演算法程式作業二

110502529 陳柏燊

Q1 演算法實作:

利用組合學，當前位置組合數 = 來自左邊的組合數加上來自下面的組合數（遇到障礙則不加）

Time complexity θ(K＊H＊W)

程式解釋:

```
int ans[1005][1005], block[1005][1005];
```

ans[i][j] 代表從起點走到(i,j)的組合數

block[i][j] 代表(i,j)的左或下是否被阻擋

0:都沒被阻擋, 1:左被阻擋, 2:下被阻擋, 3:左和下皆被阻擋

```
for (int i = 0; i <= W; i++)
    for (int t = 0; t <= H; t++) block[i][t] = 0;
```

block 初始化（ans 由於會被覆蓋因此不需要初始化）

```
for (int i = 0, a, b, c, d; i < numBlock; i++) {
    cin >> a >> b >> c >> d;
    if (a == c) block[c][d] += 2;
    else block[c][d] += 1;
}
```

Block 輸入(0:都沒被阻擋, 1:左被阻擋, 2:下被阻擋, 3:左和下皆被阻擋)

```
for (int i = 1; i <= W; i++) {
    // 0 all // 1 down // 2 left // 3 none
    for (int t = 1; t <= H; t++) {
        ans[i][t] = 0;
        if (!(block[i][t] & 1)) ans[i][t] += add(i - 1, t);
        if (block[i][t] < 2) ans[i][t] += add(i, t - 1);
        ans[i][t] %= 2552;
    }
}
```

主計算程式，從(1,1)開始計算，透過讀取 block 確認是否被阻擋，來計算是否加上來自左或下的組合數

DP 演算法實作:

dp[i] 表示不經過其他障礙物，直接從起點到第 i 個障礙物起點的組合數

遞迴式: DP[i] = table[i.fx][i.fy] -{dp[n]*table[i.fx - n.ex][i.fy - n.ey]} (n 滿足 n.ex <= i.fx and n.ey <= i.fy)
//fx :表示障礙物起點的 x　　ex : 表示障礙物終點的 x
//fy :表示障礙物起點的 y　　ey : 表示障礙物終點的 y
//遞迴式中的大括號內的 n 是代表所有終點的 x 和 y 皆小於等於 i 起點的 x 和 y 的障礙
//table[i][j] 代表從起點走到(i, j)且中間不存在任何障礙的組合數(可事先建表)

將一個 dummy 的障礙放在終點，那麼 dp[dummy]就會是答案

Time complexity θ(K * Q * Q + H * W)

程式解釋:

```
for (int i = 0; i < 1002; i++) {
    table[0][i] = 1;
    table[i][0] = 1;
}
for (int i = 1; i < 1002; i++) {
    for (int t = 1; t < 1002; t++) {
        table[i][t] += table[i - 1][t];
        table[i][t] += table[i][t - 1];
        table[i][t] %= 2552;
    }
}
```

初始化 table，並確保 mod 2552

```
for (int i = 0; i < numBlock; i++) {
    cin >> block[i][0] >> block[i][1] >> block[i][2] >> block[i][3];
}
```

輸入障礙物到一個一維 vector，element 為 array<int, 4>

```
block.push_back({ W, H, W, H });
sort(block.begin(), block.end(), cmp);
```

加入 dummy 並 sort vector 以確保 buttom up 的執行順序

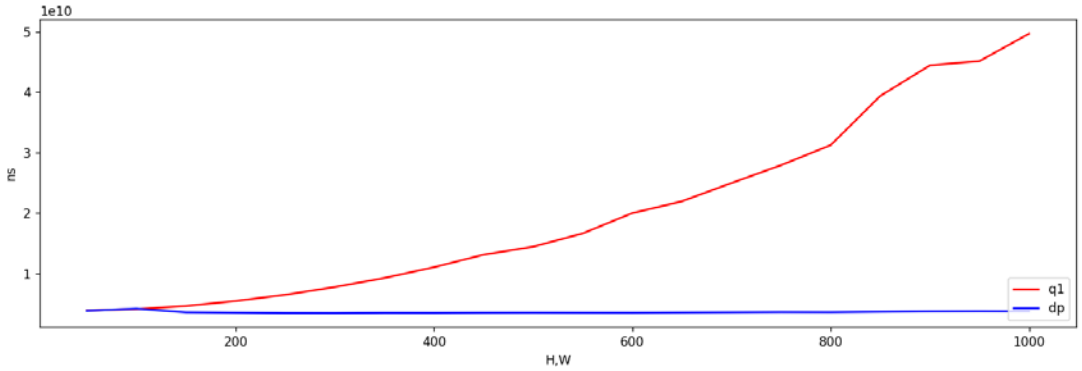
```cpp
for (int i = 0; i <= numBlock; i++) {
    dp[i] = table[block[i][0]][block[i][1]];
    for (int t = 0; t < i; t++) {
        if (block[t][2] > block[i][0] or block[t][3] > block[i][1]) continue;
        dp[i] -= dp[t] * table[block[i][0] - block[t][2]][block[i][1] - block[t][3]];
        dp[i] = ((dp[i] % 2552) + 2552) % 2552;
    }
}
```

主要計算程式

先把 dp[i] 設為 table[i.x][i.y]，再扣掉合法的障礙物(遞迴式中定義的)，最後一行是確保 dp[i] 不會變成負數

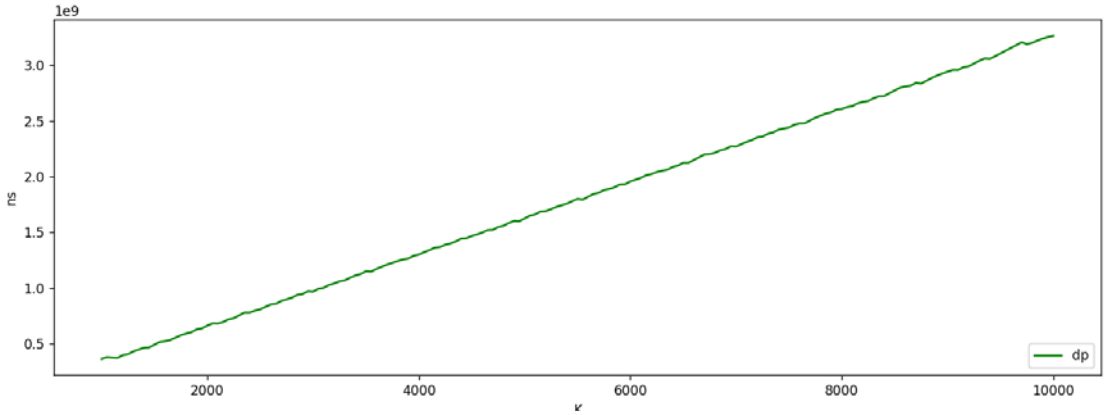
時間測試和測資設計:

我一開始是測試 K 為 10000, Q 為 100, H,W 為 50 ~ 1000 的時間比較



我們可以發現 H,W 對 dp 算法基本上沒什麼影響，且 H,W = 1000 時 q1 作法甚至需要快 50 秒。但，這時我發現，dp 作法在 k=10000, Q=100 時，耗時會超過 3 秒(有可能是我電腦比較廢)所以我接下來打算測怎樣得 k 可以確保 dp 不會超過 3 秒

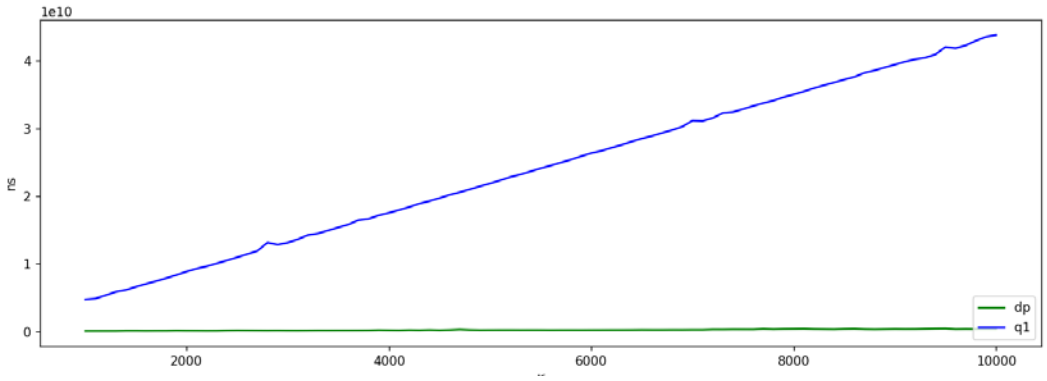
下圖為 H,W = 1000, Q = 100, k = 1000 ~ 10000 的時間比較



我們可以發現他是一個非常漂亮的斜線(為了確保穩定性我開了冷氣加超強的電風扇吹他…)大概是 k = 9000 時超過 3 秒時，我想 k = 8000 內時該都可以保證

3 秒內

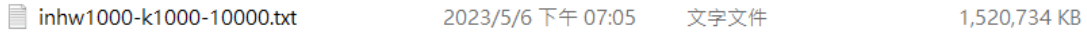
最後是如何設計出 q1 一定超時，dp 一定過的測資，由時間複雜度我們可以發現 H,W 影響 q1，q 影響 dp，於是我設計 H,W 皆為 1000，q 為 10(為了確保題目算法的正確性)，k 從 1000 ~ 10000 的測資



我們可以發現 dp 從 k = 1000 時大約是 4 秒， k = 10000 時大約是 40 秒，然後 dp 是大約 0.2 秒到 0.3 秒(對，他有增加，只是非常不明顯)，完全不會超過 3 秒，所以最後的測資就決定是 H,W = 1000, K=10000, Q = 10 來卡掉非 dp 的算法。

一些小故事(沒啥重點的東西):

第二章圖的測資高達快 1.5G 哇，生成都花了快 10 分鐘



其實大部分的數據都跑了好幾次，之前沒開冷氣加電風扇的時候，那個圖都給我上下亂跳，超級醜，於是要做報告畫圖的時候，我又全部重跑一次，才有現在這樣漂亮的圖