N –puzzle 報告　　B06505004 莊博翰

****

*　Explain briefly how you represented the Board data type.

在 code 中 Board 是以 1d Array 的形式儲存，　exp:　3 by 3 board

(0,0) ➔ 0 , (0,1) ➔1 , (0,2) ➔2 , (2,0) ➔3, (2,1) ➔4 …… , ,

為此我另外寫了

```
private int conv2dTo1d(int r, int c)
```

```
private int[] conv1dTo2d(int i)
```

兩個 function 來對 1d(儲存形式)　,2d(實際的 Board) 的 index 進行轉換

****

*　Explain briefly how you represented a search node

*　(board + number of moves + previous search node).

****

```
private class Node implements Comparable<Node>
    Board board;
    int move;
    Node previous;
```

基本上存了三樣東西，第一:是此時的 Board 的狀況，所有判斷都和這有關

第二是 move 記錄了由起始須走幾步到達現在 board 的 state，

第三是上一步的 Node，為了最後找到 answer 時可以循線找回走過的路徑

****

*　Explain briefly how you detected unsolvable puzzles.

*

*　What is the order of growth of the running time of your isSolvable()

*　method in the worst case as function of the board size n? Use big Theta

*　notation to simplify your answer, e.g., Theta(n log n) or Theta(n^3).

****

分成兩種情況，Board 邊長是奇數和邊長是偶數，不過我們都須得到一個 number of inverse : 定義如下

row-major order: 1 2 3 4 6 8 5 7
3 inversions: 6-5, 8-5, 8-7

row-major order: 1 3 4 2 5 7 8 6
4 inversions: 3-2, 4-2, 7-6, 8-6

用我們的 1d array 表示法由左向右依序選定一個值，在看他又邊有無 inverse

```
private int numberofInv() {
    int count = 0;
    for (int i = 0; i < board.length; i++) {
        for (int j = i; j < board.length; j++) {
            if (board[i] > board[j] && board[j] != 0) {
                count++;
            }
```

然後 board edge length 奇數時: 看看 number of inverse 是否為偶數
board edge length 偶數時，檢查 number of inverse+ zero 在第幾行是否為偶數

run time: Theta $((N * N)^2)$ N by N board length N*N , double loop ➔ Theta (N^4)

****

* For each of the following instances, give the minimum number of moves to
* solve the instance (as reported by your program). Also, give the amount
* of time your program takes with both the Hamming and Manhattan priority
* functions. If your program can't solve the instance in a reasonable
* amount of time (say, 5 minutes) or memory, indicate that instead. Note
* that your program may be able to solve puzzle[xx].txt even if it can't
* solve puzzle[yy].txt and xx > yy.

****

Manhattan

```
filename                          moves      time
-----------------------------------------------
puzzle28.txt                         28      0.02
puzzle30.txt                         30      0.02
puzzle32.txt                         32      0.37
puzzle34.txt                         34      0.04
puzzle36.txt                         36      0.63
puzzle38.txt                         38      0.31
puzzle40.txt                         40      0.29
puzzle42.txt                         42      0.57
puzzle44.txt                         44      0.06
puzzle46.txt                         46      0.29
puzzle48.txt                         48      1.82
puzzle50.txt                         50     10.77
```

Hamming

```
filename                       moves     time
---------------------------------------
puzzle28.txt                      28     0.26
puzzle30.txt                      30     0.23
puzzle32.txt                      32    21.38
Exception in thread "main" java.lang.OutOfMemoryError: Java heap space
```

instead

```
filename                          moves      time
-----------------------------------------------
puzzle18.txt                         18      0.01
puzzle20.txt                         20      0.01
puzzle22.txt                         22      0.02
puzzle24.txt                         24      0.05
puzzle26.txt                         26      0.07
puzzle28.txt                         28      0.14
puzzle30.txt                         30      0.17
puzzle32.txt                         32     23.18
```

****

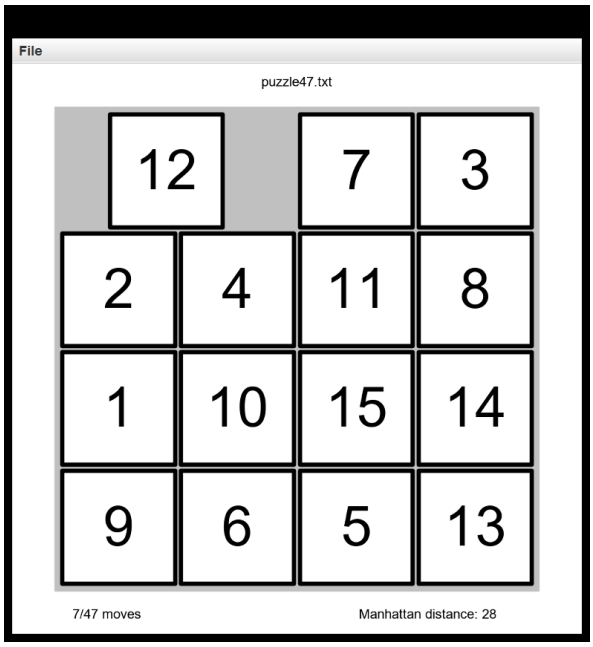
* If you wanted to solve random 4-by-4 or 5-by-5 puzzles, which
* would you prefer: a faster computer (say, 2x as fast), more memory
* (say 2x as much), a better priority queue (say, 2x as fast),
* or a better priority function (say, one on the order of improvement
* from Hamming to Manhattan)? Why?

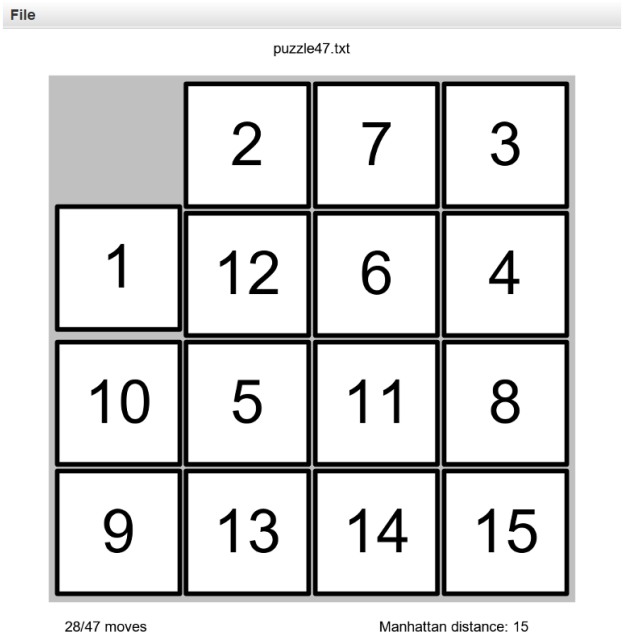
****

如果是我的話，我最想要的是 better priority function,由以上的實驗可以看到，
Manhattan 比 Hamming 快上不只一個 order ，memory 使用也少很多
而且我們會發現，在 step 數提升後，使用的時間的增長速度極快
我認為這不是比較快的電腦或比較大的記憶體可以 handle 的。
至於快速的 priority queue 也依舊無法解決 memory 的問題。
總結來說 better priority function 因該是合理的選擇。

appendix

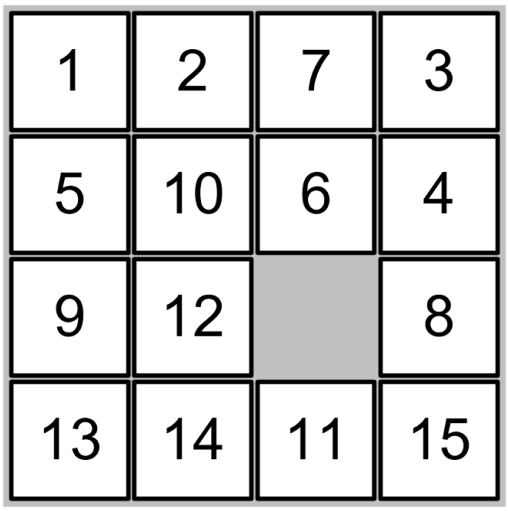
執行截圖





File

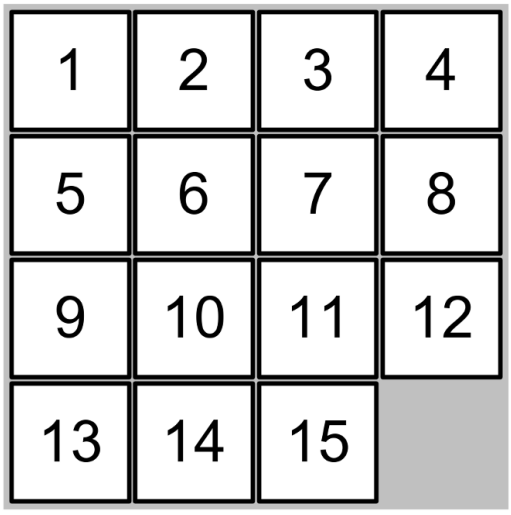
puzzle47.txt



37/47 moves Manhattan distance: 10

File

puzzle47.txt



47/47 moves Manhattan distance: 0