Algorithms and Data Structures
Jacobs University Bremen
Dr. Kinga Lipskoch

Course: CH08-320201
May 3rd, 2019

Homework 11

- Submit one ZIP file per homework sheet which contains one PDF file (including pictures, computations, formulas, explanations, etc.) and your source code file(s) with one makefile and without adding executable, object or temporary files.
- The implementations of algorithms has to be done using C, C++, Python or Java.
- The TAs are grading solutions to the problems according to the following criteria: https://grader.eecs.jacobs-university.de/courses/320201/2019_1/Grading_Criteria_ADS.pdf

Problem 11.1 *Shortest Path Algorithm* (3 points)

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 (note that in order to disprove, you only need to give a counterexample).

Problem 11.2 *Optimal Meeting Point* (7 points)

You are trying to meet your friend who lives in a different city than you and you want to meet in a city that is between where either of you live. Time is limited and you are trying to minimize the time spent traveling. So, where exactly should you meet?
You are given a graph G with nodes $V = \{0, ..., n-1\}$ that represent the cities and edges E that represent streets connecting the cities directly. The edges are associated with the distance $d(e)$, which is the time needed to travel between two cities. You are given your own city p and your friend's city q with $p, q \in \{0, ..., n-1\}$.
Implement an algorithm that finds the target city m in which you and your friend should meet in order to minimize travel time for both of you (you drive towards your meeting city simultaneously, so if you travel for x minutes and your friend travels for y minutes, then you will want to minimize $\max(x, y)$). The graph is given to you with an adjacency matrix, where each entry x_{ij} represents the time (in minutes) that it takes to travel from city i to city j. Naturally, the indices are the nodes. The algorithm should return the target city $m \in \{0, ..., n-1\}$.
The prototype of the corresponding function should be similar to:

```
int find_meetup_city(int[][] adj_matrix, int your_city, int friend_city);
```

Problem 11.3 *Number Maze* (10 points)

Consider a puzzle that consists of a $n \times n$ grid where each field contains a value $x_{ij} \in \mathbb{N}$. Our player starts in the top left corner of the grid. The goal of the game is to reach the bottom right corner with the player.
Rules of the game: On each turn, you may move your player up, down, left or right. The distance by which the player moves in a chosen direction is given by the number of its current cell. You must stay within the board (you cannot go off the edge of the board).
Example: If your player is on a square with value 3, then you may move either three steps up, down, left or right (as long as you do not leave the board).

(a) (2 points) Represent the problem as a graph problem. Formalize it by determining what is represented as the nodes and as the edges.

(b) (4 points) Implement the class `PuzzleBoard` similar to class declaration shown below.

(c) (4 points) Implement an algorithm that returns the minimum number of moves required to solve this problem. If there is no solution, your algorithm should determine this fact.

```
class PuzzleBoard {
  private:
    // your choice
  public:
    // Subpoint (b)
    PuzzleBoard(int boardSize, int[][] fields = null);
    /* constructor should create the
    graph (as you defined it in subpoint (a) with the values from fields.
    If fields is null, then initialize the fields of the board with
    random values between 1 and boardSize-1. */
    bool makeMove(int direction);
    /* makes the move (if valid), direction is 0 for up, 1
    for right, 2 for down, 3 for left. Returns true if the move was
    valid, false otherwise. */
    bool getResult();
    /* Returns false if game is not over yet, true if puzzle was solved */
    std::ostream &operator<<(std::ostream &os, PuzzleBoard const &m);
    /* in Python, this is the __str__ method. */
    // Subpoint (c)
    int solve();
    /* returns the minimum number of moves needed to solve the puzzle,
    and -1 if it is not solvable. */
}
```

How to submit your solutions

You can submit your solutions via *Grader* at **`https://grader.eecs.jacobs-university.de`** as a generated PDF file and/or source code files.
If there are problems with *Grader* (but only then), you can submit the file by sending mail to `k.lipskoch@jacobs-university.de` **with a subject line that starts with CH08-320201**.

Please note, that after the deadline it will not be possible to submit solutions. It is useless to send solutions by mail, because they will not be graded.

This homework is due by Friday, May 10th, 23:00.