Design and Analysis of Algorithms
Unit -4

Bharathi R
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Unit 4: Greedy Technique

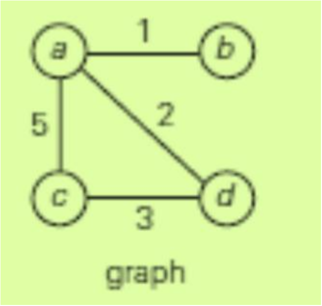
Prim's algorithm

Bharathi R

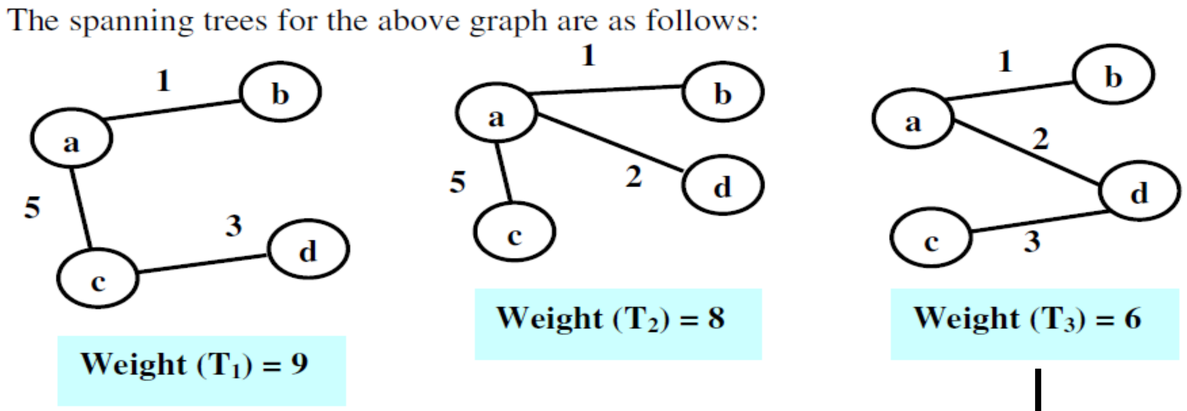
Department of Computer Science & Engineering

Minimum Spanning Tree : DEFINITIONs

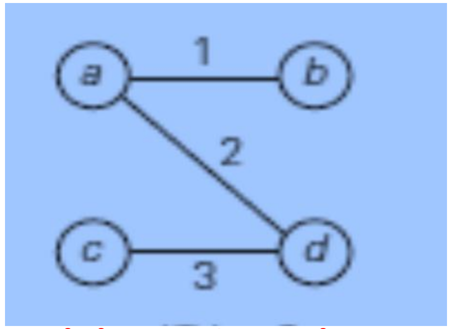
A *spanning tree* of an undirected connected graph is its connected acyclic subgraph (i.e., a tree) that contains all the vertices of the graph.



Example



Minimum Spanning Tree (MST) of a weighted, connected graph G is a spanning tree of G with minimum total weight.

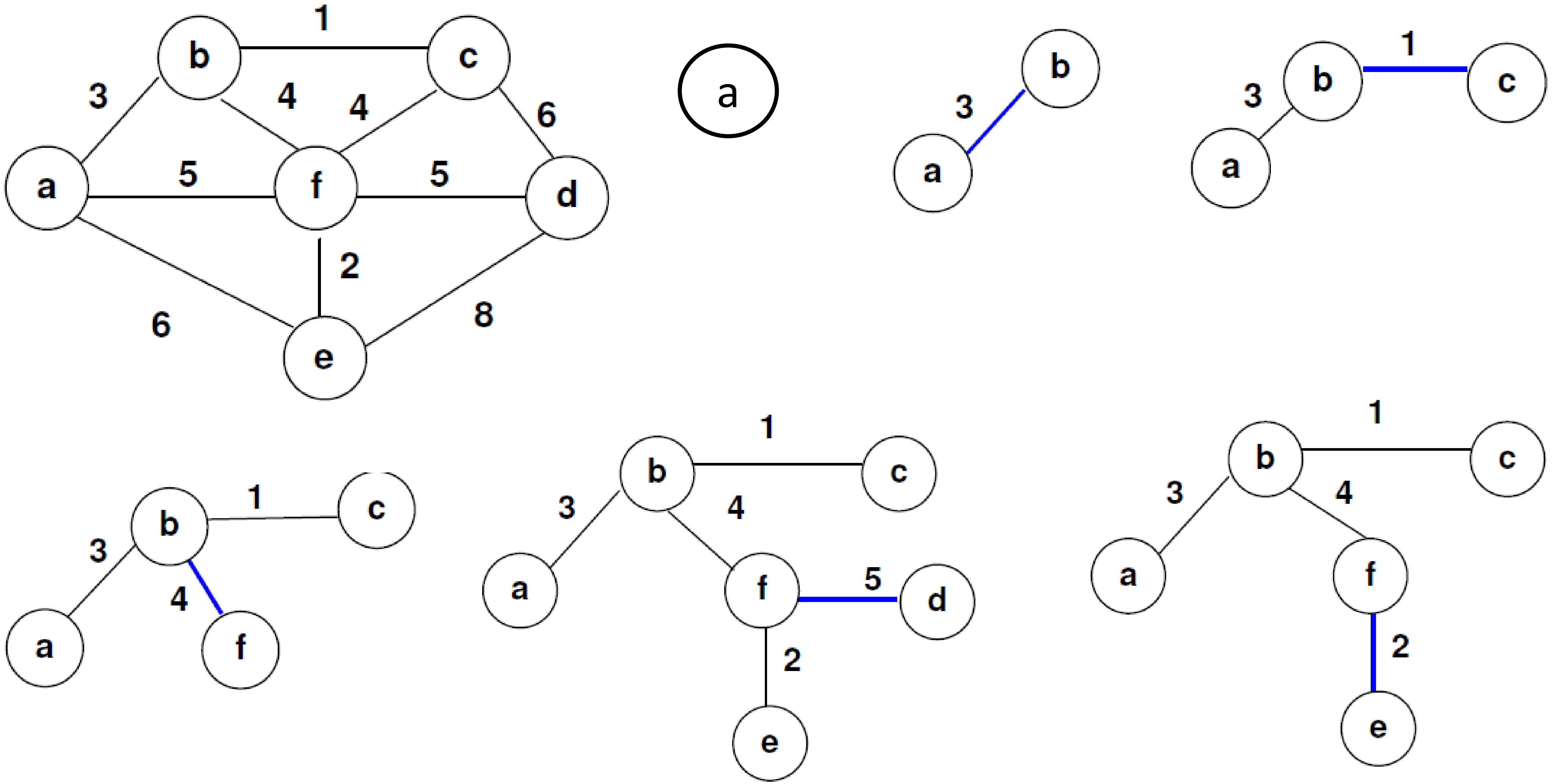


Minimum Spanning Tree

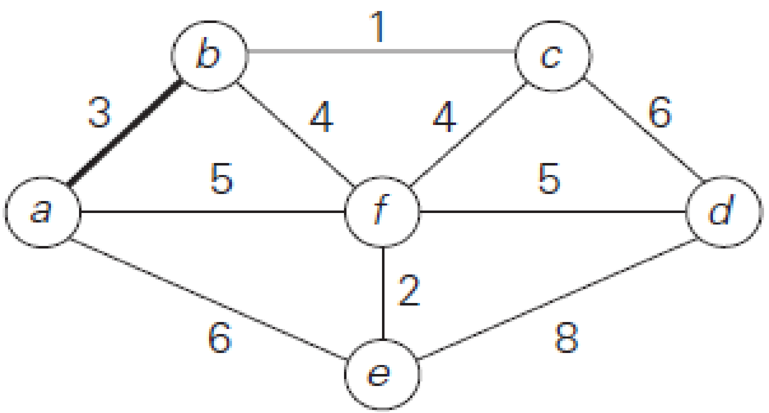
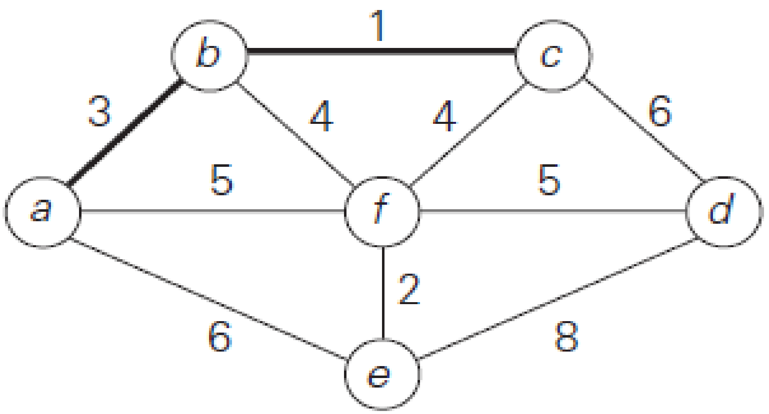
Prim's Algorithm : Greedy Approach

- Start with a tree, $\mathbf{T_1}$, consisting of one vertex (V).
- Adjacent vertices of the vertex in $\mathbf{T_1}$ are "fringe" vertices of $\mathbf{T_1}$.
- For i = 1 to |V|-1 do
  - Construct $\mathbf{T_i}$ from $\mathbf{T_{i-1}}$ by adding the fringe vertex with the minimum weight edge from the set. The vertex is removed from the set of fringe vertices.
  - Add the adjacent vertices of the vertex to the set of fringe vertices which are not in $\mathbf{T_i}$.
  - Remove vertices from the set of fringe vertices where the new vertex is one of the terminal vertex of the edge.
- Return $\mathbf{T_n}$ which is a minimum spanning tree.

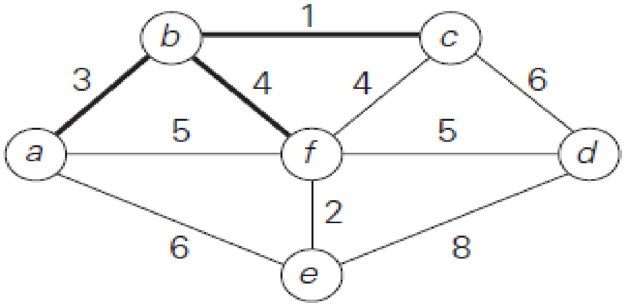
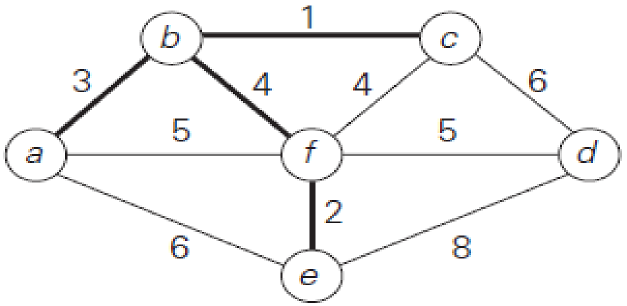
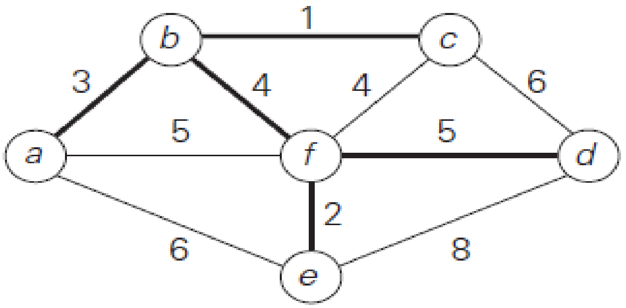
Prim's Algorithm



Prim's Algorithm

Tree vertices	Remaining vertices	Illustration
a(−, −)	**b(a, 3)** c(−, ∞) d(−, ∞) e(a, 6) f(a, 5)	
b(a, 3)	**c(b, 1)** d(−, ∞) e(a, 6) f(b, 4)	

Prim’s Algorithm

Tree vertices	Remaining vertices	Illustration
c(b, 1)	d(c, 6) e(a, 6) **f(b, 4)**	
f(b, 4)	d(f, 5) **e(f, 2)**	
e(f, 2)	**d(f, 5)**	
d(f, 5)		

ALGORITHM *Prim(G)*

//Prim's algorithm for constructing a minimum spanning tree
//Input: A weighted connected graph $G = \langle V, E \rangle$
//Output: E_T, the set of edges composing a minimum spanning tree of G
$V_T \leftarrow \{v_0\}$ //the set of tree vertices can be initialized with any vertex
$E_T \leftarrow \varnothing$
for $i \leftarrow 1$ **to** $|V| - 1$ **do**
    find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u)
    such that v is in V_T and u is in $V - V_T$
    $V_T \leftarrow V_T \cup \{u^*\}$
    $E_T \leftarrow E_T \cup \{e^*\}$
return E_T

find a minimum-weight edge $e^* = (v^*, u^*)$ among all the edges (v, u) such that v is in V_T and u is in $V - V_T$

After we have identified a vertex u^* to be added to the tree, we need to perform two operations:

Move u^* from the set $V - V_T$ to the set of tree vertices V_T.

For each remaining vertex u in $V - V_T$ that is connected to u^* by a shorter edge than the u's current distance label, update its labels by u^* and the weight of the edge between u^* and u, respectively.

How efficient is Prim's Algorithm

- The answer depends on the data structures chosen for the graph itself and for the priority queue of the set $V - V_T$ whose vertex priorities are the distances to the nearest tree vertices.

- If a graph is represented by its weight matrix and the priority queue is implemented as an unordered array, the algorithm's running time will be in $\Theta(|V|^2)$

- If a graph is represented by its adjacency lists and the priority queue is implemented as a min-heap, the running time of the algorithm is in $O(|E| \log |V|)$.

Summary

Prim's algorithm is very similar to Kruskal's: whereas Kruskal's "grows" a forest of trees, Prim's algorithm grows a single tree until it becomes the minimum spanning tree.

Both algorithms use the greedy approach - they add the cheapest edge that will not cause a cycle. But rather than choosing the cheapest edge that will connect *any* pair of trees together, Prim's algorithm only adds edges that join nodes to the existing tree.

(In this respect, Prim's algorithm is very similar to Dijkstra's algorithm for finding shortest paths.)

Text Books

Chapter 9 ,Introduction to The Design and Analysis of Algorithms by Anany Levitin

THANK YOU

Bharathi R

Department of Computer Science & Engineering

rbharathi@pes.edu