中山大学计算机学院

人工智能

本科生实验报告

（2023 学年春季学期）

课程名称：Artificial Intelligence

教学班级	信计系统班	专业（方向）	信息与计算科学
学号	22336059	姓名	董胜凡

一、 实验题目

A* & IDA*

二、 实验内容

在本次实验中，针对 Python 代码实现的 A*算法中出现的内存溢出问题，我试图重新写一份 C++代码，在内存中只存储状态的指针，这样就能大幅节省内存。同时在 C++代码中使用编译优化（-O2），大大加快了求解速度。具体的比较见实验结果展示阶段。

结果对 PPT 中测试用例 3 和 4 仍然超出内存，在此记录下我的尝试。优点是运行速度比 Python 代码加快近 100 倍。因此得出结论，内存花费过大的测试用例只能由 IDA*来完成。IDA*成功完成所有测试用例，但用时较长。

在 C++代码中针对编译器参数采取了编译优化，因此在测试 C++程序时，请直接运行可执行文件 “C++.exe”。

1. 算法原理

实现结构

定义一个 Node 类，类中的成员变量为 state、parent、move、depth。其中 state 类型为 `np.ndarray` ，保存节点棋盘的状态；parent 的类型为 `Node` ，保存该节点的父节点，目的是在找到目标解时向上回溯，从而打印出最优解的路径；move 是一个 `int` 类，用于记录该节点是将空格和哪个数字对调产生的；depth 记录了节点深度。

A*算法

A*算法是一种启发式搜索算法，用于在图中找到从初始节点（初始状态）到目标节点（目标状态）的最短路径。它结合了最好优先搜索（Best-First Search）和 Dijkstra 算法的特点，通过评估函数 f(n) = g(n) + h(n)来选择路径，其中 g(n)是从初始节点到当前节点的实际代价，h(n)是启发式函数估计的从当前节点到目标节点的最小代价，这里选择节点的曼哈顿距离作为启发值。

为了实现该算法，维护一个优先队列 `open_list` ，保存所有可扩展的节点，利用其堆排序的特性，快速的弹出具有最小 `f(n)` 值的节点；同时维护一个集合 `closed_set` ，保存所有已经扩展过的节点，利用环检测的方法加快速度。该算法能够保证给出最优解，同时尽可能加快求解速度。

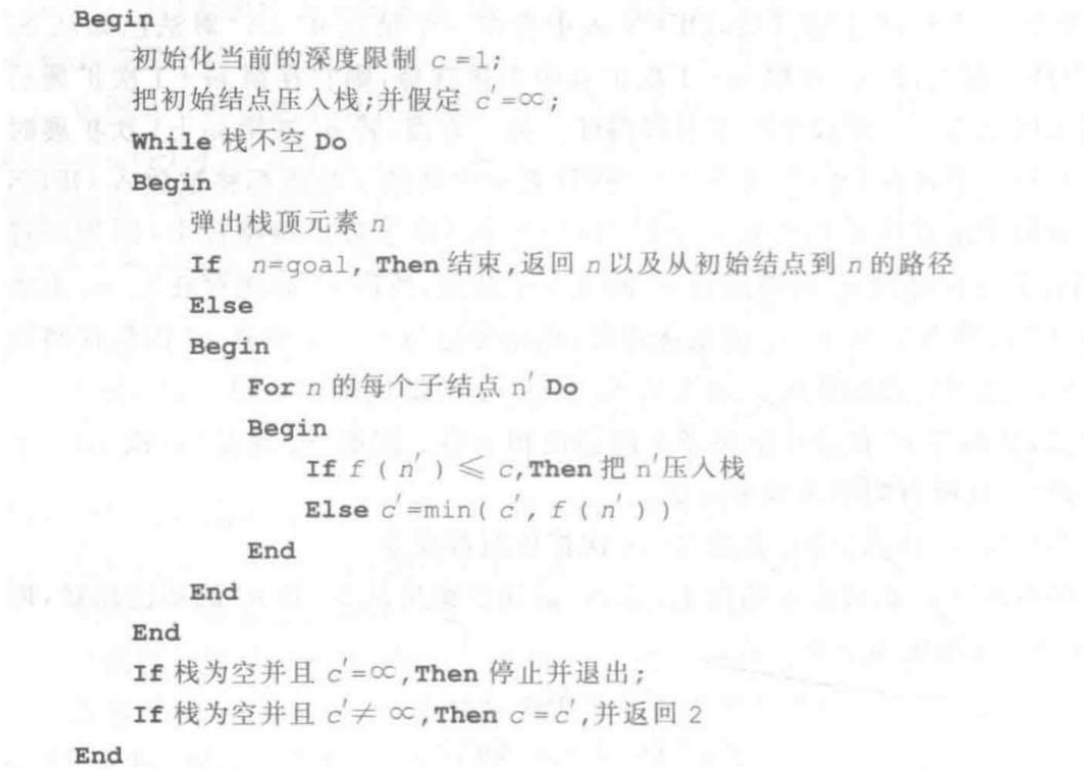
IDA*算法

迭代加深 A*（IDA*）算法是一种结合了迭代加深搜索（Iterative Deepening Search, IDS）

和 A 搜索算法特点的启发式搜索算法。它旨在解决如 15-puzzle 这样的路径寻找问题，同时减少内存消耗并尽可能找到最优解。用节点的评价值作为深度限制，每次循环如果遇到超出限制的节点，那么在下次循环时更新深度限制；如果所有节点都不超过限制，那么认为寻找完毕，找不到目标节点；如果找到目标节点则返回最终状态。利用节点的父指针向上回溯找到路径

2. 伪代码

IDA_star




```
def a_star(initial_state : np.ndarray, goal_state : np.ndarray) -> Node:
#维护优先队列  open_list  和  已经扩展的节点集合  closed_set
procedure a_star(initial_state, goal_state) returns Node:
    初始化优先队列：initial_list，并将开始节点加入
    初始化环检测表：closed_set
    while open_list is not empty
    begin
        current_node = pop_node_with_lowest_f(open_list)
        if 找到最终解
            return current_node
        current_node 加入 closed_set
        for each move, new_state in get_moves(current_node.state)
        begin
            if new_state in closed_set
                continue
            new_node := Node(new_state, current_node, move, current_node.depth + 1)
```

```
                add new_node to open_list
            end
        end
        return None
end procedure
```

3. 关键代码展示（带注释）

```
def get_moves(current_node : Node) -> list:
    #传入目前节点状态，返回移动后的状态列表（上下左右），跳过重复移动的状态
    state = current_node.state
    moves = []
    empty_index = -1
    for i in range(16):
        if state[i] == 0:
            empty_index = i
            break
    row, col = divmod(empty_index, 4)                    #找到移动前空格所在的行和列
    directions = [(-1, 0), (1, 0), (0, -1), (0, 1)]      # Up, Down, Left, Right
    for dr, dc in directions:
        new_row, new_col = row + dr, col + dc            #移动后空格所在的行和列
        if 0 <= new_row < 4 and 0 <= new_col < 4:
            new_state = copy.deepcopy(state)
            swapedNum = state[new_row * 4 + new_col]
            if swapedNum == current_node.move:           #如果移动步数重复，跳过该状态
                continue
            new_state[new_row * 4 + new_col] = 0
            new_state[row * 4 + col] = swapedNum
            moves.append((swapedNum, new_state))
    return moves
def a_star(initial_state : np.ndarray, goal_state : np.ndarray) -> Node:
    #维护优先队列  open_list  和  已经扩展的节点集合  closed_set
    initial_node = Node(initial_state, None, -1, 0)
    open_list = [initial_node]
    closed_set = set()

    while open_list:
        current_node = heapq.heappop(open_list)
        if np.array_equal(current_node.state, goal_state):    #检查是否到达终点
            return current_node
        closed_set.add(tuple(current_node.state))             #扩展过的节点加入closed_set
        for move, new_state in get_moves(current_node.state):
            if tuple(new_state) in closed_set:                #环检测
                continue
            new_node = Node(new_state, current_node, move, current_node.depth + 1)
```

```
                heapq.heappush(open_list, new_node)

            return None
def IDA_star(initial_state : np.ndarray, goal_state : np.ndarray) -> Node:
    #每次迭代更新cost_limit的值
    cost_limit = 1
    while cost_limit != 0:
        best_cost = float('inf')
        open_stack = [Node(initial_state, None, -1, 0)]
        while open_stack:
            current_node = open_stack.pop()
            if current_node.manhattan_distance == 0:                #找到最终解
                return current_node
            for move, new_state in get_moves(current_node):
                new_node = Node(new_state, current_node, move, current_node.depth + 1)
                if new_node.manhattan_distance + new_node.depth <= cost_limit:                #未越界
的情形
                    open_stack.append(new_node)
                else:
                    best_cost = min(best_cost, new_node.manhattan_distance + new_node.depth)
#越界的情形
        if open_stack == [] and best_cost == float('inf'):
            return None
        if open_stack == [] and best_cost != float('inf'):
            cost_limit = best_cost
```

4. 创新点&优化（如果有）

1、在 get_moves 函数中，利用节点的结构保存节点是移动哪一个数字产生的，这样在获取新的状态时，可以跳过重复移动的状态，加快求解速度。比如说状态 1 由空格和数字 9 交换产生，那么在产生状态 1 的子状态时，跳过将空格和数字 9 再次交换的重复情形。

2、在 A*算法中，利用优先队列快速弹出具有最佳评价值的状态节点，加快速度；加入环检测，加快速度。该算法保证最优解。

3、在 IDA*算法中，实现两个版本，第一个版本有环检测，避免扩展相同状态的节点，但有可能因此跳过最优解，加快速度；第二个版本无环检测，对于相同状态的节点采用相同的逻辑进行深度优先搜索，保证求得最优解。

4、相比于传统的曼哈顿距离作为启发式函数，这里选择**曼哈顿距离加线性冲突法**来作为启发式函数，在保证可采纳性的同时增大启发值能加快求解速度，参考【人工智能】A*算法和IDA*算法求解 15-puzzle 问题（大量优化，能优化的基本都优化了）_十五数码问题 a*-CSDN博客

三、 实验结果及分析

1、实验结果展示示例（可图可表可文字，尽量可视化）

A*算法（C++）,快速完成，仅展示 PPT 中比较难的测试用例

（1） Python 运行 100seconds，C++运行 1second

（2） Python 运行 300seconds，C++运行 3seconds

```
Please input filename:
choice  1.txt  2.txt  3.txt  4.txt  ppt1.txt  ppt2.txt  ppt3.txt  ppt4.txt
ppt1.txt
Test using A_star for ppt1.txt
Running time=  1.30216 seconds
Steps = 49
Action sequences:
6 10 9 4 14 9 4 1 10 4 1 3 2 14 9 1 3 2 5 11 8 6 4 3 2 5 13 12 14 13 12 7 11 12 7 14 13 9 5 10 6 8 12 7 10 6 7 11 15

C:\code\chem\x64\Release\chem.exe (进程 12996)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

```
Please input filename:
choice  1.txt  2.txt  3.txt  4.txt  ppt1.txt  ppt2.txt  ppt3.txt  ppt4.txt
ppt2.txt
Test using A_star for ppt2.txt
Running time=  3.77897 seconds
Steps = 48
Action sequences:
9 12 13 5 1 9 7 11 2 4 12 13 9 7 11 2 15 3 2 15 4 11 15 8 14 1 5 9 13 15 7 14 10 6 1 5 9 13 14 10 6 2 3 4 8 7 11 12

C:\code\chem\x64\Release\chem.exe (进程 3676)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

最简单的测试用例 1 和 3 只需 6ms 和 1ms

```
Please input filename:
choice  1.txt  2.txt  3.txt  4.txt  ppt1.txt  ppt2.txt  ppt3.txt  ppt4.txt
1.txt
Test using A_star for 1.txt
Running time=  0.006181 seconds
Steps = 40
Action sequences:
6 11 4 14 5 8 9 5 8 3 2 6 14 8 15 7 10 4 8 15 3 2 6 14 15 10 7 3 2 6 14 15 10 7 3 2 6 10 11 12

C:\code\chem\x64\Release\chem.exe (进程 2316)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

```
Please input filename:
choice  1.txt  2.txt  3.txt  4.txt  ppt1.txt  ppt2.txt  ppt3.txt  ppt4.txt
3.txt
Test using A_star for 3.txt
Running time=  0.001669 seconds
Steps = 40
Action sequences:
11 14 9 1 12 4 1 8 2 13 15 12 8 2 3 11 14 9 6 1 2 3 11 7 13 11 7 14 10 13 14 10 9 5 1 2 3 7 11 15

C:\code\chem\x64\Release\chem.exe (进程 19884)已退出，代码为 0。
要在调试停止时自动关闭控制台，请启用“工具”->“选项”->“调试”->“调试停止时自动关闭控制台”。
按任意键关闭此窗口. . .
```

A*算法（Python），其中 PPT 中第三和第四个超出内存

```
Test using A_star for 1.txt:
running time = 0.6231648921966553 seconds
steps = 40
initial state:
1 15 7 10
9 14 4 11
8 5 0 6
13 3 2 12
Action sequences:
6 11 4 14 5 8 9 5 8 3 2 6 14 8 15 7 10 4 8 15 3 2 6 14 15 10 7 3 2 6 14 15 10 7 3 2 6 10 11 12
moved num: 6
1 15 7 10
9 14 4 11
8 5 6 0
13 3 2 12
---------------
moved num: 11
1 15 7 10
9 14 4 0
8 5 6 11
13 3 2 12
---------------
```

```
Test using A_star for 2.txt:
running time = 0.7198882102966309 seconds
steps = 40
initial state:
1 7 8 10
6 9 15 14
13 3 0 4
11 5 12 2
Action sequences:
15 14 4 2 12 15 14 8 10 4 8 9 3 5 11 13 5 14 2 12 15 11 14 2 9 10 7 3 2 9 10 7 3 2 6 5 9 10 11 15
moved num: 15
1 7 8 10
6 9 0 14
13 3 15 4
11 5 12 2
---------------
moved num: 14
1 7 8 10
6 9 14 0
13 3 15 4
11 5 12 2
---------------
```

```
Test using A_star for 3.txt:
running time = 0.14103460311889648 seconds
steps = 40
initial state:
5 6 4 12
11 14 9 1
0 3 8 15
10 7 2 13
Action sequences:
11 14 9 1 12 4 1 8 2 13 15 12 8 2 3 11 14 9 6 1 2 3 11 7 13 11 7 14 10 13 14 10 9 5 1 2 3 7 11 15
moved num: 11
5 6 4 12
0 14 9 1
11 3 8 15
10 7 2 13
---------------
```

```
Test using A_star for 4.txt:
running time = 2.9634463787078857 seconds
steps = 40
initial state:
14 2 8 1
7 10 4 0
6 15 11 5
9 3 13 12
Action sequences:
1 8 4 1 5 11 15 3 13 15 3 10 1 5 8 4 2 1 7 14 1 7 5 3 10 6 14 5 7 2 3 7 6 14 9 13 14 10 11 12
moved num: 1
14 2 8 0
7 10 4 1
6 15 11 5
9 3 13 12
---------------
```

```
Test using A_star for ppt1.txt:
running time = 136.2001609802246 seconds
steps = 49
initial state:
14 10 6 0
4 9 1 8
2 3 5 11
12 13 7 15
Action sequences:
6 10 9 4 14 9 4 1 10 4 1 3 2 14 9 1 3 2 13 12 14 13 5 11 8 6 4 3 2 5 12 7 11 12 7 14 13 9 5 10 6 8 12 7 10 6 7 11 15
moved num: 6
14 10 0 6
4 9 1 8
2 3 5 11
12 13 7 15
---------------
```

```
Test using A_star for ppt2.txt:
running time = 328.9033064842224 seconds
steps = 48
initial state:
6 10 3 15
14 8 7 11
5 1 0 2
13 12 9 4
Action sequences:
9 12 13 5 1 9 7 11 2 4 12 13 9 7 11 2 15 3 2 15 4 11 15 8 14 1 5 9 13 15 7 14 10 6 1 5 9 13 14 10 6 2 3 4 8 7 11 12
moved num: 9
6 10 3 15
14 8 7 11
5 1 9 2
13 12 0 4
---------------
```

IDA*算法

```
Test using IDA_star_without_detection for 4.txt:
running time = 6.851652145385742 seconds
steps = 40
initial state:
14 2 8 1
7 10 4 0
6 15 11 5
9 3 13 12
Action sequences:
1 8 4 1 5 11 15 3 13 15 3 10 1 5 8 4 2 1 5 3 10 5 7 14 1 2 3 7 14 6 5 14 6 5 9 13 14 10 11 12
moved num: 1
14 2 8 0
7 10 4 1
6 15 11 5
9 3 13 12
----------------
```

IDA*算法测试用例结果（PPT3 和 PPT4）

```
Test using IDA_star_without_detection for ppt3.txt:
running time = 3974.365383922004 seconds
steps = 56
initial state:
11 3 1 7
4 6 8 2
15 9 10 13
14 12 5 0
Action sequences:
13 10 8 6 9 12 5 13 10 8 12 15 14 5 13 12 15 14 5 13 14 3 4 11 3 1 6 3 11 3 1 6 4 2 8 10 12 15 10 8 7 4 2 11 3 5 9 10 11 3 6 2 3 7
 8 12
moved num: 13
11 3 1 7
4 6 8 2
15 9 10 0
14 12 5 13
---------------
```

```
Test using IDA_star_without_detection for ppt4.txt:
running time = 13765.23746389273 seconds
steps = 62
initial state:
0 5 15 14
7 9 6 13
1 2 12 10
8 11 4 3
Action sequences:
7 9 2 1 9 2 5 7 2 5 1 11 8 9 5 1 6 12 10 3 4 8 11 10 12 13 3 4 8 12 13 15 14 3 4 8 12 13 15 14 7 2 1 5 10 11 13 15 14 7 3 4 8 12 1
5 14 11 10 9 13 14 15
moved num: 7
7 5 15 14
0 9 6 13
1 2 12 10
8 11 4 3
---------------
```