

Homework 2: Graphs

Fabian Andres Prada Nino

1) Minimum Spanning Tree

The Minimum Spanning Tree was constructed following the Prim's algorithm. The algorithm I implemented has the following particular characteristics:

Variables:

- *int numeroVertices*: Stores the amount of vertices in the graph.
- *int[] numeroDeVecinos*: This array stores the amount of neighbours each vertex have.
- *int[][][] edges*: For each vertex, this array stores the "name" of their neighbours and the weight of the edge between them. $edges[i][k][0]$ refers to the name of the k th neighbour of vertex i , and $edges[i][k][1]$ refers to the weight of the edge joining vertex i and its k th neighbour. For instance $edges[1][1][0] = 128$ and $edges[1][1][1] = 59$
- *int[] heap*: Array that stores the "name" of the vertex in each position of the heap. At the initialization *heap* is empty. *heap* stores in an Min Heap structure the vertices we have already visited according to their current distance to the tree.
- *int[] papa*: For each vertex (except the initial one) this array stores the "name" of the vertex in the tree which is closest to it. At the initialization *papa* points to 0 for all vertices.
- *int[] distancia*: For each vertex, this array stores the distance to the closest vertex in the tree (i.e., its current *papa*). At the initialization *distancia* is 0 for all vertices. (This doesn't mean that all the vertices are "visible" since the initial step. Take into account that in this implementation *heap* is empty when the algorithm starts.)
- *int[] posHeap*: For each vertex, this array stores its current position in the heap. At the initialization *posHeap* points to 0 for all vertices.
- *int ultimaPosHeap*: Stores the length of the current heap. At the initialization $ultimaPosHeap = 0$.
- *int pesoTotal*: Stores the total weight of the current tree.
- *boolean[] estaAgregado*: Indicates if a vertex was already added to the tree. Equivalently, if it was already removed from the heap.

- *int[][] spanningTree*: Array that stores the edges of the current Minimum Spanning Tree according to the order they are added. *spanningTree[i][0]* and *spanningTree[i][1]* stores the name of the vertices associated to the *i*th edge of the MST.
- *int edgeST*: Count the amount of edges already added to the Minimum Spanning Tree.

Procedure:

- Add vertex 1 to the initial position of the heap, *heap[0] = 1*, and set its parent pointing to a null position, *papa[1] = -1*.

while(heap is not empty, i.e, *heap[0]! = 0*)

- Set the variable *posicionActual* pointing to the vertex in the top of the heap, i.e., *posicionActual := heap[0]*.
- Remove *heap[0]* and set *estaAgregado[posicionActual] = true*.
- Update the status of the neighbours of *posicionActual* in the heap:

for *i = 1, ..., numeroDeVecinos[posicionActual]*

- If the neighbour *i*th was already added to MST, i.e, *estaAgregado[neighbour] == true*, do not perform any action.
- If the neighbour *i*th has not been added to the MST neither to the heap, which is verified through *papa[neighbour] == 0*, set *papa[neighbour] = posicionActual*, *distancia[neighbour] = edges[posicionActual][i][1]*. Finally add this vertex to the heap.
- If the neighbour *i*th has not been added to the MST but it is already in the heap, compare its current distance to the tree with the distance to *posicionActual*. If the distance to *posicionActual* is lower, update its parent and the distance to the tree setting *papa[neighbour] = posicionActual*, *distancia[neighbour] = edges[posicionActual][i][1]* and update its position in the heap.

end for

- Finally update the MST:

if *papa[posicionActual]! = -1* (this is only false for the first vertex)

- Add the edge joining *posicionActual* with its parent:
spanningTree[edgeST][0] = papa[posicionActual], *spanningTree[edgeST][1] = posicionActual*.

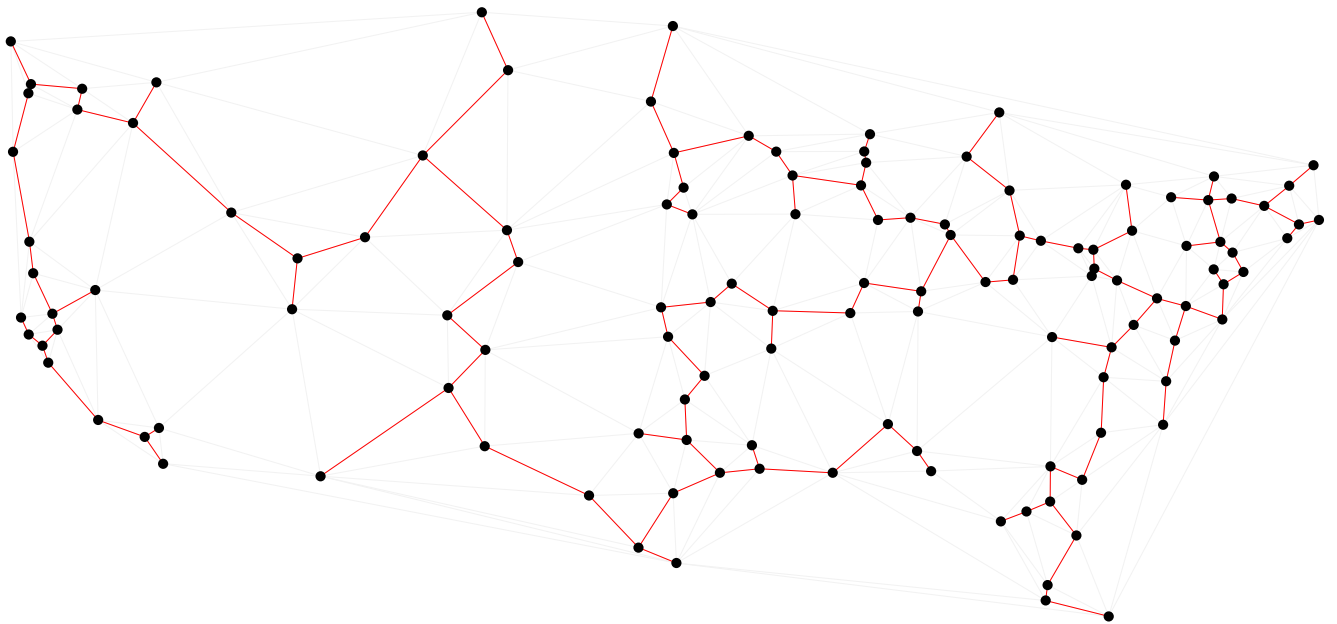
- Increase the counter of edges: $edgeST++$.
 - Update the total weight: $pesoTotal += distancia[posicionActual]$.
- end if

end while

Additionally, I managed the Min Heap structure using the following two methods:

- *upheapify()*: This method picks certain position of the heap and compare it with its predecessors. Comparisons and assignments are done iteratively until the initial value find an adequate position in the heap (one that make the structure indeed a Min Heap). This method is invoked when a new vertex is added to the heap (since they are added to the last position of the heap), and for update the position of a vertex that eventually became closer to the tree.
- *downheapify()*: Performs the heap update comparing a position with its descendents. This method is invoked once the root position is removed from the heap and replaced by the last position. *downheapify()* is precisely applied in the new root.

The results I obtained for the Minimum Spanning Tree are illustrated in the following figure:



Total Weight of the tree=22081.

As the theory asserts, we can check that any vertex is always connected to its nearest neighbour. It is also interesting to see how this tree avoid unnecessary large (heavy) edges and prefer the shortest (lightest) edges everywhere it is possible.

2)Shortest Path Tree from Vertex 104

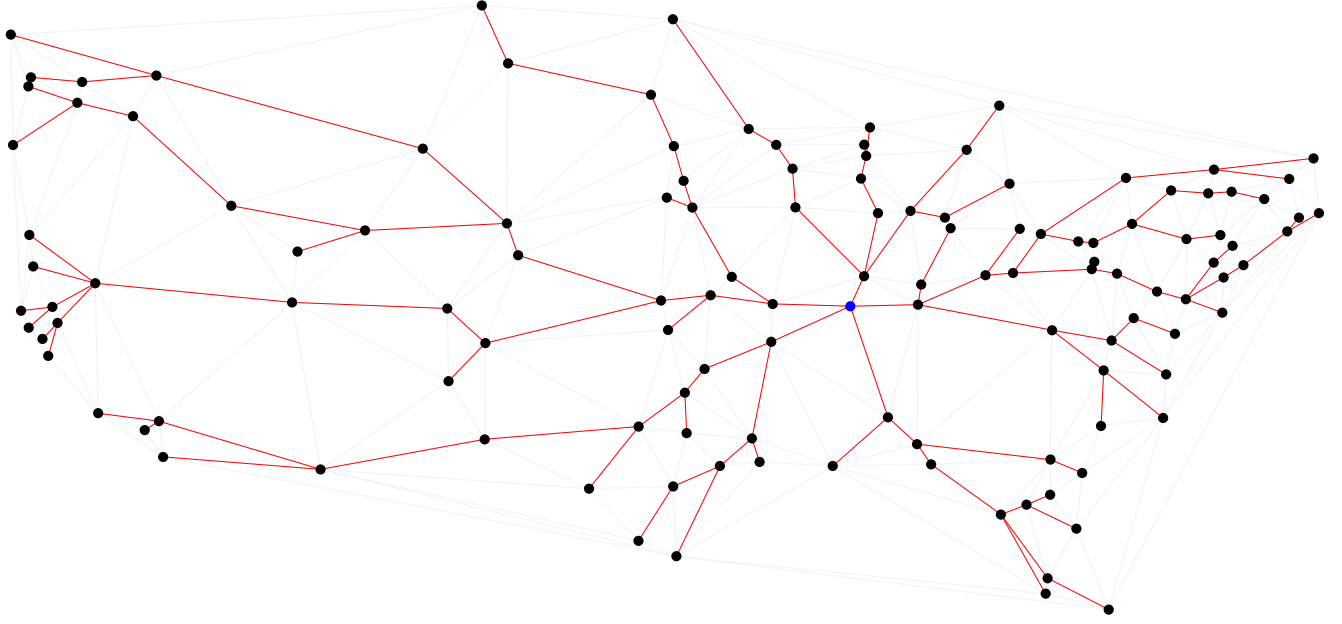
In order to find the Shortest Path Tree from vertex 104 I implemente Dijkstra's algorithm. This algorithm works almost identically to Prim's. The most important modifications in the implementation (compared to the previously described) are the following:

- The array *distancia[]* now stores the current distance from each vertex to vertex 104. At the initialization *distancia* is 0 for all verteces.(Again, this was just an initialization value and doesn't mean that all the verteces are "visible" since the inital step).
- The variables *int[][] shortestPath* and *int edgeSP* were defined in analogous way to *int[][] spanningTree* and *int edgeST* respectively.
- In the initial step add 104 to the first position of the heap, i.e, $heap[0] = 104$.
- Once we are visiting the neighbours of *posicionActual*, if the *ith* neighbour has not been added to the MST neither to the heap, set $papa[neighbour] = posicionActual$, and $distancia[neighbour] = distancia[posicionActual] + edges[posicionActual][i][1]$. Finally add this vertex to the heap.
- Once we are visiting the neighbours of *posicionActual*, if the *ith* neighbour has not been added to the MST but it is already in the heap, compare its current distance to the distance induced through *posicionActual*.

If $distancia[neighbour] > distancia[posicionActual] + edges[posicionActual][i][1]$, set $papa[neighbour] = posicionActual$ and $distancia[neighbour] = distancia[posicionActual] + edges[posicionActual][i][1]$. Finally update its position in the heap.

- To update the total weight of the tree set: $pesoTotal += (distancia[posicionActual] - distancia[papa[posicionActual]])$, that is equivalent to adding the weight of the edge between *posicionActual* and $papa[posicionActual]$

The results obtained for the Shortest Path Tree from vertex 104 were the following:



Total Weight of the tree=33314.

As expected, the tree is "centered" at vertex 104 (the blue vertex), and the the paths looks like rays with origin at this point. Circular paths around 104 does not occur anywhere.

3)Shortest Path between vertex 93 and 112

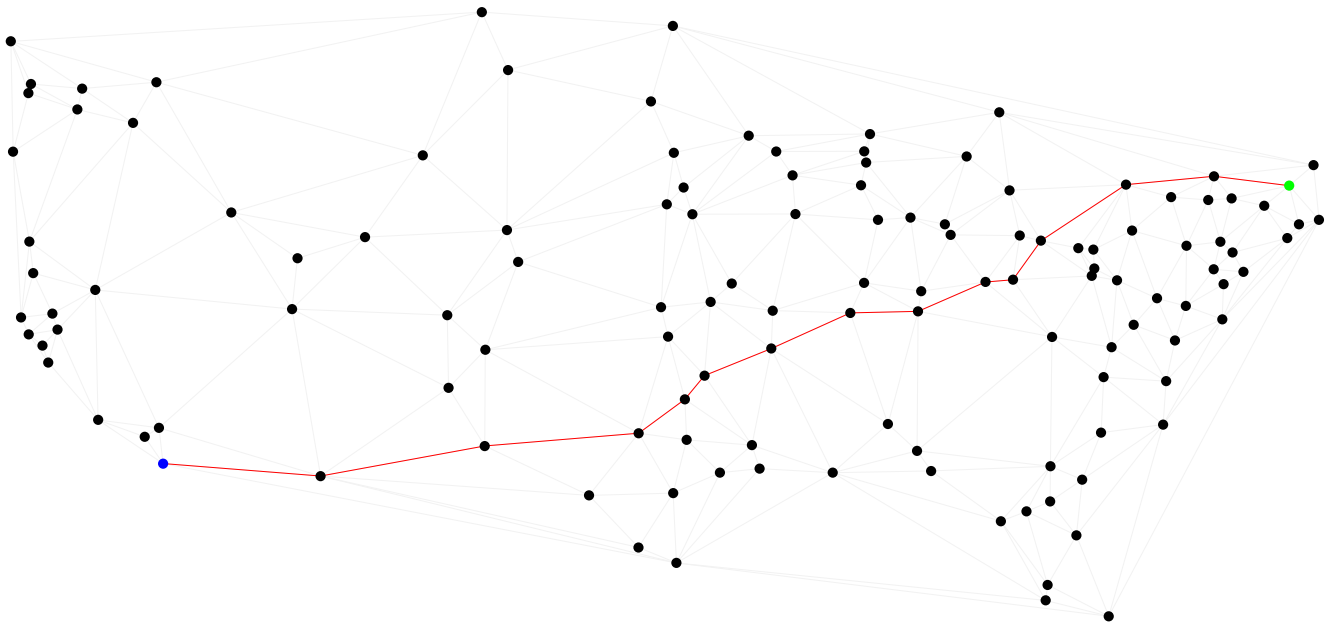
In order to find the shortest path between these two vertices I implemented the Dijkstra algorithm starting at vertex 93 and stopping the process once the vertex 112 is added to the shortest path tree. The following are the main modifications applied to the algorithm implemented in the previous section:

- The variables *int vertexS*, *int vertexT* are defined to store the "name" of the vertices to join. In this case, *vertexS* = 93 and *vertexT* = 112.
- The variable *int edgeSB* stores the amount of edges in the shortest path between *vertexS* and *vertexT*. The array *int [][][shortestBetween* stores the "name" of the vertices associated to each edge in this path.
- The while loop is executed until the first position of the heap refers to *vertexT*, i.e., **while** ((*heap*[0]! = *vertexT*)) the expansion of the shortest path tree from *vertexS* is done.
- Once *vertexT* is found, a **while** loop is invoked to count the edges of the path from *vertexS* to *vertexT*, in order to assign the memory space required for the variable

`shortestBetween[]`. Finally a **for** loop is implemented to visit all the edges of the path storing the "name" of their vertices in `shortestBetween[]`.

- The value of the weight is precisely the distance from *vertexT* to *vertexS*, i.e., $pesoTotal = distancia[vertexT]$.

The results of the implementation are illustrated in the following figure:



Total Weight of the path=4737.

The blue vertex corresponds to vertex 93 and the green one to vertex 112. This path seems as "linear" as possible. The following figures show the Shortest Path Tree from vertex 93 and the Shortest Path Tree from vertex 112 respectively. As expected, the path in the previous figure appears again in these:

