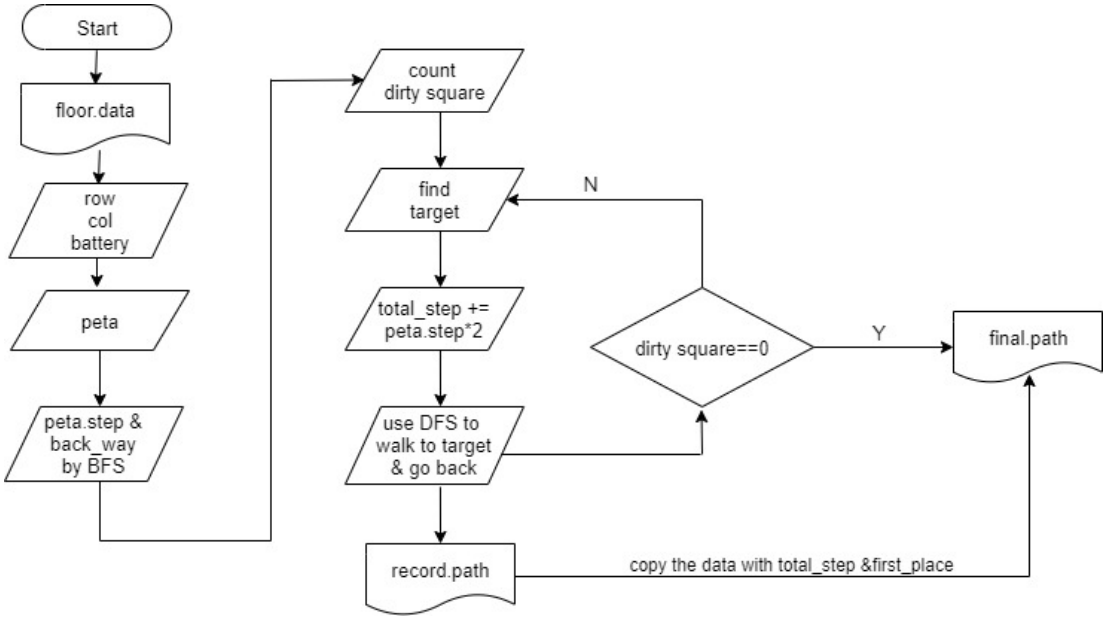
Project #2: Floor Cleaning Robot

姓名：陳泳晗　　學號：107062162

1. Project Description

1-1 Program Flow Chart



1-2 Detailed Description

打開檔案 floor.data，讀取資料，並將地板資料全部存在由 Square 組成的 2D array peta 裡。

```
typedef struct _square{
    int x;
    int y;
    char state;
    bool cleaned;
    int step;
} Square;
```

```
for(int i=0; i<row; i++){
    for(int j=0; j<col; j++){
        input >> (peta[i][j]).state;
        (peta[i][j]).y = i;
        (peta[i][j]).x = j;
        (peta[i][j]).cleaned = false;
        (peta[i][j]).step = 0;

        if((peta[i][j]).state == 'R'){
            R = peta[i][j];
            now_pos = peta[i][j];
        }
    }
}
```

接著開始用 BFS 處理每一個 Square 距離 R（機器人所在地）的最短距離，從起點開始，判斷四個方向可以走而且沒被處理過的點（peta.state==’ 0’ && peta.step==0），它的 step 等於目前所處的 peta.step+1。還有用 array back_way 記錄了上一個 square，這是為了處理之後的 path。

```
const int dir[4][2] = {{0,1}, {1,0}, {0,-1}, {-1,0}};
Square que[row*col];
```

```
int que_front=0, que_tail=0;
que[que_front] = now_pos;
que_front++;
while(que_front > que_tail){
    Square curr = que[que_tail];
    que_tail++;
    for(int detect=0; detect<4; detect++){
        int tmp_x = curr.x + dir[detect][1];
        int tmp_y = curr.y + dir[detect][0];
        if(tmp_x<0 || tmp_x>=col || tmp_y<0 || tmp_y>=row){
            continue;
        }
        Square next = peta[tmp_y][tmp_x];
        if(next.state == '1'|| next.state=='R')
            continue;
        else if(next.state=='0' && next.step==0){
            que[que_front] = next;
            que_front++;
            peta[tmp_y][tmp_x].step = peta[curr.y][curr.x].step + 1;
            back_way[tmp_y][tmp_x] = curr;
        }
```

計算總共有多少個 square 需要清理。

```
for(int i = 0; i < row; i++){
    for(int j = 0; j < col; j++){
        if(peta[i][j].state == '0' && peta[i][j].cleaned==false)
            dirty_square++;
    }
}
```

其實我之前繳交的檔案是沒辦法回到 R 的，這次寫的 code 是我朋友提供的概念，找到最遠的目標，然後用 DFS 走過去再沿路走回來，其實這是很簡單的概念，因為測資規定一定是合法的，所以這時候就完全不需要考慮電量的問題了，也不需要考慮最後要回到 R 點。

只要 dirty_square 不為 0，就繼續去找需要處理的 Square→target。不斷執行 DFS 直到走完所有點。

至於 go_to_a_square，因為是 DFS，最先開始的點最後才會結束，所以需要將起點和終點給倒過來走，這也是剛才之所以要儲存 back_way 的原因，每經過一個 square 必須要把 peta.cleaned=true，這是為了記錄哪些點已經走過。

go_back 也是一樣的概念，從終點往起點走，只是印出來的順序倒過來而已。

這裡用了一個 record.path 來存目前走過的 Square，因為這時候還不知道總共會走多少步，所以不能直接將 Square 坐標直接輸出到 final.path。

```
while(dirty_square!=0){
    int max_step = -10;
    for(int i=0;i<row;i++){
        for(int j=0;j<col;j++){
            if(peta[i][j].step > max_step && peta[i][j].state=='0' && peta[i][j].cleaned==false){
                max_step = peta[i][j].step;
                target = peta[i][j];
            }
        }
    }
    total_step += 2*target.step;

    go_to_a_square(target, now_pos);
    go_back(target, now_pos, target);
}
```

```
void go_to_a_square(Square now, Square target){
    if((now.x==target.x) && (now.y==target.y)){
        return;
    }
    else{
        go_to_a_square(back_way[now.y][now.x], target);
        if(now.state == '0' && peta[now.y][now.x].cleaned==false){
            peta[now.y][now.x].cleaned = true;
            dirty_square--;
        }
        tmp << now.y << ' ' << now.x << endl;
    }
}
```

```
void go_back(Square now, Square A, Square B){
    if((now.x==A.x) && (now.y==A.y)){
        tmp << now.y << ' ' << now.x << endl;
        return;
    }
    else{
        if((now.x!=B.x) || (now.y!=B.y)){
            tmp << now.y << ' ' << now.x << endl;
        }
        go_back(back_way[now.y][now.x], A, B);
    }
}
```

最後輸出 final.path 的時候，只要加上 total_step 和起點位置，然後再把 record.path 的資料給複製過去就好了。

```
output << total_step << endl;
output << R.y << " " << R.x << endl;
ifstream tmp_in;
tmp_in.open("record.path");

int x, y;
for(int i=0;i<total_step;i++){
    tmp_in >> y >> x;
    output << y << " " << x << endl;
}
tmp_in.close();
output.close();
return 0;
```

2. Test case Design

2-1 Detailed Description of the Test case

我設計的 test case 唯一比較有挑戰的就是我自己寫的時候卡最久的分岔路，不過只要演算法是正確的，這是完全沒問題的啦。

```
7 7 55
1111111
1100101
100R101
1101101
1000101
1000001
1111111
```

Github 網址和 commit

https://github.com/YHanTan/DS_project2