Assignment 5 (due Sunday, April 7th, 6:00pm)

Instructions:

- Hand in your assignment using Crowdmark. Detailed instructions are on the course website.
- Give complete legible solutions to all questions.
- Your answers will be marked for clarity as well as correctness.
- For any algorithm you present, you should justify its correctness (if it is not obvious) and analyze the complexity.

1. **[20 marks] Shortest Paths** Consider another variant of single source shortest paths, where you are given a directed graph $G$ (could be undirected as well), a source $s$, and edge weights that are positive integers from 1 to 10. Describe an $O(n + m)$ time algorithm to solve this version of the problem.
   [Hint: Consider a faster implementation of the main data structure used in Dijkstra’s algorithm from class.]

2. **[25 marks] NP-Completeness** Consider the following optimization problem $k$-CLOSEST-SUBSET ($KCS$-$OPT$):
   
   Input: $n$ positive integers $S : \{a_1, \ldots, a_n\}$, a number $k$, and a positive integer $W$.
   
   Output: a subset $T \subseteq S$ with exactly $k$ elements minimizing $|\sum_{a_i \in T} a_i - W|$.
   
   Let $KCS$-$OPTVAL$ be the optimal value version of $KCS$-$OPT$, i.e. $KCS$-$OPTVAL$ is the problem where you don’t have to return the optimal set $T$ but only the value $|\sum_{a_i \in T} a_i - W|$.
   
   (a) **[10 marks]** Turn $KCS$-$OPT$ into a decision problem $KCS$-$DEC$ and show that the decision problem is in $NP$.
   
   (b) **[7 marks]** Show that if you can solve $KCS$-$DEC$ in polynomial time, then you can solve $KCS$-$OPTVAL$ in polynomial time.
   
   (c) **[8 marks]** Show that if you can solve $KCS$-$DEC$ in polynomial time, then you can solve $KCS$-$OPT$ in polynomial time.

3. **[25 marks] NP-Completeness** Consider the SINK-SOURCE SUBGRAPH (SSS) problem:
   
   Input: a directed graph $G(V, E)$.
   
   Output: “YES” iff there is a subgraph $H(V, E')$, with $E' \subseteq E$, such that each $v \in V$ satisfies one of two conditions: (i) either, in-degree($v$) = 0 and out-degree($v$) > 0; or (ii) out-degree($v$) = 0 and in-degree($v$) > 0. The in/out degree conditions are within $H$. 


Let us call a subgraph $H$ satisfying the above property \textit{sink-source} subgraph. Observe that in a sink-source subgraph, there are no paths of length 2, since every vertex either has zero in-degree or zero out-degree.

(a) \([5 \text{ marks}]\) Prove that SSS is in NP.

(b) \([20 \text{ marks}]\) Prove that SSS is NP-Complete through a reduction from 3SAT. Remember to have an iff argument in your reduction.

[Hint: Consider having one vertex $u_i$ for each clause $C_i$ in the 3SAT formula. Similarly have one vertex $v_j$ for each variable $x_j$ in the 3SAT formula. If $x_j$ appears in $C_i$ have an edge from $v_j$ to $u_i$. If $\overline{x}_j$ appears in $C_i$, add another edge from $u_i$ to $v_j$. Complete your construction by adding another new vertex $t$ that only has incoming edges; and one final new vertex $s$ that has a single edge to $t$.]

4. \([30 \text{ marks}]\) \textit{Programming Question} Implement Bellman Ford's single source shortest paths algorithm with several optimizations (described below). Given a directed graph $G(V, E)$ with arbitrary edge weights, and a source $s$, you should return one of the two possible outputs:

(1) If the graph contains a negative weight cycle, you should output the string: “NEGATIVE WEIGHT CYCLE”.

(2) If there are no negative weight cycles in $G$, you should output the distances (not the paths) from $s$ to all other nodes (including $s$) in increasing order of IDs in separate lines in the “ID distance” format, where you output “inf” if the source does not have a path to a particular node $v$:

\begin{verbatim}
0 0
1 -3
2 inf
...
\end{verbatim}

The input format will be as follows. The first line will contain 3 integers "n m s", where $n$ represents the number of nodes in the graph, $m$ the number of edges, and $s$ is the source node. Then, $m$ lines will follow with 3 integers “u v w” which represent one edge from node $u$ to $v$ with weight $w$. For simplicity the weights will be integers, but they can be negative, zero, or positive. The nodes will be in the range $[0, n - 1]$ and some nodes may not have any incoming or outgoing edges, so may not appear in the input file (so your output should be “inf” for them).

In order to get your implementations working correctly and efficiently you will need to implement three optimizations:

(a) \textit{Early stopping}: Recall that if the BF algorithm converges at iteration $i < n$, i.e., the shortest distance of each vertex $v$ at iteration $i - 1$ is the same as $i$, then you can stop and conclude that the distances at iteration $i$ (and $i - 1$) are the correct shortest path distances and there are no negative weight cycles in $G$. \hfill 2
(b) **Space optimization**: Recall that in each iteration $i$, the algorithm requires the distances of vertices only from iteration $i - 1$. So you only need $2n$ space to keep the “current” iteration’s distances and “previous” iteration’s distances.

(c) **Negative weight cycle checking**: Recall that if there is a negative weight cycle in $G$, the BF algorithm will not converge. You can check if there is a negative weight cycle simply by checking that the distances in the $n$’th iteration is not the same as the distances at the $n - 1$’th iteration, i.e., there is at least one vertex $v$, whose distance decreases from iteration $n - 1$ to $n$. If you find such a vertex you should output “NEGATIVE WEIGHT CYCLE”. (Note: because this check requires running the algorithm $n$ iterations, we will give only some small input graphs that contain negative weight cycles in the tests.)

Make sure your code runs in a reasonable amount of time. Each test case will have a generous timeout deadline.

The submission guidelines follow from the previous programming question. Submit one zip file through Marmoset: https://marmoset.student.cs.uwaterloo.ca/ which must contain your code file named `a5q4.py/a5q4.cpp`. You can use only C++ or Python as Programming Languages and your program should read from standard input and write to standard output.