Search, Planning and Machine Learning; Programming Assignment 1: Search

Loes Erven s4538757, Niels Wolters s4633873

March 11, 2018

1 Specification

See the assignment (Programming assignment 1: search) on blackboard.

2 Design & Implementation

For representing the graph and its components we used a double int array. We started off using an object oriented approach with a vertex and an edge object and such, but it ended up being more complicated than useful so we thought of and used this simpler implementation instead.

Consider the following as a double int array:

```
{{0, 3, 3, 3, 0},
 {3, 0, 0, 0, 3},
 {3, 0, 0, 3, 3},
 {3, 0, 3, 0, 4},
 {0, 3, 3, 4, 0}};
```

This has 5 vertices. The numbers are the weights from edge to edge in the following way:
[0][0] == 0 is the edge from vertex0 to vertex0 and is therefore 0. 0 is thus used as infinite-like.
[0][1] == 3 is the edge from vertex0 to vertex1 and is 3.
This is the same as [1][0] as it is the same edge.

The for-loop started at line 61 is what actually finds which edges are to be connected.
For the full code and further explanation of details see section 5.

3 Runtime

We measure the runtime by counting the number of considered edges as suggested in the assignment. Counting each edge that is added to the tree is a bad idea because that would give every graph with the same size the same runtime which might not be the case as there might be more factors that contribute to runtime than size alone.

4 Experiments

How does the runtime change for increasing numbers of vertices |V| in G?

More vertices means a longer runtime. We see a non-linear increase, faster than linear. If we put a few results in a table, we get this:

nr. of vertices	runtime
2	1
3	3
4	6
5	10
6	15
7	21

A trained eye can immediately spot that the numbers on the right are triangular numbers, corresponding to the number on the right. (1 is the 2nd triangular number, 3 the 3rd, 6 the 4th, etc.)

And how if the number of edges |E| increases from |V|-1 to |V|.(|V|-1) for different |V|?

The number of edges can never be |V|.(|V|-1) as that number is always higher than the number of vertices (not counting a negative number of vertices) which would mean that there are multiple edges between the same pair of vertices which should never happen and will never happen by just increasing the number of vertices.

Does the runtime change in relation to the amount of equal edge-weights (i.e. for the extreme cases where all weights are equal c.q. all weighs are different, and intermediate cases)?

No, it appears to stay the same.

Does the runtime change in relation to the (minimum, maximum, average) magnitude of the edge weights, i.e. small weights like 1, 2 vs. large weights like 2000, 15000?

No, it appears to stay the same.

And how does the range in edge weights effect the runtime, e.g. a range of 1 ... |V| vs. a wider range of 1...Max $>$ |V| or even 1...∞?

No, it appears to stay the same.

Conclusion

The only factor that seems to influence the runtime, at least the way we defined runtime, is the amount of vertices in the graph.

5 Appendix

Java Code

```
public class Main {
        static final int Max = Integer.MAX_VALUE;

        public static void main(String[] args){

                int[][] matrix = new int[][]
                        {{0, 3, 3, 3, 0},
            {3, 0, 0, 0, 3},
            {3, 0, 0, 3, 3},
            {3, 0, 3, 0, 4},
            {0, 3, 3, 4, 0}
           }; // this has 5 vertices. The numbers are the weights from
               edge to edge in the following way:
           // [0][0] == 0 is the edge from vertex0 to vertex0 and is
               therefore 0. 0 is thus used as infinite-like.
           // [0][1] == 3 is the edge from vertex0 to vertex1 and is 3.
           // This is the same as [1][0] as it is the same edge.
                        /*{
                                {0, 2, 0, 6, 0, 6, 4, 0, 5, 0},
                                {2, 0, 3, 8, 5, 7, 0, 2, 1, 9},
                    {0, 3, 0, 0, 7, 8, 4, 8, 6, 4},
                    {6, 8, 0, 0, 9, 4, 7, 1, 5, 0},
                    {0, 5, 7, 9, 0, 8, 1, 8, 7, 6},
                    {6, 7, 8, 4, 8, 0, 2, 2, 0, 1},
                    {4, 0, 4, 7, 1, 2, 0, 0, 2, 5},
                    {0, 2, 8 ,1, 8, 2, 0, 0, 3, 3},
                    {5, 1, 6, 5, 7, 0, 2, 3, 0, 5},
                    {0, 9, 4, 0 ,6, 1, 5, 3, 5, 0} //intermediate
                   };
                   */
           System.out.println("Graph considered:");
           for (int[] x: matrix){
                   for (int y: x){
                           System.out.print(y + " ");
                   }
                   System.out.println();
           }
            int matrixDimensions = matrix.length;
                int[] visited = new int[matrixDimensions];
                int u = 0, v = 0;
                int total = 0;
                int edgeCounter = 0;

                //Initialize the array with visited edges and set all
                    non-existent edges in the array (all 0's) to max
                    int
                for(int i = 0; i < matrixDimensions; i++){
                        visited[i] = 0;
                        for(int j = 0; j < matrixDimensions; j++){
                                if(matrix[i][j]==0){
                                        matrix[i][j] = Max;
                                }
                        }
```

```
                }

                //Declare our initial vertex
                visited[0] = 1;

                //For the outer for-loop the counter has to be one
                    smaller than matrixDimensions, since there are V-1
                    edges in a MSP
                for(int counter = 0; counter < matrixDimensions - 1 ;
                    counter++){

                        int min = Max;
                        //The edges that are to be connected are found
                            here
                        for(int i = 0; i < matrixDimensions; i++){
                                if(visited[i]==1){
                                        edgeCounter++;
                                        for(int j = 0; j <
                                            matrixDimensions; j++){
                                                if(visited[j]==0){
                                                        if(min > matrix
                                                            [i][j]){ //
                                                             if true
                                                            then an
                                                            edge to
                                                            connect is
                                                            found
                                                                min =
                                                                    matrix
                                                                    [i
                                                                    ][j
                                                                    ];
                                                                u = i;
                                                                v = j;
                                                        }
                                                }
                                        }
                                }
                        }
                        visited[v] = 1;
                        total += min;

                        System.out.println("Edge connected: "+u+" -> "+
                            v+" : "+min);



                }
                System.out.println("The total weight of the spanning
                    tree is "+ total);
                System.out.println(edgeCounter + " Edges were
                    considered");
        }


}
```

Execution log

```
Execution log:

Graph considered:
0 3 3 3 0
3 0 0 0 3
3 0 0 3 3
3 0 3 0 4
0 3 3 4 0
Edge connected: 0 -> 1 : 3
Edge connected: 0 -> 2 : 3
Edge connected: 0 -> 3 : 3
Edge connected: 1 -> 4 : 3
The total weight of the spanning tree is 12
10 Edges were considered


Graph considered:
0 3 5 6 0
3 0 0 0 7
5 0 0 2 1
6 0 2 0 4
0 7 1 4 0
Edge connected: 0 -> 1 : 3
Edge connected: 0 -> 2 : 5
Edge connected: 2 -> 4 : 1
Edge connected: 2 -> 3 : 2
The total weight of the spanning tree is 11
10 Edges were considered


Graph considered:
0 2 0 6 0 6 4 0 5 0
2 0 3 8 5 7 0 2 1 9
0 3 0 0 7 8 4 8 6 4
6 8 0 0 9 4 7 1 5 0
0 5 7 9 0 8 1 8 7 6
6 7 8 4 8 0 2 2 0 1
4 0 4 7 1 2 0 0 2 5
0 2 8 1 8 2 0 0 3 3
5 1 6 5 7 0 2 3 0 5
0 9 4 0 6 1 5 3 5 0
Edge connected: 0 -> 1 : 2
Edge connected: 1 -> 8 : 1
Edge connected: 1 -> 7 : 2
Edge connected: 7 -> 3 : 1
Edge connected: 7 -> 5 : 2
Edge connected: 5 -> 9 : 1
Edge connected: 5 -> 6 : 2
Edge connected: 6 -> 4 : 1
Edge connected: 1 -> 2 : 3
The total weight of the spanning tree is 15
45 Edges were considered


Graph considered:
0 9 9 9 9 9 9 9 9 9
9 0 9 9 9 9 9 9 9 9
9 9 0 9 9 9 9 9 9 9
```

```
9 9 9 0 9 9 9 9 9 9
9 9 9 9 0 9 9 9 9 9
9 9 9 9 9 0 9 9 9 9
9 9 9 9 9 9 0 9 9 9
9 9 9 9 9 9 9 0 9 9
9 9 9 9 9 9 9 9 0 9
9 9 9 9 9 9 9 9 9 0
Edge connected: 0 -> 1 : 9
Edge connected: 0 -> 2 : 9
Edge connected: 0 -> 3 : 9
Edge connected: 0 -> 4 : 9
Edge connected: 0 -> 5 : 9
Edge connected: 0 -> 6 : 9
Edge connected: 0 -> 7 : 9
Edge connected: 0 -> 8 : 9
Edge connected: 0 -> 9 : 9
The total weight of the spanning tree is 81
45 Edges were considered


Graph considered:
0 2 3 4 5 6 7
2 0 10 23 8 37 42
3 10 0 33 28 100 22
4 23 33 0 11 13 9
5 8 28 11 0 123 47
6 37 100 13 123 0 19
7 42 22 9 47 19 0
Edge connected: 0 -> 1 : 2
Edge connected: 0 -> 2 : 3
Edge connected: 0 -> 3 : 4
Edge connected: 0 -> 4 : 5
Edge connected: 0 -> 5 : 6
Edge connected: 0 -> 6 : 7
The total weight of the spanning tree is 27
21 Edges were considered

Graph considered:
0 1 2 4
1 0 3 5
2 3 0 6
4 5 6 0
Edge connected: 0 -> 1 : 1
Edge connected: 0 -> 2 : 2
Edge connected: 0 -> 3 : 4
The total weight of the spanning tree is 7
6 Edges were considered

Graph considered:
0 1 2
1 0 3
2 3 0
Edge connected: 0 -> 1 : 1
Edge connected: 0 -> 2 : 2
The total weight of the spanning tree is 3
3 Edges were considered

Graph considered:
```

```
0 1
1 0
Edge connected: 0 -> 1 : 1
The total weight of the spanning tree is 1
1 Edges were considered

Graph considered:
0 1 2 4 7 8
1 0 3 5 9 10
2 3 0 6 11 12
4 5 6 0 13 14
7 9 11 13 0 15
8 10 12 14 15 0
Edge connected: 0 -> 1 : 1
Edge connected: 0 -> 2 : 2
Edge connected: 0 -> 3 : 4
Edge connected: 0 -> 4 : 7
Edge connected: 0 -> 5 : 8
The total weight of the spanning tree is 22
15 Edges were considered

Graph considered:
0 1000 2000 4000 7000 8000
1000 0 300 50000 9000 10000
2000 300 0 6000 1100 120000
4000 50000 6000 0 13000 14000
7000 9000 1100 13000 0 15000
8000 10000 120000 14000 15000 0
Edge connected: 0 -> 1 : 1000
Edge connected: 1 -> 2 : 300
Edge connected: 2 -> 4 : 1100
Edge connected: 0 -> 3 : 4000
Edge connected: 0 -> 5 : 8000
The total weight of the spanning tree is 14400
15 Edges were considered


Graph considered:
0 1 2 3 4 5 6 7 8 9
1 0 11 12 13 14 15 16 17 18
2 11 0 10 19 20 21 22 23 24
3 12 10 0 25 26 27 28 29 30
4 13 19 25 0 31 32 33 34 35
5 14 20 26 31 0 36 37 38 39
6 15 21 27 32 36 0 40 41 42
7 16 22 28 33 37 40 0 43 44
8 17 23 29 34 38 41 43 0 45
9 18 24 30 35 39 42 44 45 0
Edge connected: 0 -> 1 : 1
Edge connected: 0 -> 2 : 2
Edge connected: 0 -> 3 : 3
Edge connected: 0 -> 4 : 4
Edge connected: 0 -> 5 : 5
Edge connected: 0 -> 6 : 6
Edge connected: 0 -> 7 : 7
Edge connected: 0 -> 8 : 8
Edge connected: 0 -> 9 : 9
The total weight of the spanning tree is 45
```

45 Edges were considered