BLG336E Homework-2

Constructing graph using adjacency matrix and implementing different shortest path algorithms

Tolga İnkaya

Student ID: 150140715

Faculty Assistant: Şeymanur Aktı

ITU

April 22, 2020, 23.59

Contents

Chapter 1

Introduction

1.1 Background

Graph which was implemented for this assignment constructed using adjacency matrix. It is easier than to node representation in terms of implementation, but it has some drawbacks. The most important of one of these is graph search inefficiency. For node implementation, in the worst case graph search complexity is O(V*E), on the contrary, adjacency matrix has 0(V*V) which is most of time more slower than O(V*E). In the shortest path algorithm part, I implemented many different algorithms. My motivation is that not only solving this assignment's problem, but also implementing my own graph library which is implemented using the C++. It will be considered with all details in the next chapters, All algorithms that are implemented: A* Search Algorithm, Bellman Ford Algorithm, Dijkstra, Dijkstra Modified, Floyd Warshall Algorithm, KSP, Random Walk Algorithm, Recursive Approach, Traveling Salesman Problem Approach, and most important one for this homework is Yen's Kth Shortest Path Algorithm.

1.2 Problem

In this Assignment, we need to find shortest path through the home to destination and destination to home. The main problem is that two visitors are in the same place at the same time. Moreover, 30 minutes waiting time needs to be some extra searching and updating if necessary.

1.3 The Goal

Fist main goal is finding true paths in case of given distance and nodes. To be honest, it is easy to find shortest way to checking every possible paths in the graph network. This kind of approach is called brute force and I also implemented recursive algorithm which can be seen in the Algorithms folder to find all possible paths. However it is not to best way to find shortest path in case of time complexity. To handle it, it is also implemented many other more efficient algorithms. However, in this assignment I used Dijkstra modified algorithm and Yen's Kth Path Algorithm to find efficient and ture solution.

Chapter 2

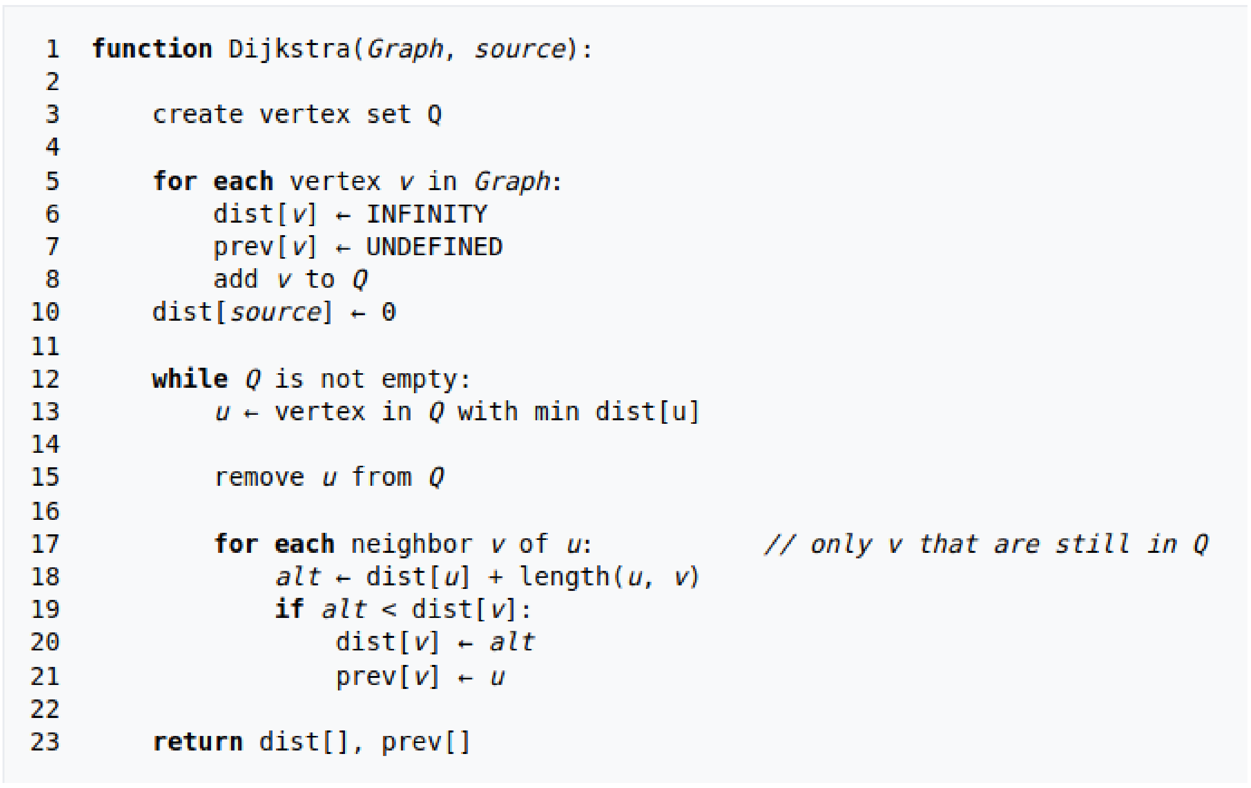
Shortest Path Algorithms

2.1 Overview

In this part I don't mention all algorithms with details. In short, A* search algorithm is kind of modified BFS and Dijkstra Algorithm using some heuristic approach. This is efficient and useful algorithm for the grid systems especially in the game programming. I also developed many games with this algorithm. However for this assignment, it is hard to find appropriate heuristic function, for this reason I don't guaranteed this algorithm I implemented to find best solution for this assignment. Secondly I implemented Bellman Ford and Floyd Warshall Algorithms. Bellman Ford Algorithm implemented for the negative weight edges, because Dijkstra's Algorithm can't calculate shortest path with negative edge. Floyd Warshall Algorithm uses dynamic programming optimization. It is like recursion but unlike the recursion it doesn't visit same route again and again, and it stored already computed and visited paths. Thanks to this optimization, search algorithm complexity is O(V*V*V) which is clearly more efficient than recursive algorithm. The other algoritms such as Random Walk and KSP are not related to this assignment. We need to focused on Dijkstra's Algorithm. For this Assignment I also implemented Yen's Algorithm to find kth shortest path. I research and read many paper and article about it. Most important are can be found in Article folder.

2.2 Implementation of Algorithms

Dijkstra is clever way to find shortest path in the graph. It is also similar to BFS, but is is not visit to unnecessary edges, for this reason it is also higly similar to A* algorithm. Moreover, Dijkstra's algorithm is very similar to Prim's algorithm for minimum spanning tree. Like Prim's MST, we generate a SPT (shortest path tree) with given source as root. We maintain two sets, one set contains vertices included in shortest path tree, other set includes vertices not yet included in shortest path tree. At every step of the algorithm, we find a vertex which is in the other set (set of not yet included) and has a minimum distance from the source.



```
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[]
```

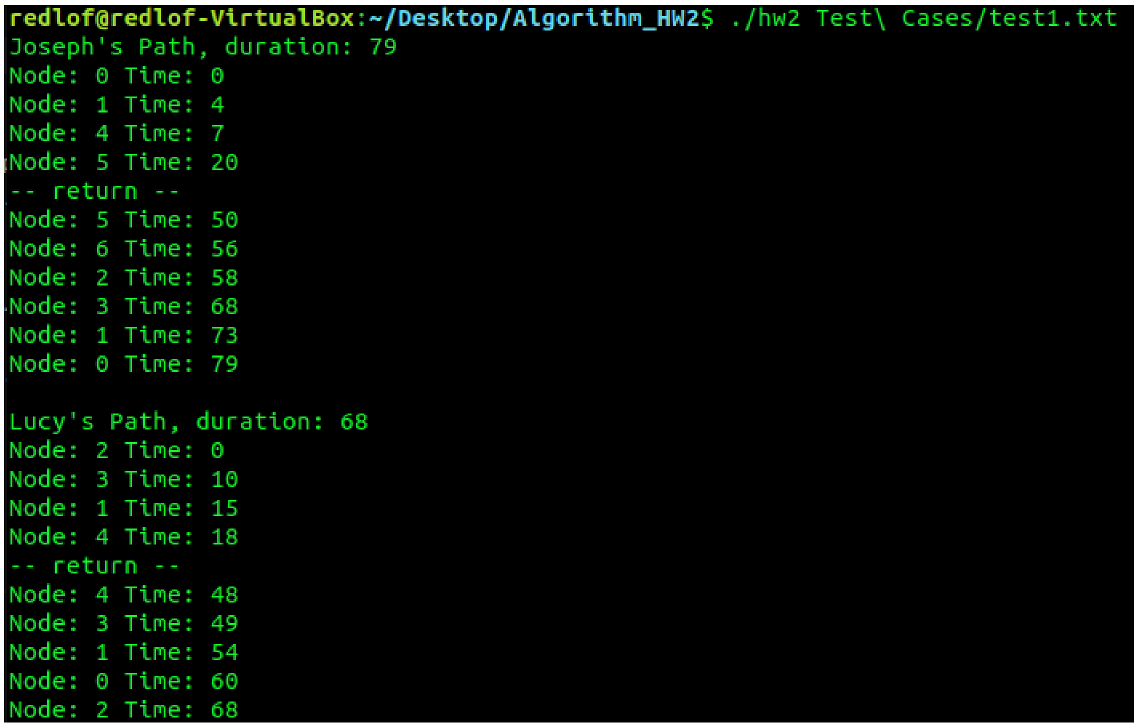
Figure 2.1: Pseudocode

2.3 Analysis of Algorithms

The time complexity of the Dijkstra's Algorithm for my implementation using the graph with adjacency matrix looks O(V*V) as there are two nested while loops. However using the some optimization, If we take a closer look, we can observe that the statements in inner loop are executed O(V+E) times (similar to BFS). The inner loop has operation which takes O(LogV) time. So overall time complexity is O(E+V)*O(LogV) which is O((E+V)*LogV) = O(ELogV). In addition, Time complexity can be reduced to O(E + VLogV) using Fibonacci Heap instead of the using Binary Heap for Priority Queue.

Chapter 3

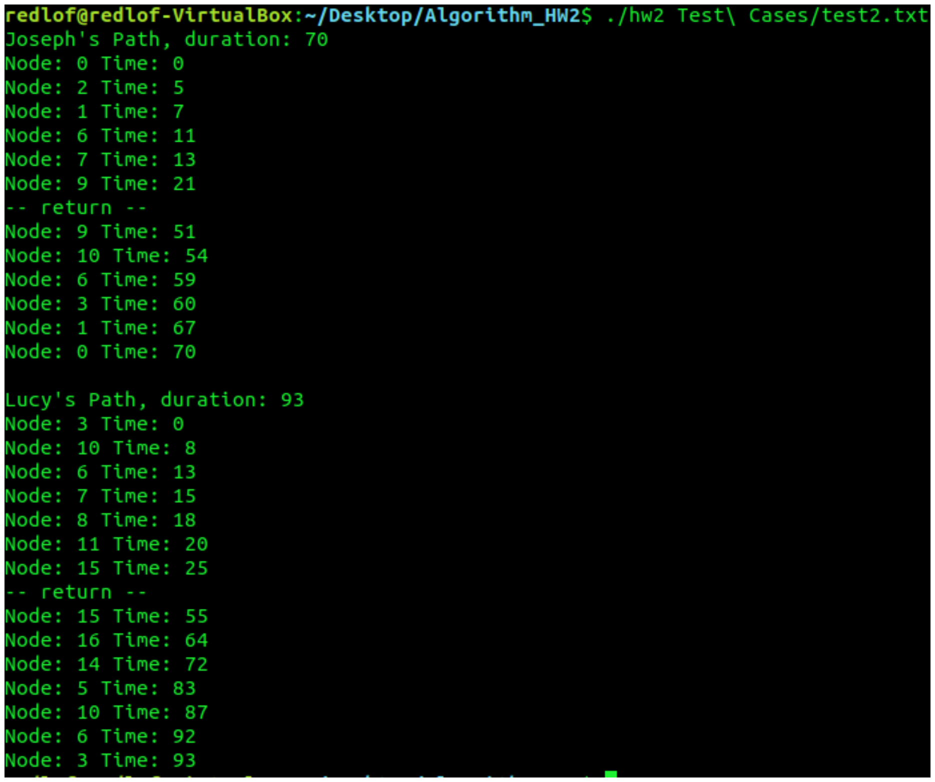
Sample Test Results



```
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$ ./hw2 Test\ Cases/test1.txt
Joseph's Path, duration: 79
Node: 0 Time: 0
Node: 1 Time: 4
Node: 4 Time: 7
Node: 5 Time: 20
-- return --
Node: 5 Time: 50
Node: 6 Time: 56
Node: 2 Time: 58
Node: 3 Time: 68
Node: 1 Time: 73
Node: 0 Time: 79

Lucy's Path, duration: 68
Node: 2 Time: 0
Node: 3 Time: 10
Node: 1 Time: 15
Node: 4 Time: 18
-- return --
Node: 4 Time: 48
Node: 3 Time: 49
Node: 1 Time: 54
Node: 0 Time: 60
Node: 2 Time: 68
```

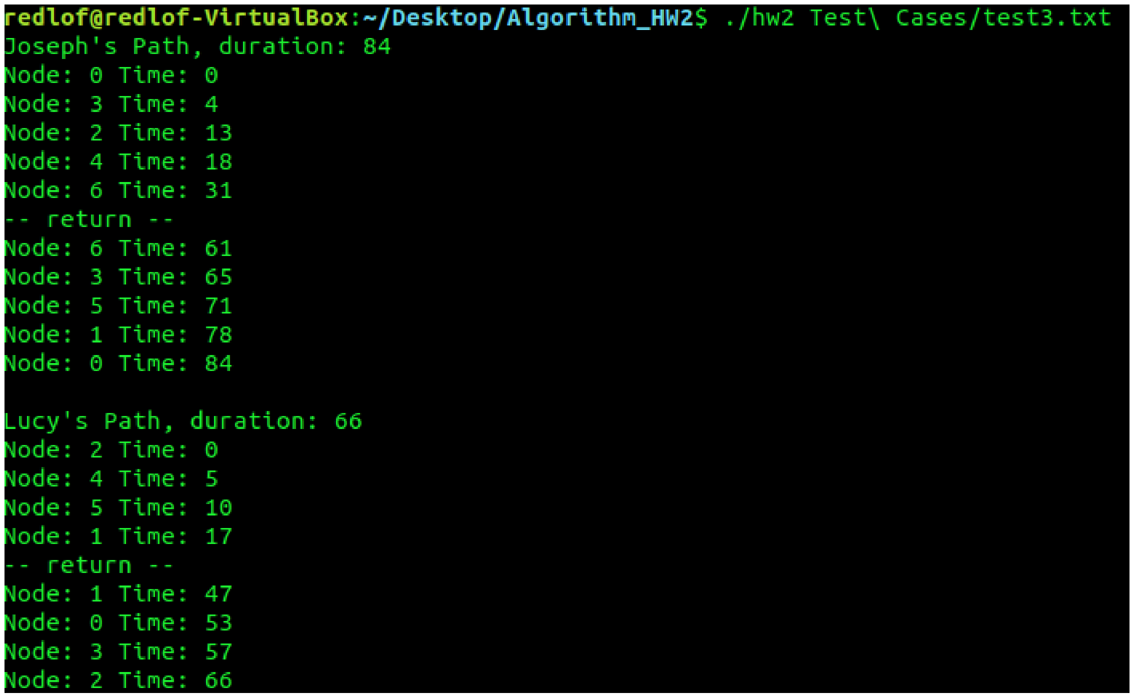
Figure 3.1



```
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$ ./hw2 Test\ Cases/test2.txt
Joseph's Path, duration: 70
Node: 0 Time: 0
Node: 2 Time: 5
Node: 1 Time: 7
Node: 6 Time: 11
Node: 7 Time: 13
Node: 9 Time: 21
-- return --
Node: 9 Time: 51
Node: 10 Time: 54
Node: 6 Time: 59
Node: 3 Time: 60
Node: 1 Time: 67
Node: 0 Time: 70

Lucy's Path, duration: 93
Node: 3 Time: 0
Node: 10 Time: 8
Node: 6 Time: 13
Node: 7 Time: 15
Node: 8 Time: 18
Node: 11 Time: 20
Node: 15 Time: 25
-- return --
Node: 15 Time: 55
Node: 16 Time: 64
Node: 14 Time: 72
Node: 5 Time: 83
Node: 10 Time: 87
Node: 6 Time: 92
Node: 3 Time: 93
```

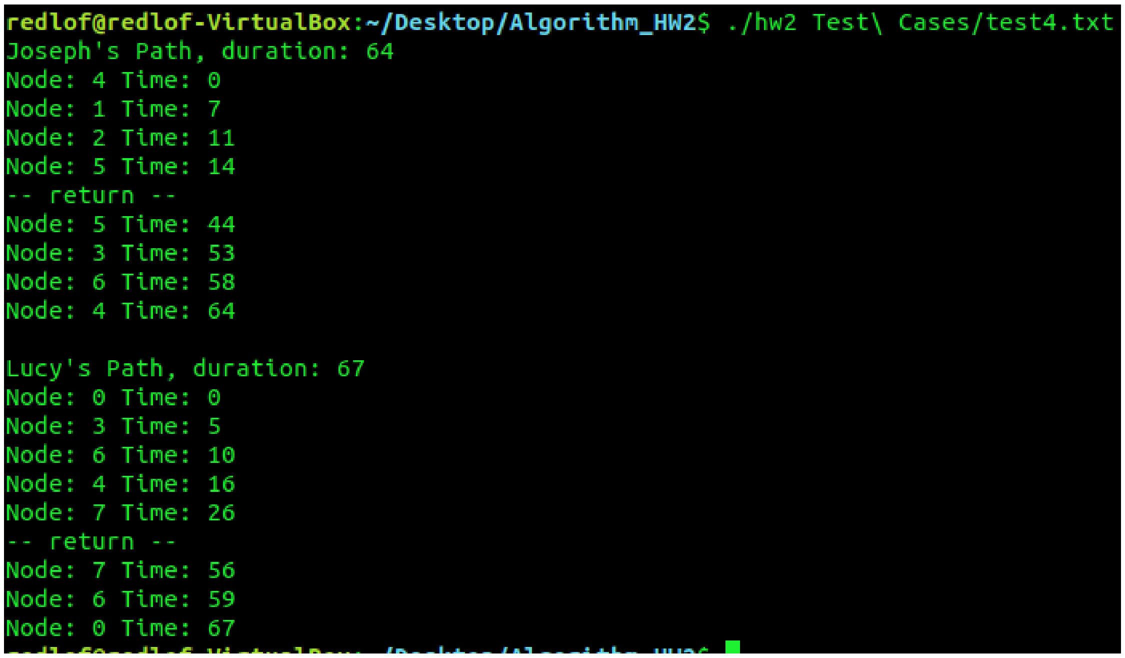
Figure 3.2



```
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$ ./hw2 Test\ Cases/test3.txt
Joseph's Path, duration: 84
Node: 0 Time: 0
Node: 3 Time: 4
Node: 2 Time: 13
Node: 4 Time: 18
Node: 6 Time: 31
-- return --
Node: 6 Time: 61
Node: 3 Time: 65
Node: 5 Time: 71
Node: 1 Time: 78
Node: 0 Time: 84

Lucy's Path, duration: 66
Node: 2 Time: 0
Node: 4 Time: 5
Node: 5 Time: 10
Node: 1 Time: 17
-- return --
Node: 1 Time: 47
Node: 0 Time: 53
Node: 3 Time: 57
Node: 2 Time: 66
```

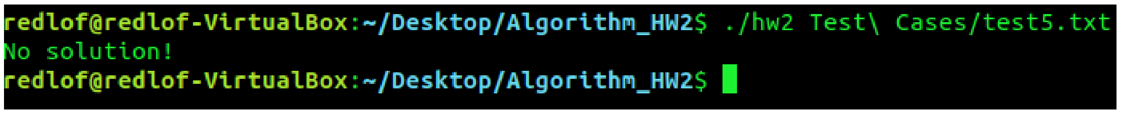
Figure 3.3



```
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$ ./hw2 Test\ Cases/test4.txt
Joseph's Path, duration: 64
Node: 4 Time: 0
Node: 1 Time: 7
Node: 2 Time: 11
Node: 5 Time: 14
-- return --
Node: 5 Time: 44
Node: 3 Time: 53
Node: 6 Time: 58
Node: 4 Time: 64

Lucy's Path, duration: 67
Node: 0 Time: 0
Node: 3 Time: 5
Node: 6 Time: 10
Node: 4 Time: 16
Node: 7 Time: 26
-- return --
Node: 7 Time: 56
Node: 6 Time: 59
Node: 0 Time: 67
```

Figure 3.4



```
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$ ./hw2 Test\ Cases/test5.txt
No solution!
redlof@redlof-VirtualBox:~/Desktop/Algorithm_HW2$
```

Figure 3.5

Bibliography

[1] Husain Aljazzar, Stefan Leue, K∗: A heuristic search algorithm for finding the k shortest paths, Artificial Intelligence, Volume 175, Issue 18, 2011, Pages 2129-2154, ISSN 0004-3702, https://doi.org/10.1016/j.artint.2011.07.003.

[2] Bi Yu Chen, Xiao-Wei Chen, Hui-Ping Chen, William H.K. Lam, Efficient algorithm for finding k shortest paths based on re-optimization technique, Transportation Research Part E: Logistics and Transportation Review, Volume 133, 2020,101819, ISSN 1366-5545, https://doi.org/10.1016/j.tre.2019.11.013.

[3] David Eppstein, Finding the k Shortest Paths, March 31, 1997

[4] Jin Y. Yen, Finding the K Shortest Loopless Paths in a Network Management Science, Vol. 17, No. 11, Theory Series (Jul., 1971), pp. 712-716