Design and Analysis of Algorithms
Unit -4

Bharathi R
Department of Computer Science & Engineering

DESIGN AND ANALYSIS OF ALGORITHMS

Unit 4: Greedy Technique

Bharathi R

Department of Computer Science & Engineering

Minimum Spanning Tree : Kruskal's Algorithm

This algorithm was described by **Joseph Kruskal** in 1956.

Prim's algorithm and Kruskal's algorithm solve the same problem (**Minimum Spanning Tree**) by applying the **greedy approach** in two different ways, and both of them always yield an optimal solution.

Minimum Spanning Tree : Kruskal's Algorithm

Kruskal's algorithm initially places all the nodes of the original graph isolated from each other, to form a forest of single node trees, and then gradually merges these trees, combining at each iteration any two of all the trees with some edge of the original graph.

Before the execution of the algorithm, all edges are sorted by weight (in non-decreasing order).

Then begins the process of unification: pick all edges from the first to the last (in sorted order), and if the ends of the currently picked edge belong to different subtrees, these subtrees are combined, and the edge is added to the answer.

After iterating through all the edges, all the vertices will belong to the same sub-tree, and we will get the answer.

Design and Analysis of Algorithms

Kruskal's Algorithm

Let G = {V,E} be weighted connected graph

The vertices are numbered 0 through $n - 1$.

V = {0,1,….n-1} a nd E =[e_1,e_2,….e_n}

Sort the edges in non-decreasing order of their weights.

•Initially, trees of the forest are the vertices (no edges).

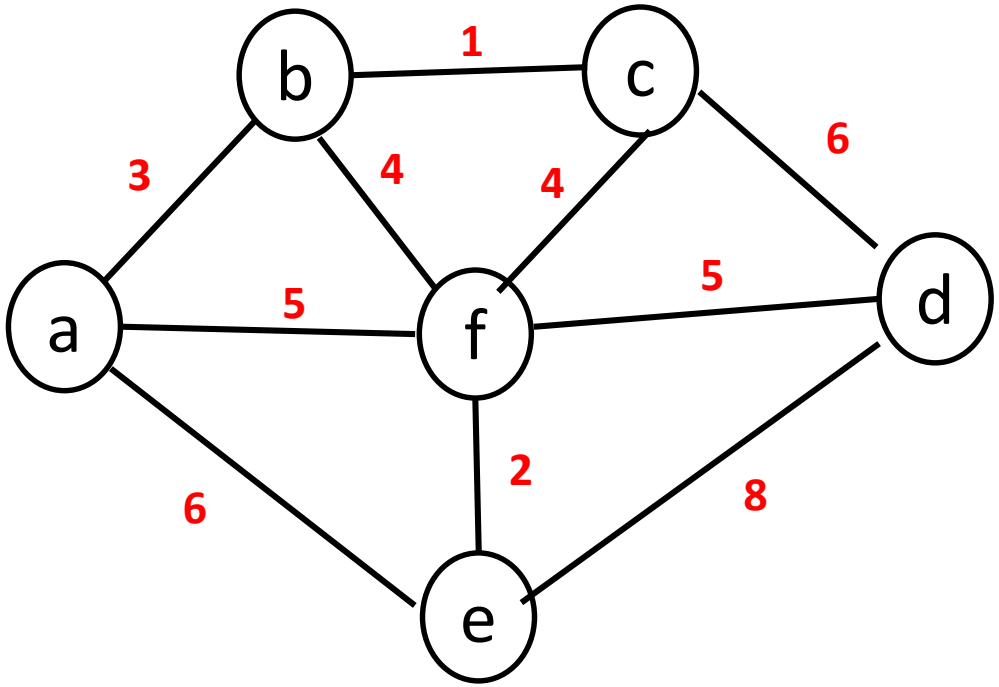
•Start with an empty minimum spanning tree and mark each edge as unvisited,

•While there are edges not yet visited or while the minimum spanning tree does not have $n - 1$ edges:

- Find the edge with minimum weight that has not yet been visited.
- Mark that edge as visited.
- If adding that edge does not create a cycle (that is, the two vertices are not already connected), add that edge to the minimum spanning tree.

Kruskal's Algorithm

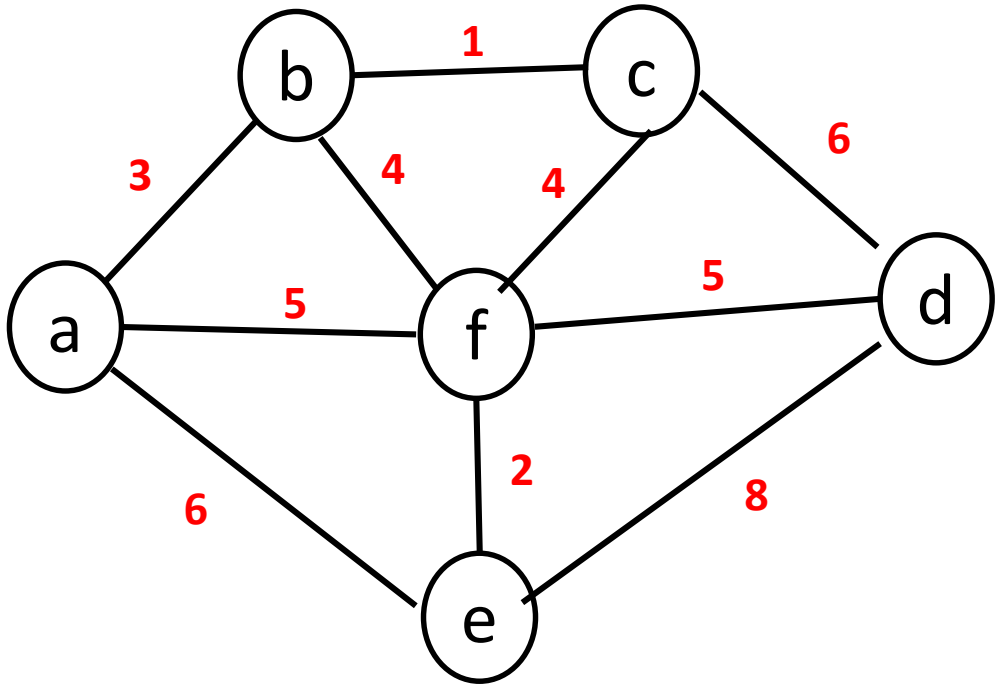
Given : Graph <V,E>



Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



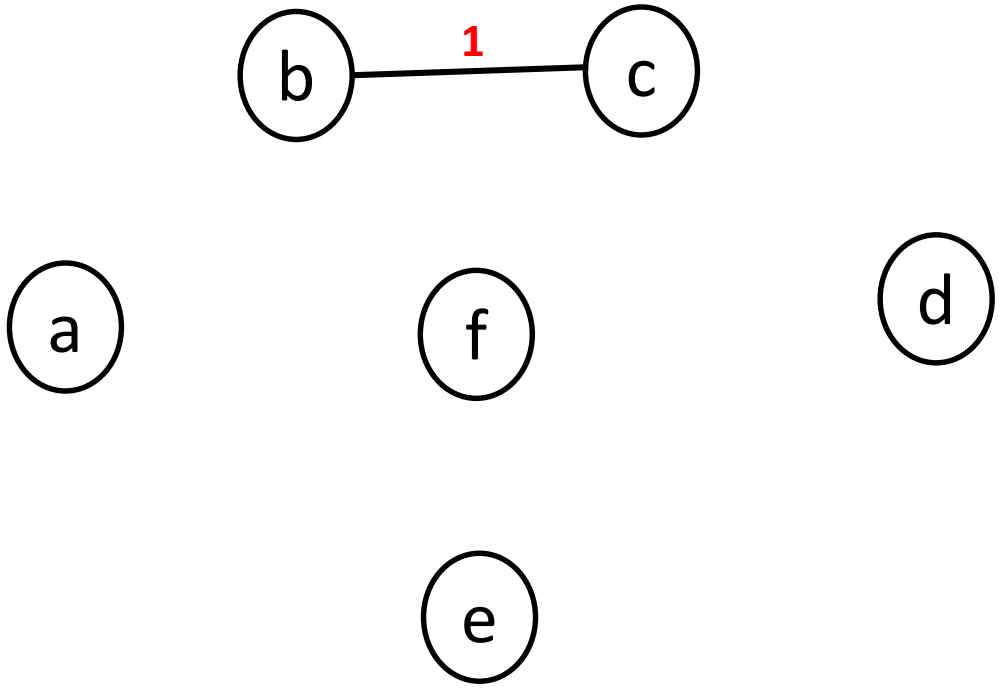
Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



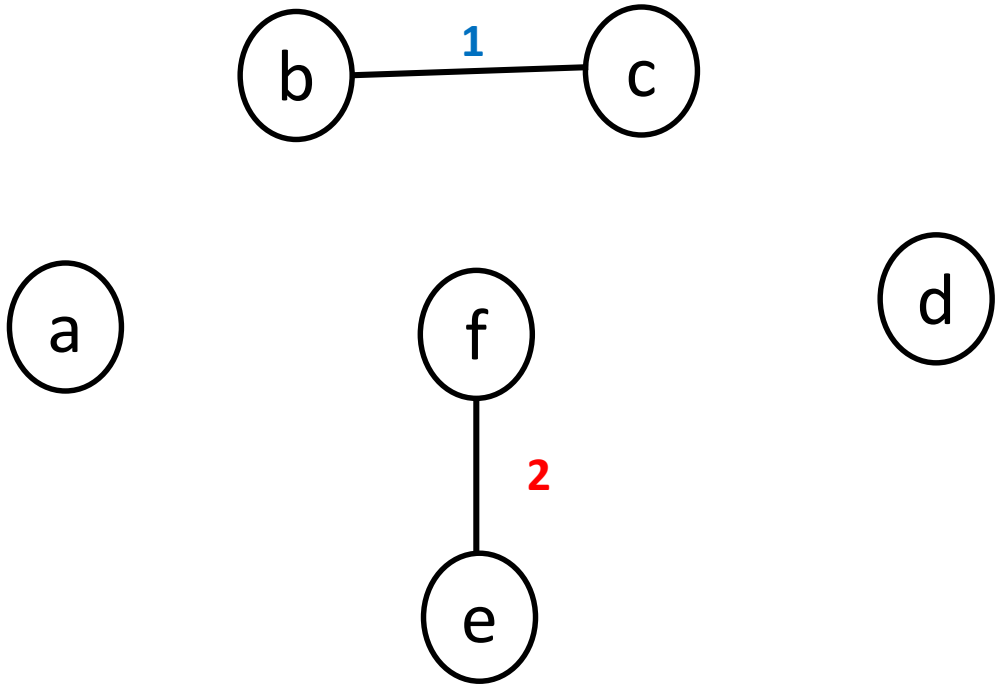
Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	**1**
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



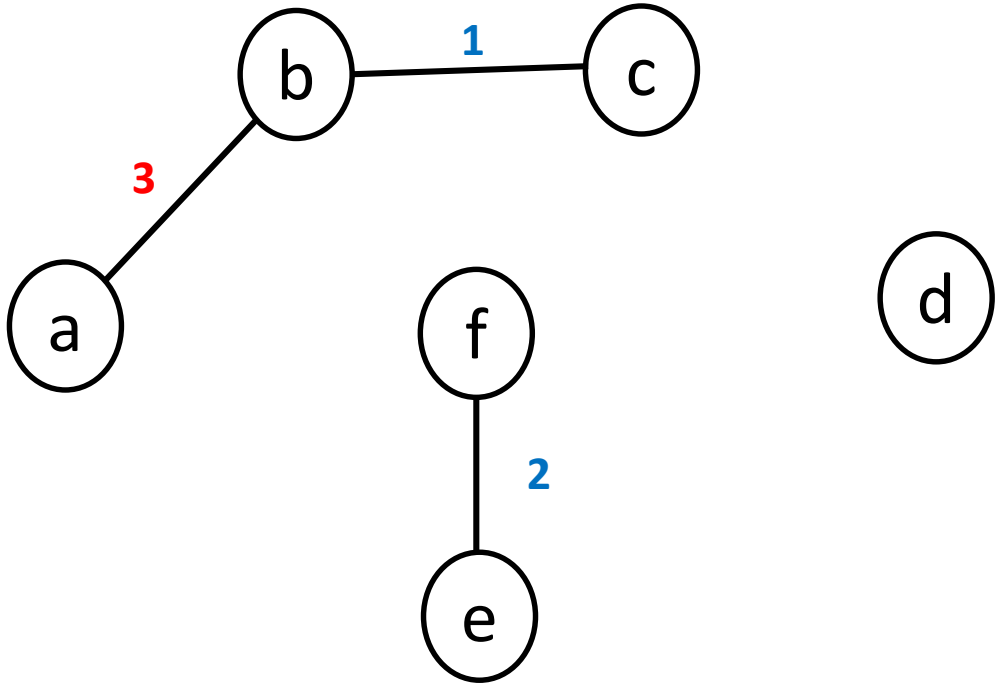
Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



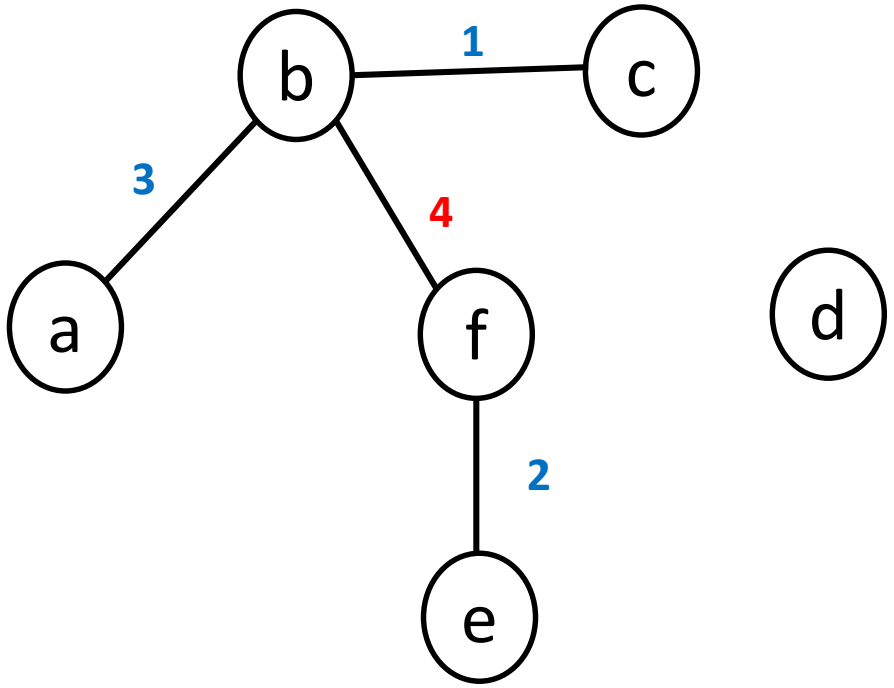
Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



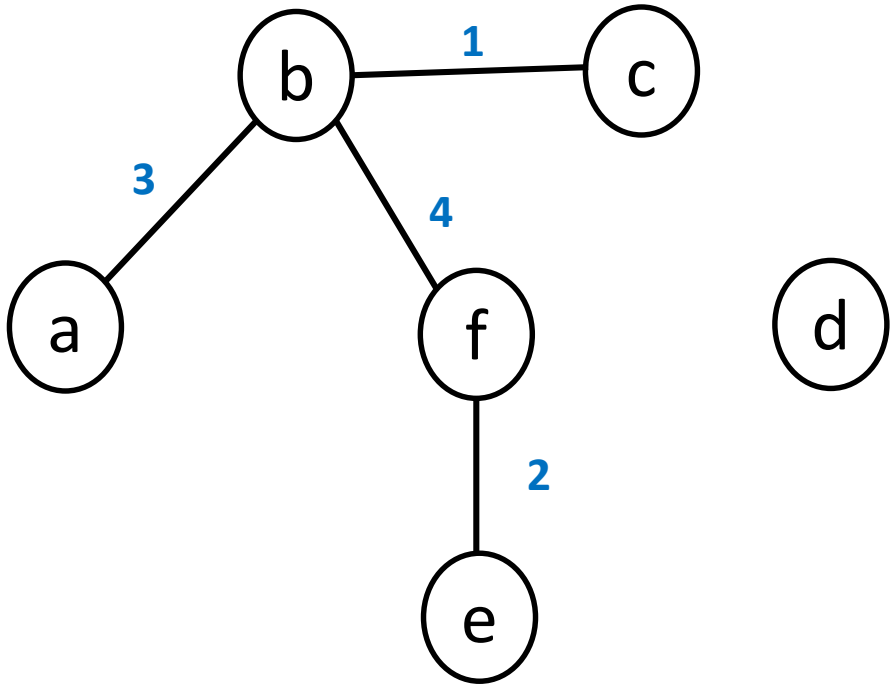
Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



Sorted list of Edges in non decreasing Order

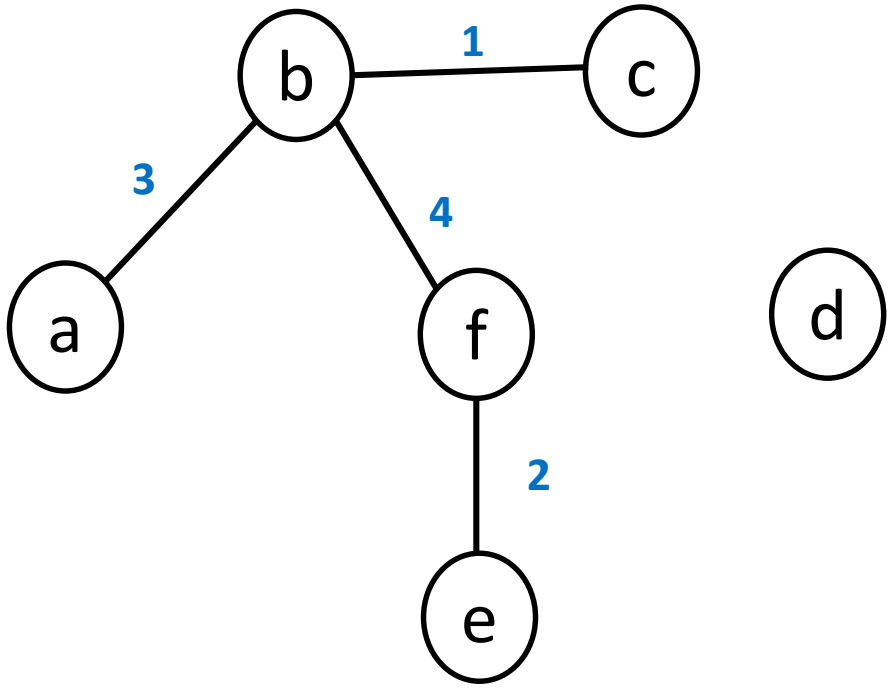
Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4 x
af	5
df	5
ae	6
cd	6
de	8

Cyclic

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

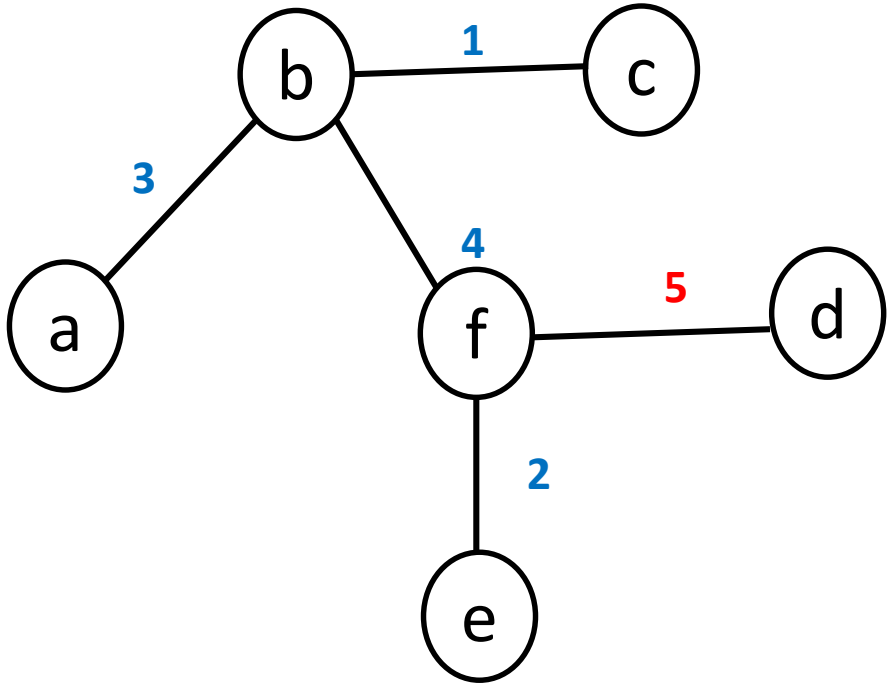
Cyclic

X

Design and Analysis of Algorithms

Kruskal's Algorithm

Given : Graph<V,E>



Sorted list of edges in non decreasing order

Edge(u,v)	Weight
bc	1
ef	2
ab	3
bf	4
cf	4
af	5
df	5
ae	6
cd	6
de	8

Acyclic

Before formalizing the above idea, lets quickly review the disjoint-set data structure.

1. *makeset*(***v***): Create a new set whose only member is pointed to by **v**. Note that for this operation v must already be in a set.
2. *find*(***v***): Returns a pointer to the set containing *v*.
3. *union*(**u,v**): Unites the dynamic sets that contain *u* and *v* into a new set that is union of these two sets.

For example, let ***S* = {1, 2, 3, 4, 5, 6}.**
Then ***makeset(i)*** creates the set ***{i}*** and applying this operation six times initializes the structure to the collection of six singleton sets:

{1}, {2}, {3}, {4}, {5}, {6}.

Performing ***union(1, 4)*** and ***union(5, 2)*** yields

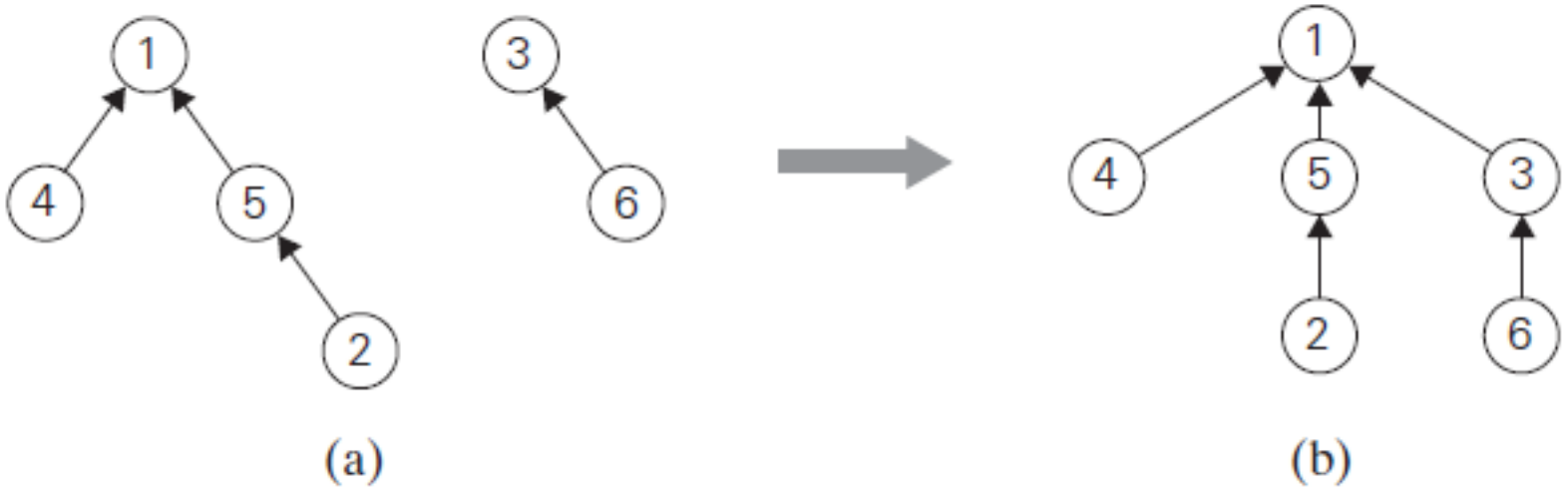
{1, 4}, {5, 2}, {3}, {6},

and, if followed by ***union(4, 5)*** and then by ***union(3, 6)***, we end up with the disjoint subsets

{1, 4, 5, 2}, {3, 6}.

(a) Forest representation of subsets {1, 4, 5, 2} and {3, 6}

(b) Result of *union*(5, 6).

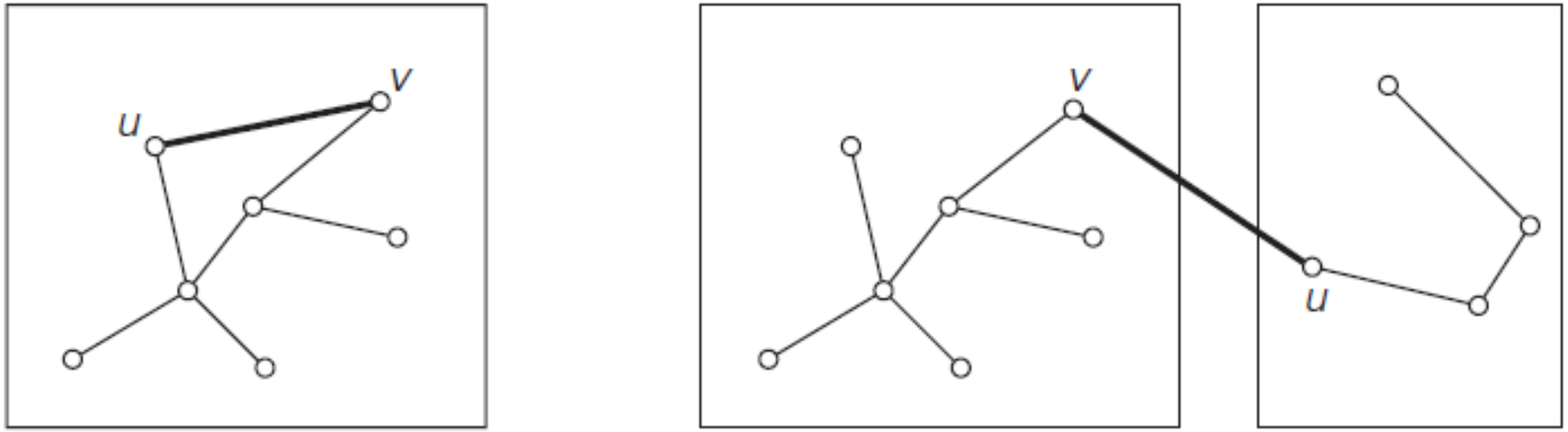


Kruskal's Algorithm

ALGORITHM *Kruskal(G)*

//Kruskal'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
sort E in nondecreasing order of the edge weights $w(e_{i_1}) \leq \cdots \leq w(e_{i_{|E|}})$
$E_T \leftarrow \varnothing$; $ecounter \leftarrow 0$ //initialize the set of tree edges and its size
$k \leftarrow 0$ //initialize the number of processed edges
while $ecounter < |V| - 1$ **do**
    $k \leftarrow k + 1$
    if $E_T \cup \{e_{i_k}\}$ is acyclic
        $E_T \leftarrow E_T \cup \{e_{i_k}\}$; $ecounter \leftarrow ecounter + 1$
return E_T

Kruskal's Algorithm



With an efficient Sorting algorithm and an Union-Find algorithm.

Efficiency: $\Theta(|E| \log |E|)$

Design and Analysis of Algorithms

Text Book

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