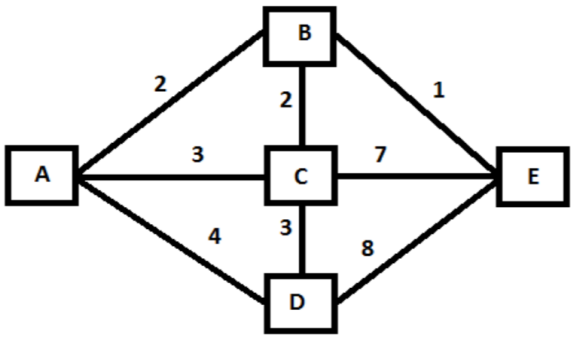
Arafat Sulaiman

Jacob Johansson

Lab 4 rapport

Part 1:

Solution of the problem of Figure 1:



To be able to find out the shortest path from A to E, we need to look through all the iterations of the weights from A-E and all the paths possible from A-E.

Iteration 1:

Node	**Initial value**
A	No direct path from A to E
B	2 (Weight from A to B)
C	3 (Weight from A to C)
D	4 (Weight from A to D)
E	No direct path from A to E

Iteration 2:

Node	Initial value
A	No direct path from A to E
B	1 (Updated weight from B to E) +2 (from previous iteration) =3
C	3 (Remains the same, no changes)
D	4 (Remains the same, no changes)
E	No direct path from A to E

Iteration 3:

Node	Initial value
A	No direct path from A to E
B	3 (Remains the same, no changes)
C	7 (Updated weight from C to E) +3 (from previous iteration) =10
D	4 (Remains the same, no changes)
E	No direct path from A to E

Iteration 4:

Node	Initial value
A	No direct path from A to E
B	3 (Remains the same, no changes)
C	10 (Remains the same, no changes)
D	8 (Updated weight from D to E) +4 (from previous iteration) =12
E	No direct path from A to E

The shortest path to E from the different nodes calculates as following:

Node A: A -> B -> E with a weight of 3.

Node B: B -> E with a weight of 1.

Node C: C -> B -> E with a weight of 3.

Node D: D -> C -> B -> E with a weight of 6.

Node E: E -> E with a weight of 0.

Part 2:

Step 1: Reading the data file:

The code starts by reading the data from the 'city 1.txt' file. Each line of the file represents a connection between two cities and the distance between them. The distances are stored in the distances dictionary, with each city's neighbors and corresponding distances as tuples in a list.

Step 2: setting up the data structures

- **shortest_path**: A dictionary where we store the shortest path from each city to the destination city.
- **optimal_values**: A dictionary where we store the optimal distance from each city to the destination city.

Step 3: Breadth-First Search Algorithm (BFS)

This function implements the BFS algorithm to be able to find the shortest paths and distances from each city to the destination city (F).

Step 4: Populating BFS Loop and the queue

The BFS algorithm starts by initializing a queue with the destination city (F). Then it iterates through the queue to explore the neighbors of each city. For each one of the neighbors, if the shortest path to that neighbor has not been set yet, the algorithm updates the shortest path and optimal distance values and adds the neighbor to the queue.

Step 5: Main Function

The main function reads the graph data, constructs the necessary data structures, and then calls the function to calculate the shortest paths and optimal distances from each city to the destination city (F).

Th results comes out to be as following:

City	Shortest Path	Distance
A	A -> B -> F	5
B	B -> F	3
C	C -> F	7
D	D -> B -> F	8
E	E -> A -> B -> F	7
F	F	0
G	G -> J -> L -> F	9
H	H -> G -> J -> L -> F	11
I	I -> K -> E -> A -> B -> F	15
J	J -> L -> F	4
K	K -> E -> A -> B -> F	10
L	L -> F	2
M	M -> F	7
N	N -> L -> F	6
O	O -> J -> L -> F	7
P	P -> O -> J -> L -> F	8
R	R -> F	3
S	S -> R -> F	7
V	V -> C -> F	16
W	W -> B -> F	7
X	X -> M -> F	8
Y	Y -> F	1
Z	Z -> M -> F	10