八数码难题实验

北方工业大学

Author: Jingbo Su

North China University of Technology

1 实验目的

1.1 熟悉人工智能系统中的问题求解过程

1.2 熟悉状态空间的盲目搜索和启发式搜索算法的应用

1.3 熟悉对八数码问题的建模、求解以及编程语言应用

2 实验原理

主要使用到了 **A-Star (A*) Algorithm** ，A* (A-Star)算法是一种静态路网中求解最短路最有效的直接搜索方法，其适用于对状态量巨大（例如. 八数码难题）的问题进行搜索效率上的优化。

A* 算法对于确保有解的问题能够获得效率上的优化，如果问题无解，那么 A* 算法会对状态进行暴搜，效率不能确保比传统 BFS 来的高。

A* 算法中重要的一步为设计估计函数 `function f()` ，其意义为对当前状态到目标状态转移消耗的代价，我将其设计为通过当前的状态到目标状态的转移代价的 **Manhattan Distance** 生成 ‼️。

公式表示为：$d(u)+f(u)\leq d(u)+g(u)=h(u)$ ，核心条件：$for\ \forall u\in seq_{st}, f(u)\leq g(u)$ 。其中 `g(u)` 表示当前状态 `u` 到 `target` 的实际距离，`f(u)` 为估计函数，`d(u)` 为从起始状态点至当前点最小距离，`h(u)` 即表示最优解。

`f(u)` 的取值条件：$0\leq f(u)\leq g(u)$ ‼️ 如果 `f(u)` 过大，则搜索不到最优解；如果 `f(u)` 过小，会退化为**Dijkstra Algorithm**，效率低下。

2.1 Proofs & Conditions

2.1.1 对于八数码问题，存在一个问题有解的充要条件：状态序列中逆序对的数量为偶数

必要性证明：如果有解 => 逆序对数量 even

空格（0）在行内（左/右）移动时对于序列：不会改变状态逆序对数量

空格（0）在列内（上/下）移动时对于序列：只会改变两个逆序对的关系，不影响总体逆序对数量的奇偶性

因此我们只需要提前处理一下，计算出起始状态的逆序对数量与目标状态的逆序对数量后对照奇偶性即可

如果奇偶性相同说明存在解可以搜索，如果奇偶性不同，那么问题无解不需要进行搜索。

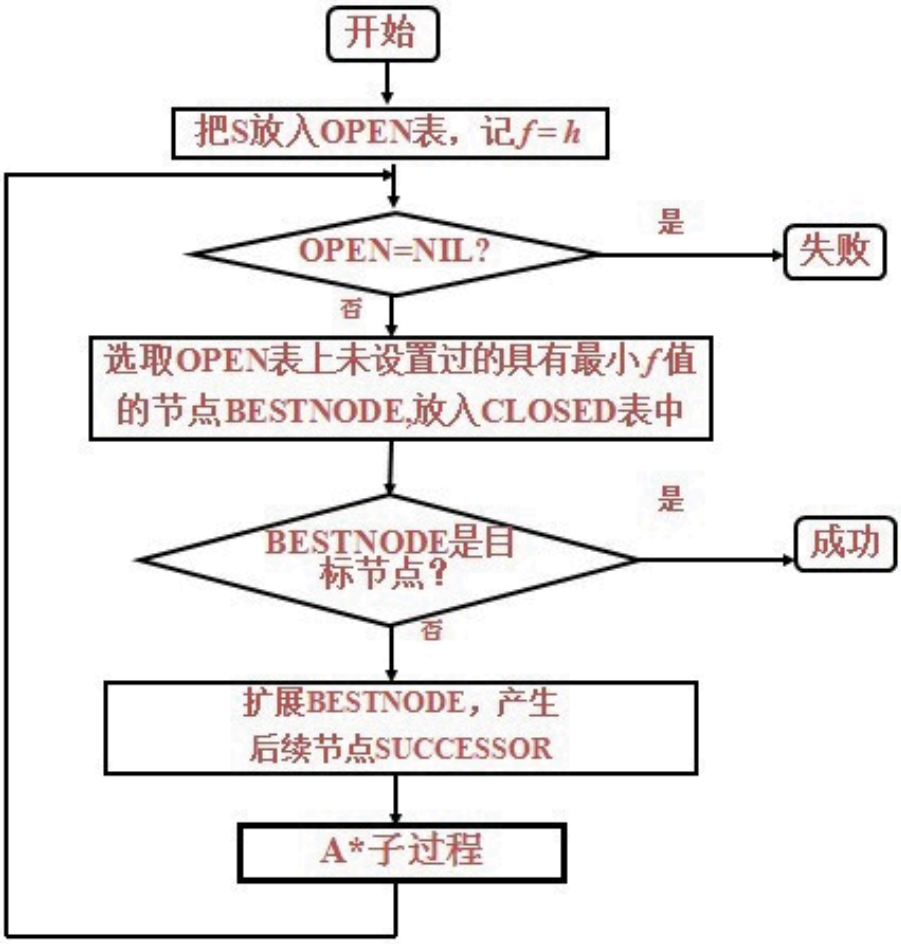
2.1.2 优先队列（小根堆）队头元素一定位于最优解路径上吗？（反证法）

如果队头元素 `T` （设为 `dist` ），如果 $dist > d_{best}$，又 $d_{best} \geq d_u + f_u$，则说明小根堆中存在比当前队头元素更小的元素（这与小根堆的性质相矛盾），因此队头元素一定位于最优解路径上。

2.1.3 A* 算法只能保证当目标状态（终点）出队时距离是最小的（最优解），不能保证搜索过程中除了终点以外出队的每一个点距离是最小的（最优解），并且不能保证搜索过程中除了终点以外的每一个点只被搜索一次

如果当前搜索的路径不是最短路径，因此它的距离一定 > 最优解，因此这条路径在搜索到终点之前一定存在某一时刻，使得堆中存在元素（实际值+估计值，并且严格 ≤ 最优解）小于当前路径得到到终点的距离，此时会回溯到前面更（最）优解继续搜索。

2.2 实验流程图


开始
把S放入OPEN表，记f=h
OPEN=NIL?
是
失败
否
选取OPEN表上未设置过的具有最小f值的节点BESTNODE,放入CLOSED表中
BESTNODE是目标节点?
是
成功
否
扩展BESTNODE，产生后续节点SUCCESSOR
A*子过程

3 实验环境

3.1 系统环境

```
Darwin bogon 21.3.0 Darwin Kernel Version 21.3.0
macOS 12.2
```

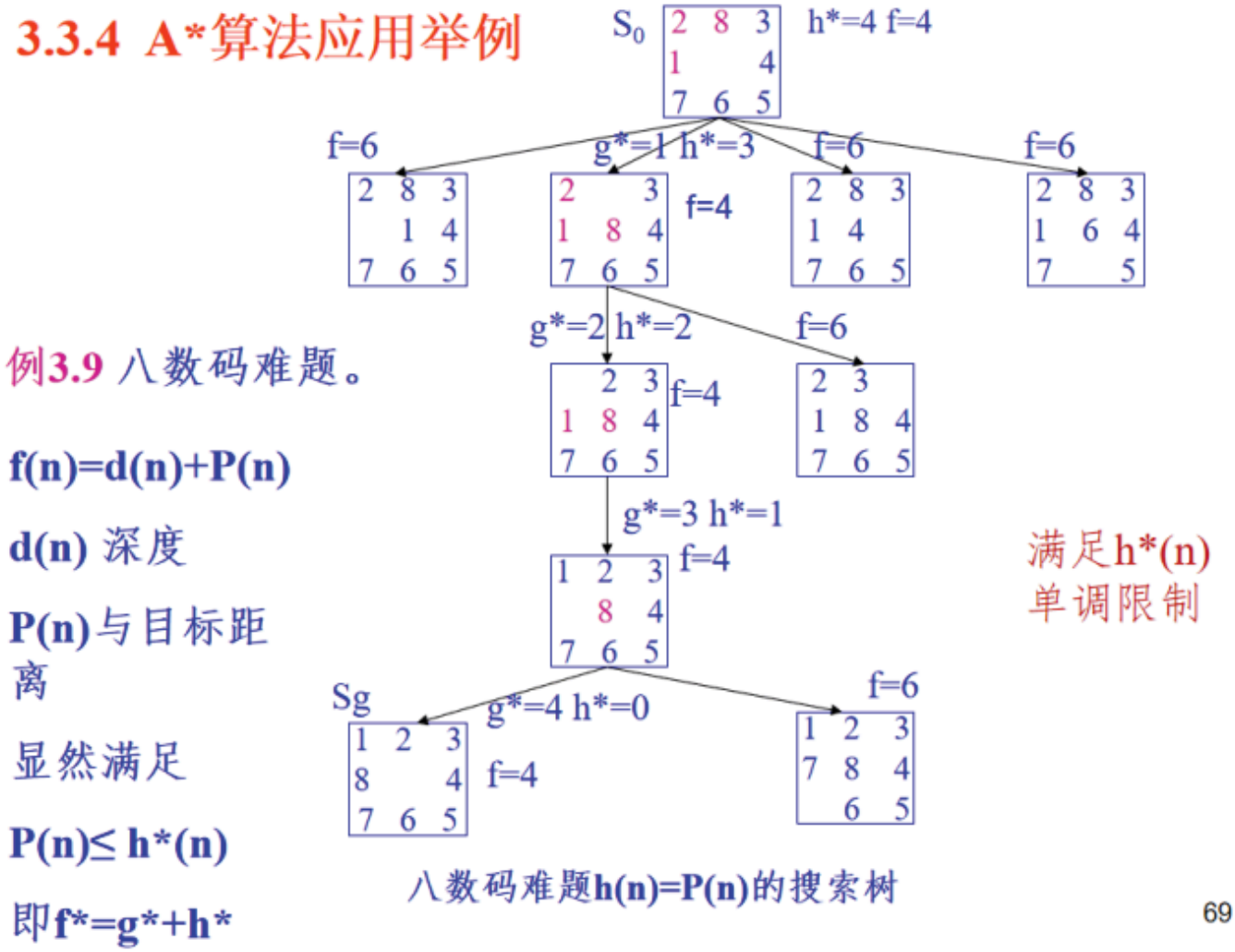
3.2 实现语言

```
C++
```

3.3 编译环境

```
Version: Apple clang version 13.0.0 (clang-1300.0.29.30)
g++ -std=c++17 -O2 -Wsign-compare -DONLINE_JUDGE -Wc++11-extensions -
Werror=return-type
```

4 实验样例



4.1 sample 1

4.1.1 in

```
2 8 3 1 0 4 7 6 5
1 2 3 8 0 4 7 6 5
```

4.1.2 out

```
Operation(s):
Step #1: Up
Step #2: Left
Step #3: Down
Step #4: Right
```

4.2 sample 2

4.2.1 in

```
2 5 4 3 0 7 1 8 6
1 2 3 8 0 4 7 6 5
```

4.2.2 out

```
No Solution...
```

4.3 sample 3

4.3.1 in

```
2 3 4 1 5 0 7 6 8
1 2 3 4 5 6 7 8 0
```

4.3.2 out

```
Operation(s):
Step #1: Left
Step #2: Down
Step #3: Right
```

```
Step #4: Up
Step #5: Up
Step #6: Left
Step #7: Left
Step #8: Down
Step #9: Down
Step #10: Right
Step #11: Up
Step #12: Right
Step #13: Down
Step #14: Left
Step #15: Left
Step #16: Up
Step #17: Right
Step #18: Right
Step #19: Down
```

4.4 sample 4

4.4.1 in

```
6 4 7 8 5 0 3 2 1
1 2 3 4 5 6 7 8 0
```

4.4.2 out

```
Operation(s):
Step #1: Down
Step #2: Left
Step #3: Left
```

```
Step #4: Up
Step #5: Right
Step #6: Up
Step #7: Left
Step #8: Down
Step #9: Right
Step #10: Right
Step #11: Up
Step #12: Left
Step #13: Down
Step #14: Down
Step #15: Left
Step #16: Up
Step #17: Up
Step #18: Right
Step #19: Down
Step #20: Right
Step #21: Down
Step #22: Left
Step #23: Up
Step #24: Right
Step #25: Up
Step #26: Left
Step #27: Down
Step #28: Left
Step #29: Down
Step #30: Right
Step #31: Right
```

4.5 sample 5

4.5.1 in

```
1 2 3 4 5 6 7 8 0
1 2 3 4 5 6 7 8 0
```

4.5.2 out

```
Operation(s):
No need to move!
```

5 代码实现

5.1 接口&类声明

```
class Solution {
 public:
  Solution(vector<int>&, vector<int>&);
  int f(vector<int>&);
  void Bfs(vector<int>&);
  void getAnswer();

 private:
  vector<int> start;
  vector<int> finish;
  vector<int> position;
  map<vector<int>, pair<vector<int>, string>> prev;
};

static constexpr int N = 9;

Solution::Solution(vector<int>& a, vector<int>& b) {
  start.resize(N);
  finish.resize(N);
  position.resize(N);
  start = a;
  finish = b;
  for (int i = 0; i < N; i++) {
    position[finish[i]] = i;
  }
}
```

5.2 接口实现

```
int Solution::f(vector<int>& status) {
  int ans = 0;
  for (int i = 0; i < N; i++) {
    if (status[i] != 0) {
      int cc = position[status[i]]; // position of each element in `target`
      ans += (int) (abs(i / 3 - cc / 3) + abs(i % 3 - cc % 3));
    }
  }
  return ans;
}

void Solution::Bfs(vector<int>& start) {
  priority_queue<pair<int, vector<int>>, vector<pair<int, vector<int>>>,
greater<pair<int, vector<int>>>> s;
  s.emplace(f(start), start);
  map<vector<int>, int> dist;
  dist[start] = 0;
  array<int, 4> dx{-1, 0, 1, 0};
  array<int, 4> dy{0, 1, 0, -1};
  array<string, 4> ops{"Up", "Right", "Down", "Left"};
  while (!s.empty()) {
    auto t = s.top();
    s.pop();
    if (t.second == finish) return;
    vector<int> now(t.second);
    vector<int> old(t.second);
    int zero = 0;
```

```
    for (int i = 0; i < N; i++) {
      if (now[i] == 0) {
        zero = i;
        break;
      }
    }
    int x = zero / 3;
    int y = zero % 3;
    int dd = dist[now];
    for (int i = 0; i < 4; i++) {
      now = old;
      int xx = x + dx[i];
      int yy = y + dy[i];
      if (0 <= xx && xx < 3 && 0 <= yy && yy < 3) {
        swap(now[x * 3 + y], now[xx * 3 + yy]);
        if (dist.count(now) == 0 || dd + 1 < dist[now]) {
          dist[now] = dd + 1;
          prev[now] = make_pair(old, ops[i]);
          s.emplace(dist[now] + f(now), now);
        }
      }
    }
  }
}

void Solution::getAnswer() {
  if (prev.size() == 0) {
    cout << "No need to move!" << endl;
    return;
```

```
  }
  vector<string> ans;
  while (finish != start) {
    ans.push_back(prev[finish].second);
    finish = prev[finish].first;
  }
  reverse(ans.begin(), ans.end());
  cout << "Operation(s):" << endl;
  for (int i = 0; i < (int) ans.size(); i++) {
    cout << "Step #" << i + 1 << ": " << ans[i] << endl;
  }
}
```

5.3 main

```
int main() {
  vector<int> start(N);
  vector<int> finish(N);
  cout << "Enter start state:" << endl;
  for (int i = 0; i < N; i++) {
    cin >> start[i];
  }
  cout << "Enter target state:" << endl;
  for (int i = 0; i < N; i++) {
    cin >> finish[i];
  }
  int rvs1 = 0;
  for (int i = 0; i < N; i++) {
    for (int j = i; j < N; j++) {
      if (start[i] != 0 && start[j] != 0 && start[j] < start[i]) {
```

```
        rvs1 += 1;
      }
    }
  }
  int rvs2 = 0;
  for (int i = 0; i < N; i++) {
    for (int j = i; j < N; j++) {
      if (finish[i] != 0 && finish[j] != 0 && finish[j] < finish[i]) {
        rvs2 += 1;
      }
    }
  }
  if (rvs1 % 2 != rvs2 % 2) {
    cout << "No Solution..." << endl;
    return 0;
  }
  Solution sol(start, finish);
  sol.Bfs(start);
  sol.getAnswer();
  cerr << "running time: " << clock() / 1000 << " ms" << endl;
  return 0;
}
```

6 收获与感悟

本次试验让我完全了解了A*算法的使用场景及使用方法，实验前通过搜集资料完成知识储备与代码能力，实验后又通过OJ找到相关题目进行知识点梳理和熟练。