Assignment 6

Due date: 10:00am, Monday, 27th of October 2014

General Instructions

You have to do this assignment as a **team of two students** if you are an undergraduate student, or **individually** if you are a postgraduate student. Individual work of an undergraduate student is also allowed and will be treated same as team work. Team members should be from the same tutorial group. All implementations have to be done in JAVA.

Submissions have to include coversheet including names, student ids, and your tutorial group such that submissions can get marked.

Submit your solutions (including printout of the source code) for Exercises 1 as well as the results of the execution of your programs to the box "ADSA" on level 4, Ingkarni Wardli (close to reception) **by the deadline**. **No late submissions will be accepted.**

In addition, submit your source code for Exercises 1 using the web submission system.

Exercise 1 *Dijkstra and Bellman-Ford algorithms (4+4+3+2 points)*

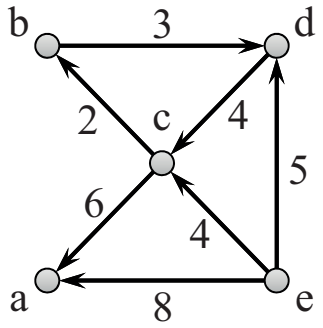
- Create in Java two methods called `Dijkstra` and `Bellman` which, respectively, implement Dijkstra's and Bellman-Ford's algorithms for the single-source-shortest path problem of a given directed graph $G = (V; E)$ and a source node s. Your implementation should use the graph implementation `AdjListGraph` of the previous assignment. Specifically, proceed as follows.
  - Modify your previous method `generateDigraph(...)` that generates a random graph consisting of $n = 15$ nodes using your or your partner's ID number as a random seed. Assume uniform distribution and with probability 1/4 produce an edge that goes from node u to v for $1 \leq u, v \leq n$, $u \neq v$. Assign to each generated edge a random integer cost in the interval $[1, \ldots, 10]$. Print the resulting graph as an output of your graph generating method and redraw it by hand as a paper variant.
  - For the implementation of `Dijkstra` use the simple settings discussed in the lecture notes. Run your `Dijkstra` method for the generated graph and find all possible shortest paths going from node s to all other nodes reachable from s. Print obtained shortest paths. Report when a node cannot be reached. Make sure that your implementation runs in time $O(n^2)$. Show the final solution by highlighting the shortest-path tree in the graph you have on paper. Numerate the sequence in which your algorithm considers the nodes.

  - Run your `Bellman` method for the generated graph and find all possible shortest paths going from node s to all other nodes reachable from s. Make sure that your implementation runs in time $O(nm)$. Confirm that both `Dijkstra` and `Bellman` solutions result in the same values of shortest paths. Print obtained shortest paths.
  - Update the existing edges of your graph and with probability 1/2 negate the cost by setting $c(u, v) = -c(u, v)$. Print the updated graph as an output of your graph generating method. Run your `Bellman` method again and check whether it determines any negative cycles in the new graph. Report the first node reached by using a negative cycle if such exists.

- If we modify the relax procedure of the Bellman-Ford algorithm so that for node v it updates its cost $d[v]$ and parent $\pi[v]$ if $d[v] \geq d[u] + cost(u; v)$ (instead of doing so only if $d[v]$ is strictly greater than $d[u] + cost(u; v)$), does the resulting algorithm still produce correct shortest-path weights and a correct shortest-path tree? Justify your answer.

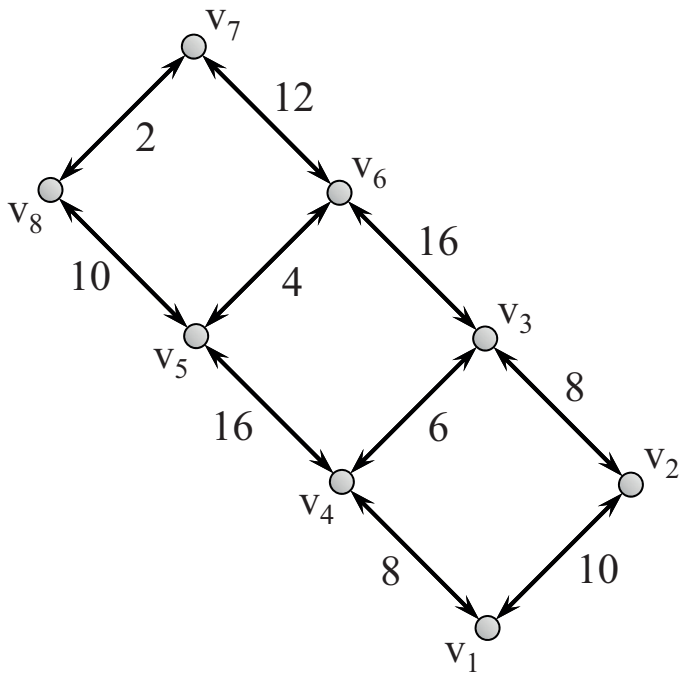
Exercise 2 *Floyd-Warshall Algorithm (4 points)*

Use the Floyd-Warshall algorithm to compute the all-pairs-shortest paths for the following graph.



Exercise 3 *Kruskal's Algorithm (3 points)*

Compute a minimum spanning for the following graph using Kruskal's algorithm. Show the status of your partial minimum spanning tree after each edge insertion and indicate for each edge whether it is included in the minimum spanning tree.



Submission instructions for programming code

First, type the following command, all on one line (replacing `aXXXXXXX` with your username):

```
svn mkdir --parents -m "ADSA"
https://version-control.adelaide.edu.au/svn/aXXXXXXX/2014/s2/adsa/assignment6
```

Then, check out this directory and add your files:

```
svn co https://version-control.adelaide.edu.au/svn/aXXXXXXX/2014/s2/adsa/assignment6
cd assignment6
svn add File1.java
svn add File2.java
...
svn commit -m "assignment6 solution"
```

Next, go to the web submission system at:

https://cs.adelaide.edu.au/services/websubmission/

Navigate to *2014*, *Semester 2, Adelaide*, *Algorithm and Data Structure Analysis*, then *Assignment 6*. Click *Make A New Submission For This Assignment* and indicate that you agree to the declaration. The script will then check whether your code compiles. You can make as many resubmissions as you like. If your final solution does not compile you won't get any marks for this solution.

The websubmission system for this assignment will open till 10am Monday October 27th, 2014.

End of Questions