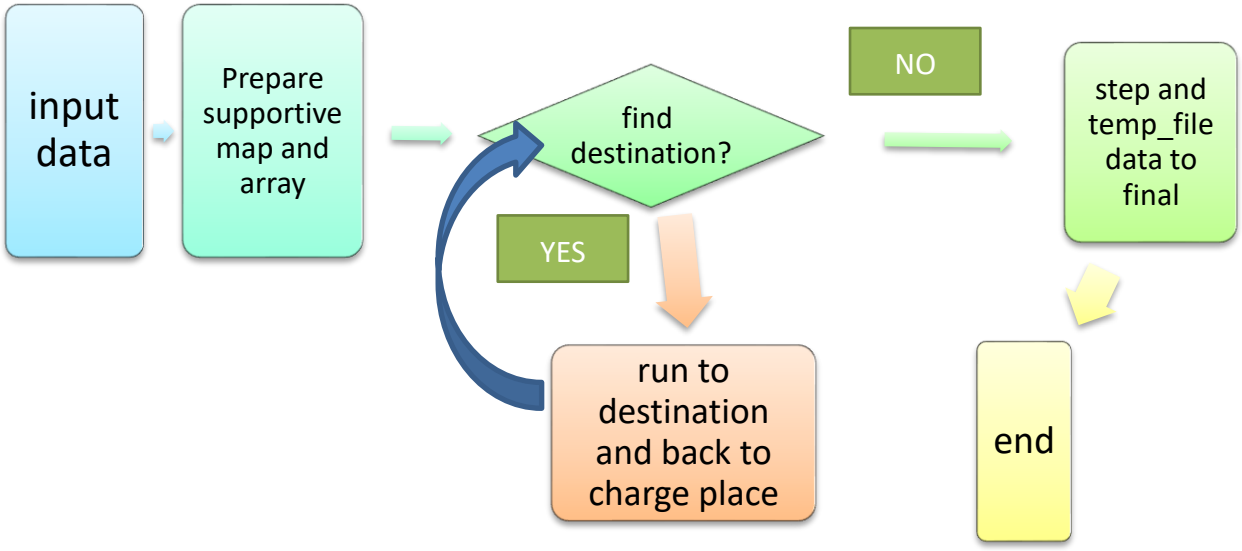
Data Structure Project 2 – Floor Cleaning Robot

107062318 李俊逸

1) Project Description

■ Program Flow Chart



概述：

首先輸入地圖、電池等資料。

再來將輔助的工具，如：地貌(0, 1, R)、充電處與每塊地板距離多遠(用BFS找)、有沒有走訪過的地圖建好。在BFS建立距離地圖的同時，還建立一個array，紀錄每塊走訪過的地板的座標以及距離，由於BFS會慢慢增大距離，array元素的距離值會隨著index越來越大。接下來就開始掃地。

掃地時，利用上述的array，檢查有無未走訪過的目的地。若有，機器人會走過去，到目的地後再走回充電站，再看array裡有沒有沒走過的地板，不斷地來來回回直到走完所有地板。

走完所有地板後，由於output一開始要先有走的步數，我先在final一開始寫入步數，再依序寫入暫存檔裡存的路徑。寫完之後關檔案，結束程式。

■ 1-2) Detailed Description

詳述：

1. FILE

有三個file，分別是負責輸入的floor.data、負責暫存路徑的temp.path以及加上步數的最後結果final.path。簡言之，先將結果存在temp.path，等到跑完得到步數後，再加上步數，並複製temp.path的資料過去。

2. Class Position

```
class position{ //用來建造queue的
    public:
        int c;
        int r;
        int d;
        position(): r(-1), c(-1){};
        position(int r, int c, int d): r(r), c(c), d(d){};
        position& operator = (const position &p){
            if(this != &p){
                c = p.c;
                r = p.r;
                d = p.d;
            }
            return *this;
        }
};
```

Position用來記錄floor的位置以及與充電站的距離。(過載寫了沒用到，因為快寫完才想到可以寫等號的過載，所以code裡面等號的部分都寫了三行。)

3. Print function

由於最大測資可到1000*1000，我把output的endl換成’\n’ 、用ios::sync_with_stdio(false)以及cin.tie(0)鬆綁I/O，把這種case的執行時間降到10秒以下。

4. Map

用infile讀char的方式，將R設-1、地板設0、牆壁為1。

5. Visitedmap

用地圖將非地板的地方設為1，代表參訪過；地板設為0，代表還沒參訪。

6. Charge_distance_map

首先將充電站的距離設0、牆壁設無限遠(1000000)，再用BFS從充電站開始走訪所有地板，把資訊用position class的array記錄下來。它被我用來計算回來的路徑。

7. BFS

我用BFS做出Charge_distance_map與(position*)destination_array，分別幫我找回來跟目的地的路徑。

實做方法是，從充電站開始，一次就是左右上下各跑1格，直到跑完所有路徑為止。

我記錄充電站位置與距離(0)在destination_array[0]裡，設兩個變數first=0, last=1，last會是新找到的點該放的位置，找到就加1；first則是在找完一個點能發展的路徑後會+1，找destination[1]的左右上下點，如果沒走過這個點，就再推進destination[last]裡，直到”first == last”時，代表能走的都已經走完了，就結束BFS。

```
void run()
{
    // run to dc, dr, and go back to cc, cr.
    // revise visitedmap using map.
    // nr, nc(global var) is unused.
    route_to_destination();
    //printmatrix(visitedmap);
    route_back_to_charge();
    //printmatrix(visitedmap);
}
```

8. find_destination

```
while(find_destination()){ /
    run(); // go to the dest
    now_life = life;
}
```

從最遠的地板開始掃(即destination_array[last])，對照visitedmap，如果沒掃過這個地方，就回傳true，並用全域變數dr, dc, dd記錄搜尋到的目的地的位置與距充電站的距離。接著跳回main，進入run()，實際走過去再走回來。

9. Route_to_destination(route_arr)

根據Charge_distance_map，從目的地走最短路徑回到充電站。走完後，route_arr裡依序會是從目的地走回充電站的最短路徑，所以要從最後一個元素讀回來，才會是從充電站到目的地的最短路徑，邊讀邊infile到temp.path之中。

10. Route_back_to_charge

回充電站比到目的地複雜一些。一開始從目的地出發，每跨出一步之前，我會先看看左右上下一格處是否有未走訪過的地板，如果那塊地板是最短路

徑地板的話，就直接走過去。如果最短路徑的地板都走訪了，我會看看非最短路徑地板是否走訪過，以及走訪那塊地板後電池還夠不夠回充電站。如果走訪過，就走最短路徑；如果未走訪過且電池夠回充電站，就走那塊地板，不夠就走最短路徑的地板。

走訪過程中不斷地執行上述、infile路徑到temp.path，直到回到充電站結束函數。接下來跳回main function，回到條件判斷(find_destination)再檢查是否有沒走訪過的地板。

11. Output result

```
outfile << cr << " " << cc << '\n';
while(find_destination()){ //if it's true, dr and dc should be in the right position.
    run(); // go to the destination and back to the charge station. battery should be cha
    now_life = life;
}
outfile.close();
temp_infile.open("temp.path");
output_result();
//cout << "Finished!" << endl << endl;
//printmatrix(map);
//print_robot_position();
delete_memory_allocation();
temp_infile.close();
remove("temp.path");
return 0;
```

跑完while迴圈後，代表地板都很乾淨了，步數也算清楚了。接下來我把temp.path關起來，再以ifstream的方式開啟temp.path。先將步數output到final.path後，再依序把temp.path的資料抓進去。最後delete開過的memory，把temp.path移除，結束程式。

2) Test case Design

- 2-1) Detailed Description of the Test case

```
10 20 150
11111111111111111111
10001000100000010001
10101010101010010101
10101010101000010101
10101010101101000101
10101010101001010101
10101010101001000001
10101010101001010101
10100010001000000101
1R11111111111111111111
```

為了使同學演算法的步數盡量差異化，我採用助教testcase的方式寫了一個必經一段很長路徑的case，最後到的地方再隨意地設一些路障，讓大家走路走起來可能會有很多重複的路徑，畢竟演算法不同，重複路徑的多寡也不同，由於要回去充電要走一大段路，少走幾格的代價會相當大。因此，也就達成”使同學間的演算法步數差異最大化”的目的。

3) Git log 紀錄

```
* 9ff249d (HEAD -> master, origin/master) first version finished, with classmate's testcase
* a717bda first version finished, with spec file
* dc5383e first version finished, with endl->\n
* dfdb4ac first version revise some mistake, add timer, 1000*1000 map
* 669ee62 first version done
* 735c629 first version with route_back_to_charge without input output file
* e5841df first version with route_to_destination
* 5caa802 first version with find_destination
* 58b7834 first version with unsorted desination_array_add some description
* e6c03e5 first version with unsorted desination_array
* e0e8147 first version with BFS
* 9fb818c first version with basic structure
* d995b1f first version with print
* 845d03c init
```

Github 網址：https://github.com/karta1502545/DS_project2