同济大学人工智能原理课程实验报告

实验题目:八数码问题

一、 实验概述

1. **实验目的**
   练习使用 A*算法搜索，通过最少的步数，将八数码问题从初始状态还原到目标状态。

2. **实验问题描述**
   八数码问题是最早的启发式搜索问题之一，其目标是将棋子水平或者竖直地滑到空格中国，直到棋盘局面和目标状态一致。首先会在 3*3 方格上随机生成八个数字并且互不相同，占据不同的位置，留有一个空格，在此基础上进行移动还原。

3. **实验原理**
   实验主要是运用启发式搜索算法——A*算法进行求解。基本原理就是找到一个启发式函数,通过对到当前状态的实际代价 g(x)和目标状态的估算代价 h(x)比较进行评选,fx+gx 小者优先，那么这样就会一步步引导我们以最少的时间到达目标状态。

4. **实验环境**
   Windows10 系统，Qt Designer，Qt Creator，visual studio 2017.

二、 算法详细设计

本次实验主要用到高效搜索算法——A*算法。具体的实现方法如下：

1. **问题转化：**注意到，八个数字进行移动等价于一个空格进行移动，这样就简化了问题，进而转变成对空格进行操作，使其每次向四个方向移动。

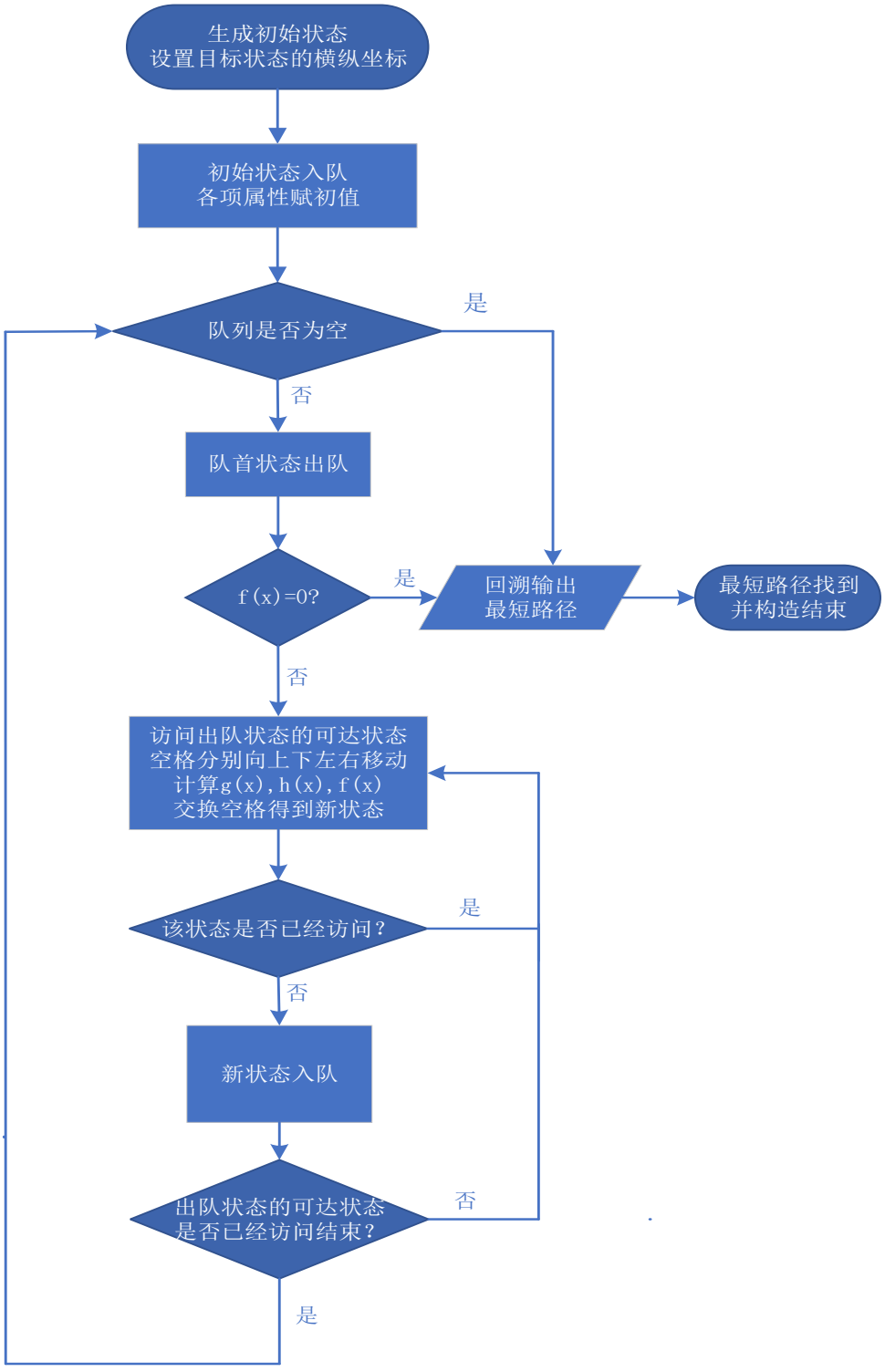
2. **启发式函数探索:**设初始状态 $(initial\_status)$ 到当前状态 $(current\_status)$ 的实际代价函数是 $g(x)$，当前状态到目标状态的估价函数为 $h(x)$，初始状态到目标状态的总代价 $f(x)=g(x)+h(x).$，其中 $h(x)$ 的具体计算，我会用一个哈希表存储目标状态，让每次得到的新状态根据目标状态的横纵坐标进行路径长度计算。

3. **BFS+启发函数+优先队列优化:**减少查询次数,普通的 BFS 会对当前状态的每个状态都进行访问，而辅助以优先队列则可以将查询时间从 $n \to \log n$。具体步骤是：每次拿到一个状态，让空格分别向上下左右移动，记录每个状态的总代价值 $f(x)$，并且对每一个状态都有一个 $step$ 属性，即它是由最初状态经过几步得到，显然当前的 $step$ 总是由当前队首的状态 $step$+1 得来的，除此之外，还要记录空格移动之后状态的前一个状态 (pre_status) 便于构造最短路径，最后把空格向某个方向移动后的新状态压入优先队列 $(priority\_queue)$，它每次都是以总代价函数 $f(x)$ 为关键值进行状态排序(此处将每个状态封装到一个结构体中)并且由队头到队尾从小到大排序，每次弹出键值最小的队头代表的状态，这样每次访问都是当前最优路径，直到队首状态的 $f(x)$=0，即到达目标状态，结束搜索，此时也就找到了从初始状态到目标状态的最少步数(最短路径)，即当前队首的 $step$ 值。

4. **最短路径构造:**上述搜索的方法可知，从初始状态到目标状态的所有遍历过的状态构成了一棵状态树，而且目标状态必定是叶子结点且只有一个，但是不一定每次从队首出来的状态属于最优解中的一个。可以确定的是，搜索树中的每一层必定含有一个属于最优解，而且每层中的解必定是可行解中最小的，这样才能满足最短路径的定义。

   在此，我从目标状态进行回溯，相当于目标状态为根节点，以每个状态节点记录的 (pre_status) 为线索进行递归，直到第一层的初始状态，递归回溯结束，最短路径构造完成。

三、 算法流程图



四、 源程序

```
#include <iostream>
#include<vector>
#include<queue>
#include<cstring>
#include<string>
#include<map>
#include<stack>
using namespace std;

int min_steps = 0;
const int maxn = 1000;
//存放访问过程中的各种状态
struct Block
{
    int x, y, fx, gx, step;
    //x,y分别是block 中数的横纵坐标,fx是启发式函数,
    //gx是起始状态到x状态的实际代价函数，step是初始状态到目标状态的路径长度
    int blk[3][3];//八数码宫格
    string pre;//存放该状态的前一个状态，便于输出最短路径
    //重载函数，便于优先队列以fx为关键值对状态进行排序
    bool operator<(const Block &b) const
    {
        return fx > b.fx;
    }
};
Block goal_blk = { 0, 0, 0, 0, 0, {1, 2, 3, 8, 0, 4, 7, 6, 5} };//目标状态
map<int, Block> blk0; //存目标状态，以及各个数字对应的横纵坐标，便于计算hx
vector<Block> adjv[maxn];//邻接表，构造搜索树，用以回溯构造最优解
stack<Block> path;//存放最优解路径的每个状态

/*重载输出运算符<<,输出每个blk*/
ostream &operator<<(ostream &out, Block b)
{
    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            out << b.blk[i][j] << " ";
        cout << endl;
    }
    return out;
}
```

```
/*计算当前状态到目标状态的曼哈顿距离*/
int hx(int blk8[][3]) //计算x状态到目标状态的估计距离
{
    int h = 0;
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            h += abs(blk0[blk8[i][j]].x - i) + abs(blk0[blk8[i][j]].y - j);
    return int(h);
}

/*
将每个状态转化成字符串，即3行按顺序排列构成字符串
用以标识其访问过及唯一性
*/
string key(int blk8[][3])
{
    string key_value = " ";
    for (int i = 0; i < 3; i++)
        for (int j = 0; j < 3; j++)
            key_value += to_string(blk8[i][j]);
    return key_value;
}

/*设置目标状态的横纵坐标*/
void Setgoalstatus()
{
    blk0[1].x = 0;
    blk0[1].y = 0;

    blk0[2].x = 0;
    blk0[2].y = 1;

    blk0[3].x = 0;
    blk0[3].y = 2;

    blk0[4].x = 1;
    blk0[4].y = 2;

    blk0[5].x = 2;
    blk0[5].y = 2;

    blk0[6].x = 2;
    blk0[6].y = 1;
```

```
    blk0[7].x = 2;
    blk0[7].y = 0;

    blk0[8].x = 1;
    blk0[8].y = 0;

    blk0[0].x = 1;
    blk0[0].y = 1;
}

/*从目标结点进行回溯，以pre为关键线索进行寻找最优解*/
void Traceback(Block b, int step)
{
    path.push(b);
    if (step == -1)
        return;
    for (int i = 0; i < adjv[step].size(); i++)
    {
        Block u = adjv[step][i];
        if (b.pre == key(u.blk))
            Traceback(u, step - 1);
    }
}

/*输出每个状态*/
void print(int blk[][3])
{

    for (int i = 0; i < 3; i++)
    {
        for (int j = 0; j < 3; j++)
            cout << blk[i][j] << " ";
        cout << endl;
    }
    cout << endl;
}
/*A*算法*/
void A_STAR(Block &b0)
{
    int direction[4][2] = { {0, -1}, {0, 1}, {-1, 0}, {1, 0} }; //左右上下移
动
    priority_queue<Block> pq;//优先队列
    map<string, int> MAP;//有前面计算出的状态转化成的字符串来表示是否已经访
```

```
b0.fx = hx(b0.blk) + b0.gx;//计算改状态的总代价函数
    pq.push(b0);
    while (!pq.empty())
    {
        Block now = pq.top();
        adjv[now.step].push_back(now);
        MAP[key(now.blk)] = now.fx;//当前状态的总代价赋给标识数组，用以判断
是否访问过
        pq.pop();
        if (hx(now.blk) == 0)//到了目标状态
        {
            min_steps = now.step;
            goal_blk.pre = adjv[now.step][0].pre;
            cout << now.step << endl;//输出最短路径长度
            return;//退出
        }
        for (int i = 0; i < 4; i++)
        {
            int posx = now.x + direction[i][0];
            int posy = now.y + direction[i][1];
            if (posx < 0 || posx > 2 || posy > 2 || posy < 0)//超出边界
                continue;
            Block b = now;
            b.blk[now.x][now.y] = b.blk[posx][posy];

            b.blk[posx][posy] = 0; //将now.blk的0和其中相邻的一个数进行交换
            b.x = posx;
            b.y = posy; //b中0的横纵坐标
            b.gx++;     //实际代价加一
            b.fx = hx(b.blk) + b.gx;//计算新状态的总代价
            b.step = now.step + 1;//步数加一
            b.pre = key(now.blk);//记录新状态的前一个状态
            if (MAP[key(b.blk)] == 0)
            {
                pq.push(b);//没访问过，入队
            }
        }
    }
}
int main()
{
    Block b;
    Setgoalstatus();
    b.step = 0;
```

```
for (int i = 0; i < 9; i++)
    {
        char c = getchar();
        b.blk[i / 3][i % 3] = c - '0';
        if (c == '0')
        {
            b.x = i / 3;
            b.y = i % 3;
        }
    }
    b.pre = key(b.blk);
    A_STAR(b);
    Traceback(goal_blk, min_steps - 1);
    while (!path.empty())
    {
        Block b = path.top();
        print(b.blk);
        path.pop();
    }//输出最短路径
    return 0;
}
```

注：以上代码只是核心代码，图形界面的代码由于比较繁冗，所以会以附件的形式进行递交。

五、 实验结果及结论

实验结论：

根据八数码的数学原理：在输入除 0 以外的八个数字序列中，起始状态和目标状态的逆序数奇偶性相同，有解；否则——A*算法在满足一定条件下找到的解必然是最优解。

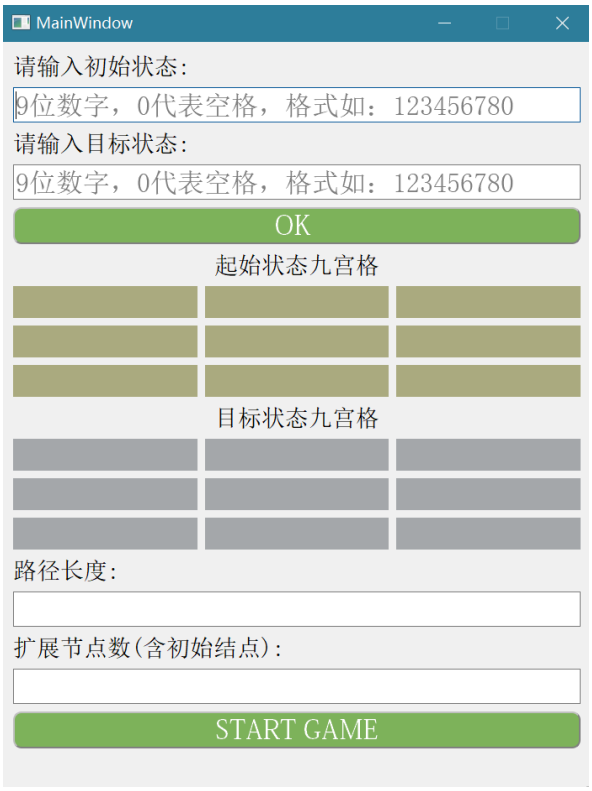
最短路得到最优解条件——A*算法的启发式函数 h 如果小于等于真实值 n 的话，那么算法是能得到最优解的，若 h 大于等于真实值 n，那么就不能保证得到最优解。

也就是——它的限制条件是 f(x) = g(x) + h(x) ，代价函数 g(x) >0 ；h(x) 的值不大于 x 到目标的实际代价 h*(x) 。即定义的 h(x) 是可纳的，是乐观的。

实验结果：

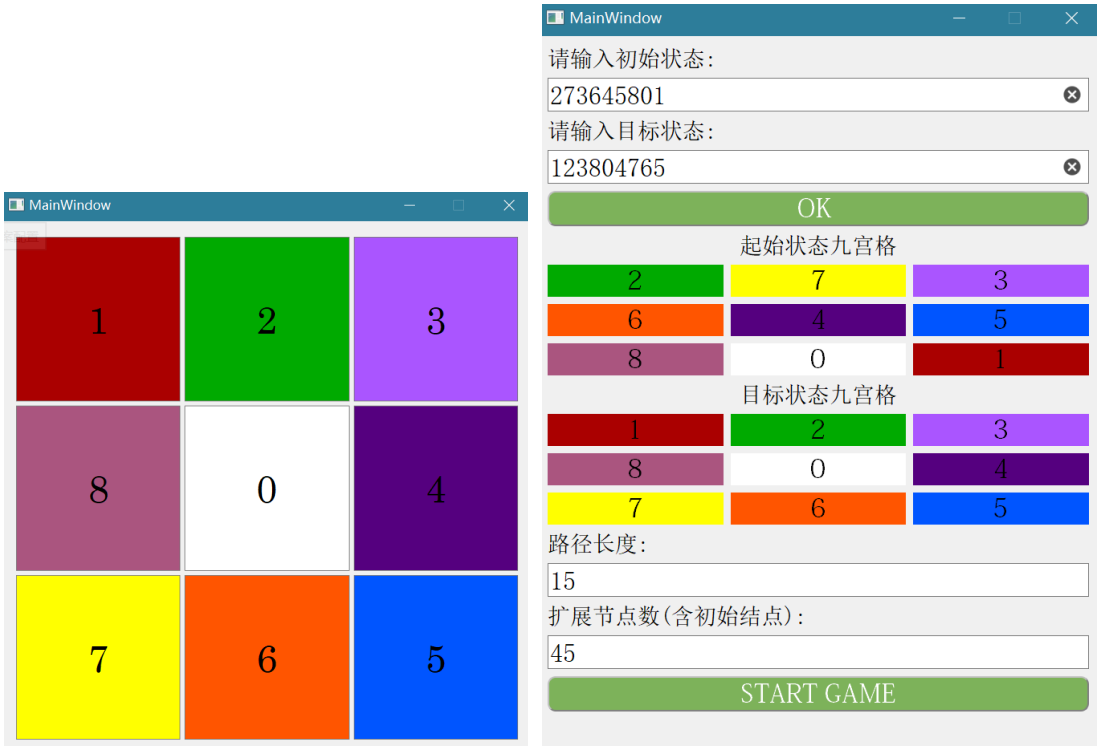
下面将会以截图的形式进行展示图形界面运行结果。

游戏启动界面：

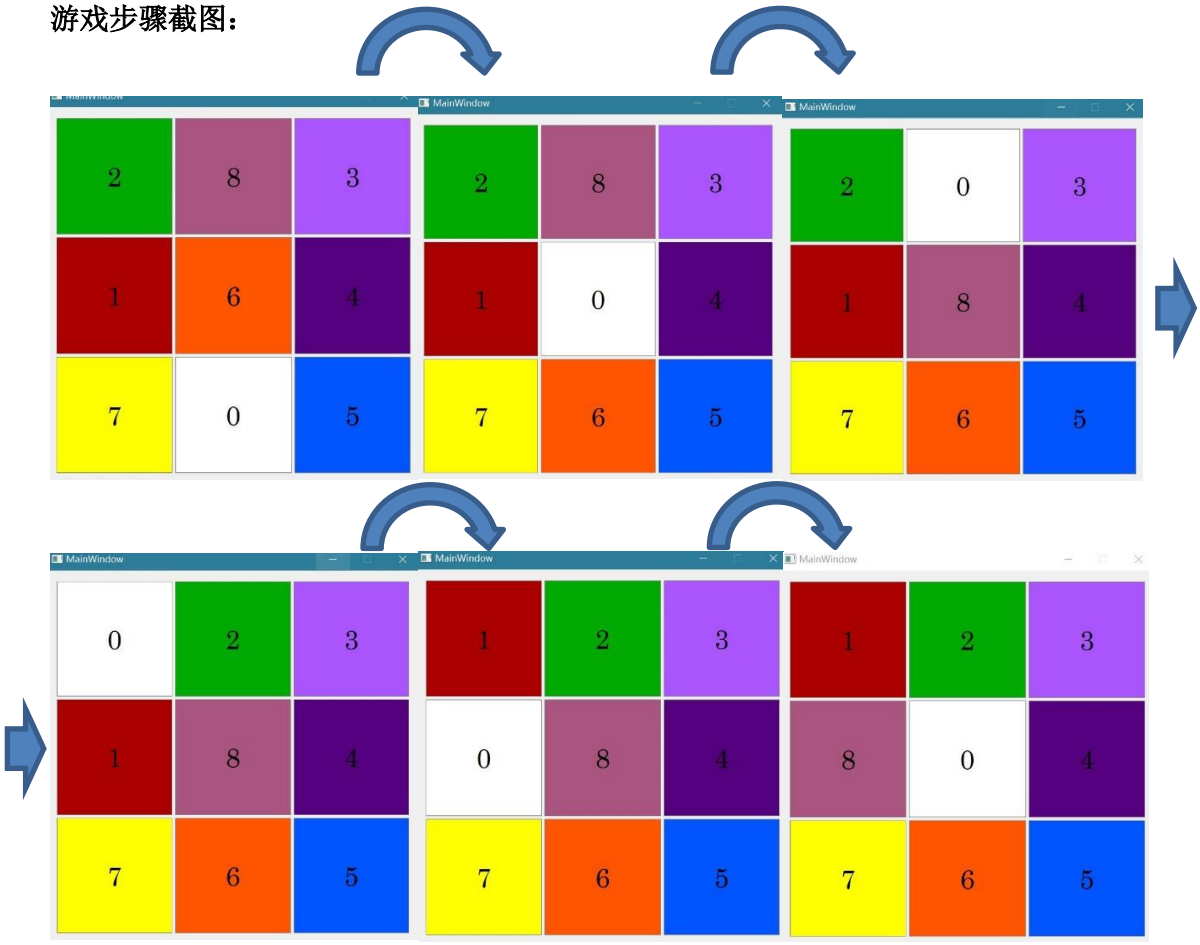


输入及错误提示界面：

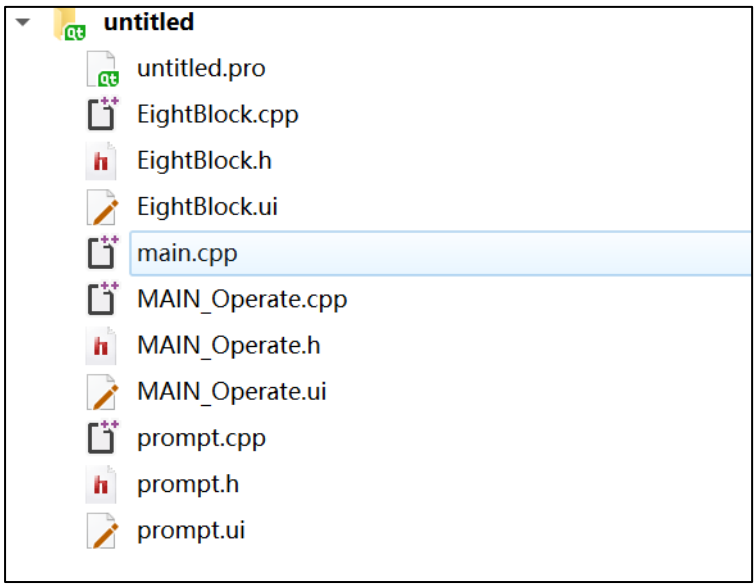
游戏正确运行界面：



游戏步骤截图：



Qt 项目文件概览：



untitled.pro:项目配置文件
EightBlock.Cpp:八数码核心函数定义文件
EightBlock.h:八数码函数声明文件
EightBlock.ui 八数码运行最短路径动态图形界面
main.cpp 总控制主函数
MAIN_Operate.cpp:主控制函数定义文件
MAIN_Operate.cpp:主控制函数声明文件
MAIN_Operate.ui:主控制设计界面（游戏启动界面）
prompt.cpp:无解提示函数定义文件
prompt.h:无解提示函数声明文件
prompt.ui:无解提示图形设计界面
因代码过于长，具体实现代码文件将在附件中呈现。

六、 小结

本次八数码实验主要练习使用 A_star 算法进行搜索，体会启发式搜索算法的精妙之处。它将 BFS 和贪婪算法结合到一起，使得我们每一次的搜索都趋向于最优，从而在满足条件的情况下找到最优解（即最短路径）。当然八数码实验使用的算法只是启发式算执法中的一种，在今后的学习中，要结合多种搜索算法，进行比较尝试，体会各种搜索算法的效率，从而对启发式算法有一个更加深刻的认识。

在本次实验中，我主要是以 bfs 为基础搜索方式，启发式函数融入其中，优先队列加以优化，从而使得启发式搜索得到实现。