ADS Homework 13

Name - Tewodros Adane
Email - tadane@jacobs-university.de

Problem 13.1 – Shortest Path Algorithm

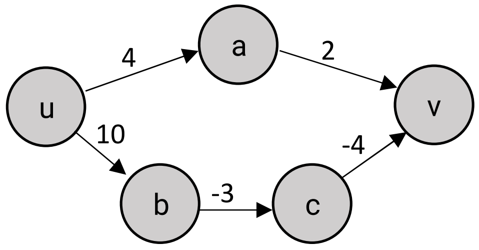
Your friend (who has not taken an "Algorithms and Data Structures" course) asks you for help on implementing an algorithm for finding the shortest path between two nodes u and v in a directed graph (possibly containing negative edge weights). The friend proposes the following algorithm:

1) Add a large constant to each edge weight such that all weights become positive.
2) Run Dijkstra's algorithm for the shortest path from u to v.

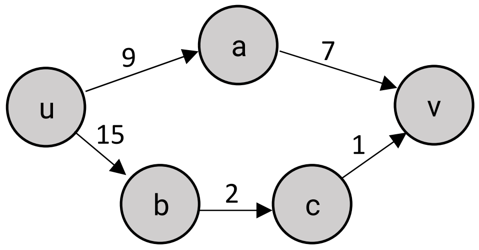
Prove or disprove the correctness of the above algorithm to find the shortest path.

Proof by Counterexample:

- Given the following graph **G** with negative edge weights.



- If we add 5 to all the edge weights to make all weights positive numbers.



Before adding an offset to the edge weights the shortest path from u to v is:

u → b → c → v (*shortest*) , which has a cost of 3

u → a → v , which has a cost of 6

However, after adding the offset the shortest path is:

$u \rightarrow b \rightarrow c \rightarrow v$	, which has a cost of 18
$u \rightarrow a \rightarrow v$ (*shortest*)	, which has a cost of 16

Therefore, adding a large constant to each edge and running Dijkstra's algorithm is not correct. This is because the distance of a path with more nodes will be offset more than the distance of a path with less nodes.

□

Problem 13.2 – Optimal Meeting Point

- The implementation and usage of an algorithm that uses Dijkstra's algorithm to find the optimal meeting point which takes least time for two people in different cities to meet is in the file "*OptimalMeetingPoint.cpp*".

Problem 13.3 – Number Maze

a) Given a Number Maze problem with **n × n** grid. I represented the puzzle problem as a graph problem by creating an adjacency matrix with **(n * n)** nodes and for all x_{ij} in the puzzle, if it is possible to go from i to j with our going off the grid then that is represented as an edge in the adjacency matrix and has a value **1** otherwise it has a value of **0**.

To solve the puzzle, I started by running Dijkstra's algorithm starting from the top-left(start) node. Then, if the bottom-right(end) node is still at $infinity$, it means that the puzzle is unsolvable, however if the end node has a non-infinite numerical value, it means the end can be reached from the start, in other words it's solvable.

b) The implementation of an algorithm that solves a number maze is in the file "*PuzzleBoard.h*", and the usage of the algorithm is in the file "*NumberMaze.cpp*".