湖北大学
HUBEI UNIVERSITY

实验一　A*算法

本周实验内容

- 使用A*算法解决8数码难题
- 完成并提交实验报告（实验报告模板可在学习通下载）

实验要求

- 编写代码使用A* 算法求解八数码难题，估价函数f(x)=g(x)+h(x)，其中g(x)为从初始状态s到当前状态的层数（结点的深度），g(s)=0；h(x)为启发式函数。
- 要求：求出以下三种启发式函数对应的求解效率，其中：

求解效率 = 最佳路径节点数 / 算法求得的路径节点数

三种启发式函数分别为：

1. h(x)=0，即宽度优先搜索；
2. h(x) = 错误位置和，即处在错误位置的数码的个数；
3. h(x) = 曼哈顿距离和。

实验要求

- **初始状态：**

[[2, 8, 3],
[1, 6, 4],
[7, 0, 5]]

- **目标状态：**

[[1, 2, 3],
[8, 0, 4],
[7, 6, 5]]

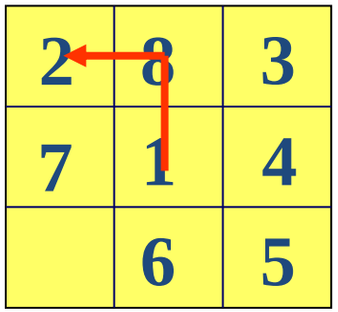
- **打印：**

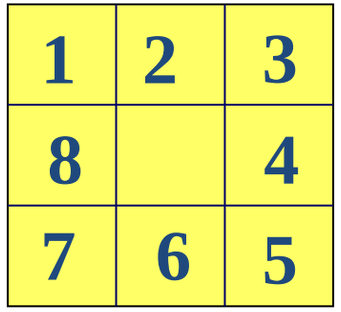
```
Step:  1
 ->
[[2 8 3]
 [1 6 4]
 [7 0 5]]
******
Step:  2
up ->
[[2 8 3]
 [1 0 4]
 [7 6 5]]
******
Step:  3
up ->
[[2 0 3]
 [1 8 4]
 [7 6 5]]
******
```

```
h = 2 ----------------
Optimal step is 6
Travel step is  6
Solving efficiency is 100.00%
```

最佳优先搜索

- 使用最佳优先搜索解决8数码难题（对应教材第3.3.2节）





✓ 曼哈顿距离 (Manhattan Distance)：两点之间水平距离和垂直距离之和

✓ 8数码难题的估价函数：数码和目标位置之间的曼哈顿距离

最佳优先搜索

曼哈顿距离$W(n)$：两点之间水平距离和垂直距离之和

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

2	8	3
7	1	4
	6	5

$W(n) = 2 + 1 + 1 + 2 = 6$

错误位置和：$1 + 1 + 1 + 1 = 4$

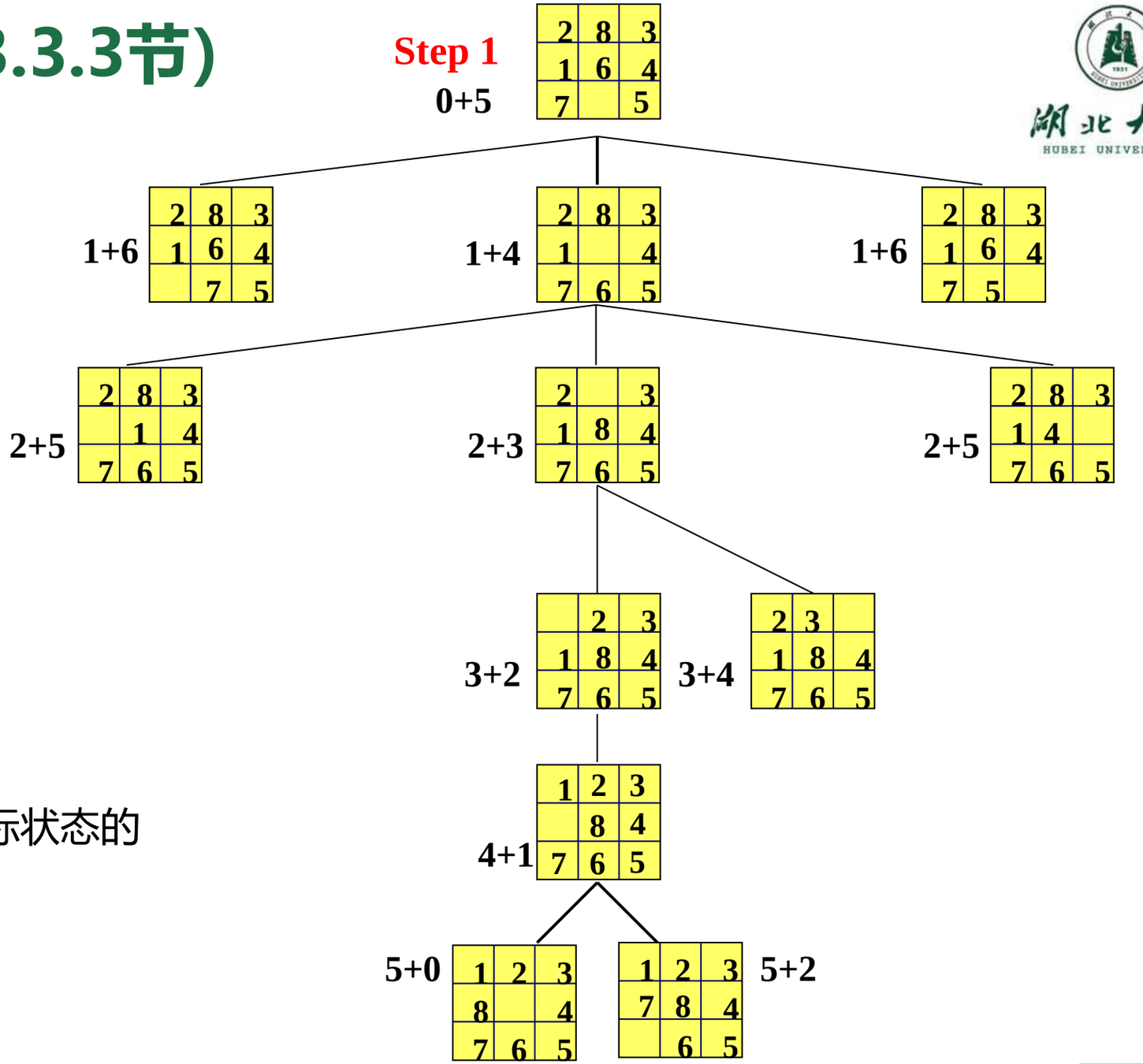
A*搜索（对应教材3.3.3节）

估价函数 $f(n) = d(n) + W(n)$

结点的深度　　曼哈顿距离

估价函数 $f(n) = g(n) + h(n)$

到达结点n的代价　　结点n到达目标状态的估计代价



编程思路

- **宽度优先搜索**

```
# name：结点的名字（ID）
# neighbors：结点的邻居（后继、后裔）
class Node:
    def __init__(self, name: str):
        self.name = name
        self.neighbors = list()
```

编程思路

- **宽度优先搜索**

```
# start：初始状态，target：目标状态
def BFS(start: Node, target: str):
    open = []
    closed = []
    visited = set()

    open.append(start)
    while len(open) > 0:
        cur_node = open.pop(0)  # 取出队列中的第一个状态
        closed.append(cur_node)
        visited.add(cur_node)
        # 找到目标状态后，即可停止搜索
        if cur_node.name == target:
            print('End.\nNode ', cur_node.name, ' is found!')
            return closed
```

编程思路

- **宽度优先搜索**

```
            # 上接while循环
            # 把当前结点的邻居加入待访问结点列表
            for neighbor in cur_node.neighbors:
                if neighbor not in visited:
                    open.append(neighbor)

    print('End.' )
    return closed
```

编程思路

- 难点：类State（状态）的定义

```
class State:
    def __init__(self, matrix, parent=None, direction_flag=None, g=0, h=0):
        self.matrix = matrix    # 状态的二维数组
        self.parent = parent    # 用于回溯，以寻找最优路径

        # 避免“走回头路”
        self.direction = ['up', 'down', 'right', 'left']
        if direction_flag:  # 不能走的方向
            self.direction.remove(direction_flag)

        self.g = g      # g(n) 状态的深度
        self.h = h      # h(n) 启发式函数，0:宽度优先，1：错位，2：曼哈顿
        self.f = self.evaluate()  # 估价函数 f(n)
```

编程思路

```
def A_star_search(start, target):
    open = []                  # 待访问结点列表
    close = []                 # 已访问结点列表
    open.append(start)         # 将初始状态加入待访问结点列表
    path = []

    while len(open) > 0:
        # 选取表中f(n)值最小的状态i放入已访问结点列表
        state_i = open.pop(0)
        close.append(state_i)
        path.append(state_i)

        # 如果i为目标状态
        if state_i.matrix == target:
            # 则回溯寻找最优路径，返回path以及最优路径
```

编程思路

```
# 计算当前结点所有的邻居（后继）
neighbors = state_i.neighbors()

# 判断后继是否已经存在于已访问列表里，避免重复访问
for neighbor in neighbors:
        if not neighbor.is_in_close(close):
            open.append(neighbor)

# 根据f(n)的值对待访问结点列表重新排序
open.sort(key=lambda s: s.getF())
```

结束语

湖北大学
HUBEI UNIVERSITY

谢 谢!