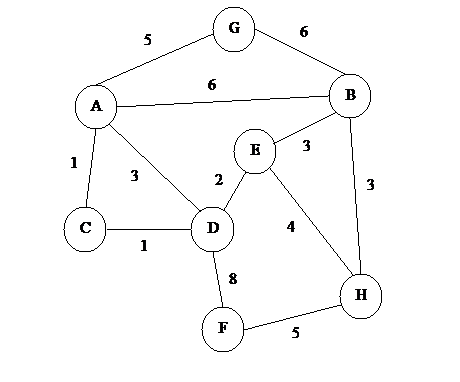
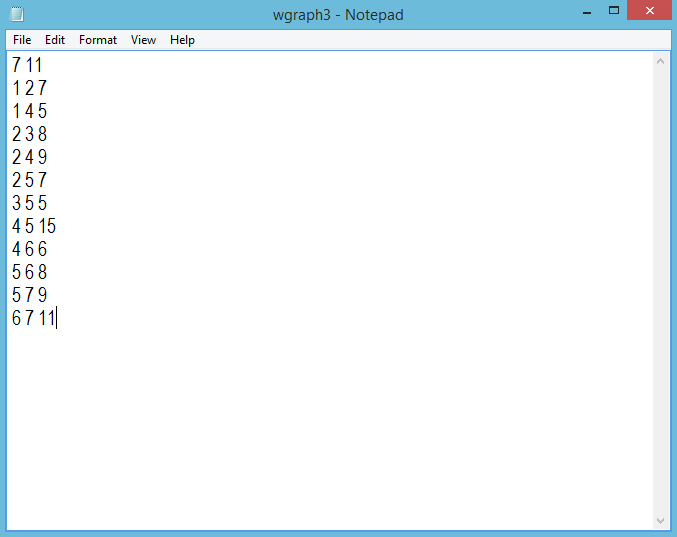
**Algorithms and Data Structures**

**Assignment**

Student: Maximilian Adrian Mihoc

No: C12728559

For this Assignment I choose to do both Kruskal-Trees and Prim’s Algorithm for finding Minimum Spanning Tree for a connected weighted graph. For the explanation of them I will use the following graph.

This graph is stored in a text file as follow:

# Kruskal-Trees Algorithm

For a better explanation of how my implementation of Kruskal Algorithm works, I will explain the main things of each class. My algorithm contains four classes: Edge, Heap, UnionFindSets and Graph classes.

**Edge Class**

This class has 3 fields that are used to store the starting point (u), the end point (v) and the weight (wgt) of an edge. This class will be used to store all necessary things of an edge. When the graph will be later stored in my program, it will be stored in an array of edges.

**Heap Class**

This heap class is containing a constructor, a Sift Down method and a remove method. I also had a display method for the heap that is commented out. I used this method to see the content of the heap so I could fix all my errors related to it.

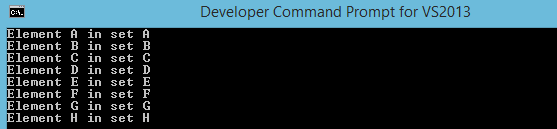
The constructor takes 2 parameters, the length of the array that has to be converted into a heap and the array of edges. It is also concerting the h array into heap using the Sift Down method.

The Sift Down method is much more complicated than the others. In order for this algorithm to work, in this method I put the code that converts an array into a heap. In my hep, I am storing the number of each edge but when this numbers are stored, they are arranged into the heap using the weight of the edge. When I wrote the code for this algorithm, this was the hardest thing that I had to do. After I found out that the main thing of this method is to store the edge numbers into the heat taking into consideration the weight of each edge, the rest of the code was much easier.

The remove method is used to remove the first element from the heap, take the last element to replace it and call the Sift Down method to make the heap as it should be.

**UnionFindSets Class**

This class contains a treeParent array, N (length of that array) a constructor and some other methods. The main thing of this class is to join 2 sets of vertices if one vertex from one set is connected to one vertex from the other set and their connection does not make a cycle in the minimum spanning tree that is going to be created.

 The constructor takes the number of vertices as a parameter, which is going to be the length of the tree parent array. This constructor it is also creating this array and it will initialise it with the required numbers. Each vertex is in his own set. This is haw the tree array is initially.

The next thing in this class is the findSet method that is taking a vertex as parameter and is returning the set in which that vertex is contained. Each set has a root and that root also represents the name of the set. In the start, because each node is contained in its own set, all vertices are roots.

When this method has to find the set for a vertex, it will check if vertex number is equal with the value from the treeParent array and it will search in that array until it will find the root of that vertex. Roots are found in tree Parent array when index is equal with value.

*While (vertex != treeParent[vertex])*

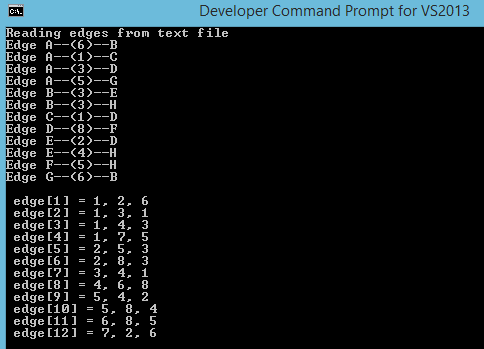
*vertex = treeParent[vertex];*

There is also a method in this class called union, which is making the union between 2 sets. The union of two sets is really easy as you just have to make a single change in the treeParent array. When I want to make the union between set1 and set2, I simply change the value from position set2 in treeParent array to be set1. (E.g. treeParent [set2] = set1; )

Another methods in this class are ShowTrees, ShowSets and ShowSet that are used for display purposes.

**Graph Class**

This class is containing the code that is doing the main part of this algorithm. It contains the number of vertices and number of edges from the graph, also an array of edges and an array of edges that will form the minimum spanning tree, both of type Edge.

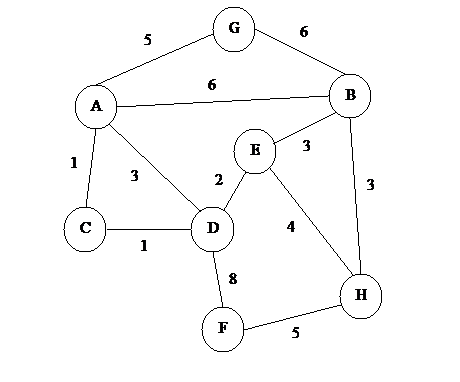
 The edges are stored from the text file in my edge array and will be stored as follows:

This class contains the method for Kruskal algorithm that it will return the mst array.

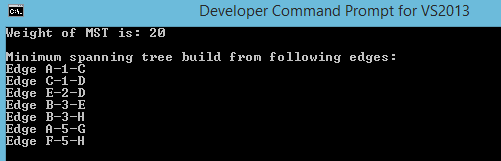
It will initially create the mst array that in won’t have any elements in the start, create the heap which is going to store the edge numbers as described above and create a partition of singleton sets for the vertices.

This method has a loop which will be iterated until the minimum spanning tree will be completed. In this loop, Edges are removed from the heap, in ascending order and the algorithm it will check if the starting point and the ending point of that edge are part of the same sets. If the elements are from different sets, the edge will be stored in mst array and the union operation for those sets will be executed. If the vertices are from the same set, there is no need for that edge to be stored in mst array because a cycle will be formatted, MST has no cycle, and another edge will be removed from heap until the minimum spanning tree will be created.

The minimum spanning tree for the above graph is:



After my implementation of Kruskal is executes, I get the same MST displayed as follows:



# Prim’s Algorithm

The output of Prim algorithm should be same as the output of Kruskal for the minimum spanning tree of that graph. I found this algorithm more complicated than Kruskal as I spend more time on this. In this algorithm I need a parent array, distance array and hPos array (storing the position for a vertex in hep).

This algorithm contains two classes Heap and Graph.

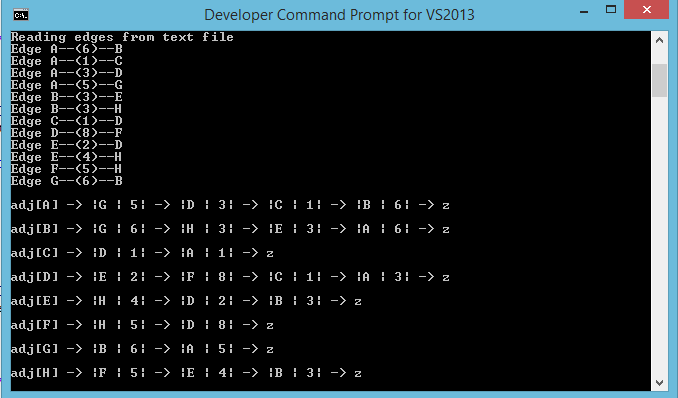
**Heap Class**

The heap contains the vertex numbers that are stored taking into account the distances. When a heap is created, the distance array and hPos array are passed as parameters in order to be used inside heap class. I spend a lot of time in order to find how to update the hPos array every time when some changes are made in heap. For example when an element have to be sifted down in heap, I was only saving in the hPos array the current element that was sifted down in his right position then I realised that I had some errors because of that and my MST was not the good one. I realised that when changes are made in the heap, more than one elements can be changed in heap and I had to keep track of all. Until realising how to update the hPos array for each change in my heap, I created a different method called FindEmelent which did the same thing but in a less efficient way. After many printings of the heap content and hPos array content, I realised where the problem was and I fixed but I left the findElement method there, commented out.

Shift Up and Sift Down methods are working similarly as they should, but instead of checking the heap content and arranging the heap for his content, I arranged the heap elements taking into consideration the distance array. The updates for hPos array are also done in these two methods.

**Graph Class**

In graph class I am reading the graph from the text file and store that in linked lists as follows.



In this class, I have the Prim method for finding MST. In this method I have three arrays, parent, dist and hPos. Dist array is initialised with the maximum value and the otre two arrays are initialised with 0. Here is the heap created and is passing the number of vertices, dist array and hPos to the heap class.

In the very start, the starting vertex is inserted into heap and a while loop will be executed until the heap will be empty. In the while loop, the first element from heap will be removed, and when this is removed, all vertices connected with this, are going to be inserted into the heap. If a vertex is already in the heap, the algorithm will check if the current distance is less than the distance it already have and if so, the Sift Up method will be called for this updated distance and if the distance is smaller than other distances from the heap, the hPos array will be also updated for all vertices. If the vertex is not in the heap, it is going to be inserted and put it in the right order with the Sift Up method.

The output of this algorithm is the following:

