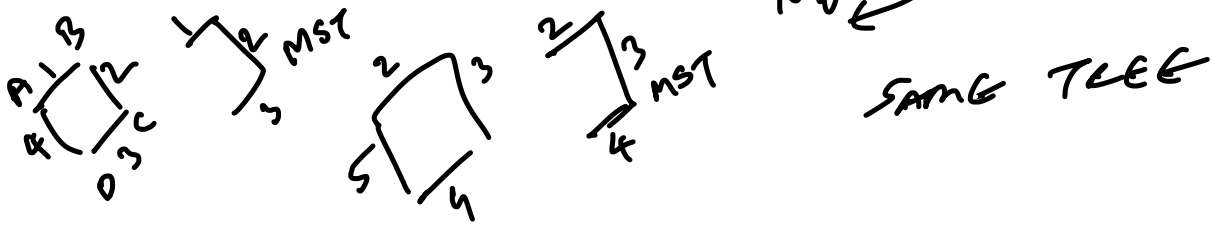
5. (a): Suppose we are given an instance of the Minimum Spanning Tree problem on a graph G. Assume that all edges costs are distinct. Let T be a minimum spanning tree for this instance. Now suppose that we replace each edge cost c_e by its square, c_e^2 thereby creating a new instance of the problem with the same graph but different costs. Prove or disprove: T is still a MST for this new instance.

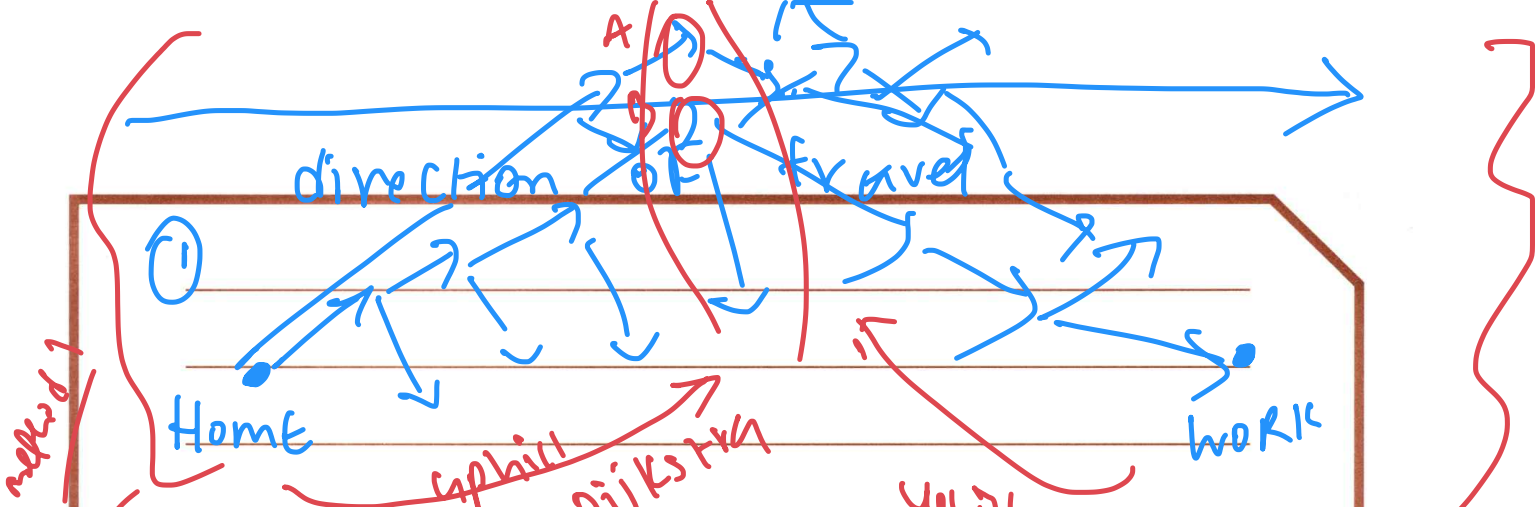
False

if c_e is -ve c_e^2 +ve

(b): Consider an undirected graph G = (V, E) with distinct nonnegative edge weights $w_e \geq 0$. Suppose that you have computed a minimum spanning tree of G. Now suppose each edge weight is increased by 1: the new weights are $c'_e = c_e + 1$. Does the minimum spanning tree change? Give an example where it changes or prove it cannot change.

NO





Modifying Dijkstra

while finding uphill node
during relaxation only relax nodes that are uphill ignore all nodes downhill

uphill

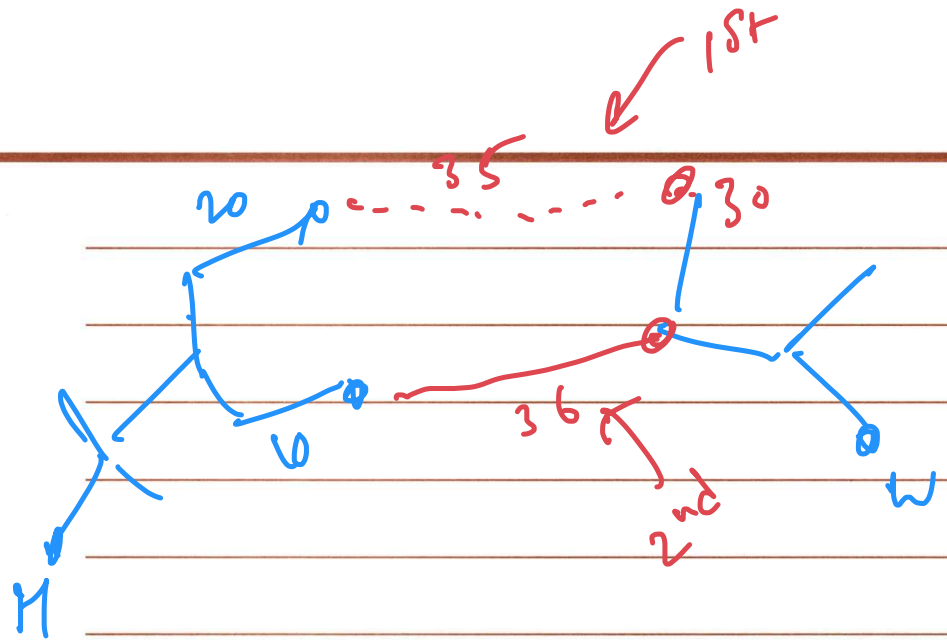
downhill

return min(A, B)
as we know dist of (Home - A, Home - B) min

method2
Convert undirected graph → directed graph

Then on moving ↗ will guarantee we are going upward

Initially this graph is undirected with elevations of each points

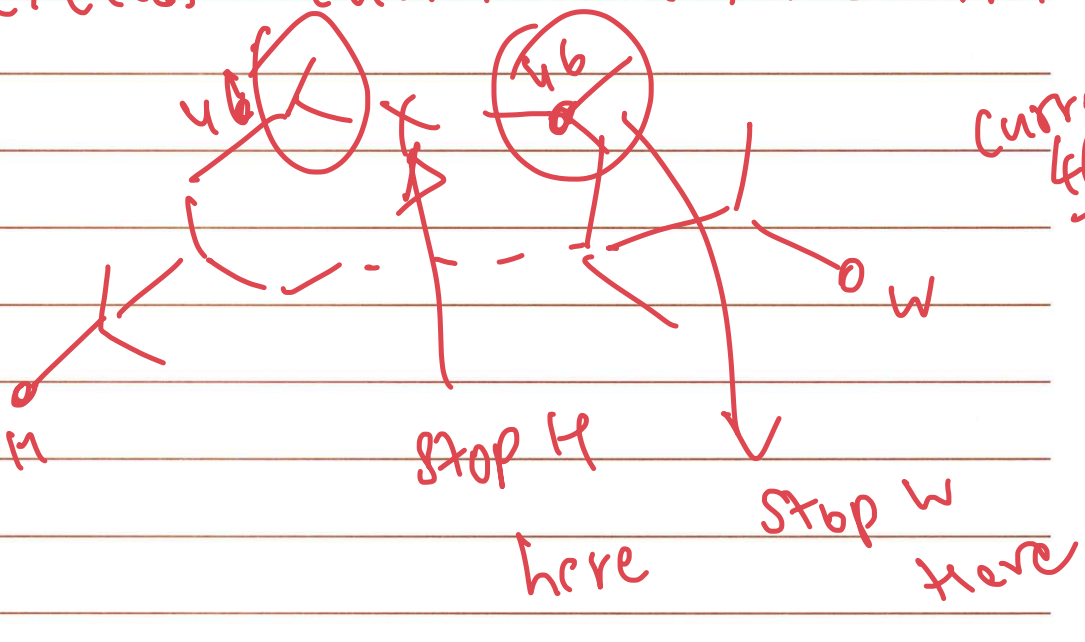


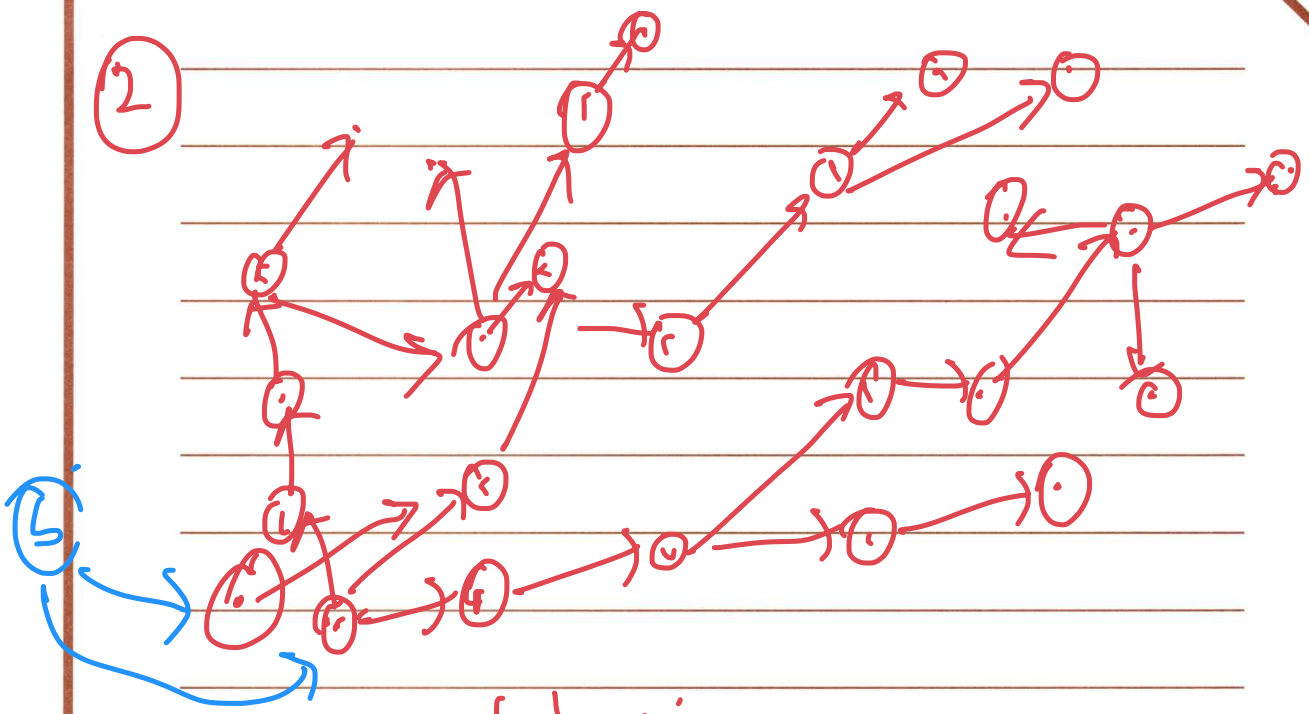
1st point might not always be best

stop parallel execution when

expanding H or W

exceeds current Total Shortest path

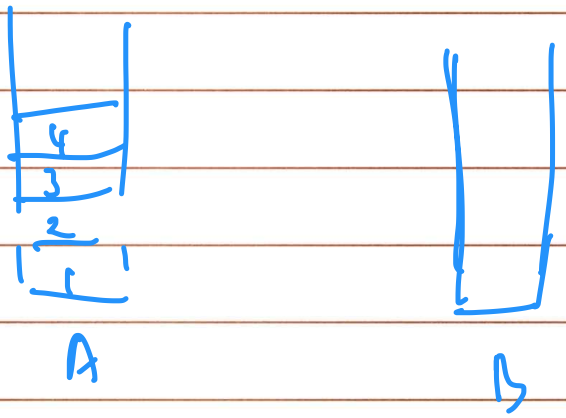




Soln1 run 'n' dijkstra from n starting nodes and get min of all ans

Soln2 add new node (S) and connect it to all starting node and run dijkstra's only once then.

(3) Implement Queue (FIFO) using two stacks

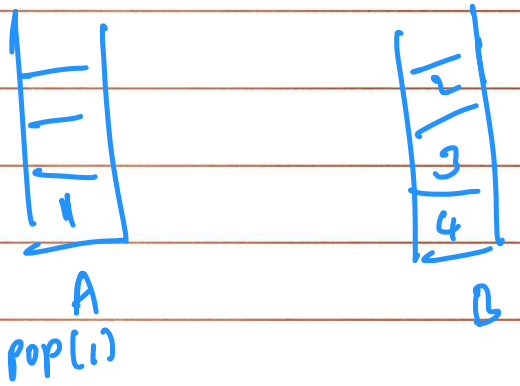


To dequeue we transfer

A: pop(4) push(4) in B

pop(3) push(3) in B

pop(2) push(2) in B



pop(1)

now dequeue will be pop() from B

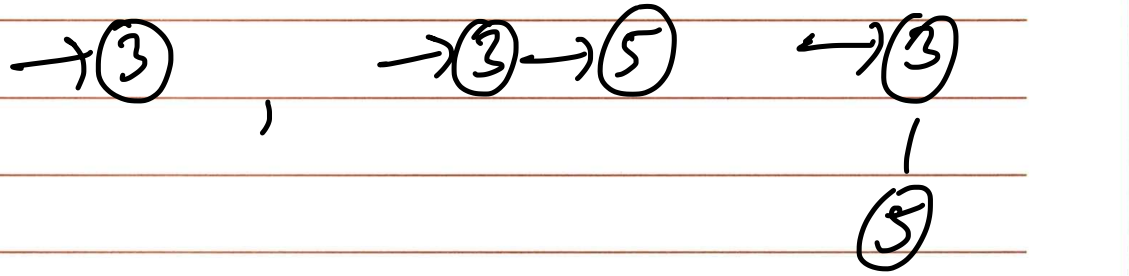
n enqueue → n pushes → n

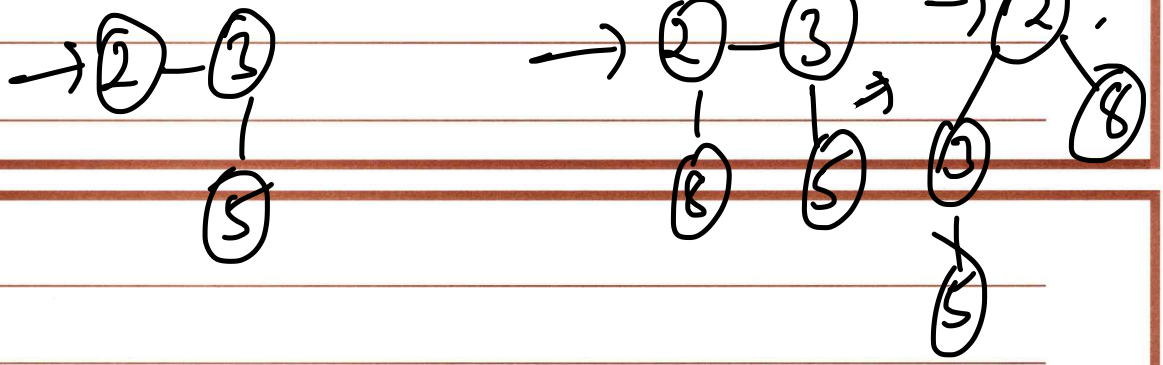
1 dequeue → (n pop, n push, 1 pop)

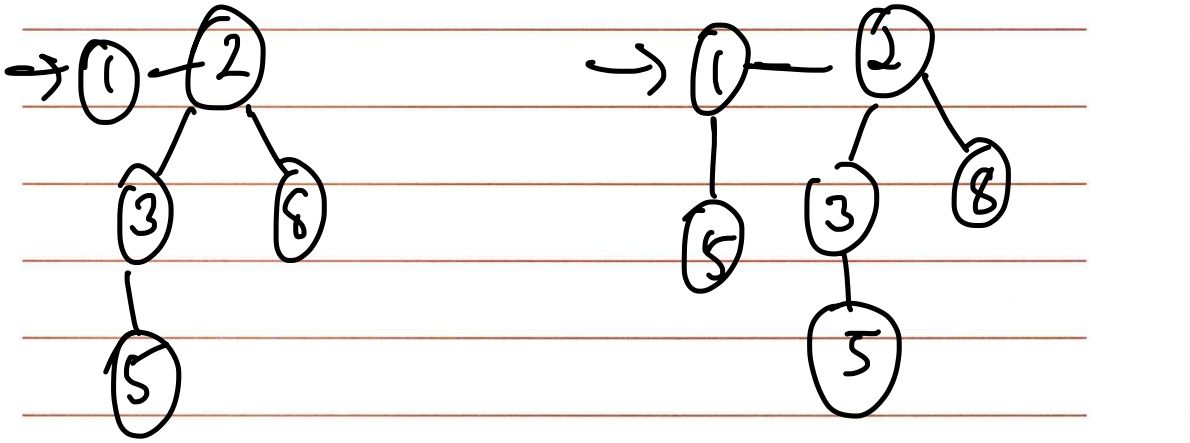
Total cost of n operation = 3n+1

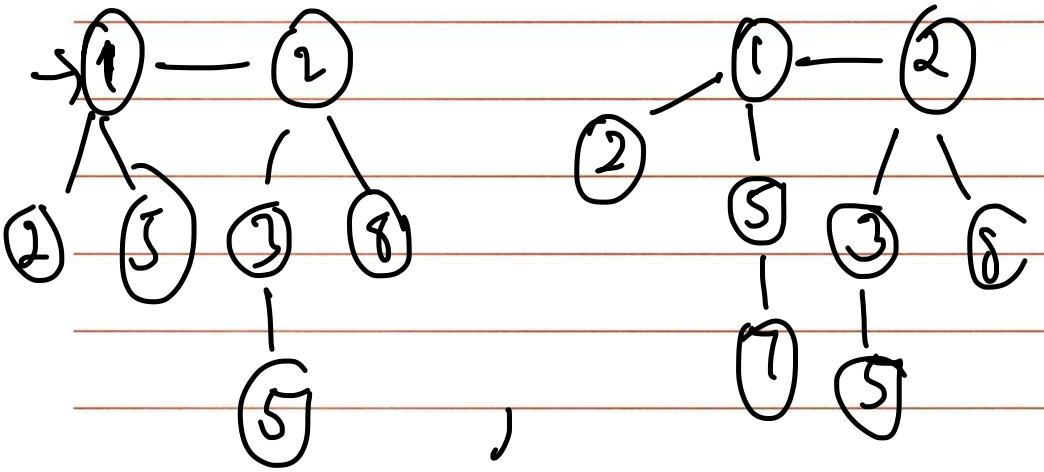
Amortized cost = $\frac{3n+1}{n}$ = O(1)

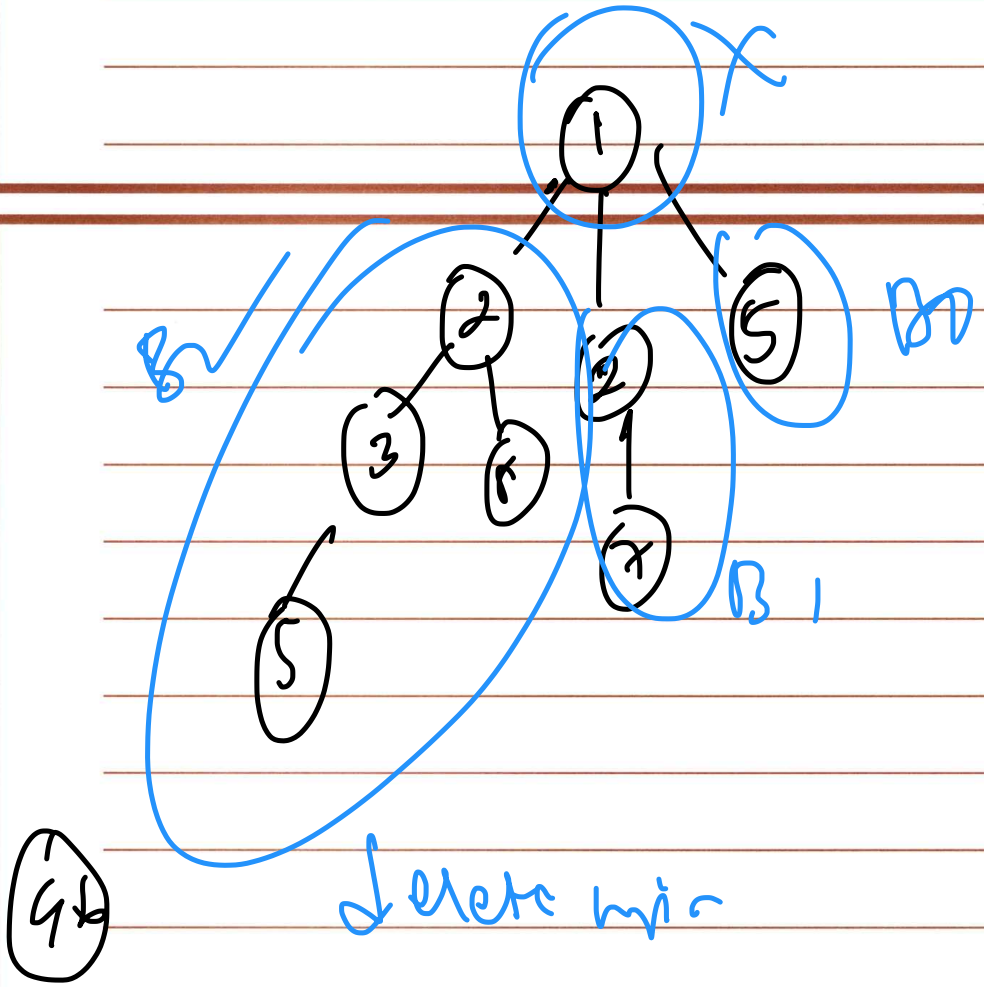
min heap ④ 3, 5, 2, 8, 1, 5, 2, 7










Delete min