HW3 Report

* Q1  
  由於迷宮只能往下走或往右走，因此定出從到終點所需的最低花費為：  
  為終點時，。再來只要從終點開始一步一步回推即可：

|  |
| --- |
| for (int i = 0; i < N; i++) {          for (int j = 0; j < M; j++) {              cin >> a[i][j];          }      }      for (int i = N - 2; i >= 0; i--) { a[i][M - 1] += a[i + 1][M - 1]; }      for (int j = M - 2; j >= 0; j--) { a[N - 1][j] += a[N - 1][j + 1]; }      for (int i = N - 2; i >= 0; i--) {          for (int j = M - 2; j >= 0; j--) {              a[i][j] += min(a[i + 1][j], a[i][j + 1]);          }      }      cout << a[0][0] << endl; |

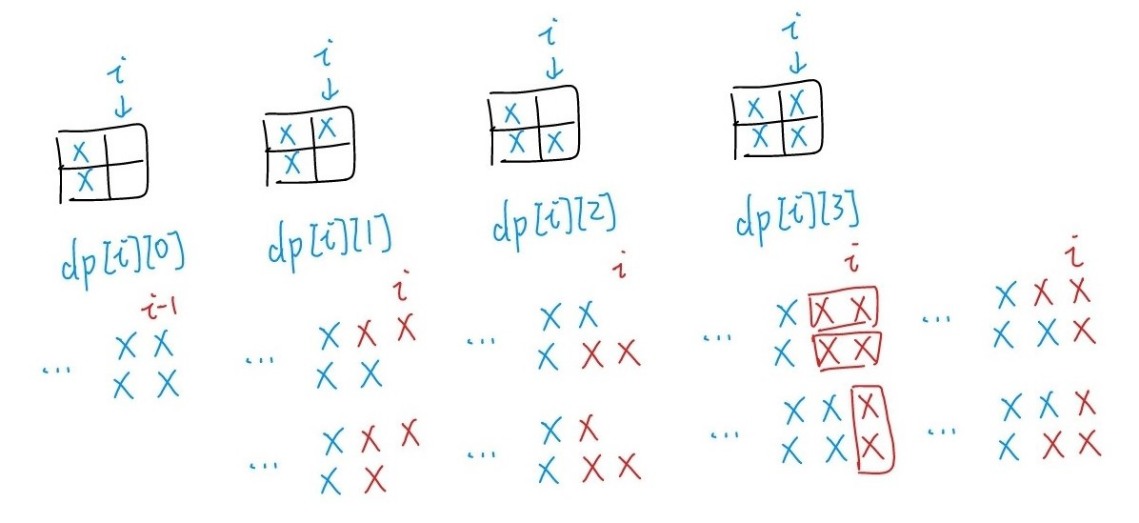
* Q2   
  其實我一開始看不太懂題目，想說去Leetcode刷個幾題找靈感結果就找到原題了，不過我還是有想到解法。這題主要是在定義對字串的三個操作，並轉換成遞廻的方程式，就可以解出來。  
  將字串使用字元的方式來index，可以把本來要算s1轉換到s2的字串轉成s1[1…i+1]轉換成s2[1…j+1]的問題（寫程式的時候我轉回從0開始）。然後定義：

其中當取到

* + 代表將s1移除一個字元
  + 代表將s2移除一個字元（可以看作是在s1插入原本s2上的字元，因此維持前一個狀態）
  + 代表將s1上的字元用s2替換掉

+1的部分就是執行操作的開銷，然後設定當i或j等於0的時候是空字串，因此base case就是，即為從空字串插入字元所需的開銷。

|  |
| --- |
| for (int i = 0; i <= s1\_len; i++) { dp[i][0] = i; }      for (int j = 0; j <= s2\_len; j++) { dp[0][j] = j; }      for (int i = 1; i <= s1\_len; i++) {          for (int j = 1; j <= s2\_len; j++) {              dp[i][j] = s1[i-1] == s2[j-1] ? dp[i-1][j-1]:                                          1 + min(dp[i-1][j], dp[i][j-1], dp[i-1][j-1]);          }      }      cout << dp[s1\_len][s2\_len] << endl; |

* 一張含有 文字, 筆跡, 字型, 書法 的圖片

  自動產生的描述Q3  
  第三題我是用一個一個col的方式來看的，然後用二進制的方式來對第i個col的不同狀態做編碼（假設前i-1個column都有排滿磚頭），如下圖所示：  
  然後底下是排出各種使用第i-1, i-2的狀態來拼出第i個狀態的方法（我一開始卡了很久，最後才發現是沒有考慮到兩個磚頭橫放的排法…）。  
  接下來是設定初始條件，由於會需要前兩個狀態的結果，因此需要先把i=0, 1時的狀態先找出來。這邊為了保險起見，所以我是用手算把這兩組狀態算出來。  
  程式如下（MOD的部分我也是弄了很久才發現…）：

|  |
| --- |
| const int MOD = 1000000007;      dp[0][3] = 1;      dp[1][0] = 1;      dp[1][3] = 1;      for (int i = 2; i <= N; i++) {          dp[i][0] = dp[i - 1][3];          dp[i][1] = (dp[i - 1][0] + dp[i - 1][2]) % MOD;          dp[i][2] = (dp[i - 1][0] + dp[i - 1][1]) % MOD;          dp[i][3] = (dp[i][0] + dp[i - 1][2] + dp[i - 1][1] + dp[i - 2][3]) % MOD;      }      cout << dp[N][3] << endl; |

可以發現0, 3; 1, 2這兩組狀態其實是重疊的，因此可以進行壓縮，最終的程式如下：

|  |
| --- |
| dp[0][0] = 1;      dp[1][0] = 1;      for (int i = 2; i <= N; i++) {          dp[i][0] = (dp[i - 1][0] + dp[i - 1][1] \* 2 + dp[i - 2][0]) % MOD;          dp[i][1] = (dp[i - 2][0] + dp[i - 1][1]) % MOD;      }      cout << dp[N][0] << endl; |