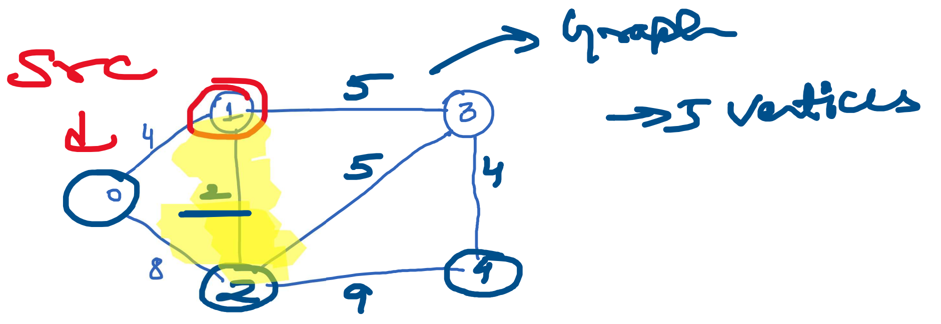
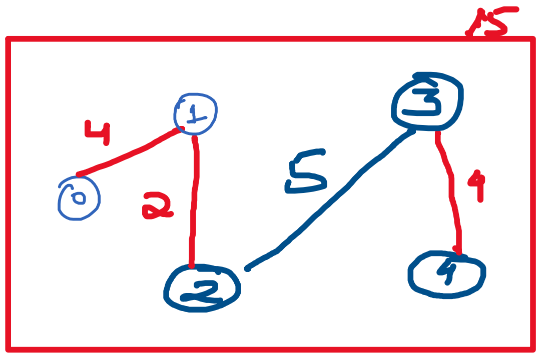
June 23 Graphs

23 June 2020 19:10

1. Prim's Algorithm
2. Detect Cycle in an Un-Directed Graph
3. Detect Cycle in a directed Graph

MST → Minimum Spanning Tree



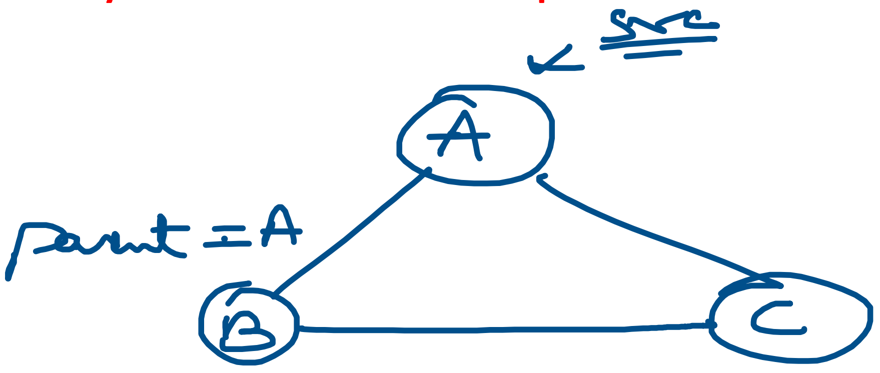


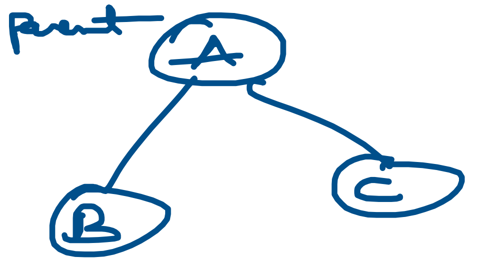
Every element of a tree has a single parent

0	1	2	3	4	
T	T	T	T	F	Visited
0	4	2	5	4	Distance
-1	0	1	1	3	Parent

0 1 4
0 2 8
1 3 5
1 2 2
2 3 5
2 4 9
3 4 4

2. Detect Cycle in an Un-Directed Graph





A	B	C
T	T	T

A	B	C
T	T	

Ek aisa neighbour hai jo ki visited hai but parent nahi hai usko matlab cycle exists.

For every visited vertex "V", if there exists an adjacent vertex "U" such that
U is already visited and U is not a parent of V, then cycle exists in graph.

```
//For Un-Directed Graphs
class DetectCycle
{
   //Adjacency List representation.
   static boolean isCyclic(ArrayList<ArrayList<Integer>> g, int V)
   {
      boolean visited[] = new boolean[V];
      for(int i=0;i<V;i++)
      {
```

MATHEMATICS → 11

```
        if(!visited[i])
        {
          if(isCyclicUtil(g,i,visited,-1)) return true;
        }
      }
      return false;
    }
    static boolean isCyclicUtil(ArrayList<ArrayList<Integer>> g,int v,boolean visited[],int parent)
    {
      visited[v] = true;
      for(Integer x:g.get(v))
      {
        if(!visited[x])
        {
          if(isCyclicUtil(g,x,visited,v)) return true;
        }
        else{
          if(x!=parent) return true;
        }
      }
      return false;
    }
  }
```

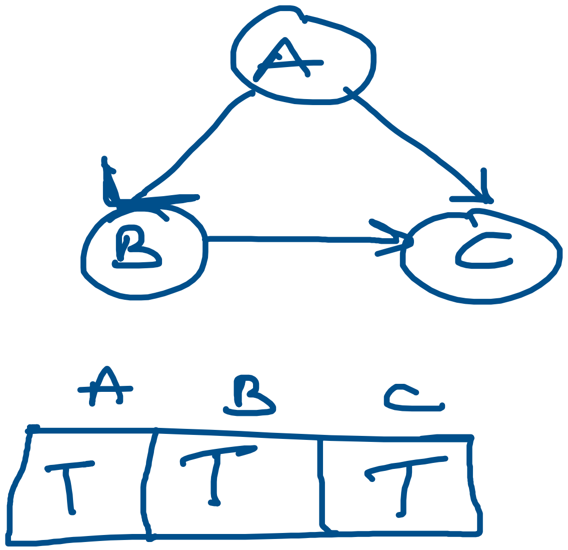
M4S

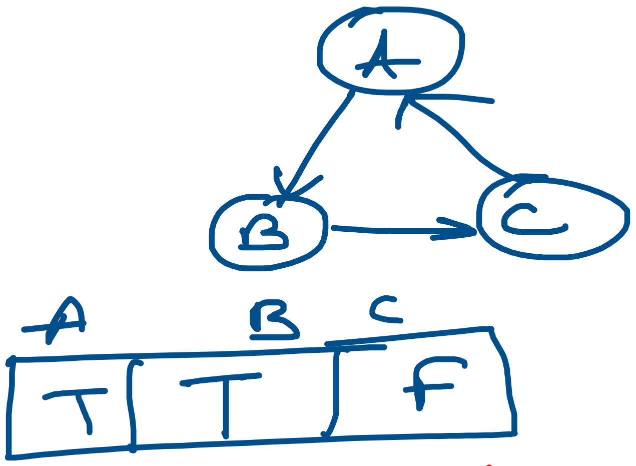
S.charAt(0)

S.length() - 2

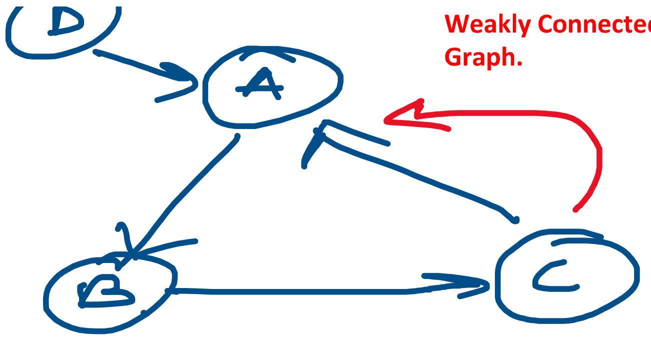
S.charAt(S.length()-1)

3. Detect Cycle in a Directed Graph





~~If there is a neighbour that is already visited means cycle - WRONG~~



Weakly Connected Graph.

Visited →

A	B	C	D
T	T	T	F

→ Our algo says cycle

currPath →

A	B	C	D
T	T	T	F

↳ Hence cycle

If there is a neighbour that is already there in the current path then there is a cycle - Correct

```
//For Directed Graphs
class DetectCycle
{
   static boolean isCyclic(ArrayList<ArrayList<Integer>> g, int V)
   {
      boolean visited[] = new boolean[V];
      boolean currentPath[] = new boolean[V];
      for(int i=0;i<V;i++)
      {
         if(!visited[i])
         {
            if(isCyclicUtil(g,visited,currentPath,i)) return true;;
         }
      }
      return false;
```

```
    }
    static boolean isCyclicUtil(ArrayList<ArrayList<Integer>> g, boolean visited[], boolean currentPath[],
int v)
    {
        visited[v] = true;
        currentPath[v] = true;
        for(Integer x:g.get(v))
        {
            if(!visited[x])
            {
                if(isCyclicUtil(g,visited,currentPath,x))  return true;
            }
            else if(currentPath[x]==true) return true;
        }
        //all neighbors traversed hence remove from current path
        currentPath[v] = false;
        return false;
    }
}
```