Yunus Emre Gürses
2099083

CENG315 ALGORITHMS – THE REPORT

1. Proposed Algorithm & Pseudocode

In the beginning, I have written a function is called 'readInput()' for reading input data from 'the3.inp'. I saved the input data to my various global variables. I have started to search about shortest path algorithms. Most useful algorithm that I found was Bellman Ford algorithm. So, I decided to use it, but I needed to modify a little bit because our graph in the homework was dynamic. After fully understanding the algorithm, I realized that the program should need distance array and parent array for both even and odd cases. shortestPath() function takes these arrays and use it in the algorithm by finding the shortest path. Also, I created two arrays for both odd and even flags. The rooms locked in even times are 0 in the 'evenflag' array and same as odd locked rooms. The rooms which are locked both even and odd times are just removed in the beginning of the program since they are locked all the time.

Instead of finding the shortest path by calling the function once, I have divided the problem. First, I implemented that ammo room number is zero case. Here, shortestPhat() function is called. I sent my parent and distance arrays as parameters. Then, 5'th parameter is the beginning room number of the graph and 6'th parameter is the destination vertex of the graph. Last parameter implies that we are starting from odd time since t=1 in the beginning. After this, shortest path function is called four times with different parameters according to the source and destination vertex.

shortestPath()

It is the function that writes distance arrays and parent arrays. For example, oddparent[2] = 3 means that 2'nd vertex's odd parent is 3'rd vertex and odddistance[2] = 5 means 2'nd vertex has a shortest distance of 5 if it arrives in odd times. To find shortest path, I implemented two if case in the two for loops. One is for 'evendistance' and 'oddparent' and one is for 'odddistance' and 'evenparent'. I checked whether its distance is less or not than the current distance same as in the algorithm. If it is less, I changed the distance and parent so that I found the shortest path. In this part, I modified the algorithm for even and odd locked vertexes. If an odd locked vertex's flag is 0 and checking the odd distance which means arrived in an odd time, its parent and distance is not saved and vice versa. After two loops finding the accurate parents and distances, 'printPath()' is called.

printPath()

This function is to save the all paths in my pathes vector which has type of vector<PathwithDistance>. 'PathwithDistance' is a struct has 'int path_cost', 'vector<int> path', 'int type' and 'vector<int> dir'. While loops in this function is to determine whether a path is found from source to destination or not. All odd and even paths and their total costs are saved to the 'pathes' vector.

I saved all the paths with odd arriving and even arriving cases. According to their costs, index of minimum cost is chosen. To print minimum cost path, the path in convenient index of 'pathes' vector is pushed respectively regarding to index of minimum cost.

Similarly, if number of ammo rooms is one, I evaluated 4 cases for minimum cost paths and chosen the best of them. 1. Best path may not include ammo room, so Doom guy goes necessary rooms without visiting the ammo room. 2. Doom guy goes to the ammo room

before going to the key room. 3. Doom guy goes to the ammo room before going to the scientist room. 4. Doom guy goes to the ammo room before going to the chamber room.

```
shortestPath()
Initialize distance and parent arrays;
Initialize source distance with zero regarding to the odd or even time;
for(total_rooms-1)
        for(corridors.size())
                if(odddistance[u]!=INT_MAX && odddistance[u]+weight < evendistance[v] &&
        evenflag[v]==1)
                        evendistance[v] = odddistance[u] + weight;
                        oddparent[v] = u;
                if(evendistance[u]!=INT_MAX && evendistance[u]+weight < odddistance[v]
                && oddflag[v]==1)
                        odddistance[v] = evendistance[u] + weight;
                        evenparent[v] = u;

Main()
Read input and initialize the flag vectors;
if(ammo_room_number==0)
        Call shortestPath() for all possibilities;
        Find and save all the paths from 1 to chamber room;
        Chose the minimum cost one;
        Push that path to the shortest paths vector;
        Print the output;
else if(ammo_room_number==1)
        Call shortestPath() without visiting the ammo room;
        Find and save all the paths from 1 to chamber room;
        Save the minimum cost one;
        Call shortestPath() for visiting the ammo room before key room;
        Find and save all the paths from 1 to chamber room;
        Save the minimum cost one;
        Call shortestPath() for visiting the ammo room before scientist room;
        Find and save all the paths from 1 to chamber room;
        Save the minimum cost one;
        Call shortestPath() for visiting the ammo room before chamber room;
        Find and save all the paths from 1 to chamber room;
        Save the minimum cost one;
        Again, find the minimum cost path among four minimum paths;
        Print the output;
```

2. Complexity Analysis

Let say E for corridor number and V for room number. My shortest path function has two loops. For ammo room number is 0, it is called five times = 5.(V-1.E) = O(E.V). For ammo room number is 1, it is called 26 times = 26(V-1.E) = O(E.V).