CS2040

Tutorial 11: Shortest Paths (II)

Nicholas **Russell** Saerang (russellsaerang@u.nus.edu)

Floyd-Warshall

```
for k in {0, …, V-1}
    for i in {0, …, V-1}
        for j in {0, …, V-1}
            D[i][j] = Math.min(D[i][j], D[i][k] + D[k][j])
```

- Can I shuffle the order of the vertices?
  e.g. for j in {V-1, 0, 1, 2, …, V-2}

- Can I shuffle the order of the iteration loop?
  e.g. for j, for i, then for k

Yes for the first one, no for the second one!
https://www.quora.com/Why-is-the-order-of-the-loops-in-Floyd-Warshall-algorithm-important-to-its-correctness

:D

TUTORIAL QUESTIONS!

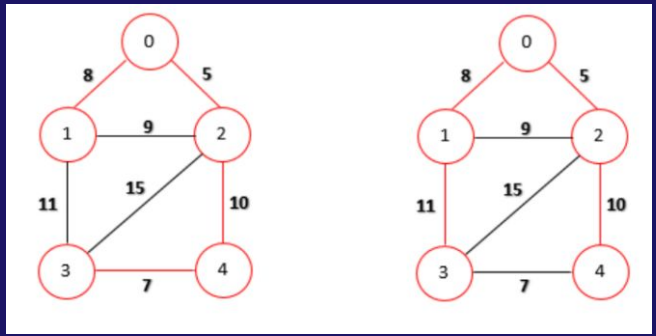
01

TRUE OR FALSE?

Question 1a

If an 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.

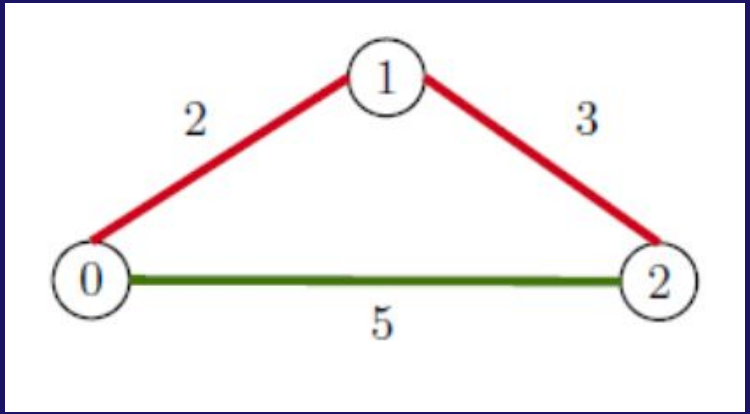
Answer:
False. The minimum spanning tree and shortest path spanning tree can be different. In fact, changing the source you start from for Prim's should result in the same minimum spanning tree, but you may get different spanning trees for different sources for shortest paths.



Question 1b

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.

Answer:
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.



Question 1c

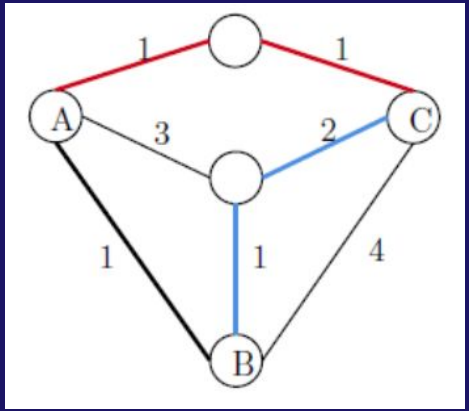
A connected, undirected graph will always form a shortest path tree with V - 1 edges.

Answer:
True, because the tree formed is also a spanning tree.

Question 1d

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.

Answer:
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.



Question 1e

The algorithm below will 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[]
```

Answer:
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.

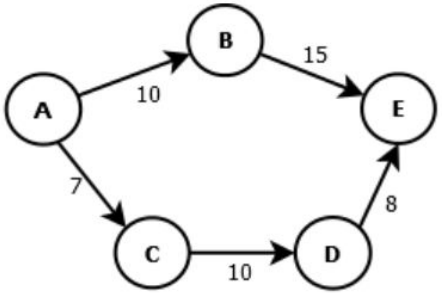
The time complexity of this algorithm is $O(V^2)$.

02

PROMOTER

Promoter

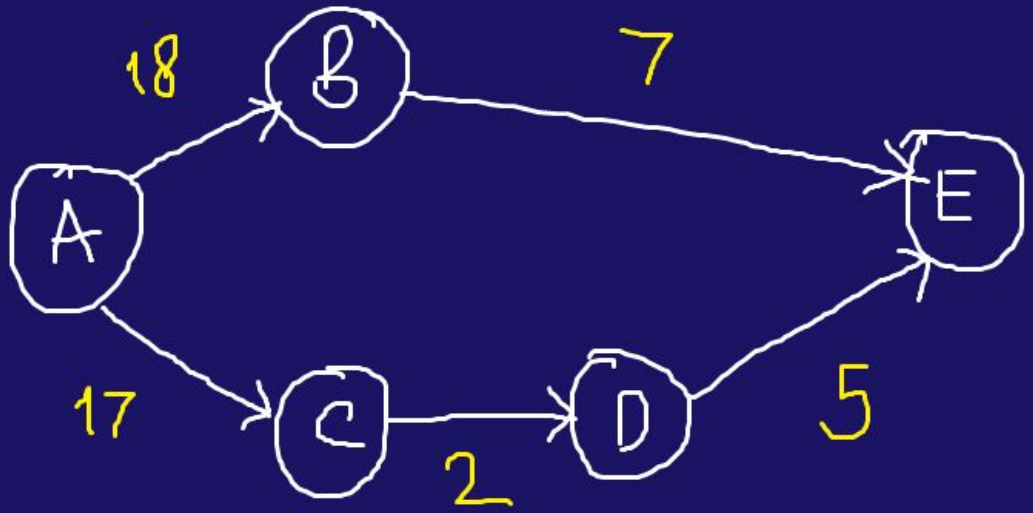
Abridged problem description:
A salesman needs to go from city A to city B such that it has the shortest distance and also passes through the fewest cities because there is a toll fee to pass a city (toll fee is the same for every city). What is the running time of your algorithm? You may assume there is a way to get from any city A to any city B.



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.

Promoter

Did you think of this?
Add 1 to every edge weight and run SSSP as usual.
Of course we can find a counterexample for that!



Promoter

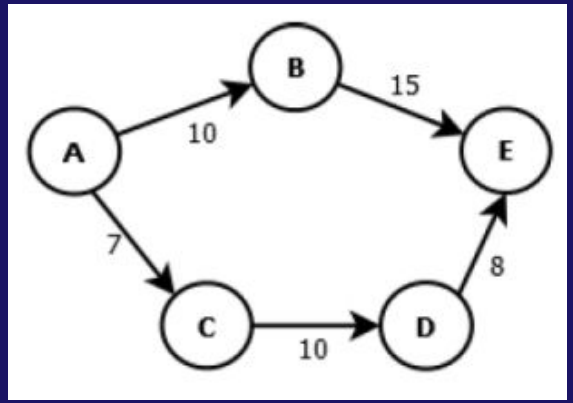
Answer:
Modify the priority queue of Dijkstra's algorithm (either original or modified can be used here; for the sample solution we show the modification to original Dijkstra). From storing pairs (D[v], v), we store triplets (D[v], H[v], v), where H[v] stores how many hops or vertices have been used in the shortest path so far.

Let's see how this algorithm works on the sample on the next slide.

Promoter

1. PQ = {(0, 1, A), (∞, ∞, B), (∞, ∞, C), (∞, ∞, D), (∞, ∞, E)}. Process (0, 1, A).
2. PQ = {(7, 2, C), (10, 2, B), (∞, ∞, D), (∞, ∞, E)}. Process (7, 2, C).
3. PQ = {(10, 2, B), (17, 3, D), (∞, ∞, 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.

Complexity: O((V + E) log V)



Dijkstra+

If the given graph contains additional parameters, then ensure that you account for all the additional parameters.
Parameters can be split into two types:

1. Those that affect your edge relaxation (e.g. whether you can pass through a certain edge)
2. Those that you also want to minimise as part of your shortest distance (e.g. this question - after minimizing shortest distance, we want to minimize hops)

For both types of parameters, you may need to add them to the priority queue, so that you know the values of the additional parameters for a given shortest distance.

Dijkstra+

Reminder that the following is the implementation of modified Dijkstra which is very customizable in terms of how the priority queue orders the elements and how the relaxation is done.

```
Let dist[v] store the shortest distance to vertex v
Let PQ be a min priority queue storing (dist, vertex) pairs

PQ.enqueue((0, s)) // store pair of (dist[u], u)
while PQ is not empty // order: increasing dist[u]
    (d, u) ← PQ.dequeue()
    if d == dist[u] // important check, lazy DS
        for each vertex v adjacent to u
            if dist[v] > dist[u] + w(u, v) // can relax
                dist[v] = dist[u] + w(u, v) // relax
                PQ.enqueue((dist[v], v)) // (re)enqueue this
```

03

LEGO MINDSTORMS EV3

Lego Mindstorms EV3

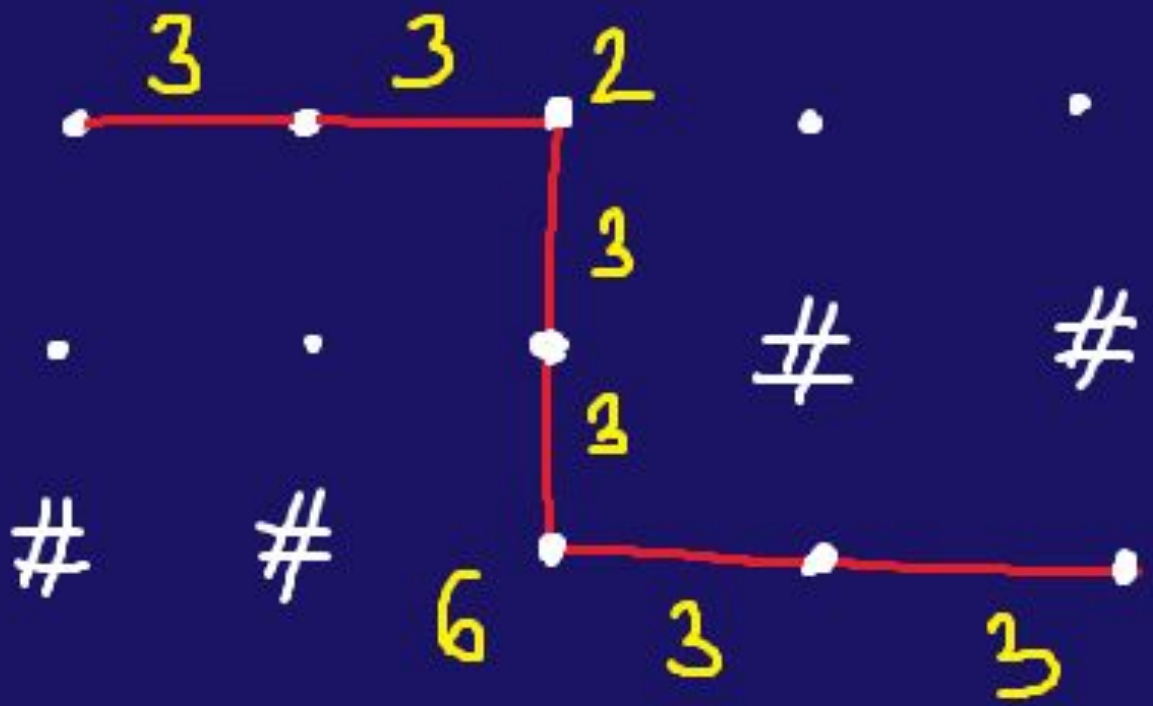
(2013/2014 CS2010 S1 WQ2)
Abridged problem description:
Steven has an m x n grid. '.' means passable cells and '#' means blocked cells. Steven's robot is initially at coordinate (0, 0), the top-left corner of the grid, and faces east.

There are two possible moves:

1. Move forward by one cell according to its 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 to find the shortest time for the robot to go from (0,0) to (m – 1, n – 1). The robot can face any direction at the destination and it cannot pass through blocked cells or go outside m x n grid.

Lego Mindstorms EV3

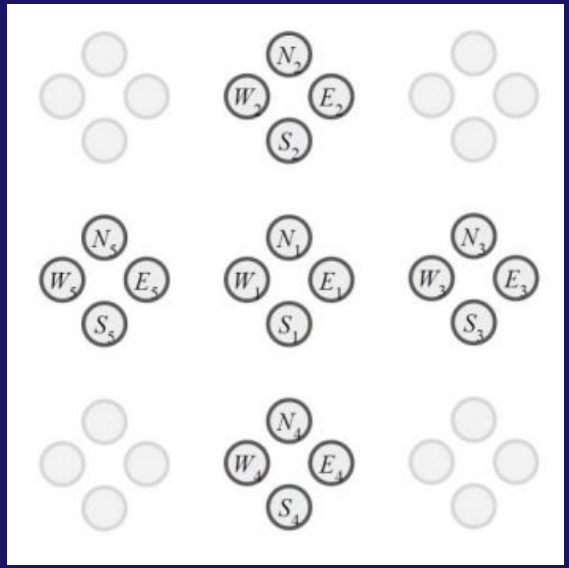


Thinking process

(Q3a) What do the vertices and edges in the graph represent?

Vertices: (row, col, direction)
Edges:

- 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.
- 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.



Thinking process

(Q3b) What is the upper bound of the number of vertices and edges in your graph?

Vertices: 4mn → O(mn)
Edges: up to 2 x 4mn = 8mn → O(mn)

(Q3c) Which specific graph problem do we want to solve?

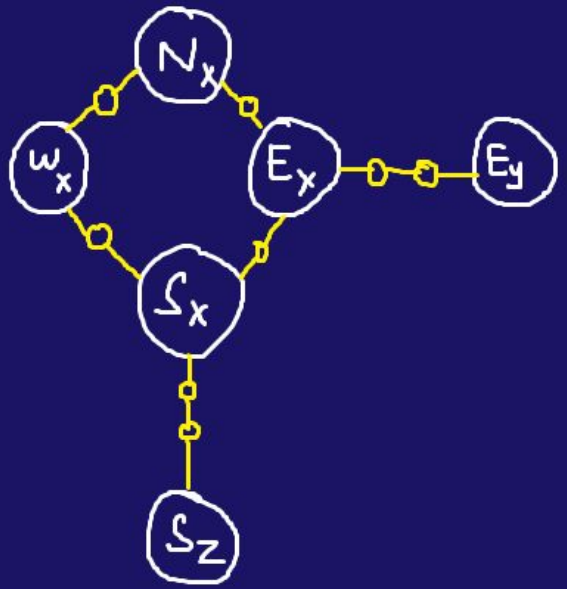
SSSP on a directed graph, but there is a cycle (e.g. rotate 4 times, move forward and turn)
This graph is called a state-space graph.

(Q3d) How can we elaborate more on what algorithm should we use?

Run modified Dijkstra from (0, 0, East) and check the shortest path from that to any of the four possible ending vertices, return the minimum one.
The time complexity is O(mn log(mn)).

Alternative solution

(Q3d) Alternative solution (self-solve)
Note that the edge weights are limited (either 2 or 3).
We can solve this in O(mn) time! How?



04

FRIENDS MEETING UP

Friends Meeting Up

Abridged problem description:
Two friends from city A and B respectively, wants to meet at city C, where:

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.

The first condition is prioritized than the second. 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.

Find that city C.

Friends Meeting Up

Answer: (Meet in the Middle technique)
Run original/modified Dijsktra two times, one from A and one from B. This part will in $O((V + E) \log V)$.

Suppose we store the shortest path estimates in two arrays D_A and D_B. Maintain two variables shortestTotal and smallestDiff, both initially set to INF.

For every vertex k:
- if $D_A[k] + D_B[k] <$ shortestTotal, set k as the best answer so far. Update shortestTotal to $D_A[k] + D_B[k]$ and smallestDiff to $|D_A[k] - D_B[k]|$.
- else if $D_A[k] + D_B[k] ==$ shortestTotal but $|D_A[k] - D_B[k]| <$ smallestDiff, set k as the best answer so far and update smallestDiff to $|D_A[k] - D_B[k]|$.

The time complexity for this part is $O(V)$.

In the end, return k as the final city C. Complexity is $O((V + E) \log V)$.

05

MCQ

Question 5a

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.

Answer: (b).
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.

Question 5b

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?

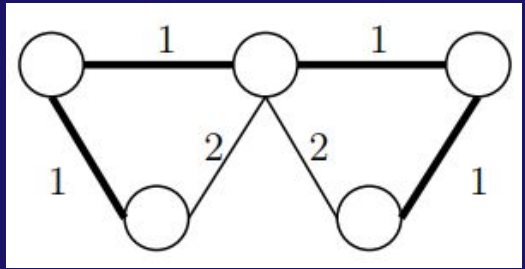
a. G will always have more than one possible minimum spanning tree.
b. The shortest paths from X to all other vertices are always unique.
c. We can obtain a spanning tree of G in O(V + E) time.
d. The Bellman-Ford algorithm applied on G from X will never produce the correct shortest paths.

Question 5b

Answer: (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 (see Q1(b)).

(d) is false because Bellman-Ford can work if there are no negative cycles, and the number of cycles does not matter.



Question 5c

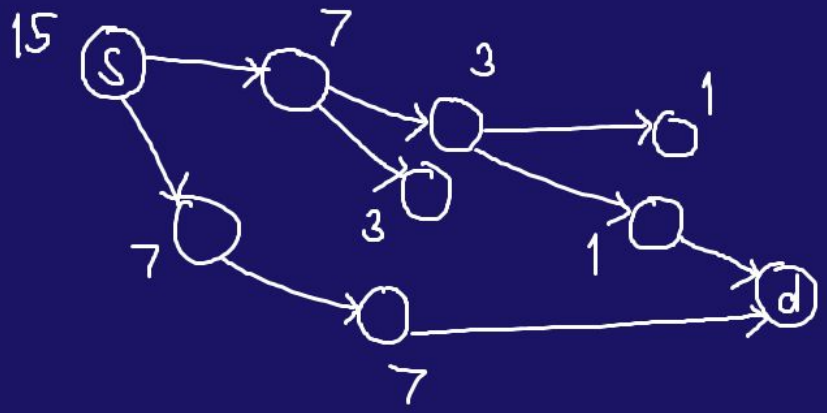
Mark creates a directed unweighted graph with V vertices and E edges. He will then pick a source vertex s, a destination vertex d, and some number K. He will now start from s and place K on s. If s has M outgoing edges, he will divide K by M (integer division), and so each neighbor of s will have the 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'/(number of outgoing edges of v) == 0.

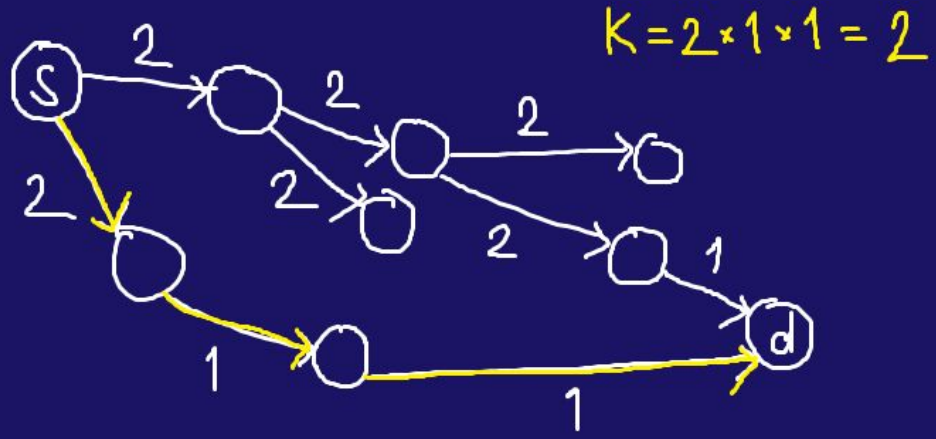
Mark wants to know what is the minimum value of K so that based on his game he can reach d. You can assume that there is always a path from x to y. The best algorithm to find the minimum value of K is equivalent to

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!)

Question 5c

The answer is (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 (remember this can still be done by the log transformation!). The cost of the shortest path from s to d is the minimum amount that K needs to be.





THANK YOU!

(for the past 13 weeks <3)