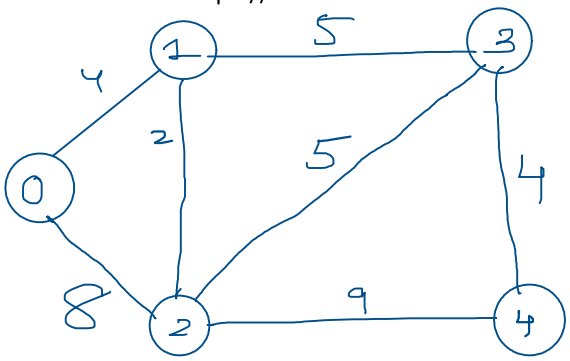
# MST & Cycle, August 07

07 August 2020 19:08

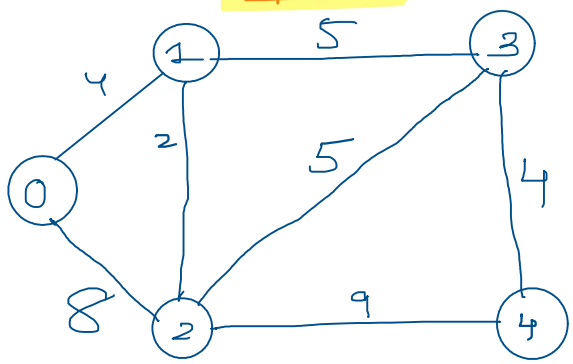
1. MST //Done
2. Kruskals - Theory Only, Implementation - HW //Done
3. Prims //Done
4. Detect Cycle in an undirected Graph //Done
5. Detect Cycle in an directed Graph //Done

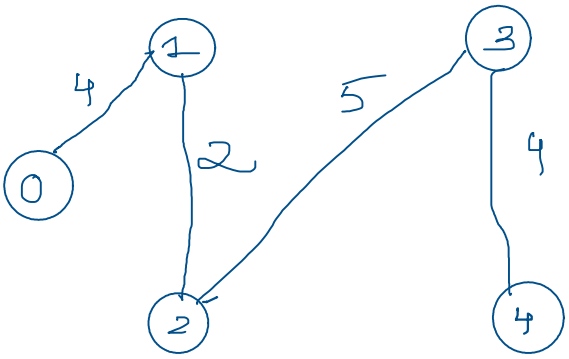
Graph




MST → MINIMUM SPANNING TREE → n Nodes, Edges

GRAPH




Kruskals Algorithm

1. Edge list with weights
   → Sort in Ascending order based on weights

Interface Comparable/ Comparator

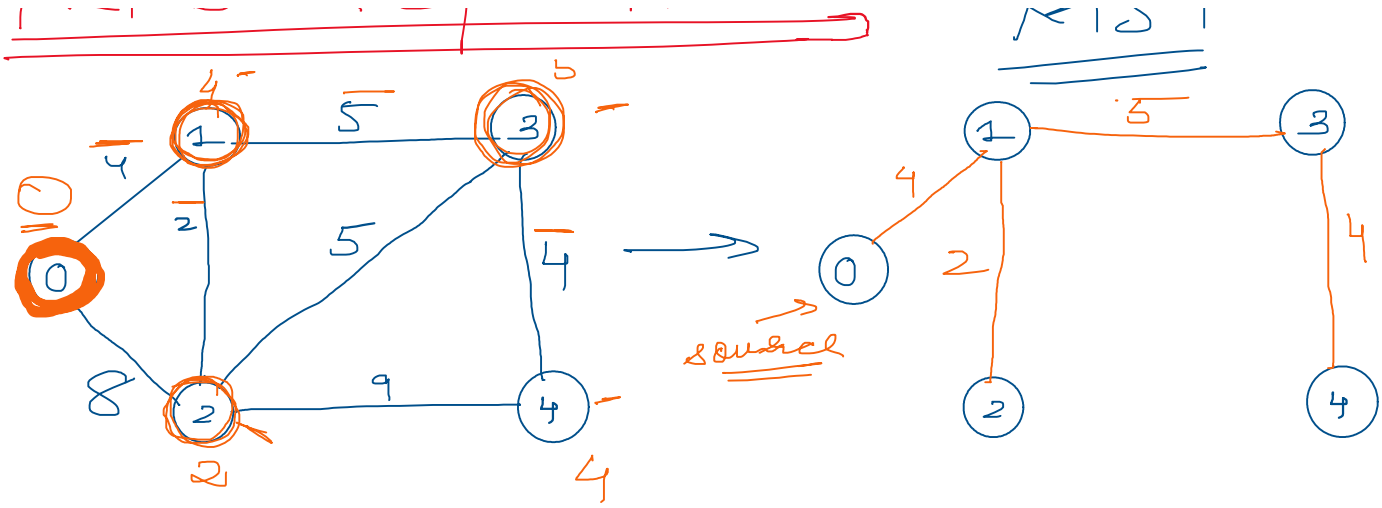
Rea

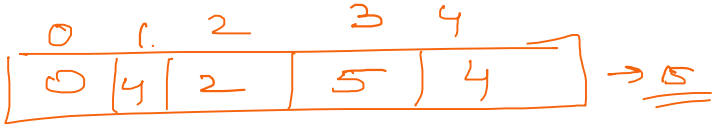
Union

2. Start picking edges from sorted list
   → Add to MST if no cycle creat
   → Stop when n-1 edges added.

Implement this

PRIMS ALGORITHM

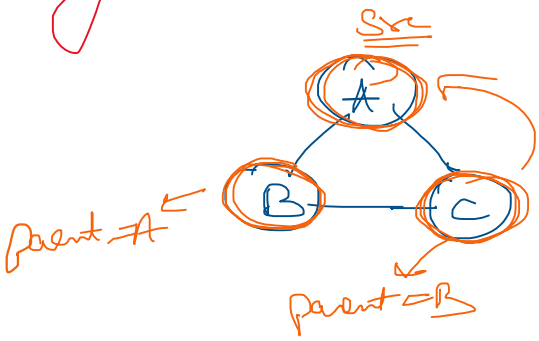


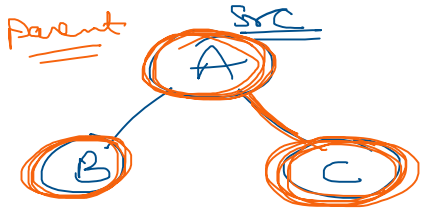




```
5
7
0 1 4
0 2 8
1 2 2
1 3 5
2 3 5
2 4 9
3 4 4
```

## Detect Cycle in an Un-Directed Graph

DFS

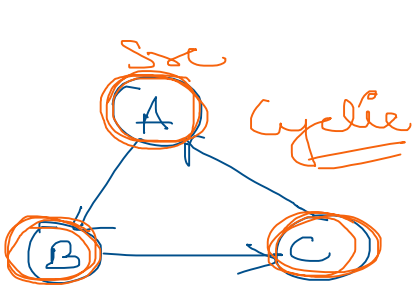


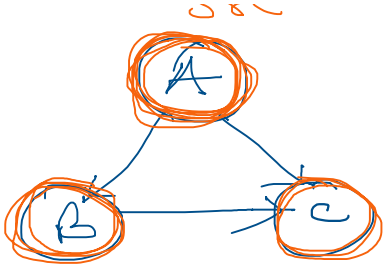



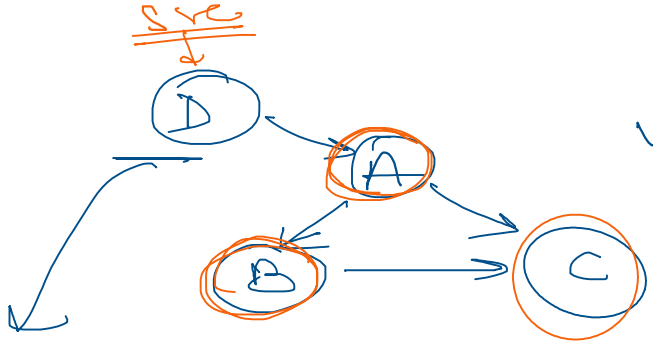

Ek aisa neighbour jo pr visited bhi hai aur parent bhi nahi h iska matlab cycle exists.

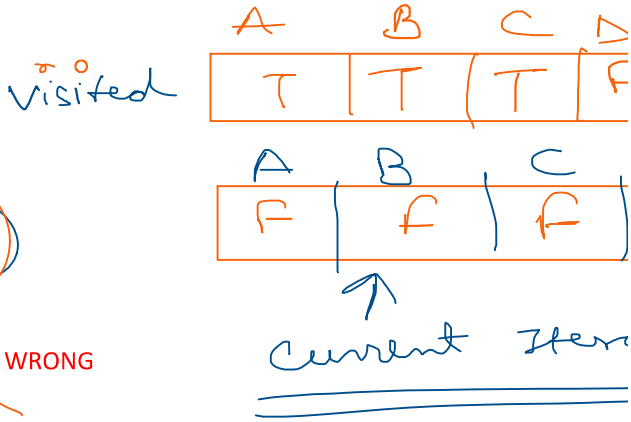
For every visited vertex "V", if there exists an adjacent vertex(neighbour) "U" such that U is already visited and U is not the parent of V, this means cycle exits.

## DETECT CYCLE IN A DIRECTED GRAPH

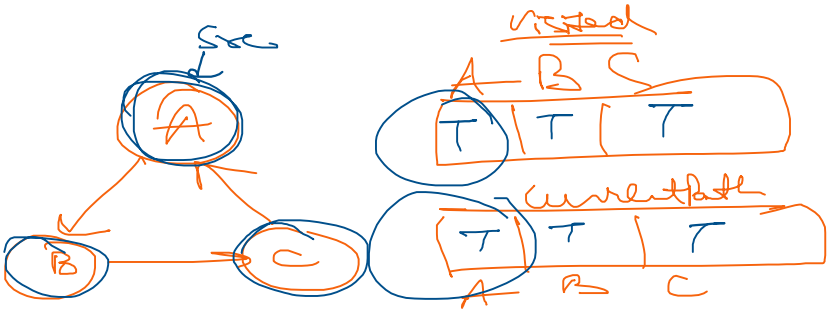



In this scenario:-
If there is a neighbour that is visited and not Parent means cycle - WRONG




```
//Undirected Graph

class DetectCycle
{
   static boolean isCyclic(ArrayList<ArrayList<Integer>> g, int V)
   {
      // add your code here
      boolean visited[] = new boolean[V];
      for(int i=0;i<V;i++)
      {
         if(!visited[i])
         {
            if(isCyclicUtil(g,i, visited, -1)) return true;
         }
      }
      return false;
   }
   static boolean isCyclicUtil(ArrayList<ArrayList<Integer>> g,int src,
                               boolean visited[], int parent)
   {
      visited[src] = true;
      for(Integer x:g.get(src)) //Getting neighbours of x
      {
         if(!visited[x])
         {
            if(isCyclicUtil(g,x,visited,src)) return true;
         }
         else //Neighbour already visited
         {
            if(x!=parent) return true; //Neighbour not parent means cycle
         }
      }
      return false;
   }
}
```

```
//Directed Graph
class DetectCycle
{
   static boolean isCyclic(ArrayList<ArrayList<Integer>> g, int V)
   {
      // add your code here
      boolean visited[] = new boolean[V];
      boolean currentPath[] = new boolean[V];
      for(int i=0;i<V;i++)
      {
         if(!visited[i])
         {
            if(isCyclicUtil(g, i, visited, currentPath)) return true;
         }
      }
      return false;
   }
   static boolean isCyclicUtil(ArrayList<ArrayList<Integer>> g, int src,
                               boolean visited[], boolean currentPath[] )
   {
      visited[src] = true;
      currentPath[src] = true;
      for(Integer x:g.get(src))
      {
         if(!visited[x])
         {
            if(isCyclicUtil(g, x, visited, currentPath)) return true;
         }
         else //Neih=ghbour is visited
         {
            if(currentPath[x]==true) //Visited & in currentPath
            {
               return true;
            }
         }
      }
      currentPath[src] = false;
      return false;
   }
}
```