演算法HW3

姓名：張峻瑋

學號：110511194

Q1：

解題思路：

本題的限制是只能往下或往右，也就是欲到達某一點只能透過其上面那點或左邊那點過來。故只需找到到達上面那點及左邊那點的最短矩離，並比較這兩點誰到目的地後距離最短，即為所求。

舉例來說：

|  |  |  |
| --- | --- | --- |
| 1 | 2 | 3 |
| 3 | 2 | 1 |
| 2 | 2 | 2 |

圖1：Example 1的題目

由圖1即可畫出DP table：

|  |  |  |
| --- | --- | --- |
| 1 | 3 | 6 |
| 4 | 5 | 6 |
| 6 | 7 | 8 |

圖2：Example 1的DP table

如第(1,1)格為2，到(0,1)的距離為3，到(1,0)的距離為4，則到(1,1)的最短距離為min(3+2, 4+2) = 5，依此推得整個DP table。

以下是建立DP table部分的程式碼：

vector<vector<int>> dp(r, vector<int>(c));

    int k = (r - 1) + (c - 1);

    for(int i = 0; i <= k; i++){

        for(int ri = i, ci = i - ri; ri + ci <= i; ri--, ci++){

                if(ri < 0 || ci >= c)

                    break;

                if(ri >= r){

                    ri = r - 1;

                    ci = i - ri;

                }

                if(ri == 0)

                    dp[ri][ci] = dp[ri][ci-1] + maze[ri][ci];

                else if(ci == 0)

                    dp[ri][ci] = dp[ri-1][ci] + maze[ri][ci];

                else

                    dp[ri][ci] = min(dp[ri-1][ci], dp[ri][ci-1]) + maze[ri][ci];

        }

    }

由於欲填該點必先知道其上面那點及左邊那點，故for迴圈的方式為：

0 1 2

3

4

|  |  |  |
| --- | --- | --- |
| (0,0) | (0,1) | (0,2) |
| (1,0) | (1,1) | (1,2) |
| (2,0) | (2,1) | (2,2) |

紅色數字表示箭頭畫過的格子其行與列之和。依序由數字小到大的箭頭指向填表。

Q2：

解題思路：

我們可將源字串與目標字串都分割成子字串，一一計算各個子字串到子字串之間的萊文斯坦距離。依序可分成4種可能：

1. 不變  
   若源字串與目標字串的最末字相同，則其萊文斯坦距離同其左上角那格。
2. 取代  
   若源字串與目標字串的最末字不同，則其萊文斯坦距離為左上角那格加1。
3. 刪除  
   將源字串最末字刪除後的字串，與目標字串比距離。其萊文斯坦距離為左邊那格加1。
4. 插入  
   將源字串於最末字後新增一字，與目標字串比距離。其萊文斯坦距離為上面那格加1。

其DP table如下：

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | H | O | R | S | E |
|  | 0 | 1 | 2 | 3 | 4 | 5 |
| R | 1 | 1 | 2 | 2 | 3 | 4 |
| O | 2 | 2 | 1 | 2 | 3 | 4 |
| S | 3 | 3 | 2 | 2 | 2 | 3 |

圖3：Q2 Example 1的DP table

|  |  |
| --- | --- |
| 取代 | 插入 |
| 刪除 | 目標點 |

圖4：計算DP table時各點之意義

以下是建立DP table部分的程式碼：

for(int i = 0; i <= target\_length; i++) {

        for(int j = 0; j <= source\_length; j++) {

            if(i == 0) {

                dp[i][j] = j;

            } else if(j == 0) {

                dp[i][j] = i;

            } else if(target[i - 1] == source[j - 1]) {

                dp[i][j] = dp[i - 1][j - 1];

            }

            else{

                dp[i][j] = min(dp[i - 1][j - 1], min(dp[i][j - 1], dp[i-1][j])) + 1;

            }

        }

    }

由於所求只需要距離，並沒有要求是哪一種字串編輯方式，故只需分成2種情況：一種是源字串與目標字串同字時，距離不變；第二種是源字串與目標串不同字時，其左上、上、左三格中挑選最小的加1即為所求。