HOMEWORK 11

ALGORITHMS AND DATA STRUCTURES (CH08-320201)

Spring 2018

Prof. Dr. Michael Sedlmair
Computer Science & Electrical Engineering
Jacobs University

Due on Monday, May 21, 2018, 23:55 (EXTENDED DEADLINE).

Submission Format

Please respect the submission format we are giving you in this homework exercise. Grading will be done on an automated basis (except for proofs which we will inspect manually). If you do not respect the submission format, your homework will be graded with 0%. You will be asked to submit a zip file on Moodle. This zip file should contain the following file structure (no subfolders and respect the **exact** filenames):

```
submission.zip
     - problem1.pdf
     - problem2.py | problem2.cpp
     - bonus.pdf
     - problem3.py | problem3.cpp
     - problem4.py | problem4.cpp & problem4.h
     - problem4.pdf
```

Problem 1: Shortest Path Algorithm

(3 points) Your friend (who hasn't taken this 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). She 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 this algorithm to find the shortest path (note that in order to disprove, you only need to give a counterexample).

Problem 2: Optimal Meeting Point *(7 points)*

You are trying to meet your friend who lives in a different city than you - for that you will want to meet in a city that is between where either of you live. Time is limited because you study Computer Science at Jacobs University, so 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 with the distance $d(e)$, which is the time needed to travel from 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 $m \in \{0, ., n-1\}$.

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

Problem 3: A Picking Order *(3*+4 points)*

Consider a group of middle school students. Whenever they gather, they instinctively establish a *picking order*. This means that for any pair of two students, one student picks on the other, making fun of him/her. The same pair of students always chooses the same picking order - no matter the time or surroundings. Surprisingly, this order can contain cycles: there may be a student S who picks on student T, who picks on student U, who again picks on student S.

(a) BONUS Prove that any set of n students can be arranged in a line such that every student picks on the student immediately to its right.

(b) You are given a directed, unweighted graph representing the picking relationships among a set of n middle school students. This graph is given to you in a $n \times n$ adjacency matrix (`true` for an edge, `false` for no edge). Implement an algorithm to compute a picking list for the students, as guaranteed by part a). The algorithm should return a list which is a permutation of the indices $i \in \{0, .., n-1\}$, such that every student i from this permutation picks on student $i+1$ (for $i < n-1$).

Note that neither the professor nor the TAs support this kind of immature middle school behavior of picking on other people. Be nice.

```
int[] picking_order(bool[][] adj_matrix);
```

Problem 4: Number Maze *(2+4+4 = 8 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. Our goal is to reach the bottom right corner.
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 don't leave the board).

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

(b) Implement the class `PuzzleBoard`, as shown below.

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

```
class PuzzleBoard {
private:
     // up to you.
public:
     // Problem 1b)
     PuzzleBoard(int boardSize, int[][] fields = null); // constructor should create the
         graph (as you defined it in 1a) 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.

    // Problem 1c)
    int solve(); // returns the minimum number of moves needed to solve the puzzle, and
        -1 if it's not solvable.
}
```

Listing 1: Methods and classes to implement for the `PuzzleBoard` class.

Remarks

- Solutions have to be handed in via Moodle by the due date. For late submissions you need to get in contact with the TAs directly.
- You need to upload the files specified in the problem statements. The PDF file `solution_detailed.pdf` can contain the solutions for your entire homework. Programming assignments need to be handed in as **Python code** (if you want to use C++ please talk to the TAs). Please write your own code. It is ok to take snippets from online resources, but they need to be clearly marked.
- Exercises marked with a * are bonus problems. These count towards your number of points for this homework. The maximum number of official points can not be exceeded.