İlkyaz Arabacı
2236933

REPORT

1. Proposed Algorithm & Pseudocode

For finding shortest paths in the weighted graph, I modified Floyd-Warshall algorithm. As in the algorithm, I compared all possible paths through the graph between each pair of rooms and stored the distance values in a matrix. But instead of storing the distance between two nodes, I also stored odd distance, even distance, alternative odd distance and alternative even distance values. Alternative distance values are used in ammo calculations. Each cell of the matrix is the Path class, which has also a vector that contains visited rooms for each of these distance values to concatenate the paths at the end.

Until applying ammo calculations to the problem, only odd and even distance values are changing. After the main loop of Floyd-Warshall algorithm, one can see the shortest paths and distances for each pair of nodes.

After the loop, I decrease the alternative distance values of a Path in the matrix by ammo amount in the room, if that room has ammo in it. The problem can be divided into three subparts: shortest from starting node to key, from key to scientist, from scientist to chamber. In the resulting matrix, only the cells that contain the necessary information for these subparts are important. Since only the rooms with ammo are changed according to the ammo amount, the critical cells will not be the final shortest paths.

For the critical cells, I update the alternative paths by changing the alternative distance values in them. For this operation, I calculate the shortest distances between these critical cells and the ones which has ammo in them. This is done with the same idea as in the main loop.

If there are more than one room that contains ammo, then I calculate the shortest distances between that rooms too and consider all of them while updating the alternative paths in the critical cells.

Finally, I choose the most effective path by combining three critical cells. I choose either odd distance, even distance, alternative odd distance or alternative even distance from each of them. If one alternative distance is already chosen, other two can not have alternative path. Because if a room with ammo is revisited, there will be no ammo in it anymore. The validity of path is also checked, by controlling if a locked room is trying to be reached at the time.

Start with initializing the matrix fWarshall with roomCount+1 x roomCount+1, each cell contains a path that has INF odd and even distance.
Add paths and weights to the odd distances of adjacents.

```
void loop() {
  for (k = 1 to roomCount+1) {
    for (i = 1 to roomCount+1) {
      for (j = 1 to roomCount+1) {
        Path first = fWarshall[i][k];
        Path sec = fWarshall[k][j];
        Path target = fWarshall[i][j];

        if (first->oddDistance + sec->oddDistance < target->evenDistance)
              target->evenDistance = first->oddDistance + sec->oddDistance;
              target->evenNodes : first->oddNodes + sec→oddNodes;

        if (first->evenDistance + sec->evenDistance < target->evenDistance)
              target->evenDistance = first->evenDistance + sec->evenDistance;
              target->evenNodes : first-> oddDistance + sec→oddDistance;

        if (first->oddDistance + sec->evenDistance < target->oddDistance)
              target->oddDistance = first->oddDistance + sec→evenDistance;
              target-> oddNodes : first-> oddNodes + sec→oddNodes;

         if (sec->oddDistance + first->evenDistance < target->oddDistance)
```

```
target->oddDistance = sec->oddDistance + first→evenDistance;
   target-> oddNodes : first-> evenNodes + sec→oddNodes;
```

Apply ammo to the columns of the rooms with ammo, meaning change the alternative values of them. ammoAmount[room number] gives the ammo amount in it. Then set only the alternative distances, do not change the odd and even distances, because they are needed at the final part:

```
void setAlternativeDistances() {
  int old_i = -1;
  for (i = 1 to roomCount+1) {
    if (ammoAmount[i] is not equal to 0) {
      if (old_i is equal to -1) {
        updateDistances(1, i, key); //find a path from 1 to key that is shorter because it goes through a room with ammo, which is i
        updateDistances(key, i, scientist);
        updateDistances(scientist, i, chamber);
      }
      else { //if there is more than one room with ammo
        updateDistancesForMoreAmmos(1, i, key, old_i); //first find a shortest path from 1 to i that goes through old_i, then find a
                                                       shortest path from 1 to key that goes through i
        updateDistancesForMoreAmmos(key, i, scientist, old_i);
        updateDistancesForMoreAmmos(scientist, i, chamber, old_i);
      }
      old_i = i;
    }
  }
}

  Path OK = fWarshall[1][key];
  Path KS = fWarshall[key][scientist];
  Path SC = fWarshall[scientist][chamber];
```

After all, combine three paths and select the most effective one which is valid. A valid path can not select more than one alternative path from three subparts and it should consider the locked room information according to the time.

2. Complexity Analysis

Assume that there are R rooms. Floyd-Warshall algorithm works on an RxR adjacency matrix. There is a comparison and insertion operation for each of the cells. There are R^2 cells. These operation should be performed for each room. As it can be seen from the pseudocode, there are nested three for loops for this. Thus, the main loop for the Floyd-Warshall algorithm has **O(R^3)** time complexity.

Then the alternative distances and paths of [x][room number with ammo] is updating where x is a number between 1 and room count. This is done for all the rooms with ammo. After that, while setting alternative distances and paths, only fWarhsall[1][key], fWarshall[key][scientist], fWarshall[scientist][chamber] has new alternative distances and paths.These are the critical cells.

Suppose there are A rooms with ammo. Then there are **AxR updates** before setting the critical cells. For each critical cell, if there is only one room with ammo there will be **1 comparison**. If there are more than one room with ammo there will be **A+1 comparisons**. For example, let there be only one room with ammo which has the number 5. Assume that the key is in room number 2. Then, for updating fWarshall[1][2], it should be checked if there is an alternative path from 1 to 5 and 5 to 2. That is enough. Now, let there be two rooms with ammo which has the numbers 5 and 6. For updating fWarshall[1][2], it should be checked if there is an alternative path from 1 to 5 and 5 to 2. Then it should be checked if there is an alternative path from 1 to 5 and from 5 to 6. Finally it should be checked if there is an alternative path from 1 to 6 and from 6 to 2.

Thus, in the worst case there are additional 3x(A+1) comparisons.

Then the total time is O(R^3 + (AxR) + 3x(A+1)).