Henzi Kou

CIS 315: Intermediate Algorithms

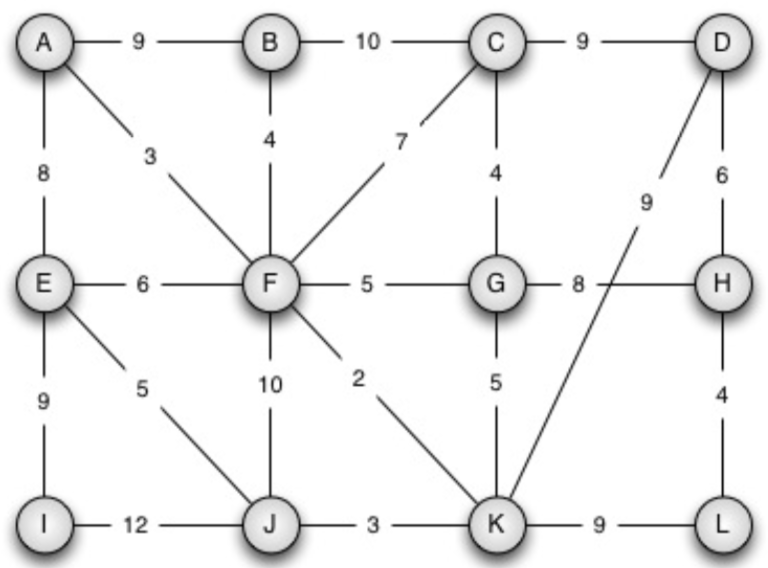
Christopher Wilson

28 April 2019

Assignment 3

1. (a) Use Kruskal’s method.

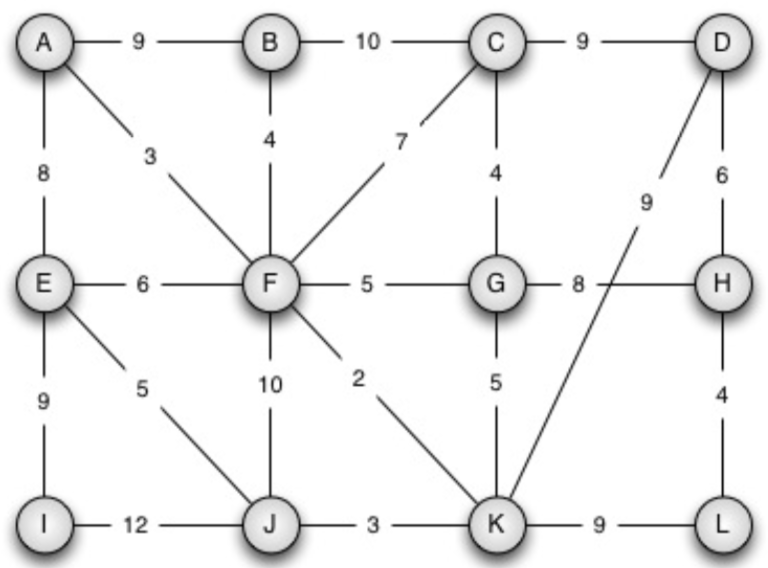
{(f, k), (a, f), (j, k), (b, f), (c, g), (h, l), (f, g), (e, j), (d, h), (g, h), (e, i)}



1. (b) Use Prim’s method.

Edges: {(a, f), (f, k), (j, k), (b, f), (f, g), (c, g), (e, j), (g, h), (h, l), (d, h), (e, i)}

Vertices: {a, f, k, j, b, g, c, e, h, l, d, i}



1. We are given a weighted graph G = (V,E) with weights given by W. The nodes represent cities and the edges are (positive) distances between the cities. We want to get a car that can travel from a start node s and reach all other cities. Gas can be purchased at any city, and the gas-tank capacity needed to travel between cities u and v is the distance W[u,v]. Give an algorithm to determine the minimum gas-tank capacity required of a car that can travel from s to any other city.

Assuming graph G is a weighted undirected graph (since roads between cities go both ways), an efficient algorithm to solve this problem is to adapt Prim’s method by taking the minimum max edge weight of a path. To do this we can create a temporary value holder for the minimum capacity, *min\_cap*. The first edge that we traverse will be set to the value of *min\_cap*. As we perform the relaxation operations on each edge, we will check for the maximum edge of the minimum spanning tree that is appended into it (as stated in CLRS pg. 636). If the current *min\_cap* is not the max weight value in the minimum spanning tree, then update *min\_cap* to the maximum weight value within the minimum spanning tree. Otherwise nothing changes. Finally we can return the *min\_cap* value to get the desired value.

1. We are given a graph G = (V,E) where V represents a set of locations and E represents a communications channel between two points. We are also given locations s, t ∈ V , and a reliability function r : V × V → [0, 1]. You need to give an efficient algorithm which will output the reliability of the most reliable path from s to t in G.

For any points u, v ∈ V , r(u, v) is the probability that the communication link (u, v) will not fail: 0 ≤ r(u, v) ≤ 1. Note that if there is a path with two edges, for example, from u to v to w, then the reliability of that path is r(u, v) · r(v, w).

An efficient algorithm to solve this problem is to adapt Dijkstra’s method by modifying the weight values that we assign to each edge and modifying the RELAX method. We want the weights to be because this converts the problem into a SINGLE-SOURCE SHORTEST PATH problem that we have similarly tackled in Assignment 2. We want the weights to be a negative logarithmic value because for all values of such that , then . By making it negative logarithmic, it converts the value from a negative number to a positive number. Thus effectively changing the problem from a minimization problem to maximization problem as desired. We will also modify the RELAX method to take the product of the probabilities to ensure that it is the maximum instead of taking the sum.

1. (Exercise 24.1-3 from CLRS) Given a weighted, directed graph G = (V,E) with no negative-weight cycles, let m be the maximum over all vertices v ∈ V of the minimum number of edges in a shortest path from the source s to v. (Here, the shortest path is by weight, not the number of edges.) Suggest a simple change to the Bellman-Ford algorithm that allows it to terminate in m + 1 passes, even if m is not known in advance.

A modification to the Bellman-Ford algorithm is to implement the RELAX method. After m passes, we can continue to apply the RELAX method because no further changes are made to *d*.Therefore at m + 1 iterations there will still be no modifications made, thus it will terminate.

BELLMAN-FORD(G, w, s)

INITIALIZE-SINGLE-SOURCE(G, s)

changes = true // Boolean to notify any changes

while changes is true

changes = false

for each edge in E

RELAX(u, v, w)

1. (Exercise 25.2-4 from CLRS) As it appears above, the Floyd-Warshall algorithm requires θ(n3) space, since we compute dij(k) for i, j, k = 1, 2, …, n. Show that the following procedure, which simply drops all the superscripts, is correct, and thus only θ(n2) space is required. More specifically, show that O(n2) space can be achieved without sacrificing correctness by dropping the superscripts in the Floyd-Warshall algorithm by computing the distance matrices D(k) in place using a single matrix D.

The procedure is correct because it modifies only the di,j entry each time k iterates and thus it will not modify any other entry. Therefore, during the kth loop, it only modifies and the entry will not be used again until k is incremented. Additionally, any dik entry will not be modified because if it were to be modified, then that means dik is an intermediate vertex, thus a cycle would exist. Hence, superscripts are not needed for the algorithm.