1. Program Description
   1. Program Flow Chart



* 1. 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>

1. 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

111111111111111111111

100000000000000000001

10R010101010101010101

100000000000000000001

101010101010101010101

100000000000000000001

101010101010101010101

100000000000000000001

101111111111111111111

101010101010101010101

100000000000000000001

101010101010101010101

111111111111111111111

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