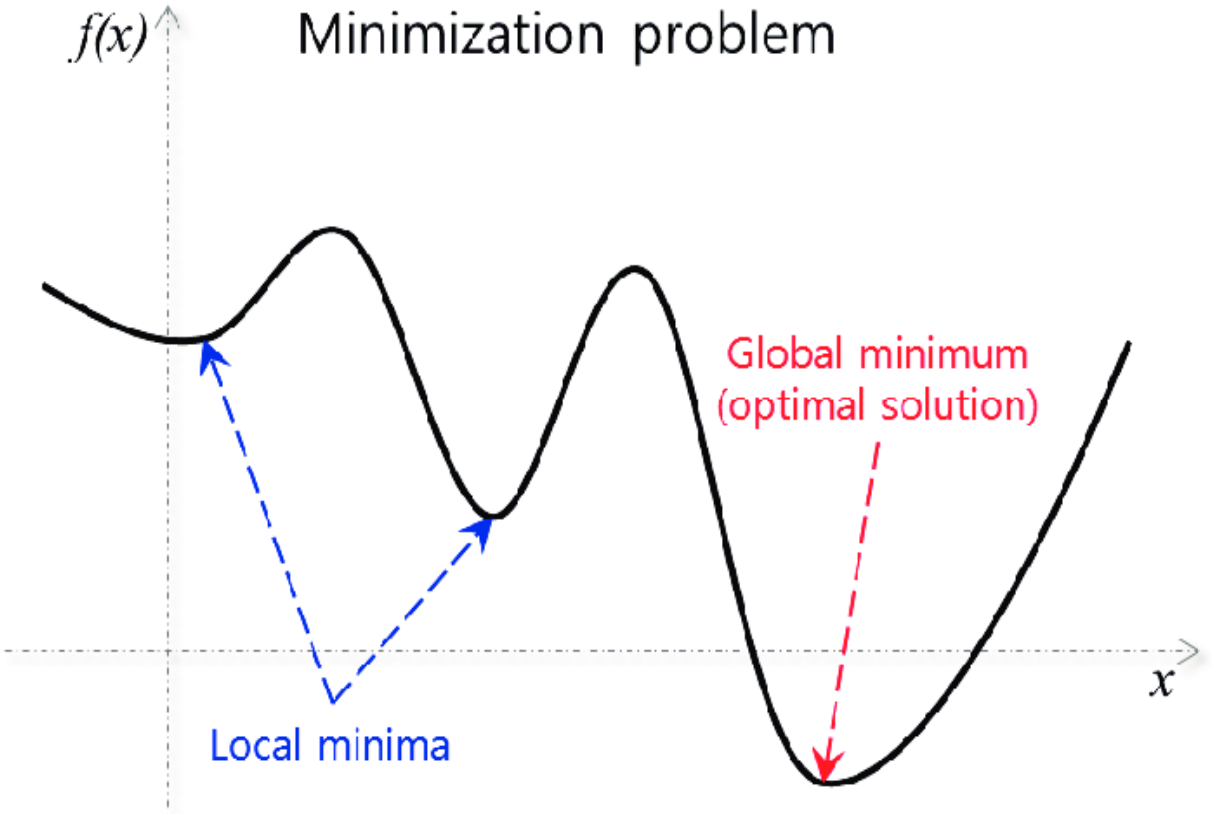
Design and Analysis of Algorithms

Dina El-Manakhly, Ph. D.

dina_almnakhly@science.suez.edu.eg

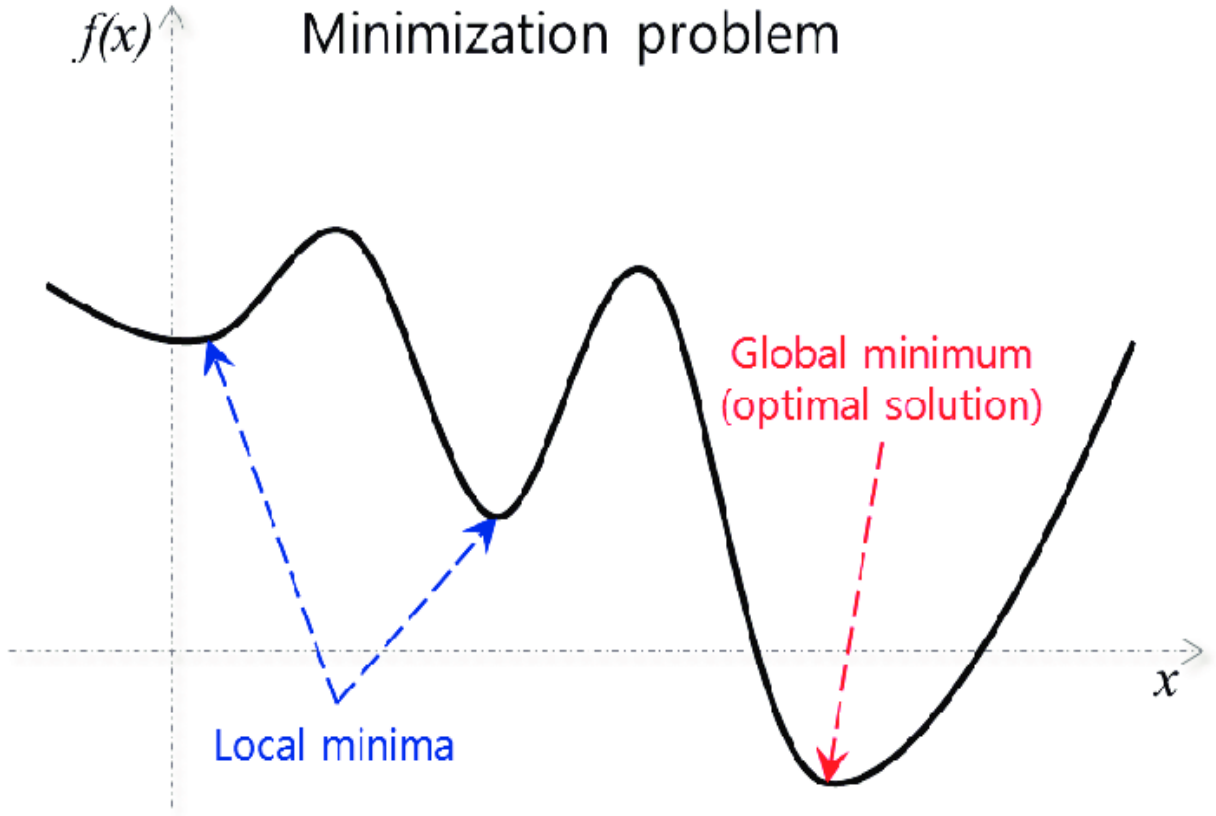
Greedy Algorithm

- In computer science, the greedy approach is often used for optimization problems, where the goal is to find the best solution from a set of possible solutions.
- A greedy algorithm is an approach for solving a problem by selecting the best option available at the moment. It doesn't worry whether the current best result will bring the overall optimal result. The hope is that by making these locally optimal choices, the algorithm will eventually reach a globally optimal solution.



Greedy Algorithm (cont'd)

- The algorithm never reverses the earlier decision even if the choice is wrong. While greedy algorithms are powerful and widely used, they are not suitable for every problem, and careful consideration must be given to the characteristics of the problem to determine whether a greedy approach is appropriate.



Greedy Algorithm (cont'd)

- We can determine if the algorithm can be used with any problem if the problem has the following properties:

1. Greedy Choice Property

This property allows a greedy algorithm to make decisions based solely on the information available at the current step without considering the consequences on future steps. The algorithm makes the best possible choice at each stage, hoping to reach a globally optimal solution.

2. Optimal Substructure

If the optimal overall solution to the problem corresponds to the optimal solution to its sub problems, then the problem can be solved using a greedy approach. This property is called optimal substructure.

Optimal Substructure Example

Find F(A): the minimum distance required to reach node J from a node A

The distance from the vertex H to the vertex J (F(H)) is 3, and the distance from vertex I to vertex J (F(I)) is 4.

F(E) = min{ 1+F(H), 4+F(I)} =min{4, 8}=4

F(F) = min{ 6+F(H), 3+F(I)}= min{9, 7}=7

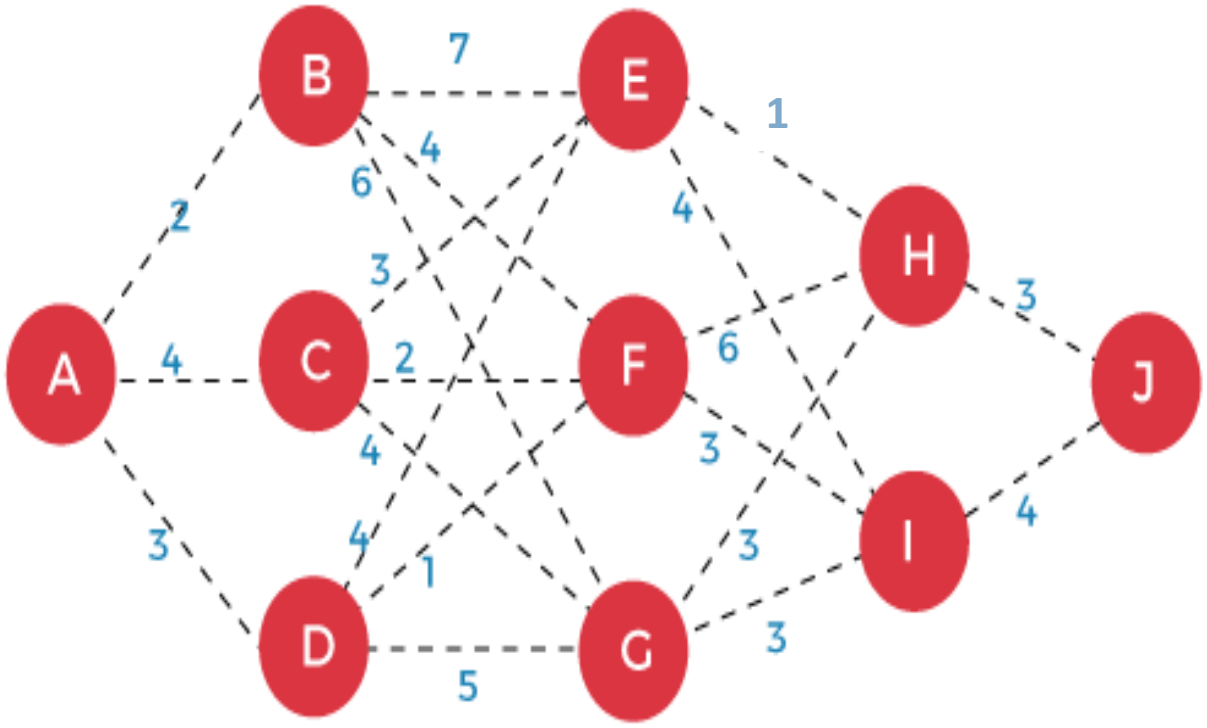
F(G) = min{ 3+F(H), 3+F(I)}=min{6, 7}=6

F(B) = min{ 7+F(E), 4+F(F), 6+F(G)}= min{11, 11, 12}=11

F(C) = min{ 3+F(E), 2+F(F), 4+F(G)}= min{7, 9, 10}=7

F(D) = min{ 4+F(E), 1+F(F), 5+F(G)}= min{8, 8, 11}=8

F(A) = min{ 2+F(B), 4+F(C), 3+F(D)}= min{13, 11, 11}=11



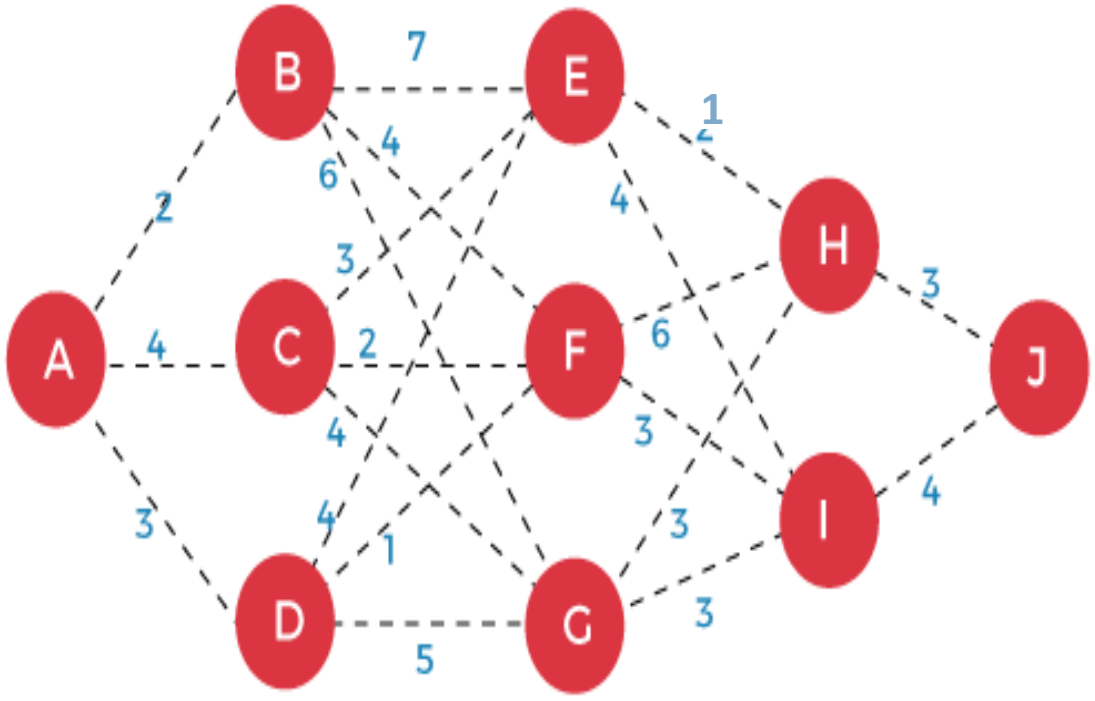
Optimal Substructure Example

Find F(A): the minimum distance required to reach node J from a node A

- Now, we have two ways to reach from the vertex A to J through vertex D:

  1) The first way is from vertex A to vertex D, then D to F, F to I and then I to J.
  2) The second way is from vertex A to vertex D, then D to E, then E to H and then H to J.

- Here, we have built an optimal solution using the sub-solutions to the problem. Therefore, we can say that the above problem has an optimal substructure.



Advantages of Greedy Approach

- The algorithm is **easier to describe**.
- This algorithm can **perform better** than other algorithms (but, not in all cases).

Drawback of Greedy Approach

- The greedy algorithm doesn't always produce the optimal solution. This is the major disadvantage of the algorithm

For example, suppose we want to find the longest path in the graph below from root to leaf:

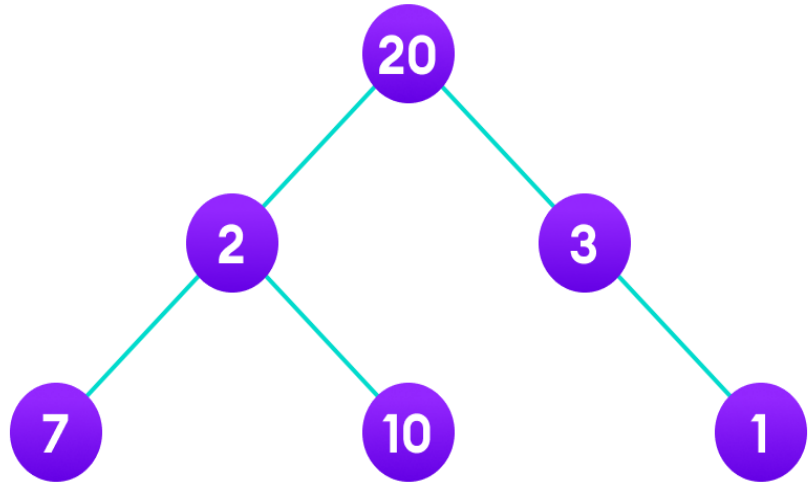
Greedy Approach

1. Let's start with the root node **20**. The weight of the right child is **3** and the weight of the left child is **2**.

2. Our problem is to find the largest path. And, the optimal solution at the moment is **3**. So, the greedy algorithm will choose **3**.

3. Finally the weight of an only child of **3** is **1**. This gives us our final result `20 + 3 + 1 = 24`.

However, it is not the optimal solution. There is another path that carries more weight (`20 + 2 + 10 = 32`) as shown in the image below.



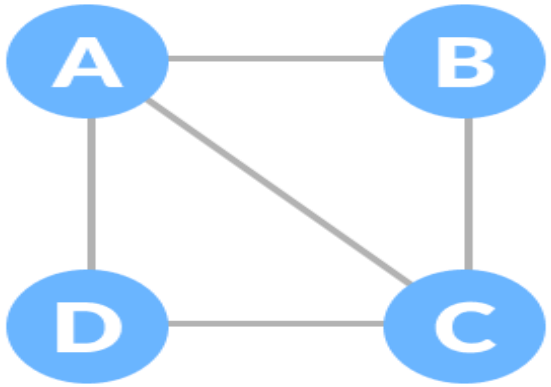
Different Types of Greedy Algorithm

- Kruskal's Minimal Spanning Tree Algorithm
- Prim's Minimal Spanning Tree Algorithm
- Single-Source Shortest Path Problem

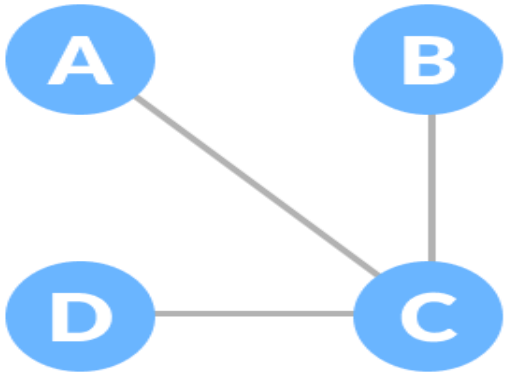
Undirected and connected graph

- An **undirected graph** is a graph in which the edges do not point in any direction (i.e. the edges are bidirectional).

- A **connected graph** is a graph in which there is always a path from a vertex to any other vertex.



Undirected Graph

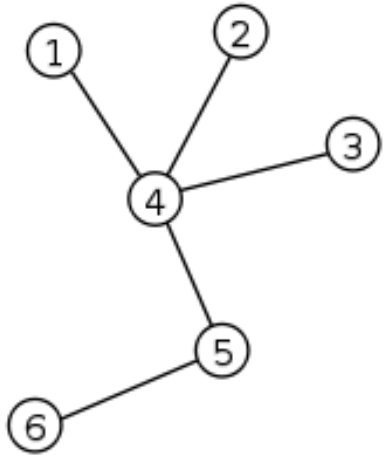


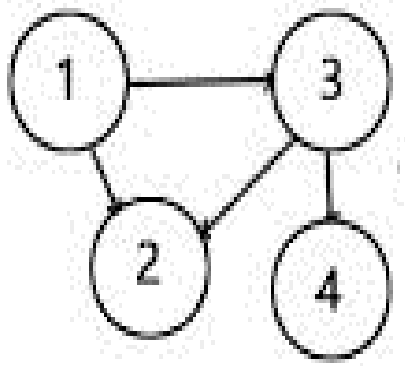
Connected Graph

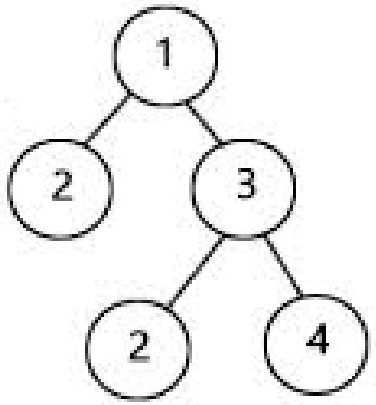
Spanning Tree and Minimum Spanning Tree

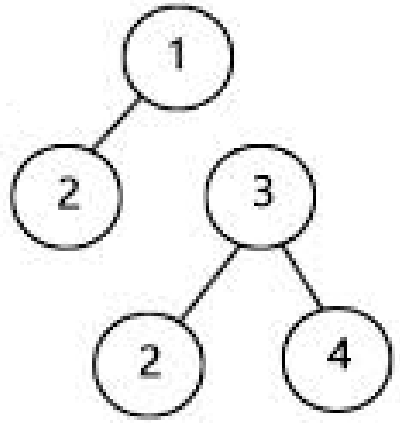
- A tree is a connected undirected graph with no simple circuits (acyclic).

- Are the following graphs trees?





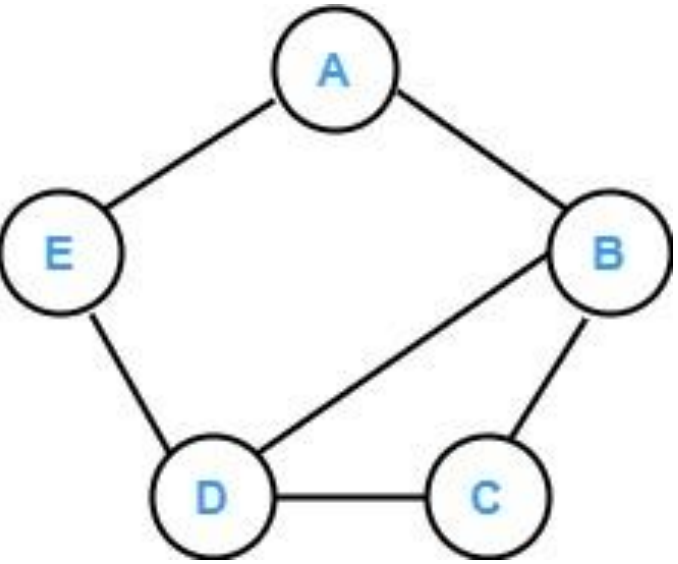




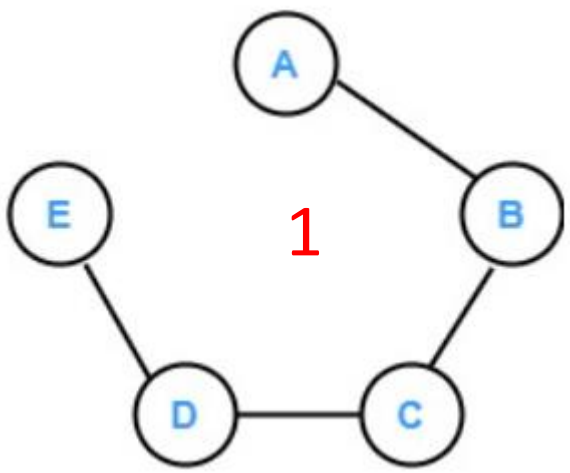
Spanning Tree and Minimum Spanning Tree (Cont'd)

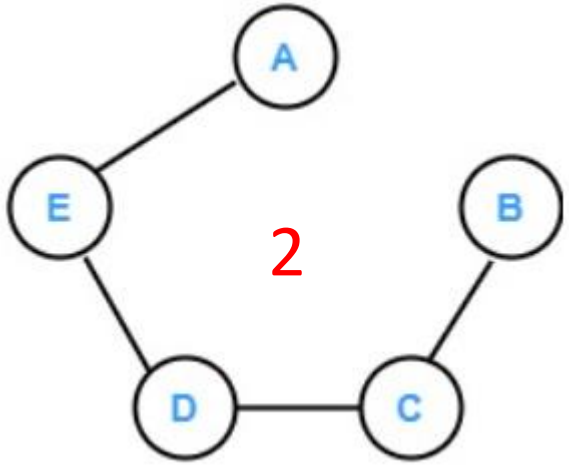
- A spanning tree is a sub-graph of an undirected connected graph, which includes all the vertices of the graph with a minimum possible number of edges. If a vertex is missed, then it is not a spanning tree.
- The edges may or may not have weights assigned to them.
- The total number of spanning trees with n vertices that can be created from a complete graph is equal to $n^{(n-2)}$.
- If we have n = 4 (vertices), the maximum number of possible spanning trees is equal to $4^{4-2} = 16$. Thus, 16 spanning trees can be formed from a complete graph with 4 vertices.
- Spanning tree has n-1 edges, where n is the number of nodes (vertices).

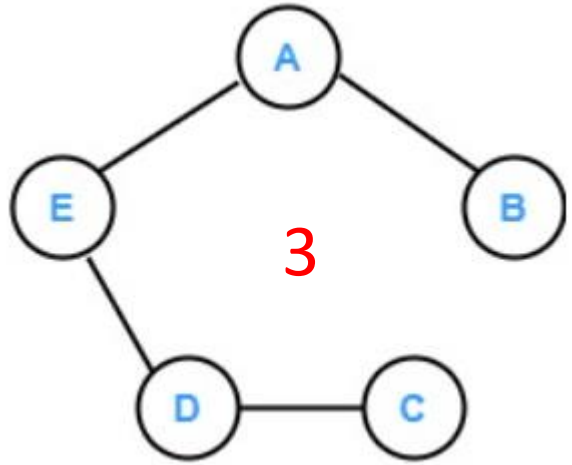
Example of a Spanning Tree

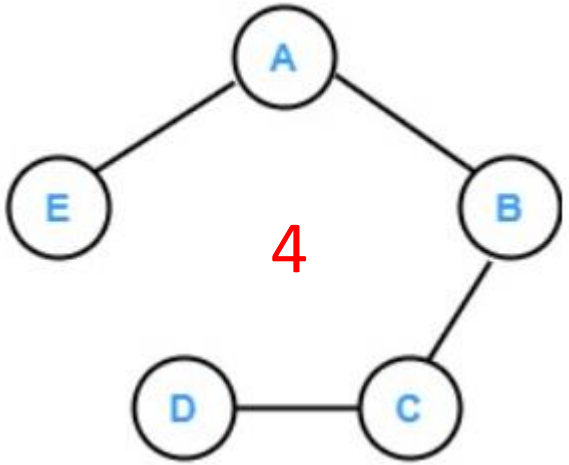


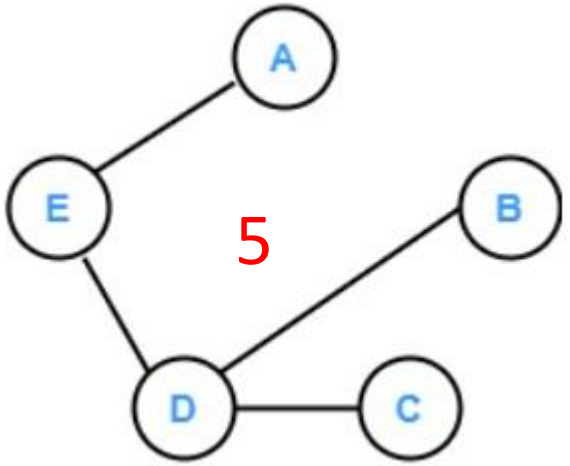
Normal graph

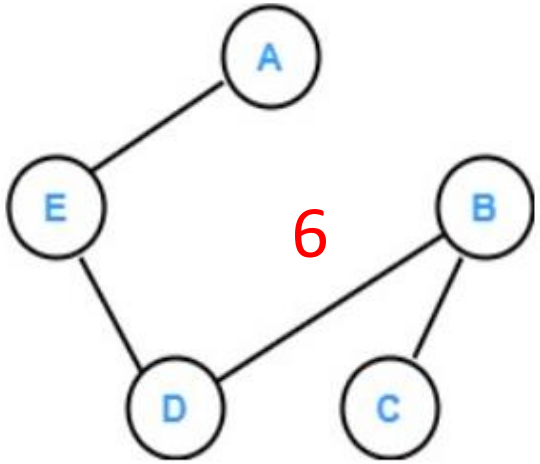












Some of the possible spanning trees that can be created

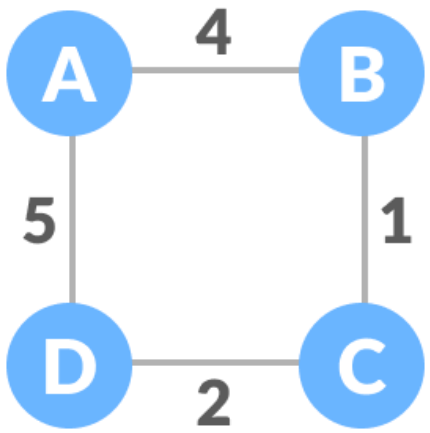
General Properties of Spanning Tree

Following are a few properties of the spanning tree connected to graph G:

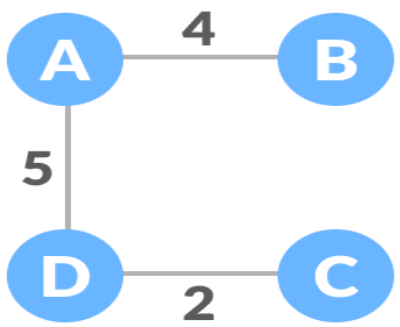
- A connected graph G can have more than one spanning tree.
- All possible spanning trees of graph G, have the same number of edges and vertices.
- The spanning tree does not have any cycle (loops).
- Removing one edge from the spanning tree will make the graph disconnected, i.e. the spanning tree is **minimally connected**.
- Adding one edge to the spanning tree will create a circuit or loop, i.e. the spanning tree is **maximally acyclic**.
- The spanning tree generated from weighted graph is called weighted spanning tree and the weight of spanning tree is equal to the sum of the weights of the edges in that spanning tree.

Minimum Spanning Tree (MST)

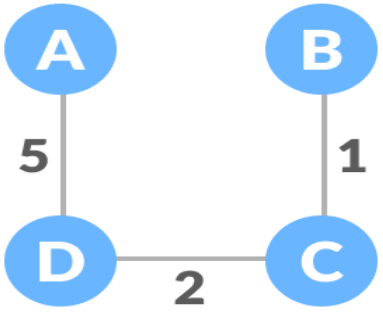
- A minimum spanning tree is a spanning tree in which the sum of the weight of the edges is as minimum as possible.
- Example of a Minimum Spanning Tree



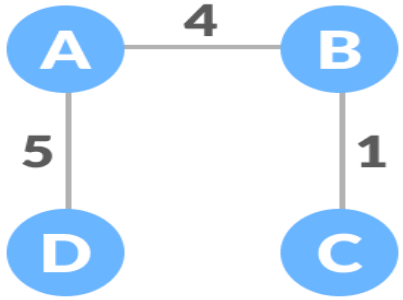
Weighted graph



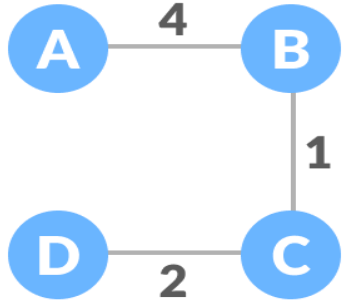
sum = 11



sum = 8



sum = 10



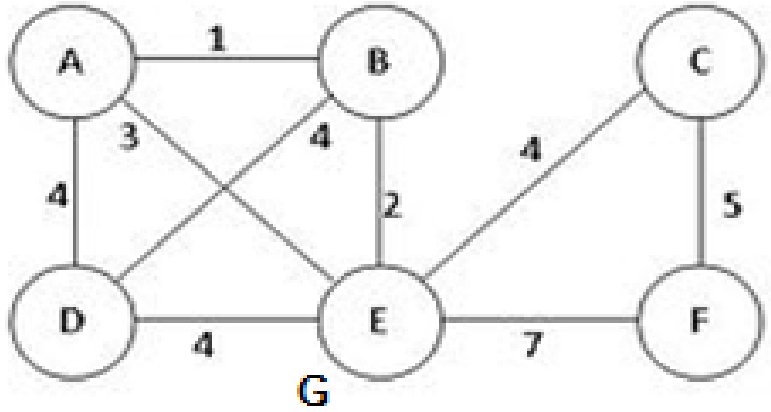
sum = 7

minimum weighted spanning tree

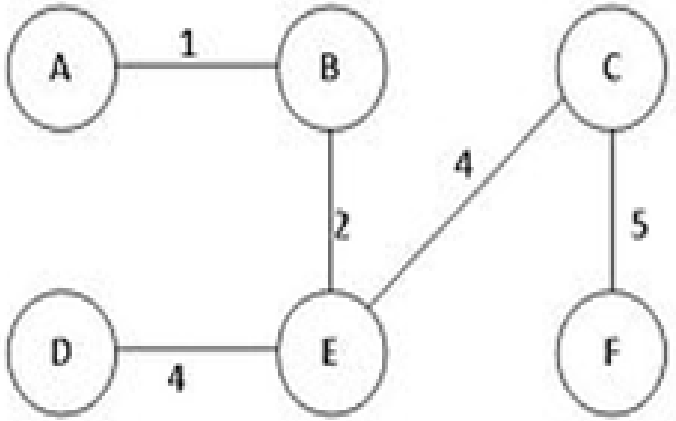
Minimum Spanning Tree (cont'd)

- Remark:
  The minimum spanning tree is not unique.

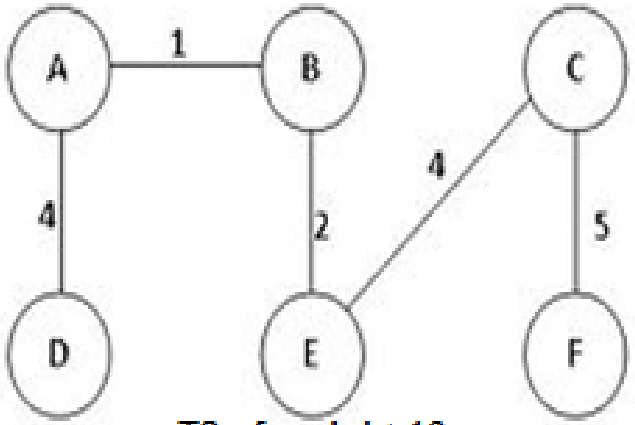
- The following graph has two minimum spanning tree:



G



T1 of weight 16



T2 of weight 16

Minimum Spanning Tree (cont'd)

The MST from a graph is found using the following algorithms:

- Kruskal's Minimal Spanning Tree Algorithm
- Prim's Minimal Spanning Tree Algorithm
- Single-Source Shortest Path Problem

Kruskal's Algorithm

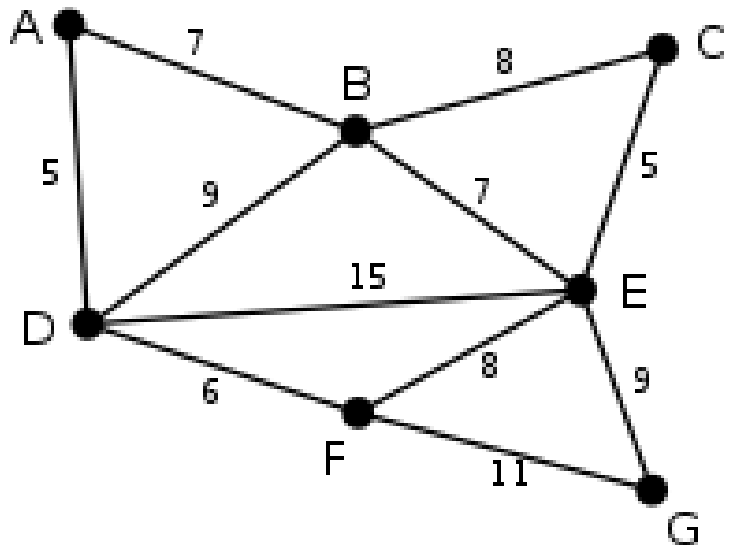
Input Connected, undirected, weighted graph, G (V, E) .

Output Minimum - weight spanning tree, T

Main Idea

1. Sort the edges in non decreasing order by weight.
2. Start by (V,T) consisting of all the vertices of the graph and none of its edges.
3. Repeat the following until the number of edges in tree is n-1. **(n is the** number of vertices**)**
    1. Let e in E-T be the current edge.
    2. If adding e to T does not create a cycle then include e in T. Otherwise, discard e.

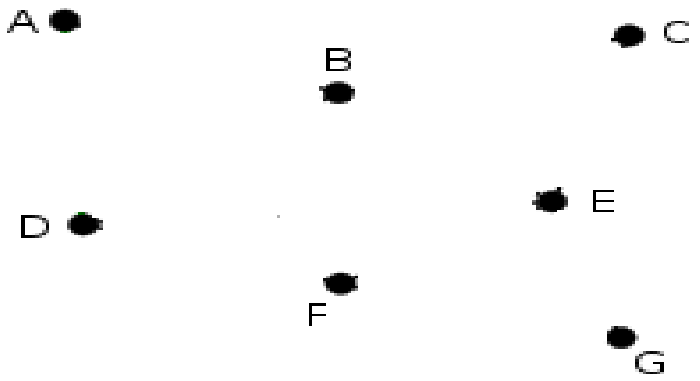
This is our original graph. The numbers near the arcs indicate their weight. None of the arcs are highlighted



- **Sort the edges of the graph as follows:**

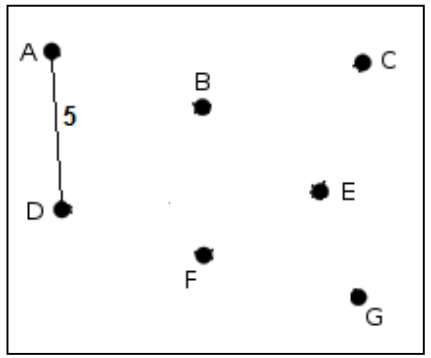
Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

- **Construct a graph T consists on n=7 vertices without any edges as follows:**

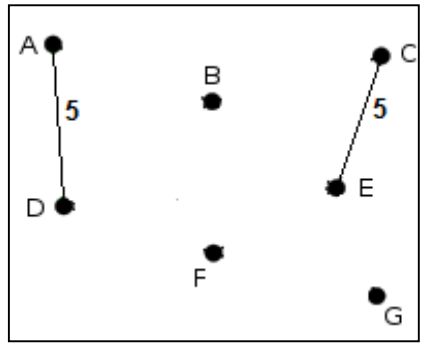


Select the minimum edge such that does not make cycle.

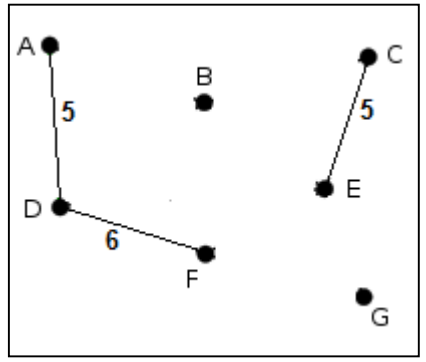
Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15



Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

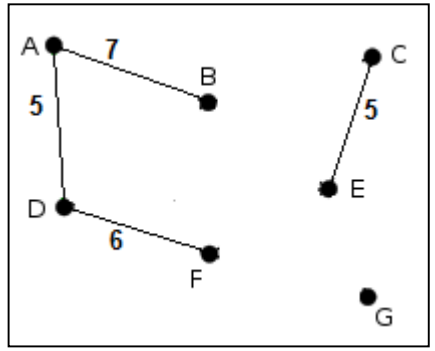


Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

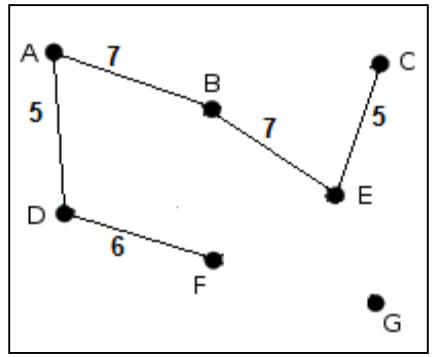


Select the minimum edge such that does not make cycle.

Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

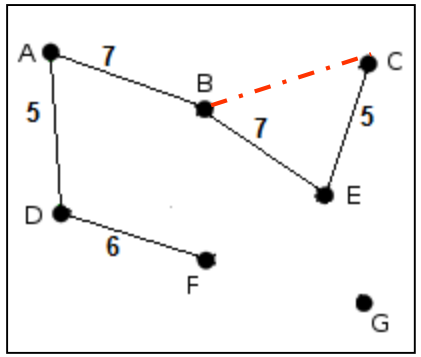


Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15



Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

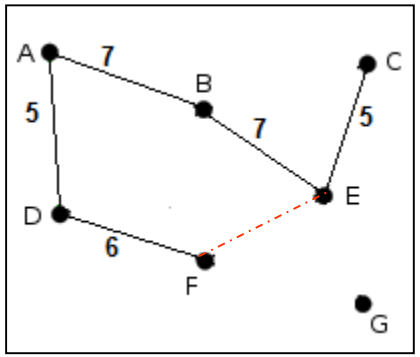
Make a cycle



Select the minimum edge such that does not make cycle.

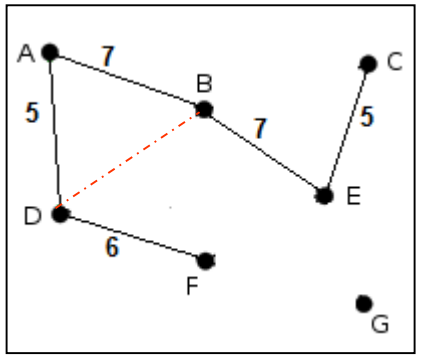
Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

Make a cycle

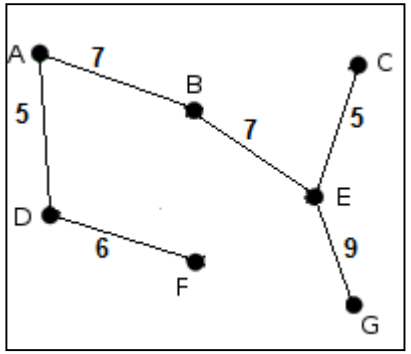


Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

Make a cycle

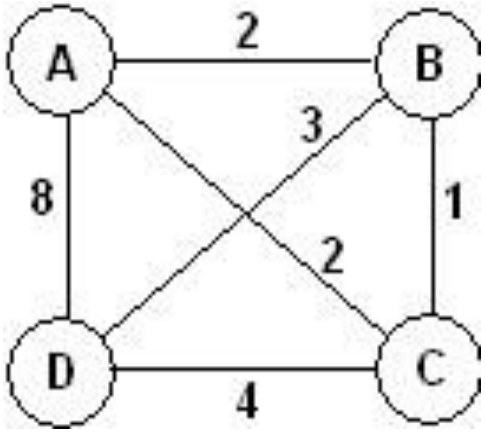


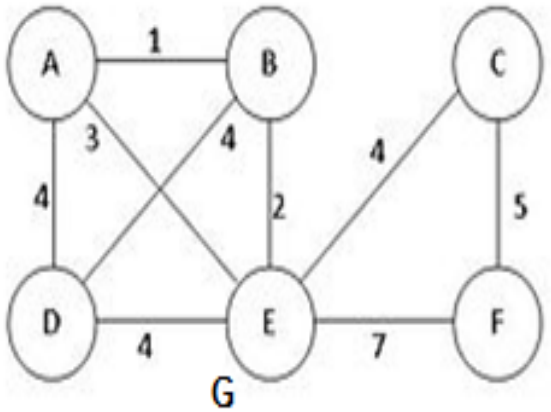
Edge	AD	CE	DF	AB	BE	BC	EF	BD	EG	FG	DE
Weight	5	5	6	7	7	8	8	9	9	11	15

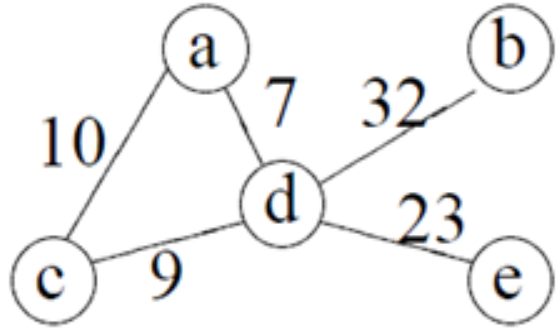


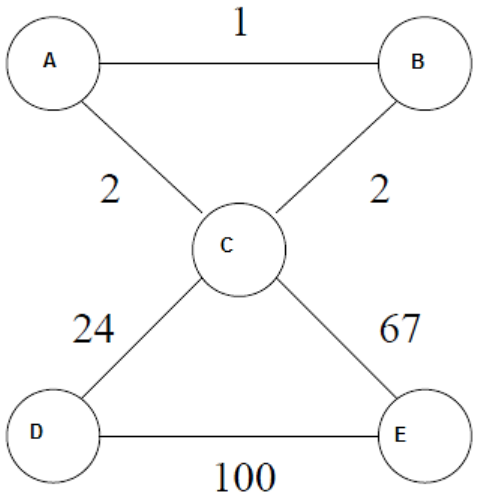
Stop the process, because the number of edges equal to the number of vertices -1. The minimum weight is 39.

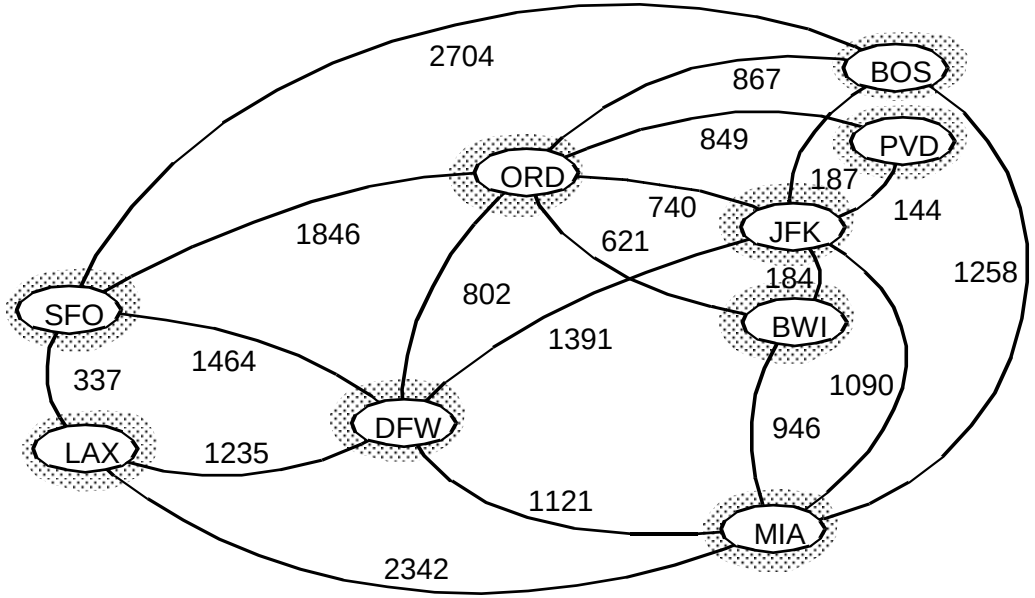
Home Work: Apply the algorithm on the following graphs to find MST











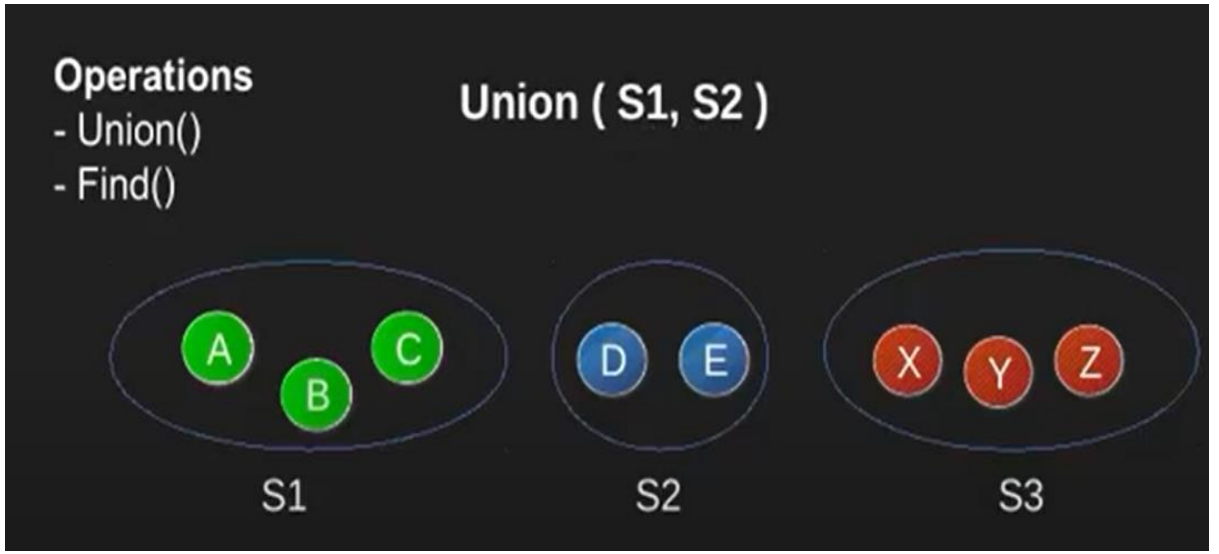
Detect Cycle in an Undirected Graph:

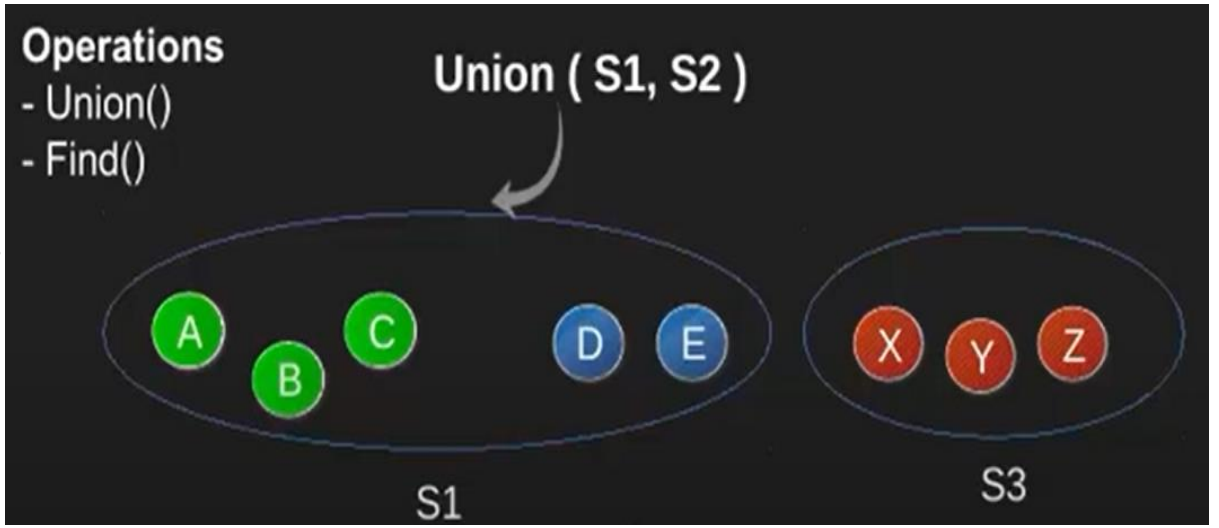
union-find algorithm

A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

Union: Join two subsets into a single subset. Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union.

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.



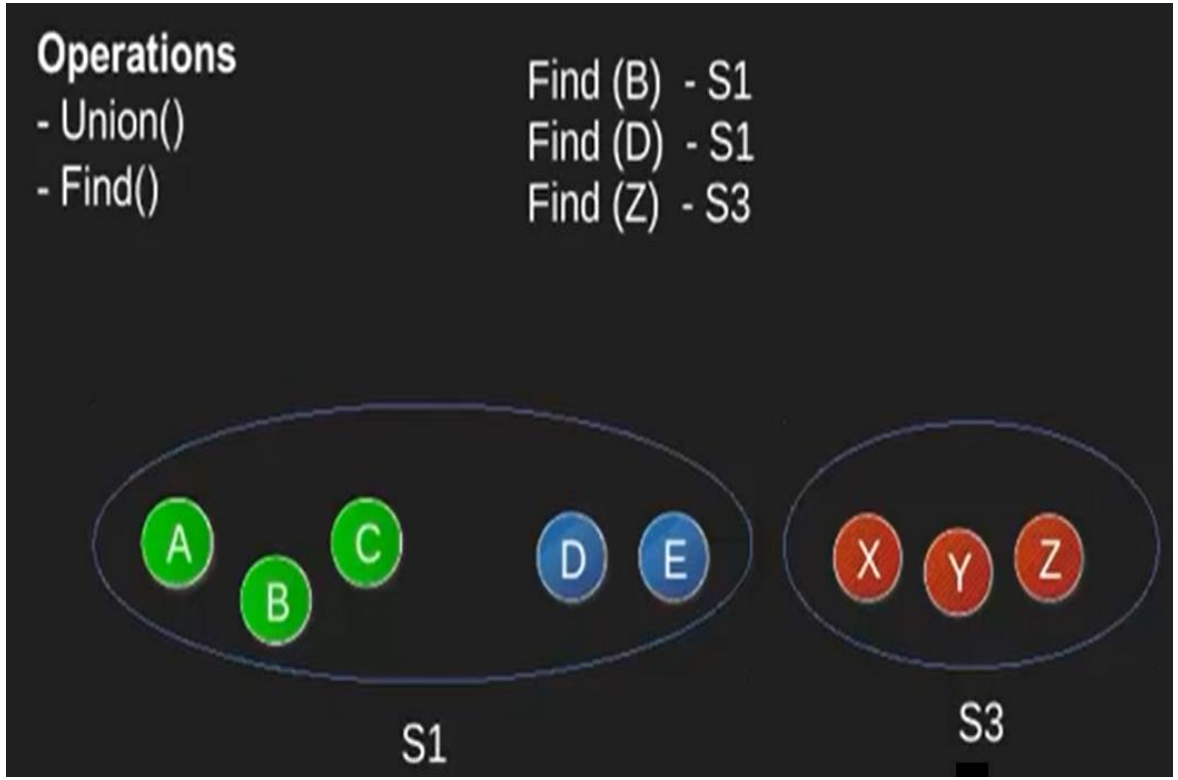


Detect Cycle in an Undirected Graph

A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

Union: Join two subsets into a single subset. Here first we have to check if the two subsets belong to same set. If no, then we cannot perform union.

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.

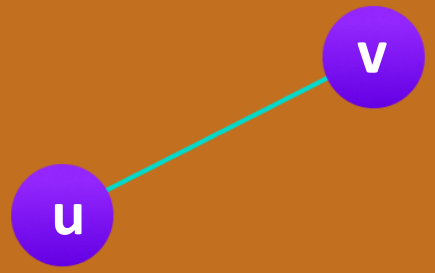


Cycle Detection

- Any minimum spanning tree algorithm revolves around checking if adding an edge creates a loop or not. The most common way to find this out is an algorithm called Union Find. The Union-Find algorithm:

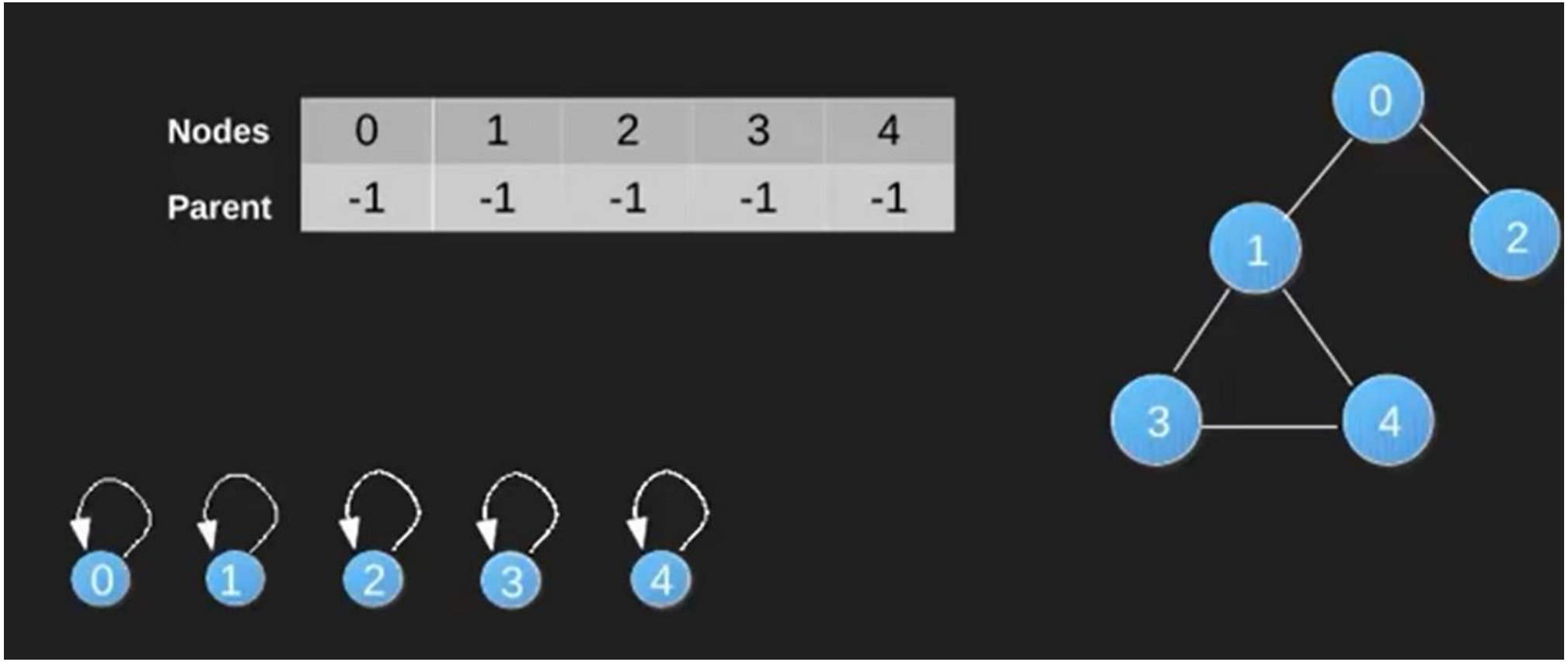
Idea : For each edge, make subsets using both the vertices of the edge. If both the vertices are in the same subset, a cycle is found.

```
for each unvisited edge (u, v) in edge set E
{
    If( Find(u) == Find(v) )
    {
        // cycle detected
    }
    else
        Union(u, v);
}
```

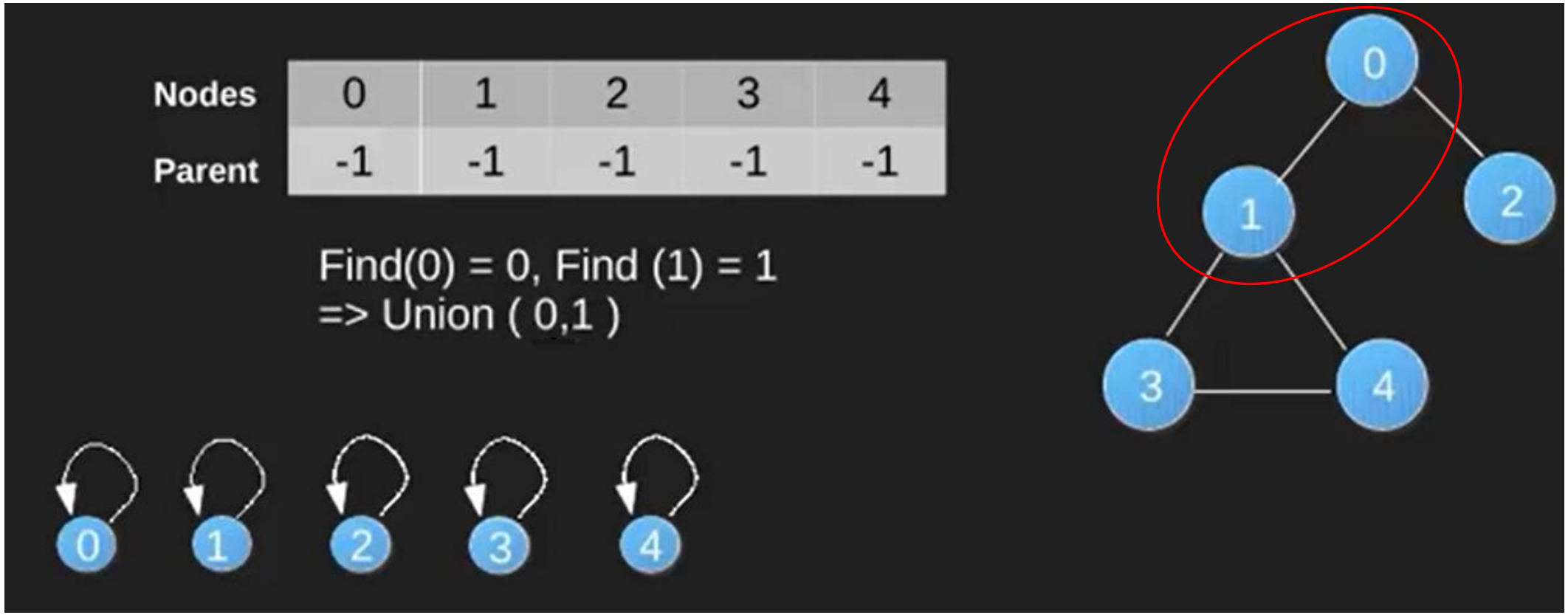


Pseudo code of union-find algorithm

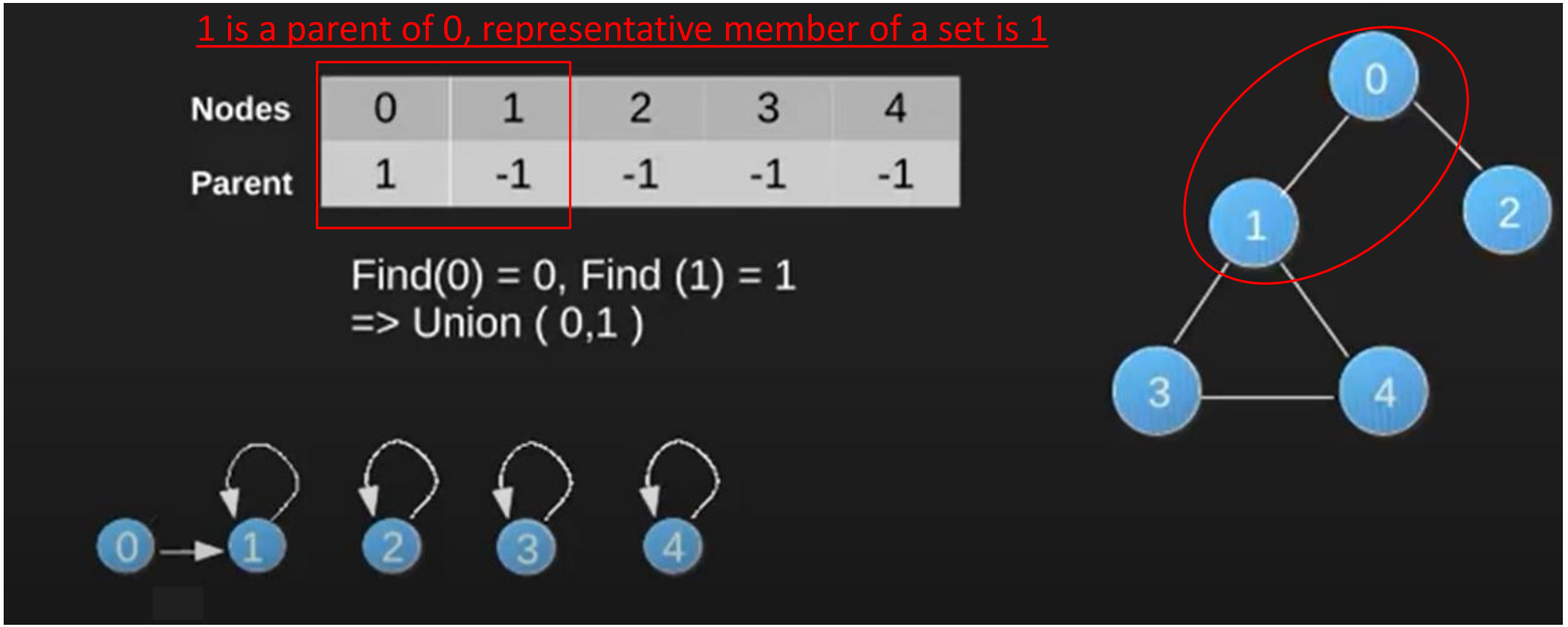
Cycle Detection (Example)



Cycle Detection (Example)

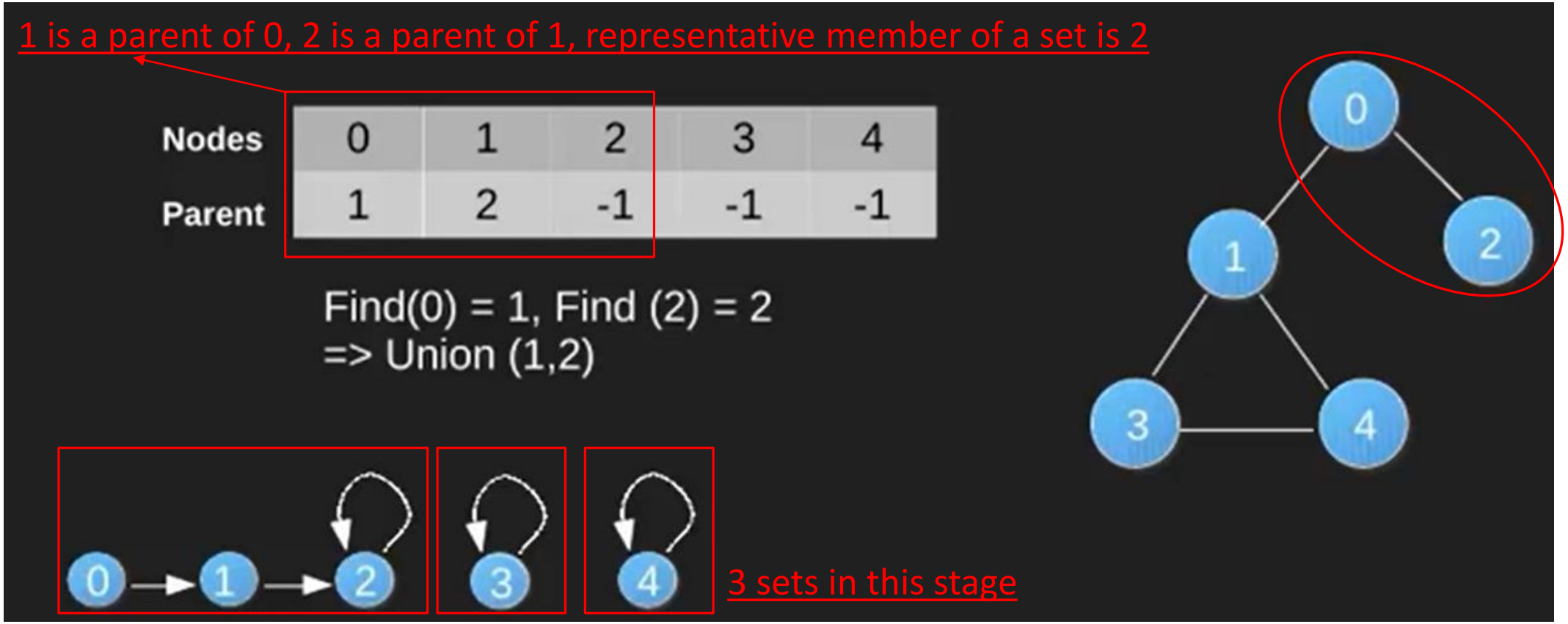


Cycle Detection (Example)

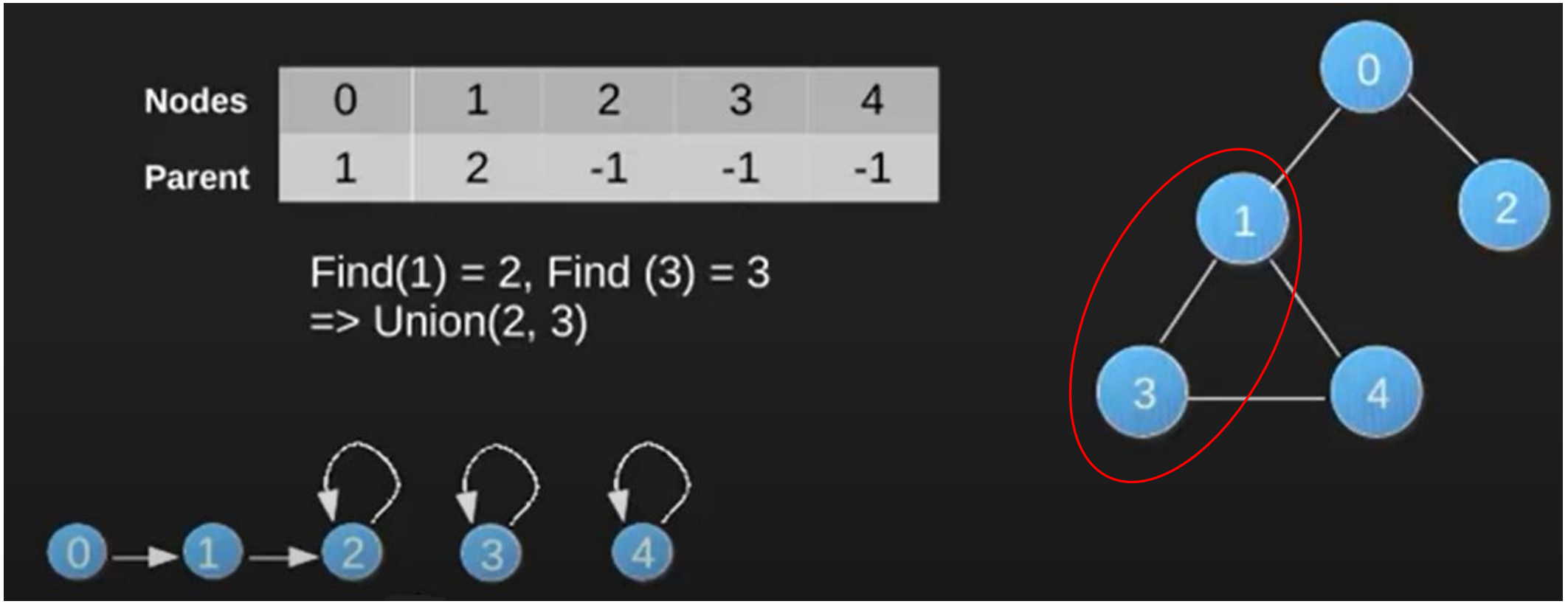


Cycle Detection (Example)

Cycle Detection (Example)



Cycle Detection (Example)



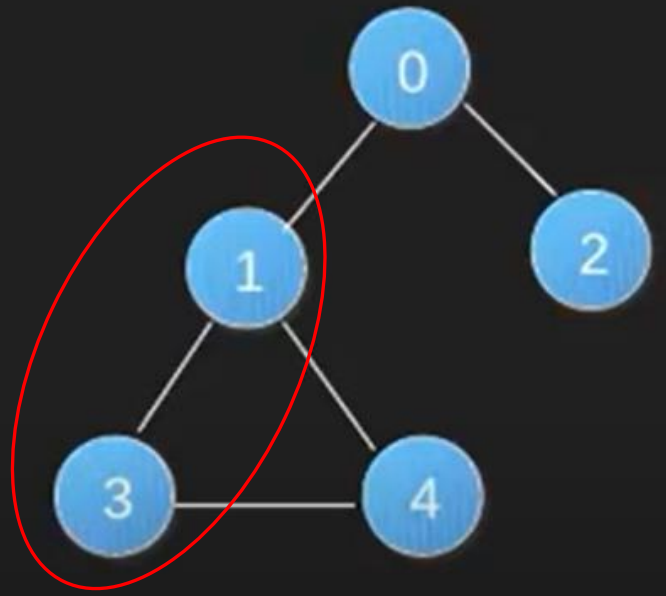
Cycle Detection (Example)

representative member of a set is 3

Nodes	0	1	2	3	4
Parent	1	2	3	-1	-1

Find(1) = 2, Find (3) = 3
=> Union(2, 3)

2 sets in this stage



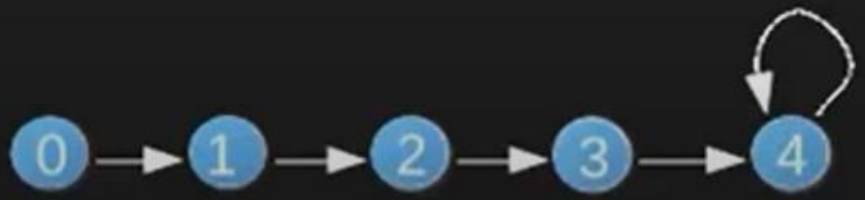
Cycle Detection (Example)

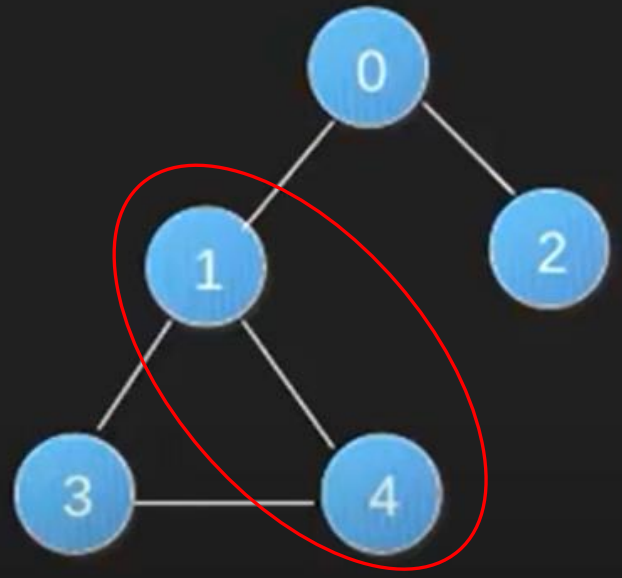
Cycle Detection (Example)

representative member of a set is 4

Nodes	0	1	2	3	4
Parent	1	2	3	4	-1

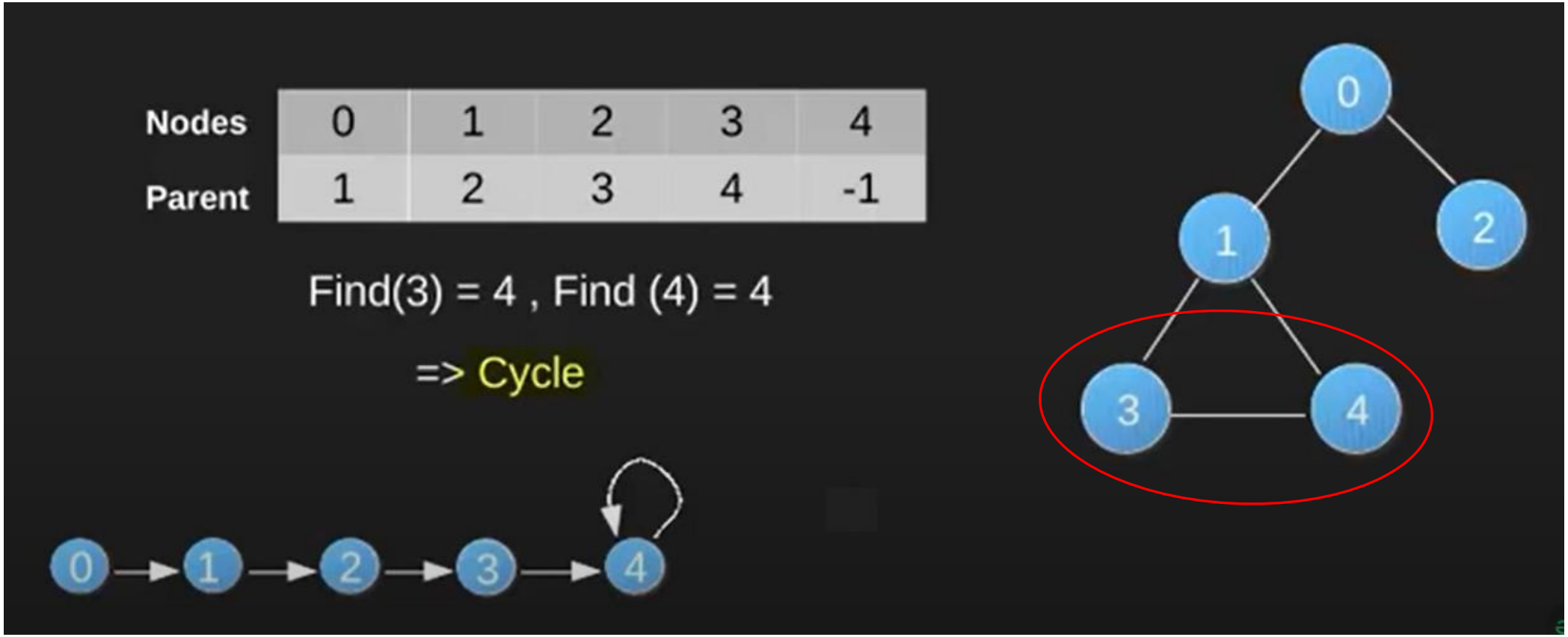
Find(1) = 3, Find (4) = 4
=> Union (3,4)





Cycle Detection (Example)

Cycle Detection (Example)



Running Time

Algorithm: **Kruskal**

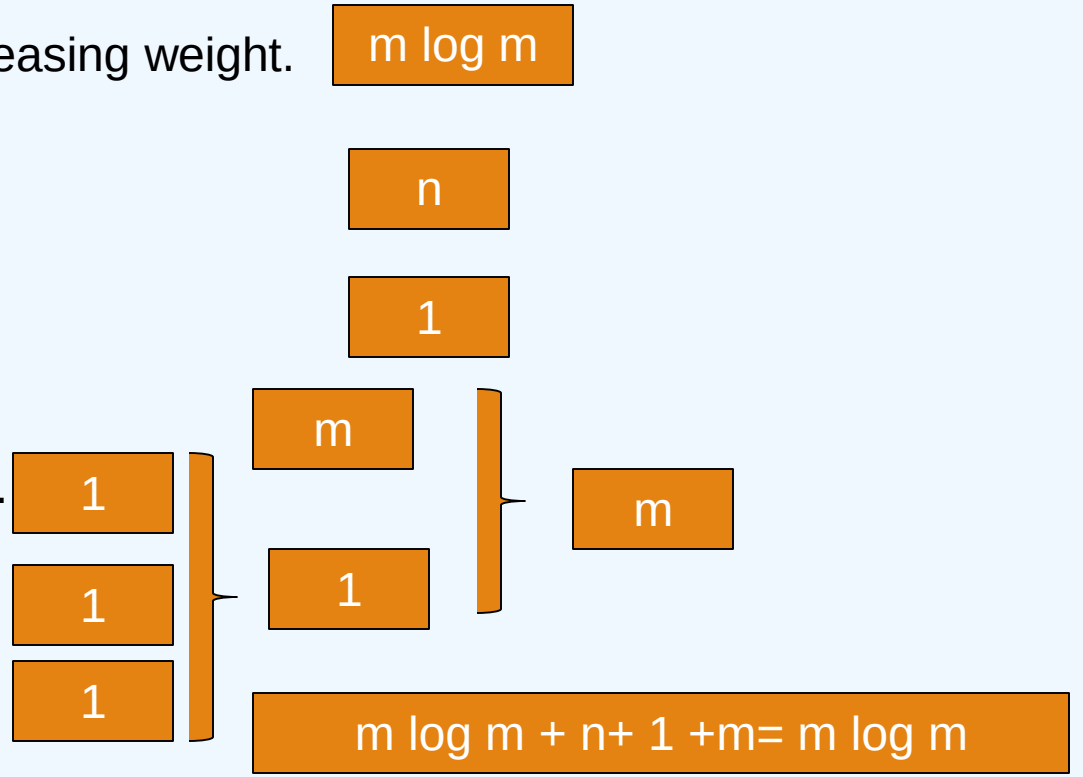
Input: A weighted connected undirected graph G=(V,E) with n vertices.

Output: The set of edges T of a minimum cost spanning tree for G.

Begin

1. Sort the edges in E by nondecreasing weight.
2. For each vertex v in V
   MAKESET({v})
3. T={}
4. While |T| < n-1 do
   let (x,y) be the next edge in E.
   if FIND(x) /= FIND(y) then
   Add(x,y) to T
   UNION(x,y)

End.



Running Time = O(m log m) (m = edges, n = nodes)