西安交通大學

XI'AN JIAOTONG UNIVERSITY

人工智能实验报告

重排九宫

课程名称：	人工智能
姓名：	计算机 001 班曾锦程
学院：	电信学部
专业：	计算机科学与技术
学号：	2203613040
指导老师：	相明

2023 年 2 月 1 日

西安交通大學实验报告

专业：计算机科学与技术
姓名：计算机 001 班曾锦程
学号：2203613040
日期：2023 年 2 月 1 日
地点：

课程名称：人工智能　　指导老师：相明　　成绩：
实验名称：重排九宫　　实验类型：设计实验　　同组学生姓名：无

一、 问题描述

“重排九宫”问题即八数码移动问题，在网站上、文曲星游戏里随处可见，游戏规则：在 $3*3$ 的九宫棋盘上摆有八个分别刻有 1-8 八个数字的将牌，棋盘中留有一个空格。允许其周围的某一个将牌向空格移动，这样通过移动将牌从给定的初始布局转变到目标布局。

二、 算法描述

1. 棋局的形式化定义

本实验将棋局定义为字符串 *seq*，如’12345678*’（图 1 所示）。

1	2	3
8		4
7	6	5

图 1: 12345678*

2. 关于可解性分析

定理 1: 称数列相邻两个数码互相交换为一次邻换，则邻换改变数排逆序数的奇偶性。

推论 1: 设数列 A 经过 n 次邻换后得到数列 B，则 n 为偶数时，R(A) 与 R(B) 奇偶性相同，n 为奇数时，R(A) 与 R(B) 奇偶性相反。

定理 2: 八数码棋局中，将牌的移动不改变状态数列逆序数的奇偶性。

推论 2: 状态数列逆序数为奇数的棋局，无论怎样移动，都无法得到状态数列逆序数为偶数的棋局，反之亦然。

结论: 起始状态数列必须与目标状态数列奇偶性相同，否则无解。

判断逆序数奇偶性时，将 * 替换成 0，来进行可解性判断。

3. 棋局操作的定义

一共四种操作：将空格上移，将空格下移，将空格左移，将空格右移，但是，并非每一个状态都可运用这四种操作，实际上，只有中心位置是空格时才可同时应用这些操作。可以通过对空格定位来实现移动操作。如果空格在四个角上，则只有两种移动操作；如果空格在单独一条边上且不在角上，则只有三种移动操作；如果空格在中间则有四种操作。

(1)　空格定位的判断

首先在字符串中寻找空格，返回下标 i，对 i 进行计算即可得到空格坐标 (x,y):

$$x = i \ mod \ 3, \ y = i \ / \ 3$$

坐标系如图 2 所示:

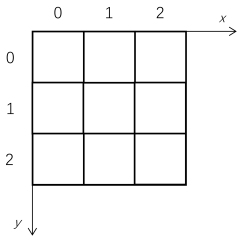


图 2: 棋局坐标系

(2)　移动操作的实现

(a) 向上移动:

$$seq[i] = seq[i-3], seq[i-3] =' *'$$

(b) 向下移动:

$$seq[i] = seq[i+3], seq[i+3] =' *'$$

(c) 向左移动:

$$seq[i] = seq[i-1], seq[i-1] =' *'$$

(d) 向右移动:

$$seq[i] = seq[i+1], seq[i+1] =' *'$$

4. 数据结构的定义

(1)　Open 表

Open 表中即存储了未扩展的状态，都以字符串或者元组 (包含字符串、深度、估值函数等) 的形式存储。由于不同算法中对选择状态的优先级不同，Open 表的数据结构也不相同。在广度优先算法中使用的是 FIFO 队列，深度优先算法使用的是 LIFO 栈，而启发式算法中则是以估值函数 $f(x)$ 为基准的优先队列。

(2) Close 表

Close 表中即存储了已扩展的状态，都以字符串的形式存储。用于限制扩展，如果扩展后的子状态已经在 Close 表中，则不予加入 Open 表，即状态判重。用于回溯移动路径。

(3) Pos 表

由于 python 中没有指针的概念，设计时加入一个 Pos 表，记录了 Close 表中节点的父节点在 Close 表中的位置，用于回溯移动路径，也可以把元组 (字符串和父节点位置) 加入 Close 表替代此表。

5. 不同算法的步骤

(1) 广度优先算法

1. 把初始节点 S_0 放入 OPEN 表。
2. 如果 OPEN 表为空，则问题无解，退出。
3. 把 OPEN 表的第一个节点取出放入 CLOSE 表（记为节点 n）。
4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 n 不可扩展，则转第 2 步。
6. 扩展节点 n，将其子节点放入 OPEN 表的尾部，并为每一个子节点都记录父节点，然后转第 2 步。

(2) 深度优先算法

1. 把初始节点 S_0 放入 OPEN 表。
2. 如果 OPEN 表为空，则问题无解，退出。
3. 把 OPEN 表的第一个节点取出放入 CLOSE 表（记为节点 n）。
4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 n 不可扩展，则转第 2 步。
6. 扩展节点 n，将其子节点放入 OPEN 表的首部，并为每一个子节点都记录父节点，然后转第 2 步。

(3) 有界深度优先算法

1. 把初始节点 S_0 放入 OPEN 表。
2. 如果 OPEN 表为空，则问题无解，退出。
3. 把 OPEN 表的第一个节点取出放入 CLOSE 表（记为节点 n）。
4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 n 的深度 $d(n) = d_{max}$，则转第 2 步（此时节点 n 位于 CLOSE 表，但并未进行扩展）。
6. 若节点 n 不可扩展，则转第 2 步。
7. 扩展节点 n，将其子节点放入 OPEN 表的首部，并为每一个子节点都记录父节点，将每一个子节点的深度设置为 $d(n) + 1$，然后转第 2 步。

(4) 代价树广度优先算法

1. 把初始节点 S_0 放入 OPEN 表，令 $g(S_0) = 0$
2. 如果 OPEN 表为空，则问题无解，退出。
3. 把 OPEN 表的第一个节点取出放入 CLOSE 表（记为节点 n）。
4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 n 不可扩展，则转第 2 步。
6. 扩展节点 n，为每一个子节点都记录父节点，计算各子节点的代价，并将各子节点放入 OPEN 表中。按各节点的代价对 OPEN 表中的全部节点进行排序 (按从小到大的顺序)，然后转第 2 步。

在本问题中，$g(x) = d(x)$, $d(x)$ 为节点的深度（也是移动空格的次数），这里退化成广度优先算法。

(5) 代价树深度优先算法

1. 把初始节点 S_0 放入 OPEN 表，令 $g(S_0) = 0$
2. 如果 OPEN 表为空，则问题无解，退出。
3. 把 OPEN 表的第一个节点取出放入 CLOSE 表（记为节点 n）。
4. 考察节点 n 是否为目标节点。若是，则求得了问题的解，退出。
5. 若节点 n 不可扩展，则转第 2 步。
6. 扩展节点 n，将其子节点按代价从小到大的顺序放到 OPEN 表中的首部，并为每一个子节点都记录父节点，然后转第 2 步。

在本问题中，$g(x) = d(x)$, $d(x)$ 为节点的深度（也是移动空格的次数），这里退化成深度优先算法。

(6) 局部择优算法

局部择优算法则是将代价函数替换为估价函数形成启发式算法，步骤与代价树深度优先算法相同:

$$f(x) = g(x) + h(x)$$

(7) 全局择优算法/A* 算法

全局择优算法 (在这个问题中全局择优算法与 A* 算法相同，因为使用的启发函数都满足下界条件，代价函数也符合条件) 则是将代价函数替换为估价函数形成启发式算法，步骤与代价树广度优先算法相同:

$$f(x) = g(x) + h(x)$$

6. 启发式函数的设置

(1) 不在位将牌数

即当前状态与目标状态对应位置逐一比较后有差异的将牌总个数。该函数满足下界条件。

(2) 将牌离“家”距离之总和

设 x_i 为将牌 i 的横坐标，y_i 为将牌 i 的纵坐标，则该函数表示为:

$$distance = \sum_{i=0}^{8}[|x_i(source) - x_i(destination)| + |y_i(source) - y_i(destination)|]$$

该函数满足下界条件。

(3)　几种函数的加权和

如果分别对以上几种函数赋予一定的权值，相加后作为最后的评价函数，或许能取各方所长，有很好的综合效果。

三、　实验结果

1.　课本例子求解

对于初始序列'2831*4765' 几种算法的结果如下：

(1)　广度优先算法/代价树广度优先算法

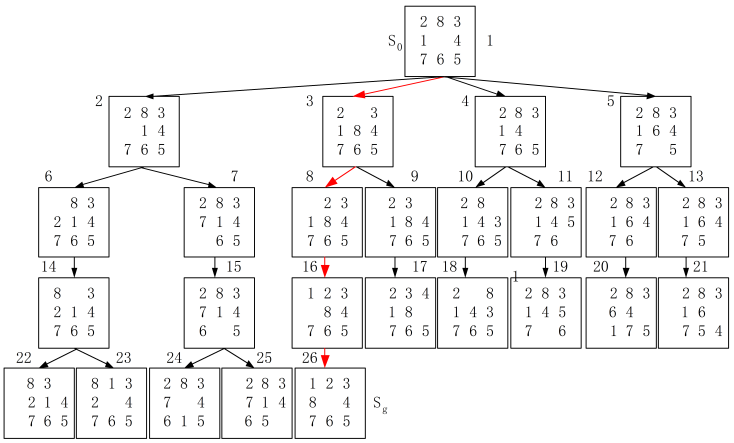


图 3: 广度优先算法/代价树广度优先算法

(2)　深度优先算法/代价树深度优先算法

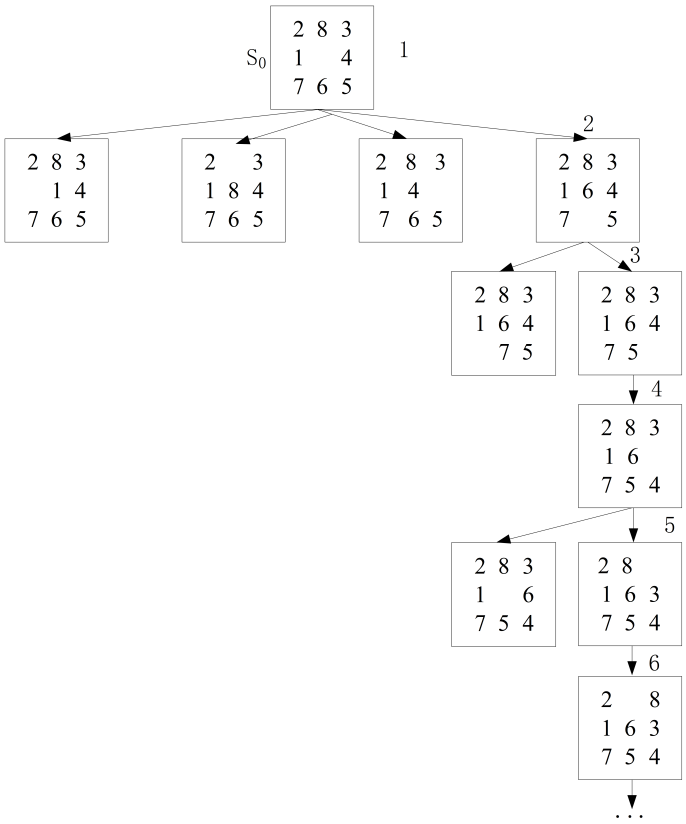


图 4: 深度优先算法/代价树深度优先算法

(3)　有界深度优先算法

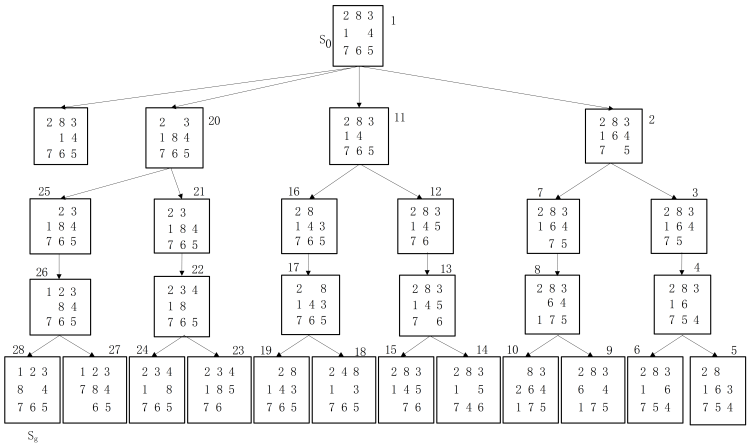


图 5: 有界深度优先算法

(4)　局部择优算法

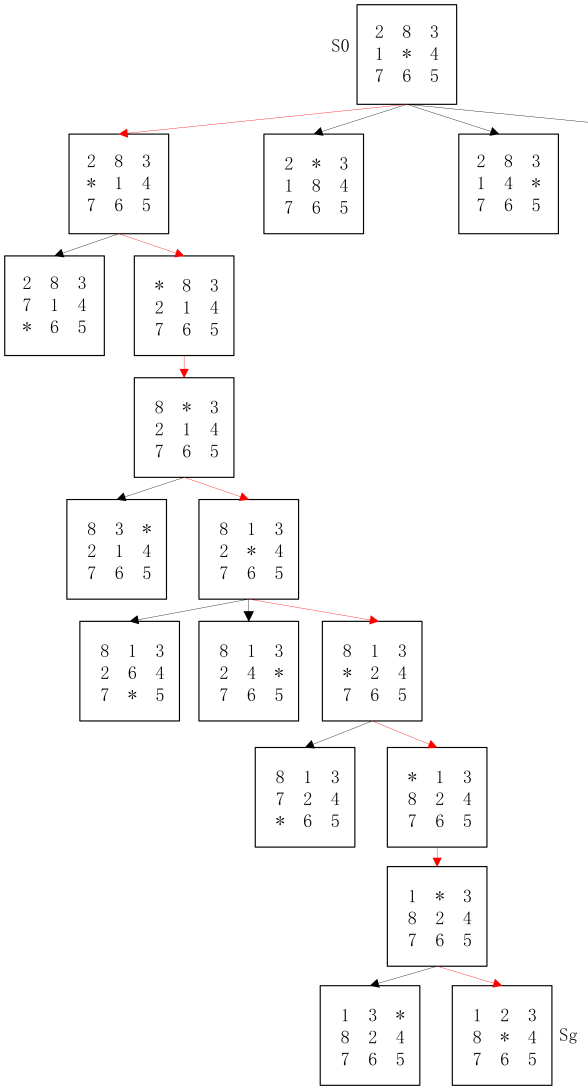


图 6: 局部择优算法

(5)　全局择优算法/A* 算法

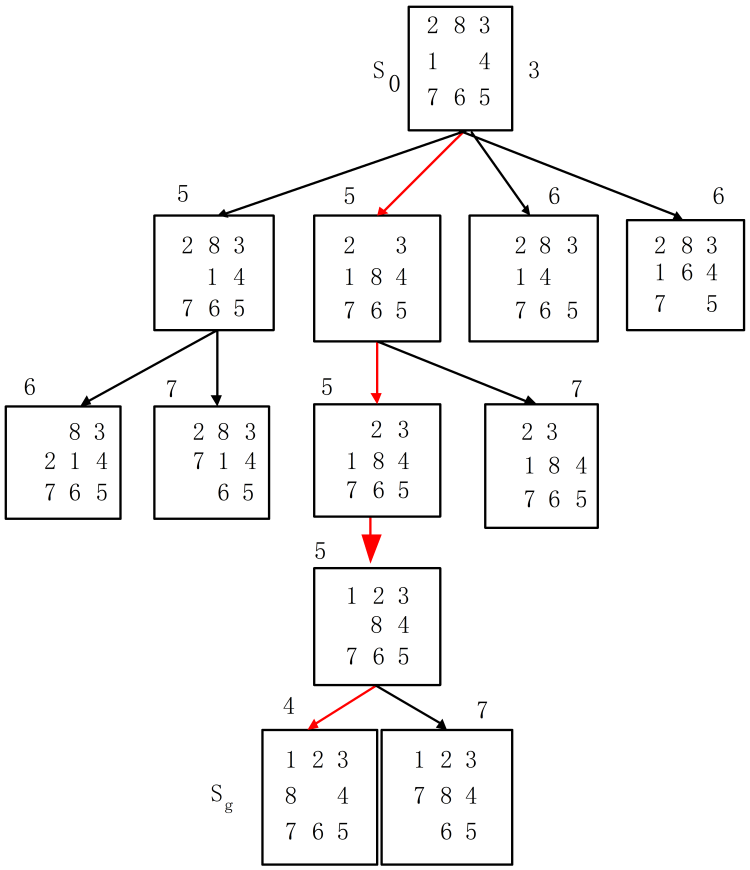


图 7: 全局择优算法/A* 算法

2.　不同启发函数测试

目标序列为 1238*4765，启发式函数 (1)(2)(3)，如表 1，2，3 所示

表 1: **启发式函数：不在位将牌数**

随机序列	Close 表大小	路径长度	运行时间
41*523867	15536	25	4.023s
51762348*	13957	25	3.095s
8645*3217	7291	23	0.845s
...	...	...	...
*36178254	1022	19	0.028s
63578142*	5549	23	0.510s

表 2: **启发式函数：将牌离“家”距离之总和**

随机序列	Close 表大小	路径长度	运行时间
41*523867	1747	25	0.076s
51762348*	531	25	0.013s
8645*3217	547	23	0.015s
...	...	...	...
*36178254	106	19	0.002s
63578142*	224	23	0.004s

表 3: **启发式函数：几种函数的加权和 (权重比 1:1)**

随机序列	Close 表大小	路径长度	运行时间
41*523867	5014	25	0.524s
51762348*	2616	25	0.154s
8645*3217	1914	23	0.091s
...	...	...	...
*36178254	313	19	0.009s
63578142*	1230	23	0.046s

四、 实验结果分析

1. 完备性与搜索效率分析

(1) 广度优先算法/代价树广度优先算法

完备性：只要问题有解，用广度优先搜索总可以得到解，而且得到的是路径最短的解。

搜索效率：广度优先搜索盲目性较大，当目标节点距初始节点较远时将会产生许多无用节点，搜索效率低。

(2) 深度优先算法/代价树深度优先算法

完备性：在深度优先搜索中，搜索一旦进入某个分支，就将沿着该分支一直向下搜索。如果目标节点不在此分支上，而该分支又是一个无穷分支，则就不可能得到解。所以深度优先搜索是不完备的，而且即使问题有解，它也不一定能求得解。而且用深度优先求得的解，不一定是路径最短的解。

搜索效率：如果目标节点恰好在此分支上，则可较快地得到解。

(3) 有界深度优先算法

完备性：如果问题有解，且其路径长度 dm，则上述搜索过程一定能求得解，且路径最短。但是，若解的路径长度 >dm, 则上述搜索过程就得不到解。

搜索效率：如果能找到解，搜索效率将会比广度优先算法高，但是 dmax 很难给出, 可以利用步进式的方法求解，但是时间会更长。

(4)　局部择优算法

完备性：使用启发式函数，对扩展节点进行排序，实质上仍然是深度优先算法为主体，搜索一旦进入某个分支，就将沿着该分支一直向下搜索。如果目标节点不在此分支上，而该分支又是一个无穷分支，则就不可能得到解，而且求得的解，不一定是路径最短的解。但是可求解性相对于深度优先算法会提高很多（对于课本例子，局部择优能求得解，但是深度优先不行）。

搜索效率：如果目标节点恰好在此分支上，则可较快地得到解，且比深度优先算法更快。

(5)　全局择优算法/A* 算法

完备性：如果启发式函数满足下界条件，则能保证 A* 总可以得到解，而且得到的是路径最短的解（A* 算法的可纳性）

搜索效率：相对于广度优先算法有很大的提高，代价函数有利于搜索的完备性，但影响搜索的效率。启发式函数有利于提高搜索的效率，但影响搜索的完备性。

2.　启发式对比分析

(1)　Close 表大小

如图 8 所示:

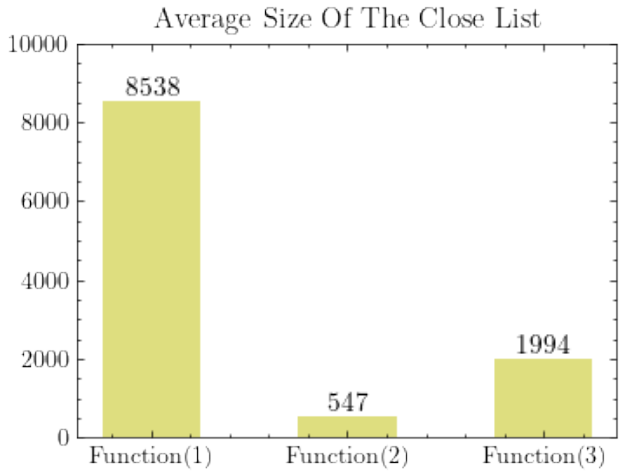


图 8: 平均 Close 表大小

可以看出效率:

启发式函数 (2) > 启发式函数 (3) > 启发式函数 (1)

(2)　路径长度大小

由于启发式函数都满足下界条件，路径大小都相同。

(3)　运行时间

如图 9 所示:

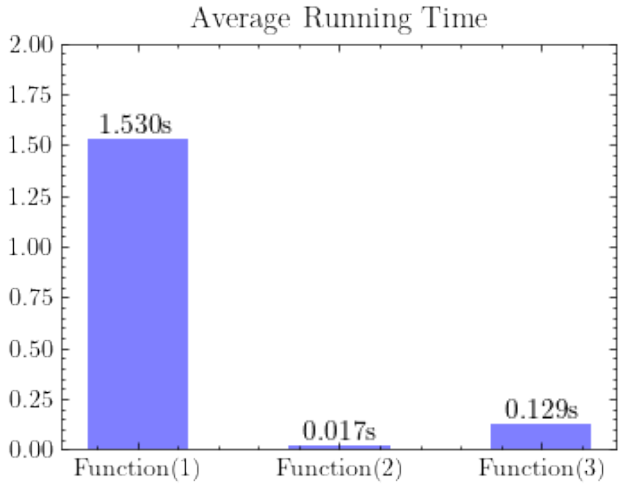


图 9: 平均运行时间

可以看出效率:

$$\text{启发式函数 (2)} > \text{启发式函数 (3)} > \text{启发式函数 (1)}$$

(4)　结论

总的来说，启发式函数 (2) 相比于启发式函数 (1) 的启发式信息更加准确，信息更足，导致启发式函数 (2) 的 A* 算法性能更加优秀，启发式函数 (3) 为启发式函数 (1)(2) 的平均值，在两者之间。启发式函数 (1) 的启发式信息最低，所以性能也最差。**也就是说可以通过挖掘更加准确的启发式函数来提高算法的搜索效率，但必须满足下界条件来确保最优解。**

五、　源代码

重排九宫实验

Step1:定义表示方式

定义重排九宫中的空格为*,即内容可以表示为12345678*

In [1]:

```
#seq = str(input('请输入序列'))
seq = '2831*4765'
```

定义函数move来模拟空格在九宫中移动

In [2]:

```
def move(seq,direction):
    pos = seq.find('*')
    seq = list(seq)
    judge = 0
    if direction == 'left':
        if pos%3 == 1 or pos%3 == 2:
            seq[pos] = seq[pos-1]
            seq[pos-1] = '*'
        else:
            judge = 1
    elif direction == 'up':
        if pos >= 3:
            seq[pos] = seq[pos-3]
            seq[pos-3] = '*'
        else:
            judge = 1
    elif direction == 'down':
        if pos <= 5:
            seq[pos] = seq[pos+3]
            seq[pos+3] = '*'
        else:
            judge = 1
    elif direction == 'right':
        if pos%3 == 0 or pos%3 == 1:
            seq[pos] = seq[pos+1]
            seq[pos+1] = '*'
        else:
            judge = 1
    seq = ''.join(seq)
    return seq,judge
```

设置函数judgement来判断该问题是否有解,从而加速搜索过程，不是必须通过检测open表是否为空来确定是否有解

- 由数学知识可知，可计算这两个有序数列的逆序值，如果两者都是偶数或奇数，则可通过变换到达，否则，这两个状态不可达。这样，就可以在具体解决问题之前判断出问题是否可解，从而可以避免不必要的搜索。

In [3]:

```
def judgement(seq):
    seq = seq.replace('*','0')
    seq1 = '123804765'
    ans1 = 0
    ans2 = 0
    for i in range(len(seq)):
        for j in range(i):
            if seq[j] > seq[i]:
                ans1 += 1
    for i in range(len(seq)):
        for j in range(i):
            if seq1[j] > seq1[i]:
                ans2 += 1
    if ans1%2 == ans2%2:
        return True
    else:
        return False
```

In [4]:

```
seq = '12345678*'
judgement(seq)
```

Out[4]:

```
False
```

In [5]:

```
seq = '2831*4765'
judgement(seq)
```

Out[5]:

True

定义函数show来显示九宫

In [6]:

```
import matplotlib.pyplot as plt
import numpy as np
```

In [7]:

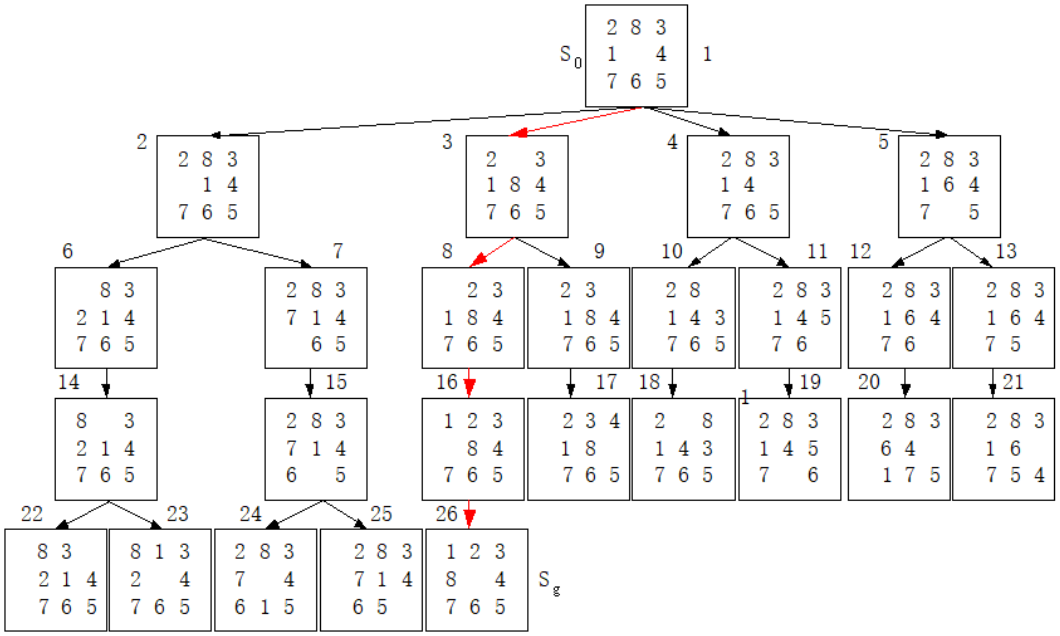
```
def show(seq):
    seq = seq.replace('*',' ')
    seq = list(seq)
    plt.figure(figsize=(3,3),dpi=40)
    plt.xlim(0,3)
    plt.ylim(0,3)
    plt.axis('off')
    x = np.arange(0,4,1)
    y = np.arange(0,4,1)
    x1 = np.ones(y.shape)*1
    x2 = np.ones(y.shape)*2
    x3 = np.ones(y.shape)*0
    x4 = np.ones(y.shape)*3
    y1 = np.ones(x.shape)*1
    y2 = np.ones(x.shape)*2
    y3 = np.ones(x.shape)*0
    y4 = np.ones(x.shape)*3
    plt.plot(x,y1,color='k',linewidth=1)
    plt.plot(x,y2,color='k',linewidth=1)
    plt.plot(x,y3,color='k',linewidth=3)
    plt.plot(x,y4,color='k',linewidth=3)
    plt.plot(x1,y,color='k',linewidth=1)
    plt.plot(x2,y,color='k',linewidth=1)
    plt.plot(x3,y,color='k',linewidth=3)
    plt.plot(x4,y,color='k',linewidth=3)
    plt.text(0.25,2.25,f'{seq[0]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(1.25,2.25,f'{seq[1]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(2.25,2.25,f'{seq[2]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(0.25,1.25,f'{seq[3]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(1.25,1.25,f'{seq[4]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(2.25,1.25,f'{seq[5]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(0.25,0.25,f'{seq[6]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(1.25,0.25,f'{seq[7]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.text(2.25,0.25,f'{seq[8]}',fontsize=30, ha='left', rotation=0, wrap=True)
    plt.show()
show(seq)
```

2	8	3
1		4
7	6	5

Step2:各种算法实现

(1) 广度优先算法

- 把初始节点S0放入OPEN表。
- 如果OPEN表为空，则问题无解，退出。
- 把OPEN表的第一个节点取出放入CLOSE表（记为节点n）。
- 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 若节点n不可扩展，则转第2步。
- 扩展节点n，将其子节点放入OPEN表的尾部，并为每一个子节点都配置指向父节点的指针，然后转第2步。(只考虑有效子节点，见一般搜索过程)



In [8]:

```
import queue
```

In [9]:

```
def BFS(seq):
    open = queue.Queue()
    open.put(seq)
    number = queue.Queue()
    number.put(0)
    seqs = []
    path = []
    pos = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while not open.empty():
            process = open.get()
            seqs.append(process)
            pos.append(number.get())
            i = i + 1
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                open.put(newseq)
                number.put(i)
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                open.put(newseq)
                number.put(i)
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                open.put(newseq)
                number.put(i)
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                open.put(newseq)
                number.put(i)
        while last != 0:
            path.append(seqs[last])
            last = pos[last]-1
        path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [10]:

```
result, seqs, pos, path= BFS(seq)
```

In [11]:

```
result
```

Out[11]:

```
'1238*4765'
```

In [12]:

```
seqs
```

Out[12]:

```
['2831*4765',
 '283*14765',
 '2*3184765',
 '28314*765',
 '2831647*5',
 '*83214765',
 '283714*65',
 '*23184765',
 '23*184765',
 '28*143765',
 '28314576*',
 '283164*75',
 '28316475*',
 '8*3214765',
 '2837146*5',
 '123*84765',
 '23418*765',
 '2*8143765',
 '2831457*6',
 '283*64175',
 '28316*754',
 '83*214765',
 '8132*4765',
 '2837*4615',
 '28371465*',
 '1238*4765']
```

In [13]:

```
pos
```

Out[13]:

```
[0,
 1,
 1,
 1,
 1,
 2,
 2,
 3,
 3,
 4,
 4,
 5,
 5,
 6,
 7,
 8,
 9,
 10,
 11,
 12,
 13,
 14,
 14,
 15,
 15,
 16]
```

In [14]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

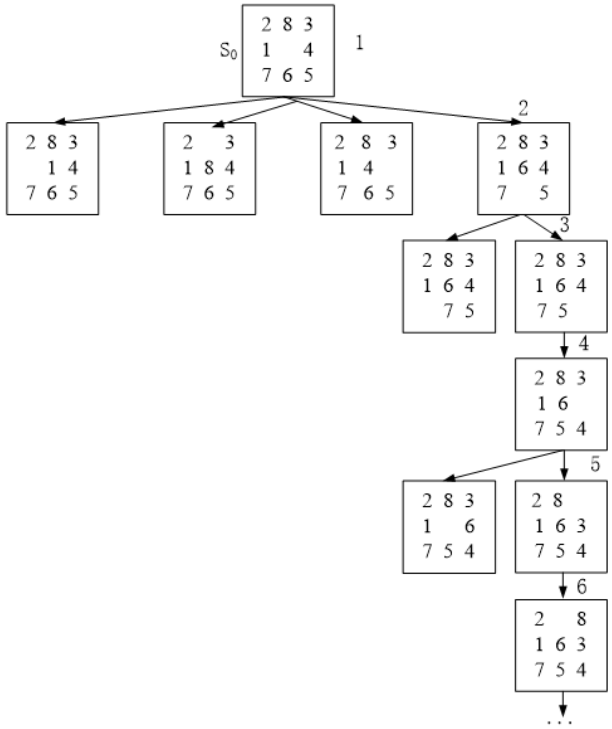
	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

(2) 深度优先算法

- 把初始节点S0放入OPEN表。
- 如果OPEN表为空，则问题无解，退出。
- 把OPEN表的第一个节点（记为节点n）取出放入CLOSE表。
- 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 若节点n不可扩展，则转第2步。
- 扩展节点n，将其子节点放入OPEN表的首部，并为每一个子节点都配置指向父节点的指针，然后转第2步。



In [15]:

```
def DFS(seq):
    open = queue.LifoQueue()
    open.put(seq)
    seqs = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while (not open.empty()) and (i <= 10):
            process = open.get()
            seqs.append(process)
            i = i + 1
            if process == '1238*4765':
                result = process
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                open.put(newseq)
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                open.put(newseq)
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                open.put(newseq)
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                open.put(newseq)
    else:
        pass
    return result,seqs
```

In [16]:

```
result,seqs = DFS(seq)
```

In [17]:

```
result
```

Out[17]:

```
'No Solution'
```

In [18]:

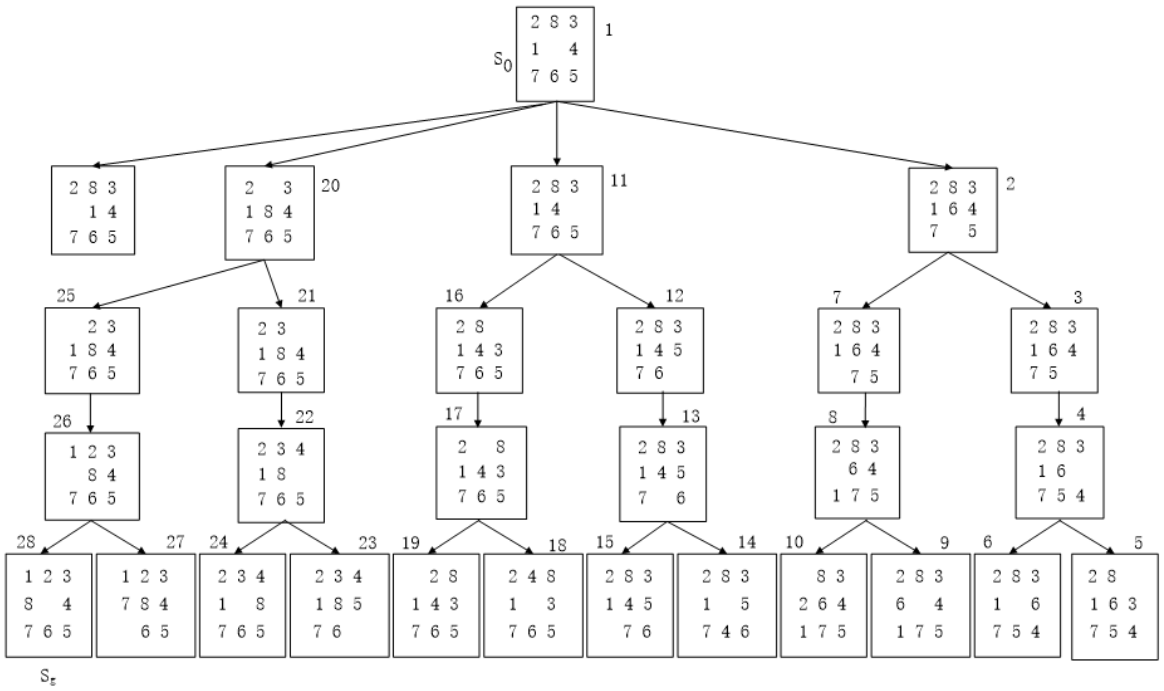
```
seqs
```

Out[18]:

```
['2831*4765',
 '2831647*5',
 '28316475*',
 '28316*754',
 '28*163754',
 '2*8163754',
 '2681*3754',
 '2681537*4',
 '26815374*',
 '26815*743',
 '26*158743']
```

(3) 有界深度优先算法

- 把初始节点S0放入OPEN表中，置S0的深度$d(S_0)=0$。
- 如果OPEN表为空，则问题无解，退出。
- 把OPEN表的第一个节点取出放入CLOSE表（记为节点n）。
- 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 若节点n的深度d(n)=dm，则转第2步（此时节点n位于CLOSE表，但并未进行扩展）。
- 若节点n不可扩展，则转第2步。
- 扩展节点n，将其子节点放入OPEN表的首部，为每一个子节点都配置指向父节点的指针，将每一个子节点的深度设置为d(n)+1，然后转第2步。


S0
1
2 8 3
1 4
7 6 5
2
2 8 3
1 6 4
7 5
3
2 8 3
1 6 4
7 5
4
2 8 3
1 6
7 5 4
5
2 8
1 6 3
7 5 4
6
2 8 3
1 6
7 5 4
7
2 8 3
1 6 4
7 5
8
2 8 3
6 4
1 7 5
9
2 8 3
6 4
1 7 5
10
8 3
2 6 4
1 7 5
11
2 8 3
1 4
7 6 5
12
2 8 3
1 4 5
7 6
13
2 8 3
1 4 5
7 6
14
2 8 3
1 5
7 4 6
15
2 8 3
1 4 5
7 6
16
2 8
1 4 3
7 6 5
17
2 8
1 4 3
7 6 5
18
2 4 8
1 3
7 6 5
19
2 8
1 4 3
7 6 5
20
2 3
1 8 4
7 6 5
21
2 3
1 8 4
7 6 5
22
2 3 4
1 8
7 6 5
23
2 3 4
1 8 5
7 6
24
2 3 4
1 8
7 6 5
25
2 3
1 8 4
7 6 5
26
1 2 3
8 4
7 6 5
27
1 2 3
7 8 4
6 5
28
1 2 3
8 4
7 6 5
Sg

In [19]:

```
def DFS_Boundary(seq,dmax = 4):
    open = queue.LifoQueue()
    depth = queue.LifoQueue()
    number = queue.LifoQueue()
    open.put(seq)
    depth.put(0)
    number.put(0)
    seqs = []
    path = []
    pos = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while not open.empty():
            process = open.get()
            seqs.append(process)
            deep = depth.get()
            pos.append(number.get())
            i = i + 1
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            if deep >= dmax:
                continue
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                open.put(newseq)
                depth.put(deep+1)
                number.put(i)
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                open.put(newseq)
                depth.put(deep+1)
                number.put(i)
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                open.put(newseq)
                depth.put(deep+1)
                number.put(i)
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                open.put(newseq)
                depth.put(deep+1)
                number.put(i)
        if result == '1238*4765':
            while last != 0:
                path.append(seqs[last])
                last = pos[last]-1
            path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [20]:

```
result,seqs,pos,path = DFS_Boundary(seq)
```

In [21]:

```
result
```

Out[21]:

'1238*4765'

In [22]:

```
seqs
```

Out[22]:

```
['2831*4765',
 '2831647*5',
 '28316475*',
 '28316*754',
 '28*163754',
 '2831*6754',
 '283164*75',
 '283*64175',
 '2836*4175',
 '*83264175',
 '28314*765',
 '28314576*',
 '2831457*6',
 '2831*5746',
 '283145*76',
 '28*143765',
 '2*8143765',
 '2481*3765',
 '*28143765',
 '2*3184765',
 '23*184765',
 '23418*765',
 '23418576*',
 '2341*8765',
 '*23184765',
 '123*84765',
 '123784*65',
 '1238*4765']
```

In [23]:

```
pos
```

Out[23]:

```
[0,
 1,
 2,
 3,
 4,
 4,
 2,
 7,
 8,
 8,
 1,
 11,
 12,
 13,
 13,
 11,
 16,
 17,
 17,
 1,
 20,
 21,
 22,
 22,
 20,
 25,
 26,
 26]
```

In [24]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

(4) 代价树广度优先搜索

- 把初始节点S0放入OPEN表，令 $g(S_0)=0$。
- 如果OPEN表为空，则问题无解，退出。
- 把OPEN表的第一个节点（记为节点n）取出放入CLOSE表。
- 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 若节点n不可扩展，则转第2步。
- 扩展节点n，为每一个子节点都配置指向父节点的指针，计算各子节点的代价，并将各子节点放入OPEN表中。按各节点的代价对OPEN表中的全部节点进行排序(按从小到大的顺序)，然后转第2步。

实际上重排九宫问题中，代价函数就应该是所移动的步数，即 $g(x)=d(x)$,其实和原本的广度优先搜索没有区别

(5) 代价树深度优先搜索

- 把初始节点S0放入OPEN表，令$g(S_0)=0$。
- 如果OPEN表为空，则问题无解，退出。
- 把OPEN表的第一个节点（记为节点n）取出放入CLOSE表。
- 考察节点n是否为目标节点。若是，则求得了问题的解，退出。
- 若节点n不可扩展，则转第2步。
- 扩展节点n，将其子节点按代价从小到大的顺序放到OPEN表中的首部，并为每一个子节点都配置指向父节点的指针，然后转第2步。

实际上重排九宫问题中，代价函数就应该是所移动的步数，即 $g(x)=d(x)$,其实和原本的深度优先搜索也没有区别

下面将代价函数替换为估价函数形成启发式算法, $d(x)$ 代表节点深度, $h(x)$ 代表节点 x 的格局与目标节点格局不相同的牌数:

$$f(x)=g(x)+h(x)=d(x)+h(x)$$

设置第一种启发式函数 $h(x)$ ，使用海明距离,不在位将牌数。

In [25]:

```
def h(seq):
    seq1 = '1238*4765'
    distance = 0
    for i in range(len(seq)):
        if seq[i] != seq1[i]:
            distance = distance + 1
        else:
            pass
    return distance
```

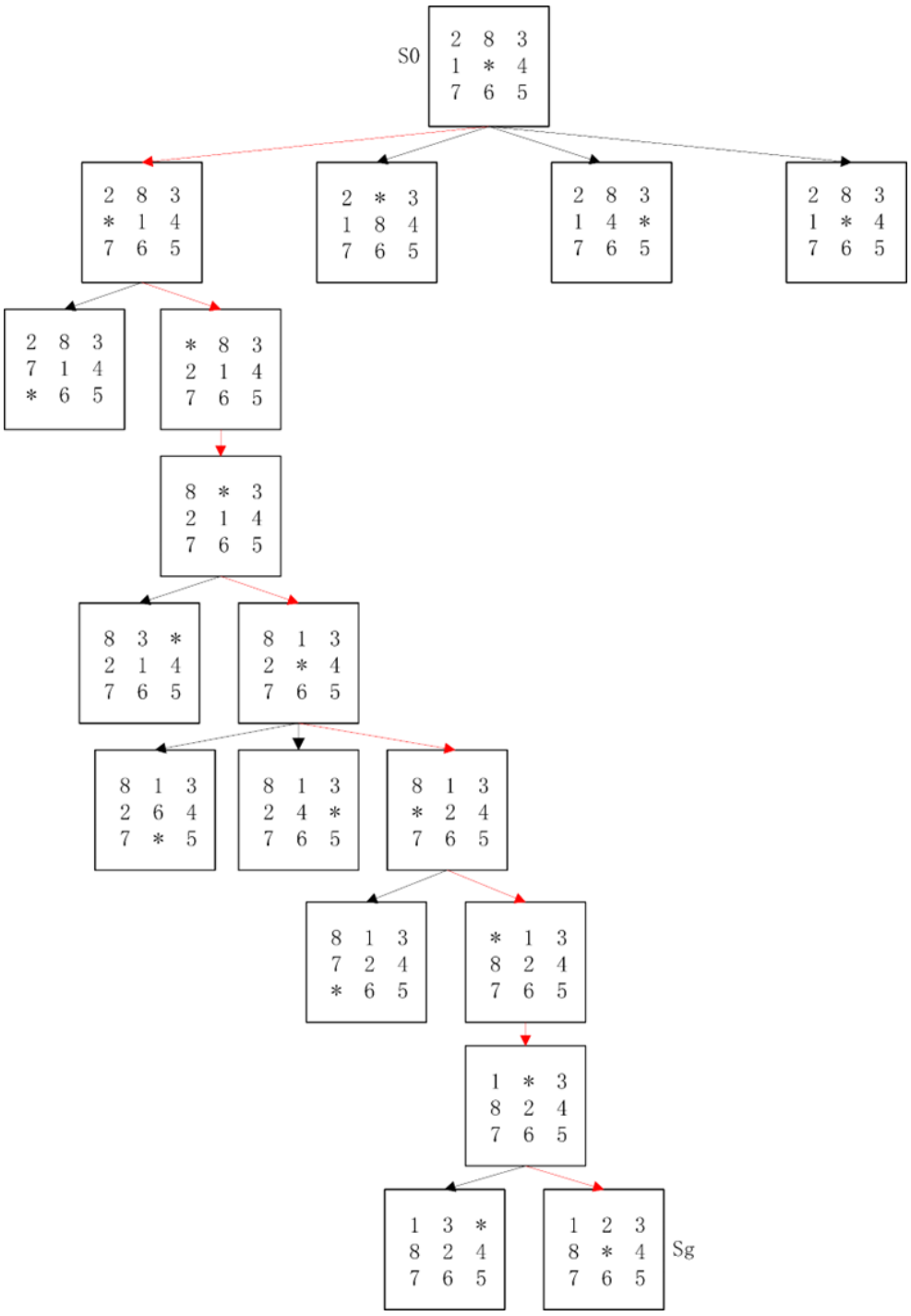
In [26]:

```
h('12345678*')
```

Out[26]:

5

(6) 局部择优算法



In [27]:

```
def Local(seq):
    open = queue.LifoQueue()
    compare = queue.PriorityQueue()
    open.put((seq,0,0))
    seqs = []
    path = []
    pos = []
    i = 0
    last = 0
    result = 'No Solution'
    if judgement(seq):
        while (not open.empty()) and (i <= 10):
            openthing = open.get()
            process = openthing[0]
            deep = openthing[1]
            pos.append(openthing[2])
            seqs.append(process)
            i = i + 1
            j = 0
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                j = j + 1
                f =  deep + 1 + h(newseq)
                compare.put((-f,newseq,deep+1,i))
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                j = j + 1
                f =  deep + 1 + h(newseq)
                compare.put((-f,newseq,deep+1,i))
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                j = j + 1
                f =  deep + 1 + h(newseq)
                compare.put((-f,newseq,deep+1,i))
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                j = j + 1
                f =  deep + 1 + h(newseq)
                compare.put((-f,newseq,deep+1,i))
            for k in range(j):
                comparething = compare.get()
                open.put((comparething[1],comparething[2],comparething[3]))
        while last != 0:
            path.append(seqs[last])
            last = pos[last]-1
        path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [28]:

```
result,seqs,pos,path = Local(seq)
```

In [29]:

```
result
```

Out[29]:

```
'1238*4765'
```

In [30]:

```
seqs
```

Out[30]:

```
['2831*4765',
 '283*14765',
 '*83214765',
 '8*3214765',
 '8132*4765',
 '813*24765',
 '*13824765',
 '1*3824765',
 '1238*4765']
```

In [31]:

```
pos
```

Out[31]:

```
[0, 1, 2, 3, 4, 5, 6, 7, 8]
```

In [32]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2	8	3
	1	4
7	6	5

	8	3
2	1	4
7	6	5

8		3
2	1	4
7	6	5

8	1	3
2		4
7	6	5

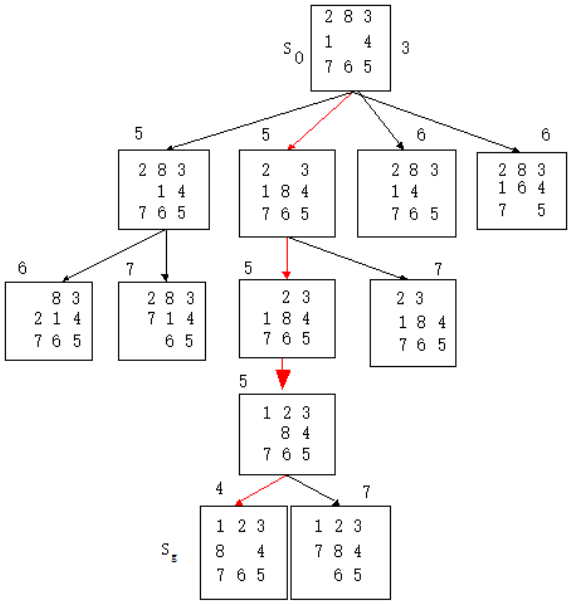
8	1	3
	2	4
7	6	5

	1	3
8	2	4
7	6	5

1		3
8	2	4
7	6	5

1	2	3
8		4
7	6	5

（7）全局择优算法



In [33]:

```
def Global(seq):
    open = queue.PriorityQueue()
    open.put((0,seq,0,0))
    seqs = []
    path = []
    pos = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while not open.empty():
            openthing = open.get()
            process = openthing[1]
            deep = openthing[2]
            seqs.append(process)
            pos.append(openthing[3])
            i = i + 1
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                f =  deep + 1 + h(newseq)
                open.put((f,newseq,deep+1,i))
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                f =  deep + 1 + h(newseq)
                open.put((f,newseq,deep+1,i))
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                f =  deep + 1 + h(newseq)
                open.put((f,newseq,deep+1,i))
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                f =  deep + 1 + h(newseq)
                open.put((f,newseq,deep+1,i))
        while last != 0:
            path.append(seqs[last])
            last = pos[last]-1
        path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [34]:

```
result,seqs,pos,path = Global(seq)
```

In [35]:

```
result
```

Out[35]:

'1238*4765'

In [36]:

```
seqs
```

Out[36]:

['2831*4765', '2*3184765', '*23184765', '123*84765', '1238*4765']

In [37]:

```
pos
```

Out[37]:

[0, 1, 2, 3, 4]

In [38]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

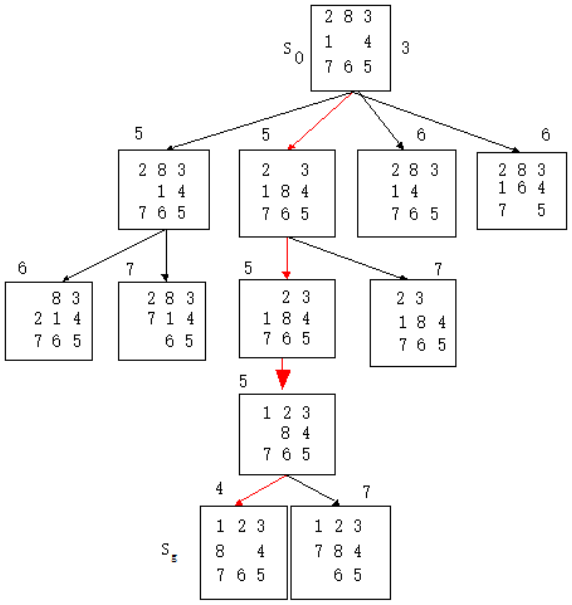
1	2	3
8		4
7	6	5

(8) A*算法

事实上A*算法与上述全局择优算法类似，但是需要证明$g(x)$、$h(x)$满足以下性质：

- 把OPEN表中的节点按估价函数 $f(x) = g(x) + h(x)$ 的值从小到大排序
- $g(x)$是从初始节点S0到节点x的路径的代价， $g(x)$是对$g^*(x)$的估计，$g(x) \geq g^*(x)$。
- $h(x)$是$h^*(x)$的下界，即对所有的x均有：$h(x) \leq h^*(x)$
- 其中，$g^*(x)$是从初始节点S0到节点x的最小代价；$h^*(x)$是从节点x到目标节点的最小代价。

$g(x)$ 是深度 $d(x)$，显然满足条件，并且 $h(x)$ 代表节点 x 的格局与目标节点格局不相同的牌数，启发式信息肯定比 $h^*(x)$ 小，所以符合条件，为 A^* 算法



设置第二种启发式函数 $c(x)$,代表当前节点与目标节点位置不同的节点的距离之和(曼哈顿距离) ,将牌离"家"距离之总和。

In [39]:

```
def c(seq):
    distance = 0
    seq1 = '1238*4765'
    for i in range(len(seq)):
        if seq[i] != seq1[i]:
            for j in range(len(seq1)):
                if seq[i] == seq1[j]:
                    x1 = i%3
                    x2 = j%3
                    y1 = i//3
                    y2 = j//3
                    distance = distance + abs(x1-x2) + abs(y1-y2)  #曼哈顿距离
        else:
            pass
    return distance
```

In [40]:

```
c('12*834765')
```

Out[40]:

4

In [41]:

```
def Astar1(seq):
    open = queue.PriorityQueue()
    open.put((0,seq,0,0))
    seqs = []
    path = []
    pos = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while not open.empty():
            openthing = open.get()
            process = openthing[1]
            deep = openthing[2]
            seqs.append(process)
            pos.append(openthing[3])
            i = i + 1
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                f =  deep + 1 + c(newseq)
                open.put((f,newseq,deep+1,i))
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                f =  deep + 1 + c(newseq)
                open.put((f,newseq,deep+1,i))
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                f =  deep + 1 + c(newseq)
                open.put((f,newseq,deep+1,i))
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                f =  deep + 1 + c(newseq)
                open.put((f,newseq,deep+1,i))
        while last != 0:
            path.append(seqs[last])
            last = pos[last]-1
        path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [42]:

```
result,seqs,pos,path = Astar1(seq)
```

In [43]:

```
result
```

Out[43]:

'1238*4765'

In [44]:

```
seqs
```

Out[44]:

['2831*4765', '2*3184765', '*23184765', '123*84765', '1238*4765']

In [45]:

```
pos
```

Out[45]:

[0, 1, 2, 3, 4]

In [46]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

设置第三种启发式函数 $k(x)$,代表两种函数的加权和

In [47]:

```
def k(seq,a):
    distance = a*h(seq) + (1-a)*c(seq)
    return distance
```

In [48]:

```
k('12345678*',0.5)
```

Out[48]:

7.5

In [49]:

```
def Astar2(seq,a):
    open = queue.PriorityQueue()
    open.put((0,seq,0,0))
    seqs = []
    path = []
    pos = []
    i = 0
    result = 'No Solution'
    if judgement(seq):
        while not open.empty():
            openthing = open.get()
            process = openthing[1]
            deep = openthing[2]
            seqs.append(process)
            pos.append(openthing[3])
            i = i + 1
            if process == '1238*4765':
                result = process
                path.append(result)
                last = pos[len(seqs)-1] - 1
                break
            #向左扩展
            newseq,judge = move(process,'left')
            if not (judge or newseq in seqs):
                f =  deep + 1 + k(newseq,a)
                open.put((f,newseq,deep+1,i))
            #向上扩展
            newseq,judge = move(process,'up')
            if not (judge or newseq in seqs):
                f =  deep + 1 + k(newseq,a)
                open.put((f,newseq,deep+1,i))
            #向右扩展
            newseq,judge = move(process,'right')
            if not (judge or newseq in seqs):
                f =  deep + 1 + k(newseq,a)
                open.put((f,newseq,deep+1,i))
            #向下扩展
            newseq,judge = move(process,'down')
            if not (judge or newseq in seqs):
                f =  deep + 1 + k(newseq,a)
                open.put((f,newseq,deep+1,i))
        while last != 0:
            path.append(seqs[last])
            last = pos[last]-1
        path.append(seqs[last])
    else:
        pass
    return result,seqs,pos,path
```

In [50]:

```
result,seqs,pos,path = Astar2(seq,0.5)
```

In [51]:

```
result
```

Out[51]:

```
'1238*4765'
```

In [52]:

```
seqs
```

Out[52]:

```
['2831*4765', '2*3184765', '*23184765', '123*84765', '1238*4765']
```

In [53]:

```
pos
```

Out[53]:

```
[0, 1, 2, 3, 4]
```

In [54]:

```
for i in range(len(path)):
    show(path[len(path)-i-1])
```

2	8	3
1		4
7	6	5

2		3
1	8	4
7	6	5

	2	3
1	8	4
7	6	5

1	2	3
	8	4
7	6	5

1	2	3
8		4
7	6	5

Step3: 进行启发式对比测试

In [55]:

```
import random
import time
def randomseq():
    seq = '12345678*'
    seq = list(seq)
    random.shuffle(seq)
    seq = ''.join(seq)
    return seq
```

In [63]:

```
def test(iteration):
    print('-------------------------------------------------------')
    for i in range(iteration):
        seq = randomseq()
        print(f'第{i+1}种随机初始序列:{seq}')
        print('-------------------------------------------------------')
        '''
        begin = time.time()
        result0, seqs0, pos0, path0 = BFS(seq)
        last = time.time()
        time0 = last - begin
        print(f'广度优先算法的Close表大小:{len(seqs0)}, 路径长度:{len(path0)}, 运行时间:' + '%.3fs'%time0)
        '''
        begin = time.time()
        result1,seqs1,pos1,path1 = Global1(seq)
        last = time.time()
        time1 = last - begin
        print(f'启发式函数1的Close表大小:{len(seqs1)}, 路径长度:{len(path1)}, 运行时间:' + '%.3fs'%time1)
        begin = time.time()
        result2,seqs2,pos2,path2 = Astar1(seq)
        last = time.time()
        time2 = last - begin
        print(f'启发式函数2的Close表大小:{len(seqs2)}, 路径长度:{len(path2)}, 运行时间:' + '%.3fs'%time2)
        begin = time.time()
        result3,seqs3,pos3,path3 = Astar2(seq,0.5)
        last = time.time()
        time3 = last - begin
        print(f'启发式函数3的Close表大小:{len(seqs3)}, 路径长度:{len(path3)}, 运行时间:' + '%.3fs'%time3)
        print('-------------------------------------------------------')
test(1)
```

```
-------------------------------------------------------
第1种随机初始序列:21758364*
-------------------------------------------------------
启发式函数1的Close表大小:6175, 路径长度:23, 运行时间:0.655s
启发式函数2的Close表大小:1791, 路径长度:25, 运行时间:0.078s
启发式函数3的Close表大小:1800, 路径长度:23, 运行时间:0.085s
-------------------------------------------------------
```