装

订

线

A*算法求解8数码问题实验报告

目录

1.实验概述

1.1 实验目的

熟悉和掌握启发式搜索策略的定义、评价函数 f(n)和算法过程，并利用 A*算法求解 8 数码问题，理解求解流程和搜索顺序。

1.2 实验内容

1）以 8 数码问题为例，实现 A*算法的求解程序，设计两种不同的启发函数 h(n)。

2）设置相同初始状态和目标状态，针对不同的评价函数求得问题的解，比较它们对搜索算法性能的影响，包括扩展节点数、生成节点数和运行时间等。画出结果比较的图表，并进行性能分析。

3）界面显示初始状态，目标状态和中间搜索步骤。

4）显示搜索过程，画出搜索过程生成的搜索树，并在每个节点显示对应节点的评价值 f(n)。以红色标注出最终结果所选用的路线。

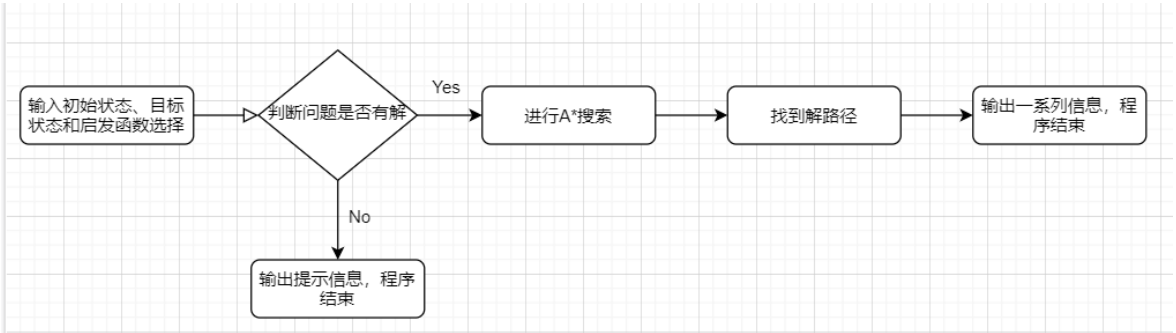
2.实验方案设计

2.1 总体设计思路与总体架构

实验首先输入初始状态、目标状态和评价函数选择。然后判断问题是否有解，若无解则结束，若有解则使用 A*搜索算法找到解。其中第一种启发函数 h1(n)为不在位的棋子数，第二种启发函数 h2(n)为所有棋子到其目标位置的距离和，评估函数 f(n)=d(n)+h(n)，其中 d(n)为所处深度。从初始状态开始进行搜索，生成一系列节点，选择扩展 f(n)最小的节点，由此继续搜索，直到找到最终状态。

由上述思路建立总体架构，共四大模块，分别为输入、判断有无解、搜索及画图。

流程图如下：



2.2 核心算法及基本原理

核心算法为 A*搜索。其思路为避免扩展耗散值已经很大的路径。评估函数 f(n) = g(n) + h(n) g(n)为到达此结点 n 时已经花费的代价,h(n)为从该结点 n 到目标结点的最小代价路径的估计值，

f(n)为经过结点 n 的最小代价解的估计代价。搜索时，在生成的一系列节点中，选择扩展 f(n)最小的节点，再由此节点继续生成并扩展，直至到达目标状态。该算法具有完备性和最佳性，复杂度较低，适合解决 8 数码问题。

装

订

线

2.3 模块设计

本实验共有四大模块分别为输入模块、判断有无解模块和搜索模块。

输入模块接收初始状态、目标状态和启发函数选择的输入。

判断有无解模块用来判断该问题是否有解。

搜索模块进行 A*搜索，找到解路径。

画图模块负责动态展示及画出搜索树。

2.4 其他创新内容或优化算法

1）将字符串转化为整形数进行存储，节省存储空间。

2）利用 map<int, bool>可直接判断节点是否被访问过。

3）使用优先队列，将生成的一系列节点按 f(n)从小到大排在队列里，供下一步扩展。

3.实验过程

3.1 环境说明

操作系统：Windows 10

开发语言：C++

开发环境：Visual Studio 2019

核心使用库：C++ STL 库、EasyX 库

3.2 源代码文件清单，主要函数清单

代码清单：

main.cpp：主函数所在源文件，用于定义一系列全局变量，接收输入，产生成功后的输出。

solvable.cpp：判断源文件，用于判断问题是否有解。

bfsHash.cpp：搜索源文件，用于进行 A*搜索。

dynamic.cpp：画图源文件，用于动态展示及画出搜索树。

all.h：头文件，包含程序所需头文件、宏定义、函数声明、全局变量声明、结构体声明。

主要函数清单：

main 函数：接收输入，产生成功后的输出。

solvable 函数：判断问题是否有解。

bfsHash 函数：进行 A*搜索。

dynamic 函数：动态展示及画出搜索树。

swap 函数：进行两个值之间的交换。

3.3 源代码

all.h：

```
#define CLOCKS_PER_SEC ((clock_t)1000)
#include<iostream>
#include<cstdio>
#include<ctime>
#include<queue>
#include<map>
#include<graphics.h>
using namespace std;
void swap(char* ch, int a, int b);
void bfsHash(int start, int zeroPos, int target, int& extend, int& generate, int choice);
void solvable();
void dynamic(int extend, int generate);
struct node
{
    int num, step, cost, zeroPos, choice;
    bool operator<(const node& a)const;
    node(int n, int s, int p, int c);
    void setCost(int choice);
};
extern char arr[10], brr[10];//初始和目标
extern map<int, bool>mymap;
extern priority_queue<node> que;//优先队列
extern int change[9][4];//每个数上下左右情况
extern char dyna[1000][10];//用于动态展示
struct paint//用于画搜索树
{
    int d;//评估函数值
    int deep;//深度
    int ch;//八数码情况
};

extern paint dynapaint[1000];
```

main.cpp：

```
#include"all.h"
char arr[10], brr[10];
bool node::operator<(const node& a)const
{
```

```
    return cost > a.cost;
}
node::node(int n, int s, int p, int c)
{
    num = n, step = s, zeroPos = p, choice = c;
    setCost(choice);
}
void node::setCost(int choice)
{
    char a[10];
    int c = 0;
    sprintf_s(a, "%09d", num);
    if (choice == 1)
    {
        for (int i = 0; i < 9; i++)
            if (a[i] != brr[i])
                c++;
        cost = c + step;
    }
    else if (choice == 2)
    {
        for (int i = 0; i < 9; i++)
            for (int j = 0; j < 9; j++)
            {
                if (a[i] == brr[j] && a[i] != '0')
                {
                    c += int(fabs(i / 3 - j / 3) + fabs(i % 3 - j % 3));
                }
            }
        cost = c + step;
    }
}
map<int, bool>mymap;
priority_queue<node> que;//优先队列
int change[9][4] = {
    {-1,-1,3,1},
    {-1,0,4,2},
    {-1,1,5,-1},
    {0,-1,6,4},
    {1,3,7,5},
    {2,4,8,-1},
    {3,-1,-1,7},
    {4,6,-1,8},
    {5,7,-1,-1}
};
char dyna[1000][10];
paint dynapaint[1000];
int main()
```

```
{
    int start, zeropos, target, extend = 0, generate = 0, choice = 1;
    clock_t starttime, finishtime;
    double  duration;
    cout << "请输入初始状态：" << endl;
    for (int i = 0; i < 9; i++)
        cin >> arr[i];
    cout << "请输入目标状态：" << endl;
    for (int i = 0; i < 9; i++)
        cin >> brr[i];
    cout << "请输入评价函数选择1/2：" << endl;//评价函数1为深度+不在位的棋子数；评价函数2为深度+所有棋子到其目标位置的距离和
    cin >> choice;
    cout << endl << "结果：" << endl;
    solvable();
    starttime = clock();
    for (zeropos = 0; zeropos < 9; zeropos++)
        if (arr[zeropos] == '0')
            break;
    sscanf_s(arr, "%d", &start);
    sscanf_s(brr, "%d", &target);
    bfsHash(start, zeropos, target, extend, generate, choice);
    finishtime = clock();
    duration = ((double)finishtime - (double)starttime) / CLOCKS_PER_SEC;//运行时间
    cout << "搜索花费的时间：" << duration << "s" << endl;
    cout << "已拓展节点数为：" << extend << endl;
    cout << "已生成节点数为：" << generate << endl;
    cout << "5秒后开始动态展示及画图" << endl;
    dynamic(extend, generate);
    return 0;
}
```

solvable.cpp：

```
#include"all.h"
void solvable()//若两个状态的逆序数奇偶性相同则有解
{
    int sum_a = 0, sum_b = 0;
    for (int i = 0; i < 9; i++)
    {
        if (arr[i] != '0')
        {
            for (int j = i + 1; j < 9; j++)
                if (arr[j] < arr[i] && arr[j] != '0')
                    sum_a++;
        }
    }
    for (int i = 0; i < 9; i++)
```

```
    {
        if (brr[i] != '0')
        {
            for (int j = i + 1; j < 9; j++)
                if (brr[j] < brr[i] && brr[j] != '0')
                    sum_b++;
        }
    }
    if (sum_a % 2 != sum_b % 2)
    {
        cout << "无解" << endl;
        exit(1);
    }

}
```

装

bfsHash.cpp：

```
#include"all.h"
void swap(char* ch, int a, int b)//交换ch[]中a,b两位置的数
{
    char c = ch[a];
    ch[a] = ch[b];
    ch[b] = c;
}
void bfsHash(int start, int zeroPos, int target, int& extend, int& generate, int choice)
{
    char temp[10];
    node A(start, 0, zeroPos, choice);//创建一个节点
    que.push(A);//压入优先队列
    ++generate;//生成节点数+1
    dynapaint[generate - 1].d = A.cost;
    dynapaint[generate - 1].ch = A.num;
    dynapaint[generate - 1].deep = A.step;
    mymap[start] = 1;//标记节点被访问过
    while (!que.empty())
    {
        A = que.top();
        que.pop();
        ++extend;//扩展节点数+1
        sprintf_s(temp, "%09d", A.num);
        strcpy_s(dyna[extend - 1], temp);
        int pos = A.zeroPos, k;
        for (int i = 0; i < 4; i++)
        {
            if (change[pos][i] != -1)
            {
                swap(temp, pos, change[pos][i]);
```

订

线

```
                sscanf_s(temp, "%d", &k);
                if (k == target)
                {
                    strcpy_s(dyna[extend], temp);
                    dynapaint[generate].d = A.step+1;
                    dynapaint[generate].ch = k;
                    dynapaint[generate].deep = A.step + 1;
                    return;
                }
                if (mymap.count(k) == 0)
                {
                    node B(k, A.step + 1, change[pos][i], choice);//创建一个新节点并压入队列
                    que.push(B);
                    ++generate;
                    dynapaint[generate - 1].d = B.cost;
                    dynapaint[generate - 1].ch = B.num;
                    dynapaint[generate - 1].deep = B.step;
                    mymap[k] = 1;
                }
                swap(temp, pos, change[pos][i]);
            }
        }
    }
    return;

}
```

dynamic.cpp：

```
#include"all.h"
void dynamic(int extend, int generate)
{
    int deepnum[5] = { 0 };
    for (int i = 0; i <= generate; i++)
        deepnum[dynapaint[i].deep]++;
    Sleep(5000);//休眠5秒
    initgraph(1440, 720);
    setbkcolor(WHITE); //设置背景色
    settextstyle(30, 30, _T("Courier"));// 设置字体
    settextcolor(BLACK);//设置字体色
    setlinecolor(BLACK);//设置线颜色
    setlinestyle(PS_SOLID, 3);//设置线形
    for (int i = 0; i <= extend; i++)
    {
        char c = i + '1';
        TCHAR s[] = _T("已扩展");//画出扩展情况
        outtextxy(500, 180, s);
```

```
        outtextxy(700, 180, c);
        outtextxy(750, 180, _T("步"));
        for (int j = 0; j < 9; j++)////动态展示
        {
            char c = dyna[i][j];
            int y = j / 3 + 1;
            int x = j % 3 + 1;
            IMAGE img;
            if (c == '0')
                loadimage(&img, _T("0.jpg"));
            if (c == '1')
                loadimage(&img, _T("1.jpg"));
            if (c == '2')
                loadimage(&img, _T("2.jpg"));
            if (c == '3')
                loadimage(&img, _T("3.jpg"));
            if (c == '4')
                loadimage(&img, _T("4.jpg"));
            if (c == '5')
                loadimage(&img, _T("5.jpg"));
            if (c == '6')
                loadimage(&img, _T("6.jpg"));
            if (c == '7')
                loadimage(&img, _T("7.jpg"));
            if (c == '8')
                loadimage(&img, _T("8.jpg"));
            putimage(100 * x, 100 * y, &img);
        }
        Sleep(3000);
        cleardevice();//清空当前绘图设备
    }
    Sleep(5000);
    settextstyle(15, 15, _T("Courier"));
    int linepointx[10] = { 0 };//扩展节点的线端点
    int linepointy[10] = { 0 };
    int count = 0;
    for (int i = 0; i <= generate; i++)
    {
        if (dynapaint[i].deep > 3)//画出前四层
            break;
        int width = 1440 / deepnum[dynapaint[i].deep];//每层每个占据宽度
        int y = 60 + 200* dynapaint[i].deep;//每层深度
        for (int j = 0; j < deepnum[dynapaint[i].deep]; j++)
        {
            int x = width/2+width*j;
            rectangle(x - 45, y - 60, x + 45, y + 60);//画矩形
            char temp[10];
            sprintf_s(temp, "%09d", dynapaint[i+j].ch);
```

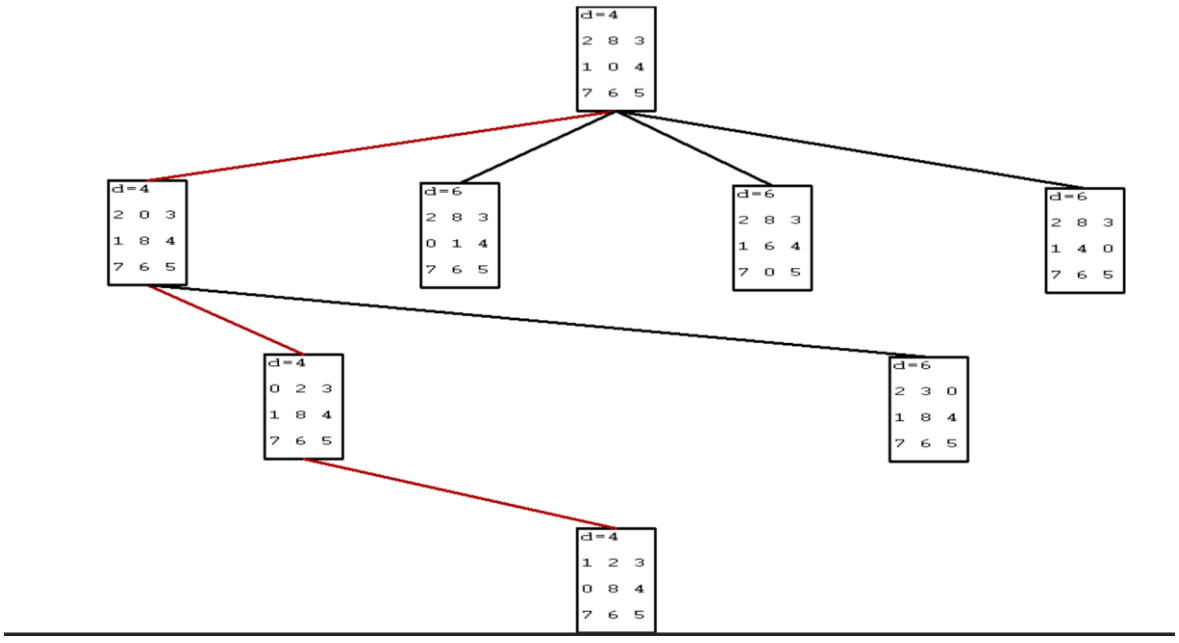
```
            if(dynapaint[i].deep>0)
                line(x, y - 60, linepointx[dynapaint[i].deep-1],
linepointy[dynapaint[i].deep-1]);//画线
            for (int k = 0; k <= extend; k++)//画红色线
            {
                if (strcmp(temp, dyna[k]) == 0)
                {
                    linepointx[count] = x;
                    linepointy[count] = y + 60;
                    count++;
                    setlinecolor(RED);
                    if (dynapaint[i].deep > 0)
                        line(x, y - 60, linepointx[dynapaint[i].deep - 1],
linepointy[dynapaint[i].deep - 1]);
                    setlinecolor(BLACK);
                    break;
                }
            }
            //画出每个的d=? 以及八数码情况
            x += 3;
            y += 3;
            char c1 = 'd';
            char c2 = '=';
            char c3 = dynapaint[i+j].d + '0';
            outtextxy(x - 45, y - 60, c1);
            outtextxy(x - 30, y - 60, c2);
            outtextxy(x - 15, y - 60, c3);
            for (int k = 0; k < 9; k++)
            {
                int yy = k / 3;
                int xx = k % 3;
                outtextxy(x - 45+xx*30, y - 30+yy*30, temp[k]);
            }
        }
        i += (deepnum[dynapaint[i].deep] - 1);
    }
    Sleep(10000);
    closegraph();

}
```

3.4 搜索过程展示

动态展示：

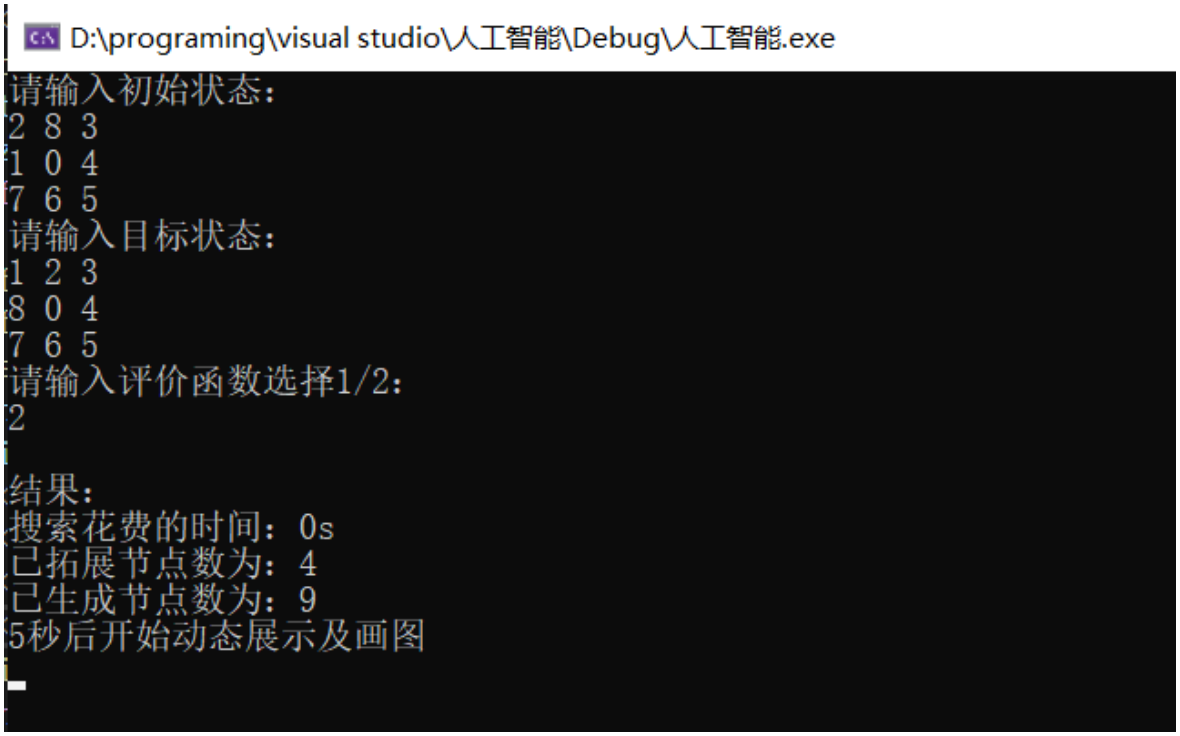
已扩展 2 步

搜索树：



3.5 实验结果展示


Microsoft Visual Studio 调试控制台
请输入初始状态:
2 8 3
1 0 4
7 6 5
请输入目标状态:
1 2 3
8 0 4
7 6 5
请输入评价函数选择1/2:
1
结果:
搜索花费的时间: 0.001s
已拓展节点数为: 5
已生成节点数为: 11
5秒后开始动态展示及画图


D:\programing\visual studio\人工智能\Debug\人工智能.exe
请输入初始状态:
2 8 3
1 0 4
7 6 5
请输入目标状态:
1 2 3
8 0 4
7 6 5
请输入评价函数选择1/2:
2
结果:
搜索花费的时间: 0s
已拓展节点数为: 4
已生成节点数为: 9
5秒后开始动态展示及画图

3.6 实验结论

	1		2		3		4		5	
初始状态	2 8 3 1 0 4 7 6 5	2 8 3 1 0 4 7 6 5	5 1 7 0 3 6 2 8 4	5 1 7 0 3 6 2 8 4	7 4 2 0 5 1 6 3 8	7 4 2 0 5 1 6 3 8	3 0 2 5 6 1 7 8 4	3 0 2 5 6 1 7 8 4	8 0 3 2 6 1 7 4 5	8 0 3 2 6 1 7 4 5
目标状态	1 2 3 8 0 4 7 6 5	1 2 3 8 0 4 7 6 5	2 8 0 4 7 3 1 5 6	2 8 0 4 7 3 1 5 6	4 8 5 6 1 0 7 2 3	4 8 5 6 1 0 7 2 3	1 7 5 2 0 3 8 6 4	1 7 5 2 0 3 8 6 4	1 5 4 6 7 8 2 3 0	1 5 4 6 7 8 2 3 0
启发函数选择	1	2	1	2	1	2	1	2	1	2
花费时间	6	2	173	10	134	14	12	1	402	19
扩展节点	5	4	11621	709	8673	731	839	35	25442	1201
生成节点	11	9	17645	1117	13752	1174	1429	63	37052	1914

注：时间单位为毫秒

由图中可知，一般情况下，选择第二种启发函数（所有棋子到其目标位置的距离和）比第一种启发函数（不在位的棋子数）更优，花费时间更少，扩展节点和生成节点更少。在较复杂的情况下，第二种启发函数在时间和空间上，比第一种启发函数小 1 到 2 个数量级。

4. 总结

4.1 实验中存在的问题及解决方案

1）搜索时如何保证扩展的是 f(n)最小的节点。刚开始时想用链表保存生成的节点，然后排序；最终通过使用优先队列解决。

2）如何保证扩展过的节点不会被重复扩展。利用 map<int, bool>解决。

4.2 心得体会

回顾这次实验，感觉十分的充实，通过认真学习、亲自动手，使我进一步了解了 A*搜索的基本知识和设计方法，为今后的学习奠定了良好的基础。理论课让我们系统地学习了搜索的相关理论。而实验课给了我们一个很好的把理论应用到实践的平台，让我们能够很好的把书本知识转化到实际能力，提高了对于理论知识的理解，认识和掌握。其次，实验很好地解决了我们实践能力不足且得不到很好锻炼机会的矛盾，通过实验，提高了自身的实践能力和思考能力，并且能够通过实验很好解决自己对于理论的学习中存在的一些知识盲点。以此次实验为例，不仅完成了实验设计，充分理解了 A*搜索，而且复习了 C++相关知识，同时加强了对 STL 的运用。

4.3 后续改进方向

1）找到更好的启发函数，进一步减小问题规模。

2）实现搜索树动态可视化，更利于理解 A*搜索。

3）将 A*搜索与其他搜索方法，例如 BFS、DFS 等对比，探索 8 数码问题的多种解决方法。

4.4 总结

搜索问题是一类常见的问题，应对搜索问题有各类搜索算法。对于 8 数码问题，A*搜索算法是一种优秀的算法，如果选择合理的启发函数，将大大地减小问题规模。因此，如何寻找合理的启发函数，便是 A*搜索的关键。另外在程序编写过程中，要充分利用 STL 库，例如优先队列、map 等，可以减小程序规模，同时提高性能。本次实验按照预期完成，A*搜索及两种启发函数设计成功，并高效地解决了 8 数码问题。但寻找其他启发函数、搜索可视化、多种搜索比较等仍可以做进一步的开发。

5.参考文献

[1]Stuart Russell, Peter Norvig.人工智能:一种现代的方法[M].清华大学出版社:北京,2013:82-95.