《人工智能导论》

实验一：启发式算法

学　号 22920212204396

姓　名 黄子安

2024 年 3 月 1 日

实验一：启发式搜索

22920212204396 黄子安

1、实验题目

为了方便检测代码的正确性，采用 AcWing 作为自动测评，对应的格式按照该网站的进行

1.1 题目描述

在一个 3×3 的网格中，1~8 这 8 个数字和一个 x 恰好不重不漏地分布在这 3×3 的网格中。例如：

1	2	3
X	4	6
7	5	8

在游戏过程中，可以把 x 与其上、下、左、右四个方向之一的数字交换（如果存在）。我们的目的是通过交换，使得网格变为如下排列（称为正确排列）：

1	2	3
4	5	6
7	8	X

例如，示例中图形就可以通过让 x 先后与右、下、右三个方向的数字交换成功得到正确排列。交换过程如下：

```
1 2 3   1 2 3   1 2 3   1 2 3
x 4 6   4 x 6   4 5 6   4 5 6
7 5 8   7 5 8   7 x 8   7 8 x
```

把 x 与上下左右方向数字交换的行动记录为 u、d、l、r。现在，给你一个初始网格，请你通过**最少的移动次数**，得到正确排列。

1.2 输入格式

输入占一行，将 3×3 的初始网格描绘出来。例如，如果初始网格如下所示：

1	2	3
X	4	6
7	5	8

则输入为：`1 2 3 x 4 6 7 5 8`

1.3 输出格式

输出占一行，包含一个字符串，表示得到正确排列的完整行动记录。如果答案不唯一，输出任意一种合法方案即可。如果不存在解决方案，则输出`unsolvable`。

2、实验思路

使用启发式搜索进行求解，对于每一个当前的状态计算对应的得分，之后根据这个得分的值作为优先级进行宽度优先搜索，从而减小搜索空间的大小，此外对局面需要进行保存，避免无用的重复搜索

这里的得分函数设置成**当前状态和目标状态对应相同字符的曼哈顿距离加上移动的步数**

```
Procedure breadth_first_search
Begin
    Open:=[start];closed:=[ ];                    {*初始化*}
    While   open ≠ [ ] do
    Begin
        从 open 表中删除第一个状态,称之为 n;
        将 n 放入 closed 表中;
        If   n=目的状态  Then Return(success);
                生成 n 的所有子状态;
        从 n 的子状态中删除已在 open 或 closed 表中出现的状态;
                                                  {*避免循环搜索*}
        将 n 的其余子状态,由不同的算法按不同的顺序加入到 open 表;
    End;
End.
```

3、代码实现

先定义一个曼哈顿距离计算函数，返回当前状态中每一个字符和目标状态该字符所在位置的曼哈顿距离

```
int score(string s){
    int tol=0;
    for(int i=0;i<9;++i){
        int t=target.find(s[i]);
        tol+=abs(i/3-t/3)+abs(i%3-t%3);
    }
    return tol;
}
```

之后实现启发式算法，该算法的核心逻辑就是在 bfs 的基础上引入得分，该得分为**曼哈顿距离加上对应移动步数**，根据这个得分作为搜索的优先级来优先选择离终点更近的情况进行搜索

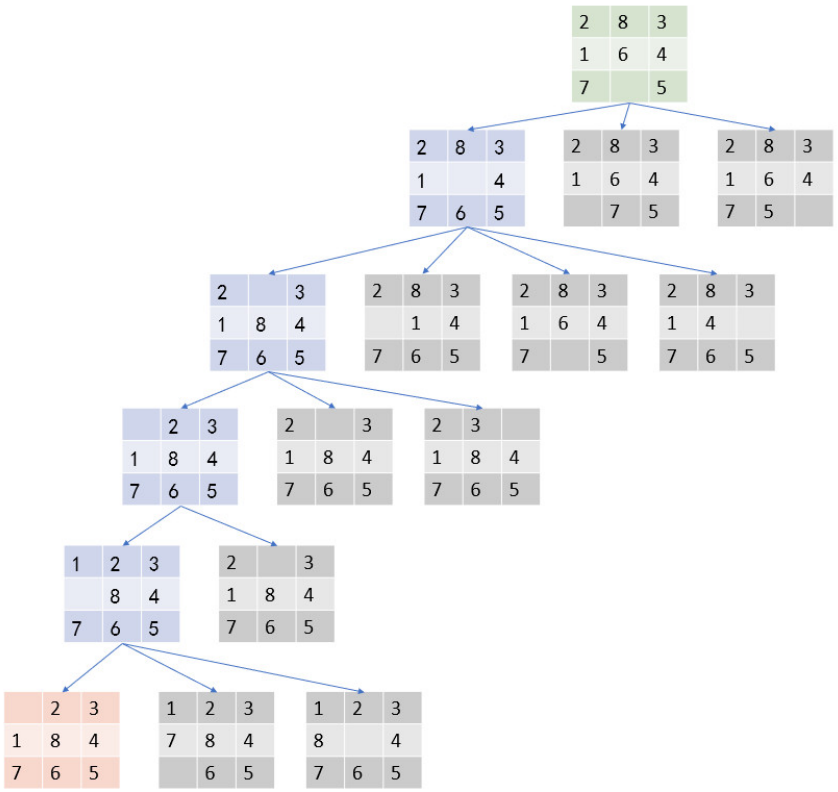
```
string bfs(){
    unordered_map<string,int> d;
    //记录起始状态到该状态需要的移动次数，同时用来记录该状态是否出现过
    priority_queue<PIS,vector<PIS>,greater<PIS>> heap;
    //优先队列，以得分为优先级进行排序
    unordered_map<string,pair<string,char>> last;
    //last保存当前状态的前一个状态和转移方式
    heap.push({score(start),start});
    char oper[]="udlr";
    int dx[4]={-1,1,0,0},dy[4]={0,0,-1,1};

    while(heap.size()){
        auto t=heap.top();
        heap.pop();
        string state=t.second;
        if(t.second==target) break;

        int x,y;
        //寻找x的位置
        for(int i=0;i<9;i++){
            if(state[i]=='x'){
                x=i/3,y=i%3;
                break;
            }
        }
        string init=state;
        //将对应的可行解加入到队列中
        for(int i=0;i<4;i++){
            int xx=x+dx[i],yy=y+dy[i];
            if(xx<0||xx>=3||yy<0||yy>=3) continue;
            swap(state[xx*3+yy],state[x*3+y]);
            if(!d.count(state)||d[state]>d[init]+1){
                d[state]=d[init]+1;
                heap.push({score(state)+d[state],state});
                last[state]={init,oper[i]};
            }
            state=init;
        }
    }
    if(heap.size()==0) return "unsolvable";
    //如果队列为空则说明无解
    string ans;
    while(target!=start){
        ans+=last[target].second;
        target=last[target].first;
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
```

完整代码如下所示：

```cpp
#include <bits/stdc++.h>
using namespace std;
typedef pair<int,string> PIS;
string start,target="12345678x";
int score(string s){
    int tol=0;
    for(int i=0;i<9;++i){
            int t=target.find(s[i]);
            tol+=abs(i/3-t/3)+abs(i%3-t%3);
    }
    return tol;
}
string bfs(){
    unordered_map<string,int> d;
    priority_queue<PIS,vector<PIS>,greater<PIS>> heap;
    unordered_map<string,pair<string,char>> last;

    heap.push({score(start),start});
    char oper[]="udlr";
    int dx[4]={-1,1,0,0},dy[4]={0,0,-1,1};

    while(heap.size()) {
        auto t=heap.top();
        heap.pop();
        string state=t.second;
        if(t.second==target) break;

        int x,y;
        for(int i=0;i<9;i++){
            if(state[i]=='x'){
                x=i/3,y=i%3;
                break;
            }
        }
        string init=state;
        for(int i=0;i<4;i++){
            int xx=x+dx[i],yy=y+dy[i];
            if(xx<0||xx>=3||yy<0||yy>=3) continue;
            swap(state[xx*3+yy],state[x*3+y]);
            if(!d.count(state)||d[state]>d[init]+1){
                d[state]=d[init]+1;
                heap.push({score(state)+d[state],state});
                last[state]={init,oper[i]};
            }
            state=init;
        }
    }
    if(heap.size()==0) return "unsolvable";
    string ans;
    while(target!=start){
        ans+=last[target].second;
        target=last[target].first;
    }
    reverse(ans.begin(),ans.end());
    return ans;
}
int main()
{
    for(int i=0;i<9;++i){
        string s;cin>>s;
        start+=s;
    }
    cout<<bfs();
    return 0;
}
```

4、实验结果

对题中所给的情况进行求解，将 target 设置成 `1238x4765`，运行后可以发现需要 5 步获得最终的答案，对应的求解步骤为 `uuldr`，绘制对应的节点状态图如下所示，可以看到搜索树的深度和搜索的节点数量很少，高效进行问题的求解：



经过 Acwing 大量数据测试通过截图：

```
    }
    return ans;
}
int main () {
    for (int i = 0;i < 9;i++) {
        cin >> ch;
        start += ch;
        if (ch != 'x') s += ch;
    }
    int cnt = 0;
    for (int i = 0;i < 8;i++) {
        for (int j = i;j < 8;j++) {
            if (s[i] > s[j]) cnt++;
        }
    }
    if (cnt & 1) puts ("unsolvable");
    else cout << Astar () << endl;
    return 0;
}
```

数据有点弱吗？可以申请加强数据　　调试代码　提交答案

代码提交状态：Accepted