Student Information

Full Name : Beste Burhan
Id Number : 2171395

Proposed Algorithm & Pseudocode

I used Dijkstra Algorithm to find shortest path. However I did some configuration on Dijkstra since I have ammo and locked rooms.
In my main function, firstly i got input file and I stored;

Ammo, room number of chamber, room number of key, room number of scientist–> int

Room numbers of locked rooms in odd time and even time –> vector <int>

Rooms, their adjencies and corridor cost –> vector <vector <pair <int,int> > > (indexes represents room number)

Room number of the rooms which contain ammo –> vector< pair<int,int» (first element of pair stands for room number, second element of pair stands for the amount of ammo)

After that, I called my shortest path algorithm three times. Firstly I called to get the shortest path from 1 to the room that has key, then I called it again to get the shortest path from the room that has key to scientist's room.Finally I called it to get the shortest path from the scientist's room to the chamber. Therefore I could find the path and the ammo amount that I spent.

```
function shortest_path(Graph, source, parents, contains_ammo, path):
for each path element p in path // path that I obtained previous calling(f.e:in first calling:1 3)
  for each element a in contains_ammo //
    if a.roomNumber == p
      a.ammo_amount =0 // set the ammo amount zero if it was used before

for each vertex v in Graph: // Initialization
  cost[v] ← INFINITY // Unknown cost from source to v
 parent[v] ← UNDEFINED // Previous node in optimal path from source
  visited[v] ← FALSE// All rooms initially in Graph(adj) (unvisited nodes)
  odd_time ← [v]=TRUE // initial room is in odd time


count = 0
currentRoom
while count < room number:
  for each p in contains_ammo
    if currentRoom== p.roomNumber and p.ammoAmount>0
      p.ammoAmount*=-1//
  visited[currentRoom]=TRUE
```

```
21    for each pr in adjecencyOfRoom // neighbour to the currentRoom
22       locked=FALSE
23       neighbour = pr.roomNumber //
24       edge = pr.costOfcorridor //
25       if odd_time[currentRoom] is TRUE
26          if currentRoom in oddLockedRoomlist
27             locked =TRUE
28       if odd_time[currentRoom] is FALSE
29          if currentRoom in evenLockedRoomlist
30             locked =TRUE
31       if locked is TRUE
32          continue
33       newCost = cost[currentRoom]+edge
34       for p in contains_ammo
35          if p.roomNumber == neighbour and p.ammoAmount>0
36             newCost = cost[currentRoom]+edge-p.ammoAmount
37       if cost[neighbour] > newCost // relaxation
38          if currentRoom's parent is neighbour
39             visited[neighbour]=false; count−; // to turn back
40             parents[neighbour].push(currentRoom)
41             cost[neighbour]=new_cost
42             oddTime[neighbour]=!oddTime[currentRoom]
43    currentRoom=extractMinimum(cost,visited) // extract room number that has minimum cost
44 return cost

     function extractMinimum(cost,visited)
2 min = INT_MAX;
3 min_index = -1;
4     for each room
5       if !visited[room] and cost[room] < min)
6          min = cost[room];
7          min_index= room;
8 return min_index;
```

Complexity Analysis

extractMinimum operation takes O(V) time. In shortest path algorithm it takes time of O(V^2). Since it looks for edges of each room it takes O(E) time. Generally, Time complexity of shortest path algorithm is O(V^2+E). Since I called this algorithm 3 times it takes 3*O(V^2+E)=O(V^2+E)