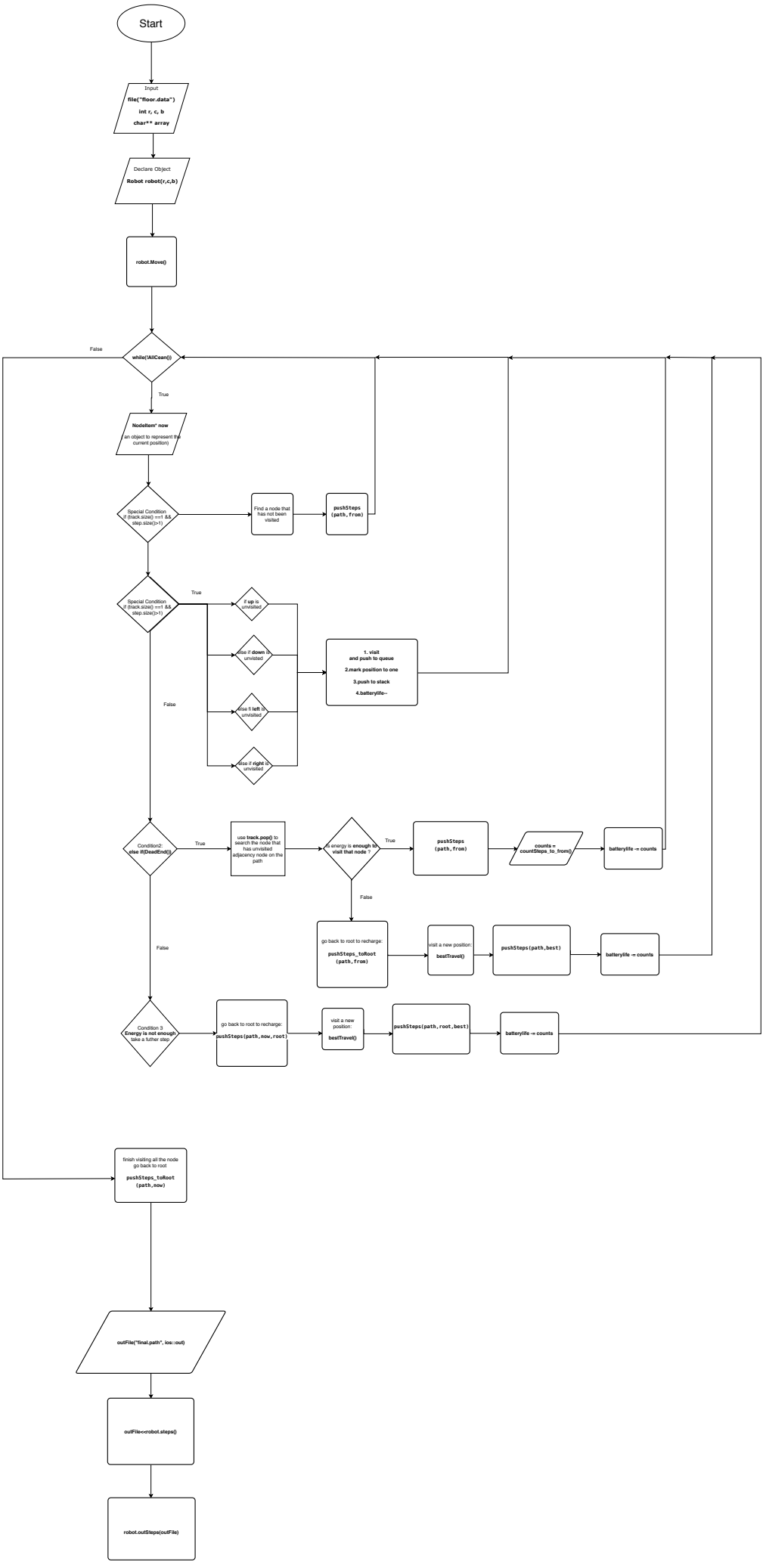
Project 2 Floor Cleaning Robot

學號 1073007S
姓名 葉致廷



Project 2 Floor Cleaning Robot

學號 1073007S
姓名 葉致廷

Class Description:

`struct NodeItem`

- struct NodeItem helps store the data of each “0”, which includes the row index, col index, and how many steps it takes to go from this position to root.

- Also, there’s a pointer called `NodeItem* parent`, pointing to its predecessor. (an important attribute used in the shortest-path calculation)

`class Robot`

`private data:`

- `NodeItem* root`:
  - Represents the information of the root

- `int** map`:
  - Stores the condition of a position. If it’s visited, then the value would be 1; otherwise, it would be 0.

- `int row, col, battery, num_of_node`:
  - Stores the numbers of the row, column, battery, and # of zeros respectively.

- `bool** mapSP` :
  - Stores the original condition of the map.

- `bool** mapcopy`:
  - Used in all kinds of Shortest_Path().

- `int** partial` :
  - Stores the # of zeros on a particular path.

- `stack<NodeItem*> track`:
  - A stack to store the last step.

- `queue<NodeItem*> steps`:
  - A queue to store the whole steps traveled.

- `NodeItem*** path`:
  - A 2D array of NodeItem*, and it represents the path to walk back to root from that position.

`public members:`

`constructor:`
- Take in the input and initialize all of the private data: `map`, `mapSP`, `partial`.

- `map[i][j]` would be 1 for the obstacles; o for the node(num_of_node is counted at the same time).

- `mapSP[i][j]` would be false for the obstacles; true for the node.

- Push 'root' into `track` and `steps`

- Call `ShortestPath_SpanningMap(root)` to get the BFS spanning-tree of the map. (This function will be elaborate later)

`get_num_node():`

- returns private data `"num_of_node"`

`mapCopyInit():`

- Initialize "mapcopy" to the identical state as "`mapSP`" every time it's called. It's used in calculating the shortest path between two nodes without interrupting anything.

`AllClean():`

- A boolean function to check whether all the nodes have been visited.

`ShortestPath_SpanningMap(Nodeitem* root):`

- Returns a 2D array that stores NodeItem* with each cell represents a node in the BFS spanning tree.
- Uses the concept of BFS to traverse the entire 0 in the map which creates a tree rooted in the position of "R" by linking the parent node to its predecessor.

- **Algorithm explanation:**
  - At a certain node, visit each of the four directions adjacency to it if any of them has not been visited `(use the boolean value of mapcopy[i][j] to determine the condition)`. Also, when a node is visited, link the parent pointer of this node to the node it just came from, which is represented by NodeItem* current. After that, push this just visited node to the queue.

  - Repeat the process as in BFS that starts from the root and we shall get a spanning tree and more importantly, a map `(NodeItem*** path)` that stores the path that goes from a certain node to the root, which is relatively time-saving.

`ShortestPath_from_to(NodeItem* from, NodeItem* to):`

- Returns a NodeItem* that represents both "to" which is the end of the shortest path that travels from "from" node to "to" node.

- **Algorithm explanation:**

- Very similar to `Shortest_Path_SpanningMap.`

- At a certain node, visit each of the four directions adjacency to it if any of them has not been visited `(use the boolean value of mapcopy[i][j] to determine the condition)`. Also, when a node is visited, link the parent pointer of this node to the node it just came from, which is represented by NodeItem* current. After that, push this just visited node to the queue. If the now-visiting node is our target position, then just break the loop after the parent pointer is correctly linked to its predecessor.

- Repeat the process as in BFS that starts from “from” and we shall get the shortest path.

`countSteps(int i, int j):`

- Count the # of steps in the shortest path at position [i, j], and luckily we’ve already stored this information in node, so the only thing needed to do is to return `path[i][j]->weight`

`countZeros(int i, int j):`

- Traverse the path, which is also a linked-list, to count the # of zeros in the shortest path at position [i, j], and then returns that number.

`countSteps_to_from(int i, int j):`

- Traverse the path, which is also a linked-list, to count the # of zeros in the shortest path results from `ShortestPath_from_to`, and then returns that number.

`bestTravel(NodeItem* current):`

- First, if the original input contains less than 700000 “0”, then returns the path that has the most # of zeros among all none traverse shortest path.
- Otherwise, just return the node that has not been visited (where map[i][j] still equals to 0)

`DeadEnd():`

- Returns true only if all of the adjacency nodes had been visited

`isValidStep(NodeItem* now, int batterylife):`

- If the robot could travel from its current position to one of its current adjacency nodes without exceeding the energy limit(must be able to walk back to root as well), then return true. All the other cases would return false.

`pushSteps(NodeItem* temp, NodeItem* from):`

- Push a path that starts from "`from`" to "`temp`" into "`steps`".
- Mark the cell on the path as "visited".

- **Algorithm explanation:**

    - Uses a stack to reverse the sequence of the path, which originally starts from "`temp`" to "`from`", and push the top element into private data "steps". Consequently, we get our desired path.

`pushSteps_toRoot(NodeItem* temp, NodeItem* from):`

- Push a path that starts from "`temp`" to "`root`" into "`steps`".
- Mark the cell on the path as "visited".
- Used when the bot needs recharging.

`outStep():`

- Write the steps in the whole process into "`final.path`".

`Move():`

- Determines where to go in the whole process. The workflow is described in the flowchart. Thus, we'll focus on how we assure the correctness of the result and the relation between data.

- **`Algorithm explanation:`**
    - First, if the energy is enough to take the next step, and the bot has not been surrounded by "1"s at the current position, then travel to the adjacency node that's still "0".

    - Second, if the current position is surrounded by "1" then start popping the private data "track" until we find a position that the bot has been to but with unvisited adjacency node. If the energy left is enough to traverse to that point then travel to it using "`pushSteps`". Otherwise, go back to the root and recharge. After recharging, use "bestTravel" to find out the target position the bot would later go to (If the total number of nodes is smaller than 700000, the target node would be the node that has the most # of remaining "0" on the path, else, the target node would the node that's still "0" and closet to the corner.

    - Third, if the bot can't afford to take a further step, then the bot would immediately rush back to the root to recharge. After recharging, use "bestTravel" to find out the target position the bot would later go to (If the total number of nodes is smaller than 700000, the target node would be the node that has the most # of remaining "0" on the path, else, the target node would the node that's still "0" and closet to the corner.

    - After all the node is traveled, travel back to the root.

- However, there's a very special test case that causes problem. Thanks to 107033137, without his/her test case, I probably would've never discovered this problem.

- The problem is that when we encounter the second situation, we expect to find a node that fits that condition. Nevertheless, it is possible that even we pop all the elements in the stack, we still could not find our targeting node. In this case, the track only contains the root while it also has been on the journey, we go back to the root but this time we go to another unvisited node with a similar approach used in "`bestTravel`".

- **The coresponding part in the program**

```
if(track.size()==1 && step.size()>1) {
            int x,y;
            for(int i = 0; i < row; i++) {
                for(int j = 0; j < col; j++) {
                    if(map[i][j] == 0) {
                        x = i;
                        y = j;
                        break;
                    }
                }
            }
            NodeItem* temp = path[x][y];
            pushSteps(temp,root);
        }
```

Test case Description:

Design goals:

1. Check the bot can finish the traversal with restricted energy.
2. Create deadends to see if one's algorithm could cop with them.
3. Create a shortcut to the root for a position at the left lower corner. (If an algorithm does not use the idea of the shortest path algorithm, then it might exceed the energy limit.)

git

history:

- **git log --oneline**

```
(base) macdeMacBook-Air-5:108_1_DS_Project_Floor-Cleaning-Robot mac$ git log --oneline
4aaa3a0 (HEAD -> master, origin/master) still needs to check all test cases again
49adc7e 1000*999 107033137 addressed
ec9b757 1000*1000 is addressed
2335183 BFS major fix
ee30ea0 Addressing large input
1b6d560 Robot_alter
6006a35 Robot_alter
0775f92 Robot_alter
08b997a Robot_alter
5afe490 Robot_alter
efe82fb Robot_alter
b0df424 ver4
cd81dc7 ver4
22a7444 add track.push(s1.top())
cc8dd47 move s4 to recharge part
bc11957 Robot_ver3
699d061 Robot_ver3
0b6b783 README.md
81279b8 README.md
1bf11bc README.md
fe6781e README.md
c9dfa36 Robot.ver2
f029795 advanced version
d207278 Dummy version vol.0
```

- **git log**

```
Branch 'master' set up to track remote branch 'master' from 'origin'.
(base) macdeMacBook-Air-5:108_1_DS_Project_Floor-Cleaning-Robot mac$ git log
commit 4aaa3a0da060ecb5dffae6675b532794d2832fa4 (HEAD -> master, origin/master)
Author: justinyeh <justinyeh1995@gmail.com>
Date:   Fri Nov 22 15:02:28 2019 +0800

    still needs to check all test cases again

commit 49adc7e26febde26b81479f56b9774883f321684
Author: justinyeh <justinyeh1995@gmail.com>
Date:   Thu Nov 21 18:06:45 2019 +0800

    1000*999 107033137 addressed

commit ec9b757f4a6bab90edd817799b9fc9277d568773
Author: justinyeh <justinyeh1995@gmail.com>
Date:   Wed Nov 20 22:19:00 2019 +0800

    1000*1000 is addressed

commit 2335183a419f160aa8fd259fc516b42865b41cb8
Author: justinyeh <justinyeh1995@gmail.com>
Date:   Wed Nov 20 16:49:23 2019 +0800

    BFS major fix
```

repository:
https://github.com/justinyeh1995/108_1_DS_Project_Floor-Cleaning-Robot