Junhyeok Jeong

CS 325 – 400 F2019

November 11, 2019

Homework 5

1)

(a) To find the fastest route from the fire station to each of the intersections, the Dijkstra's algorithm is the best way because the Dijkstra's algorithm is applied to find the shortest path from destination A to B commonly.

Vertex (Destination from G)	Distance	Path
A	12	G-E-D-C-A
B	6	G-H-B
C	8	G-E-D-C
D	5	G-E-D
E	2	G-E
F	8	G-F
G	0	G
H	3	G-H

(b) I think Dijkstra's algorithm is good for finding optimal location must be selected for the fire station such that it minimizes the time to the farthest intersection (vertex). With Dijkstra's algorithm, I should set each intersection as the fire station and then find each shortest path to other intersection. Before find the optimal location, I should set the farthest intersection F from any fire station vertex. After this, I should keep looping on each fire station vertex and then settle down the minimum value of F as the optimal location.

Pseudocode:

```
def Optimal_firesation(Graph):
    optimal_location = null
    minimum_val = unsigned integer
    for i in Graph:
        distance = Dijkstra(Graph, i)
        maximum_shortest_path = max(distance)
        if minimum_val > maximum_shortest_path:
                minimum_val = maximum_shortest_path
                optimal_location = i
        return optimal_location
```

Running time complexity of this algorithm:

Since there are f possible locations for the fire station and r roads, above algorithm takes $\Theta(f^3)$ time because the first loop takes $\Theta(f)$ and calling Dijkstra function takes $\Theta(f^2)$ inside the first loop. Therefore, the asymptotic running time of the algorithm can be finalized to $\Theta(f^3)$ for optimal location of the fire station.

(c) Based on the above algorithm, I can get a shortest path table from each source vertex to other intersections like below:

From Row to Column	A	B	C	D	E	F	G	H	Maximum_value
A	0	18	4	7	10	6	12	15	18
B	18	0	14	11	8	14	6	3	14
C	4	14	0	3	6	2	8	11	14
D	7	11	3	0	3	5	5	8	11
E	10	8	6	3	0	8	2	5	10
F	6	14	2	5	8	0	8	11	14
G	12	6	8	5	2	8	0	3	12
H	15	3	11	8	5	11	3	0	15

As a result of the above table, E is an optimal location of the fire station because it has smallest maximum value of the shortest paths.

(d) If I am available to set two fire stations for minimizing the farthest path from the fire station to other intersection, I can use the same algorithm of part (b) for solving this problem. However, since I can set up two fire stations, I need to make f by f table for recording results. To record the results on the table, I should pick a value maximum value of shortest path for each fire station pair. For example, when I fill the fire station A and B pair on the table, I should pick the maximum value 8 which is shortest path from each station to the intersection.

Pseudocode is twice of part (b) code

Running time complexity of this algorithm:

As combined two part (b)'s algorithm, the running time complexity can be expressed ϴ(f^3) + ϴ(f^3) = ϴ(f^3). However, I need to look up the result table for finding optimal pair of fire stations. Therefore, it needs to be added another iteration to find an optimal pair which is smallest maximum value of shortest path for each fire station. As a result of adding iteration, the running time complexity could be ϴ(f^4)

(e)

Pair of stations	A	B	C	D	E	F	G	H	Minimum value pair
A	X	8	14	11	8	14	6	7	6, A and G
B	8	X	6	7	10	8	12	16	6, B and C
C	14	6	X	11	8	14	6	5	5, C and H
D	11	7	11	X	8	11	7	7	7, 3 pairs
E	8	10	8	8	X	8	10	10	8, 4 pairs
F	14	8	14	11	8	X	6	6	6, 2 pairs
G	6	12	6	7	10	6	X	12	6, 3 pairs
H	7	165	5	7	10	6	12	X	5, H and C

Based on the above table, the optimal pair of the fire stations is C and H

2)

(a) To find an efficient algorithm to solve this problem, I can consider BFS or DFS. However, using BFS is more efficient than DFS on this problem because I need to compare all weights whether it is at least weight W or not on the path to vertex t. BFS allows to search all close possible ways (breadth) before goes to next depth. If BFS process finished before it reaches to t, then it indicates there is no possible way from s to t with over-weight W edges.

(b) The running time of BFS is O(V+E), V sum of vertices queue and E traversed edges.

3)

(a) To solve the wrestler rival matching problem, I choose BFS algorithm because if a vertex is Babyface, then the neighbor vertex should be Heels on a graph G with n vertices(wrestler name) and r edges (rivalry).

Wrestler BFS Pseudocode:

```
class Graph:
    self.graph = defaultdict(list) (ex) from sample input on homework description {Ace :
[Duke,Biggs], Jax : [Duke], Stone : [Biggs, Duke], Biggs : [Jax] }
    def add_edge(self, vertex, neighbor):
        self.graph[vertex].append(neighbor)
        if vertex == first vertex:
            vertex.team = babyface
        if vertex.team = babyface:
            neighbor = heels
        else:
            neighbor = babyface


def wrestler_BFS(graph, start):
    visited = []
    queue = [start]


    while queue:
        node = queue.pop(0)
        if node not in visited:
            visited.append(node)
            neighbors = graph[node]
            if neighbors has both team:
                return impossible
            for i in neighbors:
                queue.append(i)


    return possible
```

(b) The running time of above BFS algorithm is O(n+r) because it takes O(n) to assign each vertex as Babyface or Heels and takes O(r) to check edges.