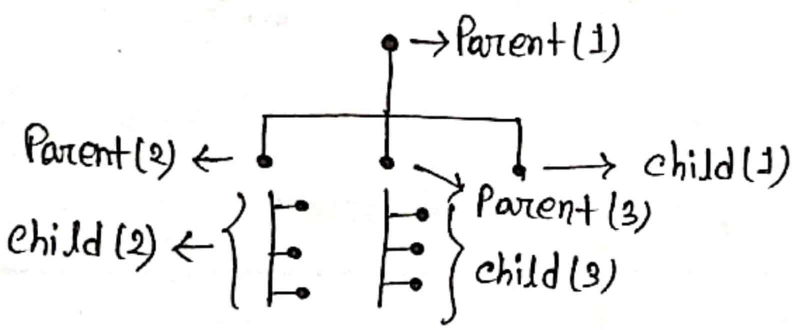
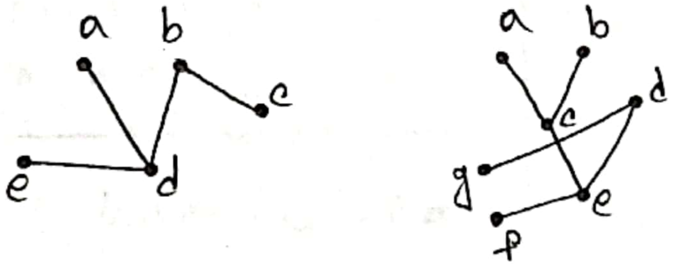
# Tree

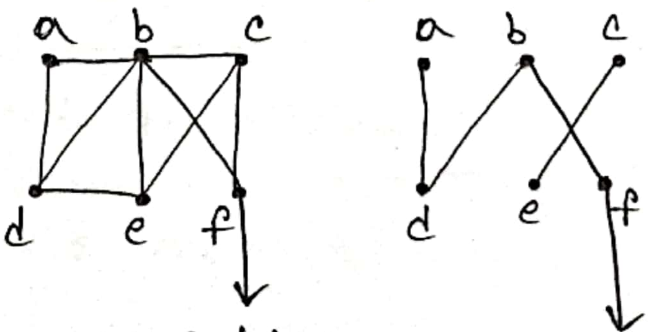
- A tree is a connected undirected graph with
  - ⇒ no simple circuits
  - ⇒ no multiple edges
  - ⇒ no loops
  - ⇒ any tree must be a simple graph
- An undirected graph is a tree if and only if there is a unique sample path between any two of its vertices.
- Consists of nodes with a parent child




Ex:




(Example of tree)

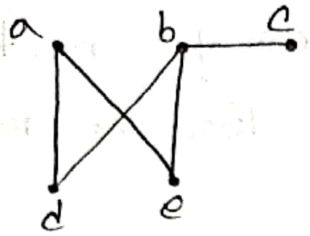



(Example of not tree)

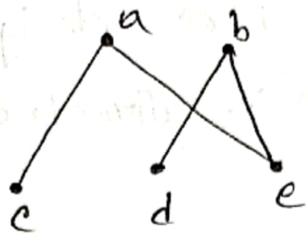
Contain a cycle or loop

Not connected graph

Which of the following graphs are trees?




(Not tree)




(A tree)

**Forest:** A forest is an undirected graph with no simple circuits

একাধিক Tree

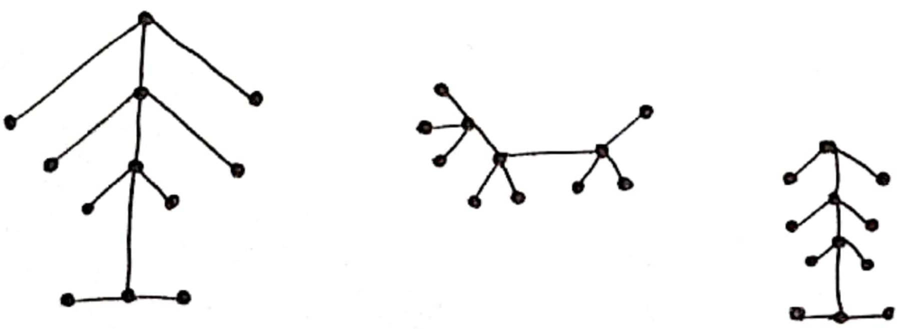
The only differences between a forest and a tree is the word connected.

⇒ If each components of a graph is connected it is tree.

⇒ If the components of a graph is not connected then it is forest.

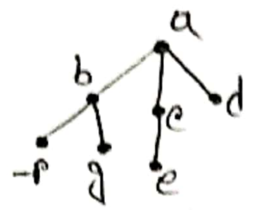
Example:

⊛ This is one graph with three connected components.



Forest example

Rooted trees: A rooted tree is a tree in which one vertex has been designated as the root and every edge is directed away from the root.




A tree with root a

Tree terminology:

① If u is the parent of v, v is called the child of u.

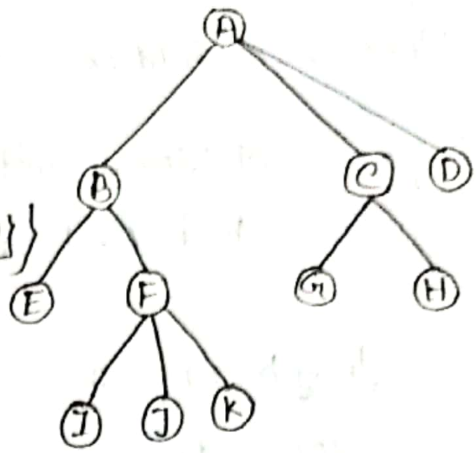
② vertices with the same parent are called siblings.

③ A vertex of a tree is called a leaf if it has no children.

④ vertices that have children are called internal vertices.

⑤ The descendants of a vertex v are those vertices that have v as an ancestor.

**Root :** node without parent (A)



**Siblings:** nodes share the same Parent {{B,C,D}, {E,F}, {I,J,K}, {G,H}}

**Internal node:** node at least one child (A, B, C, F)

**External node /leaf:** node with no child (E, I, J, K, G, H, D)

**Ancestors of a node:** Parent, grandparent, grand-grand~~child~~ Parent etc. ( উপরের দিকে যাবে) → F এর B, A এনসেস্টর

**Descendant of a node:** child, grandchild, grand-grandchild, etc. [যেমন: B এর descendant হচ্ছে E,F / F এর descendant I,J,K)

**Depth of a node:** number of ancestors (depth of A is 0).

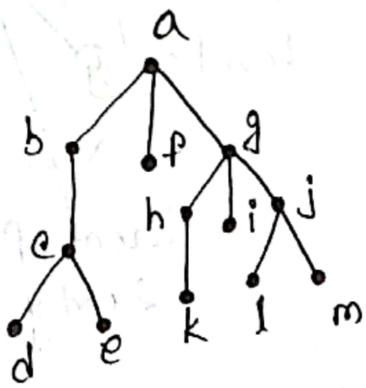
**Height of a tree:** maximum depth of any node (3).

**Degree of a node:** the number of its children.

**Degree of a tree:** maximum number of its node.

**Subtree:** tree consisting of a node and its descendant

**Properties of rooted trees:** (Example)



**Parent:** A vertex other than root is a parent if it has one or more children

→ The parent of c is b

Children – children of a is b, f and g

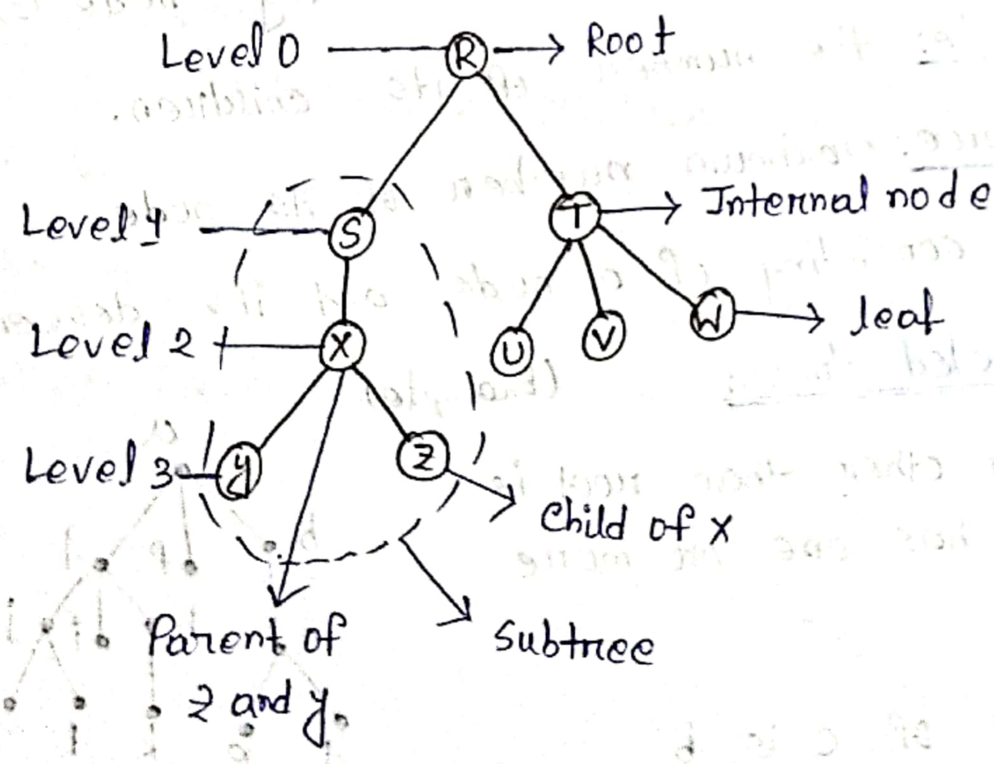
Siblings – children with the same parent vertex.
h, i and j are siblings.

Level – length of the unique path from the root to a vertex

- Vertex a is at level 0
- Vertices d and e is at level 3

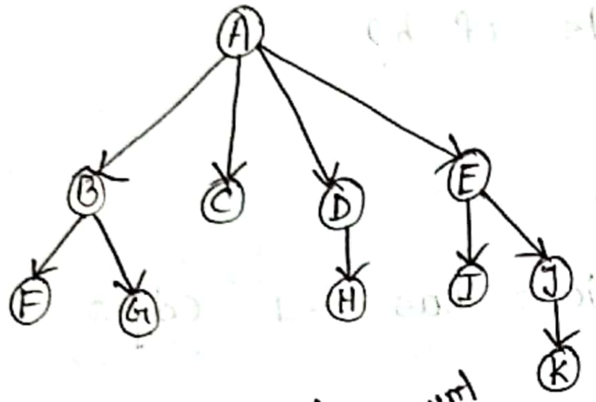
Height – Maximum level of all vertices
- height of this tree is 3.

Tree anatomy:



☐ The depth of node $n_i$ is the length of the path from root to node $n_i$

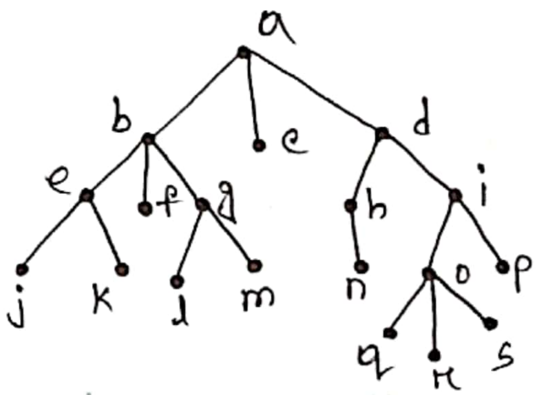
☐ The height of node $n_i$ Is the length of longest path from node $n_i$ to a leaf




Height → ~~level~~ length count
Depth – level count

| Node | Height | Depth |
|---|---|---|
| A | 3 | 0 |
| B | 1 | 1 |
| C | 0 | 1 |
| D | 1 | 1 |
| E | 2 | 1 |
| F | 0 | 2 |
| G | 0 | 2 |
| H | 0 | 2 |
| I | 0 | 2 |
| J | 1 | 2 |
| K | 0 | 3 |

Task:




① Which vertex is the root?
⇒ a

② Which vertices are internal?
⇒ a, b, d, e, g, h, i, o

③ Which vertices are leaves?
⇒ c, f, j, k, l, m, n, q, r, s, p

④ Which vertices are children of j?
⇒ No children

⑤ Which vertex is the parent of h?
⇒ d

⑥ Which vertices are siblings of o?

⇒ p

⑦ Which vertices are ancestors of m?

→ g, b, a

⑧ Which vertices are descendants of b?

⇒ e, f, g, j, k, l, m.

Theorem: A tree with n vertices has n−1 edges.

## M-ary trees:

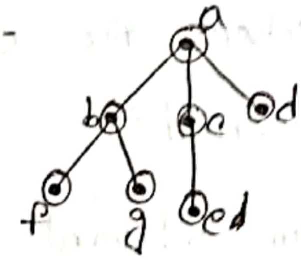
→ A rooted tree is called an m-ary tree if every internal vertex has no more than m children.

→ The tree is called a full m-ary tree if every internal vertex has exactly m children.

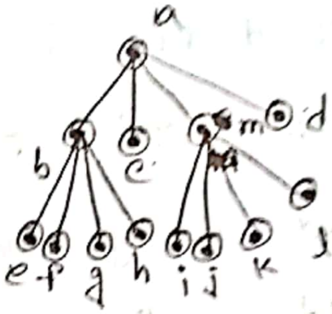
→ An m-ary tree with $m=2$ is called a binary tree.

→ A rooted m-ary tree is balanced if all leaves are at levels h or $h-1$

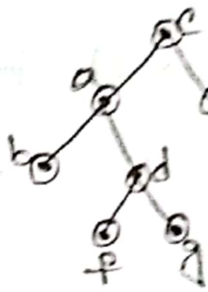
Example-




A 3-ary tree (ternary) 3টা করে



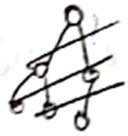

full 4-ary tree




A full binary tree $m=2$

**Theorem 1:**

A full m-ary tree with i internal vertices contains $n = mi+1$ vertices.



~~2×2+1~~ → 5

**Theorem 2:** A full m-ary tree with

(i) 'n' vertices has $i = \frac{n-1}{m}$ internal vertices and $l = \frac{(m-1)n+1}{m}$ leaves.

(ii) 'i' internal vertices has $n = mi+1$ vertices and $l = (m-1)i+1$ leaves.

(iii) 'l' leaves has $n = \frac{ml-1}{m-1}$ vertices and

$i = \frac{l-1}{m-1}$ internal vertices

Theorem 3: There are at most $m^h$ leaves in an m-ary tree of height h'.

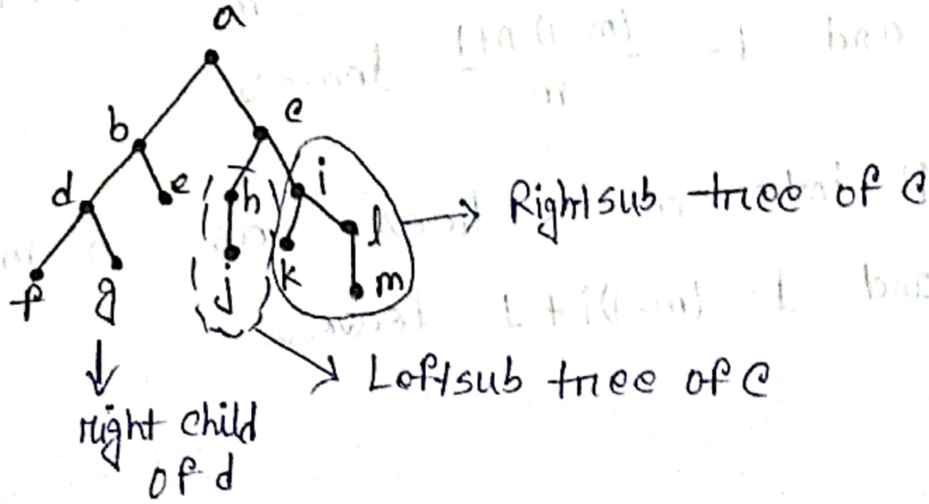
## Ordered rooted tree:

⇒ An ordered rooted tree is one where the children of each internal vertex are ordered.

⇒ In an ordered binary tree, if an internal vertex has two children, then they are called left child and right child.

⇒ A subtree rooted at the left child of a vertex is called the left subtree and subtree rooted at the right child of a vertex is called the right subtree.

Example:



## Traversal Algorithms:

A traversal algorithms is the procedure of systematically visiting each vertex of an ordered rooted tree.

⇒ Tree traversals are defined recursively.

⇒ Three types of algorithms for tree traversals are-

① Pre order traversal

② In-order Traversal

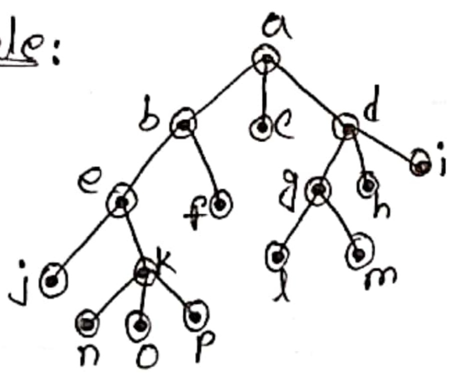
③ Post order traversal

Pre order : Root → Left → Right

In-order : Left → Root → Right

Post-order : Left → Right → Root

Example:

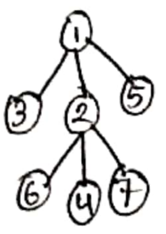



Pre order - a, b, e, j, n, o, p, f, c, d, g, l, m, h, i

~~Post~~ In order - j, e, n, k, o, p, b, f, a, c, l, g, m, d, h, i

Postorder - j, n, o, p, k, e, f, b, c, l, m, g, h, i, d, a

*




Pre → 1, 3, 2, 6, 7, 5

In → 3, 1, 6, 2, 4, 7, 5

post → 3, 6, 4, 7, 2, 5, 1

## Prefix, Infix, and postfix notation:

→ Representation of arithmetic expression involving operations $(+, -, *, /, \uparrow)$

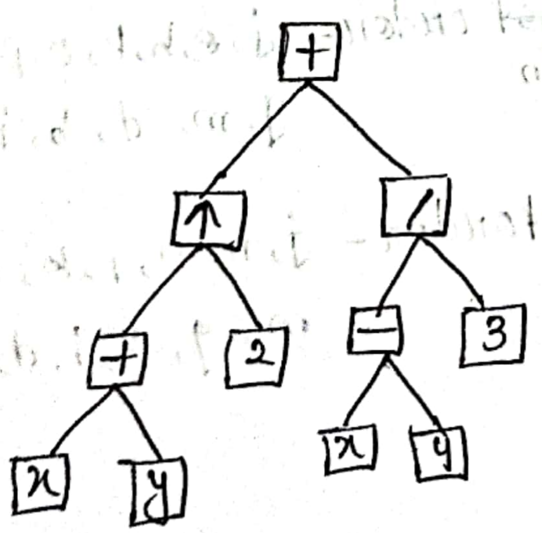
→ Parentheses is used to indicate the order of the operations.

→ internal vertices → operations
  leaves → Variable or number

→ Each operation operates on its left and right subtrees.

Ex: What is the prefix form for
$((x+y) \uparrow 2) + ((x-4)/3)$

Solution: $+ \uparrow x\, y\, 2 \;/ - x\, 4\, 3$



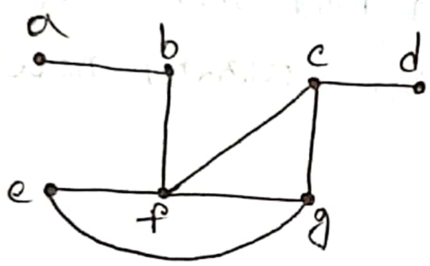
Prefix : Root + L + R

Infix : L + Root + R

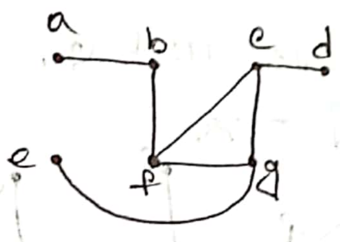
Postfix : L + R + Root

<u>Spanning trees:</u> A connected subgraph 'S' of graph G(V,E) is said to be spanning iff

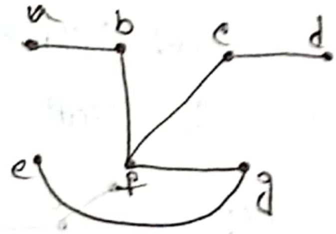
1. 'S' should be contain all vertices of 'G'.
2. 'S' should contain (|V|-1) edges




edge removed : {a,e}




{e,f}




{c,g}

- node connected না থাকলে spanning tree হয় না।
- Graph এর edge বাদদিয়ে হয় যেন Node connected থাকে and tree তে পরিণত হয়। সেটাই spanning tree.

~~Minimum spanning tree~~:

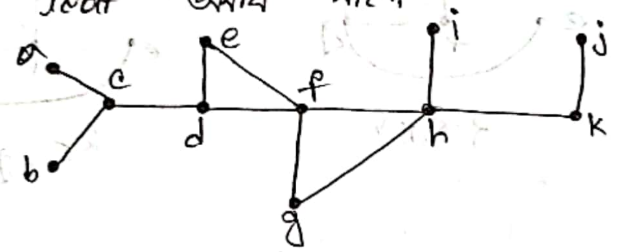
## Depth First Search Algorithm: (DFS)

The depth first search algorithm stards with the initial node of Graph G and goes leaper untill we find the goal node or the node with no children.

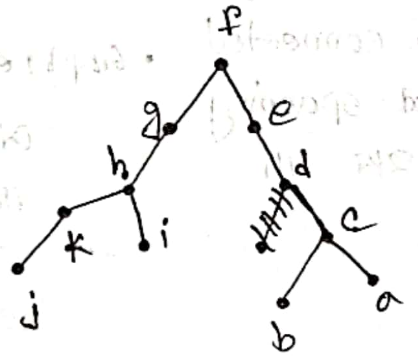
→ (DFS) goes through depth

→ এখানে একটা vertex এর depth এ গিয়ে leaf পর্যন্ত যাবে তারপর আবার rest এর মধ্যে আরেকটা nood এর leaf অবধি যাবে.

Ex:



Solution:



We start arbitraily with vertex 'f'. We build a Path by successively adding and edge that connects the last vertex added to the path and a vertex not already in the path, as long as this is Possible. The result is a path that connects f, g, h, k, and j. Next we return to k, but find no new vertices to add. So, we return to h and add the path with one edge that connects h and i. Then return to f and add the path connecting f, d, e, c and a. Finally return to c and add path connecting c and b. We know stop because all vertices have been added.

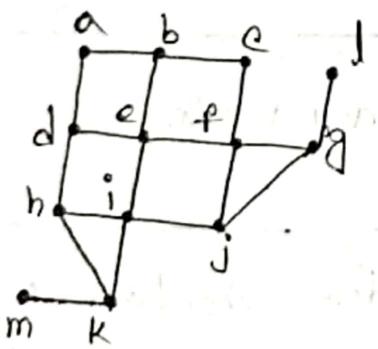
## Breadth first search Algorithm: (BFS)

Breadth-first search is an algorithm for search a tree data structure for a node that satisfies a given Property.

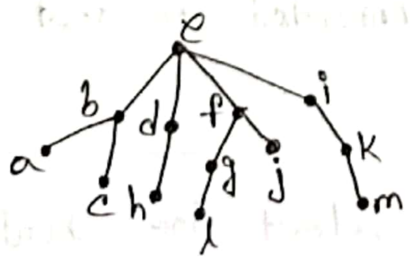
→ (BFS) goes through level

→ choose an vertex then adding the vertices and edges the vertex on the level one with the selected vertex.

Ex:




Solution:




We arbitrarily choose vertex 'e' as the root. We then add the edges from 'e' to b, d, f and i. These four vertices makeup level 1 in the tree.

Next we add the edges from b to a and e. the edges from d to h, the edges from f to j. and g, and the edge from i to k. The endpoints of these edges not at level 1 are at level 2. Next, add edges from these vertices to adjacent vertices not already in the graph. So, we add edges from g to l and from k to m. Now level 3 is made up. This is the last level as there is no new vertices to find.

## Minimum Connector Algorithms:

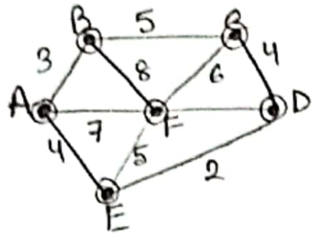
Kruskal's and prim's algorithm:

| Kruskal | Prim |
|---|---|
| ① select shortest edge in a network | ① select any Vertex. |
| ② Select the most shortest edge which does not create a cycle. | ② select the shortest edge connected to that Vertex |
| ③ Repeat step 2 untill all vertices have been connected. | ③ select the shortest edge connected to any Vertex already connected |

## Kruskal's Algorithm:

Choose the shortest edge then the most shortest

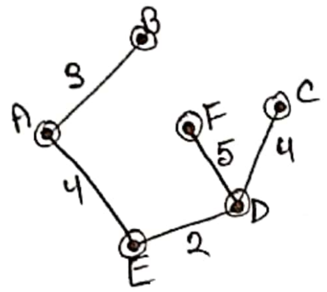
Ex:




ED - 2
AB - 3
AE - 4, CD - 4
FD - 5
BC - 5
CF - 6
BF - 8

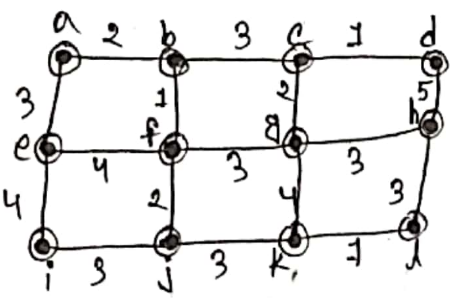
"cycle তৈরি হলে উহটা বাদ দিতে হবে"

The spanning tree:

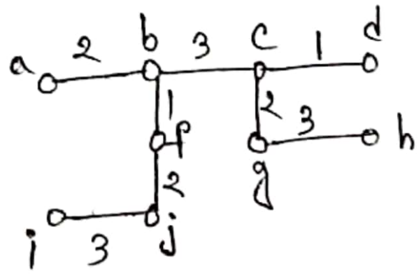



## Prim's Algorithm:

Choose any vertex and shortest edge of it but connected with the vertex.




⇒




of দিলে start দেবো যেই আগেরটা উঠাইলা add করব