Hyuntaek Oh

Amir Nayyeri

CS 325_002

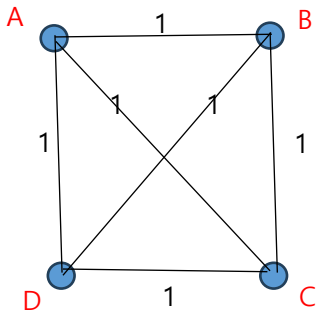
22 Feb 2024

Practice Assignment

Problem 1.

(a) Find a graph that has multiple minimum spanning trees.

The definition of a spanning tree is a subgraph that is a tree and contains every vertex of G in an undirected graph G. A graph has a spanning tree if and only if it is connected. For example, there is a graph G.



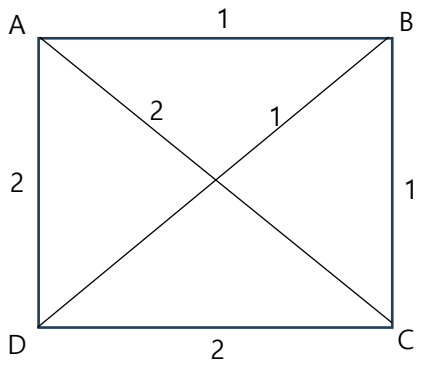
< **Figure 1.** A graph G >

The graph G has its subtrees that connect every vertex. Vertex A is connected to vertex B, C, and D. Similarly, vertex B is connected to vertex A, C, and D. All the vertexes in the graph G can be traversed from one vertex, so this graph is a spanning tree. Then, each edge between a vertex to another vertex has its own weight such as 1. Minimum spanning trees are a graph that has the minimum sum of the weights to traverse all other vertexes. To reach all the vertexes in the graph, for instance, vertex A should cost 3 to travel all other vertexes. All vertexes have the same sum of the weights to reach other vertexes. The sum of the weights in vertex A is 3, and vertex B, C, D also have the same value of the sum of the weights. It means this graph has multiple minimum spanning trees.

(b) Prove that any graph with distinct edge weights has a unique minimum spanning tree.

We can prove this question with contradiction. For example, there are multiple minimum spanning trees in a graph G that has distinct edge weights, and T1 and T2 are each of them, which have different minimum value. T1 and T2 can have same vertex but at least one that is included in T1 or T2, not both. Then, the edge 'e' in this case involved in one of them, which means the weight of 'e' in T1 or T2 can have different weights because all the weights are distinct. If the 'e' includes in T1 and is moved from T1 to T2, a new tree T' can be generated. T' has all the edges of T2 and has one more edge by deleting 'e' from T1 and adding to its tree. It contradicts that T1 is a minimum spanning tree. Thus, the graph G is unique.

(c) Find a graph with non-distinct edge weights that has a unique minimum spanning tree (can you generalize (b)?).



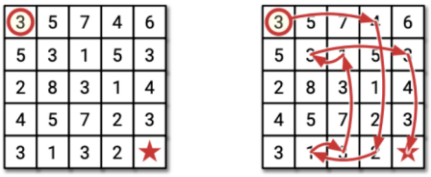
< **Figure 2.** A graph that has a unique minimum spanning tree >

As shown in Figure 2, there are non-distinct edge weights such as 1: {A, B}, {B, D}, {B, C}, 2: {A, C}, {C, D}, {A, D}, and there is only one way to form a minimum spanning tree. The minimum spanning tree will include edges {A, B}, {B, C}, and {B, D}, which are a minimum total weight of 3.

If the edge {A, C} and the edge {B, D} exchanged each other, this graph does not have a unique minimum spanning tree because both vertex A and vertex B would be possible minimum spanning tree. It contradicts the condition of a unique minimum spanning tree, so

Problem 2.

A number maze is an $n \times n$ grid of positive integers. A token starts in the upper left corner; your goal is to move the token to the lower-right corner. On each turn, you are allowed to move the token up, down, left, or right; the distance you may move the token is determined by the number on its current square. For example, if the token is on a square labeled 3, then you may move the token three steps up, three steps down, three steps left, or three steps right. However, you are never allowed to move the token off the edge of the board. Describe and analyze an efficient algorithm that either returns the minimum number of moves required to solve a given number maze, or correctly reports that the maze has no solution. For example, given the number maze in the figure below, your algorithm should return the integer 8.



< **Figure 3.** An example of maze >

We can solve this problem by using BFS (Breath First Search) with a queue because we need to search and accumulate possible paths into the queue from starting point to a goal in each step. The program should read the value of the starting point. If the value of the starting point is equal to or less than zero, then the program returns -1, which means there is no possible move. By using this value, agent can move right or down at the first step and check if the move is valid, not out of the maze. The agent is moved the directions such as up, down, left, right by the value in the grid, and the program stores the count of the moves one by one. If the agent finally reaches the goal, the program returns accumulated moves. The pseudo code of this problem 2 is below.

```
def Min_number_of_moves(maze):
        Initialize n to the length of the maze.    // Constraint (out of the maze)
        If starting point is 0:        return -1          // Condition for no possible paths
        Use the set list “visited” to remember

        Initialize queue = [(0,0,0)]
        While queue is not empty:
                Store the values from the queue into y, x, moves
```

If reach the goal:

return moves

Move agent Up, Down, Left, Right direction in each way under the conditions: not out of the maze, visited

Return -1 // No case