Graph

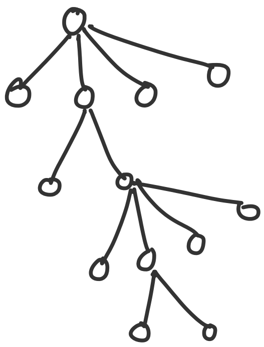
↓

A network

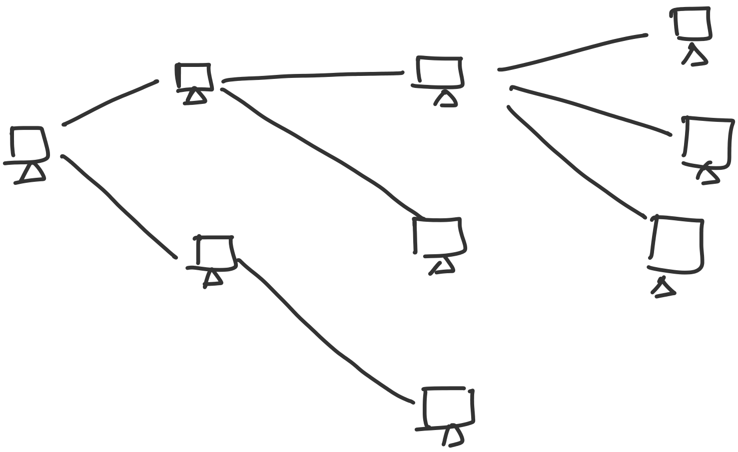
↓

can be physical - eg- social media connections, connection of computers, electrical circuit, flight paths, maps.

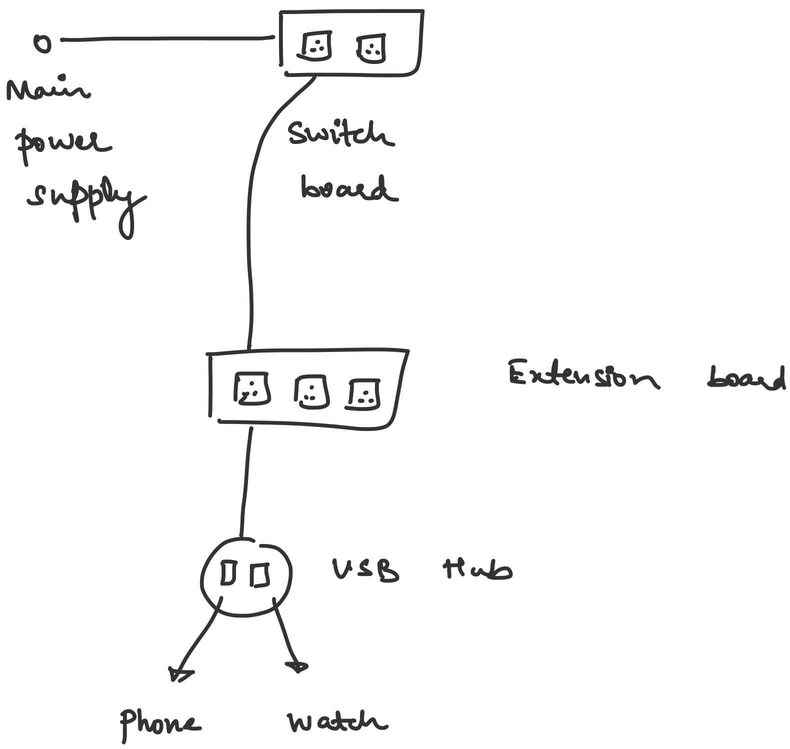
Social media connection



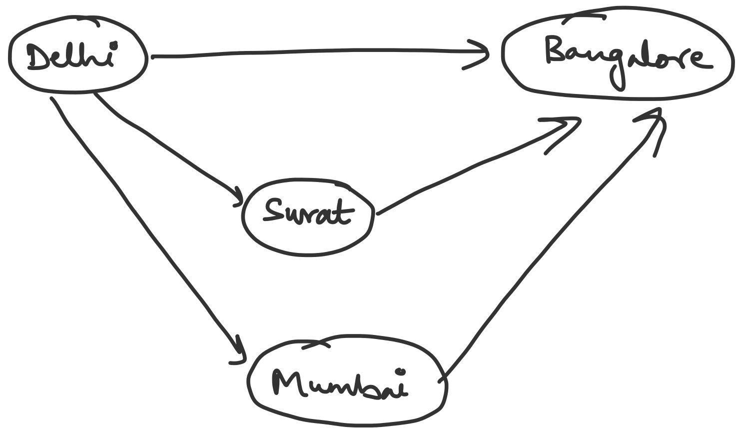
connection of computers



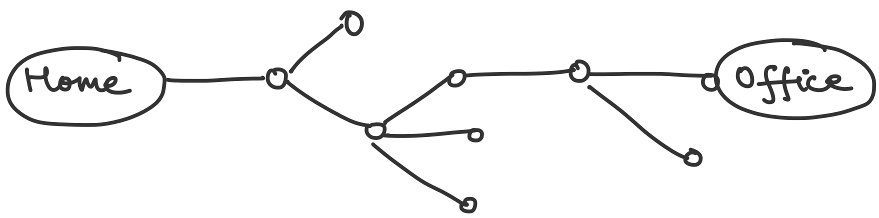
Electrical circuit



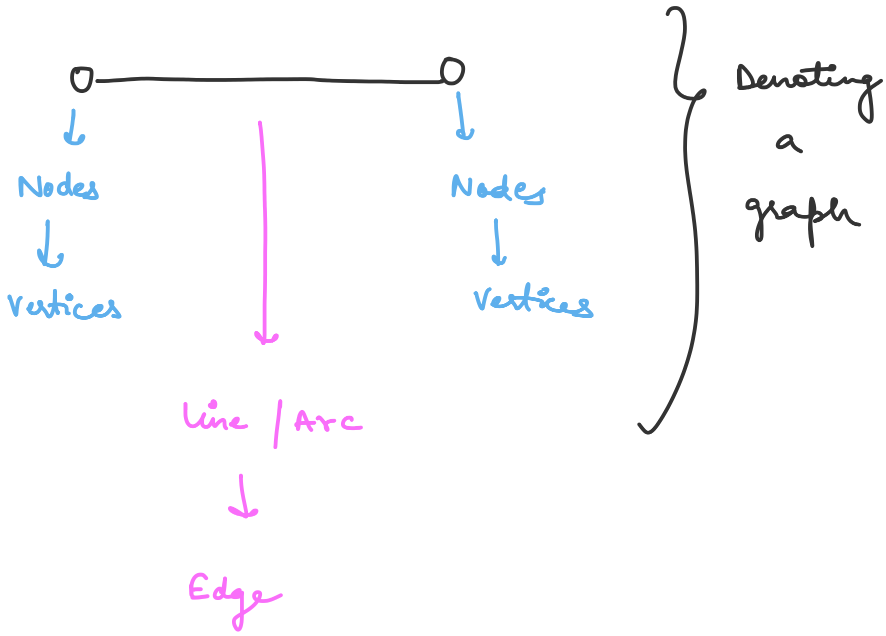
Flight paths



Maps



A graph is a non linear data structure consisting of vertices and edges.



A graph will have set of vertices (V) and set of edges (E). A graph can be denoted G (V, E).

Components of a graph –

1. Vertices (V)
   ↓
   can also be known as nodes
   ↓
   can be labelled | unlabelled.

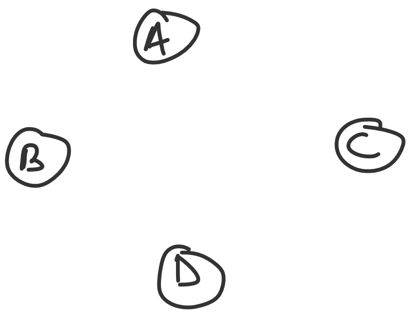
2. Edges (E)
   ↓
   line | Arc
   ↓
   can be labelled | unlabelled

Types of graphs –

1. Null graph

↓

In this there will be no edges in the graph.



2. Trivial graph

↓

Graph having single vertex

↓

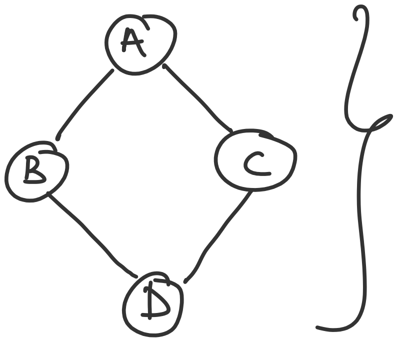
Also known as smallest graph possible.

3. Undirected graph

↓

A graph in which edges will not

have any direction.

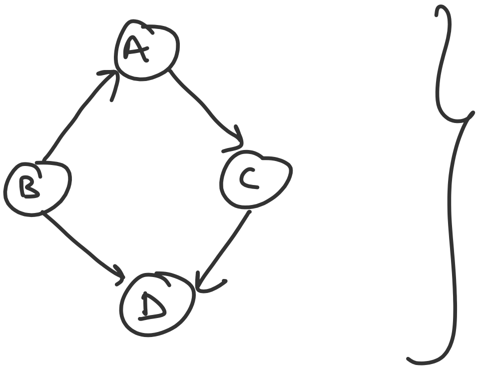


nodes are unordered pairs

4. Directed graph

↓

In this edges will have direction.

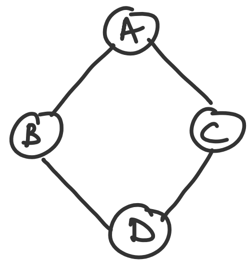


nodes are ordered pairs

5. Connected graph

↓

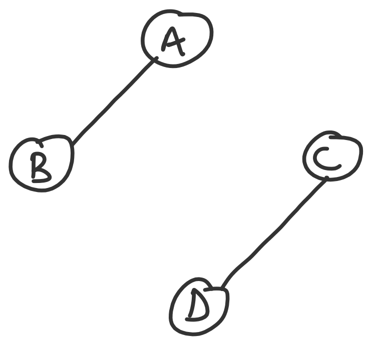
A graph in which from one node you can go to any other node.



6. Disconnected graph

↓

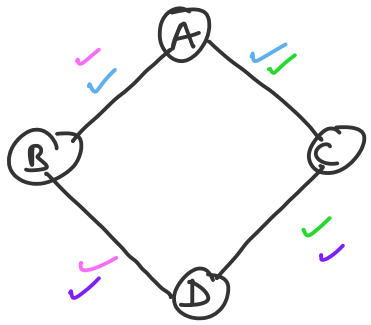
A graph in which atleast one node is not reachable from a node.



7. Regular graph

↓

The graph in which the degree of every vertex is equal to k is called as a k-regular graph.



Degree of vertex A = 2

" " " B = 2

" " " C = 2

" " " D = 2

$\underbrace{\hspace{6cm}}$

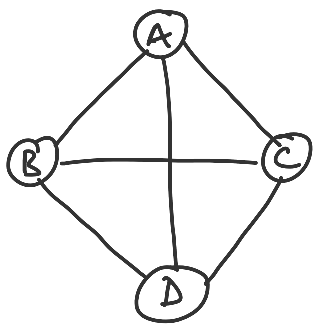
2- Regular graph

8. Complete graph

↓

In this from each node there is an

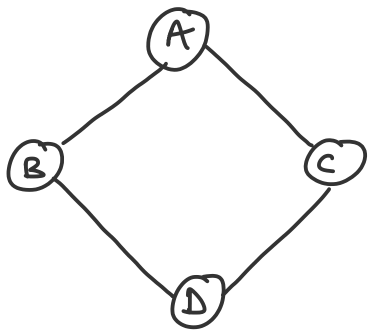
edge to each other node.



9. Cycle graph

↓

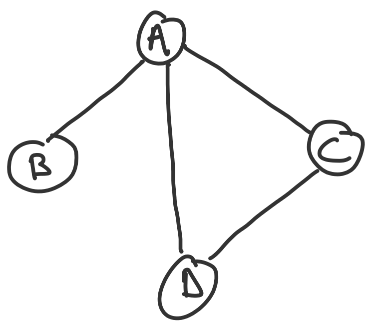
These graphs where degree of every vertex is 2.



10. Cyclic graph

↓

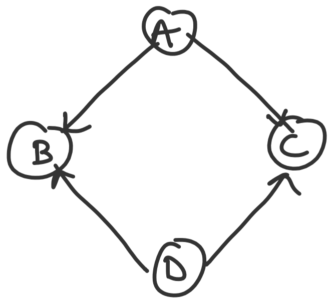
A graph consisting of atleast one cycle.



11. Directed acyclic graph

↓

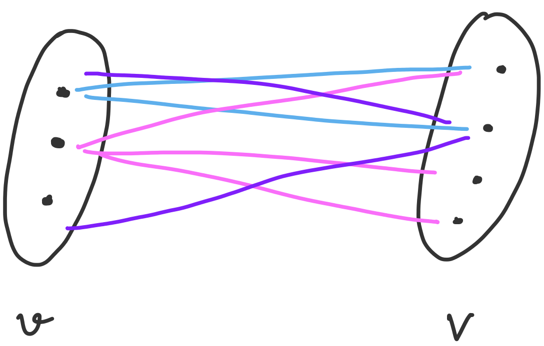
Directed graph having no cycles.



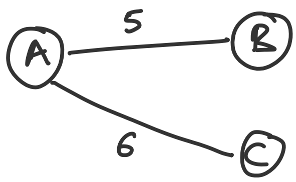
12. Bipartite graph

↓

A graph in which we will be dividing vertices into two sets such that vertex in one set will not be containing edges within the same set.



13. weighted graph



A graph in which edges are specified with a suitable weight.

weighted graph

- Undirected weighted graph
- Directed weighted graph

Tree vs graph

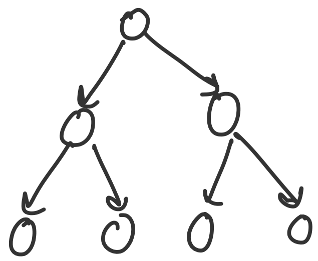
↓

Trees are restricted types of graph, just with some rules.

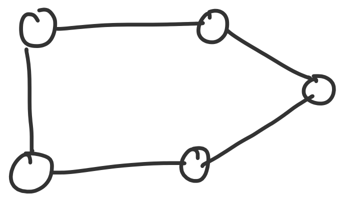
Every tree will always be a graph but not all graphs will be trees.

linked list, heaps and trees are special cases of graphs.

Tree



Graph



Representation of graphs

There are 2 ways to store graphs —

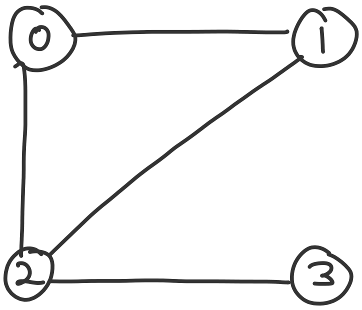
→ Adjacency matrix

→ Adjacency list

Adjacency matrix

↓

Graph is represented in the form of a 2D matrix where rows and columns denote vertices.

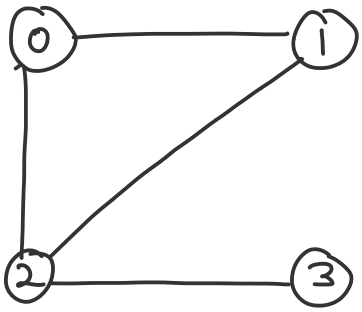


	0	1	2	3
0	0	1	1	0
1	1	0	1	0
2	1	1	0	1
3	0	0	1	0

Adjacency list

↓

In this graph is represented as collection of linked list.



0 → 1 → 2

1 → 0 → 2

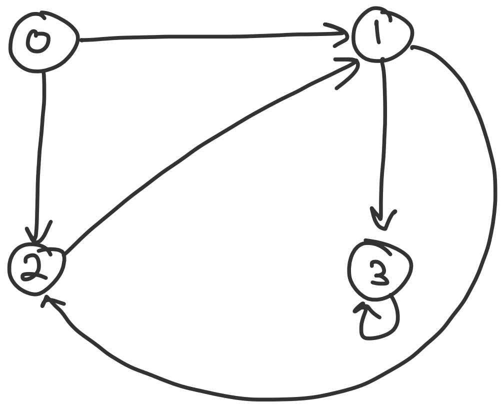
2 → 0 → 1 → 3

3 → 2

Basic operations on graph

1. Insertion of a node | edge in the graph.
2. Deletion of a node | edge in the graph.
3. Search on graph
4. Traversal of graph.

DFS (Depth first search) of a graph



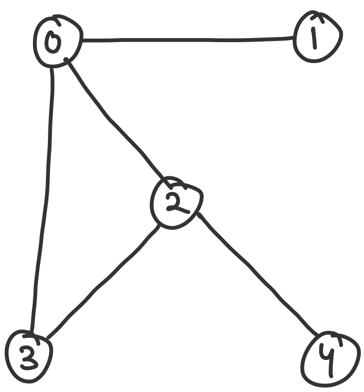
Total no. of vertices = 4

" " " edges = 6

$$\left[\begin{array}{ccc} 0 \to 1, & 1 \to 3, & 3 \to 3, \\ 1 \to 2, & 0 \to 2, & 2 \to 1 \end{array}\right]$$

DFS = 0, 1, 2, 3

Eg-



0	1	2	3	4
T	T	T	T	T

visited

Stack

1	2	3	

$\boxed{0 \to 1 \to 2 \to 4 \to 3}$ ✓

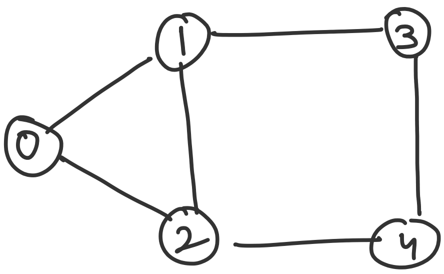
$\boxed{0}$ → Already visit → skip

4	3

2

BFS (Breadth first traversal) of a graph

Eg-



visited

0	1	2	3	4
T	T	T	T	T

Queue

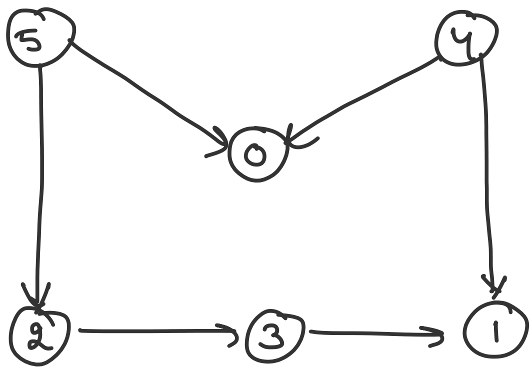
Empty

$\underbrace{0 \to 1 \to 2 \to 3 \to 4}_{\text{BFS}}$

Topological sorting

↓

Linear ordering of vertices such that if there is an edge 'u → v' where 'u' appears before 'v' in the ordering.



Eg- 5 4 3 2 1 0

$u \to v$

$4 \to 0$ ✓

$2 \to 3$ ✓

$5 \to 0$ ✓

$4 \to 1$ ✓

Note -

Topological sorting is only possible for

DAG - Directed Acyclic Graph

$\boxed{u \to v}$

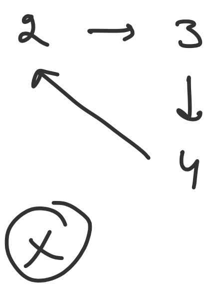
for this relation we need a

directed graph

$$\left[\begin{array}{l} 2 \rightarrow 3 \\ 3 \rightarrow 4 \\ 4 \rightarrow 2 \end{array}\right]$$

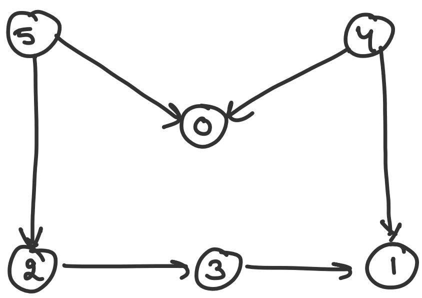
↓

Dependency



using DFS

```
for (i = 0 to 5)
{
    if (!visited[i])
    {
        DFS(i)
    }
}
```



Adjacent nodes

0 → x
1 → x
2 → 3
3 → 1
4 → 0, 1
5 → 0, 2

visited

0	1	2	3	4	5
~~F~~	~~F~~	~~F~~	~~F~~	~~F~~	~~F~~
T	T	T	T	T	T

Stack

0	1	3	2	4	5

i=0

DFS(0) → No adj nodes
↓
Mark it as done

i=1

DFS(1) → No adj nodes
↓
Mark it as done

i=2

⇒ DFS(2) → Mark as done
↓
DFS(3) → Mark as done
↓
DFS(1) → Already visited

i=3

↳ Already visited

i=4

DFS(4) → Mark done
↓
$\underbrace{\text{DFS(0) , DFS(1)}}_{\text{Already visited}}$

$i = 5$

DFS(5) → Mark done

↓

DFS(0) , DFS(2)

Already visited

Stack

0	1	3	2	4	5

↓

pop off all elements

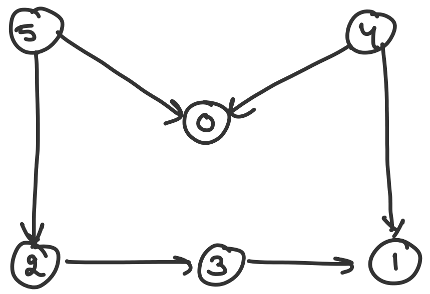
↓

$$\underbrace{5,\ 4,\ 2,\ 3,\ 1,\ 0}_{\text{Topological ordering}}$$

$$T.C. = O(E+V)$$

$$S.C. = O(n)$$

For BFS



Adjacent nodes

0 → X
1 → X
2 → 3
3 → 1
4 → 0, 1
5 → 0, 2

Queue					
~~4~~	~~5~~	~~0~~	~~2~~	~~3~~	~~1~~

	0	1	2	3	4	5
In-degree	2	2	1	1	0	0

[Sorting = 4, 5, 0, 2, 3, 1]

↓

Number of edges coming to that vertex

Insert the nodes with 0 in-degree to the queue

	0	1	2	3	4	5
In-degree	~~2~~	~~2~~	~~1~~	~~1~~	0	0
	~~1~~	~~1~~	0	0		
	0	0				

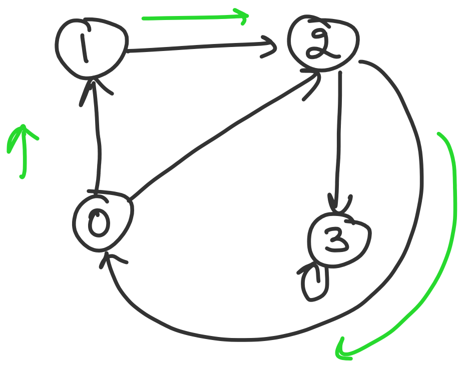
Reduce in degree by 1 for the adjacent nodes

Detect cycle in a directed graph

↓

Given root of the graph, check whether the graph contains cycle or not.

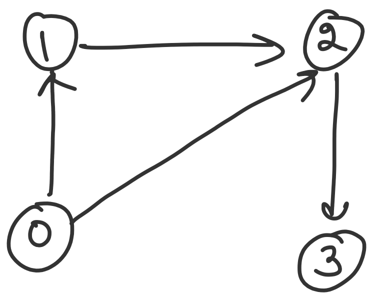
Eg—



Detects a cycle

$n = 4$ $e = 6$

Another eg—



Does not contain a cycle

$n = 4$ $e = 4$

To detect a cycle in a graph we will use DFS technique based upon the idea that a graph will have cycle if it contains back edge in graph.

↓

That node which points to previous ancestor

Steps –

1. Create a recursive dfs function which will have parameters – current node, visited array and recursion stack.

2. Mark the current node as visited and also mark the index in the recursion stack.

3. Iterate a loop for all the vertices, and for each vertex call a recursive

function to check if the node is already visited or not. —

(i) In the recursive call, find the adjacent vertex which is not yet visited —

If the adjacent vertex is already marked then return true.

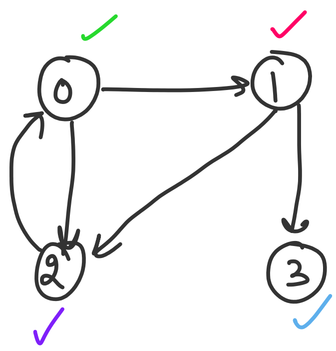
Otherwise call a recursive function on that adjacent vertex.

(ii) while returning from the recursion call, unmark the current node to represent current node is not a part of the tracing path.

4. If any of the function call returns true then stop all the future

function calls and return true.

Eg -



visited

0	1	2	3
~~F~~ T	~~F~~ T	~~F~~ T	F

Rec stack

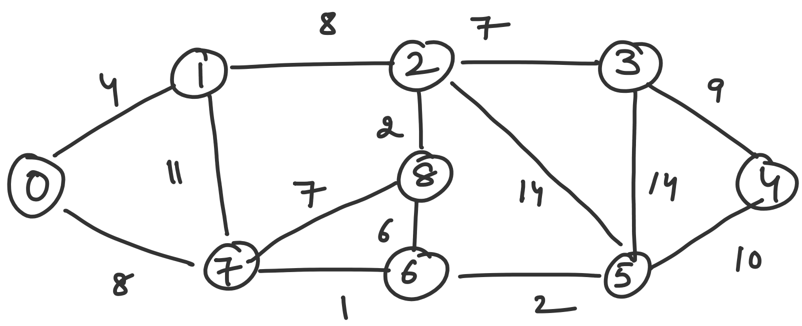
0	1	2	3
~~F~~ T	~~F~~ T	~~F~~ T	F

Adj nodes

0 → 1, 2
1 → 2, 3
2 → 0
3 → X

0 → Already visited
↓
cycle detected

Dijkstra algorithm
↓
Given a graph and a source vertex in the graph, find shortest path from source to all the vertices in the given graph.



Source = 0

Distance from 0 to 1 = 4 (0 → 1)

" " 0 to 2 = 12 (0 → 1 → 2)

" " 0 to 3 = 19 (0 → 1 → 2 → 3)

" " 0 to 4 = 21 (0 → 7 → 6 → 5 → 4)

" " 0 to 5 = 11 (0 → 7 → 6 → 5)

" " 0 to 6 = 9 (0 → 7 → 6)

" " 0 to 7 = 8 (0 → 7)

" " 0 to 8 = 14 (0 → 1 → 2 → 8)

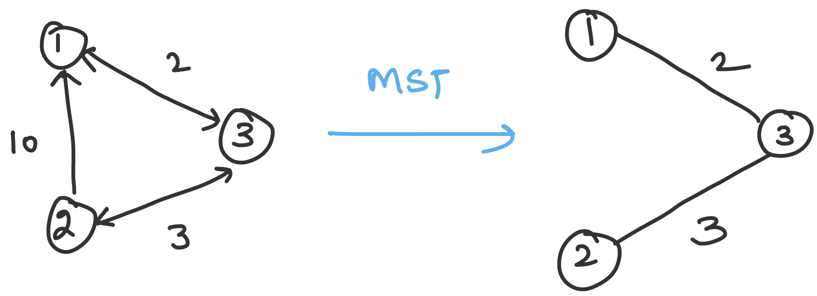
<u>output</u>

0 4 12 19 21 11 9 8 14

Minimum Spanning Tree (MST)

↓

MST is a spanning tree that has minimum weight among all the spanning trees.



MST

Kruskal's Algo   Prim's Algo

Kruskal's Algo

↓

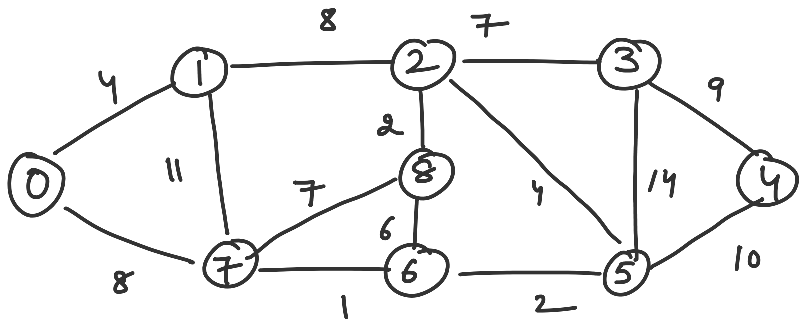
used to find MST of a weighted graph.

<u>Steps</u> –

1. Sort all the edges in the increasing order of their weights.

2. Pick the smallest edge, and then check if it forms a cycle or not. If it forms a cycle then discard it, otherwise include.

3. Repeat step no 2 unless we have $(V-1)$ edges in the spanning tree.

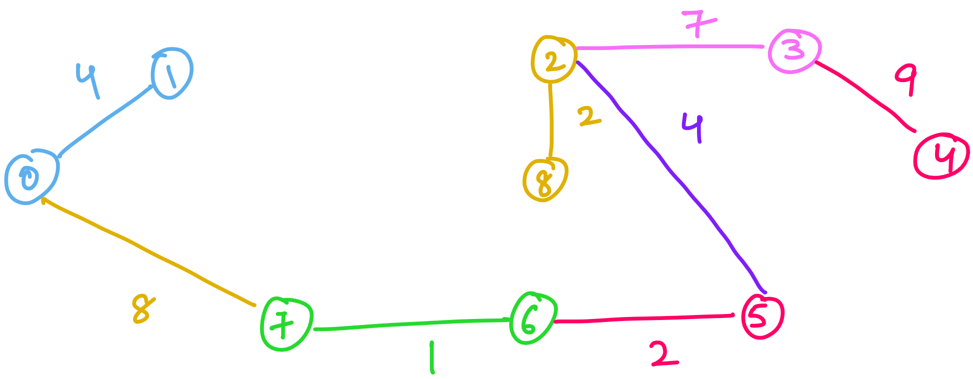


$n = 9$ $e = 14$

No of edges in MST $= (V-1)$

$= 9-1 = 8$ edges

weight	source	destination	
1	7	6	✓
2	8	2	✓
2	6	5	✓
4	0	1	✓
4	2	5	✓
6	8	6	× cycle
7	2	3	✓
7	7	8	× cycle
8	0	7	✓
8	1	2	× cycle
9	3	4	✓
10	5	4	× cycle
11	1	7	× cycle
14	3	5	× cycle



Cost of MST $= 4 + 8 + 1 + 2 + 4 + 2$

$+ 7 + 9$

$= \underline{\underline{37}}$

Prim's Algorithm

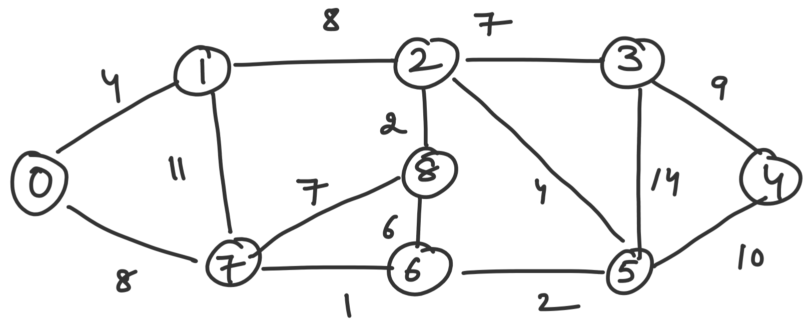
↓

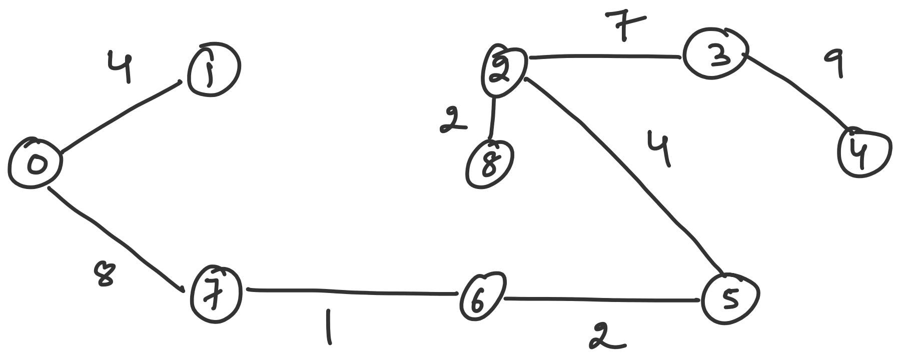
used to find MST of a directed graph.

Steps –

1. Determine a vertex as the starting vertex.
2. Find edge connecting to the other vertices.
3. Find the minimum edge out of all options.
4. Add the chosen edge in the MST, also determine that it does not form a cycle, if it forms a cycle then discard it, otherwise include it.
5. Repeat steps 2 – 4 till there are all the vertices included in the MST.

6. Return the MST and exit.





cost of MST = 4 + 8 + 1 + 2 + 4 + 2 +

7 + 9

= $\underline{\underline{37}}$