1) Project Description

1-1) Program Flow Chart

A. 先將 input 的地圖 fin 進 map[row][column]內。
B. 創建 struct，內含點座標、與 R 點距離、前一個點的座標、有否被尋訪過。
C. 將 map[][]上為 0 的點存進 struct。
D. 將所有 struct push 進 queue。
E. 利用 queue & bfs 算出 struct 內所有點離 R 的距離、前一點的座標。
F. 當 map[][]上有沒被尋訪過的點時，算出 map[][]上離 R 距離最遠的點。
G. 利用兩次遞迴 (dfs1 & dfs2)，從最遠的點開始往前一個點的座標走，直到走回 R，並將走過的點存進 queue 中 (兩次存的方式不同，如此才能印出 R-> 最遠距離->R)。
H. 當 map[][]上的點都被尋訪過，就印出 queue 的 size，再印出 queue 內的所有點。

1-2) Detailed Description

```
92  int main(int argc, char *argv[])
93  {
94      ifstream fin;
95      ofstream fout;
96      fout.open("108062303_proj2.final");
97      fin.open(argv[1]);
98
99      int row, column, life, charge_x, charge_y, dis_max=0, dis_max_x, dis_max_y, output_x[1005]
00
01      char map[1005][1005], non;
02
03      queue<node> q;
04
05      fin >> row >> column >> life;
06
07      for(int a=0;a<row;a++){
08          scanf("\n");
09          for(int b=0;b<column;b++){
10              fin >> map[a][b];
11              if(map[a][b] == '0'){
12                  map1[a][b].x = a;
13                  map1[a][b].y = b;
14                  map1[a][b].visit = 0;
15              }
16              else if(map[a][b] == 'R'){
17                  map1[a][b].x = a;
18                  map1[a][b].y = b;
19                  map1[a][b].dis = 0;
20                  charge_x = a;
21                  charge_y = b;
22              }
23          }
24      }
```

先將 input 的地圖 fin 進 map[row][column]內，並將 map[][]上為 0 的點存進 struct。

```
14  int temp_x, temp_y, flag1;
15
16  struct node{
17      int x, y;
18      int dis;
19      int path_x;
20      int path_y;
21      int visit;
22  };
```

Struct 中包含點座標、離 R 的距離、前一點的座標、有否被尋訪過。

```
126     q.push(map1[charge_x][charge_y]);
127
128     while(!q.empty()){
129         cur = q.front();
130         if(map[cur.x+1][cur.y]=='0' && cur.x+1<row){
131             map1[cur.x+1][cur.y].dis = cur.dis + 1;
132             map1[cur.x+1][cur.y].path_x = cur.x;
133             map1[cur.x+1][cur.y].path_y = cur.y;
134             q.push(map1[cur.x+1][cur.y]);
135             map[cur.x+1][cur.y] = '2';
136         }
137         if(map[cur.x-1][cur.y]=='0' && cur.x-1>=0){
138             map1[cur.x-1][cur.y].dis = cur.dis + 1;
139             map1[cur.x-1][cur.y].path_x = cur.x;
140             map1[cur.x-1][cur.y].path_y = cur.y;
141             q.push(map1[cur.x-1][cur.y]);
142             map[cur.x-1][cur.y] = '2';
143         }
144         if(map[cur.x][cur.y+1]=='0' && cur.y+1<column){
145             map1[cur.x][cur.y+1].dis = cur.dis + 1;
146             map1[cur.x][cur.y+1].path_x = cur.x;
147             map1[cur.x][cur.y+1].path_y = cur.y;
148             q.push(map1[cur.x][cur.y+1]);
149             map[cur.x][cur.y+1] = '2';
150         }
151         if(map[cur.x][cur.y-1]=='0' && cur.y-1>=0){
152             map1[cur.x][cur.y-1].dis = cur.dis + 1;
153             map1[cur.x][cur.y-1].path_x = cur.x;
154             map1[cur.x][cur.y-1].path_y = cur.y;
155             q.push(map1[cur.x][cur.y-1]);
156             map[cur.x][cur.y-1] = '2';
157         }
158         q.pop();
159     }
```

利用 queue 算出 struct 中每一點離 R 的距離、前一點的座標。

```
    while(flag == 1){
        for(int a=0;a<row;a++){
            for(int b=0;b<column;b++){
                if(map[a][b] == '2'){
                    if(map1[a][b].visit == 0){
                        flag = 0;
                    }
                }
            }
        }
        if(flag == 0){
            for(int a=0;a<row;a++){
                for(int b=0;b<column;b++){
                    if(map[a][b] == '2' && map1[a][b].visit == 0 && map1[a][b].dis > dis_max){
                        dis_max = map1[a][b].dis;
                        dis_max_x = a;
                        dis_max_y = b;
                    }
                }
            }
            dfs1(dis_max_x, dis_max_y);
            flag1 = 1;
            dfs2(dis_max_x, dis_max_y);
            dis_max = 0;

        }
        if(flag == 1)
            break;
        else if(flag == 0)
            flag = 1;
    }
```

確認 map[][]上有任何一點為 0 後，找出離 R 距離最遠的點，並利用兩次遞迴(dfs1 & dfs2)，去尋訪該路線，將所經過的點存進 queue。

```
void dfs1(int dis_max_x, int dis_max_y)
{
    if(map1[dis_max_x][dis_max_y].path_x == dis_max_x+1 && map1[dis_max_x][dis_max_y].path_y ==
        dfs1(dis_max_x+1, dis_max_y);
    }
    else if(map1[dis_max_x][dis_max_y].path_x == dis_max_x-1 && map1[dis_max_x][dis_max_y].patl
        dfs1(dis_max_x-1, dis_max_y);
    }
    else if(map1[dis_max_x][dis_max_y].path_x == dis_max_x && map1[dis_max_x][dis_max_y].path_
        dfs1(dis_max_x, dis_max_y+1);
    }
    else if(map1[dis_max_x][dis_max_y].path_x == dis_max_x && map1[dis_max_x][dis_max_y].path_
        dfs1(dis_max_x, dis_max_y-1);
    }
    x.push(dis_max_x);
    y.push(dis_max_y);
    map1[dis_max_x][dis_max_y].visit = 1;
    return;
}
```

第一次 dfs，從距離 R 最遠的點開始往回遞迴，直到走回 R 後，再將走過的點存進 queue 中，因此 queue 中存著從 R->最遠的點。

```
46  void dfs2(int dis_max_x, int dis_max_y)
47  {
48      if(map1[dis_max_x][dis_max_y].path_x == dis_max_x+1 && map1[dis_max_x][dis_max_y].path_y ==
49          if(flag1 == 1){
50              flag1 = 0;
51          }
52          else{
53              x.push(dis_max_x);
54              y.push(dis_max_y);
55          }
56          dfs2(dis_max_x+1, dis_max_y);
57      }
58      else if(map1[dis_max_x][dis_max_y].path_x == dis_max_x-1 && map1[dis_max_x][dis_max_y].patl
59          if(flag1 == 1){
60              flag1 = 0;
61          }
62          else{
63              x.push(dis_max_x);
64              y.push(dis_max_y);
65          }
66          dfs2(dis_max_x-1, dis_max_y);
67      }
68      else if(map1[dis_max_x][dis_max_y].path_x == dis_max_x && map1[dis_max_x][dis_max_y].path_
69          if(flag1 == 1){
70              flag1 = 0;
71          }
72          else{
73              x.push(dis_max_x);
74              y.push(dis_max_y);
75          }
76          dfs2(dis_max_x, dis_max_y+1);
77      }
```

第二次 dfs，一樣從距離 R 最遠的點開始往回遞迴，邊走邊將點存進 queue，直到走回 R，因此 queue 中存著從最遠的點->R。

```
193      fout << x.size() + 1 << '\n';
194      while(!x.empty()){
195          temp_x = x.front();
196          temp_y = y.front();
197          fout << temp_x << ' ' << temp_y << endl;
198          x.pop();
199          y.pop();
200      }
201      fout << charge_x << ' ' << charge_y << endl;
```

當確認 map[][]上的點都被尋訪過後，先印出 queue.size，再印出 queue 中的每個點。由於第二次 dfs 時，R 不會被遞迴到，因此 R 不會存在 queue 中，所以印出每條路線後，再印出一個 R 點，才算一條完整的路線。

2) Test case Design
2-1) Detailed Description of the Test case

4 4 10
1111
1011
1011
1R11

我做個 4*4 的 map，只有一條路可以行走，所以掃地機器人只能原路去回，非常單純。

結果：
5
3 1
2 1
1 1
2 1
3 1