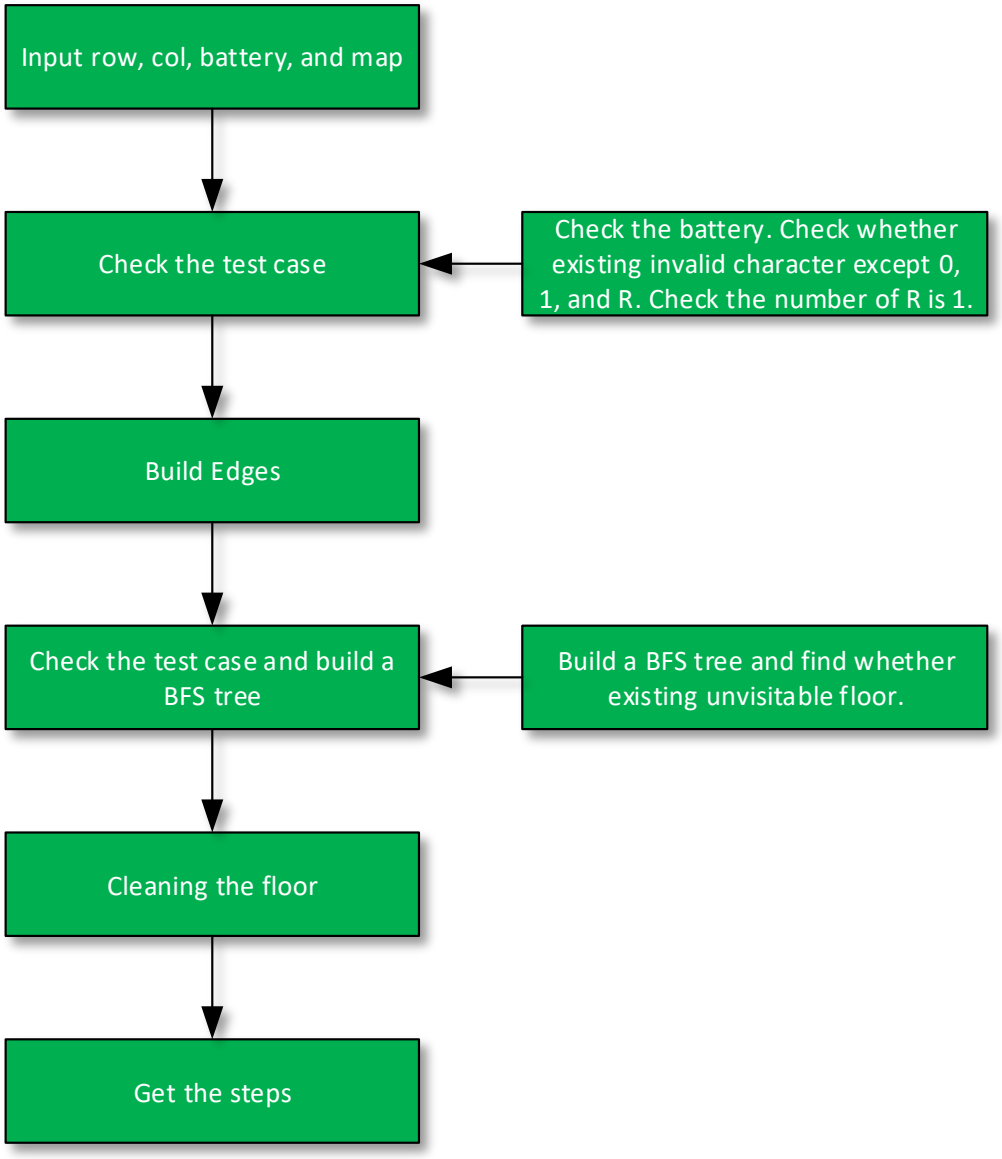
1. Program Description

1.1. Program Flow Chart



1.2. Detailed Description

一開始就是將長、寬、電池輸入後，建立一張 map，這裡的 map 是用 character 去儲存的。

```
map = new char*[row];
for (int i = 0; i < row; ++i) {
    char input;
    map[i] = new char[col];
    for (int j = 0; j < col; ++j) {
        fin >> input;
        map[i][j] = input;
    }
}
```

檢查 Input 是否違法。

```
if (battery > 2147483647) {
    fout << "Valid capacity of battery!";
    exit(-1);
}
for (int i = 0; i < row; ++i) {
    for (int j = 0; j < col; ++j) {
        if (map[i][j] != '1' && map[i][j] != '0' && map[i][j] != 'R') {
            fout << "Exist invalid parameter!";
            exit(-1);
        }
        if (i == 0 || i == row - 1 || j == 0 || j == col - 1) {
            if (map[i][j] == '0') {
                fout << "Exist invalid parameter!";
                exit(-1);
            }
        }
        if (map[i][j] == 'R')
            ++number_of_R;
    }
}
if (number_of_R != 1) {
    fout << "Valid number of R!";
    exit(-1);
}
```

建立一個 graph。

```
class graph {
private:
   node*** lists;          // 利用 map 上的每點用 linked list 儲存 edge
   int*** predecessor;     // 建立 BFS 時每點的 predecessor
   bool** finish;          // 該點是否已被掃除過
   char** map;
   int** distance;         // 每點到 R 的最短距離
   int row;
   int col;
   int battery;
   int clean_num;          // 需要被清掃的數量
   int cleaned_num;        // 已被清掃的數量
   int longest_distance;   // 離 R 最遠的距離
   int steps;              // 總步數
   int start[2];           // R 的座標
public:
}
```

建立 edges。

這裡建立 edge 的順序為下上右左。而建立 edges 的方法是利用 linked list，即矩陣中每個 element 都是 linked list。

```
            if (map[i][j] != '1') {
                int src[2] = {i, j};
                ++clean_num;
                if (i + 1 < row && map[i + 1][j] != '1') {
                    int des[2] = {i + 1, j};
                    addEdge(src, des);
                }
                if (i - 1 >= 0 && map[i - 1][j] != '1') {
                    int des[2] = {i - 1, j};
                    addEdge(src, des);
                }
                if (j + 1 < col && map[i][j + 1] != '1') {
                    int des[2] = {i, j + 1};
                    addEdge(src, des);
                }
                if (j - 1 >= 0 && map[i][j - 1] != '1') {
                    int des[2] = {i, j - 1};
                    addEdge(src, des);
                }
            }
```

然後建立 BFS tree。做完之後能夠取得最遠距離的資訊，判斷一開始輸入的 battery 是否合法，並且能夠找出哪個點為 0 但是卻未被 visited 過。

```
    for (int i = 0; i < row; ++i) {
        for (int j = 0; j < col; ++j) {
            if (map[i][j] != '1' && !visited[i][j]) {
                fout << "Exist unreachable free cells!" << endl;
                exit(-1);
            }
            if (longest_distance < distance[i][j])
                longest_distance = distance[i][j];
        }
    }
```

```
if (longest_distance * 2 > battery) {
    fout << "Lack of power!" << endl;
    exit(-1);
}
```

確認過該 map 沒有限制上或合法性上的問題後，就可以利用 BFS 進行 cleaning 的動作了。

```
當已掃數量<應掃數量時(while loop)
    得到離 R 最遠且還沒被掃過的座標 farthest
    利用 predecessor 找到 R 到 farthest 的最短路徑
    印出 R 到 farthest 的所有點，並將這些點設為「已掃過」
    當目前電力>=到原點的距離時(while loop)
        以未掃過的為優先，以「下上右左」的順序找點
            若沒有未掃過的，就以「下上右左」的順序往下走
            /*
                這一步的目的為在還有電時，盡可能找到其他還沒掃過的目標
            */
            /*
                不過事實上總共有 24 種排列組合(4!=24)，也就是說這只是一種排法，而且事實上排法會大幅影響路徑、總步數，因此若真的要找到比較好的路徑的話，就是將 24 種排法全部走過一次，然後取步數最少的為最佳解，在這份 project 中，以「下上右左」作為順序
            */
        當走到原點時，break
        /*
            因為 R 是唯一一個不是 1 但是可以在外圍的地方，若繼續跑 while 的話，可能會超出 map 的邊界
        */
    順著 predecessor 走回原點
```

上面的演算法就是 cleaning 的核心，當跑完 while loop 時，即走完全步的 map，這時候的 steps 就是我們所需要的總步數。

Code history:
https://github.com/leetoby1215/project_1/commits/master/107061144_project_2.cpp

2. Test case Design

這裡提供幾個正確以及錯誤的測資：

正確的測資：

```
13 21 56
11111111111111111111
10000000000000000001
10R010101010101010101
10000000000000000001
10101010101010101010
10000000000000000001
10101010101010101010
10000000000000000001
10111111111111111111
10101010101010101010
10000000000000000001
10101010101010101010
11111111111111111111
```

錯誤的測資：

5 5 9 11111 10001 10101 10001 1R111	5 6 30 111111 100001 10R00R 100001 111111	5 10 40 1111111111 1000101001 1000111001 1000000001 1R11111111	5 10 40 1011111111 1000000001 1000000001 1000000001 11111111R1	5 10 40 1111111111 1000000001 1000000001 1000000001 11111111r1