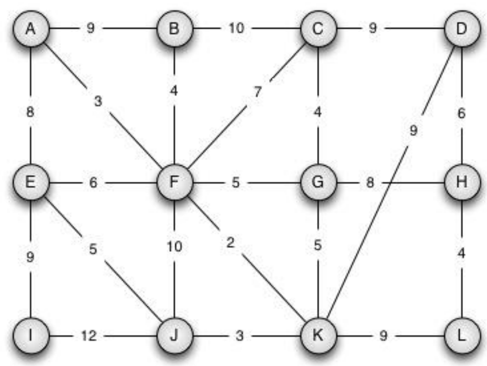
Daniel Willard

CIS 315

Written Assignment 3

Problem 1)



Edges in alphabetical order 22 EDGES

9	8	3	10	4	9	7	4	6	9	6	9	5	5	10	2	8	5	4	12
(A,B)	(A,E)	(A,F)	(B,C)	(B,F)	(C,D)	(C,F)	(C,G)	(D,H)	(D,K)	(E,F)	(E,I)	(E,J)	(F,G)	(F,J)	(F,K)	(G,H)	(G,K)	(H,L)	(I,J)

3	9
(J,K)	(K,L)

Edges in ascending order

2: (F,K)

3: (A,F) (J,K)

4: (B,F) (C,G) (H,L)

5: (E,J) (F,G) (G,K)

6: (D,H) (E,F)

7: (C,F)

8: (A,E) (G,H)

9: (A,B) (C,D) (D,K) (E,I) (K,L)

10: (B,C) (F,J)

12: (I,J)

MST: a subset of the edges of a connected, weighted, undirected graph that connects all the vertices together, without a cycle and minimum possible total edge weight.

PART A Prim's)

STEP 1: select starting node(a)

STEP2: Pick the next neighbor with smallest edge weight of the node subset you have.

12 Steps for each node

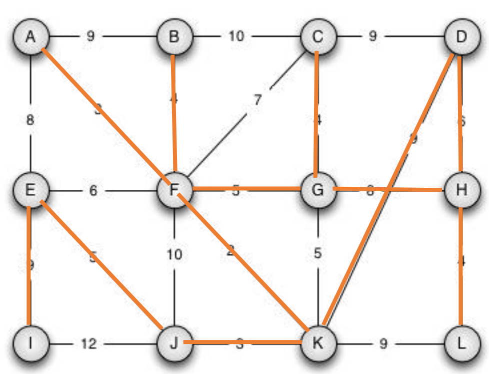
A (Initialization)

1) A F: (A,F) =3
2) A F K: (A,F) (F,K) = 5
3) A F K J: (A,F) (F,K) (K,J) = 8
4) A F K J B: (A,F) (F,K) (K,J) (F,B) = 12
5) A F K J B G: (A,F) (F,K) (K,J) (F,B) (F,G) = 17
6) A F K J B G C: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) = 21
7) A F K J B G C J: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) = 26
8) A F K J B G C J H: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H)= 34
9) A F K J B G C J H L: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H) (H,L)= 38
10) A F K J B G C J H L D: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H) (H,L) (H,D)= 38
11) A F K J B G C J H L D_2: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H) (H,L) (H,D) (D,K) = 47
12) A F K J B G C J H L D_2 I: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H) (H,L) (H,D) (D,K) (E,I) = 58

Final Subset A F K J B G C J H L D_2 I: (A,F) (F,K) (K,J) (F,B) (F,G) (G,C) (J,E) (G,H) (H,L) (H,D) (D,K) (E,I)

Edge weight =58

Exclude Edges: (A,B) (A,E) (B,C) (C,D) (C,F) (E,F) (G,K) (I,J) (K,L)



PART B Kruskal)

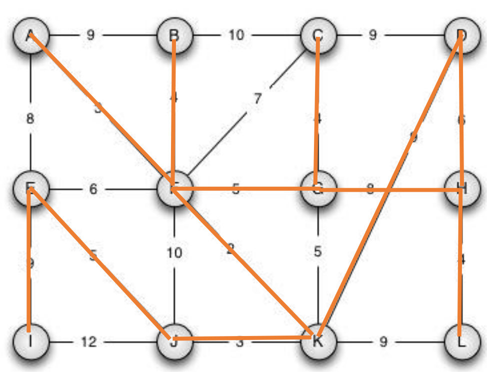
STEP 1: Sort Edges ascending alphabetical order

STEP 2: Construct MST by traversing the edges in the ordered list

22 Steps for each edge

1) (F,K)
2) (F,K) (A,F)
3) (F,K) (A,F) (J,K)
4) (F,K) (A,F) (J,K) (B,F)
5) (F,K) (A,F) (J,K) (B,F) (C,G)
6) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L)
7) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J)
8) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G)
9) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~
10) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H)
11) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~
12) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~
13) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~
14) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H)
15) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~
16) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~
17) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K)
18) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K) (E,I)
19) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K) (E,I) ~~(K,L)~~
20) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K) (E,I) ~~(K,L)~~ ~~(B,C)~~
21) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K) (E,I) ~~(K,L)~~ ~~(B,C)~~ ~~(F,J)~~
22) (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) ~~(G,K)~~ (D,H) ~~(E,F)~~ ~~(C,F)~~ ~~(A,E)~~ (G,H) ~~(A,B)~~ ~~(C,D)~~ (D,K) (E,I) ~~(K,L)~~ ~~(B,C)~~ ~~(F,J)~~ ~~(I,J)~~

Final Edge subset order: (F,K) (A,F) (J,K) (B,F) (C,G) (H,L) (E,J) (F,G) (D,H)(G,H) (D,K) (E,I) Total weight= 58



Problem 2)

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 by modifying Edge Relaxation to determine the minimum gas-tank capacity required of a car that can travel from s to any other city. For this problem, just show your modified version of the Edge Relaxation operation or function; do not provide anything else (e.g., pseudocodes of your entire algorithm). [6 points]

```
EdgeRelaxation(u, v, W, V):
#GTC: Gas Tank Capacity array
#P: parent node vertex array
#W: matrix of weights between cities
For v in V:
        For u in V:
                If GTC[v] > GTC[u] + W[u][v]:
                        GTC[v] =  GTC[u] +W[u][v]
                        P[v] = u
```

Problem 3)

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 2 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 2 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).

For this problem, you modify Edge Relaxation. Just show your modified version of the Edge Relaxation operation or function; do not provide anything else (e.g., pseudocodes of your entire algorithm). [6 points]

```
EdgeRelaxation(s, t, V):
#MR: Most reliability array to store path
#P: parent node vertex array
#W: matrix of weights between cities
#where 0 is max success
For s in V:
        For t in V:
                If MR[t] < MR[s] * r(s,t):
                        MR[t] =  MR[s] * r(s,t)
                        P[v] = s
```

Problem 4)

25.2-4

As it appears above, the Floyd-Warshall algorithm requires $\Theta(n^3)$ space, since we compute $d_{ij}^{(k)}$ for $i, j, k = 1, 2, \ldots, n$. Show that the following procedure, which simply drops all the superscripts, is correct, and thus only $\Theta(n^2)$ space is required.

FLOYD-WARSHALL$'(W)$

```
n = W.rows
D = W
for k = 1 to n
    for i = 1 to n
        for j = 1 to n
            d_ij = min(d_ij, d_ik + d_kj)
return D
```

Exercise 25.2-4 from the CLRS textbook.
More specifically, show and explain that O(n_2) 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. [5 points]

Yes, this is possible. The Idea is that instead of the algorithm shown in lab and in class where we have two matrix D(k) and a D(k-1) and we compute the minimum distance between two vertices `i` and `j` via intermediate vertex `k`. Then we store the newly computed distance into matrix D. This Results in 0(n^3) space and time complexity.

We can reduce the space if we realize one key detail, **we can use the initialized matrix D and use that matrix to create the updated matrix D prime in place then store D prime into D after the computation. This works due to the fact that minimum distance between two vertices can be computed by considering the intermediate vertices in a particular order, and this order is not affected by our optimization. We are able to cut out the D(k-1) matrix.**

We can proof this via induction: (not sure if this works?)

Base Case:

K = 1

D(K) = W => D(1) = W which is the initial distance matrix thus we can proceed.

Assume:

The optimization is correct for k = p

D(p) contains the correct distance between all pairs of vertices.

Induction:

In the p+1 iteration the algorithm updates the distance matrix D as fallows

d(i,j)= min(d(i,j), d(i,k) + d(k,j)

this update is equivalent to computing the minimum distance between `i` and `j` via an intermediate vertex `k`

Since the optimization is correct for k = p , D(p)

since the optimization is correct for k=p, D(p) contains the correct distances between all pairs and the update D(p+1) will also contain the correct distance between all pairs of vertices.

Thus by induction the optimization is correct for all k there for it has a time complexity of O(n^3) and space complexity O(n^2).

Another approach to prove this:

Let D(k) be a matrix representing the shortest distances between all pairs of vertices in the graph, using vertices 1 to k as intermediates. The entry D(i,j,k) is the shortest distance between vertices i and j using vertices 1 to k as intermediates.

The algorithm computes the matrix D(n), where n is the number of vertices in the graph, by updating the matrix D(k) for each intermediate vertex k = 1, 2, ..., n-1. The update formula for D(i,j,k+1) is:

D(i,j,k+1) = min(D(i,j,k), D(i,k+1,k) + D(k+1,j,k))

Using the approach where the superscripts are dropped, we compute the matrix D in place by updating the entries of the matrix D(k) to obtain the matrix D(k+1). The updated entry D(i,j) is:

D(i,j) = min(D(i,j), D(i,k+1) + D(k+1,j))

Since the entries of the matrix D(k) are updated in place, the space complexity remains O(n^2), and the algorithm still computes the correct result.

This proof shows that the Floyd-Warshall algorithm can be implemented with a reduced space complexity of O(n^2), without sacrificing correctness, by computing the distance matrices D(k) in place using a single matrix D.