【算法分析】

作为算法设计，我们将问题的要求改选为300步的操作序列之内的穷举，将操作序列的全部列举构成一个树状结构（用X表示初始状态）如图10.27所示。

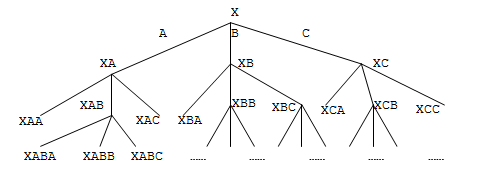


图10.27

如果按题目给定的300步的范围计算，实际可能产生的结点数目将是十分惊人的，所以要将一些不合题意的结点剪枝，经过进一步分析，会发现有些操作序列是循环的，例如XAA＝X，XBBBB＝X，XCCCC＝X，…，所以我们需要记录下不同的结点序列，一旦遇到相同的序列则删除。

例如可以用如图10.28所示的结构来记录每一步操作。

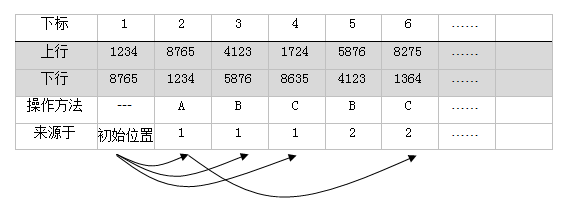


图10.28

这道题跟八数码问题类似，也就是用康托展开式压缩状态，保证每种状态不会重复出现，然后进行宽搜即可。但是由于它说需要输出字典序最小的操作序列，所以最好用宽搜而不是双向广搜。

参考代码如下所示。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99  100  101  102  103  104  105  106  107  108  109  110  111  112  113  114  115  116  117  118  119  120  121  122  123  124  125  126  127  128  129  130  131  132  133  134  135  136  137  138 | //魔板问题 — 宽搜  #include <bits/stdc++.h>  using namespace std;  bool have[45000]= {0}; //存储某状态是否存在  char did[45000]= {0}; //存储某一步的操作  int bfs[45000]= {0},last[45000]= {0},cost[45000]= {0};  //分别存储当前状态，上一个状态的序号，总共进行了多少次操作  int s[8]= {1,2,3,4,8,7,6,5},ans=0,enter=0,now=0,len=0,f=0,temp=0;  /\* s数组表示魔板的状态，ans表示目标状态，enter是为了保证输出  每60个一行，now是目前搜索到的序号，len是目前总共状态，f表示  答案的序号 \*/  int cantor() //用康托展开式压缩状态  {  int a=0,i=0,j=0,b=0;  for(i=7; i>=1; --i)  {  b=s[i]-1;  for(j=7; j>i; --j)  if(s[j]<s[i])  --b;  a+=b;  a\*=i;  }  return a;  }  int change() //把状态压缩成8位的数字  {  int a=0,i=0;  for(i=0; i<=7; ++i)  a=a\*10+s[i];  return a;  }  void turn(int num) //把数字解压  {  int i=0;  for(i=7; i>=0; --i)  {  s[i]=num%10;  num/=10;  }  return;  }  void go(char tmp)  {  ++len;  bfs[len]=change();  temp=cantor();  if(have[temp])  --len;  else  {  have[temp]=1;  cost[len]=cost[now]+1;  last[len]=now;  did[len]=tmp;  if(bfs[len]==ans)  f=len;  }  }  void go1() //操作A  {  int i=0;  for(i=0; i<4; ++i) //进行操作  {  temp=s[i],s[i]=s[i+4],s[i+4]=temp;  }  go('A'); //尝试添加  for(i=0; i<4; ++i) //还原  {  temp=s[i],s[i]=s[i+4],s[i+4]=temp;  }  }  void go2() //操作B  {  temp=s[3],s[3]=s[2],s[2]=s[1],s[1]=s[0],s[0]=temp;//进行操作  temp=s[7],s[7]=s[6],s[6]=s[5],s[5]=s[4],s[4]=temp;//尝试添加  go('B');  temp=s[0],s[0]=s[1],s[1]=s[2],s[2]=s[3],s[3]=temp;//还原  temp=s[4],s[4]=s[5],s[5]=s[6],s[6]=s[7],s[7]=temp;  }  void go3() //操作c  {  temp=s[1],s[1]=s[5],s[5]=s[6],s[6]=s[2],s[2]=temp;//进行操作  go('C'); //尝试添加  temp=s[2],s[2]=s[6],s[6]=s[5],s[5]=s[1],s[1]=temp;//还原  }    void out(int num) //输出  {  if(num!=1)  {  out(last[num]);  cout<<did[num];  ++enter;  if(enter%60==0)  cout<<"\n"; //保证每行最多60个字符  }  }  int main()  {  int i=0,j=0;  have[cantor()]=1; //确定初始状态  bfs[1]=12348765;  now=1;  len=1;  cin>>s[0]>>s[1]>>s[2]>>s[3]>>s[7]>>s[6]>>s[5]>>s[4];//保持输入顺序的正确  ans=change();  if(ans==bfs[1])  f=1;  while(f==0 && now<=len) //宽度优先搜索  {  turn(bfs[now]);  go1();  if(f==0)  go2();  if(f==0)go3();  ++now;  }  if(f==0)  cout<<-1<<"\n";  else  {  cout<<cost[f]<<"\n";  out(f);  if(enter%60!=0)  cout<<"\n"; //保证最后一行有回车  }  return 0;  } |

更精简代码：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69 | //魔板问题-STL  #include<bits/stdc++.h>  using namespace std;  string goal;  map<string,string>m;  queue<string>q;  void A(string x)  {  string t=x;  for(int i=0; i<4; i++)  swap(x[i],x[7-i]);  if(!m.count(x)) //没有出现过  {  q.push(x);  m[x]=m[t]+'A';  }  }  void B(string x)  {  string t=x;  x[0]=t[3],x[1]=t[0],x[2]=t[1],x[3]=t[2],x[4]=t[5],x[5]=t[6],x[6]=t[7],x[7]=t[4];  if(!m.count(x))  {  q.push(x);  m[x]=m[t]+'B';  }  }  void C(string x)  {  string t=x;  x[1]=t[6],x[2]=t[1],x[5]=t[2],x[6]=t[5];  if(!m.count(x))  {  q.push(x);  m[x]=m[t]+'C';  }  }  int main()  {  freopen("Magic.in","r",stdin);  freopen("Magic.out","w",stdout);  char c;  for(int i=0; i<8; i++)  {  cin>>c;  goal+=c;  getchar();  }  q.push("12345678");  m["12345678"]="";  while(!q.empty())  {  A(q.front());  B(q.front());  C(q.front());  if(m.count(goal)!=0)  break;  q.pop();  }  cout<<m[goal].size()<<endl<<m[goal]<<endl;  return 0;  } |