程序设计实习

郭炜　微博 http://weibo.com/guoweiofpku
http://blog.sina.com.cn/u/3266490431
刘家瑛 微博 http://weibo.com/pkuliujiaying

广度优先搜索

八数码问题

八数码问题 ，单向广搜，最简单做法，POJ 891MS HDU TLE

```
#include <iostream>
#include <bitset>
#include <cstring>
using namespace std;
int goalStatus;  //目标状态
bitset<362880> Flags; //节点是否扩展的标记
const int MAXS = 400000;
char result[MAXS];   //结果
struct Node {
        int status; //状态，即排列的编号
        int father; //父节点指针
        char move;   //父节点到本节点的移动方式 u/d/r/l
        Node(int s,int f,char m):status(s), father(f),move(m) { }
        Node() { }
};
Node myQueue[MAXS];  //状态队列，状态总数362880
int qHead;    int qTail; //队头指针和队尾指针
char sz4Moves[] = "udrl"; //四种动作
unsigned int factorial[21];  //存放0-20的阶乘。21的阶乘unsigned放不下了
```

```
unsigned int GetPermutationNumForInt(int * perInt,int len)
//perInt里放着整数 0 到 len-1 的一个排列，求它是第几个排列
//len不能超过21
{
        unsigned int num = 0;
        bool used[21];
        memset(used,0,sizeof(bool)*len);
        for( int i = 0;i < len; ++ i ) {
                unsigned int n = 0;
                for( int j = 0; j < perInt[i]; ++ j) {
                        if(! used[j] )
                                ++n;
                }
                num += n * factorial[len-i-1];
                used[perInt[i]] = true;
        }
        return num;
}
```

```
template< class T>
unsigned int GetPermutationNum( T s1, T s2,int len)
//给定排列，求序号 。[s1,s1+len)里面放着第0号排列，[s2,s2+len)是要求序号的排列
//两者必须一样长，len不能超过21
//排列的每个元素都不一样。返回排列的编号
{
        int perInt[21]; //要转换成 [0,len-1] 的整数的排列
        for( int i = 0;i < len; ++i )
                for( int j = 0; j < len; ++j )  {
                        if( * ( s2 + i )  == * (s1+j)) {
                                perInt[i] = j;
                                break;
                        }
                }
        unsigned int num = GetPermutationNumForInt(perInt,len);
        return num;
}
```

```
template <class T>
void GenPermutationByNum(T   s1, T  s2,int len, unsigned int No)
//根据排列编号，生成排列 len不能超过21
{ //[s1,s1+len) 里面放着第0号 permutation,，排列的每个元素都不一样
        int perInt[21]; //要转换成 [0,len-1] 的整数的排列
        bool used[21];
        memset(used,0,sizeof(bool)*len);
        for(int i = 0;i < len; ++ i ) {
                unsigned int tmp;  int n = 0;int j;
                for( j = 0; j < len; ++j ) {
                        if( !used[j] ) {
                                if( factorial[len - i - 1] >= No+1)  break;
                                else No -= factorial[len - i - 1];
                        }
                }
                perInt[i] = j;
                used[j] = true;
        }
```

```
        for( int i = 0;i < len; ++i )
                * ( s2 + i ) = * ( s1 + perInt[i]);

}
int StrStatusToIntStatus( const char * strStatus)
{//字符串形式的状态，转换为整数形式的状态(排列序号）
        return GetPermutationNum( "012345678",strStatus,9);
}
void IntStatusToStrStatus( int n, char * strStatus)
{//整数形式的状态(排列序号），转换为字符串形式的状态
        GenPermutationByNum((char*)"012345678",strStatus,9,n);
}
```

```
int NewStatus( int nStatus, char cMove) {
//求从nStatus经过 cMove 移动后得到的新状态。若移动不可行则返回-1
        char szTmp[20];    int nZeroPos;
        IntStatusToStrStatus(nStatus,szTmp);
        for( int i = 0;i < 9; ++ i )
                if( szTmp[i] == '0' ) {
                        nZeroPos = i;
                        break;
                } //返回空格的位置
        switch( cMove) {
                case 'u': if( nZeroPos - 3 < 0 )  return -1; //空格在第一行
                        else {     szTmp[nZeroPos] = szTmp[nZeroPos - 3];
                                szTmp[nZeroPos - 3] = '0'; }
                        break;
                case 'd': if( nZeroPos + 3 > 8 )  return -1; //空格在第三行
                        else {     szTmp[nZeroPos] = szTmp[nZeroPos + 3];
                                szTmp[nZeroPos + 3] = '0';}
                        break;
```

```
            case 'l':   if( nZeroPos % 3 == 0)  return -1; //空格在第一列
                        else {      szTmp[nZeroPos] = szTmp[nZeroPos -1];
                                    szTmp[nZeroPos -1 ] = '0'; }
                        break;
            case 'r':   if( nZeroPos % 3 == 2)  return -1; //空格在第三列
                        else {      szTmp[nZeroPos] = szTmp[nZeroPos + 1];
                                    szTmp[nZeroPos + 1 ] = '0';             }
                        break;
        }
        return StrStatusToIntStatus(szTmp);
}
```

```
bool Bfs(int nStatus) { //寻找从初始状态nStatus到目标的路径
        int nNewStatus;      Flags.reset(); //清除所有扩展标记
        qHead = 0;           qTail = 1;
        myQueue[qHead] = Node(nStatus,-1,0);
        while ( qHead != qTail) { //队列不为空
                nStatus = myQueue[qHead].status;
                if( nStatus == goalStatus ) //找到目标状态
                        return true;
                for( int i = 0;i < 4;i ++ ) { //尝试4种移动
                        nNewStatus = NewStatus(nStatus,sz4Moves[i]);
                        if( nNewStatus == -1 )   continue; //不可移，试下一种
                        if( Flags[nNewStatus] )  continue; //扩展标记已经存在，则不入队
                        Flags.set(nNewStatus,true); //设上已扩展标记
                        myQueue[qTail++] =
                              Node(nNewStatus,qHead,sz4Moves[i]); //新节点入队列
                }
                qHead ++;
        }
        return false;
}
```

```
int main(){
        factorial[0] = factorial[1] =1;
        for(int i = 2;i < 21; ++i )
                factorial[i] = i * factorial[i-1];
        goalStatus = StrStatusToIntStatus("123456780");
        char szLine[50];  char szLine2[20];
        while( cin.getline(szLine,48)) {
                int i,j;
                //将输入的原始字符串变为数字字符串
                for( i = 0, j = 0; szLine[i]; i ++ ) {
                        if( szLine[i] != ' ' ) {
                                if( szLine[i] == 'x' )  szLine2[j++] = '0';
                                else  szLine2[j++] = szLine[i];
                        }
                }
                szLine2[j] = 0; //字符串形式的初始状态
                int sumGoal = 0;  //从此往后用奇偶性判断是否有解
```

```
for( int i = 0;i < 8; ++i )
        sumGoal += i -1;
int sumOri = 0;
for( int i = 0;i < 9 ; ++i ) {
        if( szLine2[i] == '0')
                continue;
        for( int j = 0; j < i; ++j ) {
                if( szLine2[j] < szLine2[i] && szLine2[j] != '0' )
                        sumOri ++;
        }
}

if( sumOri %2 != sumGoal %2 ) {
        cout << "unsolvable" << endl;
        continue;
}
//上面用奇偶性判断是否有解
```

```
                if( Bfs(StrStatusToIntStatus(szLine2))) {
                        int nMoves = 0;
                        int nPos = qHead;
                        do {  //通过father找到成功的状态序列，输出相应步骤
                                result[nMoves++] = myQueue[nPos].move;
                                nPos = myQueue[nPos].father;
                        } while( nPos); //nPos = 0 说明已经回退到初始状态了
                        for( int i = nMoves -1; i >= 0; i -- )
                                cout << result[i];
                }
                else
                        cout << "unsolvable" << endl;
        }
}
```

时间复杂度，就是状态总数