Shri G. S. Institute of Technology and Science
Department Of Computer Engineering
CO 3463: Design and Analysis of Algorithms
Lab Assignment # 06 (Graph Algorithms)
Marks: 10 points

Submission Date: 2 April 2017@23:59
Demo Date: 3 April 2017 - 7 April 2017
Late Submission: Not allowed
No copying allowed. If found then students involved in copying will fail in this course.

Note: For the each program, draw a graph between the size of inputs and computational time required by yours program.

Q. 41. In this programming assignment, we will be implementing a simple graph traversal algorithm, known as Breadth First Search or BFS.
You will be given a 10 X 10 matrix, where every element represents a block of the city. The city will have three types of blocks : free block, dead-ends, and target. Free blocks are represented as single digit numbers from 0-9 (repetitions allowed), dead-ends as 'x' and target as 't'. Note that there will be exactly one target block in the city i.e., only one element with 't' in the matrix. Also , you will be given a 10 × 10 matrix in which a traversal exists from the initial block (0,0) to the target block. [So , it is also guaranteed that block (0,0) doesn't contain 'x']
(Note : Block (i,j) means the element in the (i+1)th row and (j+1)th column of the given matrix)
Initially , all blocks are marked unvisited .In the beginning, we add the block (0,0) to the queue .[By adding the block (i,j) to the queue, we mean that x-coordinate of the block (i,j) i.e. i needs to be pushed to the queue of x and the y-coordinate of block i.e. j to the queue of y].We also mark this block as visited .You need to perform the following sequence of operations in order :

0. If you are at target block, stop ; else go to step 1.
1. Look right : if the block is either free or target (not 'x') and also unvisited, add it to the queue , and mark it as visited.*
2. Look down : if the block is either free or target (not 'x') and also unvisited, add it to the queue , and mark it as visited.*
3. Look left : if the block is either free or target (not 'x') and also unvisited, add it to the queue , and mark it as visited.*
4. Look up : if the block is either free or target (not 'x') and also unvisited, add it to the queue , and mark it as visited.*
(Skip any of the corresponding steps 1,2,3 or 4 if right,down,left and upper

blocks do not exist respectively.)
5. Dequeue the head element.
6. If there are element(s) in the queue : (This is true for the given test cases)
   Go to next block on the front(head) of the queue. Then go to step 0.

* Note that when you add a block to the queue in your code, you also need to set the block to be a visited one.

(Definitely, the elements of the queue in Step-5 would not be empty as the test cases provided are the cases which definitely have a possible traversal.)

You need to write a function "move" which just adds the elements to the queue (x-coordinates to queue of x and y-coordinates to the queue of y checking if the element is visited or not (based on the "visited" vector)). Note that after you push block(i,j) to the queue, you need to mark it as visited by updating "visited" vector in your code and also just push the corresponding character in that block to the "answer" vector. This vector contains the order in which the blocks are pushed to the queue and will be used to verify the correctness of the order of elements added to the queue by you.

Q. 42. Frog Jumping: The latest hit on TV is a jumping game played on a giant rectangular chessboard. Each participant dresses up in a green frog suit and starts at the top left corner of the board. On every square there is a spring-loaded launcher that can propel the person either to the right or down.

Each launcher has two quantities R and D associated with it. The launcher can propel the person upto R squares to the right and upto D squares down. The participant can set the direction of the launcher to Right or Down and set the number of squares to jump to any number between 1 and R squares when jumping right, or between 1 and D squares when jumping down. The winner is the one who can reach bottom right corner of the chessboard in the smallest number of jumps. For instance, suppose you have 3 × 4 chessboard as follows. In each square, the pair of numbers indicates the quantities (R,D) for the launcher on that square.

(1,2) (1,2) (1,2) (2,1)

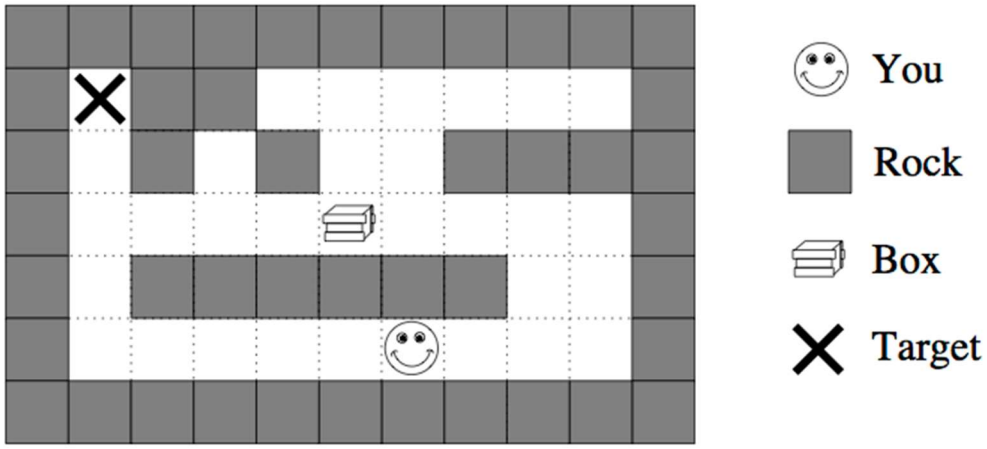
(3,1) (1,1) (1,2) (1,2)

(1,1) (1,1) (1,2) (2,2)

Here, one way to reach the bottom right corner is to first jump 1 square right, then jump 2 squares down to the bottom row, then jump right two times, one square a time, for a total of 4 jumps. Another way is to first jump 1 square down, then jump 3 squares right to the last column and finally jump one square down to the bottom right corner, for a total of 3 jumps. On this board, it is not possible to reach the bottom right corner in fewer than 3 jumps.

Your task is to write a program to calculate the smallest number of jumps needed to go from the top left corner to the bottom right corner, given the layout of the launchers on the board.

Q. 43. This is based on an old computer game called *Sokoban*. Imagine you are standing inside a two-dimensional maze composed of square cells which may or may not be filled with rock. You can move north, south, east or west one cell at a step. These moves are called *walks*. One of the empty cells contains a box which can be moved to an adjacent free cell by standing next to the box and then moving in the direction of the box. Such a move is called a *push*. The box cannot be moved in any other way than by pushing, which means that if you push it into a corner you can never get it out of the corner again. One of the empty cells is marked as the target cell. Your job is to bring the box to the target cell by a sequence of walks and pushes. As the box is very heavy, you would like to minimize the number of pushes. Write a program that will work out the best such sequence?



Q. 44. Write a program to compute a topological ordering of a given directed acyclic graph (DAG) with n vertices and m edges.

Q. 45. Given an undirected graph with n vertices and m edges, write a program to check whether it is bipartite graph.

Q. 46. Given a directed graph with possibly negative edge weights and with n vertices and m edges, write a program to check whether it contains a cycle of negative weight.

Q. 47. Given n points on a plane and an integer k, write a program to compute the largest possible value of d such that the given points can be partitioned into k non-empty subsets in such a way that the distance between any two points from different subsets is at least d.

Q. 48. Write a program for TSP problem.

Q. 49. Write a program using graph algorithm for yours own two real world problems and also find its complexity.