# 《人工智能》实验二报告

|  |  |  |  |
| --- | --- | --- | --- |
| 姓名 | 孙潇桐 | 学号 | 2021117405 |
| 实验地点 | 院楼321机房 | 实验日期 | 2023-10-23 |

**一、实验内容**

Building the following state space search algorithms in Prolog or Lisp or Java :

（1）depth-first or breadth-first ( iterative, not recursive )

（2）best-first search

* These search control algorithms should be applied to one of the following problems:

（1）The 8-puzzle (p. 89), and

（2）The Tile Problem ( p. 162, no. 5 ).

* Note that these problems also give the students a nice opportunity to consider various representations and their appropriateness for each problem.

1. **实验原理**

**实验分为两个部分**

**第一个部分我选择使用bfs。就是直接暴力搜索，每次扩展的节点就是将空格移动到四个方向（如果移动合法的话），然后按层扩展，每次扩展所有的子节点。为了防止算法绕圈，加入了去重操作，不再扩展之前扩展过的状态。**

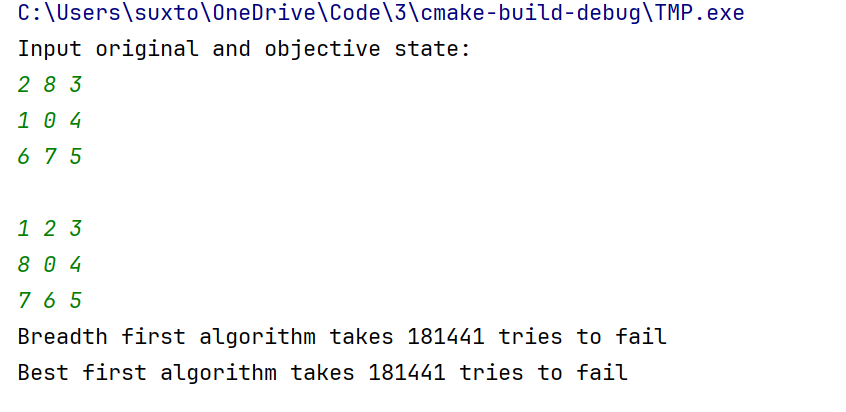
**第二个实验，我使用的启发函数是当前状态和目标状态的曼哈顿距离来优化bfs，优先扩展距离小的状态。**

**实验代码：**

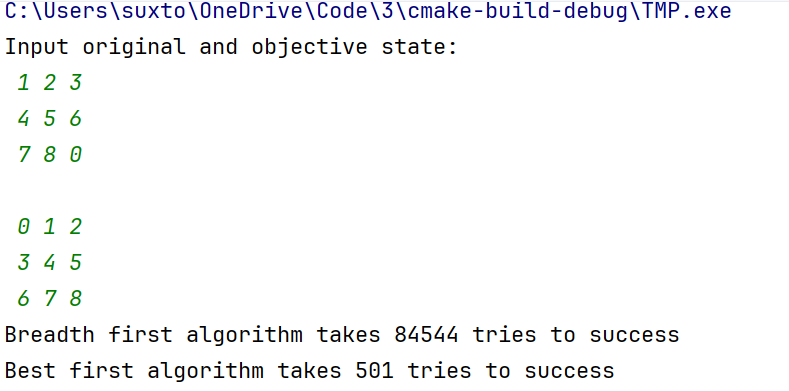
#include<iostream>  
#include<queue>  
#include<array>  
#include <cmath>  
#include <set>  
  
using namespace std;  
  
constexpr array<int, 4> dr{1, -1, 0, 0}, dc{0, 0, 1, -1};  
  
using grid = array<array<int, 3>, 3>;  
using pii = pair<int, int>;  
array<pii, 9> pos;  
set<int> vis;  
grid obj;  
  
struct node {  
 grid g;  
 int step;  
 int val = 0;  
 int od = 0;  
 pii zero;  
  
 explicit node(grid grid, int s) : g(grid), step(s) {  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 3; j++) {  
 od \*= 10;  
 od += grid[i][j];  
 auto [r, c] = pos[grid[i][j]];  
 val += abs(i - r) + abs(j - c);  
 }  
 }  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 3; j++) {  
 if (!grid[i][j]) {  
 zero = {i, j};  
 goto done;  
 }  
 }  
 }  
 done:;  
 }  
  
  
 bool operator<(const node &n) const {  
 return this->val > n.val;  
 }  
  
 [[nodiscard]] bool done() const {  
*// for (int i = 0; i < 3; i++) {  
// for (int j = 0; j < 3; j++) {  
// if (g[i][j] != obj[i][j])return false;  
// }  
// }  
// return true;* return val == 0;  
 }  
  
};  
  
istream &operator>>(istream &is, grid &g) {  
 for (auto &r: g) {  
 for (auto &x: r) is >> x;  
 }  
 return is;  
}  
  
  
ostream &operator<<(ostream &os, grid &g) {  
 for (auto &r: g) {  
 for (auto &x: r) {  
 os << x << ' ';  
 }  
 os << '\n';  
 }  
 return os;  
}  
  
  
template<typename T>  
void extend(T &q, node &now) {  
 auto [r, c] = now.zero;  
 for (int i = 0; i < 4; i++) {  
 int nr = r + dr[i];  
 int nc = c + dc[i];  
 if (nr < 0 || nr > 2 || nc < 0 || nc > 2) continue;  
 grid new\_grid = now.g;  
 swap(new\_grid[r][c], new\_grid[nr][nc]);  
 node new\_node(new\_grid, now.step + 1);  
 if (vis.count(new\_node.od)) continue;  
 vis.insert(new\_node.od);  
 q.emplace(new\_node);  
 }  
}  
  
pair<int, bool> best\_first(grid ori) {  
 int tries = 0;  
 vis.clear();  
 priority\_queue<node> pq;  
 pq.emplace(ori, 0);  
 while (!pq.empty()) {  
 tries++;  
 auto now = pq.top();  
 if (pq.top().done()) {  
 return {tries, false};  
 }  
 pq.pop();  
 extend(pq, now);  
 }  
 return {tries, true};  
}  
  
pair<int, bool> breadth\_first(grid ori) {  
 int tries = 0;  
 vis.clear();  
 queue<node> q;  
 q.emplace(ori, 0);  
 while (!q.empty()) {  
 tries++;  
 auto now = q.front();  
 if (q.front().done()) {  
 return {tries, false};  
 }  
 q.pop();  
 extend(q, now);  
 }  
 return {tries, true};  
}  
  
void get\_pos() {  
 for (int i = 0; i < 3; i++) {  
 for (int j = 0; j < 3; j++) {  
 pos[obj[i][j]] = {i, j};  
 }  
 }  
}  
  
int main() {  
 grid ori;  
 cout << "Input original and objective state:\n";  
 cin >> ori >> obj;  
 get\_pos();  
 auto [t, f] = breadth\_first(ori);  
 cout << "Breadth first algorithm takes " << t << " tries to " << (f ? "fail" : "success") << '\n';  
 tie(t, f) = best\_first(ori);  
 cout << "Best first algorithm takes " << t << " tries to " << (f ? "fail" : "success") << '\n';  
}

1. **实验过程以及结果分析**

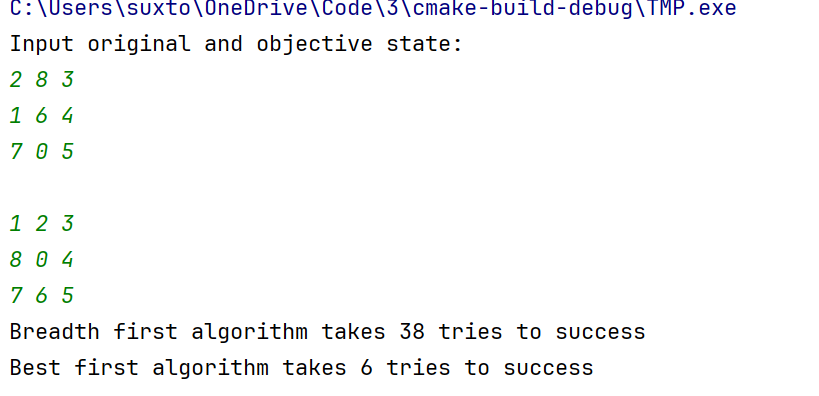
**样例一：无法得到结果的情况，因为无法得到正确结果，所以扩展了所有结点，在这种情况下best-first搜索并没有太大优势**

****

**样例二：复杂的移动，移动越复杂，启发函数提供的优化就越明显**

****

**样例三：简单的移动**

****

1. **实验总结**

**这次我完成了普通bfs和启发函数优化的bfs，很显然，启发函数对时间复杂度的优化是巨大的，不管是在简单的情况还是复杂的情况，不过根据实验结果情况越复杂的时候优化的幅度就越大。以后写搜索算法的时候应该注意启发函数的选择。**