






一、軟硬體規格

電腦：MacBook Air (13-inch, Early 2014)

處理器	1.4GHz 雙核心 Intel Core i5 -4260U (Turbo Boost 可達 2.7GHz) , 配備 3MB 共享 L3 快取
記憶體	4 GB 1600 MHz DDR3
啟動磁碟	Macintosh HD
顯示卡	Intel HD Graphics 5000 1536 MB
作業系統	macOS Sierra 10.12.4
編譯器	Xcode Version 7.3.1

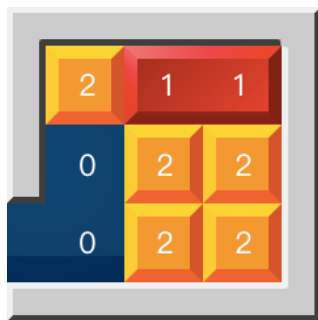
二、輸入檔

盤面以二維陣列儲存，不同種類的方塊以不同數字表示，紅色方塊為移出之目標方塊。

1	2	3	4	5
				

難度一、二和三為樂和遊戲網站上的盤面。

難度一的表示方法如下：



輸入之 txt 檔：

第一行有兩個數字 M, N , 接下來有 M 行，每行有 N 個數字，如下所示。

```

3 3
2 1 1
0 2 2
0 2 2

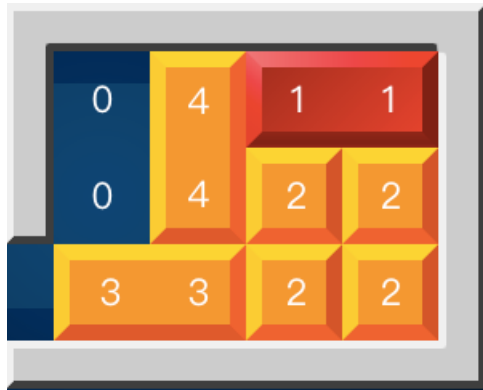
```

```

int state[3][3]={2,1,1},{0,2,2},{0,2,2}};

```

難度二：（ 難度二、三的輸入檔格式同難度一 ）



難度三：



難度四、五為另外測試用的輸入檔。

難度四

```

4 6
2 2 2 2 1 1
3 3 2 2 2 2
2 2 2 4 2 4
0 0 2 4 2 4

```

難度五

```

5 4
2 4 5 5
2 4 5 5
0 2 3 3
2 0 2 2
0 2 3 3

```

三、 執行方法

以 Xcode 執行 IDS.xcodeproj 及 IDAStar.xcodeproj。非 OSX 系統可能需要用 gcc 或其他編譯器執行 IDS 和 IDAStar 下的 main 檔。

四、 IDS 及 IDA* 之實作

1. 使用資料結構

```

struct Point
{
    int x=0, y=0;
};

map<string, int> visitedSet; // for checking duplicated states
int vnodes=0; // count all visited nodes

int M=3;
int N=3;

```

Point：用來存 x, y 座標。

visitedSet：儲存已拜訪過的 state。key 為 2D state 資料轉為 string，value 存

Depth。

vnodes：計算總共拜訪過幾個 state。

```

class State
{
public:
    int state[10][10]={2,1,1},{0,2,2},{0,2,2}};
    string ss; // change 2d array state to string
    int depth=0;

```

設一個 State 的 class，裡面存有盤面的二維陣列、轉為 string 盤面及深度。

2. 重複盤面的解決方法

```

int canAdd2Frontier()
{
    auto it=visitedSet.find(ss);
    if(it==visitedSet.end()) //not found in closed set
    {
        return 1;
    }
    else if(it->second>depth)//state in closed set is deeper
    {
        return 1;
    }
    return 0;
}

```

在每次展開新盤面時，先檢查是否要加入待展開的 deque 裡。

有兩個情況能加入：

- 1) 不在已展開的 set 裡。
- 2) 在已展開的 set 裡但 depth 較淺。

3. 展開盤面

```

void expand(deque<State> *frontier) // expand new states
{
    Point p0[2];

    findzero(p0);

    int x[2]={0},y[2]={0};
    x[0]=p0[0].x;
    y[0]=p0[0].y;
    x[1]=p0[1].x;
    y[1]=p0[1].y;

    for(int i=0; i<2; ++i) // there are two blanks, so check two times
    {
        if(x[i]!=M-1)//not last row, can move up
        {
            // printf("1\n");
            if(state[x[i]+1][y[i]]!=0) /*
            {
                State s(state,depth+1);

                if(state[x[i]+1][y[i]]==2) /*
            {

```

...

首先找出兩個空白的位置，並檢查此兩個空白的上下左右能否移動，如果可以

且不是重複盤面，則加入待展開盤面，此處程式碼較長故不贅述。

4. IDS

```
int dfs(int limit)
{
    deque<State> frontier;
    frontier.push_back(*this);
    setVisited();

    while(!frontier.empty())
    {
        if(frontier.back().goal()==1)
        {
            printf("Solution Found:\n");
            frontier.back().print();
            frontier.clear();
            return 1;
        }
        else
        {
            if(frontier.back().depth >= limit )
            {
                //printf("over limit, pop\n");
                frontier.pop_back();
            }
            else
            {
                State tmps;
                tmps = frontier.back();
                frontier.pop_back();
                tmps.print();
                tmps.expand(&frontier);
            }
        }
    }
    if(frontier.empty()) printf("Solution Not Found\n");
    return 0;
}
```

IDS 即是限制層數的 DFS，因此函式傳入 limit，從零開始往上加，找到答案即回傳 1，反之則回傳 0。frontier 所存的是待展開的盤面，如果小於 limit 即可展開。

5. IDA*

```
int IDAstar(int gx, int &flimit, bool &ans)
{
    if(gx==0) //new round, clear all visited state
    {
        visited.clear();
        //printf("clear visited\n");
    }
    setVisited(gx);

    int hx=h2(); //count heuristic
    if(gx+hx>flimit)
    {
        //printf("gx+hx>f-limit gx=%d hx=%d\n", gx, hx);
        return gx+hx;
    }
    if(goal()) {ans=true; printf("gx(depth)= %d\n", gx); return gx;}

    int nextf=INT_MAX;

    Point p0[2];
    findzero(p0);
```

... (展開盤面)

以遞迴的方式實作 IDA*，首先判斷 gx 是否等於零，若等於零的話即為新開始的一輪，因此要將以拜訪的 state 清空。接著計算 heuristic 的值（兩種不同的 h 函式在下面說明。）假如 $g + f > f\text{-limit}$ ，不能再往下展開了，回傳 $g + h$ 。

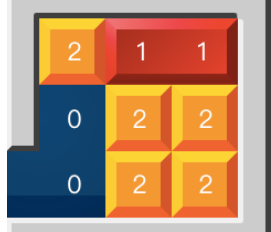
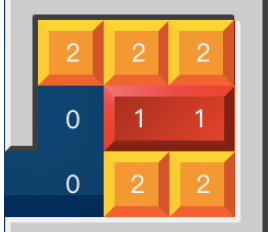
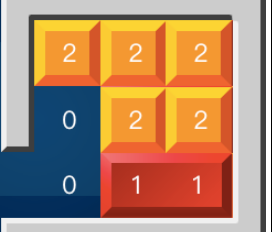
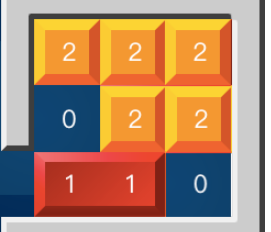
```
if(canAdd(ss, gx+1))
{
    setVisited(gx+1);
    int c = IDAstar(gx+1, flimit, ans);
    if(ans) return c;
    nextf=min(nextf,c);
}
changeback(inits);
```

每次展開新盤面後，需判斷是否能繼續展開，若能展開就繼續的回下去，直到 $h + f > f\text{-limit}$ 。若回傳的 $h + f$ 小於目前的 $next - f$ ，則將 $next - f$ 設為 $h + f$ 。

6. Heuristic function

1) Manhattan distance

判斷紅色目標方塊距離出口的 $x+y$ 值。範例如下：

$h1 = 3$	$h1 = 2$	$h1 = 1$	$h1 = 0$
			

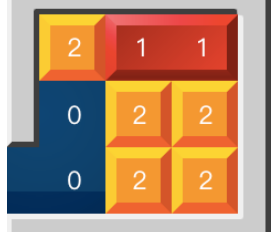
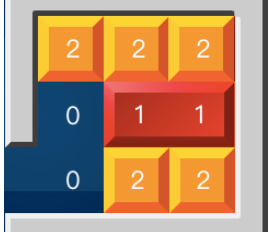
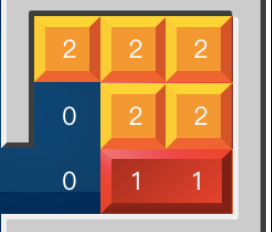
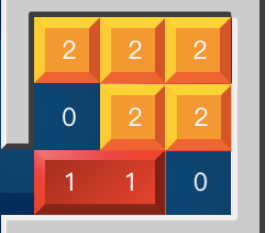
```

int h1() //Manhattan
{
    int h;
    for(int i=0; i<M; i++)
    {
        for(int j=0; j<N; j++)
        {
            if(state[i][j]==1||state[i][j]==5)
            {
                return h=abs(M-1-i)+j;
            }
        }
    }
    return 0; //wrong
}

```

2) Misplaced

回傳目標位置有幾個不是紅色方塊。範例如下：

$h1 = 2$	$h1 = 2$	$h1 = 1$	$h1 = 0$
			

```

int h2() // misplaced
{
    int m=0, ans=0;
    for(int i=0; i<M; ++i)
    {
        for(int j=0; j<N; ++j)
        {
            if(state[i][j]==1)
            {
                ans=1;
            }
            else if(state[i][j]==5)
            {
                ans=5;
            }
        }
    }

    if(ans==1)
    {
        if(state[M-1][0]!=1) m+=1;
        if(state[M-1][1]!=1) m+=1;
    }
    else if(ans==5)
    {
        if(state[M-1][0]!=5) m+=1;
        if(state[M-1][1]!=5) m+=1;
        if(state[M-2][0]!=5) m+=1;
        if(state[M-2][1]!=5) m+=1;
    }
    return m;
}

```

7. 各種盤面的測試結果

		Depth	Time (second)	Visited Nodes	Space (estimate)
難度一	IDS	15	0.032416	1340	0.51MB
	IDA* (h1)	15	0.040462	2058	0.78MB
	IDA* (h2)	15	0.045599	2061	0.78MB
難度二	IDS	32	1.967607	50293	19.18 MB
	IDA* (h1)	32	1.667854	51648	19.70 MB
	IDA* (h2)	32	1.892808	51651	19.70 MB
難度三	IDS	49	20.516617	485156	185.07MB
	IDA* (h1)	49	19.406290	493429	188.22 MB
	IDA* (h2)	49	19.792034	493434	188.23 MB

難度四	IDS	38	537.074044	9266221	3534.78 MB
	IDA* (h1)	38	442.467082	7578151	2890.83 MB
	IDA* (h2)	38	462.315164	7578267	2890.88 MB
難度五	IDS	54	340.599657	6032511	2301.22 MB
	IDA* (h1)	54	292.156200	5260917	2006.88MB
	IDA* (h2)	54	269.069846	5260945	2006.89MB

8. 結果分析

- 1) 經過盤面的 `trace` 後，發現三種算法都能得到最佳解。
- 2) 以 IDA* 的兩個不同的 `heuristic function` 來看，不管任何盤面，h1 所拜訪的節點數都比 h2 還少。因此儘管兩個函式都是可接受的，但 h1 函式還是較好。
- 3) IDS 和 IDA* 在不同盤面的效能不一定哪一種比較好，但在盤面越複雜時 IDA* 的效能會比 IDS 好。
- 4) 在難度五和難度三中，答案方塊都是 2×2 ，所需走的步數會比較多（層數較多）。

五、遇到的困難

1. 剛開始設計盤面時沒有把相同方塊設成一樣的數字，因此會有遇到重複的盤面但卻沒判斷出來的狀況，程式也會跑很久。
2. 一開始在判斷重複盤面時，忘記要考慮 `depth` 的問題，因此儘管能算出結果但卻不是 `optimal`。例如在第四層展開了一種 `state A`，將 A 加入 `visited`，之後在第二層遇到了 `state A` 就不會再展開了，但其實這條路能夠更快走到終點。之後在 `map` 把 `depth` 存在 `value` 就能避免這種情況發生，也能得到最佳解。
3. 把所有拜訪的盤面印出來發現會影響效能，不印展開過程的盤面後速度會加快。

六、參考資料

1. Basic-AI-BFS-DFS-IDS-Hill-Climbing

<https://github.com/sideris/Basic-AI-BFS-DFS-IDS-Hill-Climbing/blob/master/8-puzzle.cpp>

(參考使用以 IDS 解 8-puzzle。)

2. Algorithm Wiki - Iterative deepening depth-first search

http://will.thimbleby.net/algorithms/doku.php?id=iterative_deepening_depth-first_search

(以動畫圖示來一步一步解釋 IDDFS。)

3. 演算法筆記 - 應用：8 Puzzle Problem

<http://www.csie.ntnu.edu.tw/~u91029/State.html#4>

(參考使用 IDA*解 8-puzzle。)