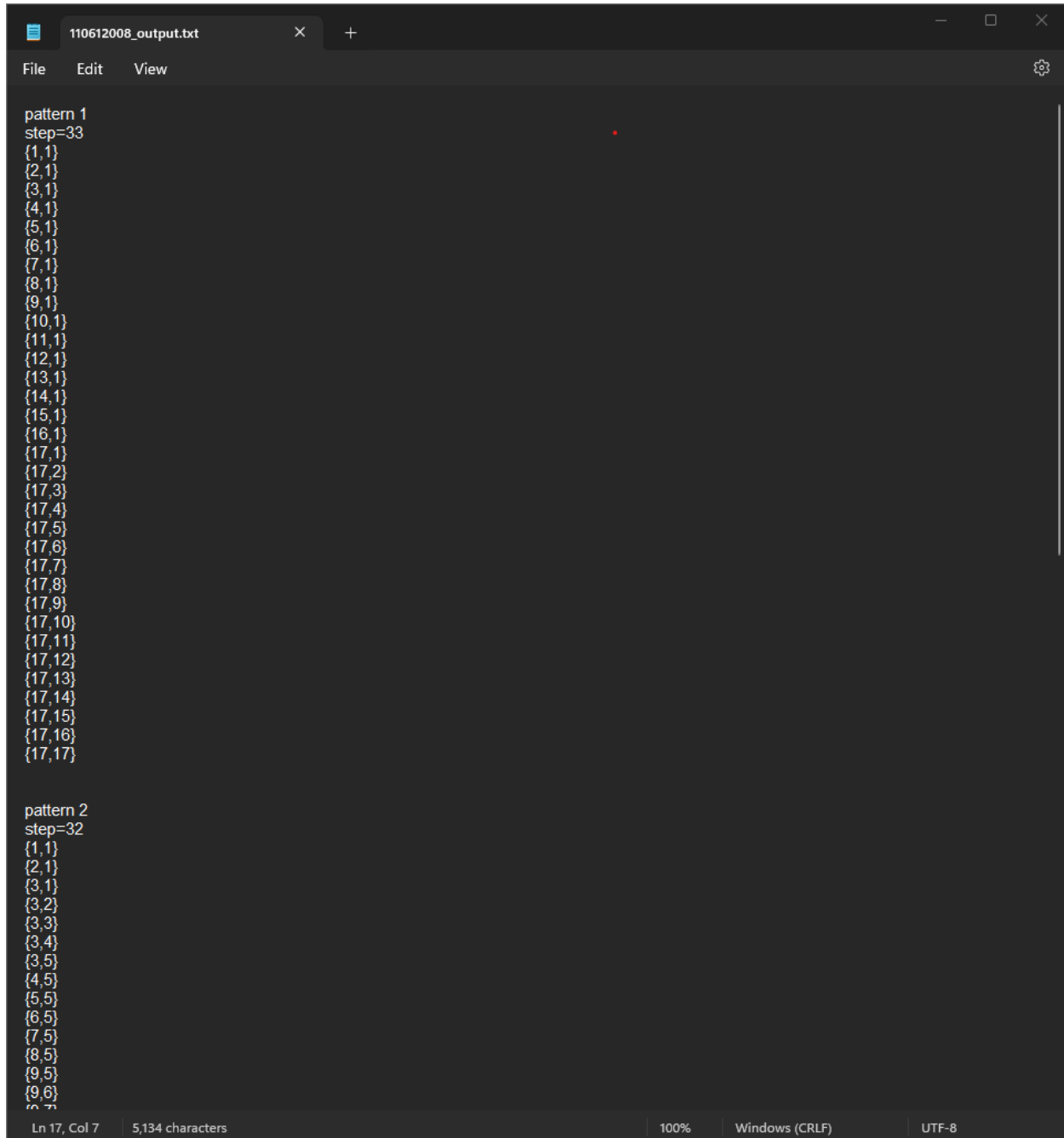


Algorithm_HW8

110612008 沈昱翔

- Output results:



```
110612008_output.txt
File Edit View
pattern 1
step=33
{1,1}
{2,1}
{3,1}
{4,1}
{5,1}
{6,1}
{7,1}
{8,1}
{9,1}
{10,1}
{11,1}
{12,1}
{13,1}
{14,1}
{15,1}
{16,1}
{17,1}
{17,2}
{17,3}
{17,4}
{17,5}
{17,6}
{17,7}
{17,8}
{17,9}
{17,10}
{17,11}
{17,12}
{17,13}
{17,14}
{17,15}
{17,16}
{17,17}

pattern 2
step=32
{1,1}
{2,1}
{3,1}
{3,2}
{3,3}
{3,4}
{3,5}
{4,5}
{5,5}
{6,5}
{7,5}
{8,5}
{9,5}
{9,6}
{9,7}
```

Ln 17, Col 7 | 5,134 characters | 100% | Windows (CRLF) | UTF-8

- Variable design:
 1. 設計一個 `struct` 去描述一個座標點的三項特性(顏色、距離、`parents`)。
 2. 設計兩個控制座標移動的數列 `dx[]`、`dy[]`，兩個數列合起來由左至右是右上左上。
 3. 將迷宮(`maze`, `graph`)設計為一個固定大小的矩陣。
 4. 將點 `s(start)`、點 `v`、點 `u` 設計為 `0~17*17-1` 的數列，缺點是在 `x, y` 值有變化的時候需要轉換，要花時間用清楚轉換成座標的模式，優點是好比較，而且較省記憶體，因此我覺得這個資料結構比較適合存取，如此一來 `vertex` 的存取結構也不會那麼複雜。
 5. `Q` 用 `queue` 來存取，`enqueue`, `dequeue` 的動作跟這個存取機制非常適配。
- Code design:(省略老師上課有講到的設計和偽代碼的內容)
 1. `void print_path(int s, int v):`
 在講義的偽代碼中，副程式輸入端裡面有三個變數，加上副程式裡面有用 `v.pi`，總共有四個變數，在設計的過程中發現 `G` 的引用是多餘的，同時，我們如果需要用到 `v.pi`，就需要將另一個程式 `void BFS(const int G[n][n], int s)` 的 `vertices` 拿出來用，也就是說要 `return`，會把程序變得很複雜，也會花費存取空間，所以我把座標的特性 `vertices` 提出來變成全域變數，並讓 `print_path` 在 `BFS` 定下答案所有的 `vertices` 後執行。
 2. `void BFS(const int G[n][n], int s):`
 內容跟偽代碼中沒什麼不同，最後等到 `BFS` 的動作完成後輸出步數(發生在終點，也就是 `17*17-1` 的 `vertices distance`，再用 `print_path` 遞迴回去找他的 `parents`，他 `parents` 的 `parents`，直到回溯到起點。
 3. `int main():`
 引用 `ifstream`(因為 `print_path` 會用到 `ofstream`，所以在全域就宣告完成)開啟檔案，若開啟失敗會有警示，並終止運算。利用迴圈將資料一輪一輪的送進去(一次一個迷宮)，因為作業說明有提到助教會測資 20 個迷宮，所以第 89 行便執行 20 次迴圈，一次迴圈中，會先存取迷宮到 `maze[n][n]` 裡面，再做 `BFS` 的運算和 `print_BFS`。
- 心得:
 這次作業因為是最後一次，特別想了很多怎麼樣存取可以讓演算法的運算時間或是空間最佳化，畢竟演算法在之後是一個工具，在更大型的程式會一次引用多種演算法，如果每個資料存取都很肥大，會浪費記憶體和花心力優化演算法的研究生、教授心血。另外，這次也學著不無腦用 `vector`(`vector` 的存取真的很方便、很直觀)，用數列去存取迷宮。