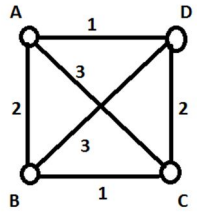
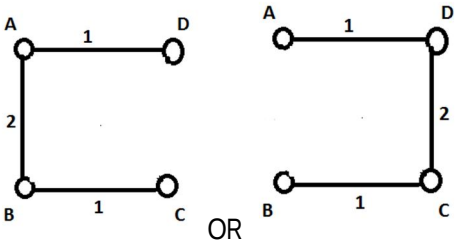
CONTINUE WORKING WITH PRIM'S ALGORITHM ::

We are using Prim's Algo as an measure to find out a tree consisting of all the vertices out of a given graph with the minimum total weight.

For Example Let's take a Graph as ->



Now, we can see that the tree with total Minimum weight that can be found are ->



Both of them with total weights = 4 .

Now, Let us try and find out the result with the help of Prim's Algorithm :

The following steps will be required::

1) Start with any vertex . For example pick A as initial vertex for the tree.
2) Go through the adjacency list of A in order to see all the edges connected to it as candidates and picking out the minimum weighted edge out of them.
3) The process of step 2 requires the use of creating heap based on the adjacency list of vertex and then heapifying it every time based on weights , we pick out a minimum value out of it or we go on adding new edges as leaves at its bottom. **This is the same process we have done in 8 March class please see its notes for detailed process of adding,picking out minimum and heapifying.
4) Now , every time you pick out the minimum weighted edge from the heap,you are required to mark the status corresponding to its vertices in isOutput column of the table. The isOutput column will then help in checking for the edge picked out as minimum from the heap as if its both the ends are not previously in the output tree or not. If both ends are marked in isOutput then we need to reject that edge as then it will create a cycle or loop and if both ends are not present in isOutput marked table then choose that edge as minimum and add it to the output tree and mark its corresponding vertices in isOutput column.

5) Now once you have picked an edge with the least weight joining initial vertex to some next vertex ,After that go through the adjacency list of next vertex and add the edges to the heap at bottom and heapify the heap based on the weights of edges . You also need to consider the condition as discussed in step 4 means you will not be adding edges whose both ends are previously in the isOutput column.
6) Since we are using a heap and continuously heapifying it, Everytime we add an edge to it we are keeping track of the shortest edge connecting the vertices.

Following these steps lets look on our graph::

Create the following table

Vertex **isOutput**

A

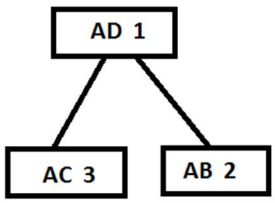
B

C

D

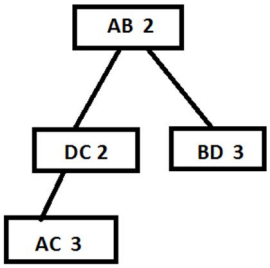
Now we are going to start with A so mark A and add all its connecting edges from its adjacency list to heap and then heapify it:

AD 1 , AC 3 , AB 2 ->

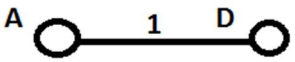


Take out min^M AD and mark D, next go to adjacency list of D and add all its connecting edges from its adjacency list to heap and then heapify it:

DA 1 can't add both A & D in isOutput marked , DC 2 , BD 3 ->



The ouput tree:

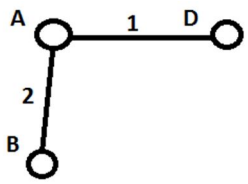


The Table after this will look like:

Vertex	**isOutput**
A	yes
B	
C	
D	yes

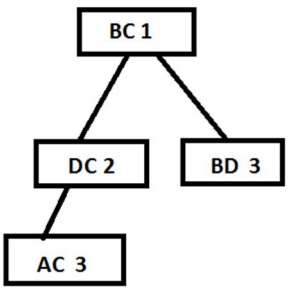
Take out min^M AB and mark B, next go to adjacency list of B and add all its connecting edges from its adjacency list to heap and then heapify it :

Output tree:



Vertex	**isOutput**
A	yes
B	yes
C	
D	yes

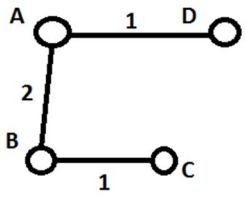
For B - > AB 2 can't add , BC 1 , BD 3 can't add:



Pick out BC and mark C:

Vertex	**isOutput**
A	yes
B	yes
C	yes
D	yes

Output tree:



Now , Since we have used minimum V-1 no. of edges and marked all the vertices this is the final output tree same as expected beforehand with minimum weight as 4.

Time Comlexity analysis for this Algorithm::

Time for creating Table = **O(V).**

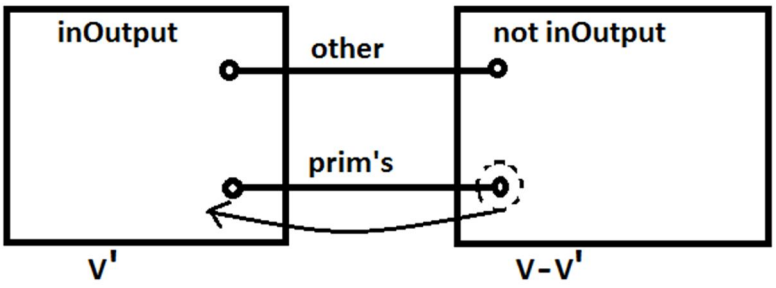
Time for number of times heapifying = **O(E*logE).** Derived as no. of edges are **O(degree of edges)** and this is done V times with heapifying taking **O(logE)** every time.

Time for number of removals = **O(V * logE)** since there are atmost V removals which can be maximum written upto order of **O(E*logE).**

Thus total time complexity can be given as O(E*logE).

To work out the correctness of Algorithm and saying that it is going to provide the optimal solution:

1) This will give one of the optimal solution tree with the minimum total weight from the set of optimal solutions.
2) For a graph with only 1 edge it will be true and give the basis condition.
3) The strategy of Prim's Algorithm work in a manner that given a set of edges in Output we are going to select an edge based on vertex which is not currently in output with the minimum weight and then push it into output for making edge of the output tree. If there is some other vertex possible given by some other algorithm then too we can say that these two vertices should be same and only one should be present otherwise there will be a cycle or it should be a different edge having the same weight.



Working with Kruskal's Algorithm :

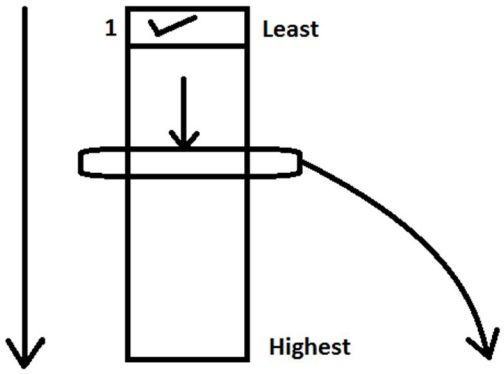
It is also based on the theory given V vertices we need to choose minimum V-1 no. of edges and create an output choosing the edge with the minimum weight and adding the output to list of outputs if the generated output doesn't have a cycle .

Checking the condition that it will always give the solution with minimum total weight::

First it will sort the Edges in sorted order of their weights from least to the highest.

For one element it will always choose the edge with the least weight and thus it works for graph with a single edge.

And for more than one edge if we go on adding edges to the tree with going down in the sorted array and

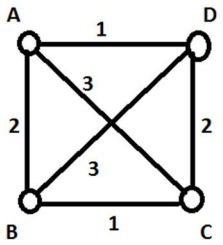


If this edge creates a cycle then we need to remove it or if we want to include it then we are required to remove some edge/edges above it so that it won't make any cycle but then we are removing edges with less weight with those of higher weight so it will result in more weighted output . Thus bringing in this edge we need to remove an edge with an equivalent weight to it.

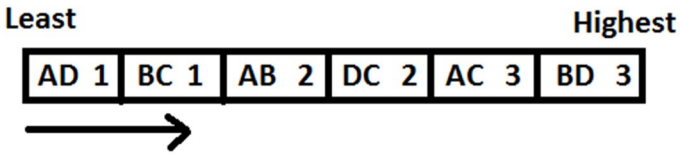
Thereby Kruskal's Algorithm will give minimum weight.

Understanding Kruskal's Algorithm:

Let's take the same example again :



This time make a sorted array according to weight of edges :



Start with picking the first element of the array

Table::

Vertex	isOutput	Component
A		1
B		2
C		3
D		4

Pick AD as minimum and mark A & D in isOutput column.

Vertex	isOutput	Component
A	yes	1
B		2
C		3
D	yes	1

NOTE::Now this time, we are also requiring an additional column as "component" which will mark all the connected items as one single component and everytime we will connect two vertices we will follow that we will make the value of higher component field of others connected to same as that

of lower one. This field will help us in decision making for selecting edges at that time when both the vertices connected to it are already in isOutput marked but the output tree is not entirely connected. Connect the vertices if their component values are different otherwise not.

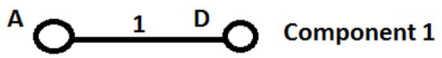
So make component field value of D same as that of A = 1.

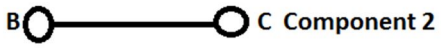
So , next step pick BC as minimum and mark B & C in isOutput column.

Make component field value of C same as that of B = 2.

Vertex	**isOutput**	**Component**
A	yes	1
B	yes	2
C	yes	2
D	yes	1

TREE::



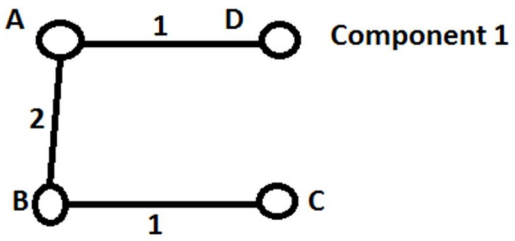


Next pick AB as minimum (**NOTE::** we can pick it even if A & B are marked in isOutput field because A & B have different **component** values means the output tree is not entirely connected.)

Make component values of all the connected fields same as that of lower one as of A = 1. Now , we have all the component values as same so we have a connected tree with minimum total weighted edges.

Vertex	**isOutput**	**Component**
A	yes	1
B	yes	1
C	yes	1
D	yes	1

Tree::



We are using the components as if they are same for an edge then it will create an cycle and if not then we will add it.

When making a component change from higher to lower value currently we are making a linear search for all the same higher values and replacing them with the required lower value.

Time Complexity analysis::

Time required for sorting : Min^m = **O(E*logE).**

Time to update components(**O(V)**)*no. of times(V) = **$O(V^2)$**

Constant time for rejecting an edge if both its ends are marked and component value is the same * maximum E times = **O(E)** .

Total = **$O(E*logE + V^2 + E)$**

So current total time complexity is **O(E*logE) .** In this the only asymptotic time complexity that can be reduced is no. of times we update components.

We will try to reduce its value further to Log(V) or Constant in next class...

Created and submitted by::

Rajat Kulshreshtha

11010846

Dated on 11th March 2013