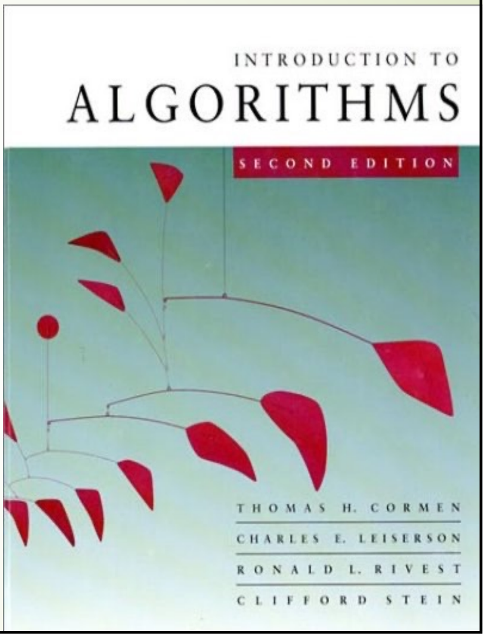
Minimum Spanning Trees

SUBJECT-CODE : KCS-503

Dr. Ragini Karwayun



1

Minimum Spanning Tree

<u>What is a Spanning Tree?</u>

Given an undirected and connected graph G=(V,E), a spanning tree of the graph G is a tree that spans G (that is, it includes every vertex of G) and is a subgraph of G (every edge in the tree belongs to G)

2

Minimum Spanning Tree

What is a Minimum Spanning Tree?

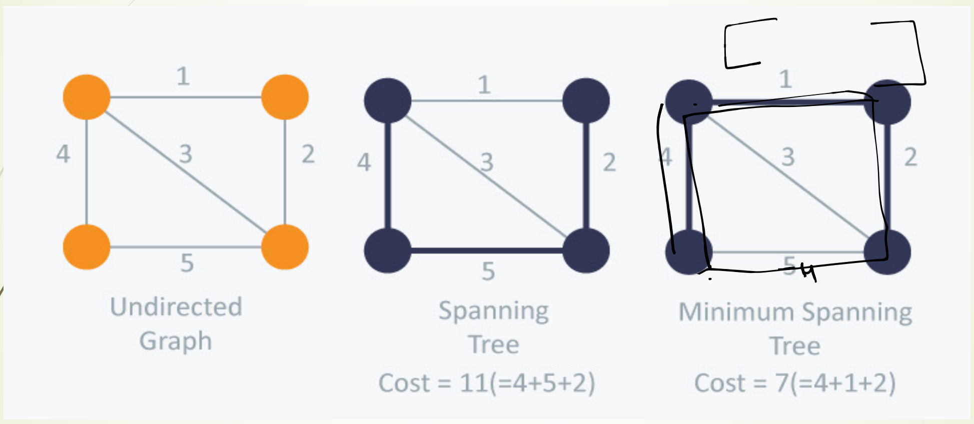
weight are unique.
① 4

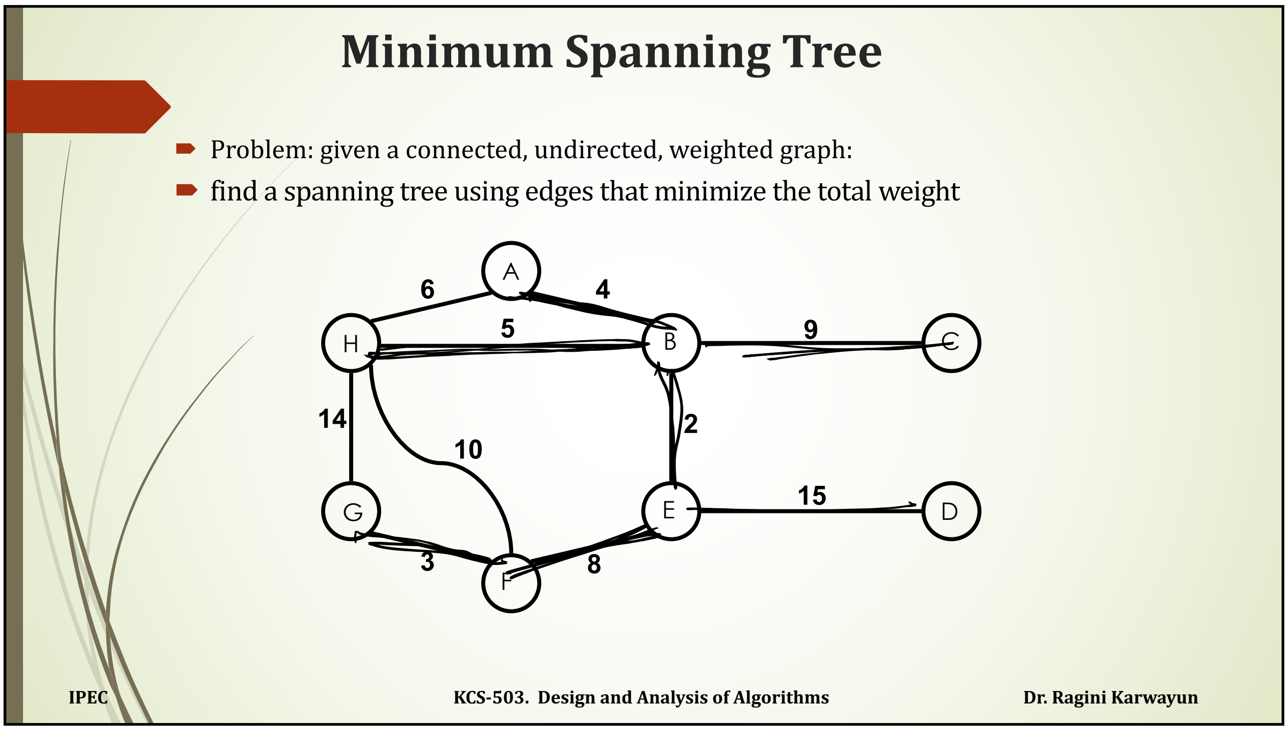
- **The cost of the spanning tree is the sum of the weights of all the edges in the tree. There can be many spanning trees. Minimum spanning tree is the spanning tree where the cost is minimum among all the spanning trees. There also can be many minimum spanning trees.**

(1 2 3)
only be one.

- **Minimum spanning tree has direct application in the design of networks. It is used in algorithms approximating the travelling salesman problem, multi-terminal minimum cut problem and minimum-cost weighted perfect matching.**

Minimum Spanning Tree



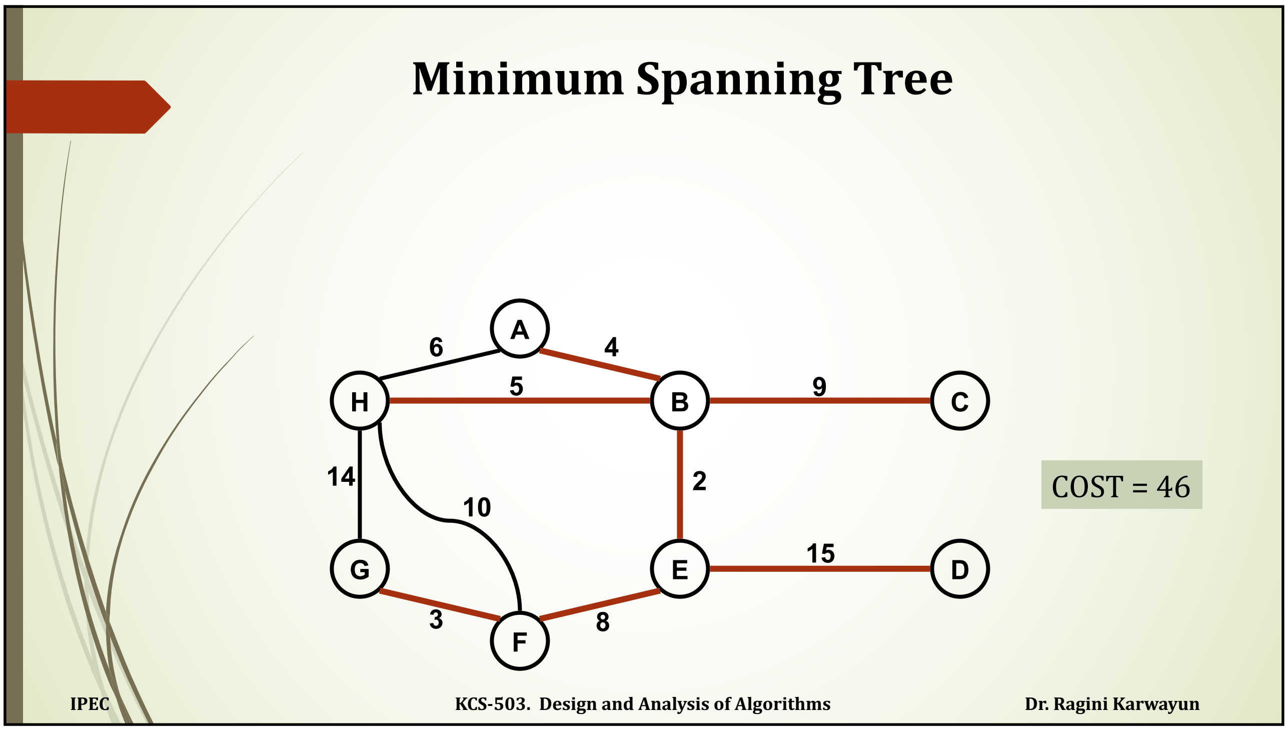


Minimum Spanning Tree

- Problem: given a connected, undirected, weighted graph:
- find a spanning tree using edges that minimize the total weight

A
B
C
D
E
F
G
H
6
4
5
9
14
10
2
15
3
8

5

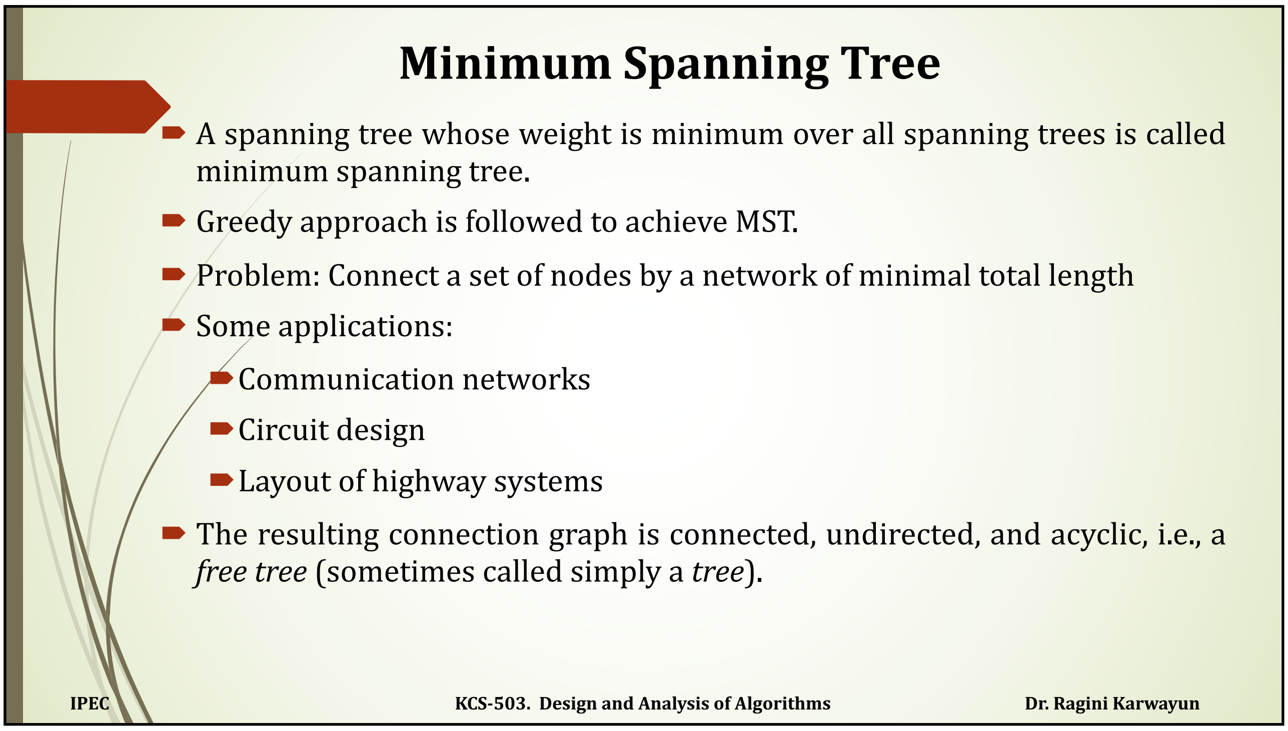


Minimum Spanning Tree

A
B
C
D
E
F
G
H
6
4
5
9
14
10
2
15
3
8

COST = 46

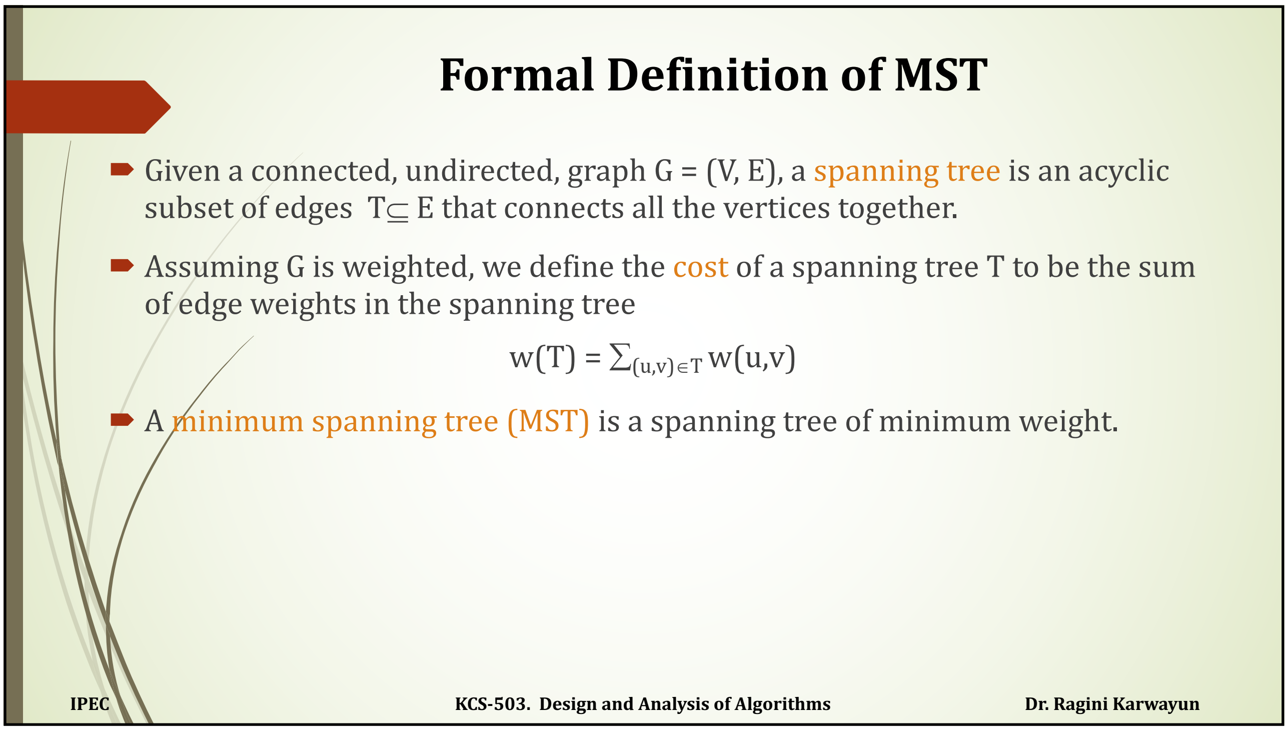
6



Minimum Spanning Tree

- A spanning tree whose weight is minimum over all spanning trees is called minimum spanning tree.
- Greedy approach is followed to achieve MST.
- Problem: Connect a set of nodes by a network of minimal total length
- Some applications:
  - Communication networks
  - Circuit design
  - Layout of highway systems
- The resulting connection graph is connected, undirected, and acyclic, i.e., a *free tree* (sometimes called simply a *tree*).

7



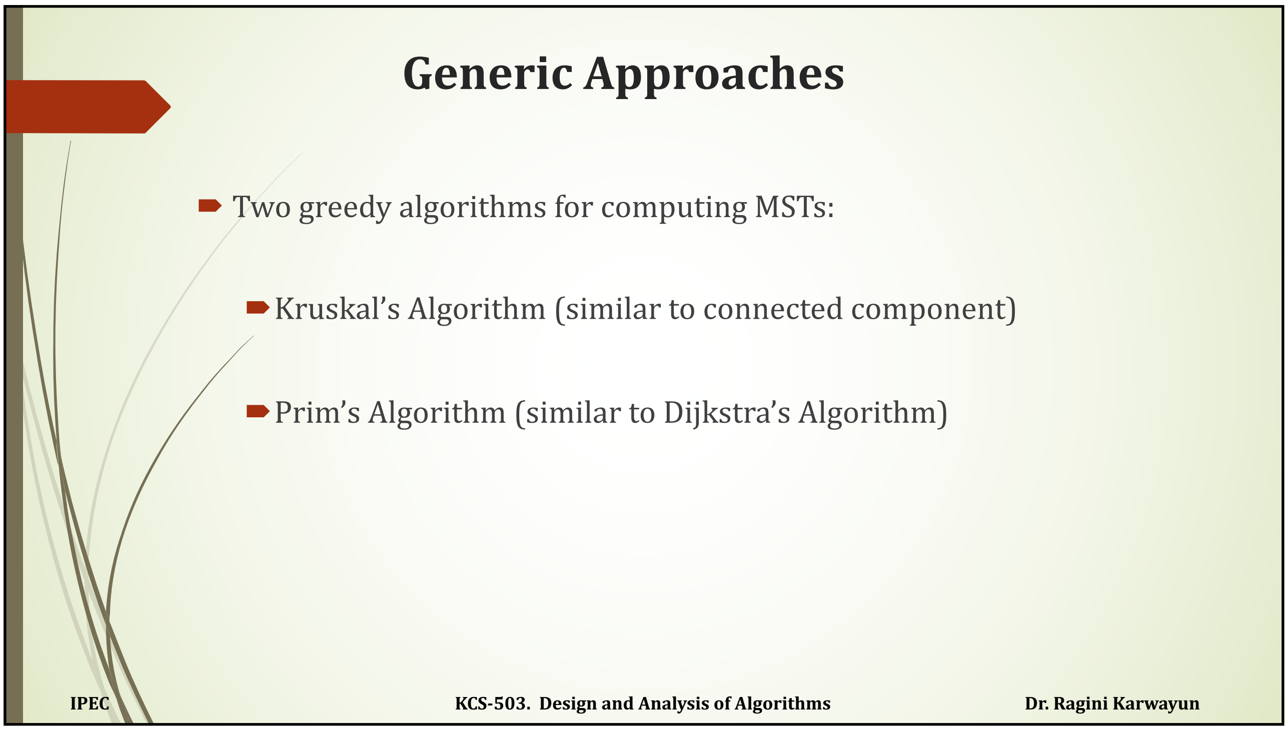
Formal Definition of MST

- Given a connected, undirected, graph G = (V, E), a spanning tree is an acyclic subset of edges $T \subseteq E$ that connects all the vertices together.
- Assuming G is weighted, we define the cost of a spanning tree T to be the sum of edge weights in the spanning tree

$$w(T) = \sum_{(u,v) \in T} w(u,v)$$

- A minimum spanning tree (MST) is a spanning tree of minimum weight.

8

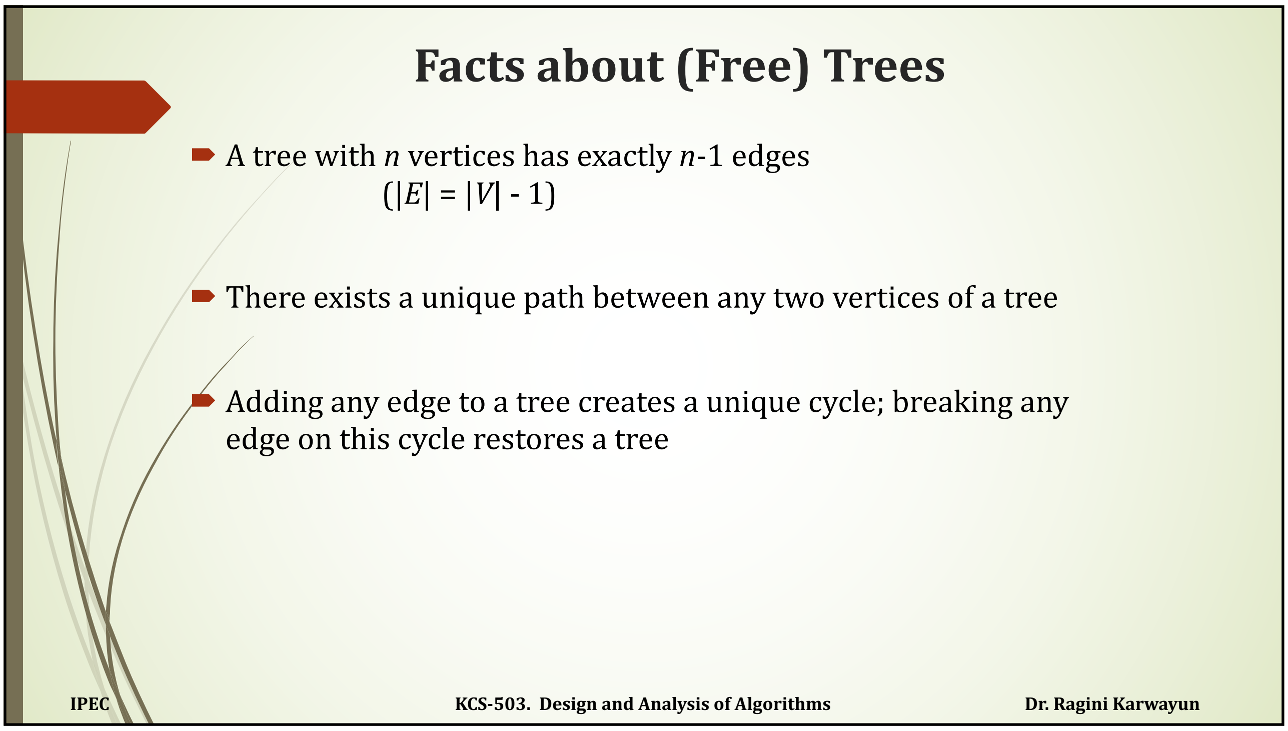


Generic Approaches

- Two greedy algorithms for computing MSTs:
  - Kruskal's Algorithm (similar to connected component)
  - Prim's Algorithm (similar to Dijkstra's Algorithm)

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

9

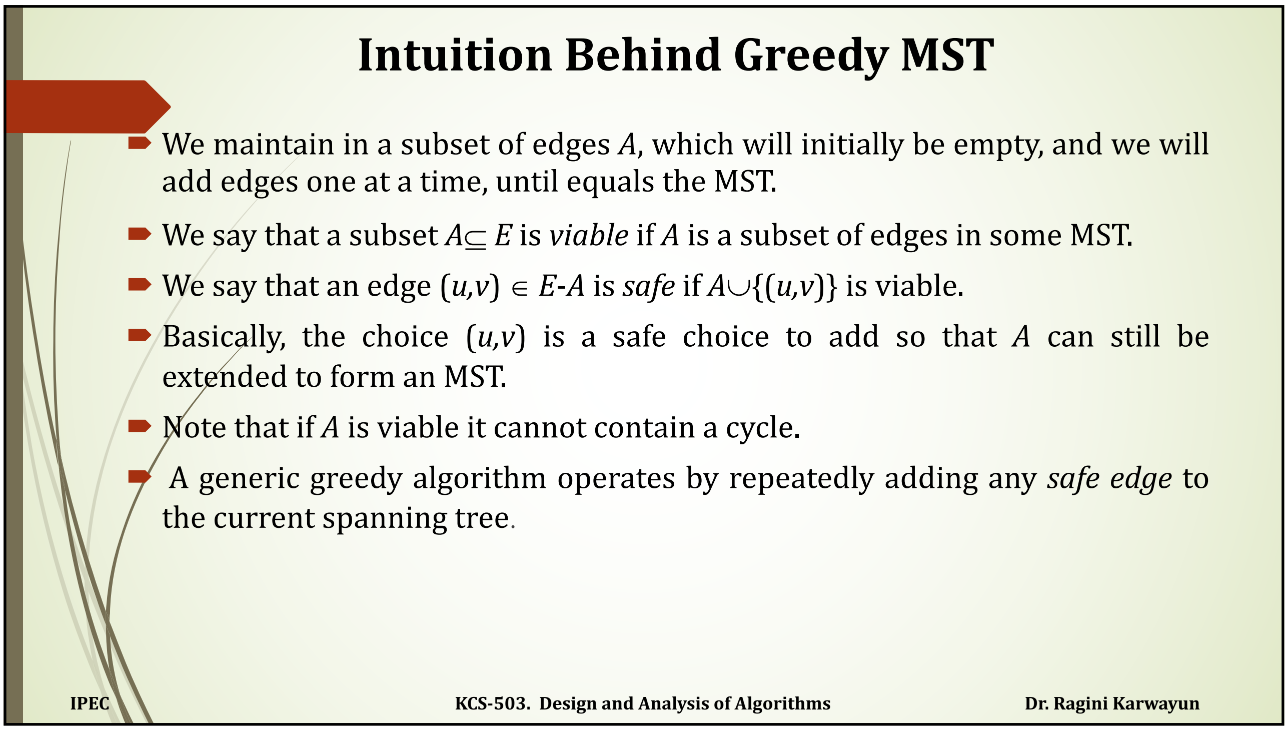


Facts about (Free) Trees

- A tree with *n* vertices has exactly *n*-1 edges
  ($|E| = |V| - 1$)
- There exists a unique path between any two vertices of a tree
- Adding any edge to a tree creates a unique cycle; breaking any edge on this cycle restores a tree

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

10

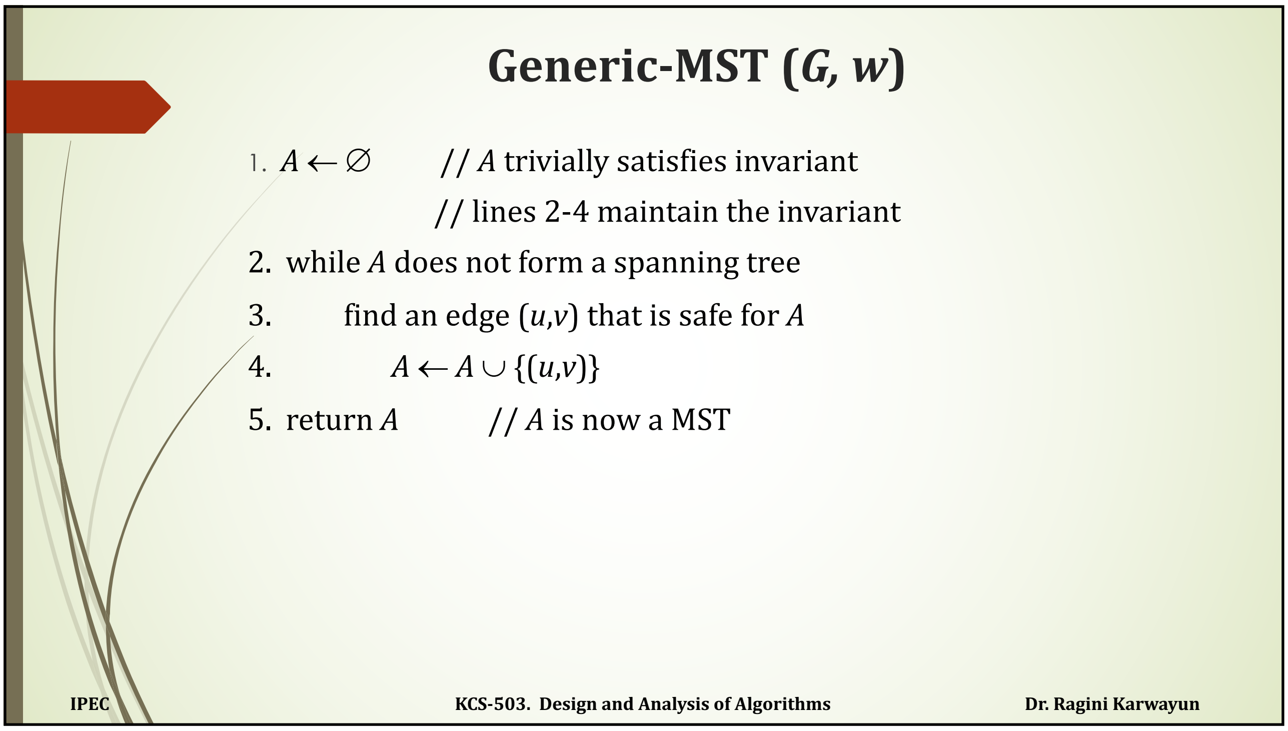


Intuition Behind Greedy MST

- We maintain in a subset of edges A, which will initially be empty, and we will add edges one at a time, until equals the MST.
- We say that a subset $A \subseteq E$ is *viable* if A is a subset of edges in some MST.
- We say that an edge $(u,v) \in E\text{-}A$ is *safe* if $A \cup \{(u,v)\}$ is viable.
- Basically, the choice (u,v) is a safe choice to add so that A can still be extended to form an MST.
- Note that if A is viable it cannot contain a cycle.
- A generic greedy algorithm operates by repeatedly adding any *safe edge* to the current spanning tree.

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

11



Generic-MST (G, w)

```
1. $A \leftarrow \varnothing$          // $A$ trivially satisfies invariant
                      // lines 2-4 maintain the invariant
2. while $A$ does not form a spanning tree
3.       find an edge $(u,v)$ that is safe for $A$
4.             $A \leftarrow A \cup \{(u,v)\}$
5. return $A$            // $A$ is now a MST
```

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

12

MST-Prim(*G, w, r*)

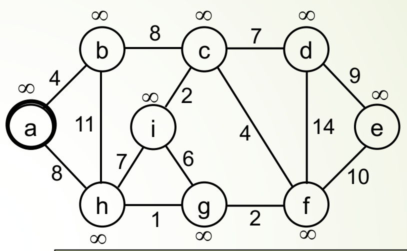
```
1.  $Q \leftarrow V[G]$
2.  for each vertex $u \in Q$        // initialization: $O(V)$ time
3.      do $key[u] \leftarrow \infty$
4.         $\pi[u] \leftarrow$ NIL
4.  $key[r] \leftarrow 0$            // start at the root
5.  $\pi[r] \leftarrow$ NIL          // set parent of $r$ to be NIL
6.  while $Q \neq \varnothing$           // until all vertices in MST
7.        $u \leftarrow$ Extract-Min($Q$)     // vertex with lightest edge
8.       for each $v \in adj[u]$
9.             if $v \in Q$ and $w(u,v) < key[v]$
10.               $\pi[v] \leftarrow u$
11.                  $key[v] \leftarrow w(u,v)$   // new lighter edge out of $v$
12.                  decrease_Key($Q, v, key[v]$)
```

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

13

PRIM(V, E, w, r)

```
1.  Q ← $\varnothing$
2.  for each u $\in$ V
3.      do key[u] ← $\infty$
4.         $\pi$[u] ← NIL
5.         INSERT(Q, u)
6.  DECREASE-KEY(Q, r, 0)
7.  while Q $\neq \varnothing$
8.      do u ← EXTRACT-MIN(Q).
9.          for each v $\in$ Adj[u]
10.         do if v $\in$ Q and w(u, v) < key[v]
11.              then $\pi$[v] ← u
12.                 DECREASE-KEY(Q, v, w(u, v))
```



Q = {a, b, c, d, e, f, g, h, i}

$V_A = \varnothing$.

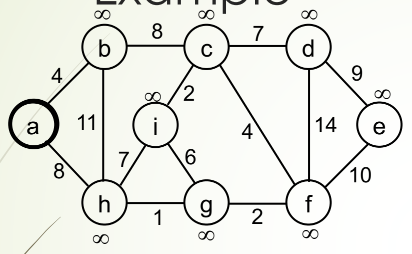
a,	b,	c,	d,	e,	f,	g,	h,	i
∞	∞	∞	∞	∞	∞	∞	∞	∞

DECREASE-KEY(Q, a , 0)

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

14

Example

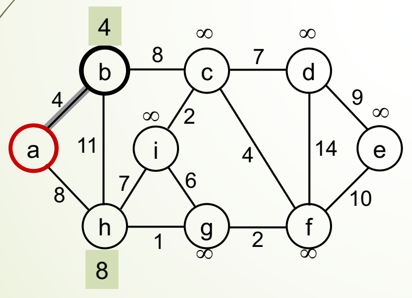


Q = {a, b, c, d, e, f, g, h, i}

Extract-MIN(Q) ⇒ a

V_A = {a}.

a,	b,	c,	d,	e,	f,	g,	h,	i
0	∞	∞	∞	∞	∞	∞	∞	∞



key [b] = 4 π [b] = a

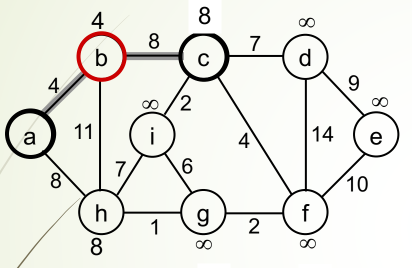
key [h] = 8 π [h] = a

Q = {b, c, d, e, f, g, h, i}

V_A = {a}. Extract-MIN(Q) ⇒ b

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	∞	∞	∞	∞	∞	8	∞

Example



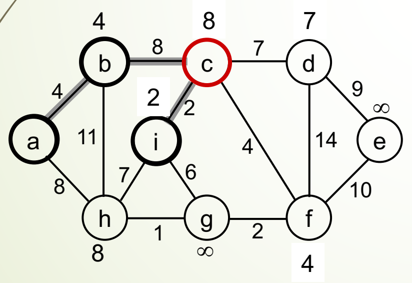
key [c] = 8 π [c] = b

key [h] = 8 π [h] = a - unchanged

Q = {c, d, e, f, g, h, i} V_A = {a, b}

Extract-MIN(Q) ⇒ c

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	∞	∞	∞	∞	8	∞



key [d] = 7 π [d] = c

key [f] = 4 π [f] = c

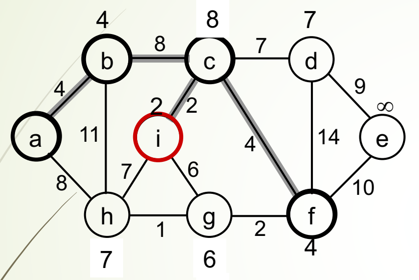
key [i] = 2 π [i] = c

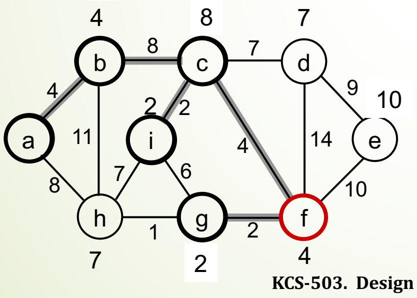
Q = {d, e, f, g, h, i} V_A = {a, b, c}

Extract-MIN(Q) ⇒ i

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	∞	4	∞	8	2

Example





key [h] = 7   π [h] = i
key [g] = 6   π [g] = I
Q = {d, e, f, g, h} V_A = {a, b, c, i}
Extract-MIN(Q) ⇒ f

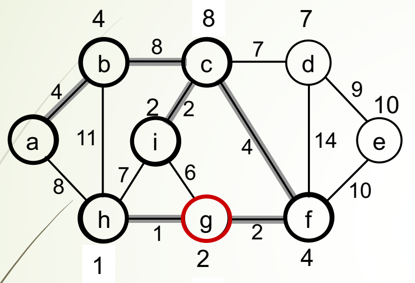
a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	∞	4	6	7	2

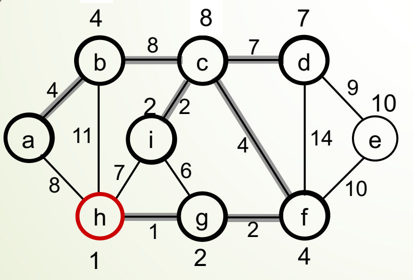
key [g] = 2   π [g] = f
key [d] = 7   π [d] = c unchanged
key [e] = 10   π [e] = f
Q = {d, e, g, h} V_A = {a, b, c, i, f}
Extract-MIN(Q) ⇒ g

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	10	4	6	7	2

17

Example





key [h] = 1   π [h] = g
Q = {d, e, h} V_A = {a, b, c, i, f, g}
Extract-MIN(Q) ⇒ h

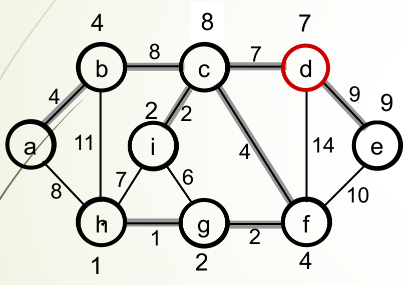
a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	10	4	6	7	2

Q = {d, e} V_A = {a, b, c, i, f, g, h}
Extract-MIN(Q) ⇒ d

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	10	4	6	7	2

18

Example



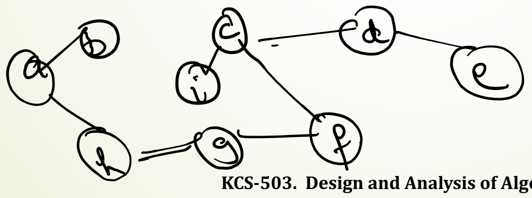
key [e] = 9   π [e] = d

Q = {e} V_A = {a, b, c, i, f, g, h, d}

Extract-MIN(Q) ⇒ e

a,	b,	c,	d,	e,	f,	g,	h,	i
0	4	8	7	10	4	6	7	2

Q = ∅ V_A = {a, b, c, i, f, g, h, d, e}



PRIM(V, E, w, r)

```
1.  Q ← ∅
2.  for each u ∈ V
3.       do key[u] ← ∞
4.          π[u] ← NIL
5.          INSERT(Q, u)
6.  DECREASE-KEY(Q, r, 0)        ► key[r] ← 0
7.  while Q ≠ ∅
8.       do u ← EXTRACT-MIN(Q)
9.            for each v ∈ Adj[u]
10.                do if v ∈ Q and w(u, v) < key[v]
11.                     then π[v] ← u
12.                          DECREASE-KEY(Q, v, w(u, v))
```

Total time: $O(V\lg V + E\lg V) = O(E\lg V)$

Lines 2–5: $O(V)$ if Q is implemented as a min-heap

Line 7: Executed V times

Line 8: Takes $O(\lg V)$

Lines 7–8: Min-heap operations: $O(V\lg V)$

Line 9: Executed O(E) times

Line 10: Constant

Lines 11–12: Takes $O(\lg V)$

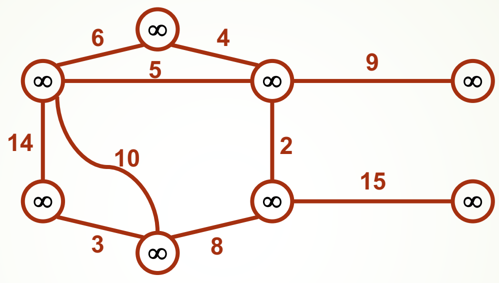
Lines 9–12: $O(E\lg V)$

Performance of PRIM's Algorithm

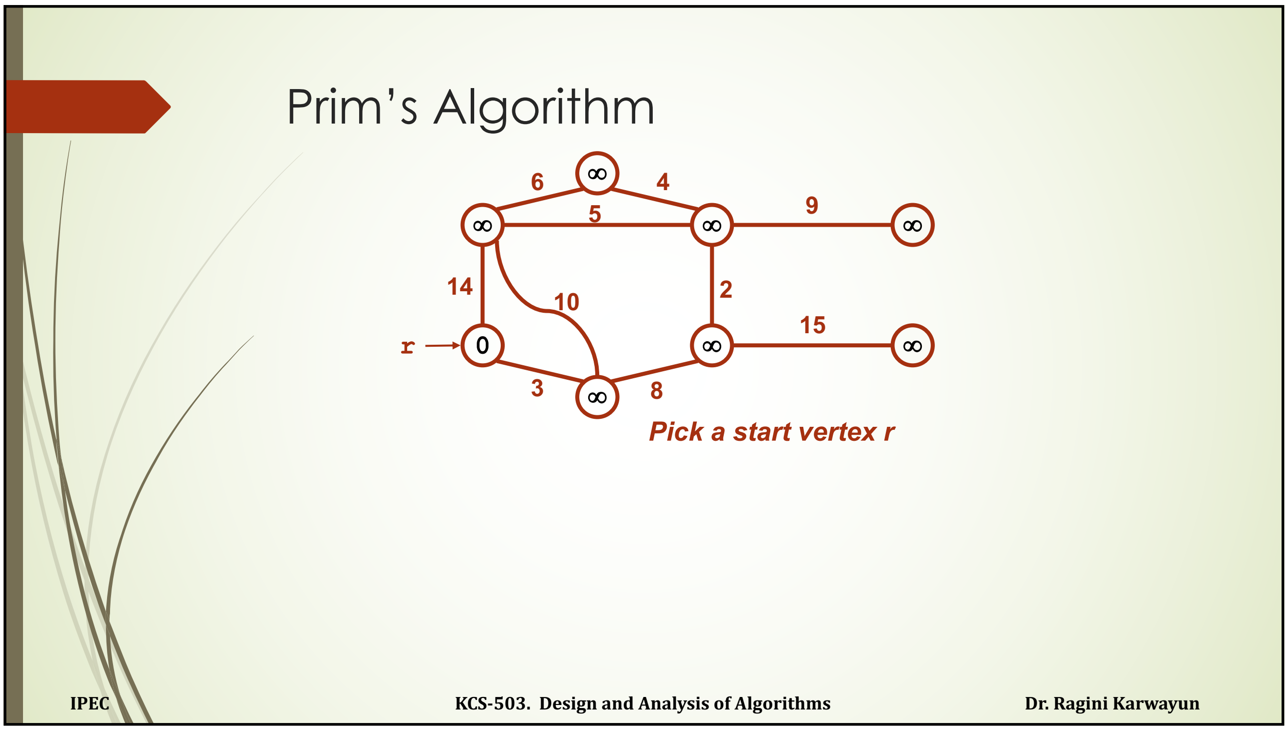
Total time: O(VlgV + ElgV) = O(ElgV)

- Time Complexity of the above program is O(V^2) ,if the input graph is represented using adjacency list,
- The time complexity of Prim's algorithm can be reduced to O(E log V) with the help of binary heap.
- The time complexity of Prim's algorithm can be reduced to O(E + log V) with the help of Fibonacci Heap

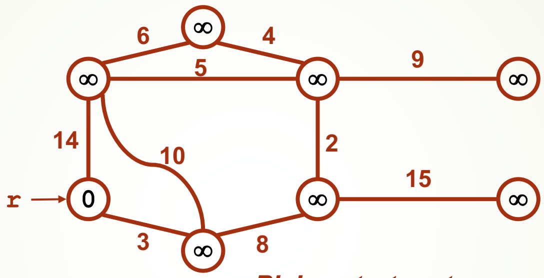
Prim's Algorithm



Run on example graph



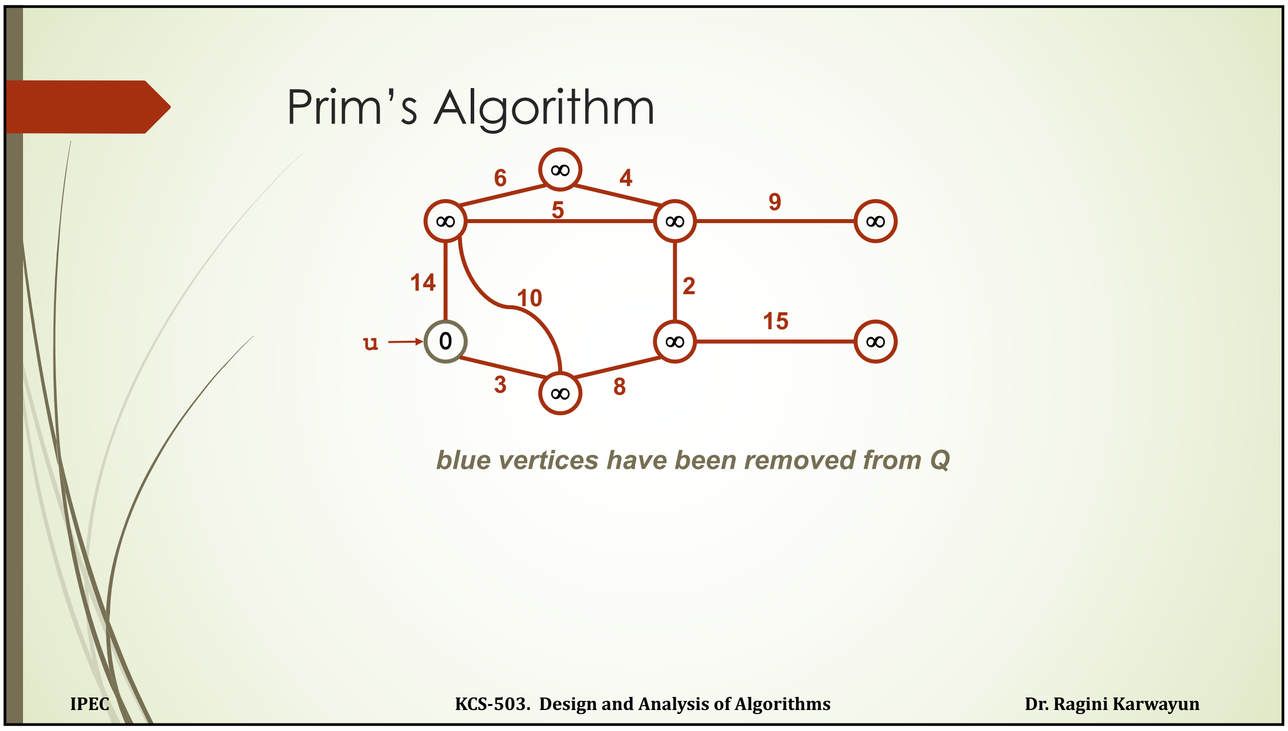
Prim's Algorithm



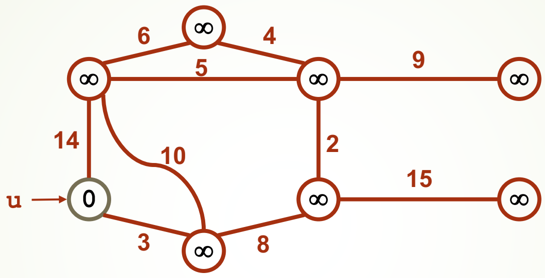
Pick a start vertex r

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

23



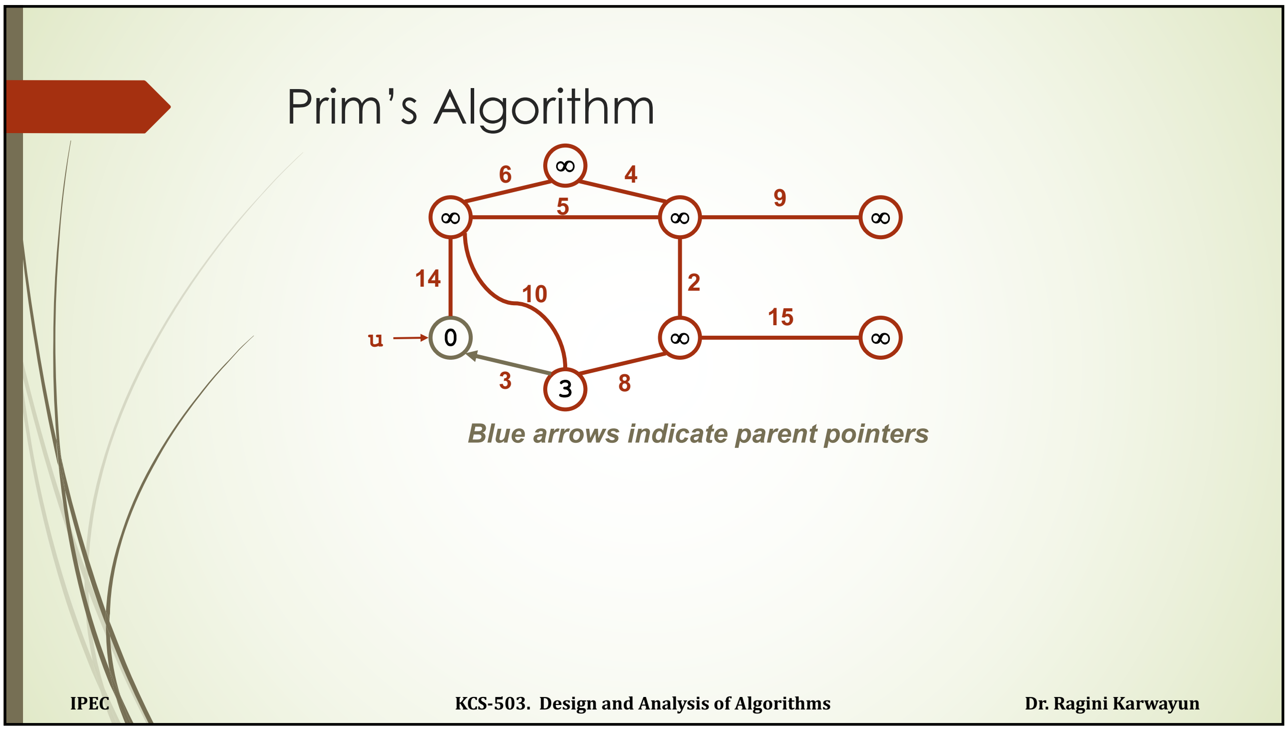
Prim's Algorithm



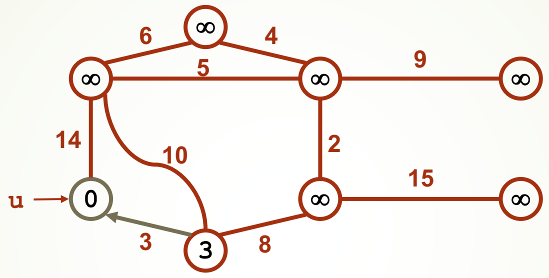
blue vertices have been removed from Q

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

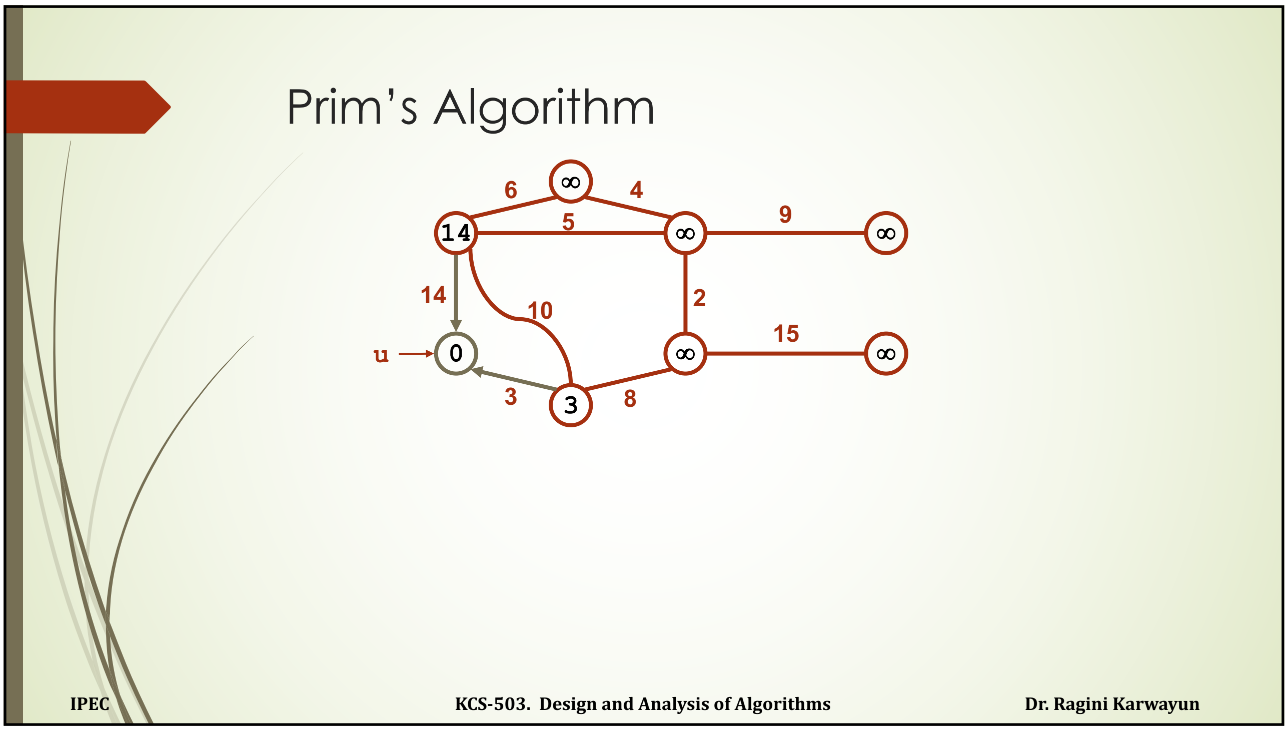
24



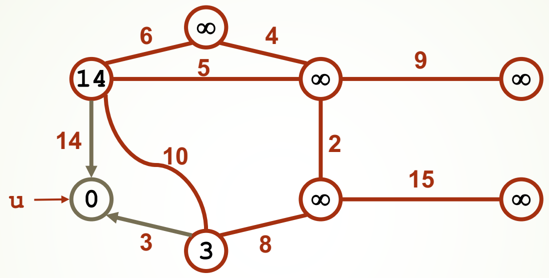
Prim's Algorithm

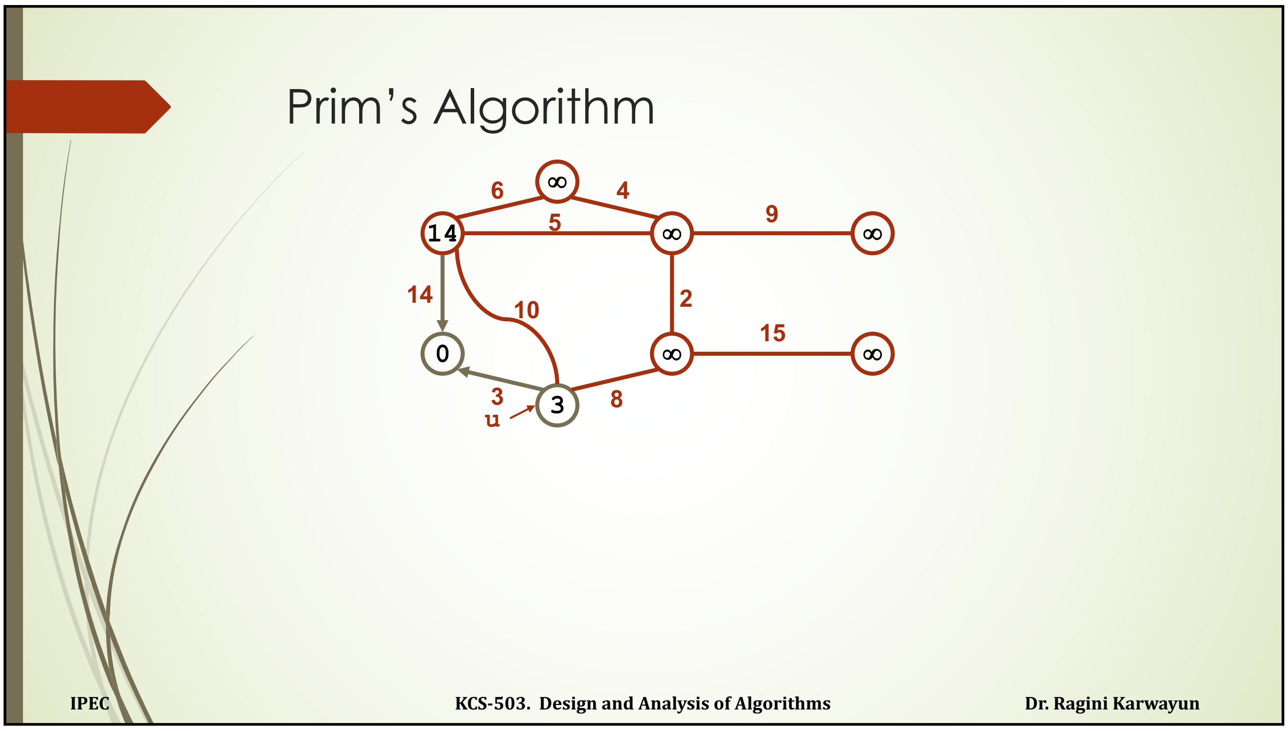


Blue arrows indicate parent pointers

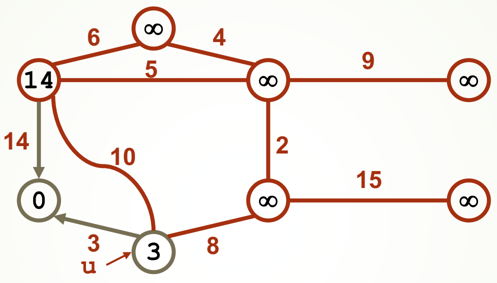


Prim's Algorithm



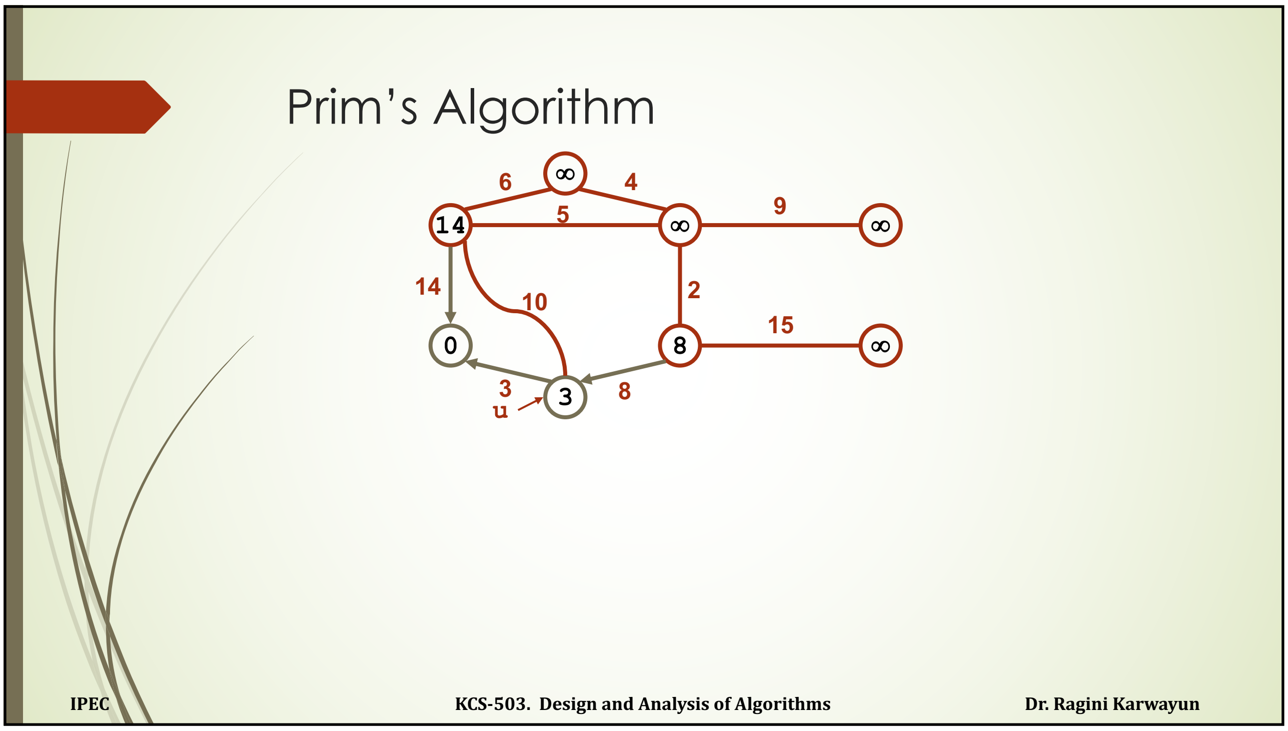


Prim’s Algorithm

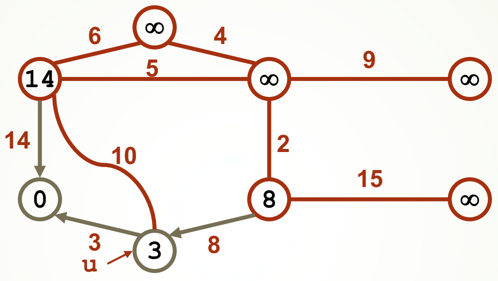


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

27

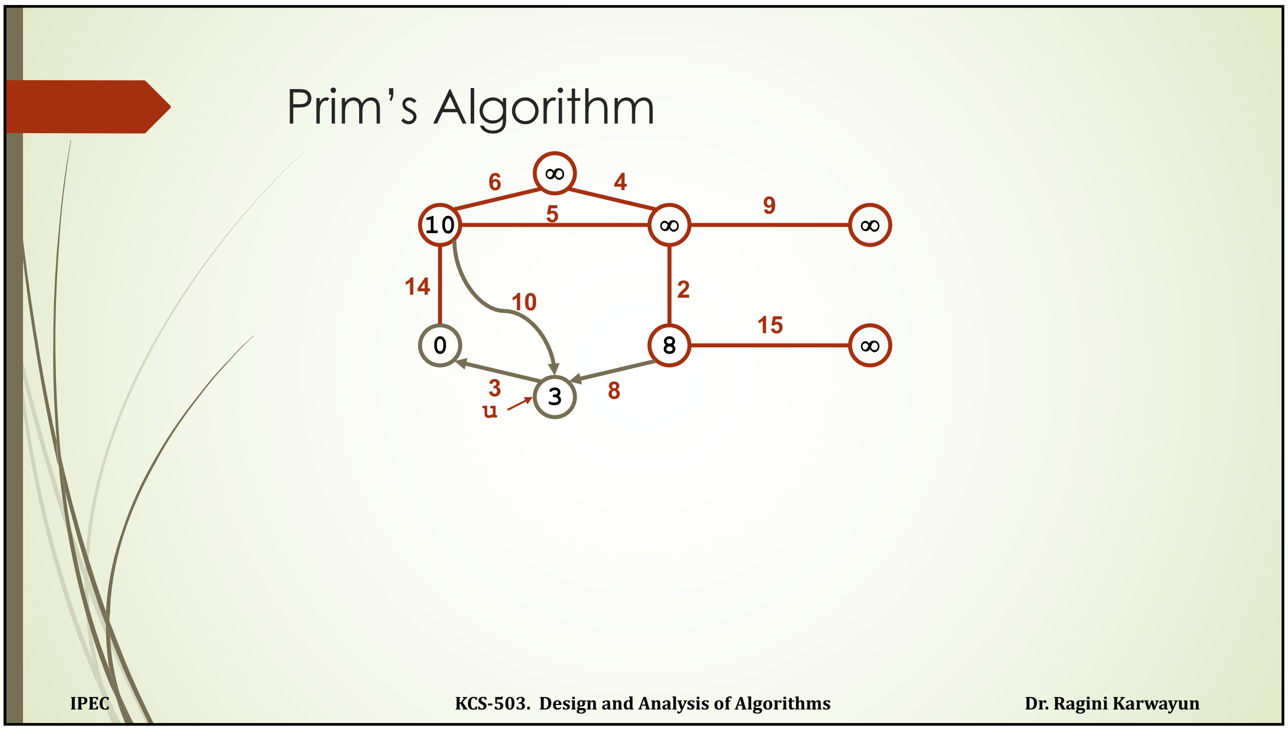


Prim’s Algorithm

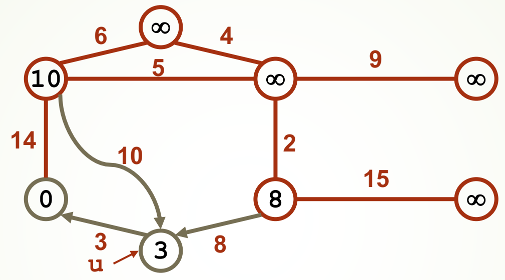


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

28

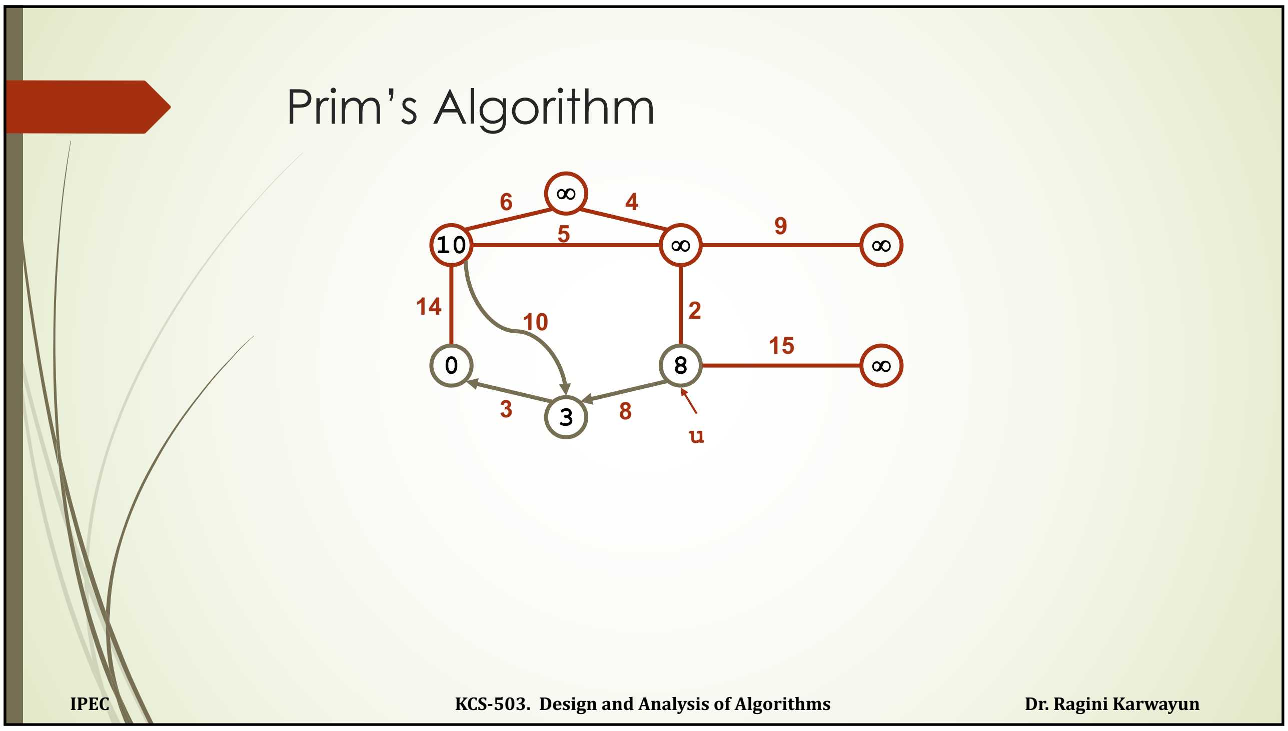


Prim's Algorithm

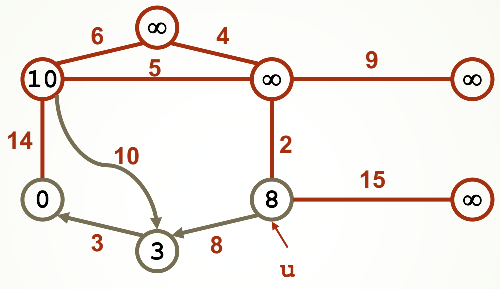


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

29

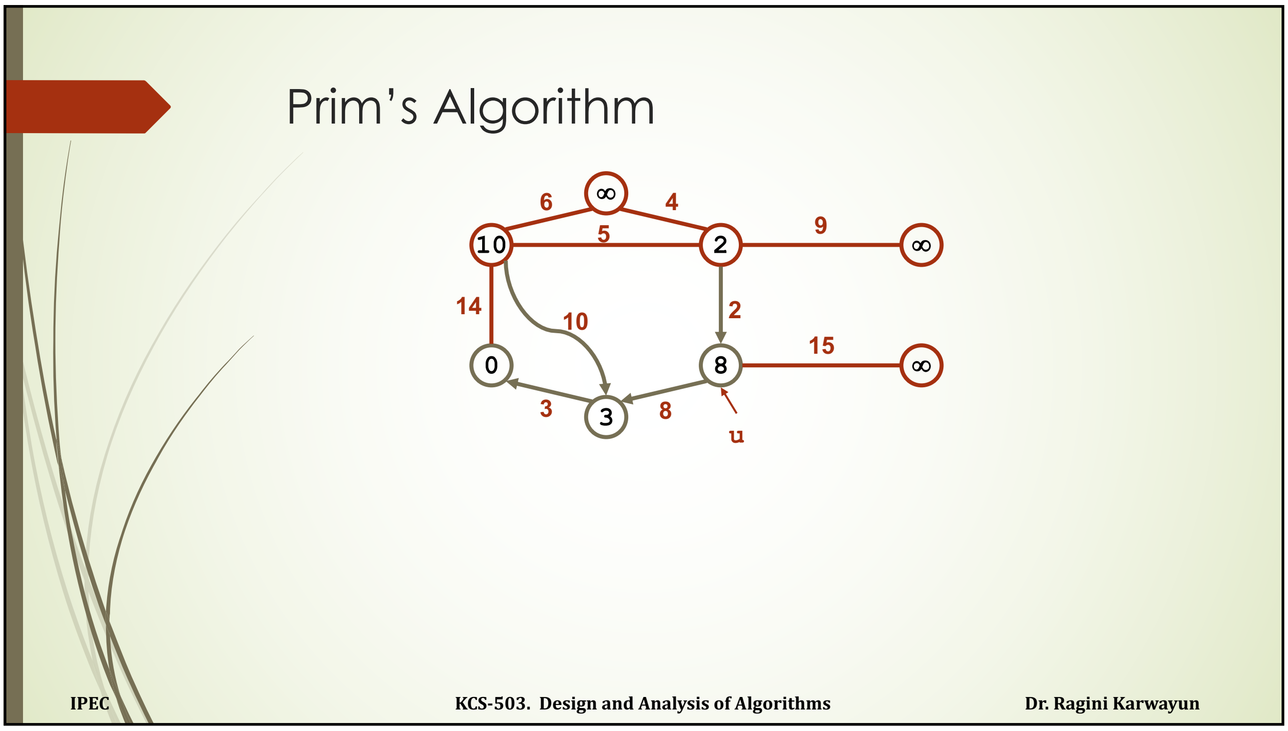


Prim's Algorithm

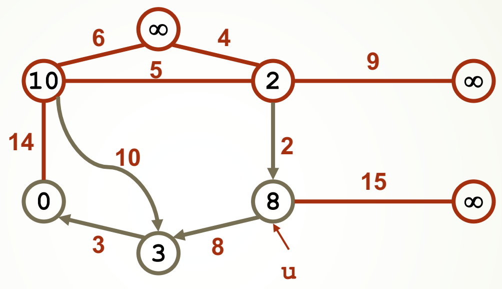


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

30

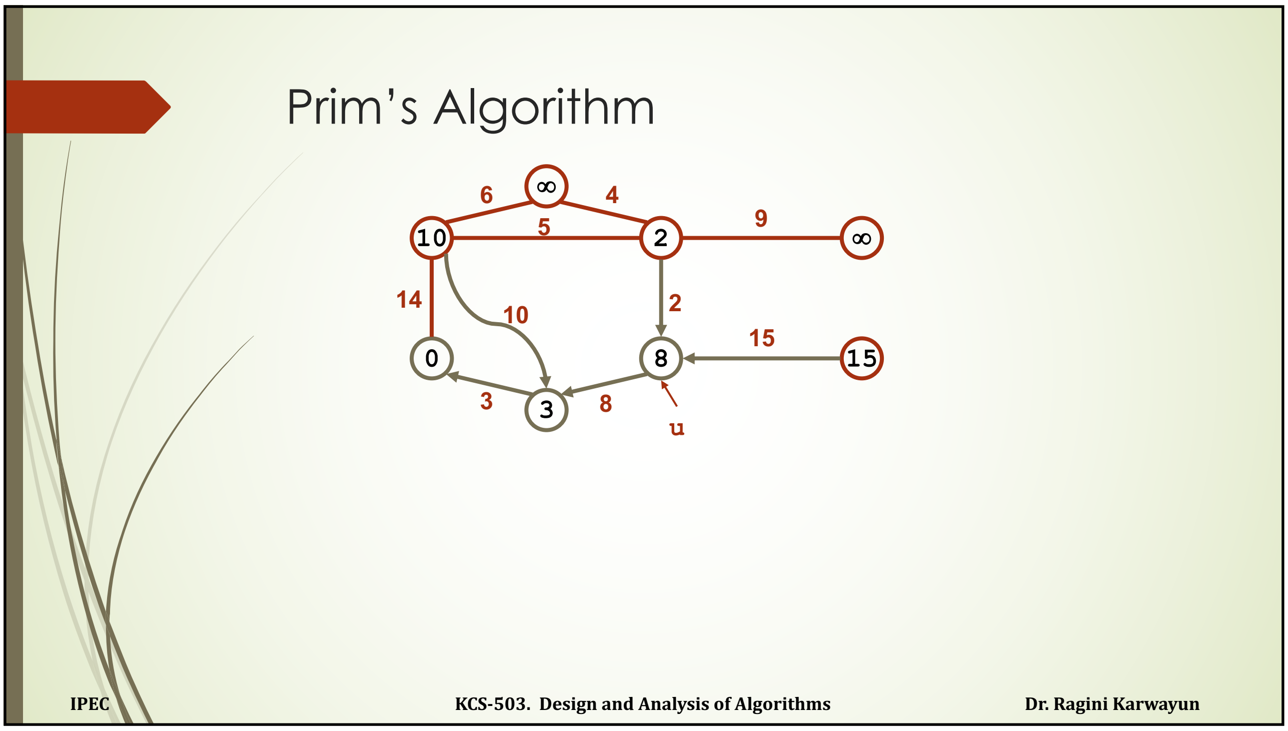


Prim's Algorithm

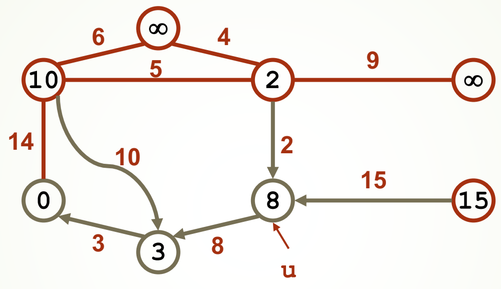


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

31

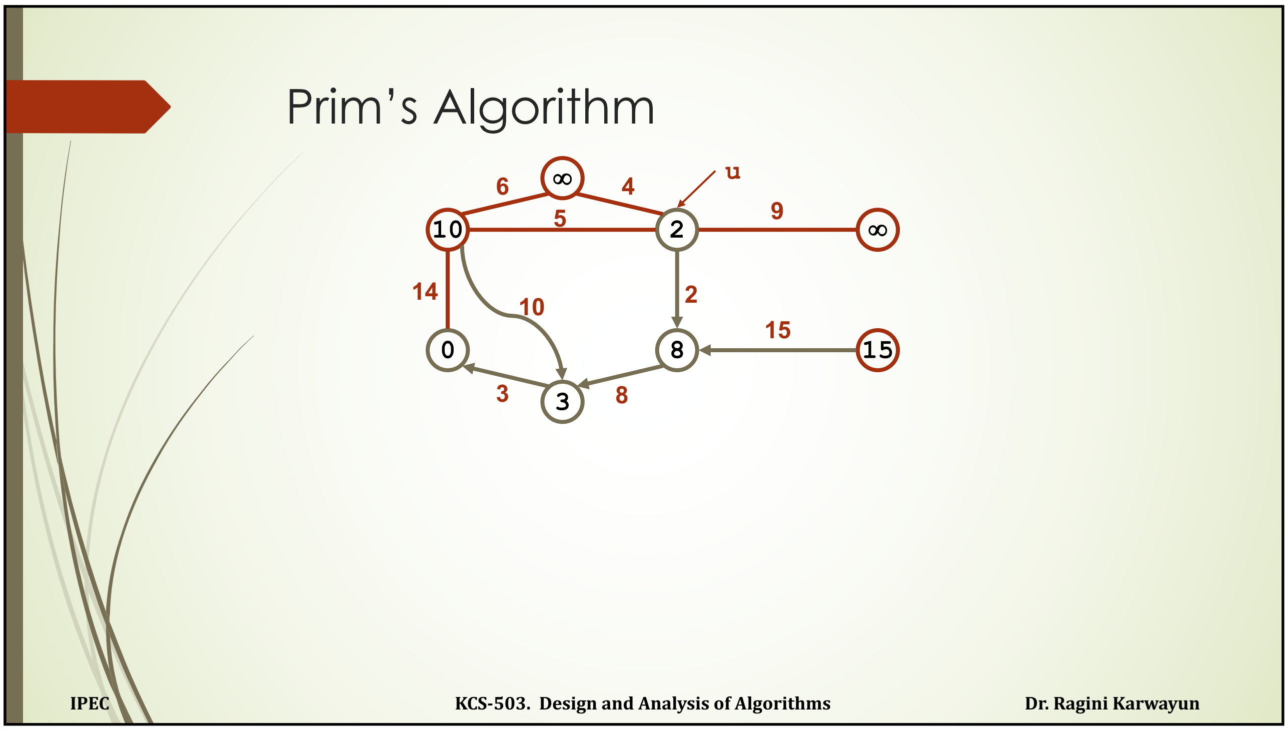


Prim's Algorithm

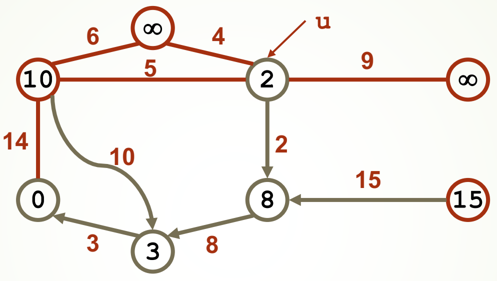


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

32

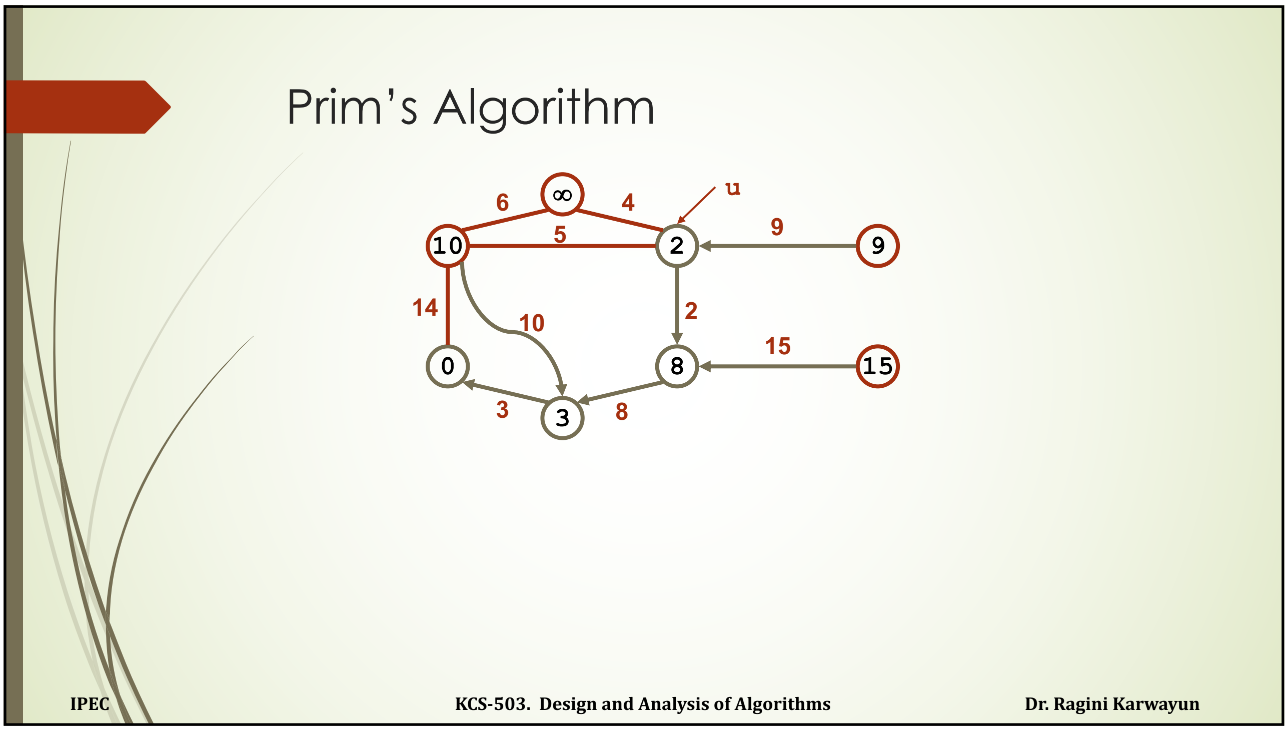


Prim’s Algorithm

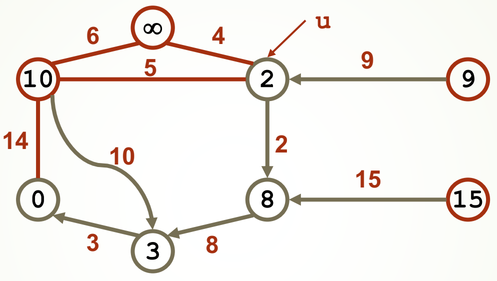


IPEC   KCS-503. Design and Analysis of Algorithms   Dr. Ragini Karwayun

33



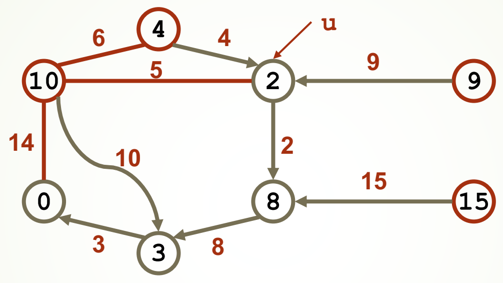
Prim’s Algorithm



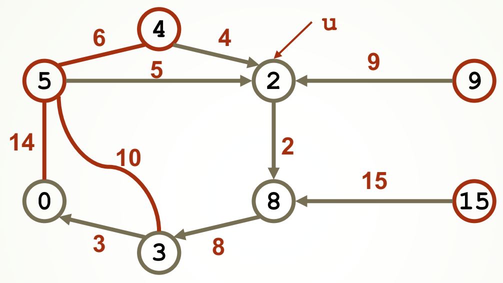
IPEC   KCS-503. Design and Analysis of Algorithms   Dr. Ragini Karwayun

34

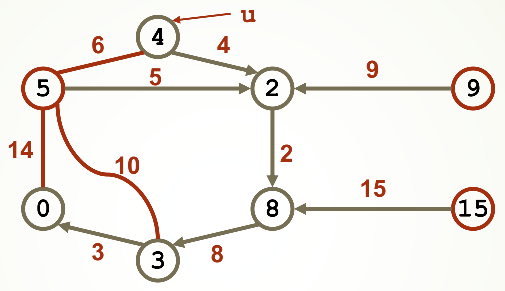
Prim's Algorithm



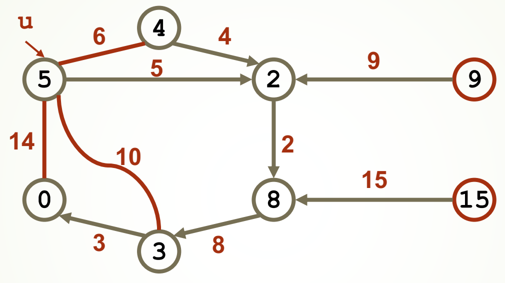
Prim's Algorithm

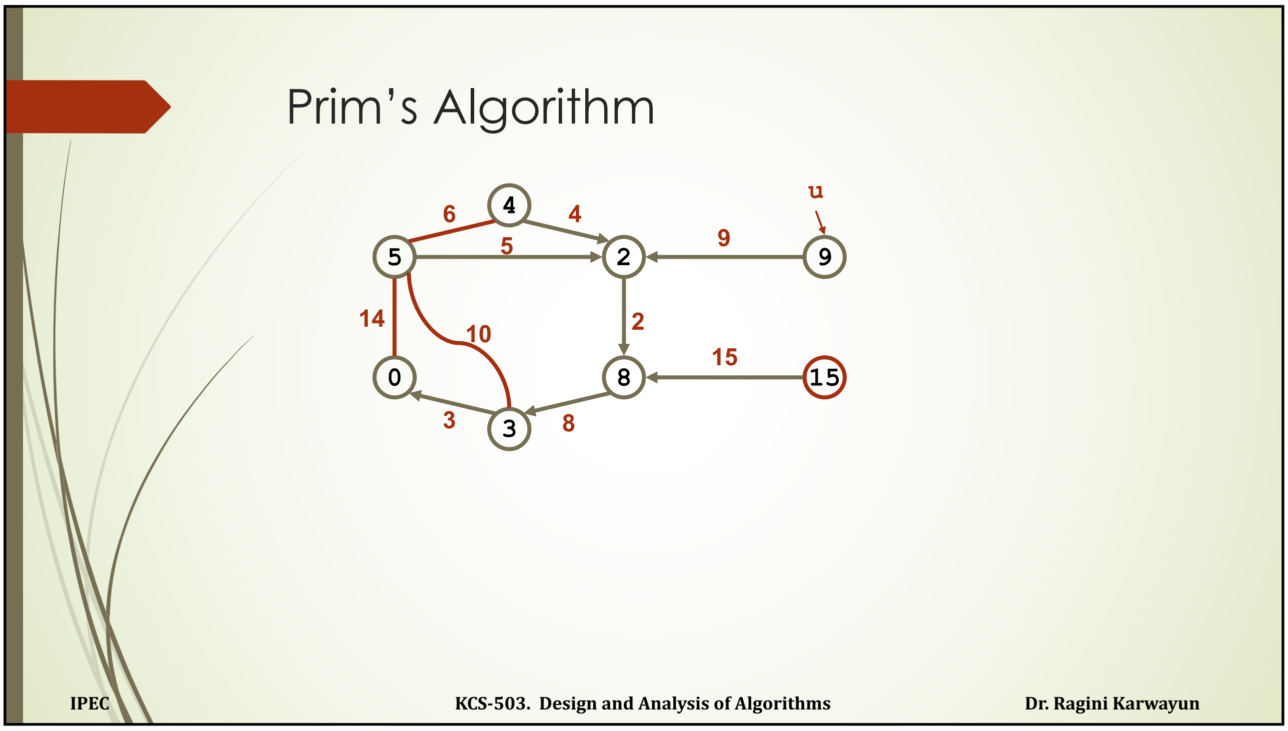


Prim's Algorithm

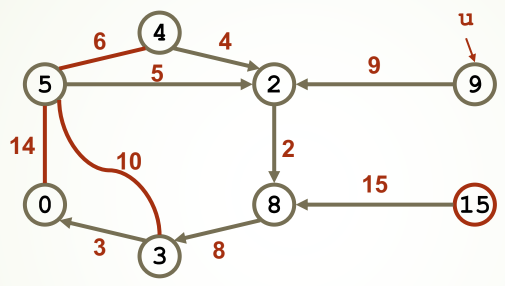


Prim's Algorithm



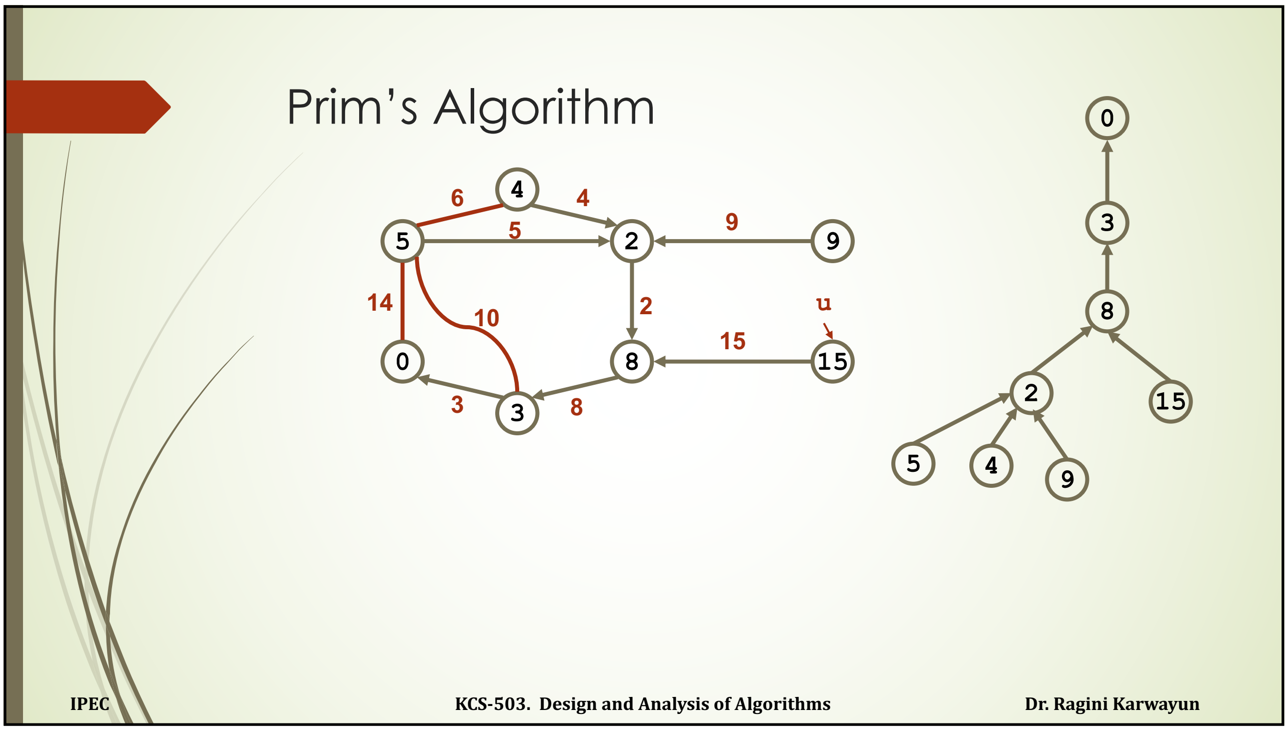


Prim's Algorithm

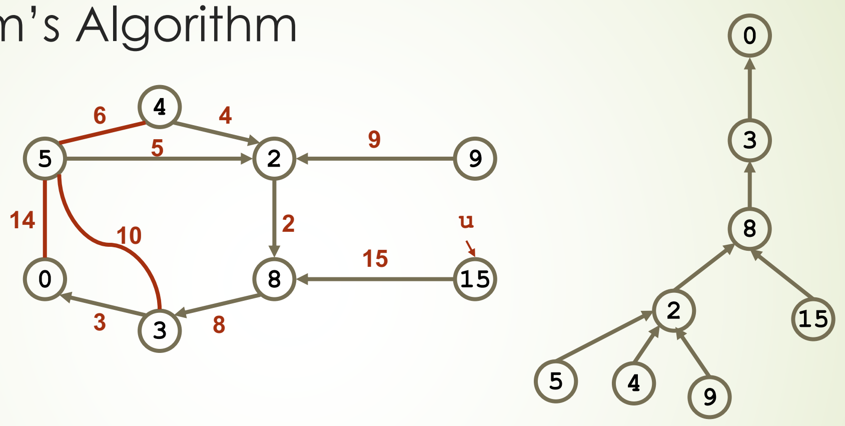


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

39



Prim's Algorithm



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

40

Basics of Kruskal's Algorithm

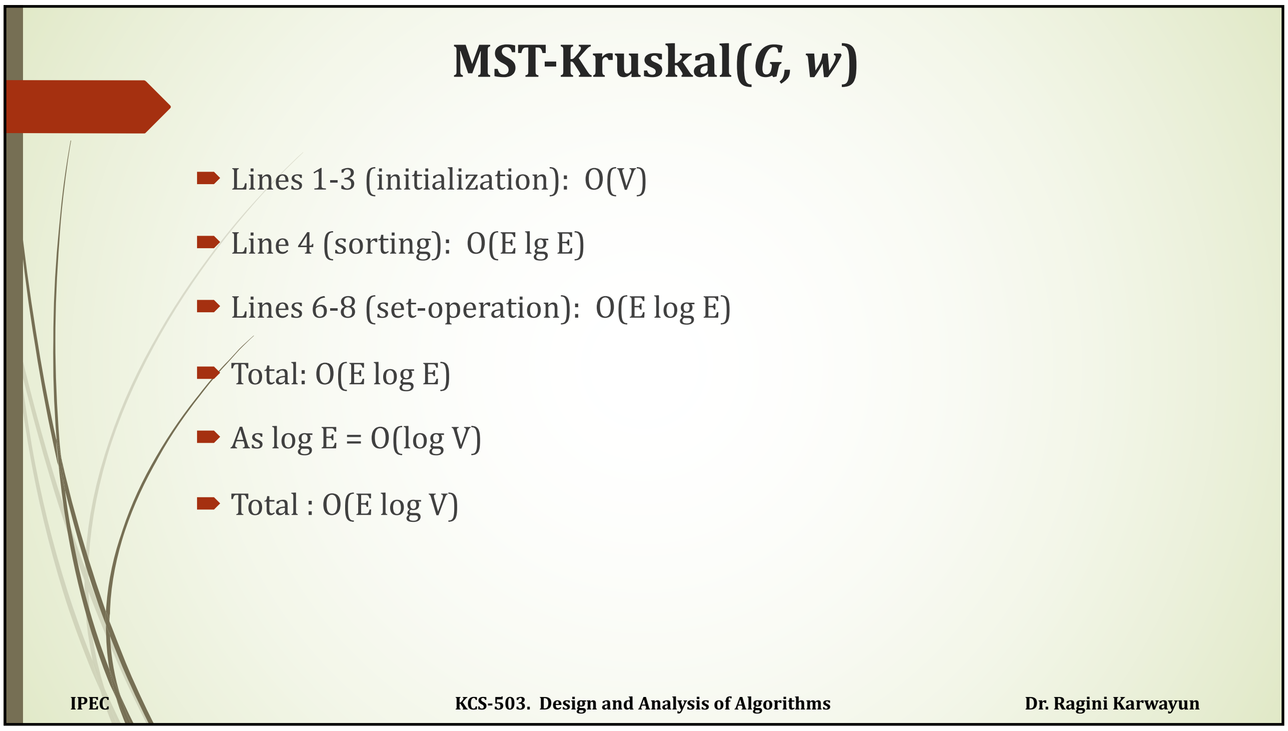
- Attempts to add edges to *A* in increasing order of weight (lightest edge first)
  - If the next edge does not induce a cycle among the current set of edges, then it is added to *A.*
  - If it does, then this edge is passed over, and we consider the next edge in order.
  - As this algorithm runs, the edges of *A* will induce a forest on the vertices and the trees of this forest are merged together until we have a single tree containing all vertices.

41

MST-Kruskal(*G, w*)

```
1. $A \leftarrow \varnothing$                                // initially A is empty
2. for each vertex $v \in V[G]$                    // line 2-3 takes $O(V)$ time
3.     do Make-Set($v$)                            // create set for each vertex
4. sort the edges of $E$ by nondecreasing weight $w$
5. for each edge $(u,v) \in E$, in order by nondecreasing weight
6.     do if Find-Set($u$) $\neq$ Find-Set($v$)         // $u$ & $v$ on different trees
7.            then $A \leftarrow A \cup \{(u,v)\}$
8.                 Union($u,v$)
9. return $A$                                      Total running time is $O(E\ lg\ E)$.
```

42

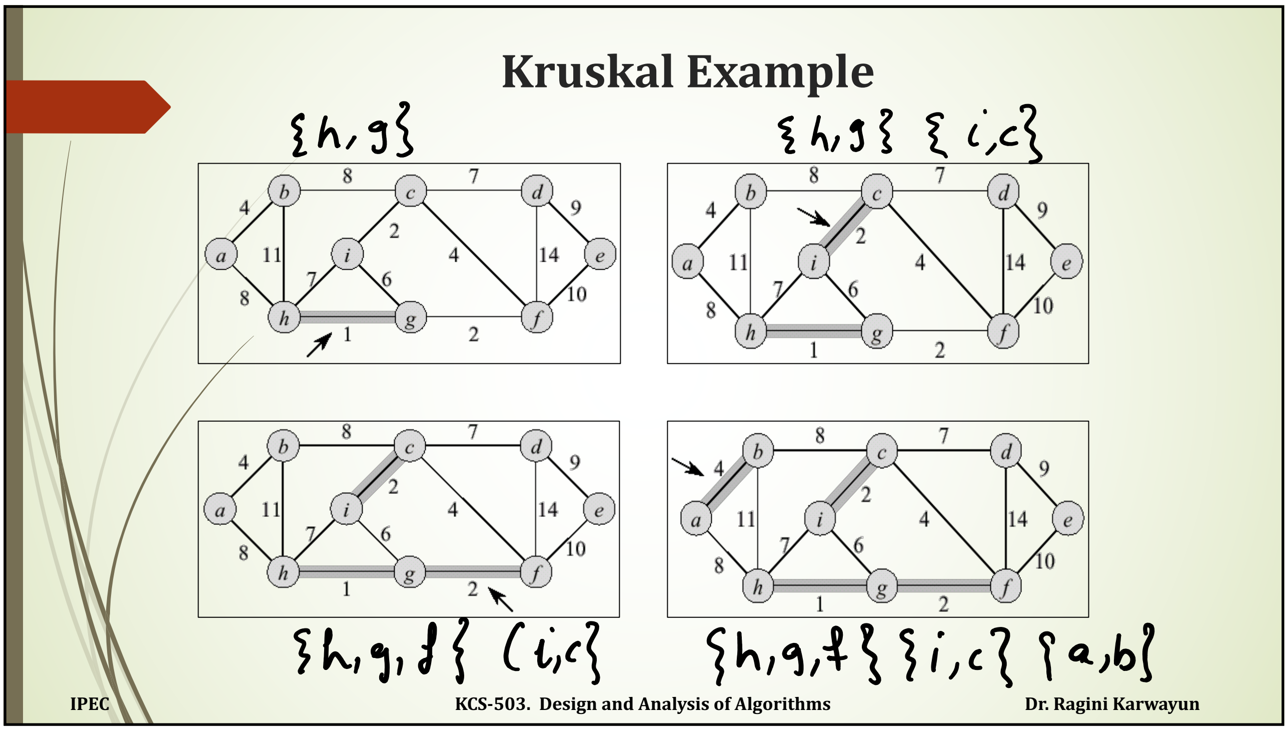


MST-Kruskal(G, w)

- Lines 1-3 (initialization): O(V)
- Line 4 (sorting): O(E lg E)
- Lines 6-8 (set-operation): O(E log E)
- Total: O(E log E)
- As log E = O(log V)
- Total : O(E log V)

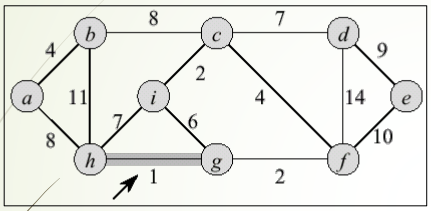
IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

43

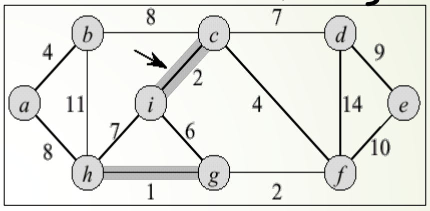


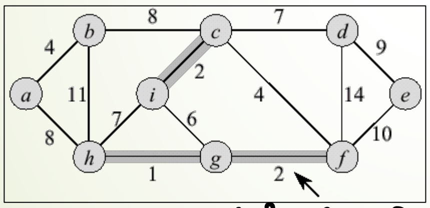
Kruskal Example

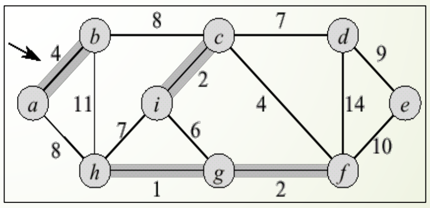
$\{h, g\}$



$\{h, g\}$ $\{i, c\}$







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

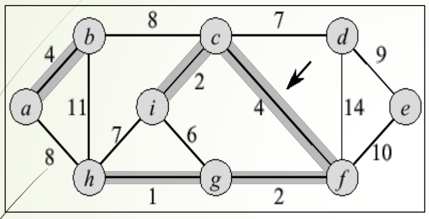
IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

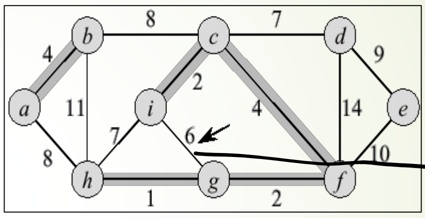
44

Kruskal Example

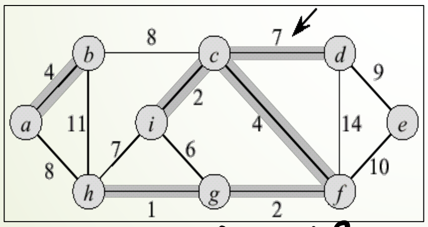
{h,g,f,c,i} {a,b}

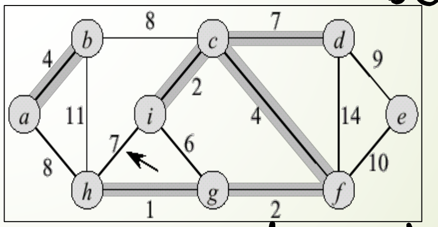
{h,g,f,i,c} {a,b}





reject as i,g belong to same set





{h,g,f,i,c,d} {a,b}

reject as i,h belong to same set

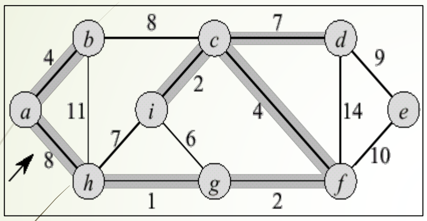
IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

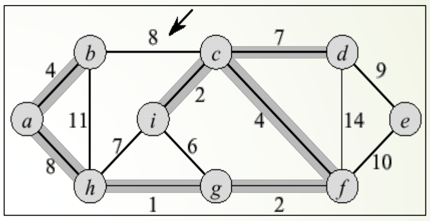
45

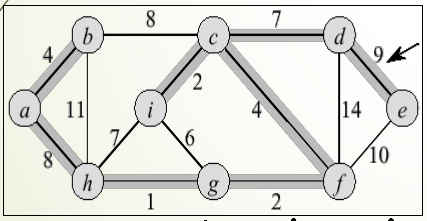
Kruskal Example

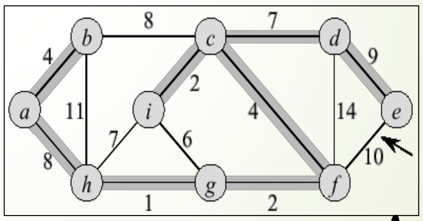
{h,g,f,i,c,d,a,b}

reject







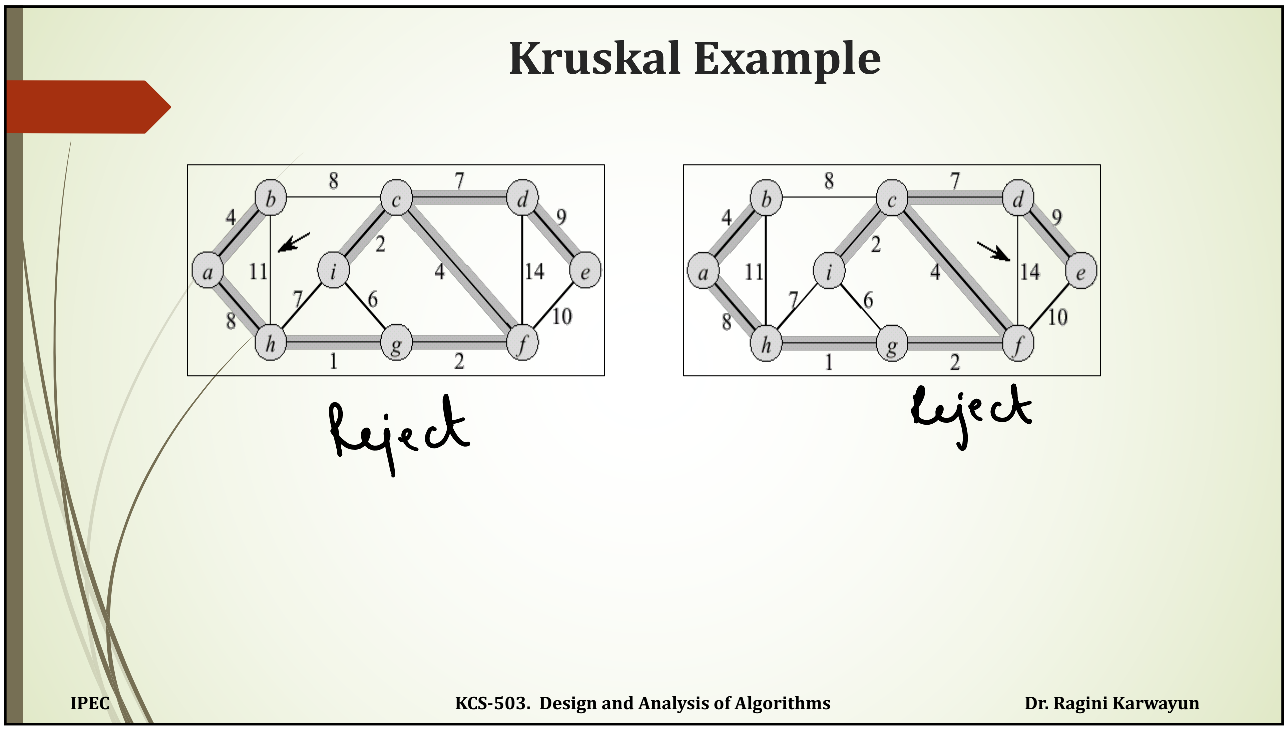


{h,g,f,i,c,d,a,b,e}

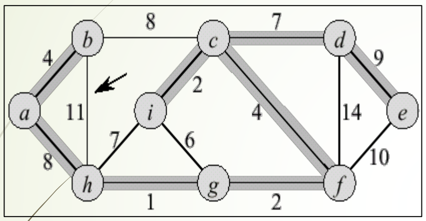
reject.

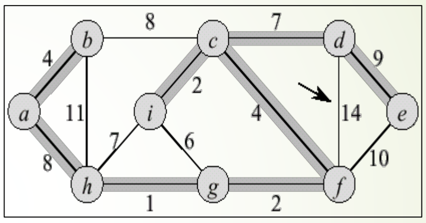
IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

46



Kruskal Example



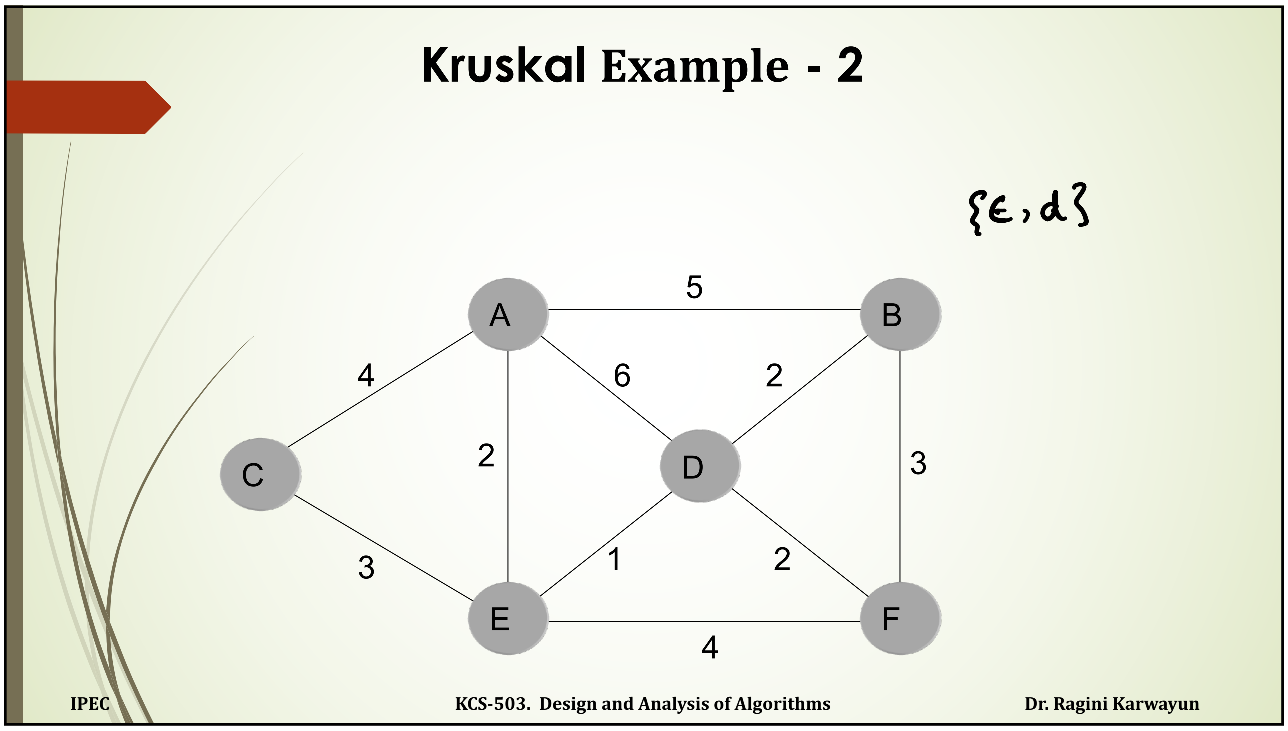


Reject

Reject

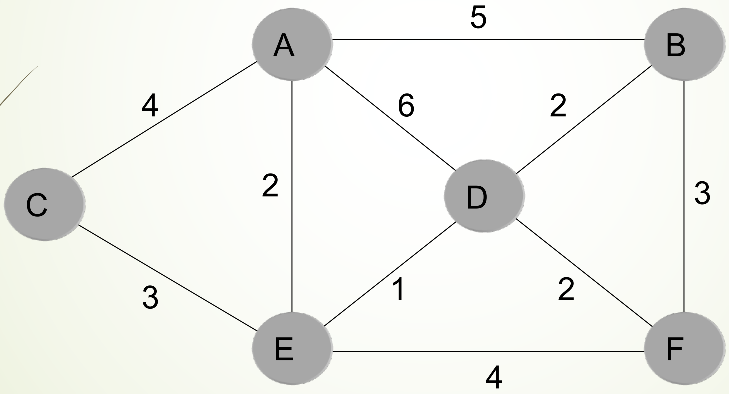
IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

47



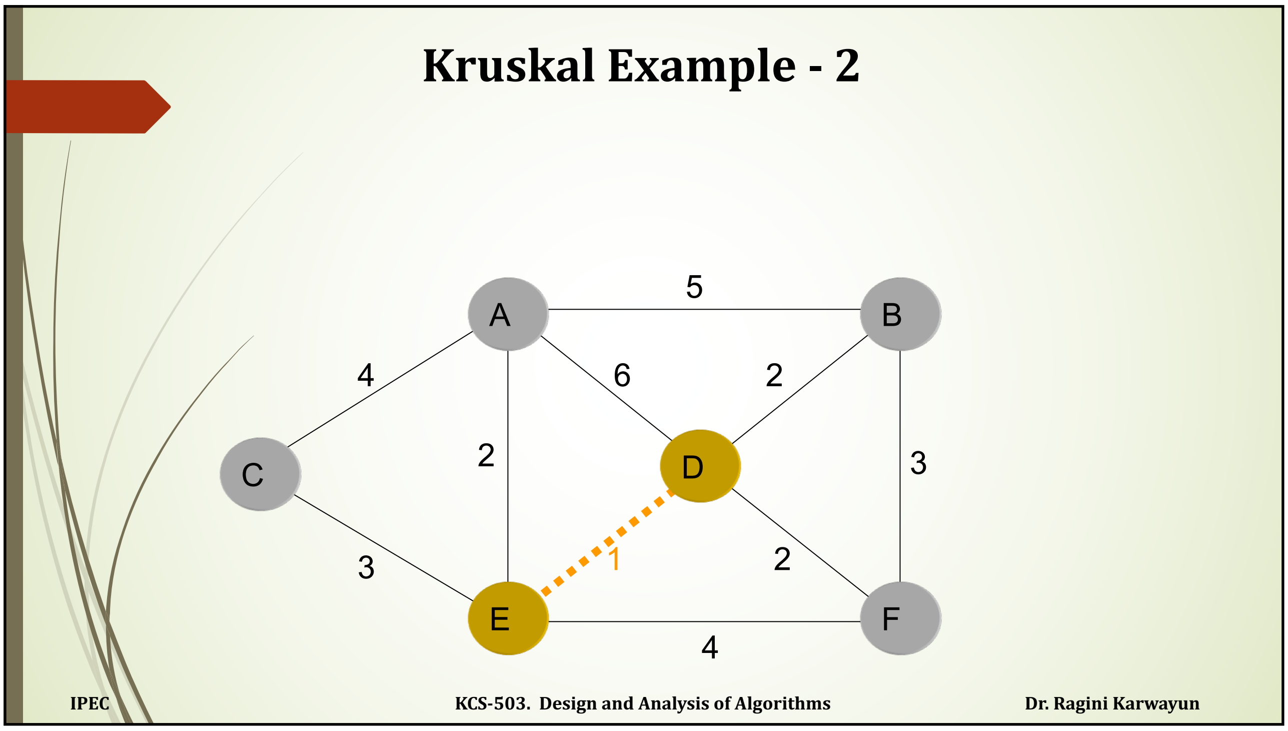
Kruskal Example - 2

{E, d}

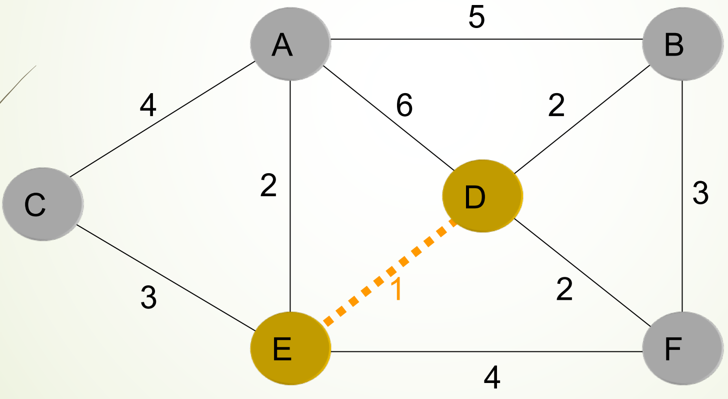


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

48

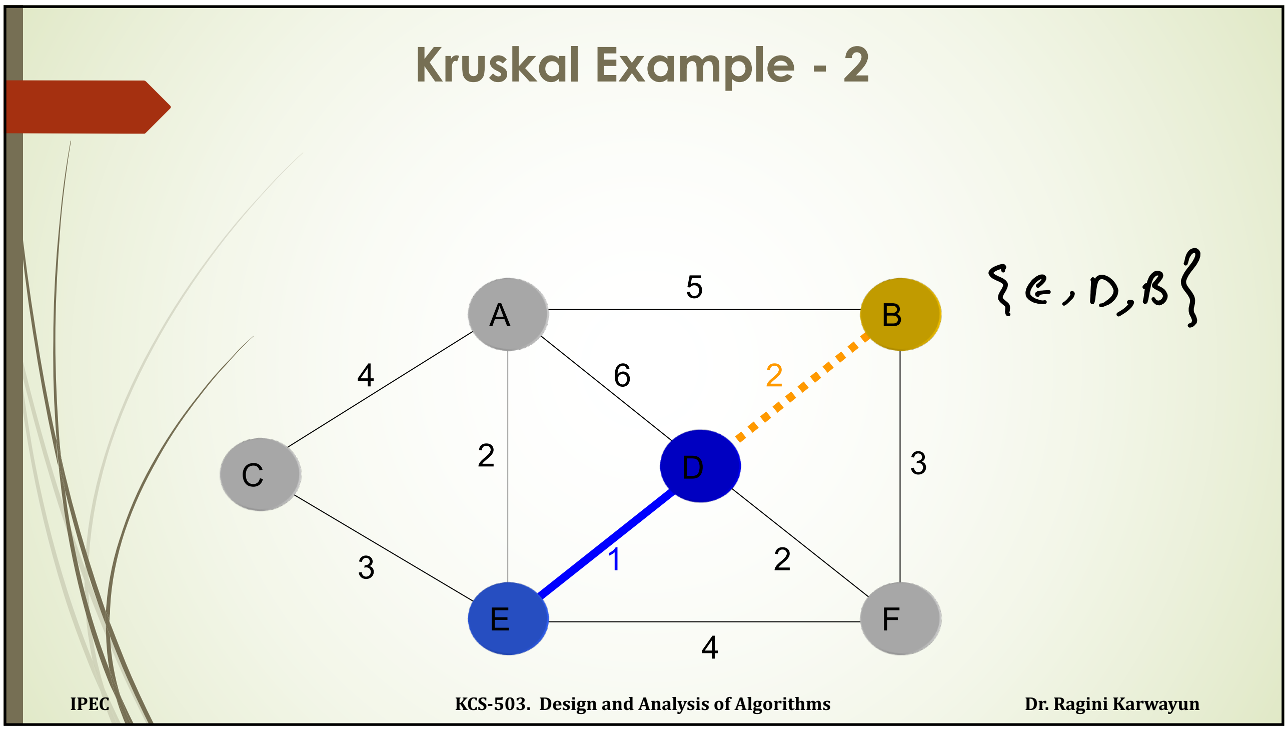


Kruskal Example - 2

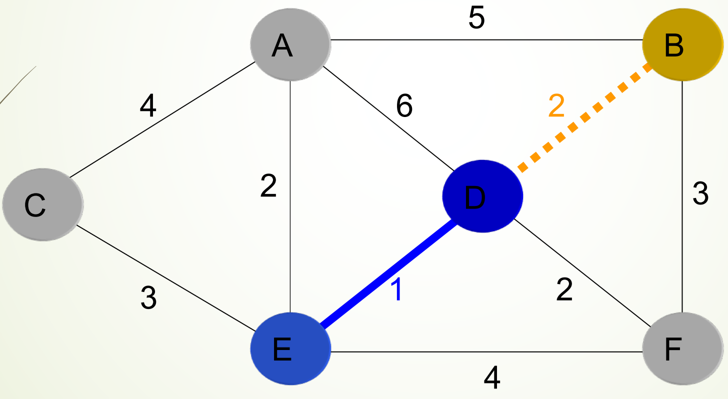


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

49



Kruskal Example - 2

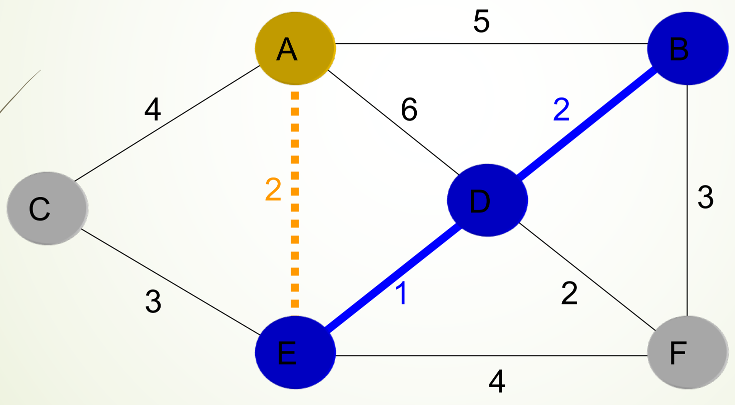


{E, D, B}

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

50

Kruskal Example - 2

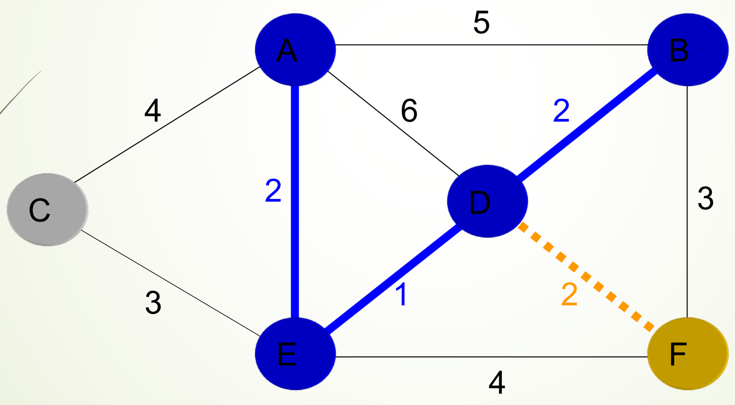


{ E, D, B, A }

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

51

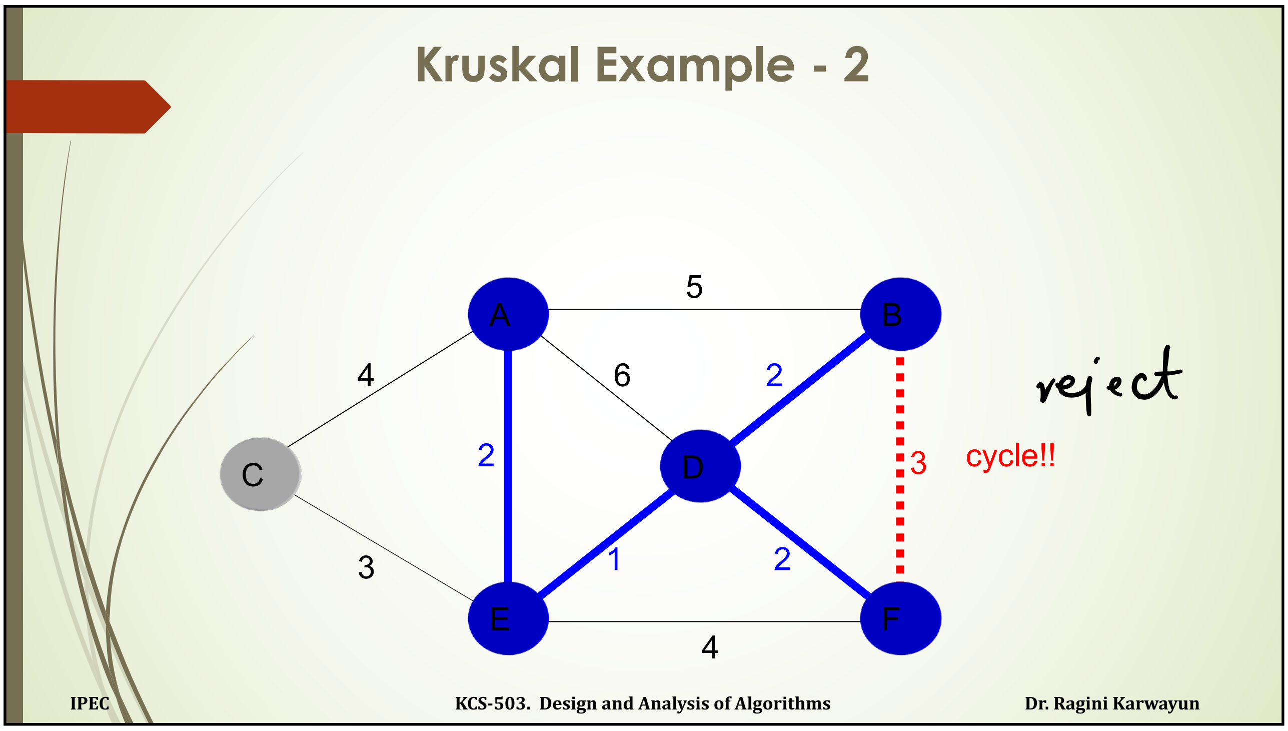
Kruskal Example - 2



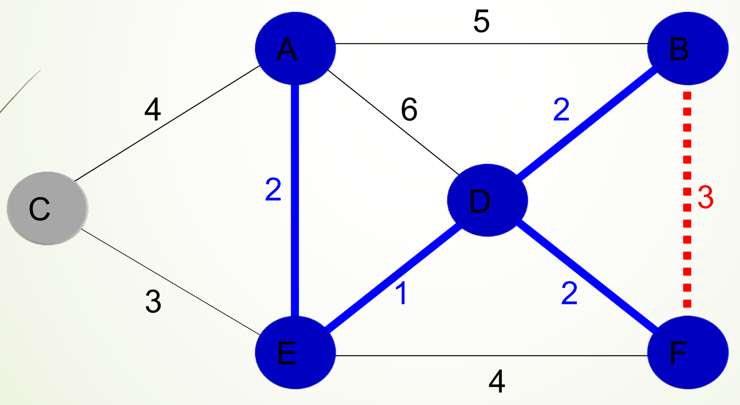
{E D B A F}

IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

52

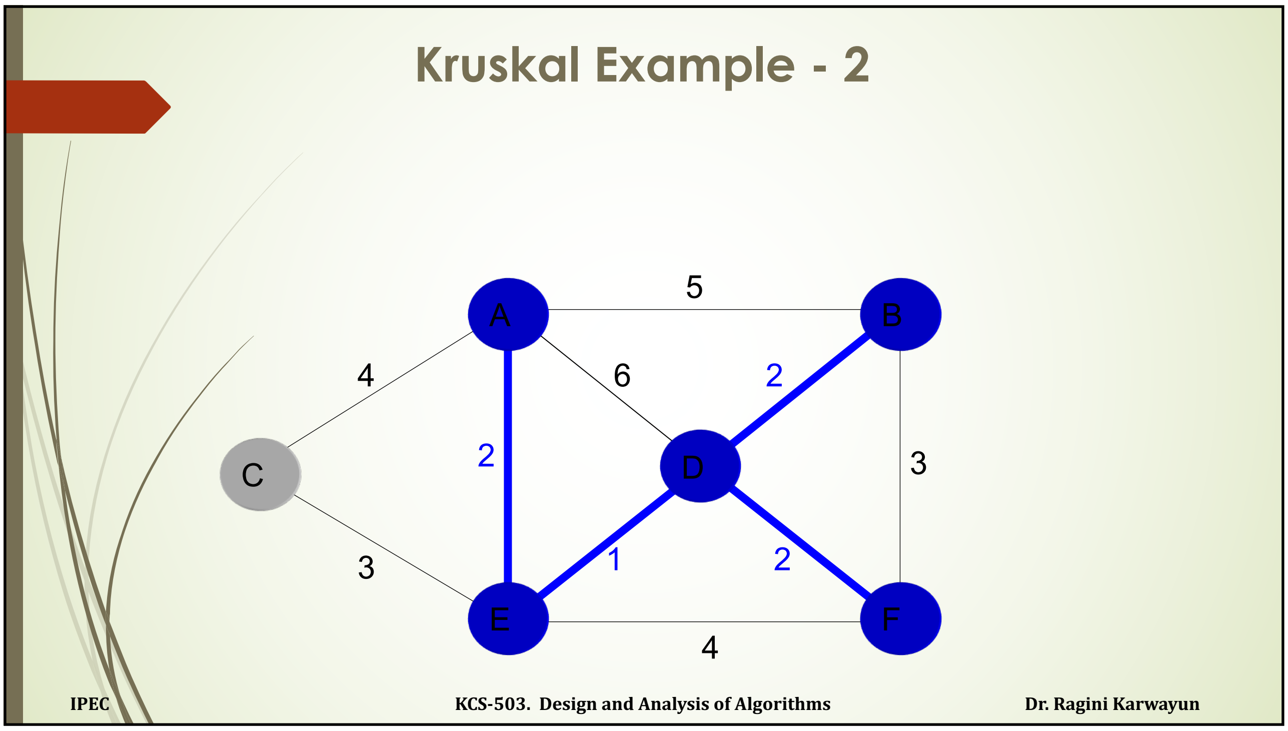


Kruskal Example - 2

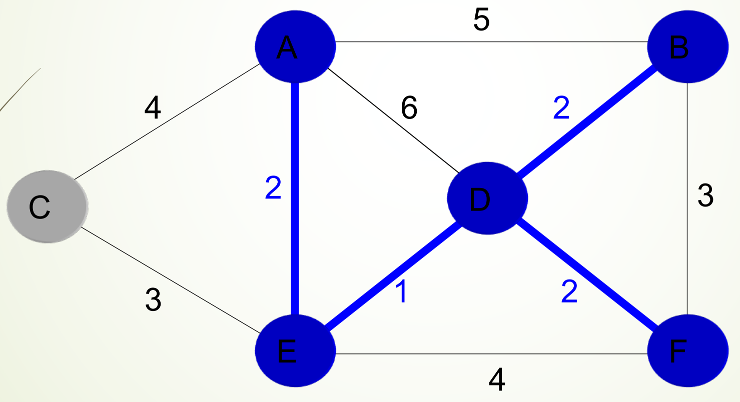


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

53

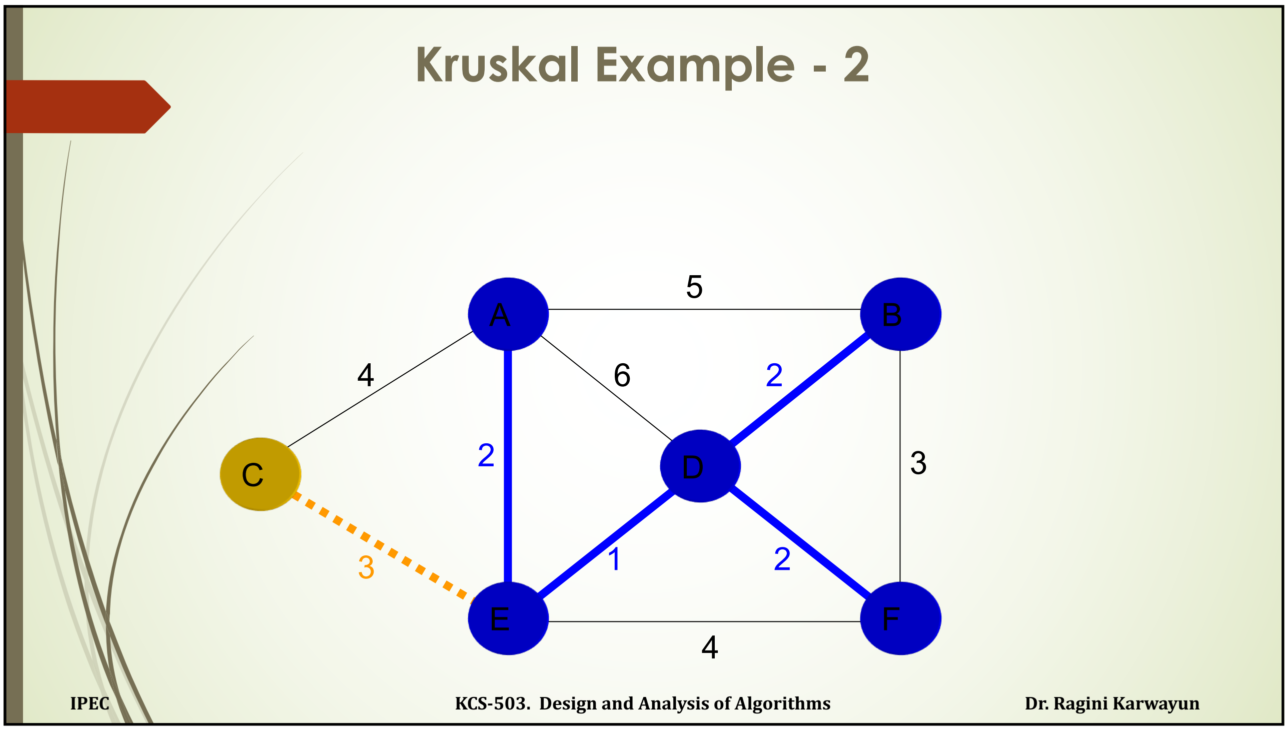


Kruskal Example - 2

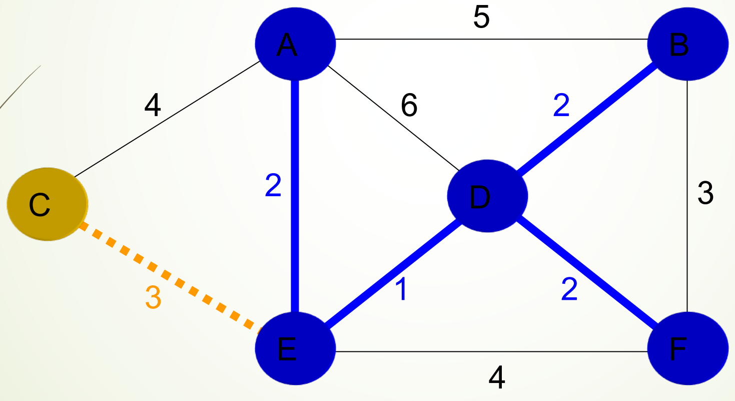


IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

54

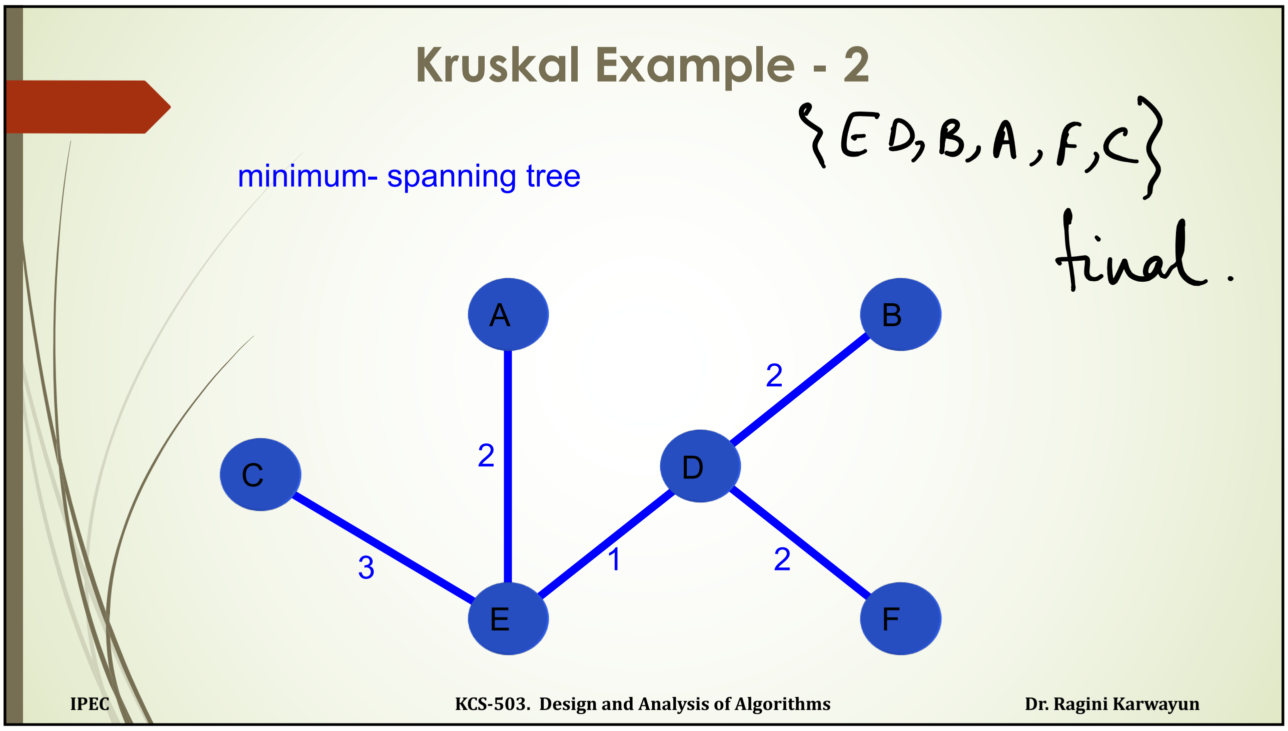


Kruskal Example - 2



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

55

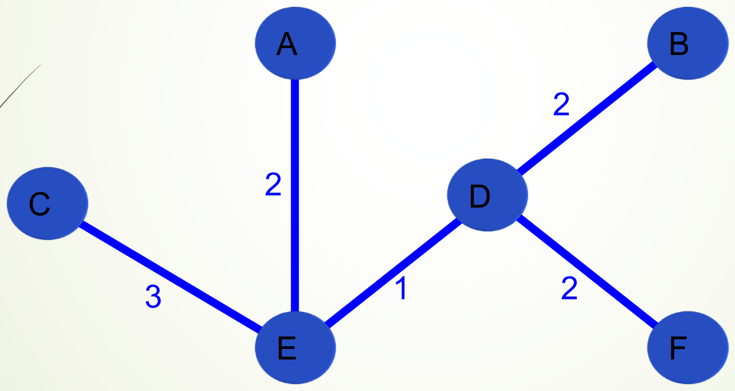


Kruskal Example - 2

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

final.

minimum- spanning tree



IPEC KCS-503. Design and Analysis of Algorithms Dr. Ragini Karwayun

56