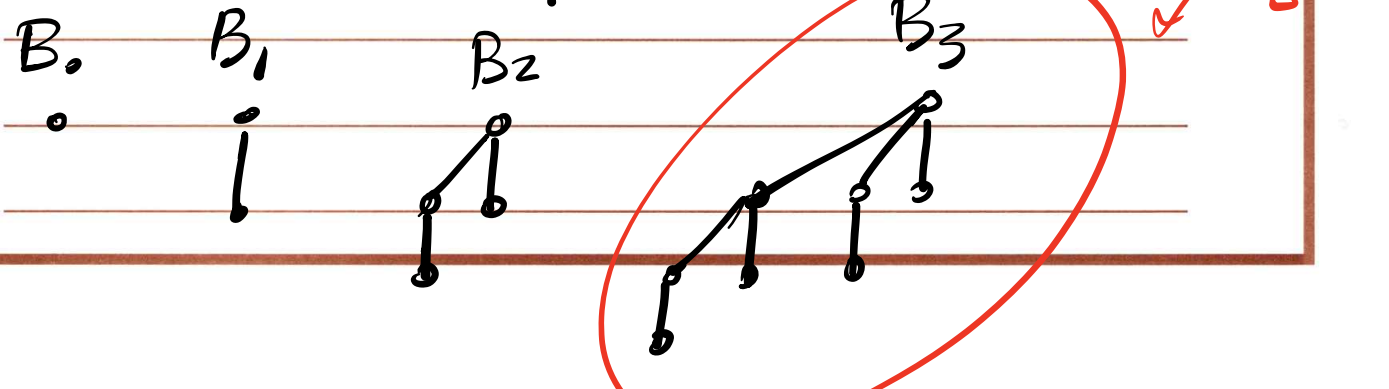
Other Heaps

Def A binomial tree B_k is an ordered tree defined recursively:

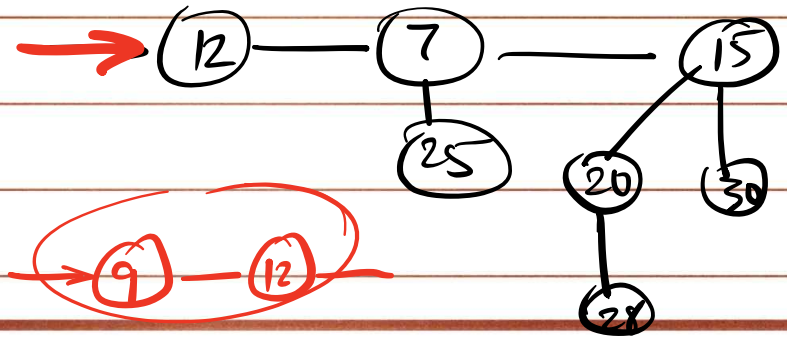
- Binomial tree B_0 consists of one node
- Binomial tree B_k consists of 2 binomial trees B_{k-1} that are linked together such that root of one is the leftmost child of the root of the other.

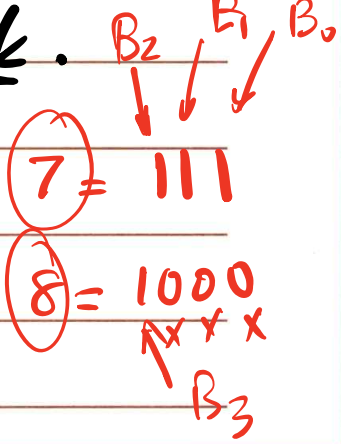


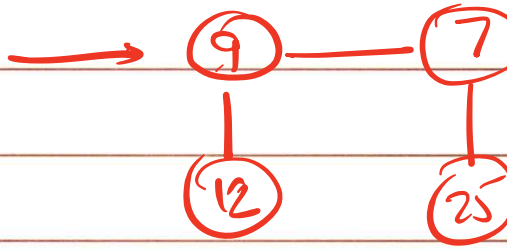
Def. A binomial heap H is a set of binomial trees that satisfies the following properties

1- Each binomial tree in H obeys the min-heap property

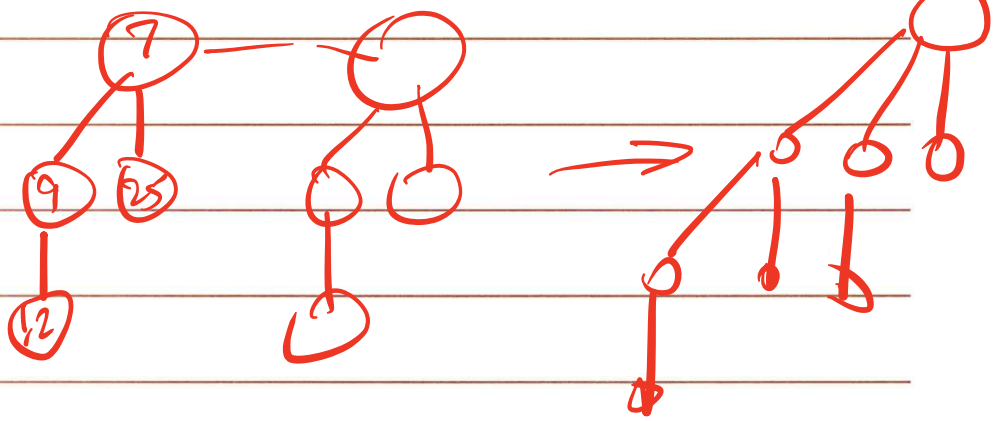
2- There is at most one binomial tree in H whose root has degree k.







Find & insert fak $O(\lg n)$



Amortized Cost Analysis

Ex. 1

```
for i = 1 to n
    push or Pop
endfor
```

push takes O(1)
Pop takes O(1)

Our worst case run time complexity = $O(n)$

Ex. 2

```
for i = 1 to n
    push or pop or multipop
endfor
```

push takes O(1) worst-case
pop takes O(1) worst-case
multipop takes O(n) worst-case

Our worst case run time complexity = $O(n^2)$

Aggregate Analysis

- We show that a sequence of n operations (for all n) takes worst-case time $T(n)$ total.

- So, in the worst case, the amortized cost (average cost) per operation will be $T(n)/n$

observation: Multipop takes $O(n)$ time if there are n elements pushed on the stack

A sequence of n pushes takes $O(n)$
multipop takes $O(n)$ worst-case

$T(n) = O(n)$
when amortized over n operations,
average cost of an operation $= O(n)/n = O(1)$

Ex. 2 for $i = 1$ to n $O(1)$ $O(1)$

$O(1)$ → push or pop or multipop

end for

push takes $O(1)$ amortized

pop " $O(1)$ "

multipop " $O(1)$ "

our worst case runtime complexity = $O(n)$

$\Theta(n)$

Accounting Method

- We assign different charges (amortized costs) to different operations

- If the charge for an operation exceeds its actual cost, the excess is stored as credit.

- The credit can later helps pay for operations whose actual cost is higher than their amortized cost.

- Total credit at any time = total amortized cost − total actual cost

Total
- ^Credit can never be negative.

Ex. 2

```
for i = 1 to n
    push or pop or multipop
endfor
```

try # 1

charge

assign a ~~cost~~ of 1 to each operation

op.	charge	Actual cost	total credit
push	1	1	0
push	1	1	0
multipop	1	2	-1

try # 2

assign charges as follows:

push 2 → O(1)

pop 0 → O(1)

multipop 0 → O(1)

op.	charge	Actual cost	total credit
push	2	1	1
push	2	1	2
multipop	0	2	0

Ex. 2

```
for i = to n
    push or pop or multipop
end for
```

	Amortized cost
push	$O(1)$
pop	$O(1)$
multipop	$O(1)$

worst-case runtime complexity = $O(n)$

- Fibonacci heaps are loosely based on binomial heaps.

- A Fibonacci heap is a collection of min-heap trees similar to Binomial heaps, however, trees in a Fibonacci heap are not constrained to be binomial trees. Also, unlike binomial heaps, trees in Fibonacci heaps are not ordered.

- Link to Fibonacci heap animations:

www.cs.usfca.edu/ngalles/JavascriptVisual/FibonacciHeap.html

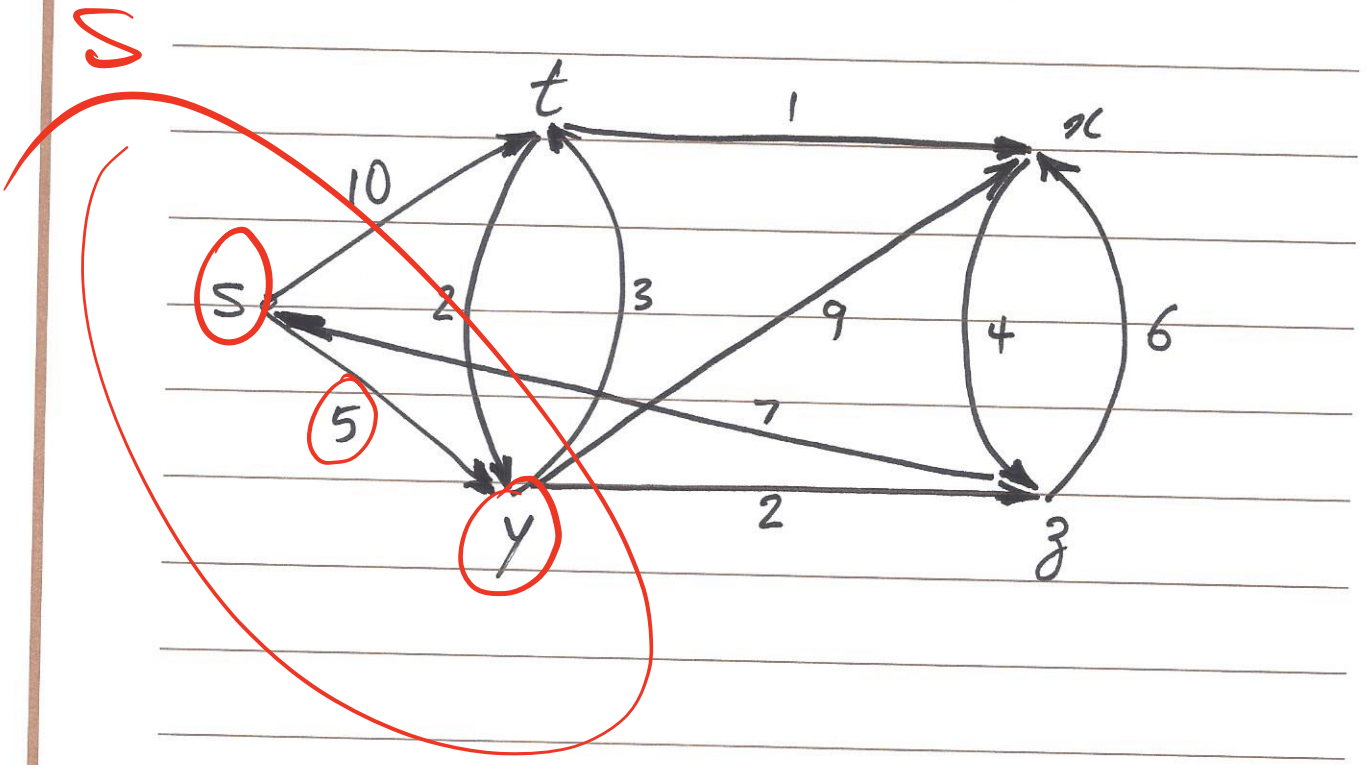
Amortized Costs

	Binary Heap	Binomial Heap	Fibonacci Heap
Find-Min	$O(1)$	$O(\lg n)$	$O(1)$
Insert	$O(\lg n)$	"	$O(1)$
Extract-Min	"	"	$O(\lg n)$
Delete	"	"	$O(\lg n)$
Decrease-key	"	"	$O(1)$
Merge	$O(n)$	"	$O(1)$
Construct	$O(n)$	$O(n)$	$O(n)$

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 $V - s$.

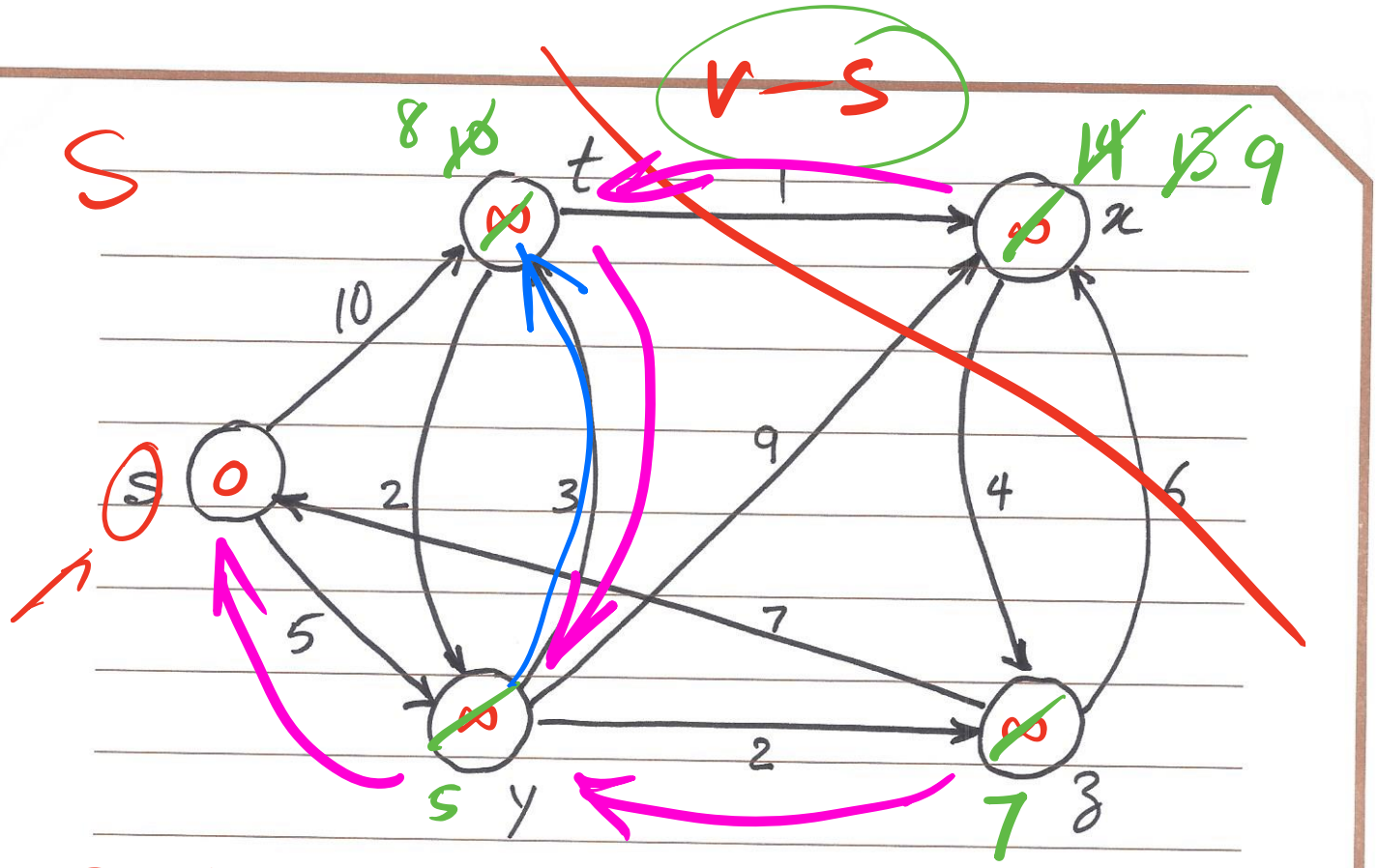


Solution

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.



$S_1 = \{s\}$, $S_2 = \{s, y\}$, $S_3 = \{s, y, z\}$, $S_4 = \{s, y, z, t\}$

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

Dijkstra's DIS.

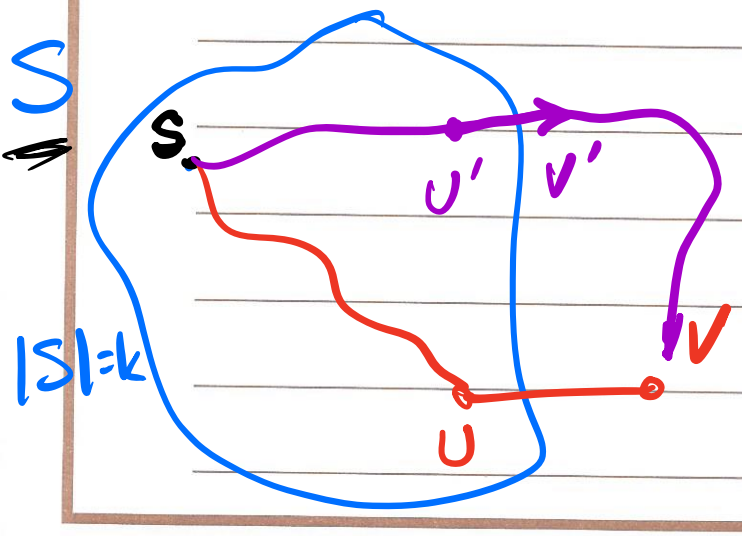
Proof of Correctness

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.



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

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 $= \infty$, except for s where d(s) = 0)

While $S \neq V$

  v = Extract_Min (Q)

  S = S ∪ {v}

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

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

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

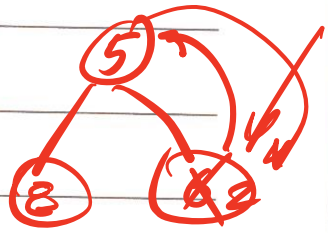
  end for

end while

$O(n^2)$ → $O(m)$

Complexity Analysis

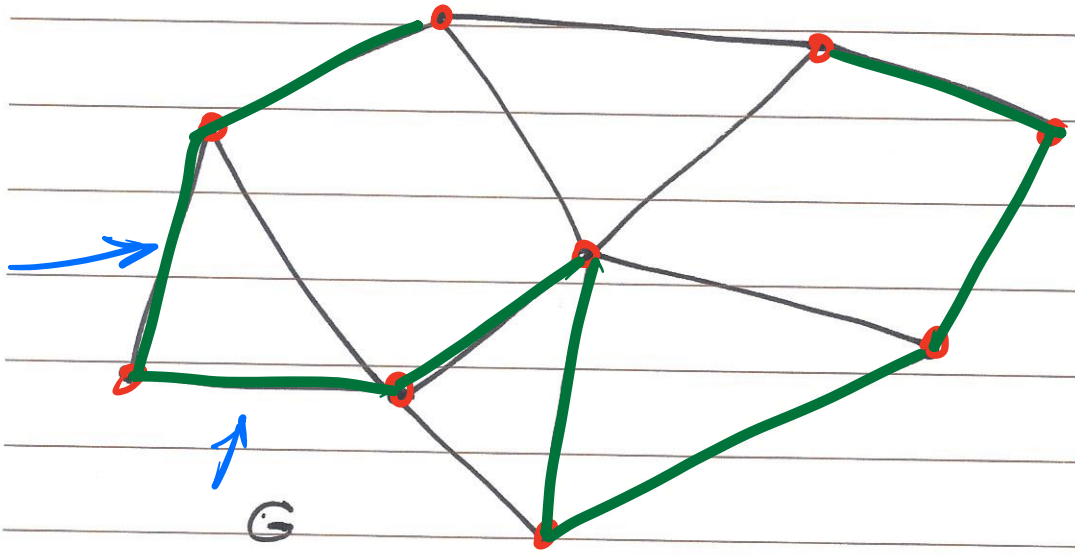
- Initialize Priority Queue $O(n)$
- Max. no. of Extract-Min op's : n
- Max. no. of Decrease-key op's : $O(m)$



	Binary Heap	Binomial Heap	Fibonacci Heap
n. Extract-Min's	$O(n \lg n)$	$O(n \lg n)$	$O(n \lg n)$
m. Decrease-key's	$O(m \lg n)$	$O(m \lg n)$	$O(m)$
Total	$O(m \lg n)$	$O(m \lg n)$	$(m + n \lg n)$
Sparse graph	$O(m \lg n)$	$O(n \lg n)$	$(n \lg n)$
Dense "	$O(n^2 \lg n)$	$O(n^2 \lg n)$	$O(n^2)$

Problem Statement

Find minimum cost network that connects all nodes in G.



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

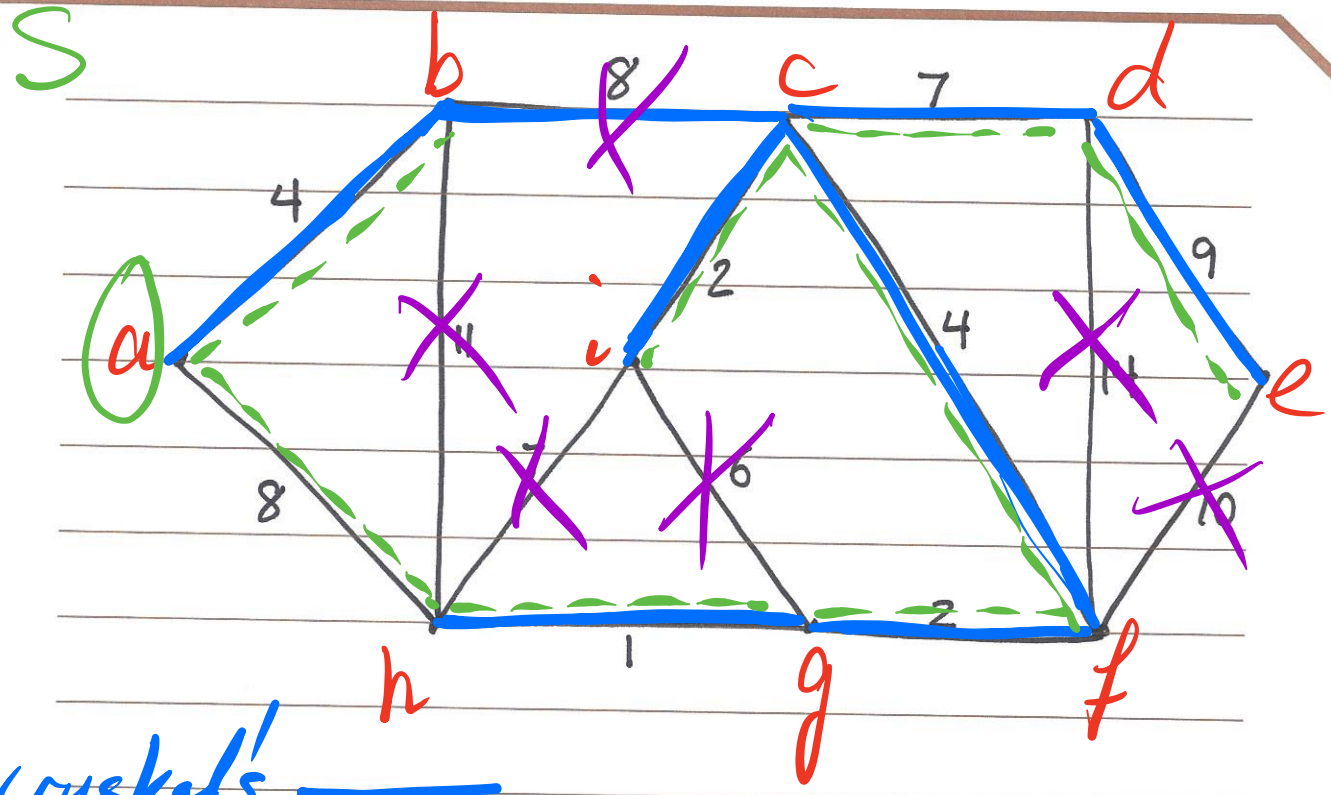
Sol. 2: Similar to Dijkstra's algorithm, start with a node set S (initially the root node) on which a minimum spanning tree has been constructed so far. At each step, grow S by one node, adding the node v that minimizes the attachment cost.

Prim's

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

Reverse-Delete

V-S



Kruskal's ——

Prim's - - - - -

Reverse-Delete X

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.