《当代人工智能》实验报告

报告题目：当代人工智能——A*算法

姓　　名：

学　　号：

完成日期：2022 年 4 月 3 日星期日

华东师范大学数据科学与工程学院实验报告

课程名称：当代人工智能	**年级**：2019	**上机实践成绩**：
指导教师：李翔	**姓名**：	**学号**：
上机实践名称：Project 2 A*算法		**上机实践日期**：3 月

当代人工智能——A*算法

一、 项目目的

掌握 A*搜索算法，并了解不同的搜索算法。学习使用 A*算法求解实际问题。

二、 项目任务

两道算法题，只能用 A*算法实现。

三、 项目要求

1、 编程实现，不限语言。

2、 每道题有 5 个测试输入，你需要用你的算法去预测对应输出。

3、 撰写实验报告，介绍算法设计思路。并在实验报告里添加一节，把对应测试输入的输出加在该节中。

四、 实验环境

使用 C++语言在 CodeBlocks 的环境下进行代码的编写，完成运用 A*算法求解实际问题的任务。

执行代码依赖的库：

```
#include <iostream>
#include <queue>
#include <vector>
#include <map>
#include <cmath>
```

#include <iostream>

#include <queue>

#include <vector>

#include <map>

#include <cmath>

五、 算法简介

UCS 搜索算法回首过去，展开当前已花费代价最小的节点，用 g(n)表示已选路径的代价。贪心 Greedy 算法展望未来，展开启发式估计代价最小的节点，用 h(n)表示到达目标结点的代价。

A*算法结合了 UCS 和 Greedy 的优点，同时回首过去和展望未来，展开当前已花费代价和启发式估计代价总和最小的节点，用 f(n) = g(n) + h(n) 表示已选路径的代价与到达目标结点的代价的总和。

A* Search 中单调队列按照 f(n) = g(n) + h(n)的值从小到大进行维护，每次展开队首元素，即 f(n)最小的节点。

A*算法终止条件。第一次遇见目标节点时不能退出，继续将其加入队列（此时不一定最优），当目标节点从队列中弹出时终止。

A*算法最优的前提。满足 Admissible Heuristics（容许的启发），即估计代价 h 应在 0 和真实代价之间，0 < h(n) < h*(n)，h*(n) 是真实的代价。

六、 实验结果

Q1: 小明玩球测试输出：

输入输出样例:

```
test input 1:  Find the state.
cost:  22
test input 2:  Find the state.
cost:  26
test input 3:  Find the state.
cost:  21
test input 4:  Find the state.
cost:  26
test input 5:  Find the state.
cost:  0

Process returned 0 (0x0)   execution time : 0.103 s
Press any key to continue.
```

测试输入：

输入 1 ： 024657318

输出 1 ： 22

输入 2 ： 587346120

输出 2 ： 26

输入 3 ： 375148206

输出 3 ： 21

输入 4 ： 512768340

输出 4 ： 26

输入 5 ： 123804765

输出 5 ： 0

Q2：爱跑步的小明测试输出：

输入输出样例:

```
5 8 7
5 4 1
5 3 1
5 2 1
5 1 1
4 3 4
3 1 1
3 2 1
2 1 1
test input:  Find the state.
cost:
1
2
2
3
6
7
-1

Process returned 0 (0x0)   execution time : 0.825 s
Press any key to continue.
```

测试输入：

输出 1 ： 1

2

2

输出 2 ： 2

3

3

3

输出 3 ： 2

3

3

3

4

4

7

8

-1

-1

-1

-1

输出 4 ： 2

3

4

4

5

6

7

7

输出 5 ： 2

3

4

4

5

6

7

7

7

8

8

8

8

9

9

10

七、 实验过程（算法设计思路）

Q1: 小明玩球

小明在一个九宫格中随机摆了八个球，每个球上标有 1-8 中的某一数字（球上数字不重 复）。九宫格中留有一个空格，该空格用0 表示。空格周围的球可以移动到

空格中。现 在，给出一种初始布局（即初始状态）和目标布局（本题的目标布局设为 123804765），现在小明想找到一种最少步骤的移动方法，实现从初始布局到目标布局 的转变，你能帮帮他吗？

Q1：算法设计思路：

注：在本题中，测试用例直接以全局变量的形式定义出了，不必再进行输入。

（1）设计启发式函数 h(n)

```
// 启发式函数h
int compute_h(void)
{
    int h = 0;
    for(int number = 1; number <= 8; number++)
    {
        int this_i, this_j;
        int goal_i, goal_j;
        for(int i = 0; i < 3; i++)
        {
            for(int j = 0; j < 3; j++)
            {
                if(state_goal_matrix[i][j] == '0' + number)
                {
                    goal_i = i;
                    goal_j = j;
                }
                if(this->state_matrix[i][j] == '0' + number)
                {
                    this_i = i;
                    this_j = j;
                }
            }
        }
        h += abs(this_i - goal_i) + abs(this_j - goal_j);
    }
    return h;
}
```

在本题中，状态 n 与目标状态的估计距离为：每一个数字（数字 1—8）在此状态中的位置与该数字在目标状态中位置的**曼哈顿距离**的总和。例如，数字 1 在状态 A 中的位置为（0，0），数字 1 在状态 B 中的位置为（1，2），则数字 1 的曼哈顿距离为 3，即横纵坐标的差值绝对值的和。那么状态 A 与状态 B 的估计距离为，数字 1—8 的曼哈顿距离总和。

容许的启发：

容易看出，在排除移动干扰的情况下，把某数字移动到目标位置的最小距离代价为它的曼哈顿距离，因此，把所有元素都移动到目标位置的最小代价为每个数字距离目标位置的曼哈顿距离的总和，即为 h(n)。考虑干扰移动的情况，实际移动代价 h*(n)必定大于等于 h(n)。因此，该启发为容许的启发。

（2）设计面向对象类

```
// 状态类
class State_Info
{
public:
    string state_str;
    int cost_g;
    int cost_h;
    int cost_f;
    int zero_i;
    int zero_j;
    char state_matrix[3][3];

    State_Info(string input_str, int g)
    {
```

在 State_Info 保存每个状态的信息，以字符串和矩阵的方式保存当前的状态，以及保存 g(n),h(n),f(n)等。同时，在类中编写必要的成员函数，使之具备以下功能：

1、 构造函数初始化成员变量;

2、 计算启发式函数 h(n)的估计代价，并计算 f(n);

3、 根据目前的状态，生成下一步可到达的状态，即相邻的节点（通过移动数字 0）；

（3）A*算法求解

求解步骤如下：

1、 构造升序队列，队列中为 pair 元素，pair 中第一个元素为 f(n)的

值，pair 中第二个元素为指向保存某状态的 State_Info 类实例的指针。升序队列通过 f(n)的值排序，队首元素的 f(n)值最小。

2、 构造 map，记录已经从队列中弹出的元素。本题为寻找最优的代价，因此不必重复访问。

3、 构造循环。在每次循环中，弹出队首元素（f 值最小的元素），并将此元素下一步能够到达的状态加入至队列中（已经弹出的元素不可到达，通过 map 判断是否是已经弹出的元素）。循环的终止条件为，从队列中弹出目标状态。该目标状态的 g(n)（f(n)与 g(n)相等）即为最小代价，最少步骤的移动方法。

```
void A_star_algorithm(string state_input_str)
{
    priority_queue <pair<int, State_Info*>, vector<pair<int, State_Info*> >, greater<pair<int, State_Info*> > > q;
    State_Info* state = new State_Info(state_input_str, 0);
    q.push(make_pair(state->cost_f, state));

    State_Info* tmp = q.top().second;
    map<string, int> visited;
    int n = 0;
    while(q.size() && tmp->state_str != state_goal_str)
    {
        n++;
        q.pop();
        visited[tmp->state_str] = 1;
        vector<string> next = tmp->generate_next();
        for(unsigned i = 0; i < next.size(); i++)
        {
            if(visited.count(next[i]) == 1)  continue;
            State_Info* state_next = new State_Info(next[i], tmp->cost_g + 1);
            q.push(make_pair(state_next->cost_f, state_next));
        }
        tmp = q.top().second;
    }
    if(q.size() == 0)
    {
        cout << "Don't find the state." << endl;
    }else{
        cout << "Find the state." << endl;
        cout << "cost:  ";
        cout << tmp->cost_g << endl;
        //cout << tmp->state_str << endl;
        //cout << n << endl;
    }
}
```

Q2: 爱跑步的小明

众所周知，小明身材很好。但自从他博士毕业当老师后，他就自我感觉身体变差了，于 是他就想锻炼了。为了不使自己太累，他提出一种从山顶跑步到山脚的锻炼方法。 千寻万觅，终于在郊区找到这样一座山。这座山有 N 个地标，有先行者在这些地标之间 开辟了 M 条道路。并且这些地标按照海拔从低到高进行了编号，例如山脚是 1，山顶是 N。 小明这个人对跑步的方式很挑：

（1）只跑最短路径。但一条最短路径跑久了会烦，需要帮他设计 K 条最短路径。

（2）不想太累，每次选道路的时候只从(海拔的)高处到低处。

现在问题来了，给你一份这座山地标间道路的列表，每条道路用(X , Y , D)表示，表示 地标 X 和地标 Y 之间有一条长度为 D 的下坡道路。你来计算下小明这 K 条路径的对应长度，看看小明的锻炼强度大不大？

Q2: 算法设计思路：

（1）设计启发式函数 h(n)

在本题中，状态 n 与目标状态的估计距离为：该状态与该状态下一步可到达状态的距离中的最小值。例如对于以下输入：

```
5 8 7
5 4 1
5 3 1
5 2 1
5 1 1
4 3 4
3 1 1
3 2 1
2 1 1
```

状态 5 到目标状态 1 的启发估计距离为 1，因为状态 5 到下一步可到达状态的距离分别为 1，1，1，1，其中最小值为 1。

容许的启发：

容易看出，某状态与该状态下一步可到达状态的距离中的最小值，必定小于此状态到目标状态的实际距离。因为，该状态到目标状态的最短路径中，该状态到下一个状态的距离必定大于上述中的最小值。因此该启发为容许的启发。

```
void compute_f(int M, int route[][3])
{
    if(state == 1)
    {
        return;
    }
    int Min = unlinked;
    for(int i = 0; i < M; i++)
    {
        if(route[i][0] == state && route[i][2] < Min)
        {
            Min = route[i][2];
        }
    }
    this->cost_h = Min;
    this->cost_f = cost_g + cost_h;
}
```

（2）设计面向对象类

```
// 状态类
class State_Info
{
public:
    int state;
    int cost_g;
    int cost_h;
    int cost_f;

    State_Info(int input_state, int g)
    {
```

在 State_Info 保存每个状态的信息，用数字表示当前状态，即当前位置，同时保存 g(n),h(n),f(n)等信息。并且，在类中编写必要的成员函数，使之具备以下功能：

1、 构造函数初始化成员变量;

2、 计算启发式函数 h(n)的估计代价，并计算 f(n);

3、 根据目前的状态，生成下一步可到底的状态，即相邻的节点

（通过路径判断）；

（3）A*算法求解

求解步骤如下：

1、 构造升序队列，队列中为 pair 元素，pair 中第一个元素为 f(n)的值，pair 中第二个元素为指向保存某状态的 State_Info 类实例的指针。升序队列通过 f(n)的值排序，队首元素的 f(n)值最小。

2、 根据输入的路径构造邻阶矩阵记录距离，避免每次遍历输入路径，减少计算代价。

3、 构造循环。在每次循环中，弹出队首元素（f 值最小的元素），并将此元素下一步能够到达的状态加入至队列中（由于本题寻找 K 条最短路径，因此只要存在路径，已经弹出的元素也可到达）。循环的终止条件为，从队列中弹出的目标状态数达到 K（已经找到 K 条最短路径）或者队列中的元素已经全部弹出（最短路径不足 K 条）。

4、 若弹出的目标状态数不足 K，在输出结果中应相应补 -1。

```
vector<int> shortest_cost;
priority_queue <pair<int, State_Info*>, vector<pair<int, State_Info*> >, greater<pair<int, State_Info*> > > q;
State_Info* tmp = new State_Info(N, 0);
q.push(make_pair(tmp->cost_f, tmp));

int n = 0;
while(!q.empty() && shortest_cost.size() < K)
{
    q.pop();
    //cout << "state:" << tmp->state << endl;
    //cout << "cost_g:" << tmp->cost_g << endl;
    //cout << "cost_f:" << tmp->cost_f << endl;
    if(tmp->state == 1)
    {
        shortest_cost.push_back(tmp->cost_g);
    }
    vector<int> next = tmp->generate_next(M, route);
    for(unsigned i = 0; i < next.size(); i++)
    {
        int cost_g_next = tmp->cost_g + matrix[tmp->state][next[i]];
        State_Info* state_next = new State_Info(next[i], cost_g_next);
        state_next->compute_f(M, route);
        q.push(make_pair(state_next->cost_f, state_next));
    }
    tmp = q.top().second;
}

if(q.size() == 0)
{
    for(int i = K - shortest_cost.size(); i > 0; i--)
    {
        shortest_cost.push_back(-1);
    }
}
```

八、 实验总结

在本次实验中，我完成了使用 A*算法求解实际搜索问题的任务。在运用 A*算法时，进行了启发式函数 h(n)的设计,从而求得 f(n) = g(n) + h(n)。为保证求得最优的结果，在设计启发式函数时应注意满足 Admissible Heuristics（容许的启发）条件，即估计代价 h 应在 0 和真实代价之间，满足 0 < h (n)< h*(n)，其中 h*(n) 是真实的代价。

九、 源代码

源代码附在文件内；

（包含第一题与第二题）；