CS 372
ADVANCED ALGORITHMS
Spring 2017

ASSIGNMENT 2
Due Date: *Sunday, April 16th, 2017, 23:59*

Assignment Submission: *Turn in your assignment by the due date through LMS. Prepare a Python file (.py) that has all functions described in the assignment. Name the file as '<your first name>_<your last name>_assignment2.py'. Put your name and student ID to the left top corner of the file as comment. Also, prepare another file (Word, PDF, TXT) that describes your algorithms. Name the file as '<your first name>_<your last name>_<assignment2>.(word,pdf,txt)'.*

All work in questions must be your own; you must neither copy from (including online resources) nor provide assistance to anybody else. If you need guidance for any question, talk to the instructor or TA.

In all parts of this assignment, you are given a grid (that varies in different questions) as an input. Your goal is to write a Python function that finds the ***minimum cost*** walking distance from top-left corner (S: start) to the bottom-right corner (F: finish) of the grid.

You are required to solve all questions with a graph algorithm in this assignment. Represent the grid and possible moves with a graph. Essentially, each grid location can be represented with a node (V) and the possible moves can be represented with edges (E). After you get the graph representation, you can use shortest path algorithms in your solutions for different cases.

PART A (40 points)

In this part, each cell in the grid has a ***positive*** cost that you need to pay if you pass through it while walking from S to F. Furthermore, there may be obstacles in some cells as indicated with "X" as you can see in the example provided below.

S	X	1	4	9	X
7	7	4	X	4	8
3	X	3	2	X	4
10	2	5	X	3	F

Grid will be represented as a two-dimensional list. For example, for the grid above, the representation is as follows:

G = [[0, 'X', 1, 4, 9, 'X'], [7, 7, 4, 'X', 4, 8], [3, 'X', 3, 2, 'X', 4], [10, 2, 5, 'X', 3, 0]]

If the number of rows in the grid is n and the number of columns is m, your starting point (S) is (0, 0) and the target (F) is (n-1, m-1). In this example, you start from (0, 0) and want to reach to (3, 5).

You can only move to **left, right, up** and **down**. As long as there is no obstacle, the cell costs will be added up as you walk through in the grid. You can also **jump over an obstacle** but cannot stop there. For example you can move from (0, 0) to (0, 2) directly by passing through the obstacle but you cannot move to (0, 1). So, in the minimum cost path that will be outputted by your function, the locations of the obstacles that you jump over must not be printed. However, there is an ***extra cost of jumping over obstacle***. When you jump over an obstacle, the cost of the target cell is doubled. For example, if you jump from (0, 0) to (0, 2) in the example above, since the cost of target cell (0,2) is 1, the cost will be 2 (1x2) instead of 1. Similarly, if you jump from (1,4) to (3,4) over the obstacle at (2,4), the cost will be 6 (3x2). ***If there are more than one obstacles next to each other, you cannot jump over them.***

Write a function named '***find_shortest_path(G)'*** that finds the minimum cost path from the starting cell (S) to the target cell(F).

Input: Two dimensional list (G) that represents the grid.

Output: Print the minimum cost needed to go from S to F, the number of steps and the actual path.

Example: For the grid above a possible output is:

Minimum cost : 18

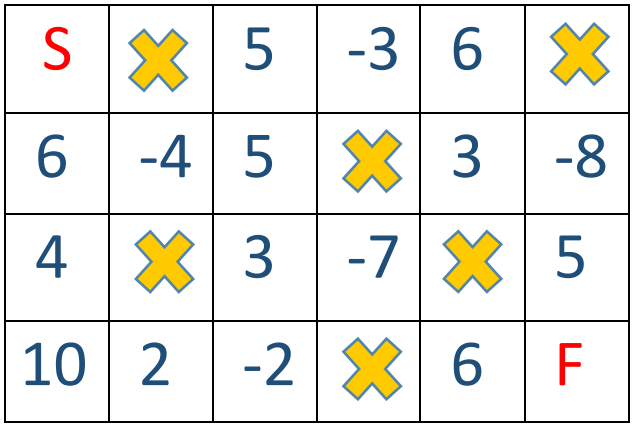
Steps: 5

Path: (0, 0), (0, 2), (0, 3), (2, 3), (2, 5), (3, 5)

Complexity of your algorithm should be ***O((V+E)*log V)*** in this question. The time spent for the initial graph creation is not included in the complexity calculation.

PART B (30 points)

In this part, you are going to solve different variation of the part A. In this part, the grid might have cells with **negative cost** and you can only move: **right** and **down**. The goal is the same with the last part: to find the minimum cost path from S to F. The negative costs are going to decrease your overall cost.The cost of jumping over the obstacles is the same with part A. If the cost of the cell we jump on is negative, since the cost is doubled, it is actually beneficial in this case.



Write a Python function named '***find_shortest_path_with_negative_costs(G)***' that finds the minimum cost path from the starting cell (S) to the target cell(F).

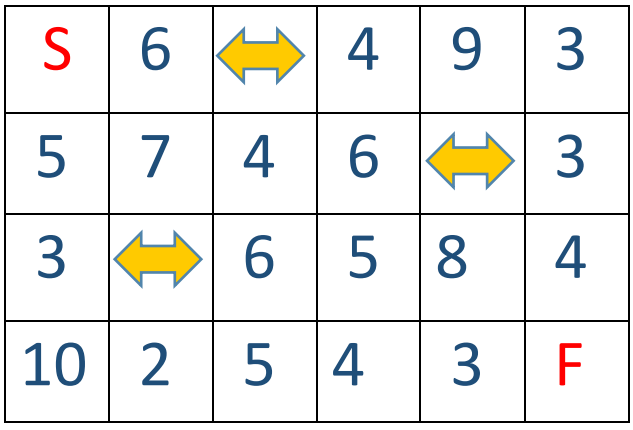
Input: Two-dimensional list (G) that represents the grid

Output: Print the minimum cost needed to go from S to F, the number of steps and the actual path.

This problem can be solved with an ***O(V+E)*** complexity algorithm (think about DAGs). The time spent for the initial graph creation is not included in the complexity calculation.

PART C (30 points)

In this part, you will have a grid with a number of ***checkpoints*** and all ***positive costs***. In this part, there are ***no obstacles***. You will again need to find the minimum cost path from S to F as in Part A; however, the path has to ***pass through at least one of the checkpoints***. So, develop and implement an algorithm that find the minimum cost path from S to F that passes through at least one of the checkpoints.



The checkpoints have no cost. In the grid above, there are 3 checkpoints indicated with arrows as you can see. This grid is represented as follow:

G = [[0, 6, 'Check', 4, 9, 3], [5, 7, 4, 6, 'Check', 3], [3, 'Check', 6, 5, 8, 4], [10, 2, 5, 4, 3, 0]]

One possible solution could be finding minimum cost paths from S to all checkpoints and then finding the minimum cost path from each checkpoint to F by running a separate shortest path algorithm from each checkpoint. Then, add up the costs from S to checkpoints and from checkpoints to F, and pick the one with the minimum total cost. The complexity of this algorithm would be ***O(k*(V+E)*logV)*** where ***k*** is the number of checkpoints.

The main goal of this part is to find a solution that has better complexity than ***O(k*(V+E)*logV)***. You should be able to find a solution whose complexity does not depend on the number of check points (k), namely (***O(V+E)*logV)***). The time spent for the initial graph creation is not included in the complexity calculation.

Write a Python function named '***find_shortest_path_with_checkpoint(G***)' that finds the minimum cost path from the starting cell (S) to the target cell(F) while ensuring that this path also goes through at least one of the checkpoints.

Input: Two-dimensional list (G) that represents the grid

Output: Print the minimum cost needed to go from S to F while passing through at least one of the checkpoints, the number of steps and the actual path.

For all questions, in addition to your Python code, do not forget to provide descriptions of your algorithms in Word, PDF or TXT.