Report - assignment 7

Camilla Kristiansen Birkelund IDATT2101 October 24th, 2022

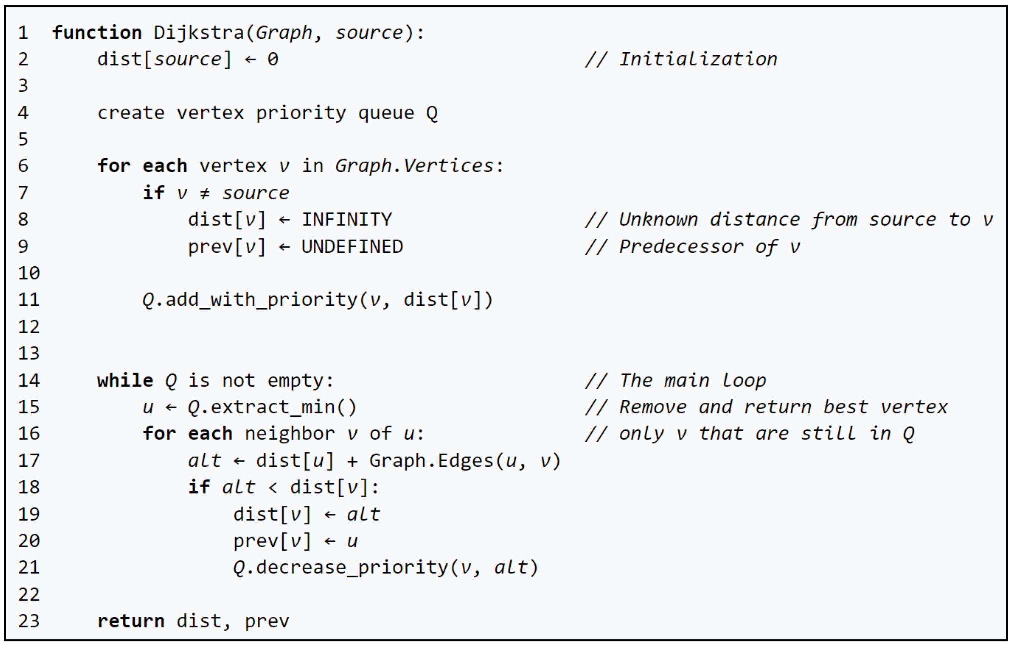
Assignment description

The assignment was to implement either Dijkstra's or Prim's algorithm. The program should also be able to present a result of the algorithm by printing the result after using the algorithm on the files "vg1" and "vg5". The chosen algorithm for this solution is Dijkstra's.

Implementation

The source code for this solution can be viewed in the file "dijkstrasalgorithm.java". The implementation has five classes, which represents the different structures needed to perform Dijkstra's algorithm on a graph. The Node class and the Edge class represent the nodes and edges in a graph, and only contains variables and constructors. The QueueElement class represents the different elements in the priority queue, and has a compareTo method and an equals method. The DijkstraResult class contains a toString method to print the results of the algorithm.

The Graph class represents the graph with methods to read and create a graph from a file (fromFile), and to perform Dijkstra's algorithm. The fromFile method takes in the string value of a filename as a parameter and returns a graph object. The method for Dijkstra's algorithm was implemented by using the following pseudocode as a guideline. The solution provided is altered to fit the rest of the program's code, such as taking in only one parameter (int startnode, which is the node the algorithm should start from) and returning a DijkstraResult object.



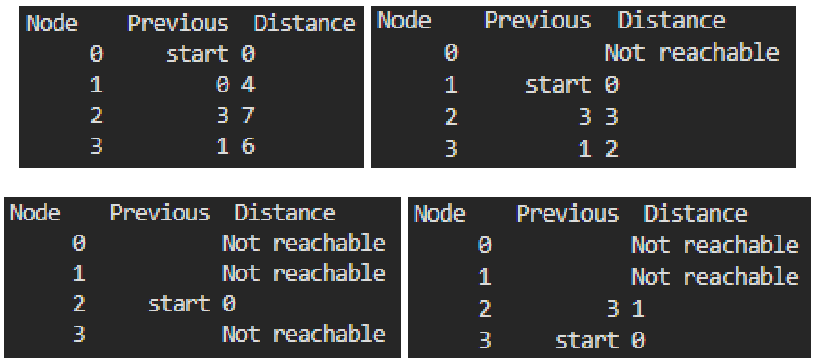
```
function Dijkstra(Graph, source):
    dist[source] ← 0                           // Initialization

    create vertex priority queue Q

    for each vertex v in Graph.Vertices:
        if v ≠ source
            dist[v] ← INFINITY                 // Unknown distance from source to v
            prev[v] ← UNDEFINED                // Predecessor of v

        Q.add_with_priority(v, dist[v])


    while Q is not empty:                      // The main loop
        u ← Q.extract_min()                    // Remove and return best vertex
        for each neighbor v of u:              // only v that are still in Q
            alt ← dist[u] + Graph.Edges(u, v)
            if alt < dist[v]:
                dist[v] ← alt
                prev[v] ← u
                Q.decrease_priority(v, alt)

    return dist, prev
```

Figure 1: The pseudocode used for inspiration for Dijkstra's algorithm (Wikipedia, n.d.).

Using the library java.util.PriorityQueue in the implementation of Dijkstra's algorithm had several advantages, such as the methods remove and offer, in addition to keeping the code shorter and more concise due to not having to implement a heap-based priority queue structure from scratch.

Results

The assignment required the program to be able to run the files "vg1", "vg2", "vg3", "vg4" and "vg5" with any chosen start node. To test if the program was able to do this, the file "vg1" was run with all possible start nodes.



```
Node    Previous  Distance
   0       start 0
   1           0 4
   2           3 7
   3           1 6
```

```
Node    Previous  Distance
   0             Not reachable
   1       start 0
   2           3 3
   3           1 2
```

```
Node    Previous  Distance
   0             Not reachable
   1             Not reachable
   2       start 0
   3             Not reachable
```

```
Node    Previous  Distance
   0             Not reachable
   1             Not reachable
   2           3 1
   3       start 0
```

Figure 2: The results of running the file "vg1" with all different start nodes.

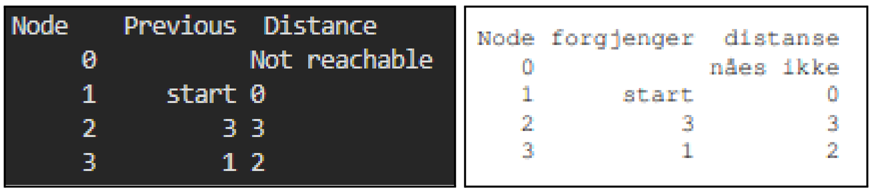
The results of the test confirmed that the implementation was able to run with any chosen start node (as long as the node exists in the graph). The runtime for the algorithm on each file was taken to evaluate the efficiency of the implementation.

Filename	vg1	vg2	vg3	vg4	vg5
Runtime	0.07 seconds	0.07 seconds	0.32 seconds	4.32 minutes	0.06 seconds

Figure 3: The runtimes for each file when using the implemented algorithm.

The runtime of this implementation is relatively fast due to using a heap-based priority queue, but the program could be even faster if the solution was written in C/C++, or another language that provides more detailed control.

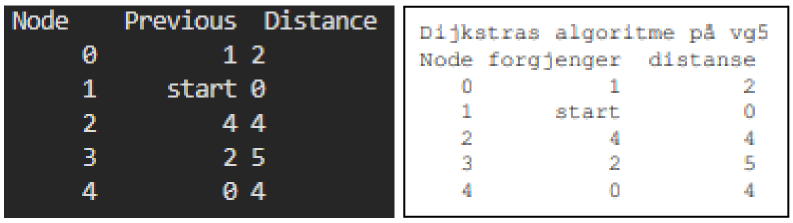
When running “vg1” and “vg5” the results were equal to the results in the assignment description, which means that the implementation of Dijkstra’s algorithm works as expected.



```
Node     Previous  Distance
    0            Not reachable
    1      start 0
    2          3 3
    3          1 2
```

```
Node  forgjenger   distanse
  0                nåes ikke
  1        start          0
  2            3          3
  3            1          2
```

Figure 4: The results of running the algorithm for file “vg1” on the left side. The expected results from the assignment description on the right side.



```
Node     Previous  Distance
    0          1 2
    1      start 0
    2          4 4
    3          2 5
    4          0 4
```

```
Dijkstras algoritme på vg5
Node  forgjenger  distanse
  0          1          2
  1      start          0
  2          4          4
  3          2          5
  4          0          4
```

Figure 5: The results of running the algorithm for file “vg5” on the left side. The expected results from the assignment description on the right side.

Sources

Wikipedia (n.d.). *Dijkstra's algorithm*. https://en.wikipedia.org/wiki/Dijkstra%27s_algorithm