厦門大學

软件学院

《人工智能导论》实验报告

题　　目　启发式搜索

姓　　名　陈澄

学　　号　32420212202930

班　　级　软工三班

实验时间　2024/03/07

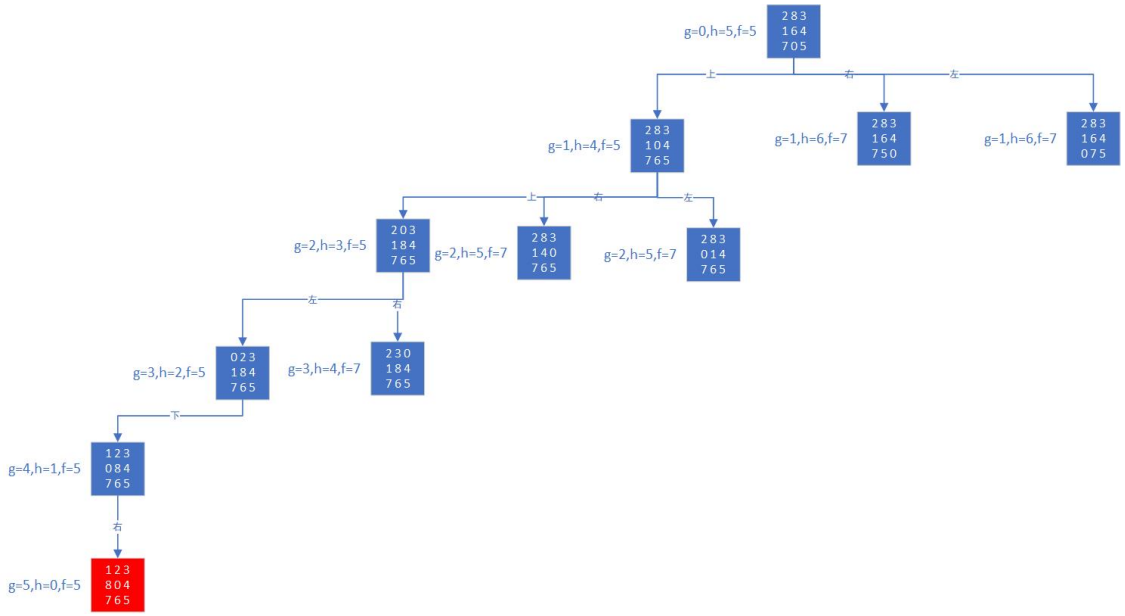
2024　年　03　月　07　日

1 实验目的

利用启发式搜索（A*算法）解决 8 数码问题

2 实验步骤

1. 画出八数码解过程的节点状态图。



2. 编程解决八数码问题

该代码定义 State 结构，用于表示八数码问题的状态，包括当前的棋盘布局、到达该状态的成本 g、估计到目标状态的成本 h、空白格的位置以及到达该状态的路径。aStar 函数实现了 A*算法，使用优先队列（基于估价函数 f 值排序）来选择下一步扩展的状态。在扩展状态时，代码会生成所有可能的后继状态，并更新它们的 g、h 和 f 值，然后将它们加入优先队列中进行处理。

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

using namespace std;

const int N = 3; // 3x3的八数码问题
struct State {
    vector<vector<int>> board;
    int g; // 到达当前状态的代价
    int h; // 启发式估计到目标状态的代价
    int f; // f = g + h
    pair<int, int> zero; // 记录空白格的位置（即数字0的位置）
    string path; // 记录到达当前状态的路径

    bool operator>(const State& s) const {
        return f > s.f;
    }
};

// 计算曼哈顿距离
int heuristic(const vector<vector<int>>& current, const vector<vector<int>>& goal) {
    int dist = 0;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            if (current[i][j] == 0) continue; // 忽略空白格
            for (int m = 0; m < N; ++m) {
                for (int n = 0; n < N; ++n) {
                    if (current[i][j] == goal[m][n]) {
                        dist += abs(i - m) + abs(j - n);
                    }
                }
            }
        }
    }
    return dist;
}

// 检查状态是否为目标状态
bool isGoal(const vector<vector<int>>& state, const vector<vector<int>>& goal) {
    for (int i = 0; i < N; ++i)
        for (int j = 0; j < N; ++j)
            if (state[i][j] != goal[i][j])
                return false;
    return true;
}

// 打印路径
void printPath(const string& path) {
    for (char move : path) {
        switch (move) {
        case 'U': cout << "Up "; break;
        case 'D': cout << "Down "; break;
        case 'L': cout << "Left "; break;
        case 'R': cout << "Right "; break;
        }
    }
    cout << "\n";
}

// A* 算法
void aStar(vector<vector<int>> start, vector<vector<int>> goal) {
    priority_queue<State, vector<State>, greater<State>> openSet;
    map<vector<vector<int>>, bool> visited;

    pair<int, int> zeroPos;
    for (int i = 0; i < N; ++i) {
        for (int j = 0; j < N; ++j) {
            if (start[i][j] == 0) {
                zeroPos = { i, j };
                break;
            }
        }
    }

    State initialState{ start, 0, heuristic(start, goal), heuristic(start, goal), zeroPos, "" };
    openSet.push(initialState);

    while (!openSet.empty()) {
        State currentState = openSet.top();
        openSet.pop();//从open表中删除第一个状态

        visited[currentState.board] = true;

        if (isGoal(currentState.board, goal)) {
            cout << "Solution found!\n";
            printPath(currentState.path);
            return;
        }//如果已经达到目标状态则返回

        vector<pair<int, int>> directions = { {0, 1}, {0, -1}, {1, 0}, {-1, 0} }; // 右，左，下，上
        string moves = "RLDU"; // 对应方向的移动

        for (int i = 0; i < 4; ++i) {
            pair<int, int> nextZero = { currentState.zero.first + directions[i].first, currentState.zero.second + directions[i].second };

            if (nextZero.first >= 0 && nextZero.first < N && nextZero.second >= 0 && nextZero.second < N) {
                State nextState = currentState;
                swap(nextState.board[currentState.zero.first][currentState.zero.second], nextState.board[nextZero.first][nextZero.second]);
                nextState.zero = nextZero;
                nextState.g++;
                nextState.h = heuristic(nextState.board, goal);
                nextState.f = nextState.g + nextState.h;
                nextState.path += moves[i];
                //printPath(currentState.path);
                if (!visited[nextState.board]) {
                    openSet.push(nextState);
                }
            }
        }
    }

    cout << "No solution found.\n";
}

int main() {
    vector<vector<int>> start = { {2, 8, 3}, {1, 6, 4}, {7, 0, 5} };
    vector<vector<int>> goal = { {1, 2, 3}, {8, 0, 4}, {7, 6, 5} };

    aStar(start, goal);

    return 0;
}
```

3. 运行结果展示

如图：空白格需要移动五次：上上左下右，即可到达目标状态

3 实验遇到的问题及其解决方法

无

4 我的体会

启发式搜索是一种在解决问题时用以指导搜索方向的策略，它通过评估哪些路径最有可能达到目标来减少搜索空间，从而提高搜索效率。启发式搜索的核心在于启发式函数，一个好的启发式函数可以显著提高搜索效率，因为它能有效地指导搜索过程朝着更有希望的方向前进。设计一个既准确又高效的启发式函数是实现有效启发式搜索的关键。