More Spanning Tree Algorithms

Instructor: Krishna Venkatasubramanian

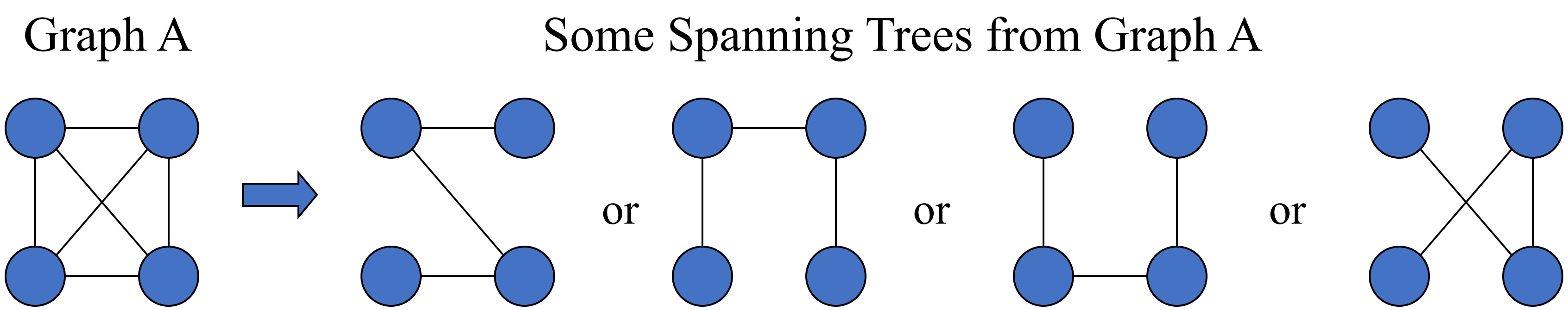
CSC 212

Announcement

- Quiz 6 (Last quiz) next Tuesday
  - Dec 10
- Assignment 3 due next Tuesday by midnight
  - Dec 10
- Academic Accommodations
  - Please email me if you need accommodations for the final exam – we can work something out
  - Please do by this tomorrow, Friday (Dec 6) 5pm
- Course Evaluation

Spanning Trees (RECAP)

- **A spanning tree of a graph is just a subgraph that contains all the vertices and is a tree.**
- **A graph may have many spanning trees.**
- **Spanning trees are defined for *connected undirected* graphs**
- **Since there are trees ➔ They have no cycles**



Algorithms for Obtaining the Minimum Spanning Tree

- **Kruskal's Algorithm**
- **Prim's Algorithm**

Both of these are Greedy Algorithms

Kruskal's Algorithm: Overview (RECAP)

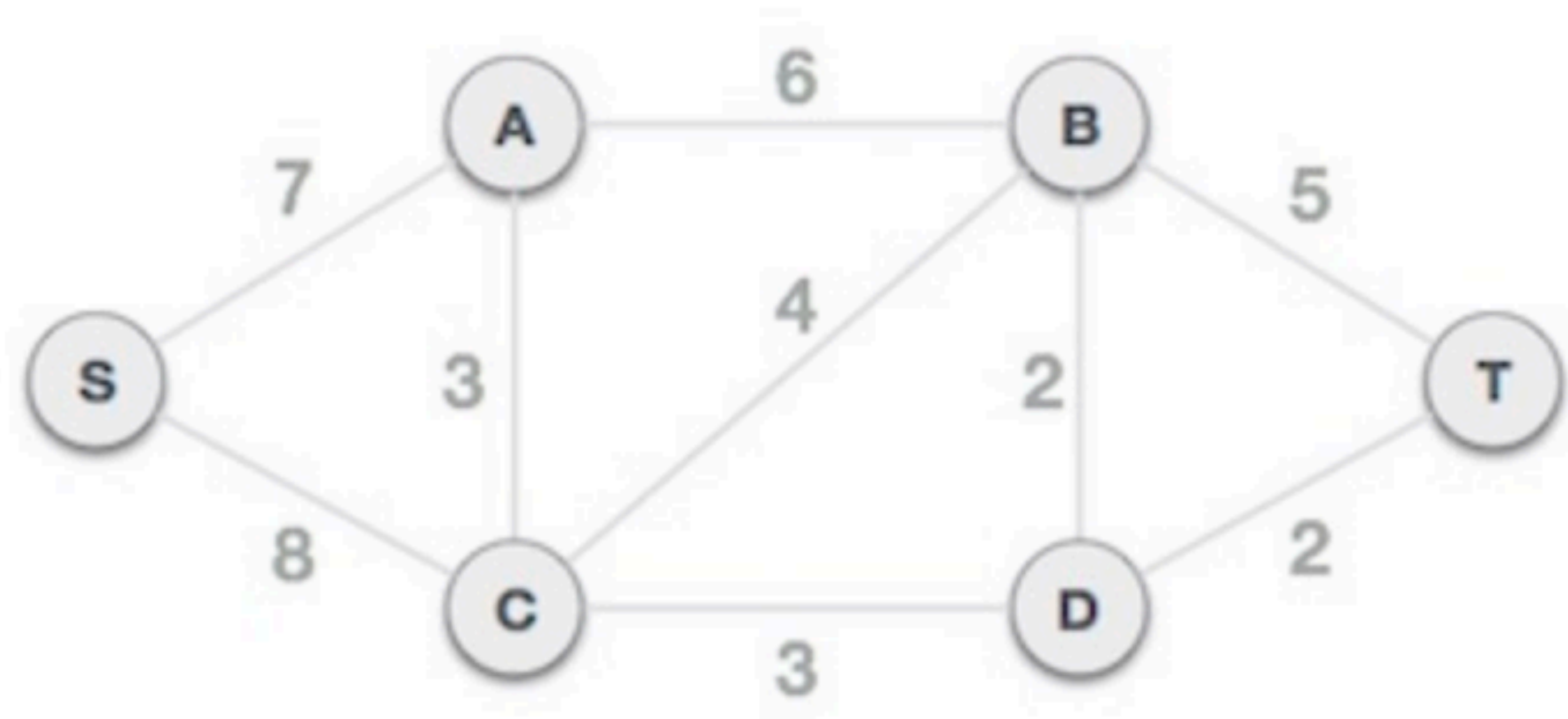
1. The forest is constructed - with each node in a separate tree.

2. The edges are placed in a min-priority queue.

3. Until we've added n-1 edges,
    1. Extract the cheapest edge from the queue,
    2. If it forms a cycle, reject it,
    3. Else add it to the forest. Adding it to the forest will join two trees together.

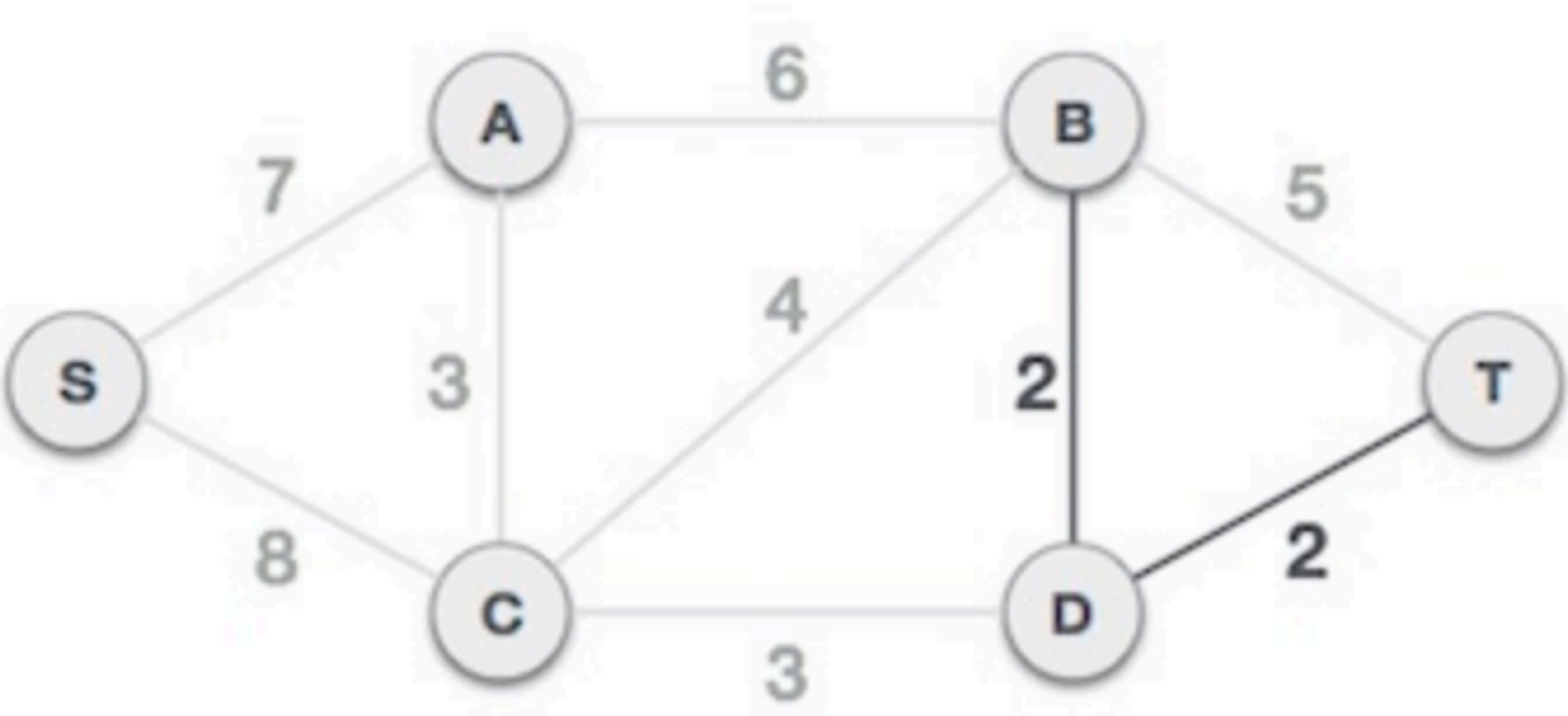
- If we start with n nodes (n separate trees)
- Each step we connect two trees
- Then we need (n-1) edges to get a single tree

Kruksal’s MST Example



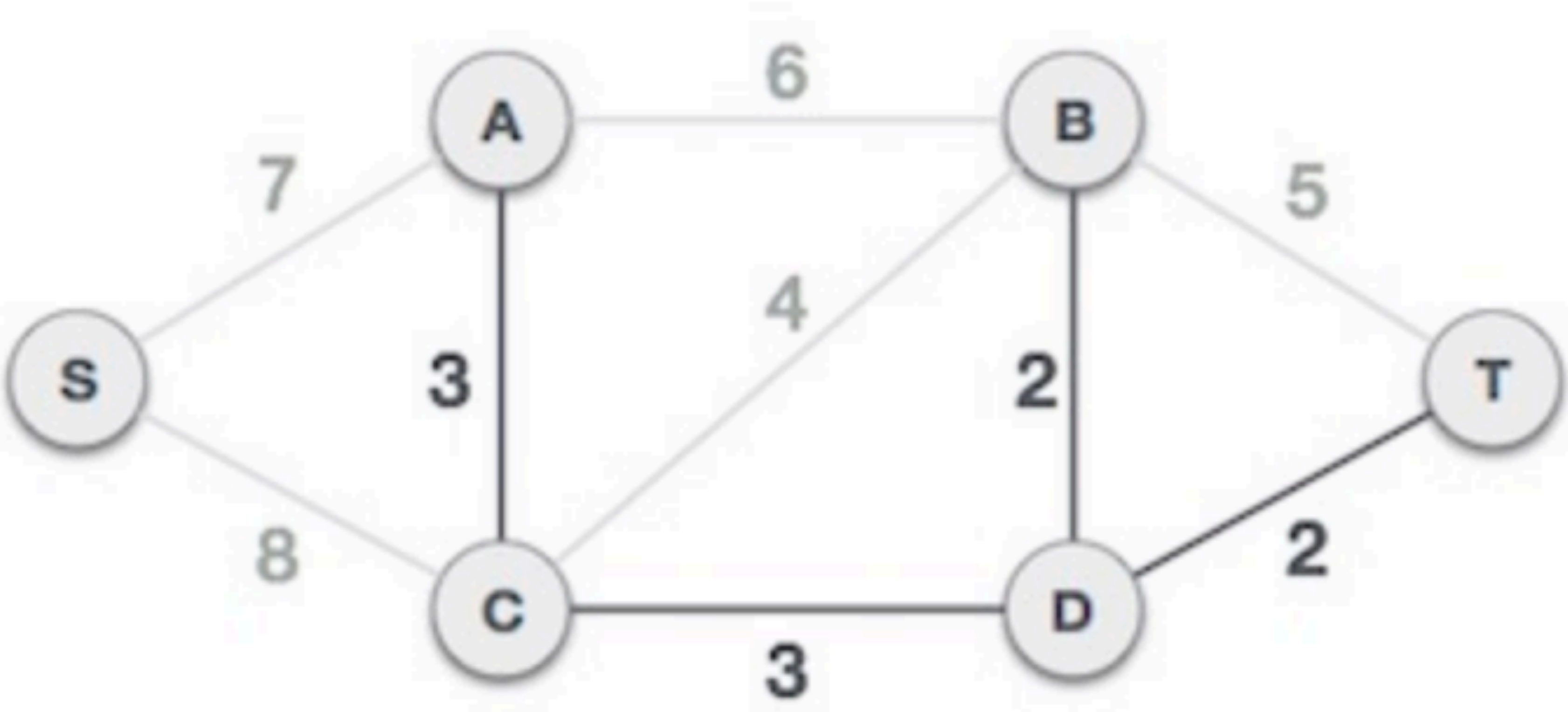
Find the minimum spanning tree for this graph using Kruksal’s method

Kruksal's MST Example



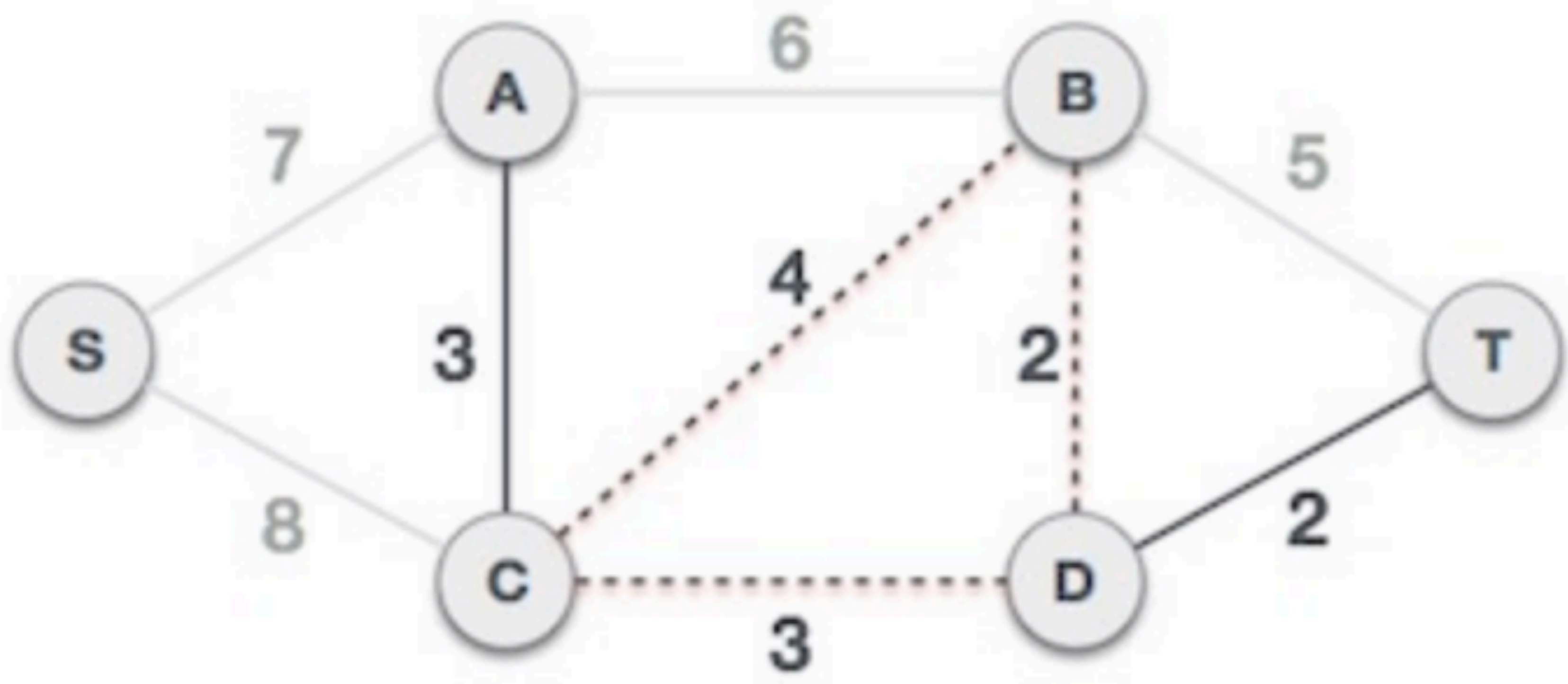
- The least cost is 2 and edges involved are B,D and D,T. We add them.
- Adding both these edges does not create loops

Kruksal’s MST Example



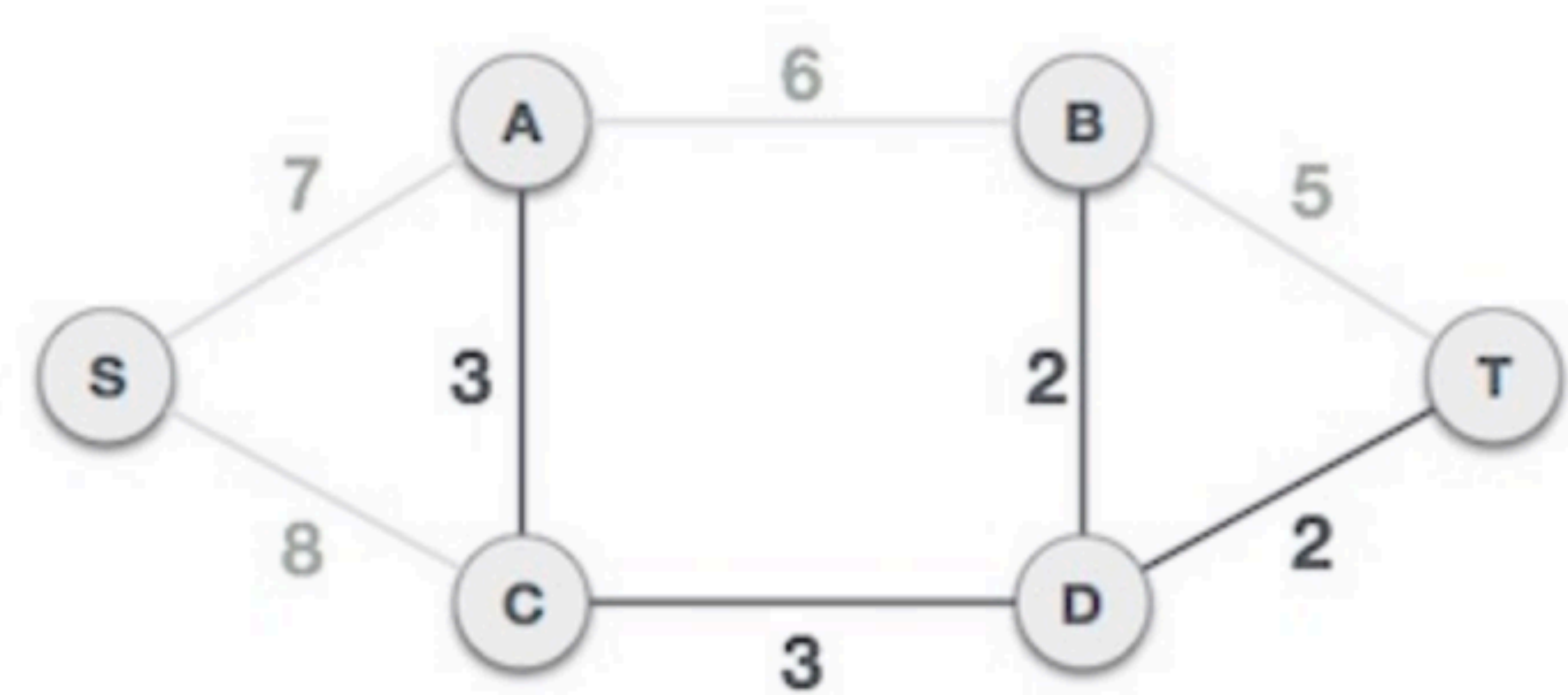
- Next cost is 3, and associated edges are A,C and C,D.
- Adding both edges to MST don’t lead to loops

Kruksal’s MST Example



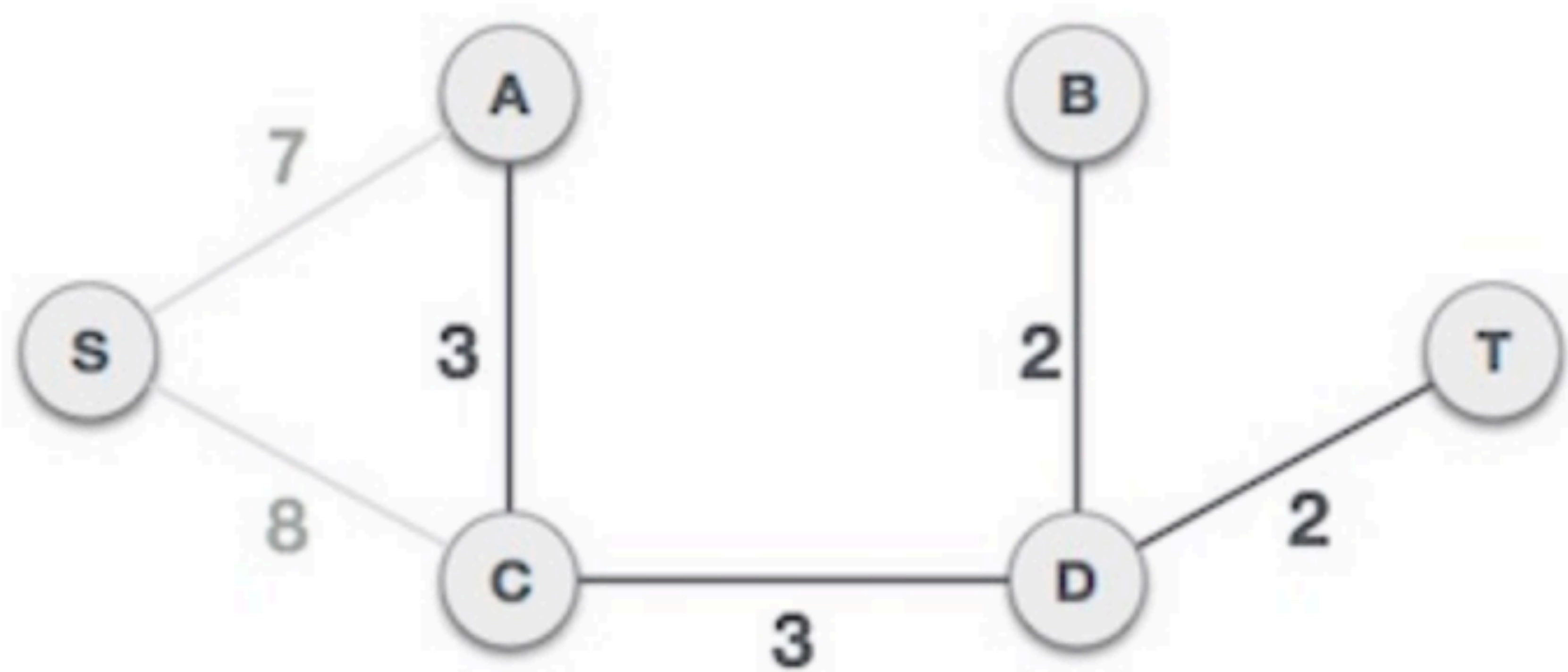
- Next cost in the table is 4, and we observe that adding it will create a loop in the MST
- Ignore edge C,B

Kruksal’s MST Example



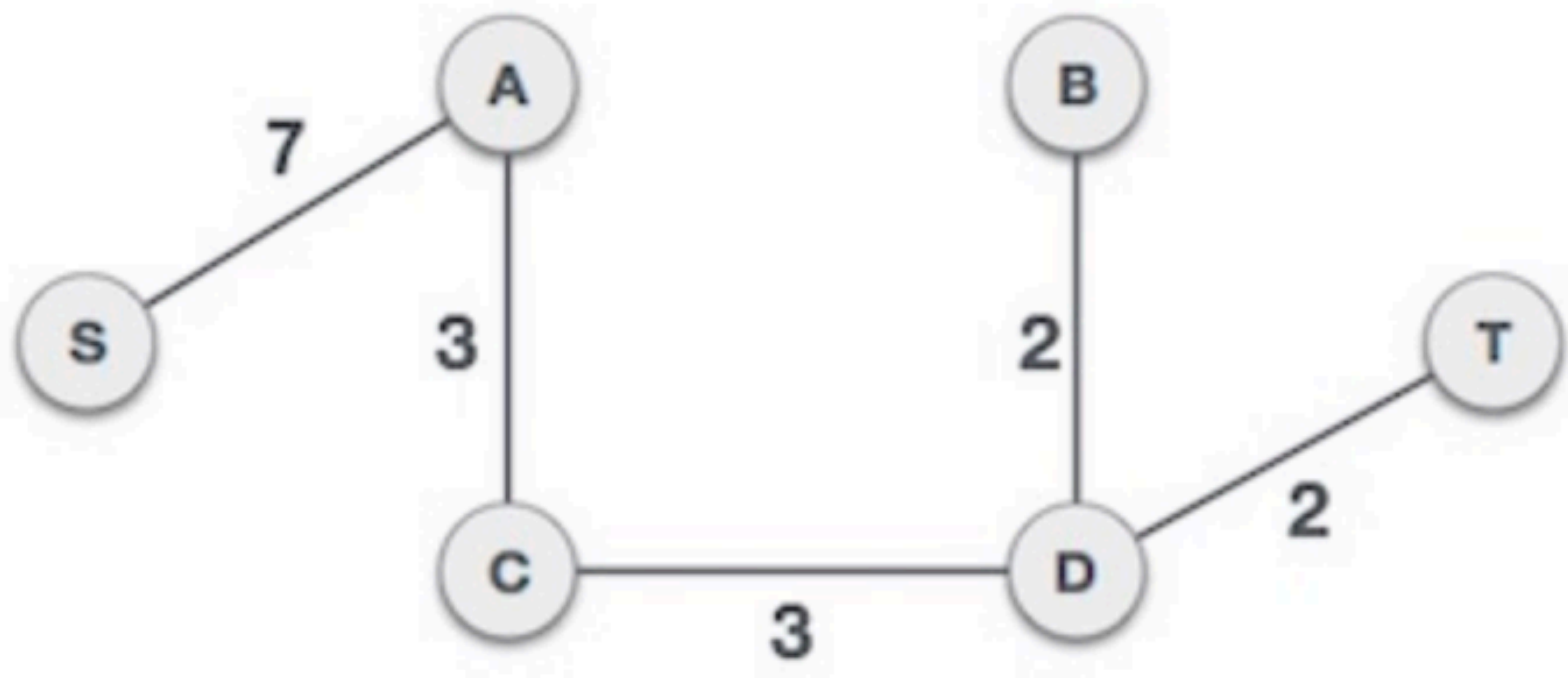
- We observe that edges with cost 5 and 6 also create circuits. We ignore them and move on.

Kruksal’s MST Example



- Now we are left with only one node to be added.
- Between the two least cost edges available 7 and 8, we shall add the edge with cost 7.

Kruksal's MST Example



COST = 17

Priority Queues (Recap)

Priority Queues

- A **heap-based** way for prioritizing things
  - It's called Queues, but it's implemented using a HEAP

- A queue where we add objects, each with a value ("priority").

- Priority queues are very common for *job scheduling*

- Two Types:
  - **Max-Priority Queue** ← we use MAX-HEAPS
  - **Min-Priority Queue** ← we use MIN-HEAPS

Operations on Priority Queues (Assume MIN-HEAP Priority Queue)

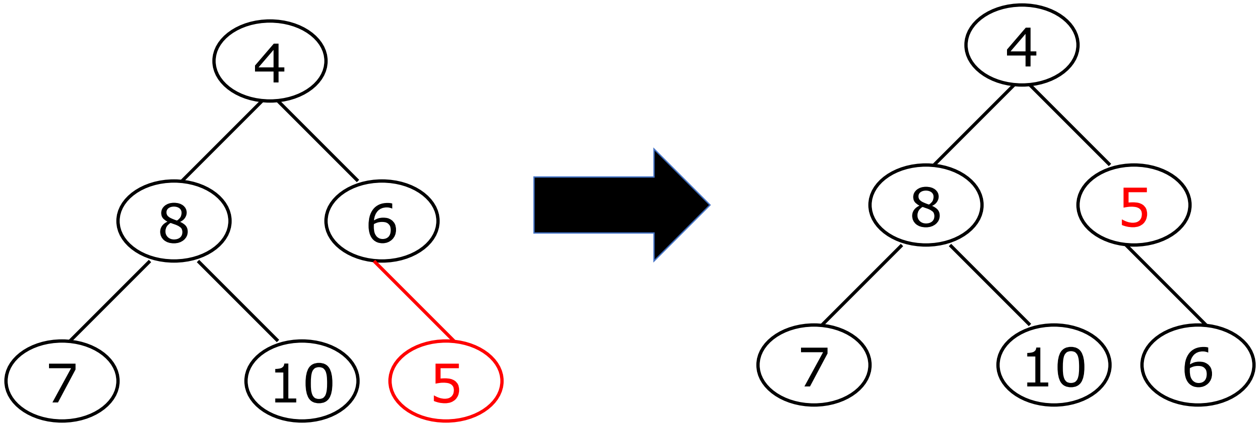
- 1 → **Add** a **new** object with priority K
- 2→ **Return/Extract** the object with the **lowest** priority
- 3 → **Remove** the object with the **lowest** priority
- 4 → **Decrease** the **priority** of object O

Heap data structure can implement all these operations efficiently

1- Add New Object With Priority 5

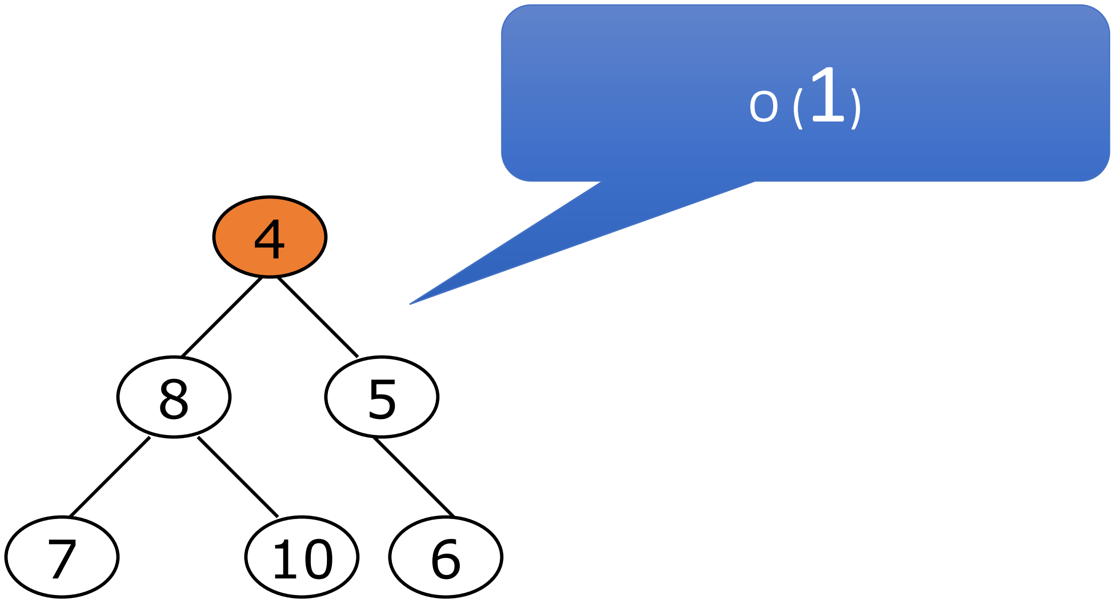
- Add the object to the heap
- Check parent and move node **upward iteratively**

O (Log n)



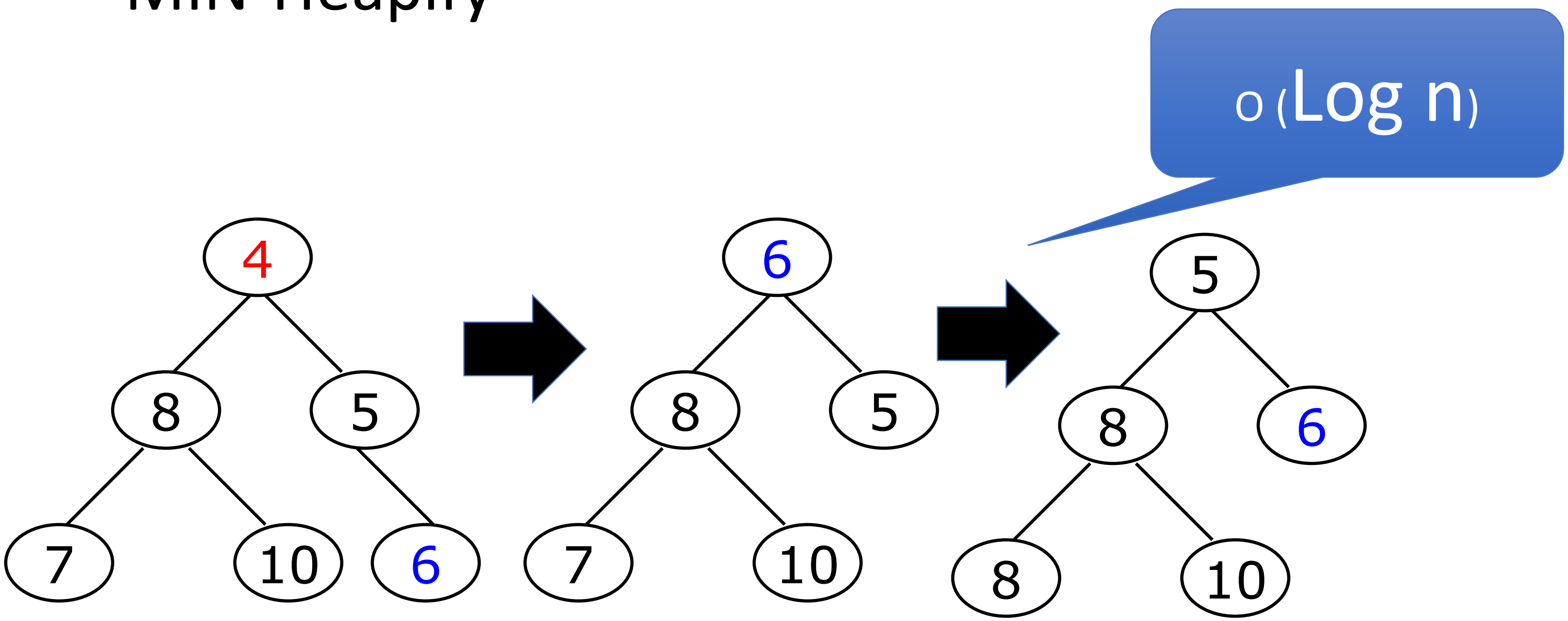
2- Return/Extract the Lowest-Priority Object

- Return the root of the tree
- Same as: Return the first element in the Heap array
- In our example, return 4

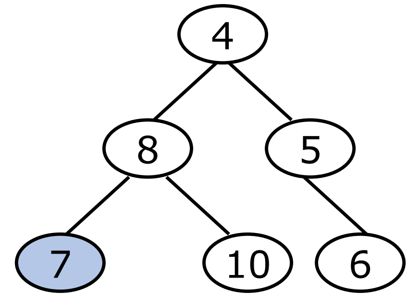


3- Remove the Lowest-Priority Object

- Remove the root of the tree
- Replace it with the lowest right-most object
- MIN-Heapify

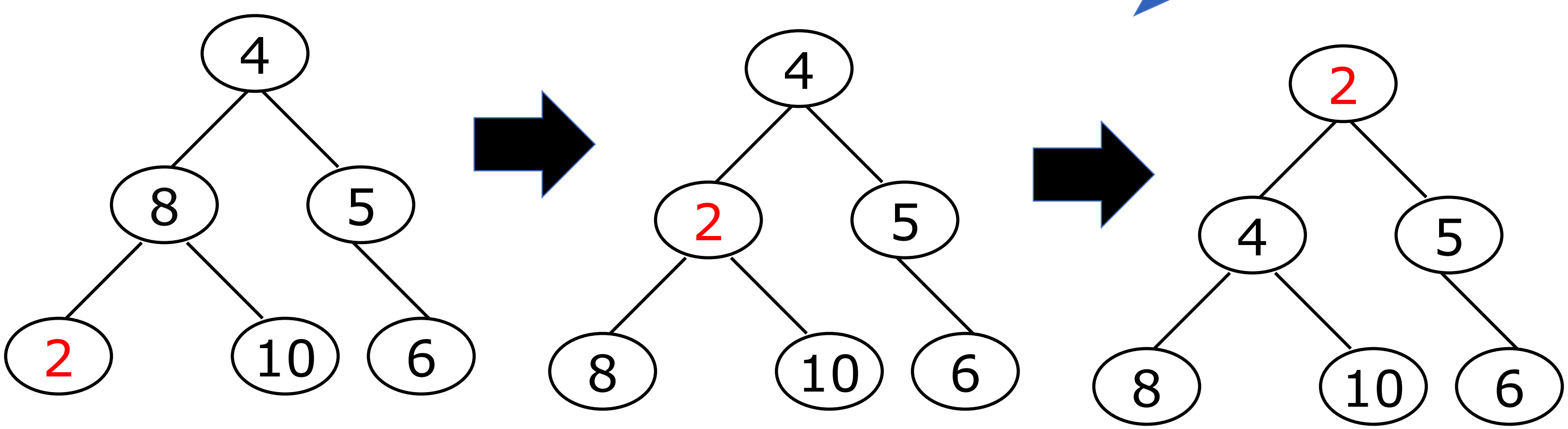


4- Decrease the Priority of Object O



- Change 7 to 2
- Check parent and move node **upward iteratively**

O (Log n)



Graph Asymptotic Growth Rates

More Asymptotic Growth Rates

- A graph G has to elements
  - V – vertices
  - E – edges

- Complexity of graph algorithms is usually expressed in terms of a function of V and E

- So it’s good to have an intuition for the **relative run-time complexity** seen in graph algorithms

Order the following (in the worst case)

- **$O(E)$, $O(EV)$, $O(E^2)$, $O(V^2)$, $O(V)$**
  - **$O(V)$ < [$O(E)$ ~ $O(V^2)$]** < $O(EV)$ < $O(E^2)$
  - Observation:
    - $E = V^2 - V/2$. Therefore $V < E$ and $O(V^2) < O(E^2)$
    - However, $O(E) \sim O(V^2)$
    - $O(EV) \sim O(V^3)$, $O(E^2) \sim O(V^4)$, therefore $O(EV) < O(E^2)$
      - ~ is the symbol for equivalence here
    - $O(E) < O(EV)$, of course.

- **$O(EV)$, $O(V \log V)$, $O(E \log E)$, $O(E \log V)$, $O(E+V)$**
  - **$O(V \log V) < O(E+V) <$ [$O(E \log E) \sim O(E \log V)$]** < $O(EV)$
  - Observation:
    - $O(E+V) \sim O(V^2)$. Therefore, $O(V \log V) < O(E+V)$
    - $O(E+V) \sim O(E)$. Therefore, $O(E+V) < O(E \log E)$
    - $O(E \log E) \sim O(E \log V)$. Since $E = O(V^2)$, then $O(\log E) = O(2 \log V) = O(\log V)$

Order the following (in the worst case)

- So what's the relationship between
- **{O(V) < [O(E) ~ O(V^2)]} and {O(V log V)< O(E+V) < [O(E log E) ~ O(E log V)]}**
  - O(V) < O(V log V) < **[O(E) ~ O(V^2) ~ O(E+V)]** < [O(E log E) ~ E (log V)]
  - Observation:
    - E ~ V^2. Therefore O(V log V) < O (E)
    - O(V) < O(V log V) just like O(n) < O(n log n)
    - O(E+V) ~ O(E) and O(E+V) ~ O(V^2)

Overall Ordering (in the worst case)

[O(log E) ~ O(log V)] < O(V) < O(V log V) < [O(E) ~ O(V^2) ~ O(E+V)] < [O(E log E) ~ O(E log V)] < O(EV) < O(E^2)

O(E log E) and O(E log V) are equivalent as O(log E) and O(log V) are equivalent
O(E), O(V^2), and O(E+V) are equivalent

Back to MST: Prim's Algorithm

Prim's Algorithm

1) Assign a key value to all vertices in the input graph. Initialize all **key values as INFINITE**.

2) Assign **key value as 0 for the first vertex** so that it is picked first and call it ROOT
   a) The **parent** of the root is NIL

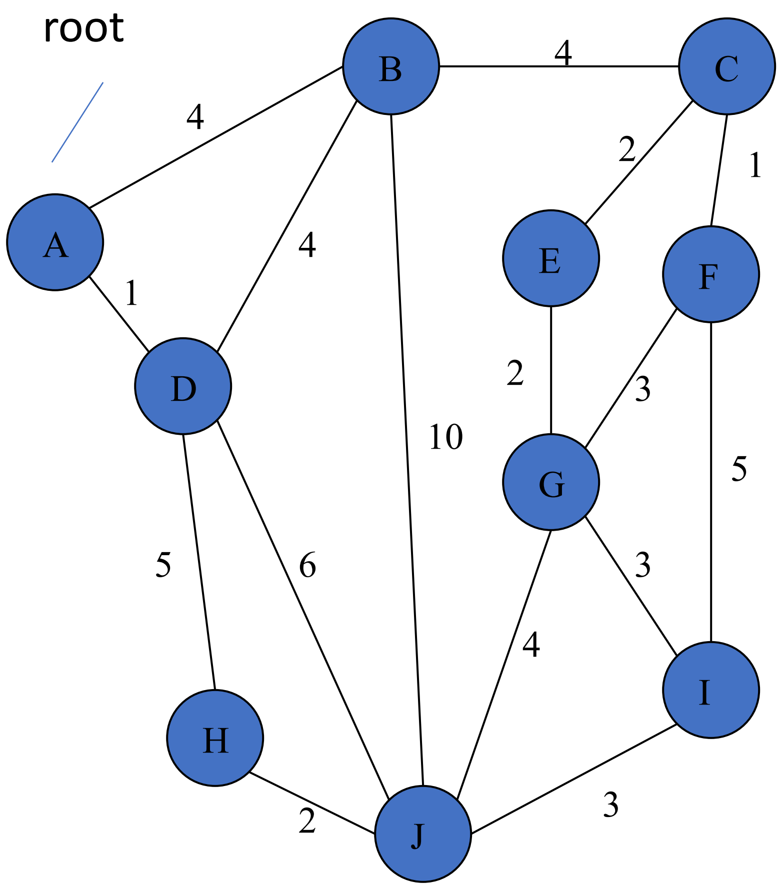
3) Add all nodes into a MIN-HEAP Priority Queue → Q

The greedy choice

4) While Q not empty
....**a)** Pick a vertex *u* that has **minimum key value (slide 18)**
....**b)** Include *u* to mstSet (set that lists MST).
....**c)** Update key value of **all adjacent vertices (v)** of *u (which is not it's parent)*.
- For every adjacent vertex *v*, if weight of edge *u-v* is **less** than the previous key value of *v*, update the key value as weight of edge *u-v*
- *Make u the **parent** of v*
- ***Update the priority Q (slide 20)****(i.e., min heap as we have new weights)*

... **d)** Remove u from Q

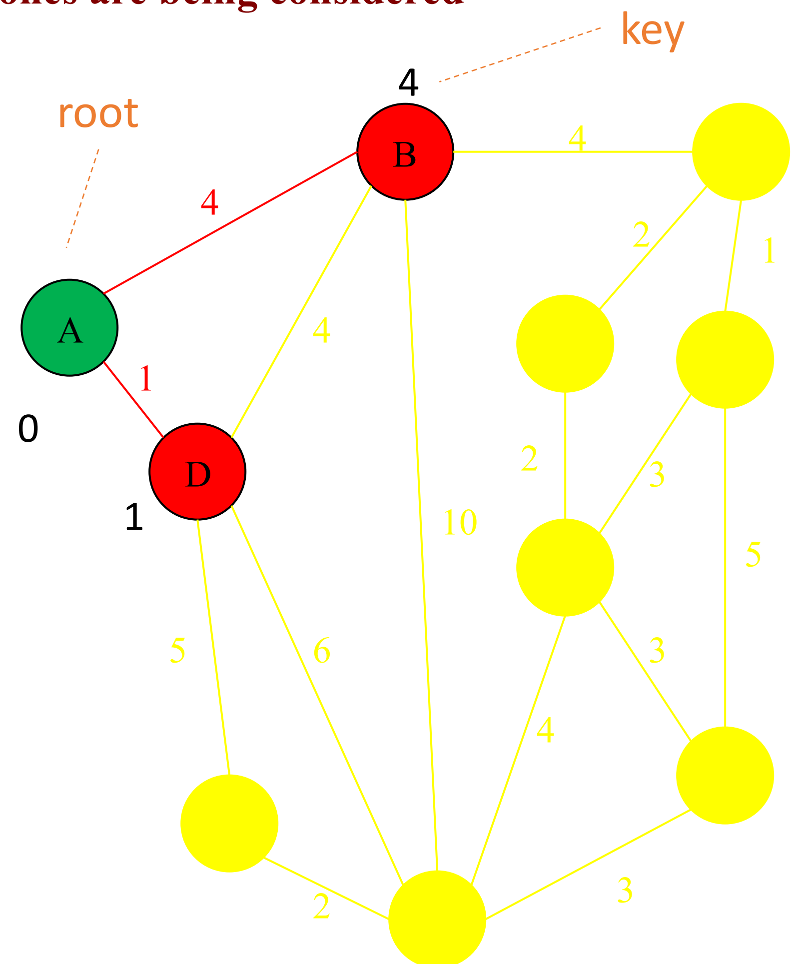
Prim’s Algorithm Example



mstSet = {NULL}

Prim's Algorithm Example

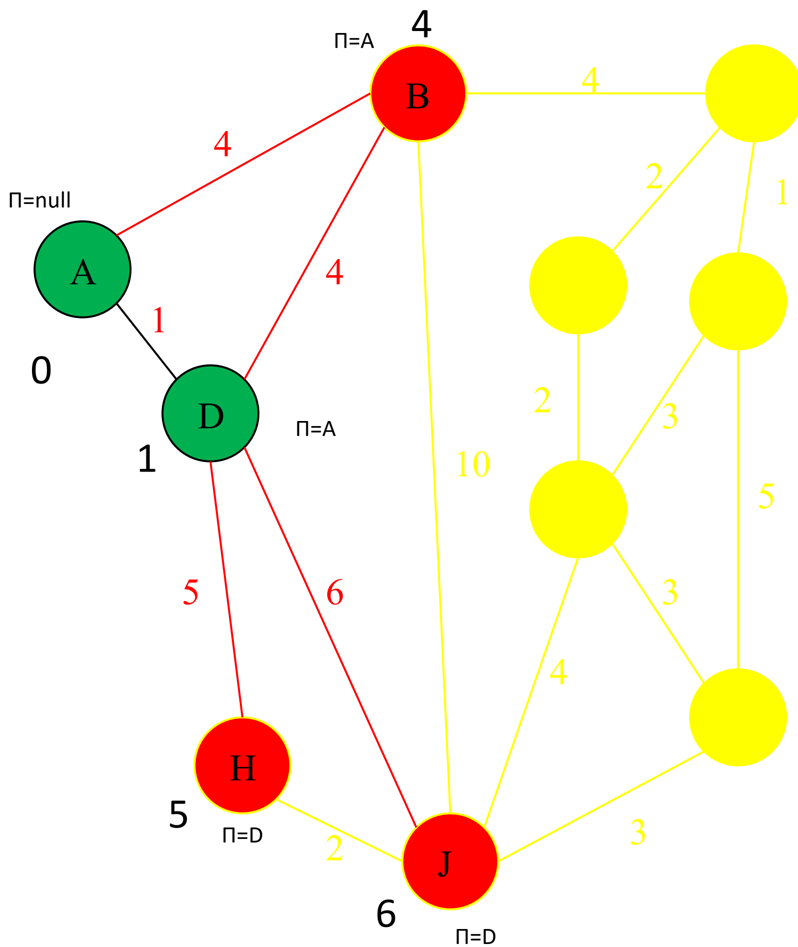
-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered



mstSet = {A}

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

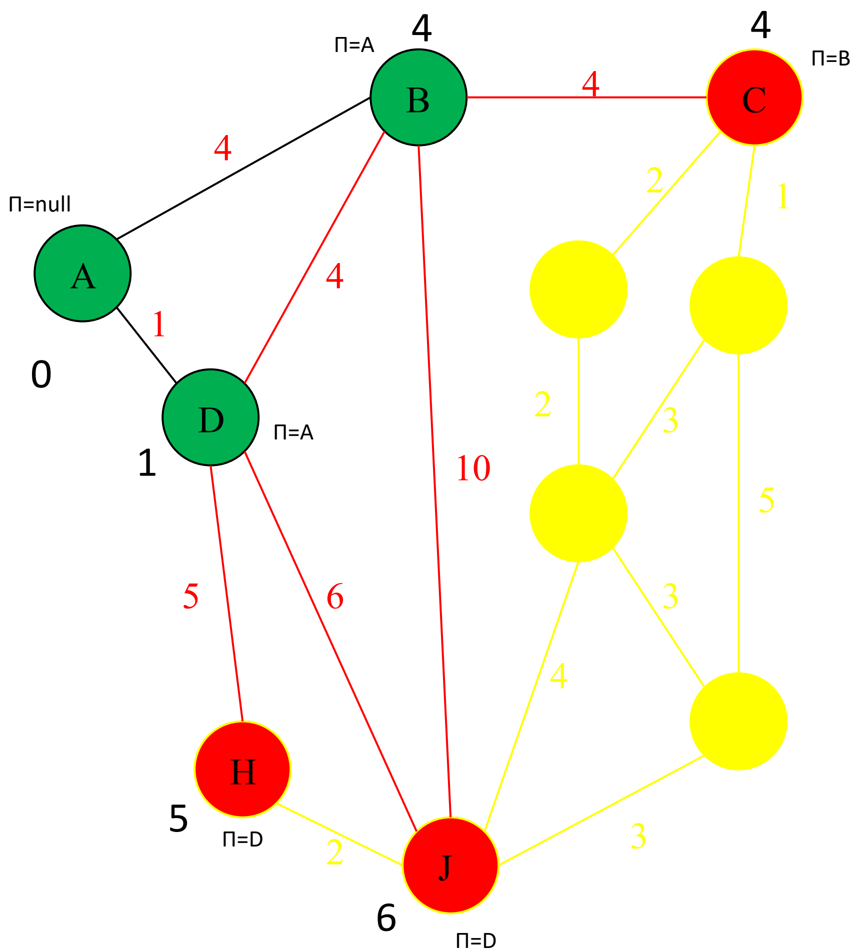


mstSet = {A,D}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

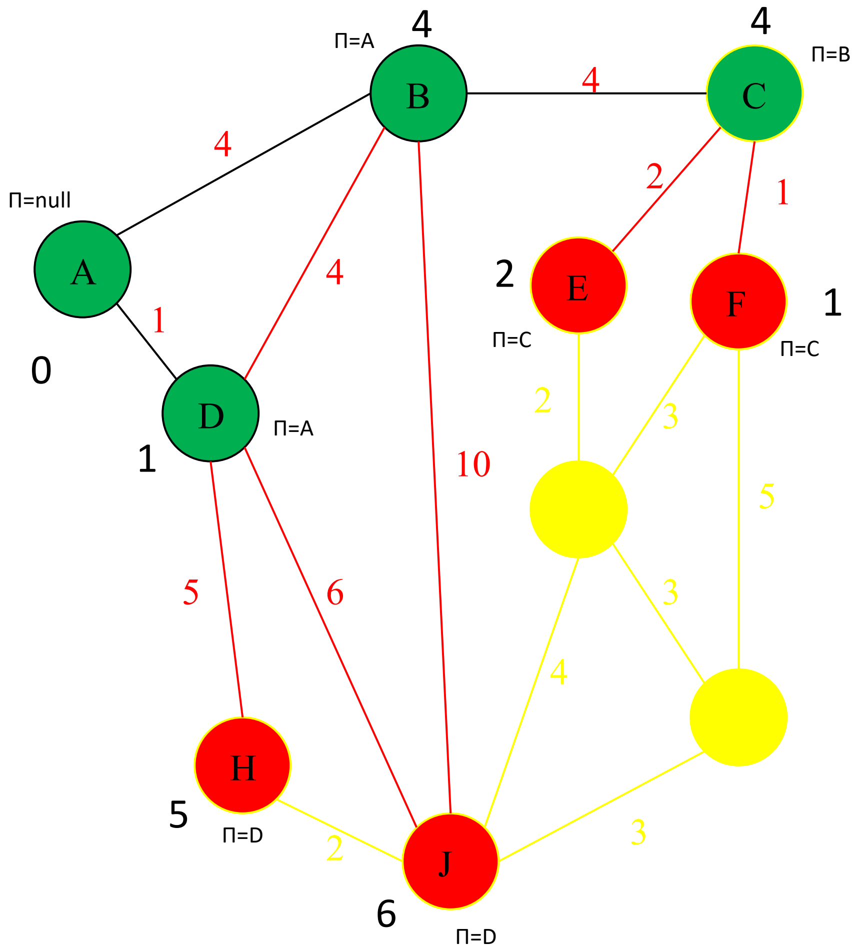


mstSet = {A,D,B}

The symbol Π is the parent of the vertex

Prim’s Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

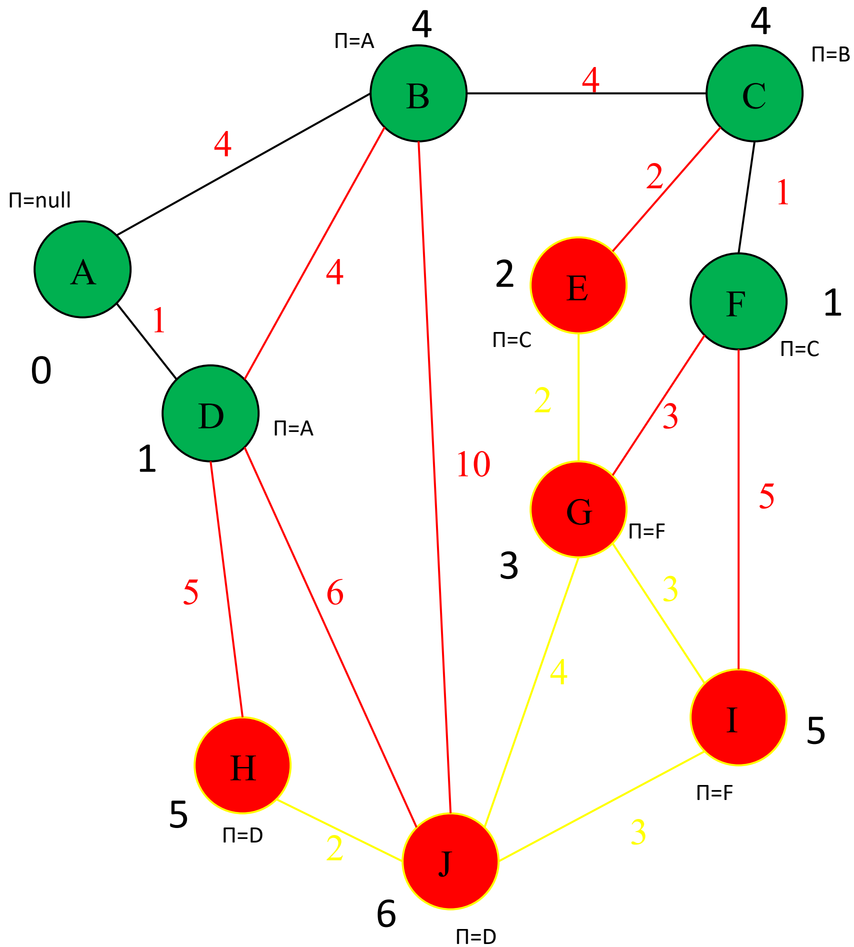


mstSet = {A,D,B,C}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

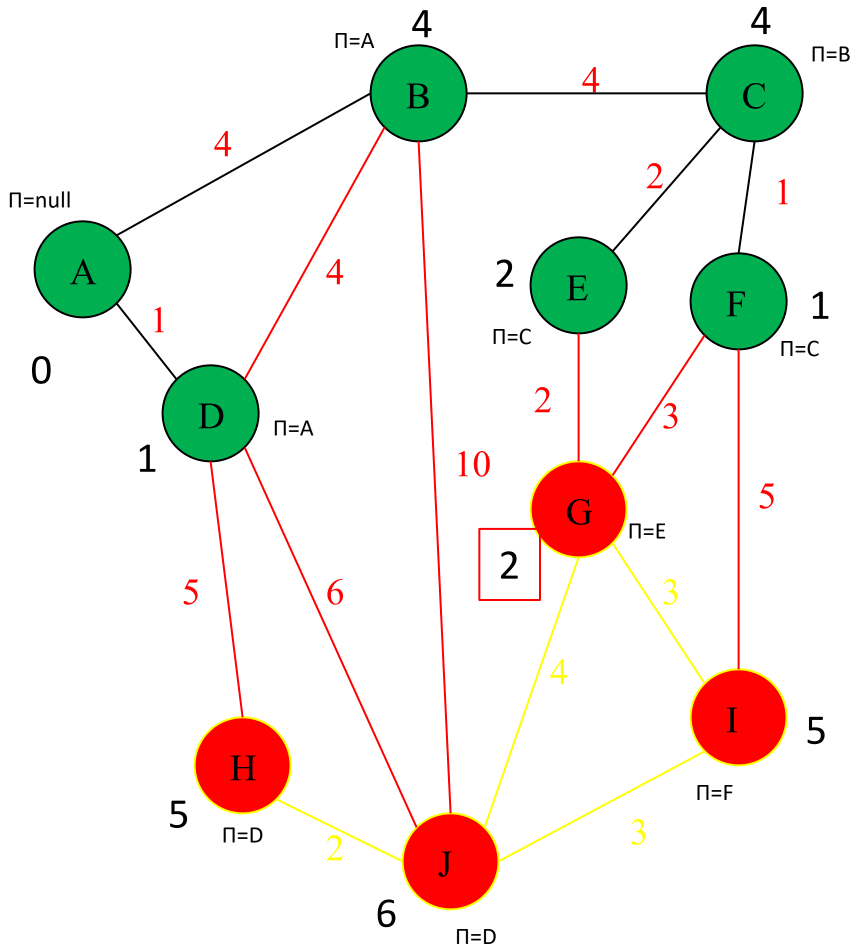


mstSet = {A,D,B,C,F}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

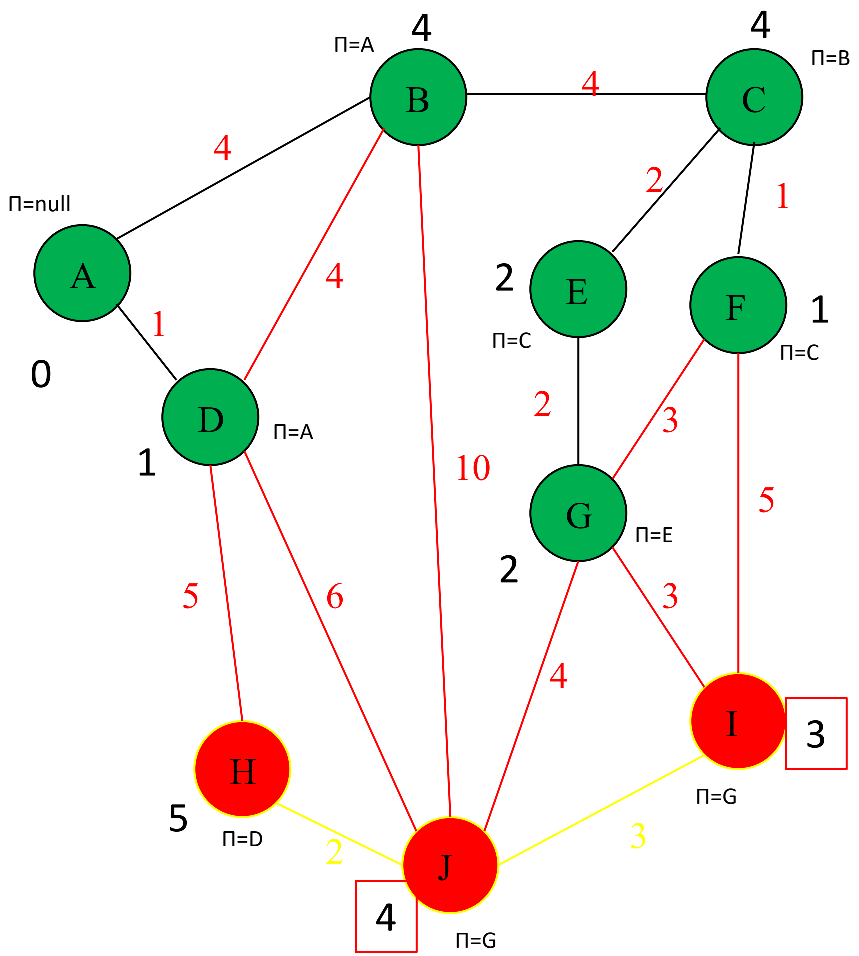


mstSet = {A,D,B,C,F,E}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

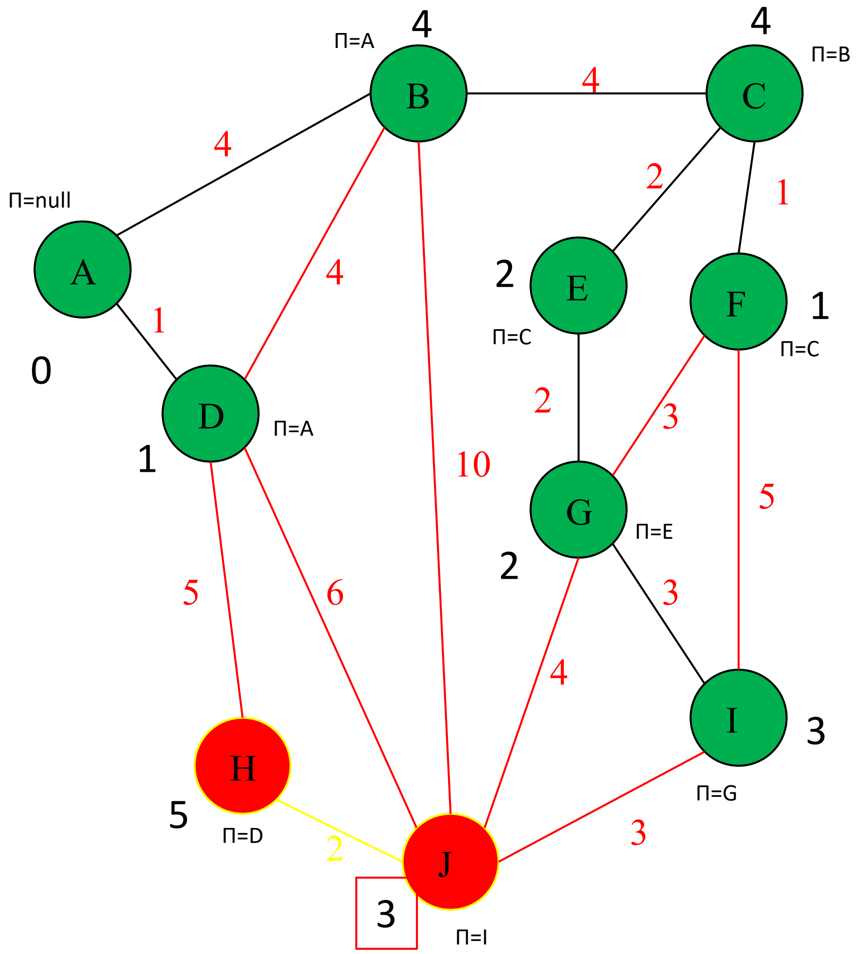


mstSet = {A,D,B,C,F,E,G}

The symbol Π is the parent of the vertex

Prim’s Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

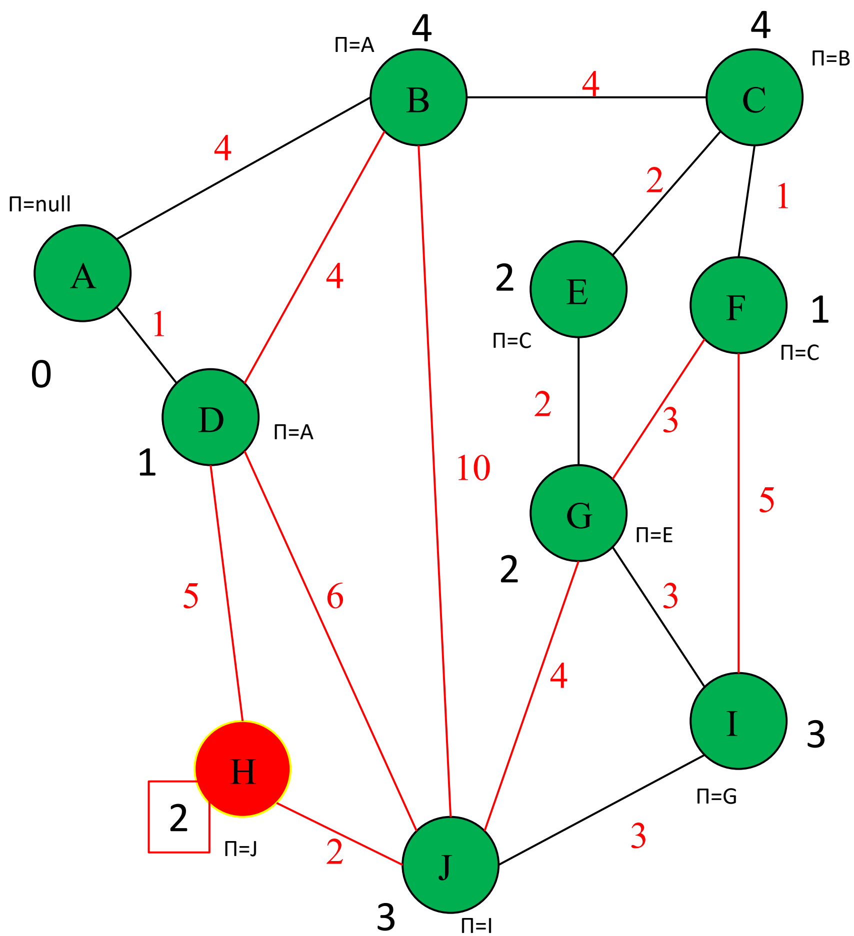


mstSet = {A,D,B,C,F,E,G,I}

The symbol Π is the parent of the vertex

Prim’s Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

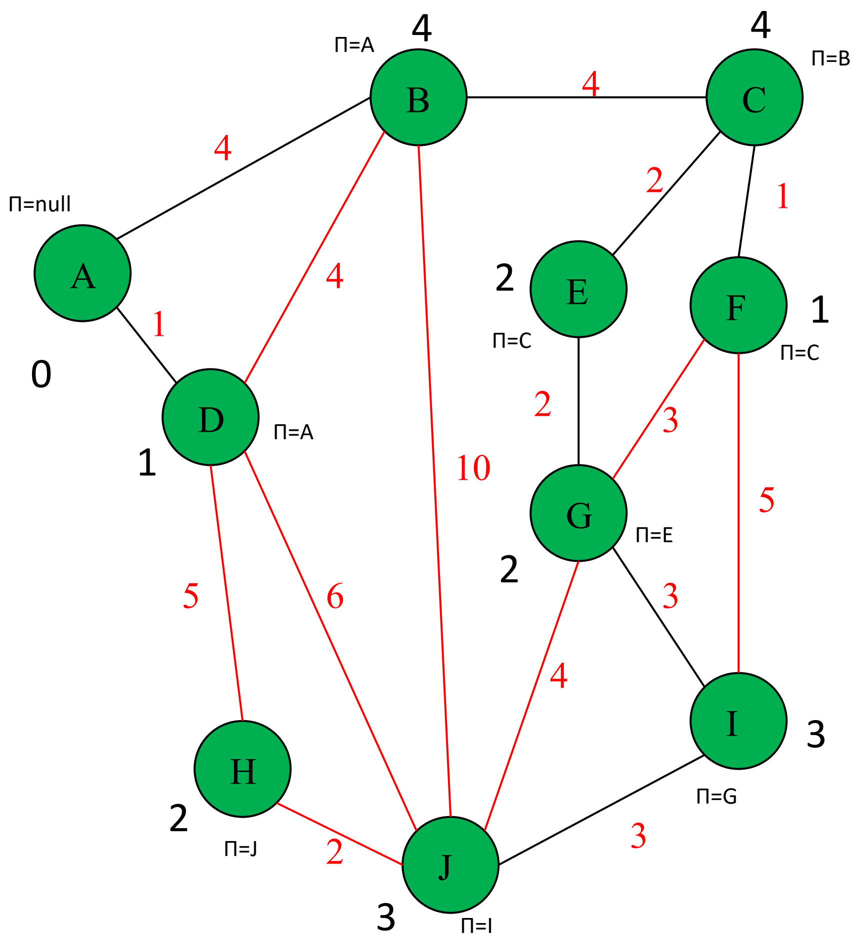


mstSet = {A,D,B,C,F,E,G,I,J}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered

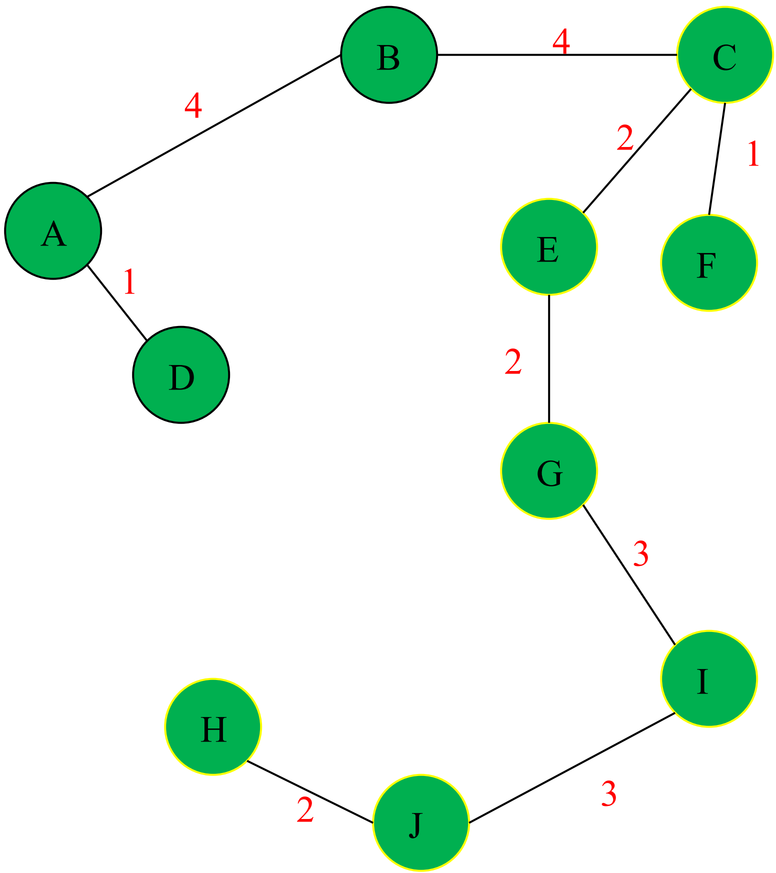


mstSet = {A,D,B,C,F,E,G,I,J,H}

The symbol Π is the parent of the vertex

Prim's Algorithm Example

-Think of the yellow nodes as cost ∞ (not valid)
-Only the red ones are being considered



Shows the **order in which vertices added** to the MST

Of course this set should have all the vertices in the graph

mstSet = {A,D,B,C,F,E,G,I,J,H}

Total cost = 22

Krusksal's and Prim's need not produce the same MST but cost will be the same

Prim's Algorithm

1: O(V)

1) Assign a key value to all vertices in the input graph. Initialize all **key values as INFINITE**.

2: O(1)

2) Assign **key value as 0 for the first vertex** so that it is picked first and call it ROOT
   a) The **parent** of the root is NIL

3: O(V log V)

3) Add all nodes into a MIN-HEAP Priority Queue → Q

The loop repeats V times

4) While Q not empty
....**a)** Pick a vertex *u* that has **minimum key value (slide 18)**
....**b)** Include *u* to mstSet (set that lists MST).
....**c)** Update key value of **all adjacent vertices (v)** of *u* (<u>*which is not i*</u>
- For every adjacent vertex *v*, if weight of edge *u-v* is **<u>les</u>**
  key value of *v*, update the key value as weight of edge
- *Make u the **parent** of v*
- ***Update the priority Q (slide 20)****(i.e., min heap as we have new weights)*

... **d)** Remove u from Q

4a: O(log V).

4(a) is called called V times in the loop.
Total cost = O(V log V)

4b: O(1).

4(c): O (log V)

4(c): Has to update for each edge in the graph.
Total cost = O(E log V)

Prim's Algorithm

1: O(V)

1) Assign a key value to all vertices in the input graph. Initialize all **key values as INFINITE**.

2: O(1)

2) Assign **key value as 0 for the first vertex** so that it is picked first and call it ROOT
   a) The **parent** of the root is NIL

3: O(V log V)

3) Add all nodes into a MIN-HEAP Priority Queue → Q

4) W

....**a**)

....**b**)

Total: O(V) + O(1) + O(V log V) + O(V log V) +O(1) + O(E log V) → O(E Log V)

....**c)** Update key value of **all adjacent vertices (v)** of *u (which is not i*

- For every adjacent vertex *v*, if weight of edge *u-v* is **les**
  key value of *v*, update the key value as weight of edge
- *Make u the **parent** of v*
- ***Update the priority Q (slide 20)****(i.e., min heap as we have new weights)*

... **d)** Remove u from Q

4(a) is called called V times in the loop.
Total cost = O(V log V)

4b: O(1).

4(c): O (log V)

4(c): Has to update for each edge in the graph.
Total cost = O(E log V)

Algorithms for Obtaining the Minimum Spanning Tree

- **Kruskal's Algorithm**
- **Prim's Algorithm**

Both of these are Greedy Algorithms


That's all Folks!
Any Question?