ReadMe

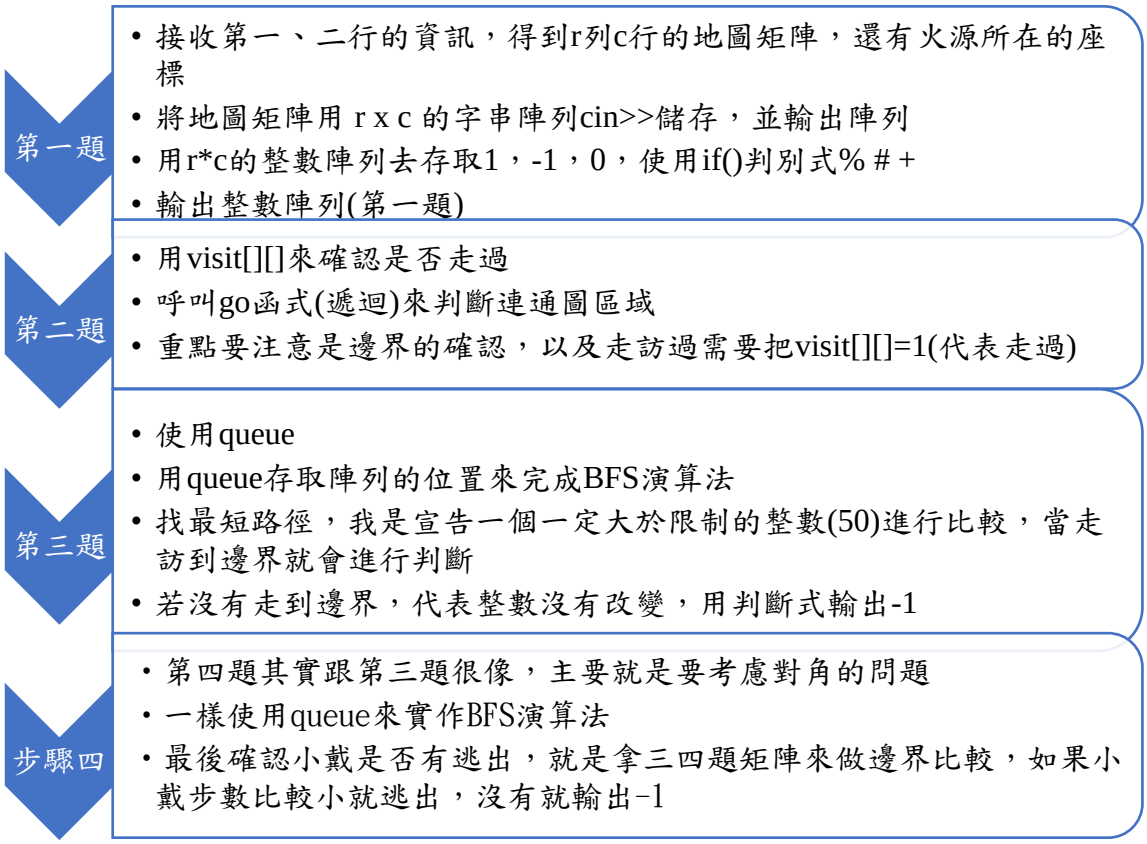
I. 開發紀錄

1. 開始執行[2020/5/27]
2. 成功使用亂數配置金剛、小戴、小戴可行走的路(使用 for 迴圈、亂數、if)[2020/5/27]
3. 搞錯題目意思，直接 cin 輸入即可
4. 解決第二題，使用 DFS 來遞迴造訪，遇到小 bug(輸出不對)[2020/5/28]
5. 解決第二題，不能用 if else 如果要同時上下左右判定的話，需要使用四個 if[2020/5/28]
6. 進入第三題，二維陣列跟佇列思考如何實現 BFS[2020/5/28]
7. 跟同學討論，結論出三種可能可行方法
   a. 使用節點跟線來作圖，參考助教範例實作
   b. 使用 struct 存取陣列以及距離和陣列的位置訊息，再用 queue 存取位置訊息
   c. (自己的想法，實踐成功)直接把矩陣的位置變成一個 int，i*row+j 來存入 queue，再取出來轉回去 row 及 column 的位置[2020/5/28]
8. 遇到小問題，路徑不對[2020/5/28]
9. 原來是剛開始存取的小戴位置是-1，導致累加出了問題，已解決[2020/5/28]
10. 進入第四題[2020/5/29]
11. 其實第三題第四題相似只是要把對角考慮進去，第四題完成[2020/5/30]
12. 因為我四題分開模擬做，做成功再合併，發現參數宣告相同，進行合併調整[2020/5/31]
13. 發現上述的座標位置會導致錯誤，重新調整，使用 i*col+j 解決問題[2020/5/31]
14. 發現第四題嚴重的 BUG 緊急解決[2020/5/31]
15. 發現嚴 BUG 小戴在邊界，第三題，第四題會有錯[2020/5/31]

II. 編譯執行說明

1. 我的程式使用 c++編譯，編譯，執行檔跳出終端機，將作業提供的測資 copy 貼上，按 enter 即可執行，有使用到 遞迴 <queue> 用於 dfs,bfs 演算法的實作

III. 程式執行流程



A. 第二題

1. 資料結溝說明
   a. 用大小 r x c 的字元陣列 map，用來存取測資，再用相同大小的整數陣列 werit，用來把$%+轉換成 0、-1、1，程式所用的空間複雜度大概是 O(n)
2. 演算法說明
   a. 做雙層 for 迴圈，用 row，col，做大小邊界
   b. 用 if else if 判斷式
      1. 如果是+，整數陣列 werit 就在此位置存 1
      2. 如果是%，整數陣列 werit 就在此位置存-1
      3. 如果是$，整數陣列 werit 就在此位置存 0
   c. 用雙層 for 迴圈輸出 werit[i][j]
   d. 時間複雜度 O(r*c)如果 rc 相同就是 O(n^2)

B. 第三題

1. 資料結溝說明
   a. 用大小 30 x 30(一定大於限制) 的全域變數陣列 visit，表示每個座標位置已經拜訪的痕跡，1 代表已拜訪，0 則是這個座標還沒有走過，程式所用的空間複雜度大概是 O(n_2)
   b. 用大小 30 x 30(一定大於限制) 的全域變數陣列 third，初始化每個值為 0，用來存取小戴的位置(-1)，跟所走的步數，程式所用的空間複雜度大概是 O(n_2)
2. 演算法說明
   a. 宣告一個佇列 queue 名稱是 quack
   b. 用雙層 for 迴圈，迴圈內 if 判斷式尋找小戴數值(-1)，找到-1，宣告一個整數 w 存取小戴位置 i*col+j，將 w push 進 queue 裡面
   c. 使用 while 迴圈，條件是確認是否 queue 為空
   d. 把 queue 的第一個值存入整數 current 中，轉換成二維陣列的座標，開始進行上下左右判斷
      1. (判斷向右會不會超出邊界)&&(可不可以行走)&&(有無走訪)
         a. 已走訪 visit[][]=1
         b. 這個位置是上個位置+1
         c. 將位置轉換丟回 queue 中
      2. 向左向上向下都是同樣做法
      3. 尋找最短路徑，宣告整數 distance 為 50(一定大於限制)
         a. 如果走訪到邊界，就跟 third[i][j]做比較來獲得最短路徑
   e. 發現嚴重 BUG，小戴在邊界，最短路徑應為 0(已解決)
   f. 時間複雜度::O(n^2)

C. 第四題

1. 資料結溝說明
   a. 用大小 30 x 30(一定大於限制) 的全域變數陣列 visit_fire，表示每個座標位置已經拜訪的痕跡，1 代表已拜訪，0 則是這個座標還沒有走過，程式所用的空間複雜度大概是 O(n_2)
   b. 用大小 30 x 30(一定大於限制) 的全域變數陣列 fire，初始化每個值為 0，用來存取火源的位置(-1)，跟所走的步數，程式所用的空間複雜度大概是 O(n_2)

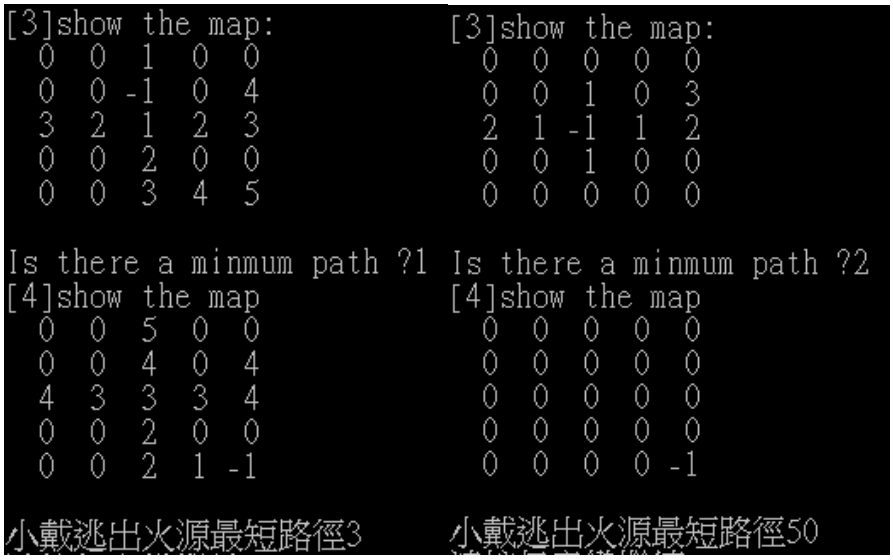
2. 演算法說明
   a. 其實第四題跟第三題做法很像
   b. 宣告一個佇列 queue 名稱是 fire_queue
   c. 剛開始 cin>>的火源位置當作初始位置，把它 push 到 fire_queue
   d. 使用 while 迴圈，條件是確認是否 queue 為空

e. 把 queue 的第一個值存入整數 current 中，轉換成二維陣列的座標，開始進行上下左右對角判斷

1. (判斷向右會不會超出邊界)&&(可不可以行走)&&(有無走訪)
   a. 已走訪 visit[][]=1
   b. 這個位置是上個位置+1
   c. 將位置轉換丟回 queue 中
2. 向左上下對角都是同樣做法，主要需要注意的地方是判斷邊界是否超過

f. 確認小戴是否逃出去

1. 做四個 for 迴圈，使用{}包覆讓 i 可以做完迴圈就釋放記憶體，讓 i 可以重複宣告
2. 四個迴圈就是用來做邊界比較，如果 third[i][j]<fire[i][j]，代表小戴成功逃出，用整數 datastr 去存取小戴逃出的最短路徑
3. 發現嚴重的 BUG
   a. 因為我是看測資來做，測資一般都會讓火蔓延，如果火剛開始就被困住，那 **datastr 無法有存取**
   b. 還有一個問題，如果火燒燒到邊界一定會比小戴慢，那就必須比較哪一個是最短路徑，而不是按順序找剛好有就存取。
      i. 以下錯誤附圖



圖片(左)有多個成功逃出路徑但沒有做比較的錯誤

圖片(右)原本火就沒有蔓延，導致 f.2 的方法無法進行，逃出路徑為我初使設值

c. 還有一個很有趣的問題就是，有兩種狀況一種狀況是火沒有燒出去，一種是小戴被火追上了，這個也要考慮進去，原本我只是把火沒有燒出去加進去判別，但這樣太草率了，會把原本考慮逃不出去的條件弄掉

d. 跟朋友討論一下，朋友有提出假如用第二題的連通圖來判斷，假設火源的連通圖位置跟小戴不同，代表火不會蔓延
→但火源會有對角燒，所以無法用連通圖判斷

g. 時間複雜度:O(n^2)