Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Übungen zur Vorlesung Informatik II (D-BAUG) FS 2017

D. Sidler, F. Friedrich

http://lec.inf.ethz.ch/baug/informatik2/2017

Problem set # 10 **15.5.2017 – 23.5.2017**

Problem 10.1. Online Median

For this task open the project on codeboard: https://codeboard.ethz.ch/inf2baugex10t01

In the exercise lessons you learned how the heap data structure can be used to implement an online median procedure. The basic trick is to employ two heaps, a min-heap and a max-heap that contain the upper and lower halves of the data, respectively. Then the median is either the root of the min heap or the root of the max heap or the average of the two.[1]

Implement exactly this algorithm. First study the code of `ArrayHeap.java` which was partially introduced in the exercise session. Then complete two functions in `OnlineMedian.java`. Start of with

```
public void Insert(double value) { .. }
```

Next you should complete the method:

```
public double GetMedian() { ... }
```

Do not be surprised if your solution for this task is very short - it is supposed to be like this.

To test your implementation, you can use the following commands which are implement in the `Main` class:

```
insert 5.5
median
```

The `insert` command takes a value (5.5) and inserts into the heap. The `median` command returns the median of all values currently in the heap.

To run the automatic test, un-comment the annotation `@RunTests` at the beginning of the class `Main`. Once you pass the automatic test you can submit your program.

Problem 10.2. Dijkstra's Algorithm

Now you are asked to implement Dijkstra's shortest path algorithm that was introduced in the lecture, please open the code template at https://codeboard.ethz.ch/inf2baugex10t02.

While all parts that are needed for the implementation have been demonstrated in the lecture already, we recommend that you again study

- `Edge.java` - implementation of an edge of the graph
- `Node.java` - implementation of a node within the graph
- `Heap.java` - a heap implementation as seen in the previous task. Required for the algorithm

Once you familiarized yourself with the code, you should try to complete the method `ShortestPath` inside `Graph.java`.

Since the code is a bit more involved than most of the code you have seen so far we basically just left out gaps in the algorithm and your task is to fill them in. Use the visual illustration of the algorithm from the lecture for help. Every `???` should only require a single line of code to fill it. See next page for the code with the gaps.

To test your implementation, you can use the following commands which are implement in the `Main` class:

```
edge a b 8
path a b
```

[1] Recall that the median of a set with an even number of data points is the average of the two most centered data points.

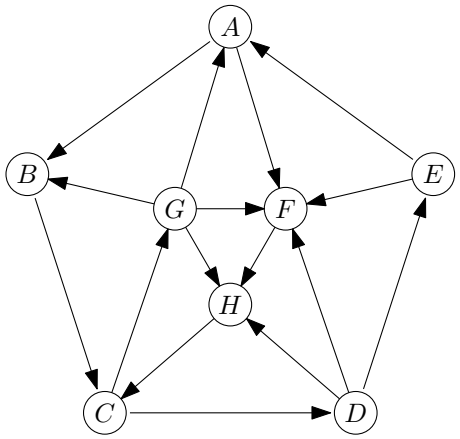
The edge command insert a new edge from a to b with distance 8 into the graph. The `path` finds the shortest path from a to b and prints it out.

To run the automatic test, un-comment the annotation `@RunTests` at the beginning of the class `Main`. Once you pass the automatic test you can submit your program.

```
public Vector<Node> ShortestPath(Node S, Node E) {
    //we start of by traversing through all nodes
    for (Node node : nodes) {
        ??? //for each node update its pathlen value to Integer.MAX_VALUE
        ??? //and set its parent node to null
    }

    ??? //create a vector to store the path
    ??? //create a Heap to work with during the algorithm
    ??? //set the path length at the startnode to 0
    Node newNode = start;

    while (newNode != null && newNode != end) {
        Vector<Edge> edges = ??? //get all the edges from the newNode

        //for each of the edges
        for (Edge edge : edges) {
        ??? //calculate the length the newNode plus this edge length
        ??? //get the Node that we would reach with this edge
        ??? //get the PathLength that node had so far (the one we just reached)
        if (newLength < prevLength) { //if previous is longer (which is always true if we did not
              visit that node yet)
            ??? //set its length to the newly calculated length
            ???//set its parent to the node we just came from
            if (prevLength == Integer.MAX_VALUE) { // we did not see this node yet
                ??? //therefore we would like to insert it to the heap
            } else {
                ??? //otherwise update its value in the heap
            }
        }
    }
    newNode = ??? //we then continue by taking node with shortest path out of the heap (the root)
                  //and the simply continue the loop from there
}


//now we backtrack from the final node
while (newNode != null) {
    path.add(newNode);
    newNode = newNode.GetPathParent();
}

return path;
}
```

Problem 10.3. Depth-first-search and breadth-first-search

Both depth-first-search (DFS) and breadth-first-search (BFS) admit a certain degree of freedom: one may choose the order in which the neighbors of a vertex are considered. For the graph shown below, let us assume that both DFS and BFS visit the neighbors of a vertex in alphabetical order.



1. Give the DFS and BFS ordering of the graph (i.e., the sequence in which the vertices are visited by DFS and BFS, respectively), starting from vertex A.
2. Is there a starting vertex in this graph from which the DFS ordering is the same as the BFS ordering?

Submission link: https://codeboard.ethz.ch/inf2baugex10t03