**实 验 报 告**

**学 院：** 计算机科学与技术学院

**班 级：** 计算机初阳211

**学 号：** 202136900115

**姓 名：** 林 宸

**课 程：** 人工智能基础

**指导教师：** 周昌军

**完成时间：** 2023 **年** 12 **月** 14 **日**

**浙 江 师 范 大 学 制**

**A\*算法求解八数码问题**

**问题分析**

实验要求用A\*算法来实现将一个任意排列变换成一个指定排列。

解的存在性：A\*算法的本质是启发式搜索，可以以一个较快的速度找到解。但是需要注意的是，并不是任意一个排列都是有解的。我们不妨用0来代替原本状态里的空格，那么可以证明，如果两个排列的逆序数不同，该问题就是无解的，否则有解。

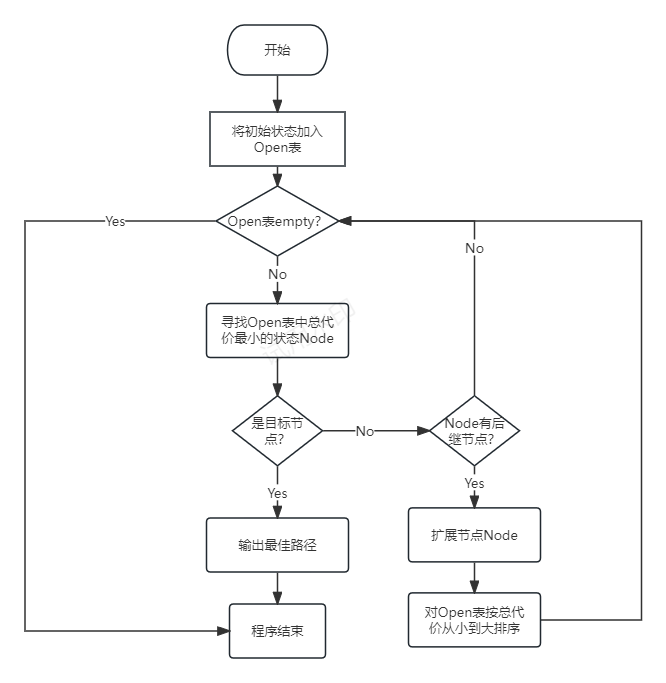
搜索策略：为了找到解决方案，我们需要一种有效的搜索策略。A\*算法是一种启发式搜索算法，它使用一个估计值来指导搜索的方向。在这个问题中，我们可以使用启发式函数来估计两个状态之间的距离，从而，每次转移到代价最小的点去，可以更快到达终点状态。

评估函数：可考虑每个状态n的估价函数值为两个分量：从起始状态到状态n的实际代价g(n)以及从状态n到达目标节点的估价代价法f(n)，这里我们将g(n)定义为走到状态n的最小步数，这是一个很自然的定义，因为它就代表了到达该状态的代价。定义f(n)为状态n与目标状态的曼哈顿距离，也就是对应数字的曼哈顿距离之和。

边界条件：在搜索过程中，我们需要设置一些边界条件来避免搜索不必要的状态。例如，如果一个状态已经证明不是解决方案的一部分，我们可以将其标记为不可达，并从搜索中排除。

**算法设计**

A\*算法可以通过bfs来实现。以下是程序的大致流程图



为了方便，我们可以使用一个优先队列来实现Open表。这需要我们自定义排序方式，那么就按照之前所分析的，取f(n)+g(n)的尽可能小的状态放在队头即可

另外一开始就可以把是否有解给判断出来

**代码**

#include<bits/stdc++.h>

#include <cstdio>

#include <queue>

using namespace std;

#define endl '\n'

#define ll long long

#define low(x) x&(-x)

#define pll pair<ll,ll>

#define mk make\_pair

string s1,s2;

ll base=131;

ll mod=998244353;

map<int,vector<ll>> mp;

map<string,string> pre;

void init()

{

mp[0]={1,3};

mp[1]={0,2,4};

mp[2]={1,5};

mp[3]={0,4,6};

mp[4]={1,3,5,7};

mp[5]={2,4,8};

mp[6]={3,7};

mp[7]={4,6,8};

mp[8]={5,7};

}

ll hash(string s)

{

ll ans=0;

for(auto a:s)

{

ans=(ans\*base%mod+(int)(a-'0'))%mod;

}

return ans;

}

map<string,int> vis;

map<string,int> stp;

int gt\_rod(string s)

{

ll cnt=0;

for(int i=0;i<s.size();++i)

{

for(int j=i+1;j<s.size();++j)

{

if(s[j]=='0') continue;

if(s[i]>s[j]) cnt++;

}

}

return cnt%2;

}

int gt\_dif(string s1,string s2)

{

ll cnt=0;

for(int i=0;i<s1.size();++i)

{

if(s1[i]=='0') continue;

if(s1[i]!=s2[i]) cnt++;

}

return cnt;

}

struct ty

{

string s;

int step;

int dif;

bool operator <(const ty & b) const

{

return dif+step>b.dif+b.step;

}

};

priority\_queue<ty> q;

int pos=0;

int ma=1e9;

string Pre;

void Solve()

{

while(!q.empty())

{

auto [a,step,dif]=q.top();

if(!vis[a]) vis[a]=1;

if(gt\_dif(a,s2)==0)

{

ma=min(ma,step);

return;

}

q.pop();

pos=0;

for(int i=0;i<a.size();++i)

{

if(a[i]=='0')

{

pos=i;break;

}

}

Pre=a;

for(auto k:mp[pos])

{

swap(a[pos],a[k]);

if(!vis[a])

{

if(stp.count(a)==0)

{

stp[a]=1e9;

}

if(step+1<stp[a])

{

pre[a]=Pre;

stp[a]=step+1;

q.push({a,stp[a],gt\_dif(a,s2)});

}

}

swap(a[pos],a[k]);

}

}

}

void solve()

{

init();

cin>>s1>>s2;

if(gt\_rod(s1)!=gt\_rod(s2))

{

cout<<"NO"<<endl;

return;

}

ty ini={s1,0,gt\_dif(s1,s2)};

q.push(ini);

Solve();

cout<<ma<<endl;

vector<string> ans;

while(1)

{

ans.push\_back(s2);

if(pre.count(s2)==0) break;

s2=pre[s2];

}

reverse(ans.begin(),ans.end());

for(auto an:ans)

{

for(int i=0;i<3;++i)

{

cout<<an[i\*3]<<' '<<an[i\*3+1]<<' '<<an[i\*3+2]<<endl;

}

cout<<endl;

}

}

int main()

{

ios::sync\_with\_stdio();cin.tie(0);cout.tie(0);

// ll t;cin>>t;while(t--)

solve();

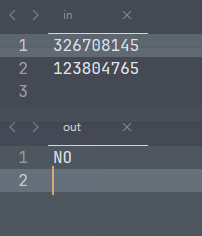
return 0;

}

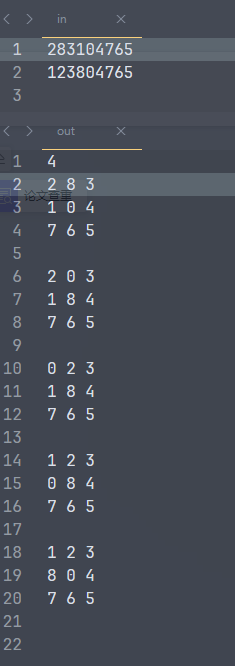
**运算结果**

输入是一个初始状态以及终止状态。如果状态有解，输出最小步骤以及每一步的变化。否则输出NO表示无解

对于给定的样例，输出结果如下。这是因为它们的逆序对奇偶性不同



我们换一个样例



可以发现这个确实有解，并且只需要四步即可。可以证明4步确实是最优解

**实验总结**

通过本次实验，我对A\*算法以及广度优先搜索有了更深的理解。A\*算法的本质是在搜索的基础上加以优化，通过引入代价函数的概念来实现在搜索过程中动态调整搜索策略，从而能够更快地找到最优解。