

RESCUE BOT SIMULATION

Terms Used:

Deadline: Time left for the people to survive.

Computation time: *Time to go from green room to another room and back (10, 20, 30, 40, 50, 60 time units) OR **time to go from entry/exit point to room and back, depending on situation explained

Note: The following explanation is what we thought of actually in the beginning. However while coding we realized, that this method could be approached in one more way. Both methods are basically the same, but their steps are reversed.

Reverse Black Hole

The main idea used here is **earliest deadline first (our own version)** cum **least computation time**. We have split the problem into 4 parts.

1. Getting people directly out from lower building and getting people in the lower building to the lower green room.
2. Getting people in the upper building to upper green room
3. Getting people in the upper green room to exit.
4. Getting people in lower green room to exit.

The focus is naturally on the green rooms

The time taken from lower green room to exit is 30 units and from upper green room is 80/90 units (depending on whether edge B exists or not). We are going to put people in the green room **only** if upon reaching there they have the **minimum necessary time to get out**.

Each room is split into parts such that they don't have more than two people (e.g. For a room having 5 people there are 3 groups of 2, 2, 1)

Now we are going to explain the approach for Part 2 first.

Earliest Deadline First – This approach is used for the top building (**Part 2**) . The people in the rooms (grouped into 2s or 1s) are sorted in the order of least deadline. If the deadlines are same, they are sorted again by their computation time and then by the no of people in that group.

Based on this sorted table, we calculate who all to save and in which order.

Note: This is done assuming that the robot goes to the higher building first without any activity in the lower. This will be the temporary order of saving people.

There are a few flaws in this approach and we have tried to solve them

1. For the group having only 1 person, we search for similar groups of 1 lying on the way to the green room.
2. We have found that even though people of a certain lower deadline can be saved, by eliminating them, we can save comparatively more people of a higher deadline. This happens when the lower deadline people have a high *computation time, say above 30. To try to solve this flaw, we selectively eliminate people with high *computation time and check in which scenario we get maximum number of people in green room.

Note: So far we have considered going directly to upper building.

Now that we have got the best saving simulation for upper building, we decide whom to save down. Obviously spending time down in the lower building will kill a few from the upper building who were saved previously. The idea is to eliminate the first few entries of the previous simulation and use that time to save a few of the bottom ones.

That's where least computation time (LCT) comes into the picture. We use LCT for the lower building. Least computation Time (for lower building). Now in Part 1, we have to decide in what proportion the time has to be assigned to save people directly and to save people via the green room. We have imaginarily split the bottom building into two parts depending on the proximity of the rooms to the green room and entry/exit point. One part contains those rooms which are closer to the green room and the other contains those rooms which are closer to entry/exit.

For both these parts, we arrange the people according to least computation time first, if same then according to least deadline.

Black Hole Approach

(Read this only after reading the above contents)

All saves/pick ups are done only if they are able to reach exit alive else they are not even considered.

Here we approach Part 1 (again LCF), then we obtain part 2 (again EDF).

Now starting from zero with increments of 10 time units , we give time to the bot to concentrate on bottom building i.e. first case the bot has 0 units to save people down , hence it goes up directly (subtract everyone's time on top by 80). Then in second case it can spend 10 units trying to save people down and then go to top (here subtract 90 from top). Here a temporary table is prepared for every case and is dynamic.

However a problem is encountered here , whether to spend that time for people closer to entry/exit point (computation time 10,20 ..) or people close to the green room (30 units for reaching there , then computation time 10, 20 ...)

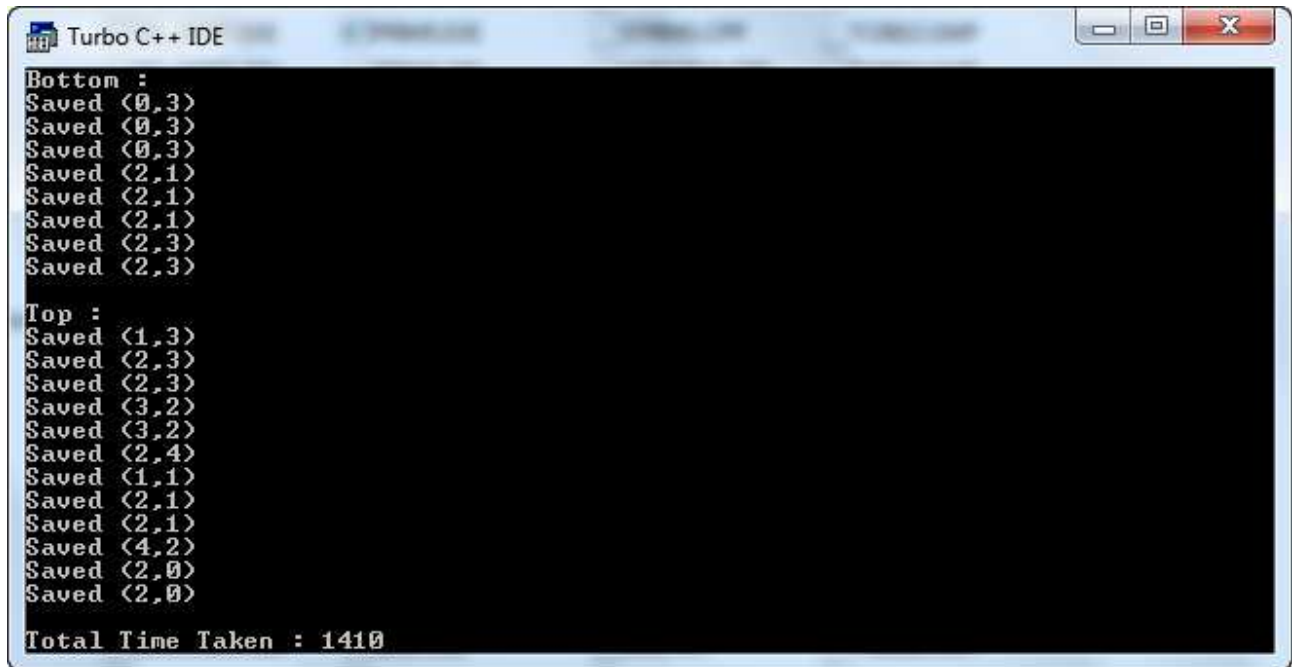
Here an assumption is made. We first consider the people closest to the entry/exit point and pick up those only with computation time of 10 (We ignore the ones with 20, can't help it because code becomes very confusing). Then when the time given to down section appropriately increases to a certain amount we go to the bottom green room as well and here we spend the remaining time to save 10's 20's n so on. (Now we subtract 60 + time given to bottom, from up building's people).

The best case will be the people maximum people saved.

Algorithm

1. Save the people in the room nearest to the entry/exit.
2. Find earliest deadline first for the top building and earliest computation time first for the bottom building.
3. Now for different amounts of time spent in each section find out what the number of people who can be saved in each section (Bottom & Top)
4. The Maximum people will be saved in one of the instances calculated in the previous step.
5. Save these guys in the green rooms, and compute the total elapsed time.

Output



```
Turbo C++ IDE
Bottom :
Saved (0,3)
Saved (0,3)
Saved (0,3)
Saved (2,1)
Saved (2,1)
Saved (2,1)
Saved (2,3)
Saved (2,3)

Top :
Saved (1,3)
Saved (2,3)
Saved (2,3)
Saved (3,2)
Saved (3,2)
Saved (2,4)
Saved (1,1)
Saved (2,1)
Saved (2,1)
Saved (4,2)
Saved (2,0)
Saved (2,0)

Total Time Taken : 1410
```

Understanding the output:

- Coordinates in each of the buildings are numbered starting from the left that's (0, 0) to the rightmost (0, 4) and so on. Thus the top rightmost element is (4, 4). The green room is at (2, 2) in both the conditions.
- Total Time taken is the time taken to save maximum possible people from the building.

Scope for Improvement

- At the moment our code ignores the 1's and doesn't pair them up together if they lie en route to the green room. (We tried this, but couldn't complete this due to paucity of time.)
- Ignoring the rooms near the entry/exit other than the room right next to the entry/exit.
- Didn't take people to the green room when travelling initially from the entrance to the bottom green room and later from bottom green room to top green room.

Requirements

1. Turbo C++ Compiler for Windows/DOS
2. Computer with at least 16Mb of RAM

How to Run

1. Initialize the arrays in the main() function appropriately.
2. Modify the travelling times in the subsequent code in the main() function.
3. Run!

Team

1. Karthik Kastury (Final Year Information Science, NMAMIT, Nitte)
2. Raison D'Souza (Final Year Electronics & Communication, NMAMIT, Nitte)
3. Kenneth D'Souza (Final Year Electronics & Communication, NMAMIT, Nitte)