CS2040S: Data Structures and Algorithms

Tutorial Problems for Week 13: Shortest Paths (II)

For: 8 Nov 2021, Tutorial 12

Solution: Secret! Shhhh... This is the solutions sheet.

Problem 1. True or False?

For each of the following, determine if the statement is True or False, justifying your answer with appropriate explanation.

a) If a undirected graph has a unique minimum spanning tree, and we run Prim's and any single source shortest path algorithm from the same source, the final result of edges chosen will form the same spanning tree.

b) If the weights of all edges in a positively weighted graph are unique, there is always a unique shortest path (the smallest total cost is unique) from a source to a destination in such a graph.

c) A connected, undirected graph will always form a shortest path tree with $V - 1$ edges.

d) The shortest path from any vertex A to any vertex B is the combination of the shortest path from A to any vertex C and shortest path from C to vertex B.

e) The algorithm below will still give the correct shortest path.

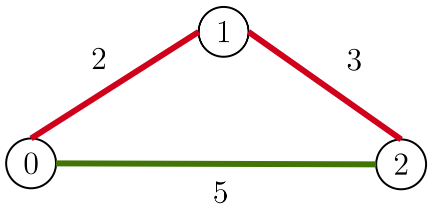
```
function Dijkstra(Graph, source):

    create vertex set Q

    for each vertex v in Graph:
        dist[v] = INFINITY
        prev[v] = UNDEFINED
        add v to Q
    dist[source] = 0

    while Q is not empty:
        u = vertex in Q with min dist[u]

        remove u from Q

        for each neighbor v of u: // only v that are
           still in Q
            alt = dist[u] + length(u, v)
            if alt < dist[v]:
                dist[v] = alt
```

```
                    prev[v] = u
        return dist[], prev[]
```

Solution:

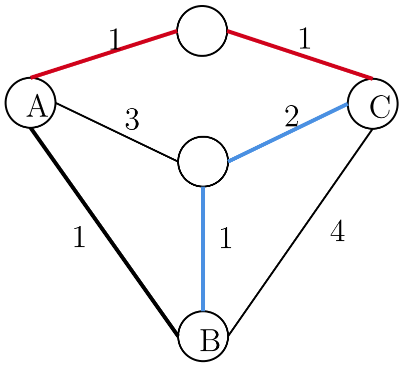
a) False. The minimum spanning tree and shortest path spanning tree can be different. In fact, changing the source you start from for Prim's should result in the same minimum spanning tree, but you may get different spanning trees for different sources for shortest paths.

b) False. There can be more than one paths with same weight. Consider the example below, where there are 2 possible shortest paths from vertex 0 to vertex 2, but all edge weights are unique.



c) True, because the tree formed is also a spanning tree.

d) False. A simple counter example is a directed graph, where A has a path to B, but there are no paths between A and C and C and A. Another example is shown below.

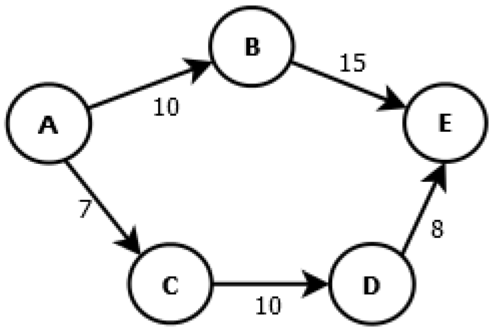


The red edges denote the shortest path from A to C, and the blue edges denote the shortest path from C to B. However, the shortest path from A to B is merely the direct edge from A to B.

e) True. We are still processing each vertex in the correct order, since we are always removing the vertex with the minimum distance from the vertex set Q in $O(V)$ time. Each relaxation will now take $O(1)$ time per edge considered. The time complexity of this algorithm is $O(V^2)$.

Problem 2. Promoter

A salesman frequently needs to drive from one city to another to promote his products. Since time is of the essence, he wants to use the shortest route to get from one city to another. However in every city he passes he will have to pay a toll fee. The toll fee is the same for every city and it is a positive number. Therefore, given two different routes of the same distance (positive numbers) to get from city A to city B, he will prefer the one which passes through fewer cities. An example is shown below. Propose the best algorithm using what you have learnt so far (and a bit more), so that the salesman will choose a route from any source city A to any destination city B such that it has the shortest distance and also passes through the fewest cities (you may assume there is a way to get from any city A to any city B). What is the running time for your algorithm?



To get from A to E, route A,B,E is preferred over route A,C,D,E even though both have the same cost 25, since A,B,E goes through fewer cities.

Solution: Modify the priority queue of Dijkstra's algorithm (either original or modified can be used here; for the sample solution we show the modification to original Dijkstra). From storing pairs `(D[v], v)`, we store triplets `(D[v], H[v], v)`, where `H[v]` stores how many hops / vertices have been used in the shortest path so far. On the sample graph above, the execution of Original Dijkstra's is shown below. Observe the last step involving vertex E.

1. PQ $= \{(0, 1, A), (\infty, \infty, B), (\infty, \infty, C), (\infty, \infty, D), (\infty, \infty, E)\}$. Process $(0, 1, A)$.
2. PQ $= \{(7, 2, C), (10, 2, B), (\infty, \infty, D), (\infty, \infty, E)\}$. Process $(7, 2, C)$.
3. PQ $= \{(10, 2, B), (17, 3, D), (\infty, \infty, E)\}$. Process $(10, 2, B)$.
4. PQ $= \{(17, 3, D), (25, 3, E)\}$. Process $(17, 3, D)$. Note that it will not change $(25, 3, E)$ as $(25, 3, E)$ is better than $(25, 4, E)$ from $(17, 3, D)$.
5. PQ $= \{(25, 3, E)\}$. Process $(25, 3, E)$, but E has no outgoing edges.
6. PQ $= \{\}$. Terminate algorithm.

Problem 3. Lego Mindstorms EV3 (2013/2014 CS2010 S1 WQ2)

Steven has a new toy, LEGO Mindstorms EV3 (which he will pass to his daughter Jane in a few years' time). He wants to command his robot to move from one cell to another cell in an $m \times n$ grid containing mostly '`.`' (passable cells) and some '`#`' (blocked cells). The diagram below shows a sample 4×7 grid with 8 obstacle cells (the '`#`'s):

```
. . . . . . .
. # # # # # .
. # . . . # .
. . . # . . .
```

Initially, Steven's robot is at coordinate (0, 0), the top-left corner of the grid, and faces east. At each cell, Steven's robot can only do one of the two actions below:

1. Move forward by one cell according to it's current direction. For example, if the robot currently at coordinate (0, 0) and faces east, it will be in coordinate (0, 1) and still faces east after this action. Such action consumes 3 seconds.
2. Rotate the robot by 90 degrees clockwise (turn right); the robot stays in the current cell. For example, if the robot currently at coordinate (0, 0) and faces east, it will still be in coordinate (0, 0) but now faces south after this action. Such action consumes 2 seconds.

Give an algorithm that Steven can use to upload to his robot so that his robot can move from coordinate (0, 0) to coordinate $(m-1, n-1)$ using the minimum amount of time (you can assume there is at least 1 such path)! Of course, Steven's robot cannot go outside the $m \times n$ grid and cannot move to a blocked cell throughout the execution of the program. When the robot stops at coordinate $(m-1, n-1)$, it can face any direction.

On the sample grid above, the best robot path is as indicated below with a total time of 29 seconds, i.e. move forward 6 times (18 seconds), turn right (2 seconds), and then finally move forward 3 times (9 seconds). On another sample grid below, the best robot path is as indicated below with a total time of 20 seconds, i.e. turn right (2 seconds), move forward 2 times (6 seconds), turn right 3 times so that the robot faces south $\rightarrow$ west $\rightarrow$ north $\rightarrow$ east (6 seconds), and finally move forward 2 times (6 seconds).

```
. # #
. # #
. . .
```

Convert the following into a graph problem by answering the following questions.

Problem 3.a. What do the vertices and the edges of your graph represent?

Solution: Vertices = (row, col, direction). Many students omit the important parameter: 'direction' and thus their solutions become haywire. There are two types of edges:

- For moving forward: Connect two '.' vertices in the grid if they share a common border and face the same side. Assign weight 3 for such edges.
- For turning right: Connect two vertices in the grid if both are at the same coordinate but with different clockwise direction (i.e. east to south, south to west, west to north, and north to east). Assign weight 2 for such edges.

Problem 3.b. What is the upper bound of the number of vertices and edges in your graph?

Solution: We can obtain the upper bound more simply by considering no obstacles in the grid. In this case, there are $4mn$ possible vertices and up to $2 \times 4mn = 8mn$ possible edges. Thus, $V = O(mn)$ and $E = O(mn)$.

Problem 3.c. What is the graph problem that you want to solve?

Solution: Single-Source Shortest Paths on weighted graph, but this graph is a State-Space graph. The source is vertex (0, 0, East). The destination is vertex ($n-1$, $n-1$, any direction). This is not a DAG since there are cycles:

- Turning right 4 times is cyclic, e.g. (0,0,E) $\rightarrow$ (0,0,S) $\rightarrow$ (0,0,W) $\rightarrow$ (0,0,E)
- Repeated moving forward and turn is also cyclic: (0,0,E) $\rightarrow$ (0,1,E) $\rightarrow$ (0,1,S) $\rightarrow$ (0,1,W) $\rightarrow$ (0,0,W) $\rightarrow$ (0,0,N) $\rightarrow$ (0,0,E)

Problem 3.d. What is the most appropriate graph algorithm to solve this problem and it's time complexity? (Note: What should the terms in the time complexity be expressed as?)

Solution: Modified Dijkstra (can be better than original Dijkstra for cases where E < $O(mn)$). Run Dijkstra's from the source (0, 0, East), then check the shortest path cost to each of the 4 possible ending vertices and return the minimum one. The time complexity in the worst case is O(mn log(mn)).

Note: *It is possible to actually solve this problem with a time complexity of $O(V+E)$ by changing the way you model the graph. The hint here is that the edge weights of each graph in the model described above is always 2 or 3.*

Problem 4. Friends meeting up (2015/2016 CS2010 S2 WQ2)

Two friends lives in cities A and B respectively, and they want to meet up. In order to convenience the both of them, they come up with a strategy to meet up at a city C, where C satisfies the following 2 conditions:

1. The sum of their total travelled distance to C is the smallest.
2. The absolute difference between the distance travelled by each person is minimized.

Note that the **first condition is more important than the second**. Come up with an efficient algorithm to find the city they should meet up in. If there are multiple candidate cities which satisfy the above criteria just output one of them. You can assume that there exist at least 1 such city C and also the roads and cities can be represented as an undirected connected weighted graph, where the edge weight is the distance between the connected cities.

Solution: Run original/modified Dijkstra (same time complexity here since connected graph) using A and B as source vertex. You will have two shortest path distance arrays, one for A (D_A) and another for B (D_B). Now go through each vertex C and find cost of shortest path from A to C and B to C by checking their respective arrays (which can be done in constant time). Calculate both total distance travelled ($D_A[C]+D_B[C]$) and absolute difference in distance travelled ($|D_A[C] - D_B[C]|$). Maintain 2 variables `shortestTotalDist` and `smallestDiff` to represent the shortest total distance and smallest absolute difference in distance travelled found so far. If a shorter total distance is found, update city C to be the current vertex and update `shortestTotalDist` and `smallestDiff`. If the total distance for current city is equal to `shortestTotalDist`, compare the absolute difference in distance travelled to smallestDiff and update as necessary. After going through all vertices, a valid city C must be found. Time taken is $O((E+V)\log V)$ for running Dijkstra 2 times and then $O(V)$ for going through all the vertices. Thus total time complexity is $O((E+V)\log V)$.

Problem 5. Multiple-Choice Questions

Problem 5.a. Which of the following properties of a Binary Search Tree (BST) is false?

a) The right descendants of the root will contain keys larger than the key of the root.

b) Every node of a BST will have two child nodes..

c) The time complexity of inserting into a BST is $O(n)$ time.

d) The left child of the root has a key that is smaller than the key of the root.

Solution: Option (b) is false. A BST will always have at most two child nodes, but it may have only one child node, or be a leaf node with no child nodes.

Problem 5.b. Consider a connected, undirected graph G which can have two or more cycles, and a vertex X in the graph. Which of the following statements is true?

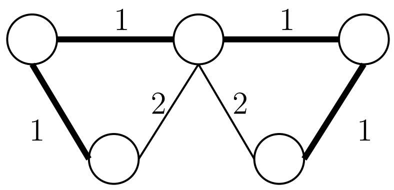
a) G will always have more than one possible minimum spanning tree.

b) The shortest paths from X to all other vertices are *always* unique.

c) We can obtain a spanning tree of G in $O(V+E)$ time.

d) The Bellman-Ford algorithm applied on G from X will never produce the correct shortest paths.

Solution: Option (c). (a) is false because a non-unique MST can occur when there exists a cycle with non-unique maximum weight edge or a cut of the graph where the cut-set has non-unique minimum weight edge, independent of the number of cycles. An example is shown in the diagram below. (b) is false because there can be multiple shortest path from a source to the same vertex (from Q1(b)). (d) is false because Bellman-Ford can work if there are no negative cycles and the number of cycles does not matter.



Problem 5.c. Mark has designed a graph-based game. He will first create a directed unweighted graph with V vertices and E edges. He will then pick a source vertex s, a destination vertex d, and some number K. He will now start from s and place K on s. If s has M outgoing edges, he will divide K by M (integer division), and so each neighbor of s will have the value K/M placed on them. This will continue for each vertex with a value placed on them and will stop if d is reached, or the number K' placed on the vertex v is such that $K'/(\text{number of outgoing edges of v}) == 0$. Mark wants to know what is the minimum value of K so that based on his game he can reach d. You can assume that there is always a path from x to y.

The best algorithm to find the minimum value of K is equivalent to

a) solving a MST problem with time complexity $O(E \log V)$

b) solving a SSSP problem with time complexity $O((V+E)\log V)$

c) solving a SSSP problem with time complexity $O(VE)$

d) solving a SSSP problem with time complexity $O(V+E)$

e) modelling the graph and computing all possible paths from s to d and find the minimum cost path with time complexity $O(V!)$

Solution: Option (b). Simply transform the graph so that the weight of each outgoing edge of a vertex v is weighed by the number of outgoing edges from v. Then it is simply Dijkstra from source vertex s, and instead of addition of the edge weights you perform multiplication. The cost of the shortest path from s to d is the minimum amount that K needs to be.