Theory of Computer Game Hw1 資工所碩一 R05922068 彭宇劭

source code: nonogram.py / nonogram.c

執行方式：

python3 nonogram.py [input_file_path] [size_of_nonogram]
gcc -o nonogram nonogram.c

./nonogram [input_file_path] [size_of_nonogram]

Implement Algorithm: BFS, DFS, DFS with cutoff

說明：

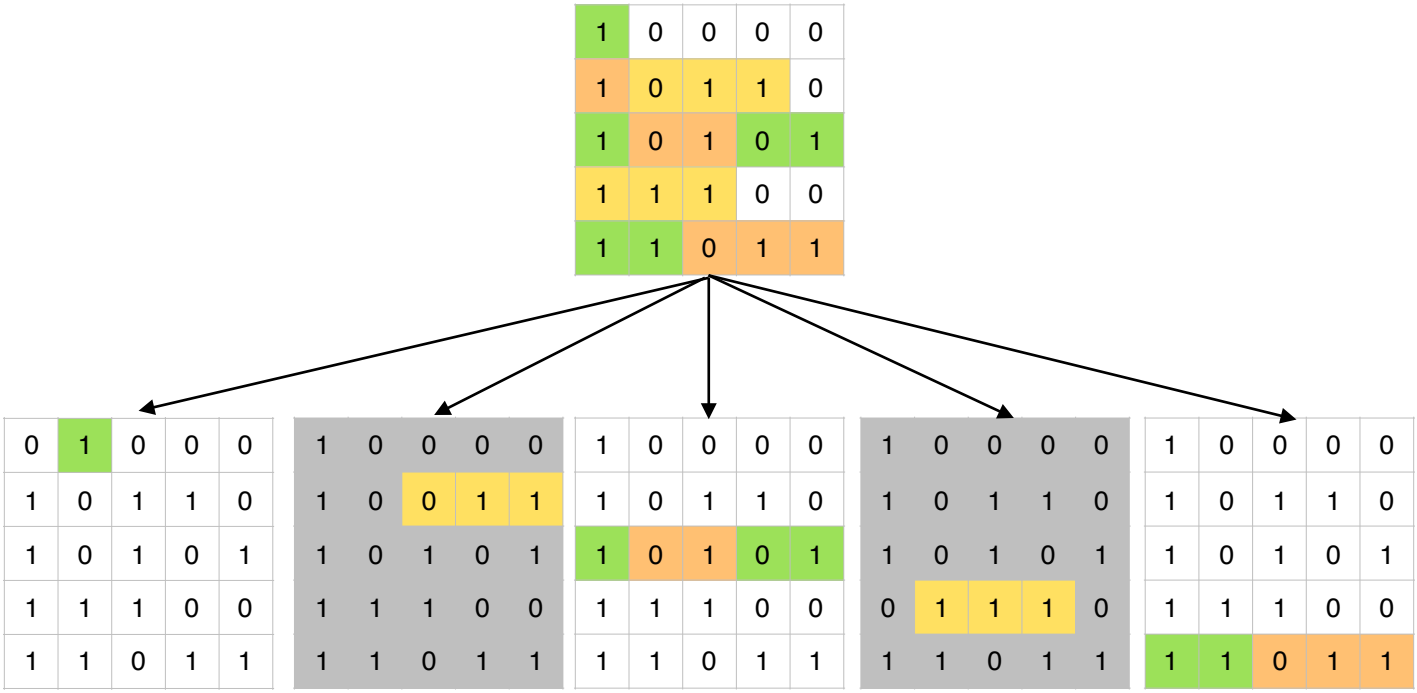
由於 Nonogram 不同於15-puzzle沒有起始盤面，也沒有目標盤面，如一開始就知道目標盤面那就不用做直接輸出答案，所以在解這個問題前必須先將盤面做一些定義。

- 起始盤面：為了降低複雜度，在起始盤面中利用Row或Column其中一邊的規則將點分堆，如下圖所示。

1	1	0	0	0	0
1 2	1	0	1	1	0
1 1 1	1	0	1	0	1
3	1	1	1	0	0
2 2	1	1	0	1	1

有了起始盤面，接下來就可以由此盤面開始移動，利用BFS或DFS方法搜尋。

- **BFS**
  在BFS搜尋中由淺到深一層層的搜索，這邊定義每一層就是動一個方塊一步，其中因為兩組不同的小組中間必須有一個0隔開，所以直接把一個零包進去當成同一個小組，如下圖所示。



每一層就只將一個小組動一步，其中有些Row是無法移動的，如上圖中間及右邊那兩個盤面，移動盤面後將目前盤面跟Column的規則做檢查，如果都符合則找到結果，如有不符合則將盤面放進Queue中，再Dequeue出一個盤面將他移動一步，做檢查不符合就Enqueue，直到找到結果！
此方法可以找到最佳解（這邊指的是移動最少步的解）不過對於這個遊戲沒有意義，因為這邊的步數是自己定義出來的。

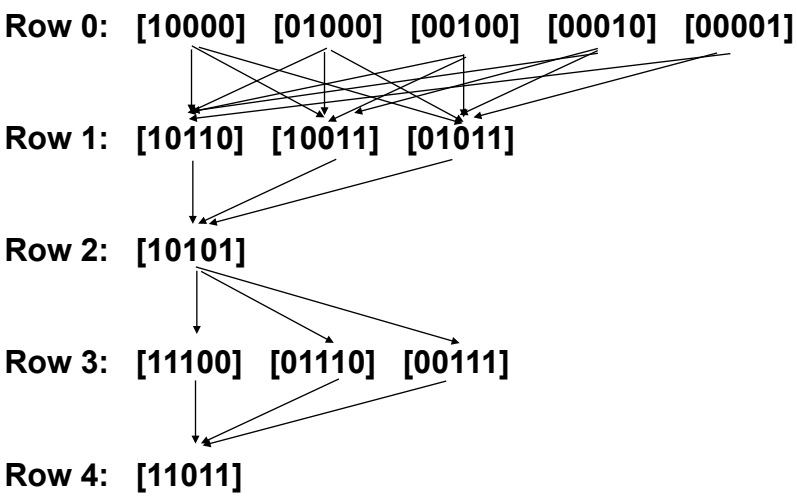
- **時間複雜度分析**
  若每個Row有一個1： $O\left((n-1)^n\right)$
  若每個Row有兩個分開的1： $O\left(\left(\frac{((n-3)+2)!}{2!(n-3)!}\right)^n\right)$
  若每個Row有三個分開的1： $O(1)$

  由此可知，當不同小組的數量介於中間時，會有最多可能，整個時間複雜度太多可能性，但可以看得出來是呈指數成長
- **空間複雜度**
  Worse Case 搜到最後一個盤面才找到答案 $O\left(n^n\right)$

BFS搜尋利用C程式撰寫，執行速度較快，在5*5中可以秒解（可能性太少），可是到10*10就須花很多時間，更不用說是15*15。

- **DFS**
  這部分改用Python寫，分群概念如上述BFS，只是在搜尋時有不同的差別。
  一開始一樣先利用Row的規則找出一個起始盤面，將不同的Row分別找出各自的所有排列可能，再利用Backtracking的方式執行DFS搜尋。

1	0	0	0	0
1	0	1	1	0
1	0	1	0	1
1	1	1	0	0
1	1	0	1	1



- **時間複雜度分析**
  由上面的例子直接搜到底不加任何Cut-Off： $T = 5*3*1*3*1 = 45$
  General Case: 跟上面BFS差不多 $O(n^n)$
  5*5可以秒解，但10*10需要跑非常久（半小時以上）
- **空間複雜度**
  僅有所有可能的排列盤面：5+3+1+3+1=13
  General case: $O(n^2)$

- **DFS ＋ Cut-off (Best Search 類似剪枝概念)**
  前幾個步驟剛 DFS一模一樣，先利用Row規則，產生出每個Row可能的排列組合盤面，只是在做Backtracking時，每往下一層就檢查一次，如當下就發現不符合Column規則，則停止搜索直接Backtrack，通常在兩三步就會發現錯誤不再繼續搜尋下去，降低搜索的深度讓速度大幅提升。
  5*5可以秒解，10*10大概需要1-3秒的時間（有些case會比較久一點點）
  15*15需要更多時間

Conclusion:
DFS可以降低大量的空間複雜度，增加Cut-Off機制可以讓搜尋時間大幅的降低，不同的定盤面方法會有不同的解法，我這邊用分組的方式有點像是直接把Huristic加進去，如果要讓速度再提升應該要想辦法讓排列數再降低，例如可以利用一些知識先將確定的點加進去，這樣可以把一些排列方法排除，讓速度提升。