- 開發紀錄

  1.實作第一題、第二題 DFS，第二題 chased 測資不過，其他 ok[2020-05-25]。

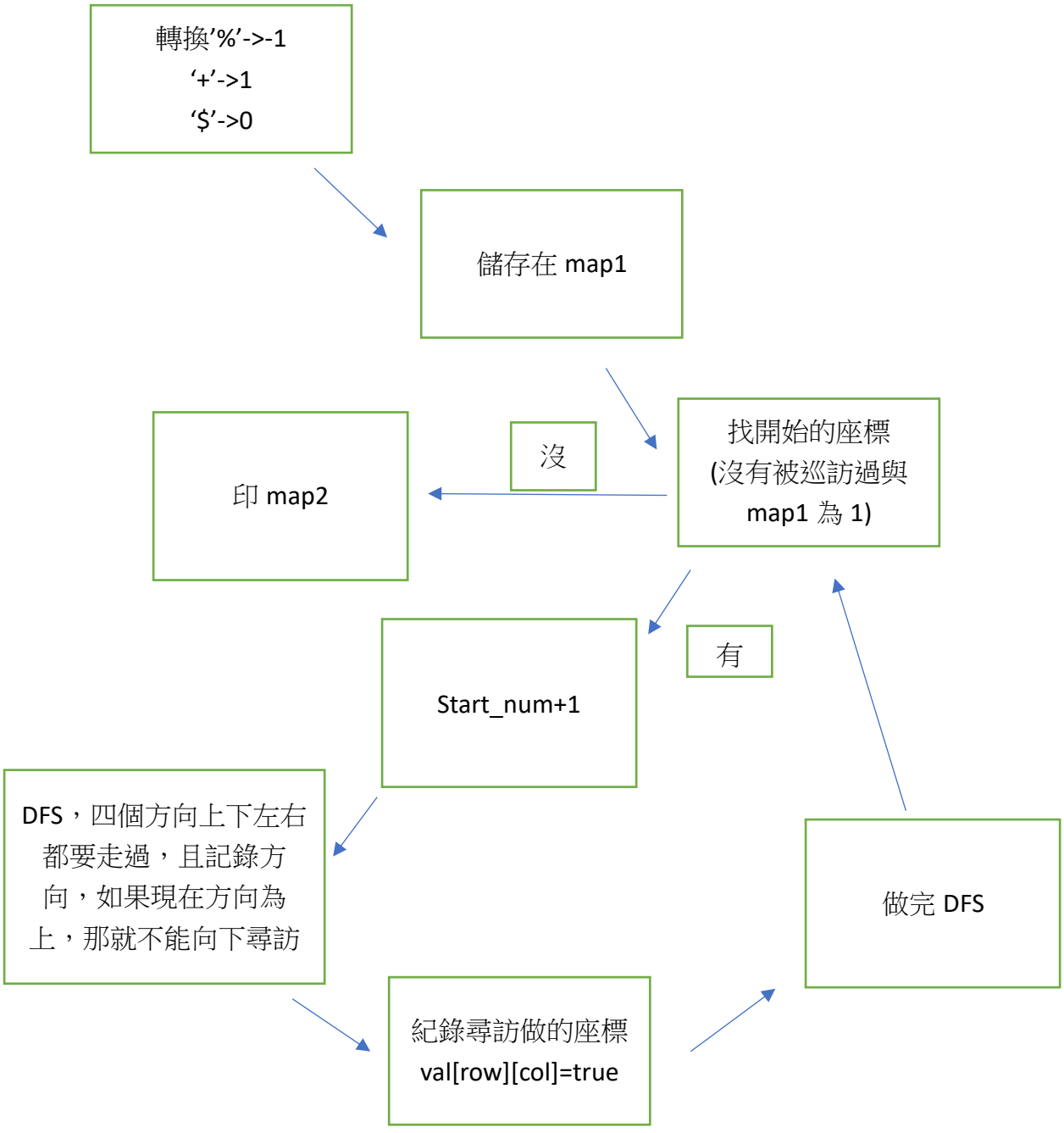
  2.Debug 第二題與實作第三題(註:bug 成功修復第三題測資也都沒問題)。[2020-05-30]

  3.實作第四題(註:全部檢測一遍，3 個測資都過) [2020-05-31]。
- 編譯執行說明

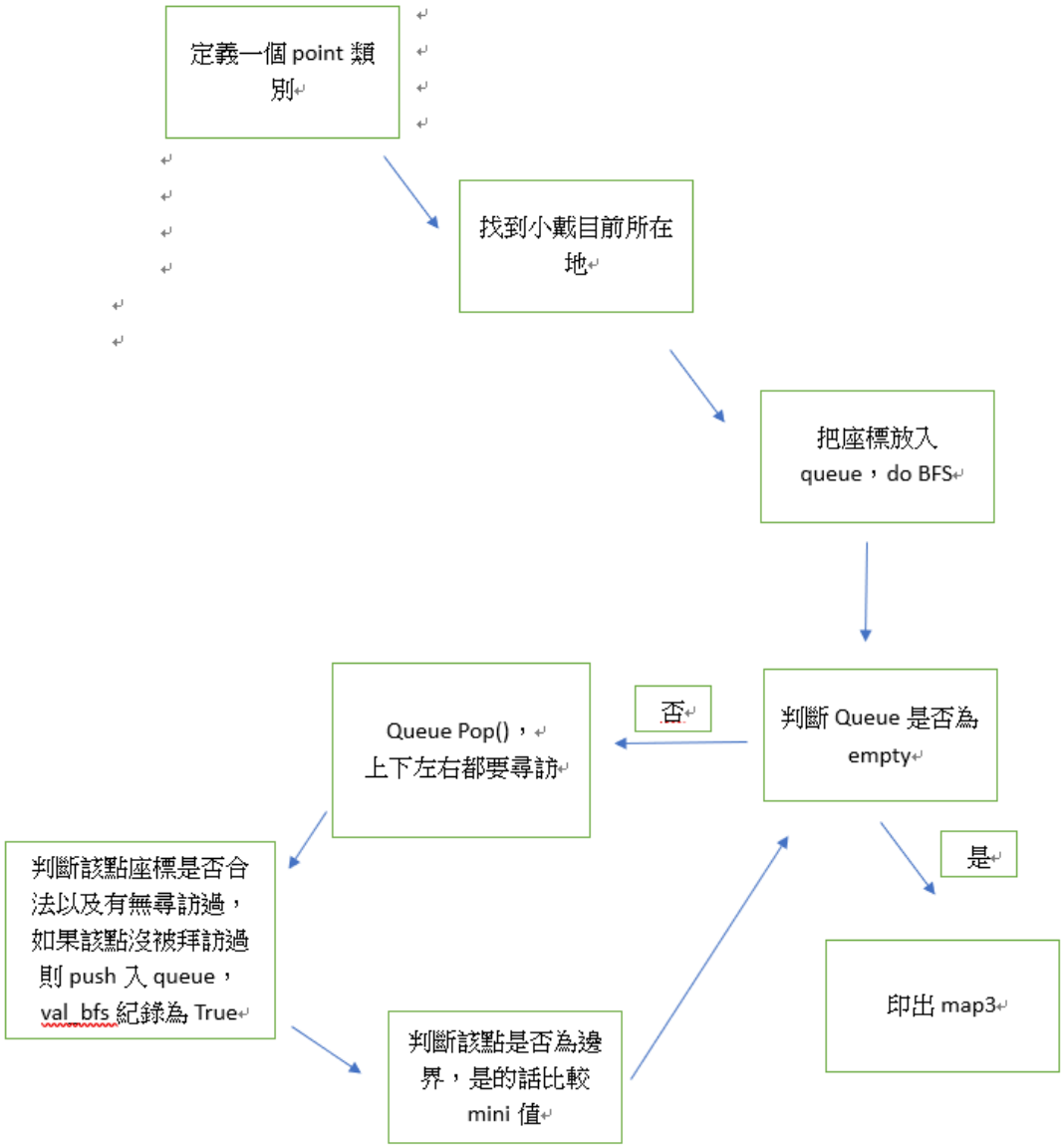
  1.我的程式使用 C++編譯。

  2.第三、四題在做 BFS 用到 Queue。
- 程式執行流程

  第一&第二題



第三題



第四題

1.第 4 題就是在第三題的 BFS 上多判斷斜方向而已，至於求最小步數則是在 map4(火燒到的時間)完成後，重新再做一次第三題的 BFS，並且要判斷每一步是否會被火燒到。

- 第 2 題 的 資料結構和演算法
  - 資料結構說明

    1.用 map2[20][20]整數陣列來記錄第二次要印的圖。

    2. val[20][20]布林陣列來記錄是否拜訪過，有設 True，無設 False。

    3.考慮到 stack，dfs 空間複雜度大概跟圖的長寬有關->O(r*c)(最多)

- 演算法說明

  1.尋找可以走的點，找到 start_num+1。

  2.從可以走的點開始呼叫 dfs，每拜訪一個點，val 設為 true。

  3.沿著通一個方向找，不行就換另一個方向，且如果現在方向為上，那就不能向下尋訪，以此類推。

  4.當在 stack 裡的點上下左右都走過之後，再繼續尋找可以走的點。

  5.最大複雜度大概為 O(4^n)(上下左右)，n 最大為 r*c。

- 第 3 題 的 資料結構和演算法
  - 資料結構說明

    1.用 map3[20][20]整數陣列來記錄第 3 次要印的圖。

    2. val_bfs[20][20] 布林陣列來記錄是否拜訪過，有設 True，無設 False。

    3.queue 用來存拜訪過的點。

    4.考慮 queue，複雜度為 O(r*c)(最多)
  - 演算法說明

    1.找到小戴位置 push 入 queue 中。

    2.做 BFS，有拜訪過的點 val_bfs 設為 true，沒有設為 false。

    3.queue pop，把鄰近可走的點都 push 入 queue 中，，步數為舊點步數+1。

    4.複雜度跟 DFS 差不多。
- 第 4 題 的 資料結構和演算法
  - 資料結構說明

    1. 1.用 map4[20][20]整數陣列來記錄第 4 次要印的圖。

    2. val_bfs[20][20] 布林陣列來記錄是否拜訪過，有設 True，無設 False。

    3.queue 用來存拜訪過的點。

    4.考慮 queue，複雜度為 O(r*c)(最多)
  - 演算法說明

    0.初始化 map3、val_bfs 陣列。

    1.找到火位置 push 入 queue 中。

    2.做 BFS，有拜訪過的點 val_bfs 設為 true，沒有設為 false。

    3.queue pop，把鄰近可走的點都 push 入 queue 中，除了上下左右還要檢查對角，時間為舊點時間+1。

    4.複雜度跟 DFS 差不多。

    5.找最小逃生的步數，則是把第三題的 BFS 重做一遍，並且要判斷該點是否會被火燒到(與 map4 比較，若該點小於 map4 的值，則可以走)。