第二章

状态空间表示 & 图搜索策略

状态空间表示（对应教材第2.1节）

- 十五数码难题 (15 puzzle problem)

滑动拼图 / 华容道拼图

状态空间表示

- **十五数码难题 (15 puzzle problem)**

初始状态

11	9	4	15
1	3		12
7	5	8	6
13	2	10	14

如何恢复?

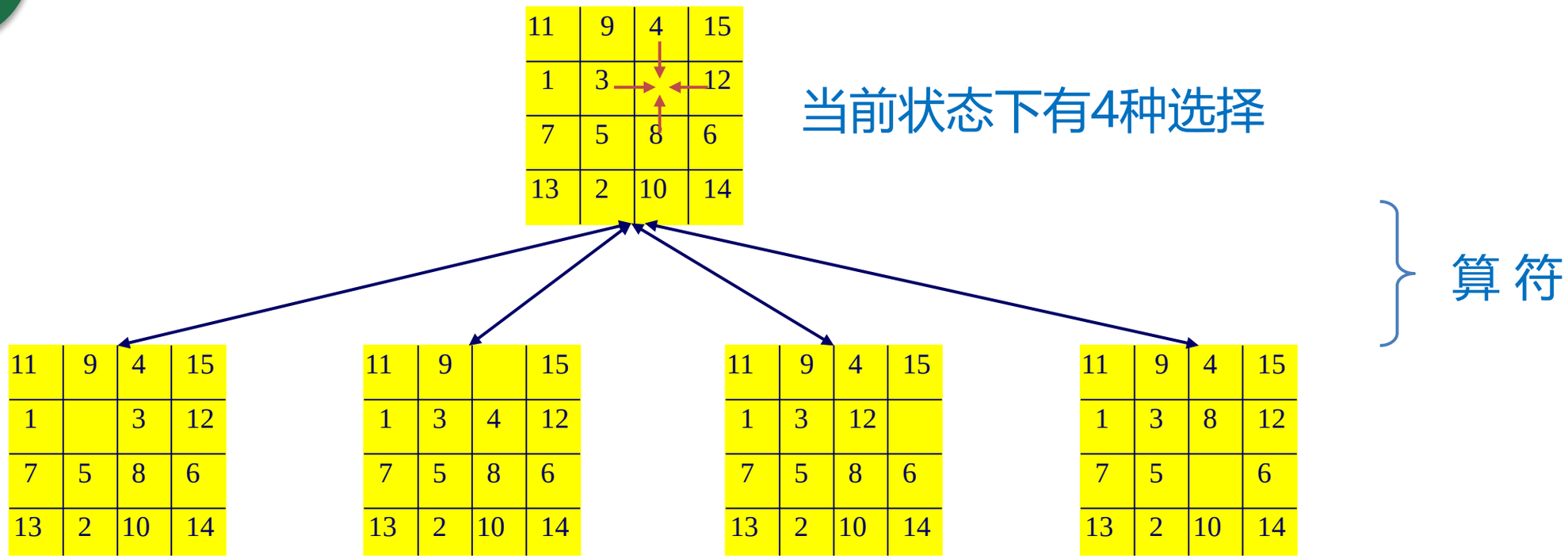
1	2	3	4
5	6	7	8
9	10	11	12
13	14	15	

目标状态

打乱的拼图　　有序的拼图

- **状态 (state)**：描述某类不同事物间的差别而引入的一组最少变量 $Q = [q_1, q_2, \dots, qn]^T$

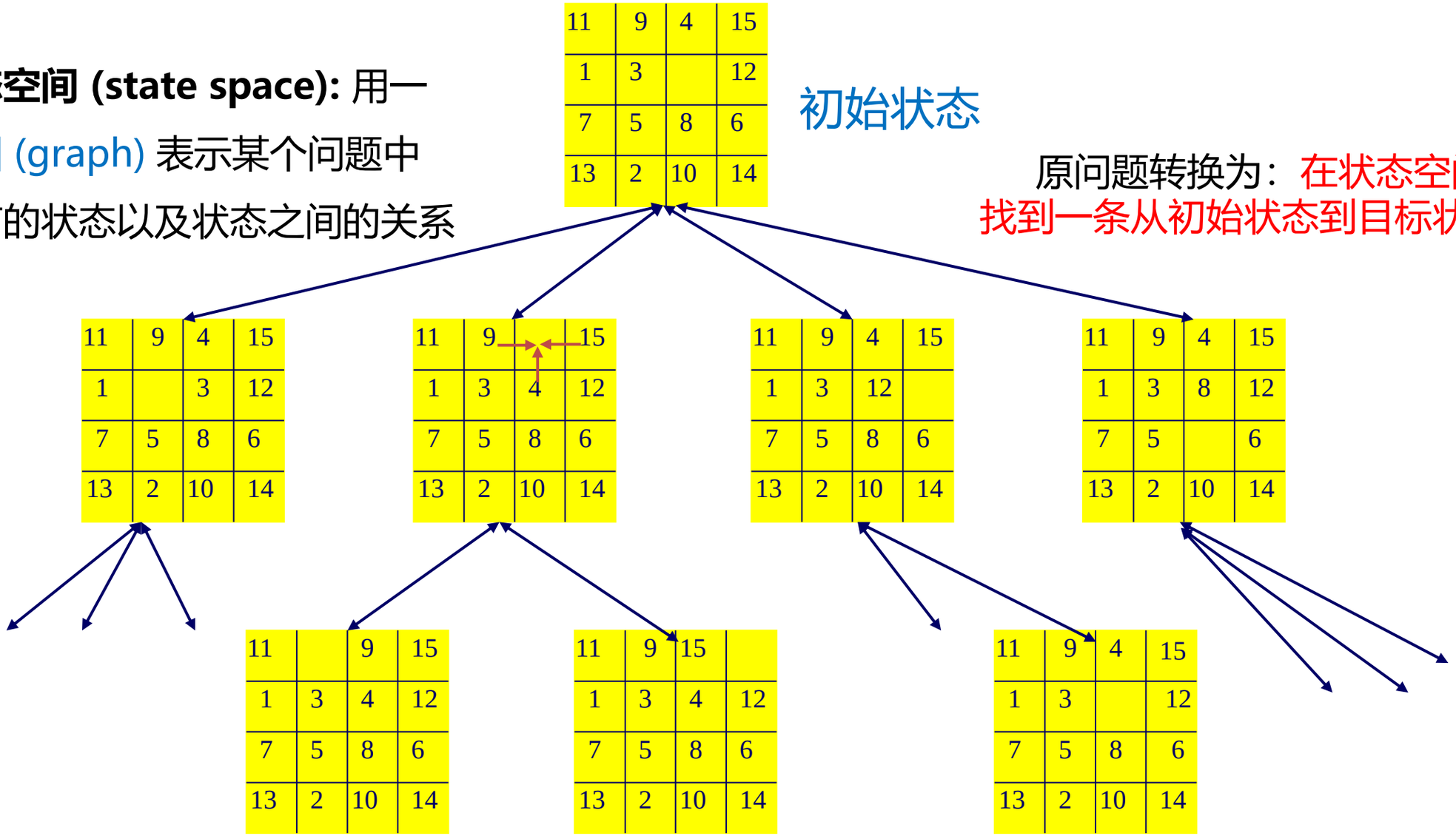
状态空间表示



- **算符 (operator):** 将一个状态转换至另一个状态

状态空间表示

- **状态空间 (state space):** 用一张图 (graph) 表示某个问题中所有的状态以及状态之间的关系



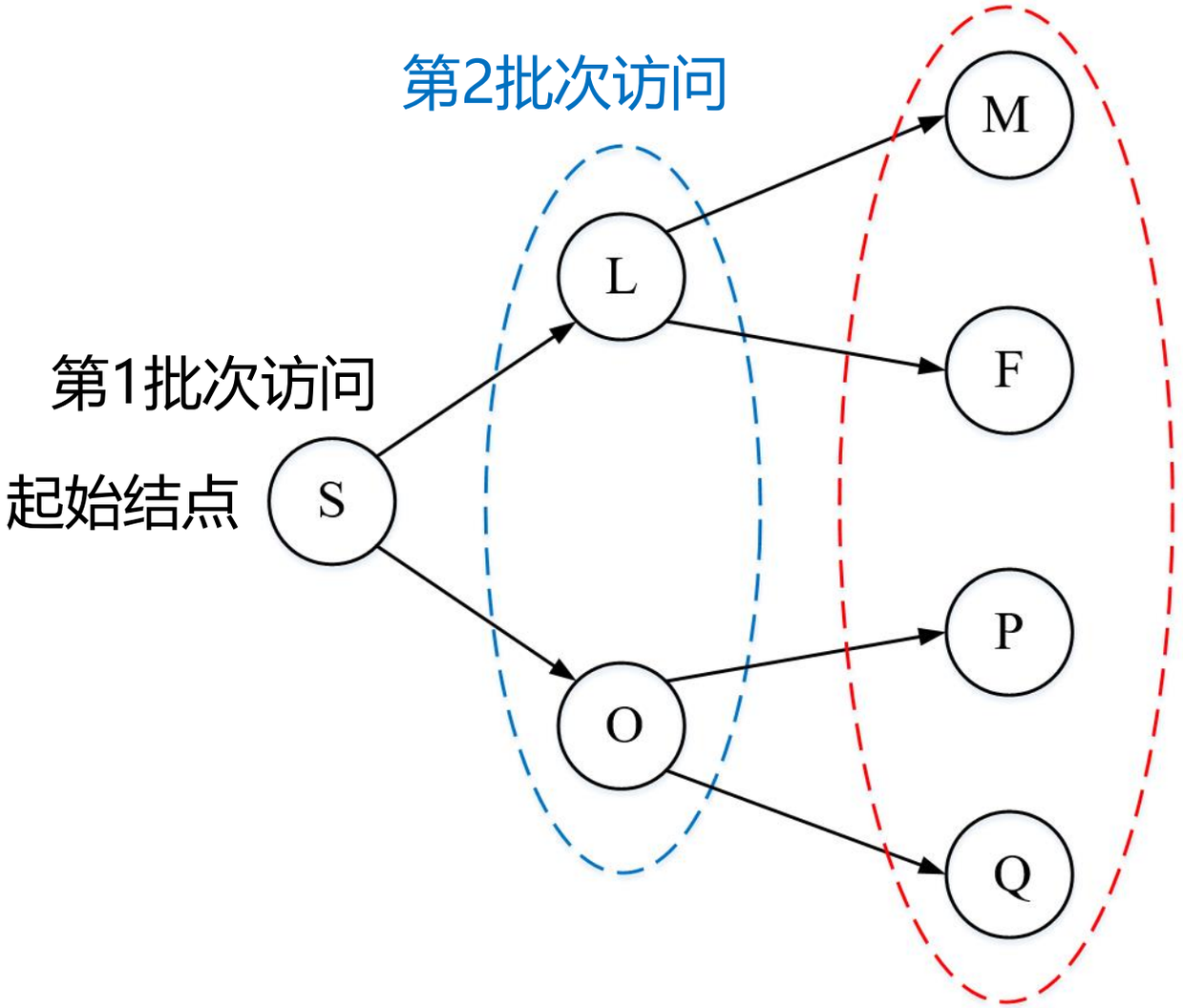
图搜索策略（对应教材第3.1节，3.2节，3.3节）

- **教材第47页，2.1.2节最后一句话：**各种问题都可以用状态空间加以表示，并用状态空间搜索法来求解

- **图的遍历算法 (graph traversal)**
  - ✓ 宽度优先搜索 (breadth-first search, BFS)
  - ✓ 深度优先搜索 (depth-first search, DFS)

宽度优先搜索

- **以教材第85页，图3.3为例**

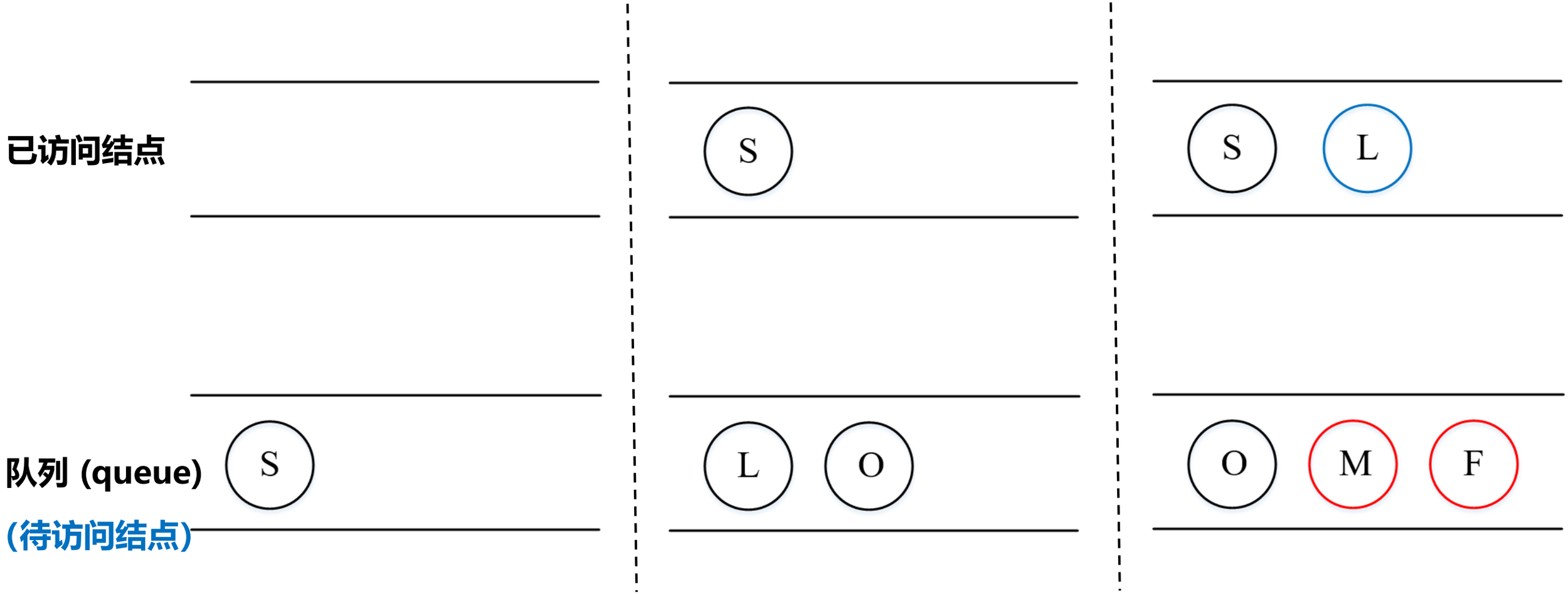



```
def BFS(node){
...
}
```

宽度优先：访问完某个结点n后，将结点n的所有邻居 (neighbor) 加入队列（queue），等待访问。

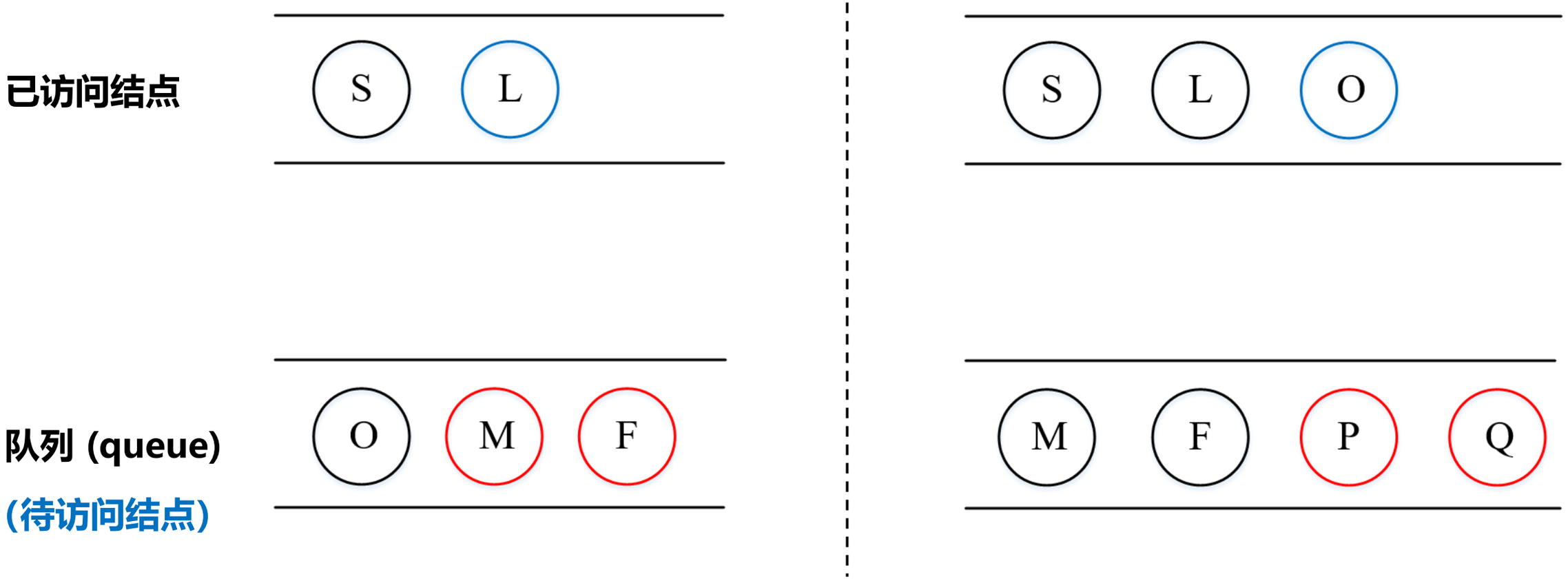
宽度优先搜索

- 如何编程实现?



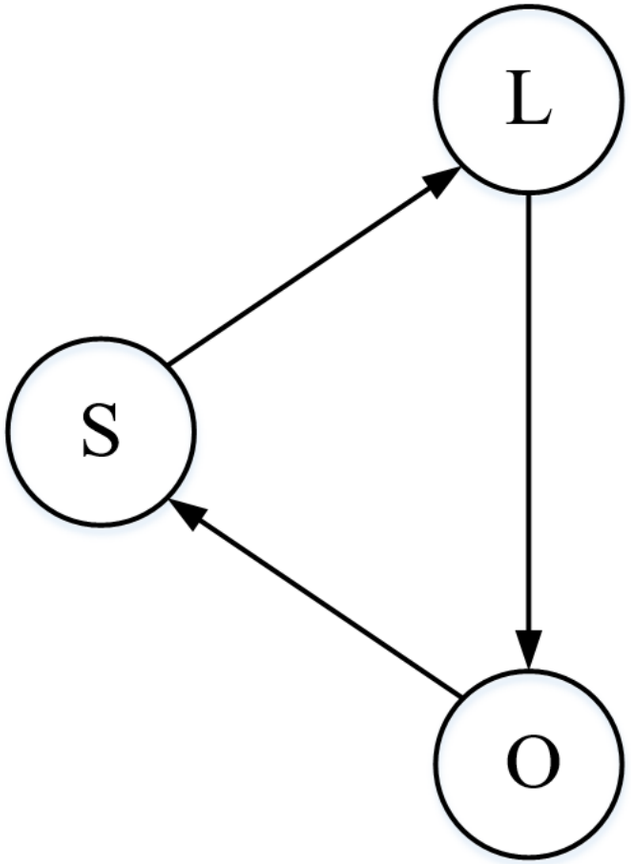
宽度优先搜索

- 如何编程实现?



宽度优先搜索

- **如何避免重复访问？（注意：教材第3.2节给出的算法没有避免重复访问！）**



避免重复：设置一个集合(set)，存储已经访问过的结点；在搜索的过程中，如果发现某个结点已经被访问过，则不将其添加进队列。

宽度优先搜索

- **Python实现**

```
class Node:
    def __init__(self, name: str):
        self.name = name
        self.neighbors = list()


def BFS(start: Node):
    open = []          # 待访问结点列表（队列）
    closed = []        # 已访问结点列表
    visited = set()    # 用于判断某个结点是否已经被访问过
```

宽度优先搜索

- **Python实现**

```
def BFS(start: Node):
    open = []
    closed = []
    visited = set()

    open.append(start)  # 把初始结点加入队列
    while len(open) > 0:
        # 把队列中的第一个结点取出
        # 加入已访问结点列表和集合中
        cur_node = open.pop(0)
        closed.append(cur_node)
        visited.add(cur_node)
        print(cur_node.name, end=' -> ')
```

宽度优先搜索

- **Python实现**

```
def BFS(start: Node):
    open = []
    closed = []
    visited = set()

    open.append(start)
    while len(open) > 0:
        # 把队列中的第一个结点取出
        # 加入已访问结点列表和集合中
        cur_node = open.pop(0)
        closed.append(cur_node)
        visited.add(cur_node)
        print(cur_node.name, end=' -> ')
        # 最后把当前结点的邻居加入待访问结点
        for neighbor in cur_node.neighbors:
            if neighbor not in visited:
                open.append(neighbor)

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

宽度优先搜索

- **Python实现**

```
node_S = Node('S')
node_L = Node('L')
node_O = Node('O')
node_M = Node('M')
node_F = Node('F')
node_P = Node('P')
node_Q = Node('Q')

node_S.neighbors.append(node_L)
node_S.neighbors.append(node_O)

node_L.neighbors.append(node_M)
node_L.neighbors.append(node_F)

node_O.neighbors.append(node_P)
node_O.neighbors.append(node_Q)
```

```
search_result = BFS(node_S)
for node in search_result:
    print(node.name, end=' -> ')
print('End.')
```

运行结果：

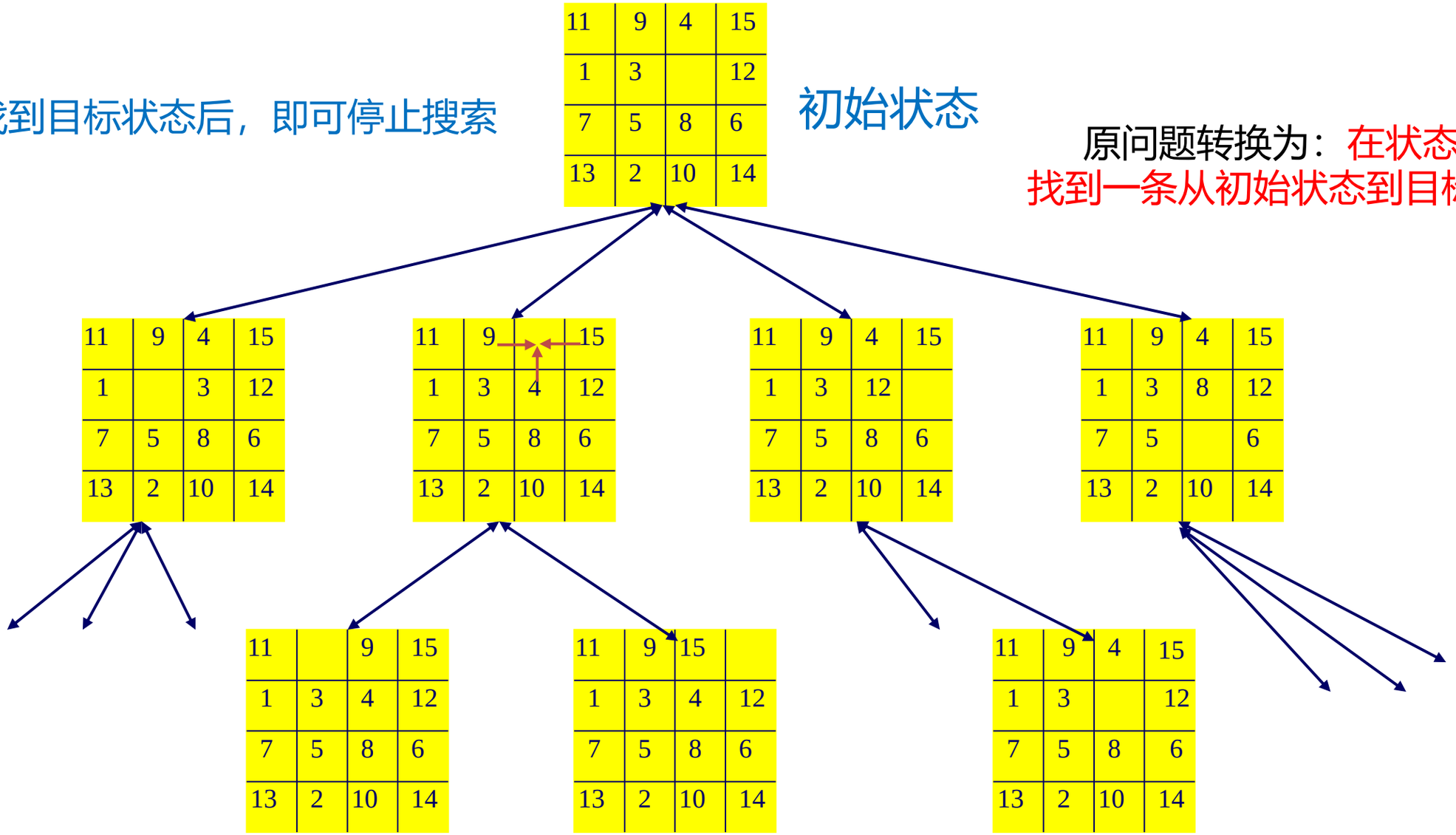
```
D:\Software\anaconda3\envs\matplot\python.exe
S -> L -> O -> M -> F -> P -> Q -> End.
S -> L -> O -> M -> F -> P -> Q -> End.

Process finished with exit code 0
```

状态空间表示

找到目标状态后，即可停止搜索

初始状态

原问题转换为：在状态空间中，
找到一条从初始状态到目标状态的路径



宽度优先搜索

```
# 增加一个target参数，代表目标状态
def BFS(start: Node, target: str):
    open = []
    closed = []
    visited = set()

    open.append(start)
    while len(open) > 0:
        cur_node = open.pop(0)
        closed.append(cur_node)
        visited.add(cur_node)
        print(cur_node.name, end=' ->  ‘)
        # 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
```

宽度优先搜索

- **Python实现**

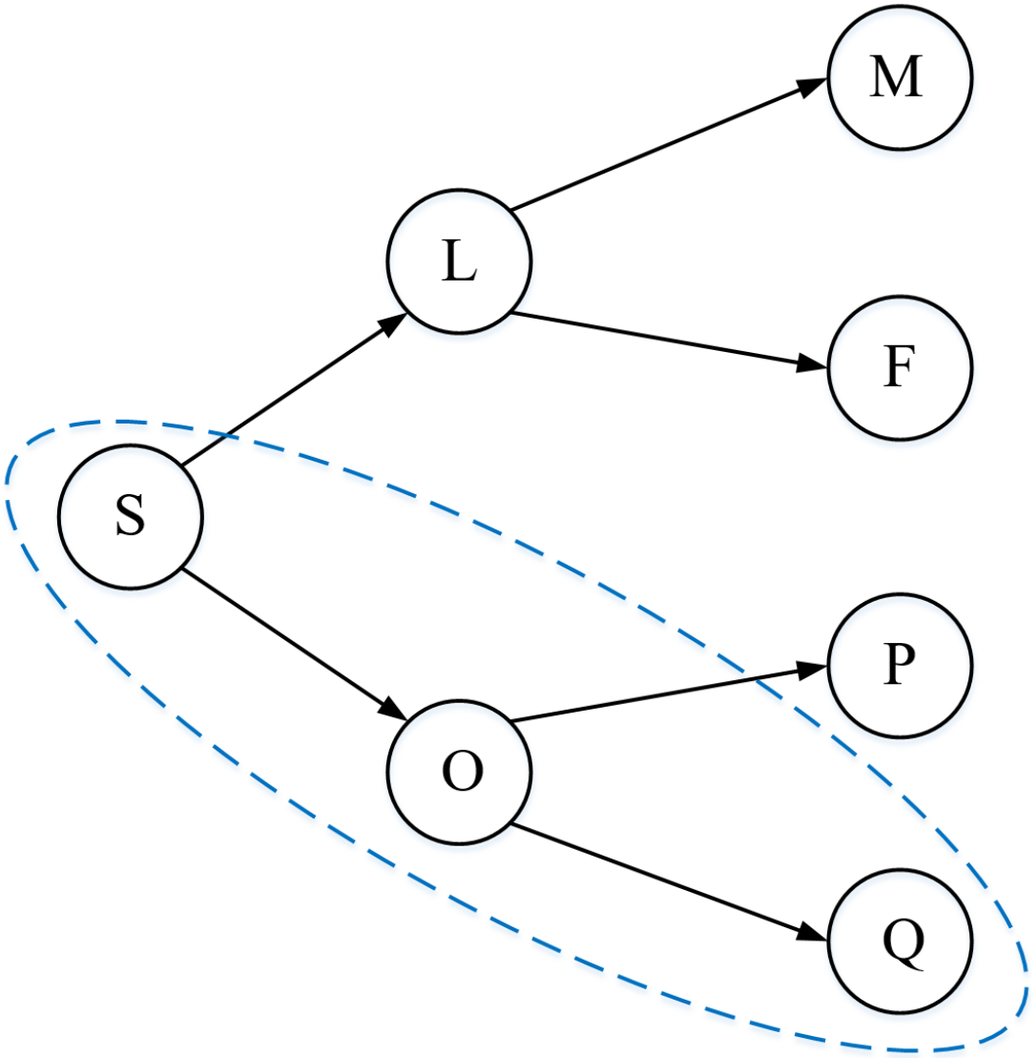
```
# 搜索结点O
search_result = BFS(node_S, 'O')
for node in search_result:
    print(node.name, end=' -> ')
print('End.')
```

运行结果：

```
D:\Software\anaconda3\envs\matplot\python.exe
S -> L -> O -> End.
Node  O  is found!
S -> L -> O -> End.

Process finished with exit code 0
```

深度优先搜索

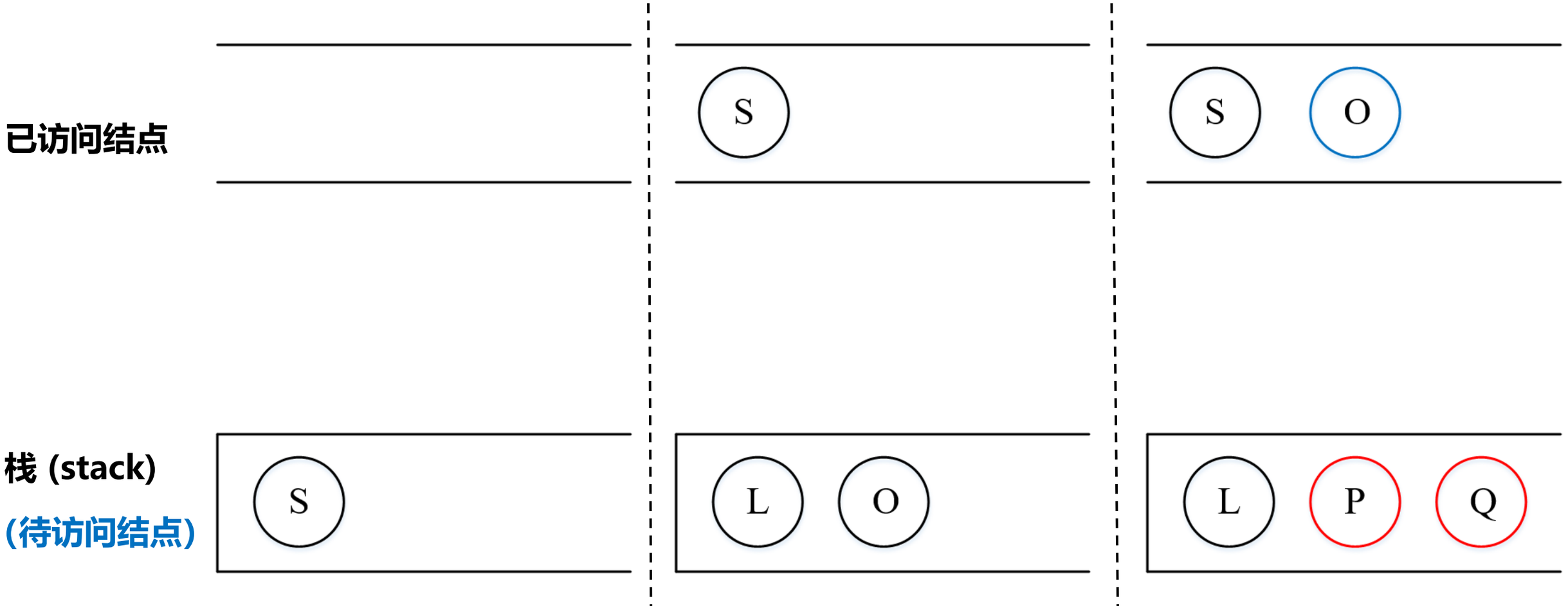
- **以教材第85页，图3.3为例**



深度优先：访问完某个结点N后，将结点N的所有邻居 (neighbor) 压入栈（stack），等待访问。

深度优先搜索

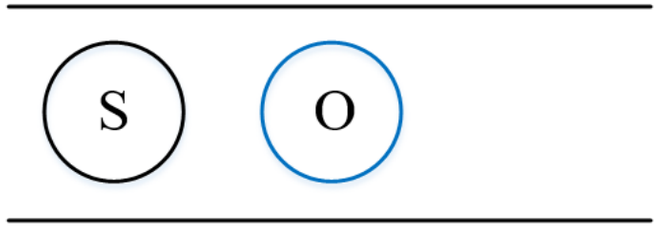
- 如何编程实现?

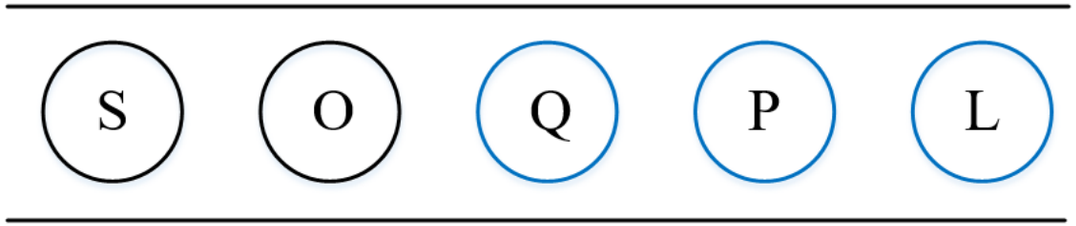


深度优先搜索

- 如何编程实现?

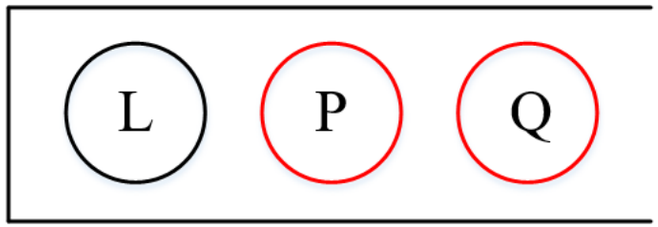
已访问结点





栈 (stack)

(待访问结点)



深度优先搜索

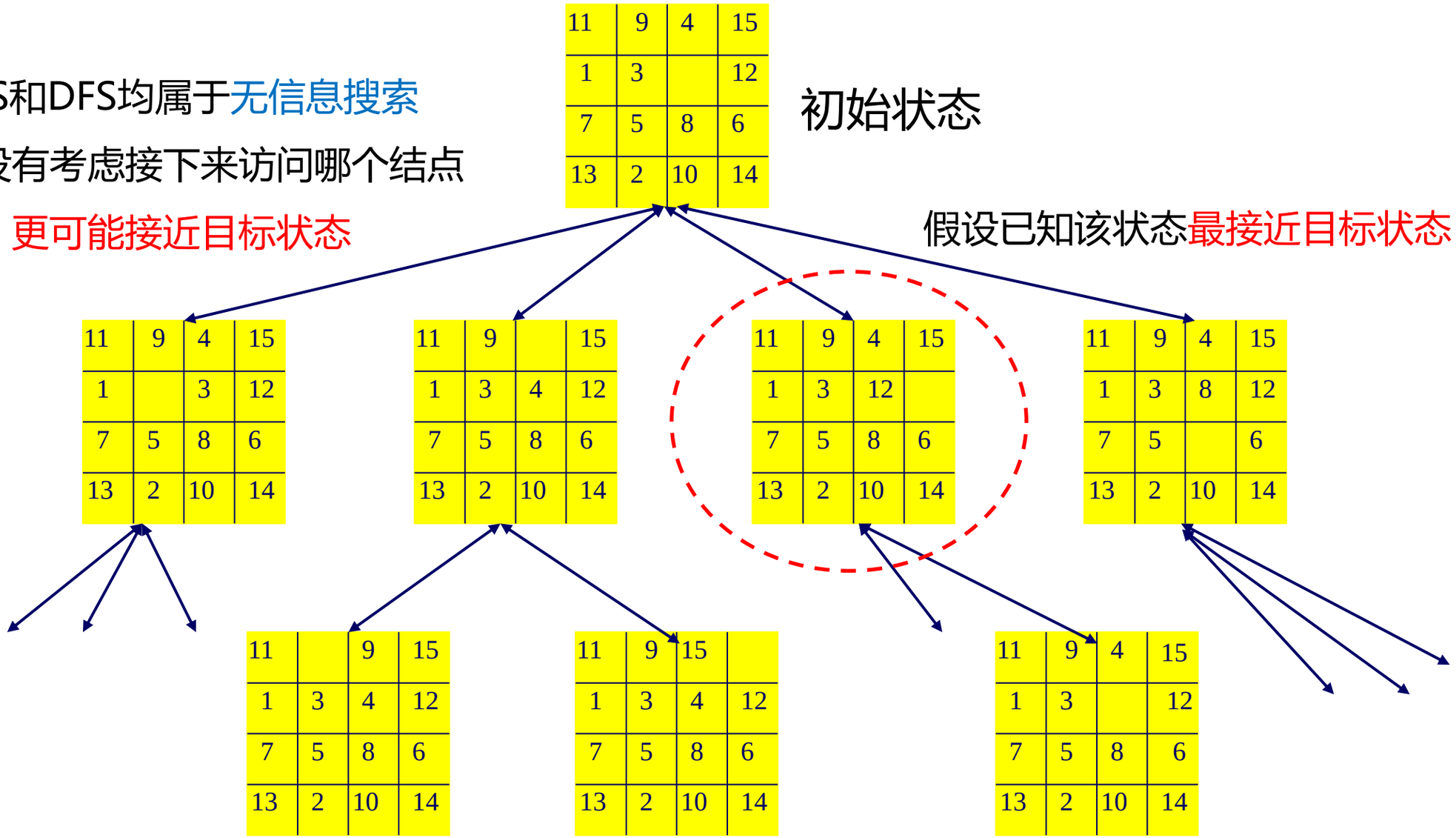
- **Python实现**

```
def BFS(start: Node):
    open = []
    closed = []
    visited = set()

    open.append(start)
    while len(open) > 0:
        # 把队列中的第一个结点取出
        # 加入已访问结点列表和集合中
        cur_node = open.pop(0)
        closed.append(cur_node)
        visited.add(cur_node)
        print(cur_node.name, end=' -> ')
        # 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
```

图搜索策略

BFS和DFS均属于无信息搜索

它们没有考虑接下来访问哪个结点



图搜索策略

- **无信息搜索 (uninformed search)**：搜索策略没有使用与某个具体问题有关的知识。

- **有信息搜索 (informed search)**：搜索策略使用了与某个具体问题有关的知识，以更加高效地寻找该问题的解。也称为启发式搜索 (heuristically search)。

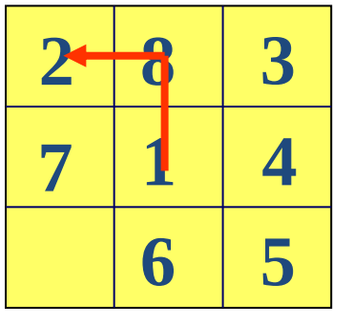
有信息搜索（对应教材3.3.1节）

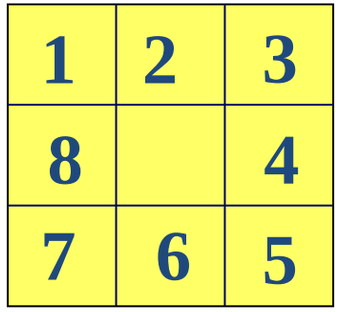
- **与目标状态的距离：**使用一个启发式函数 (heuristic function) / 估价函数 (evolution function) $f(n)$ 进行估算（具体问题具体设计）。

- **最佳优先搜索 (best-first search)：**访问完当前结点n后，接着访问n的邻居中离目标状态最近的结点。也被称为有序搜索 (ordered search)。

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）

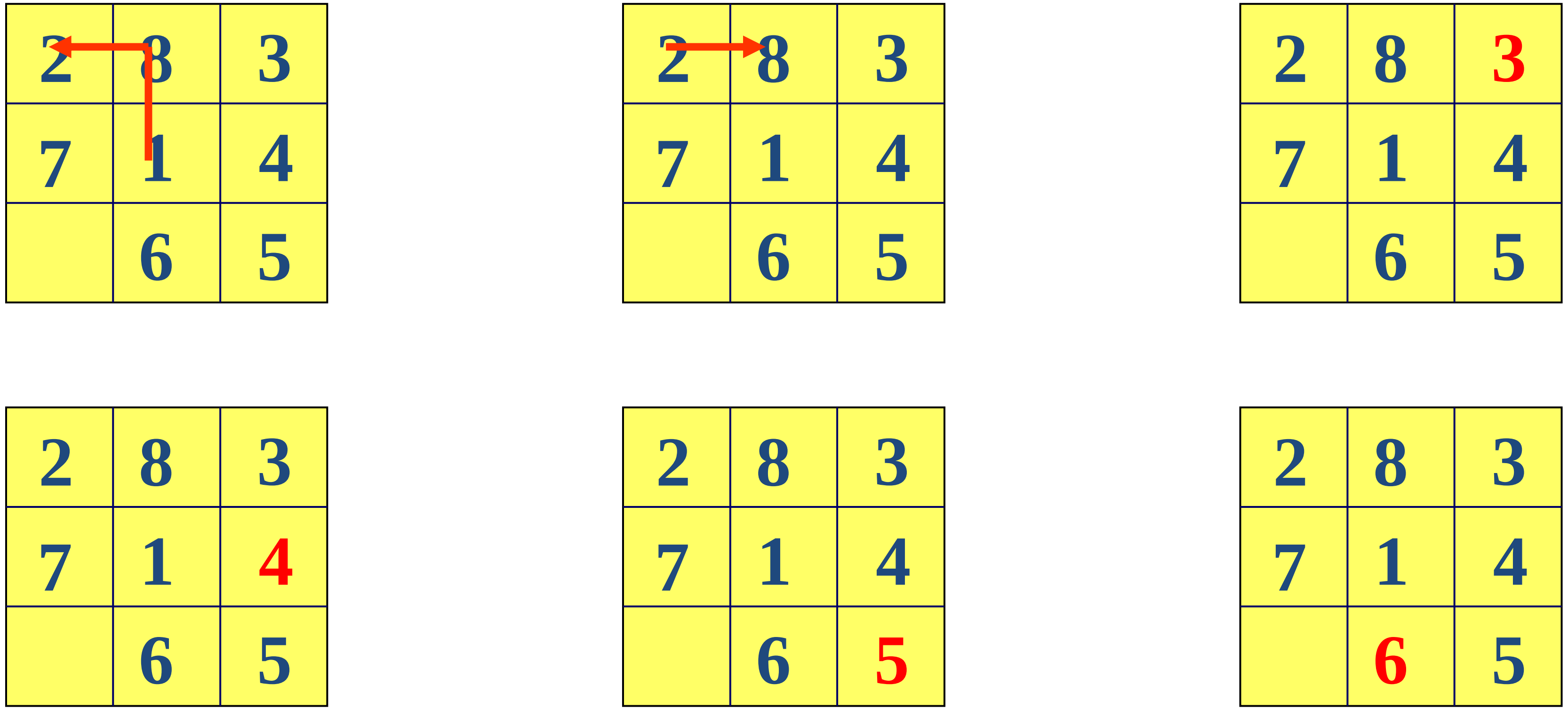


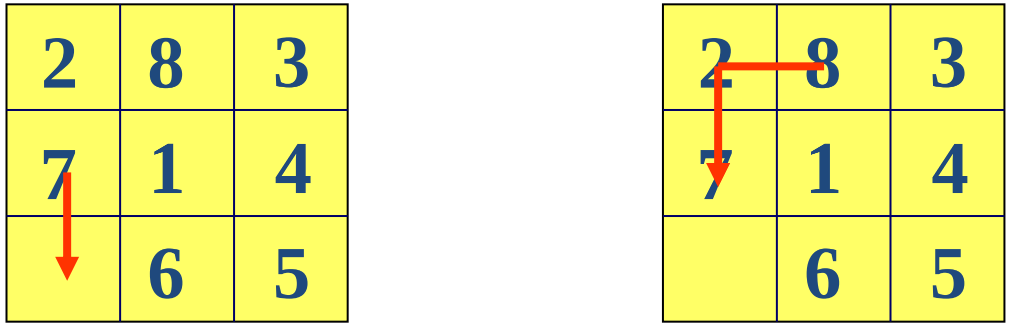


- ✓ 曼哈顿距离 (Manhattan Distance)：两点之间水平距离和垂直距离之和
- ✓ 8数码难题的估价函数：数码和目标位置之间的曼哈顿距离

最佳优先搜索

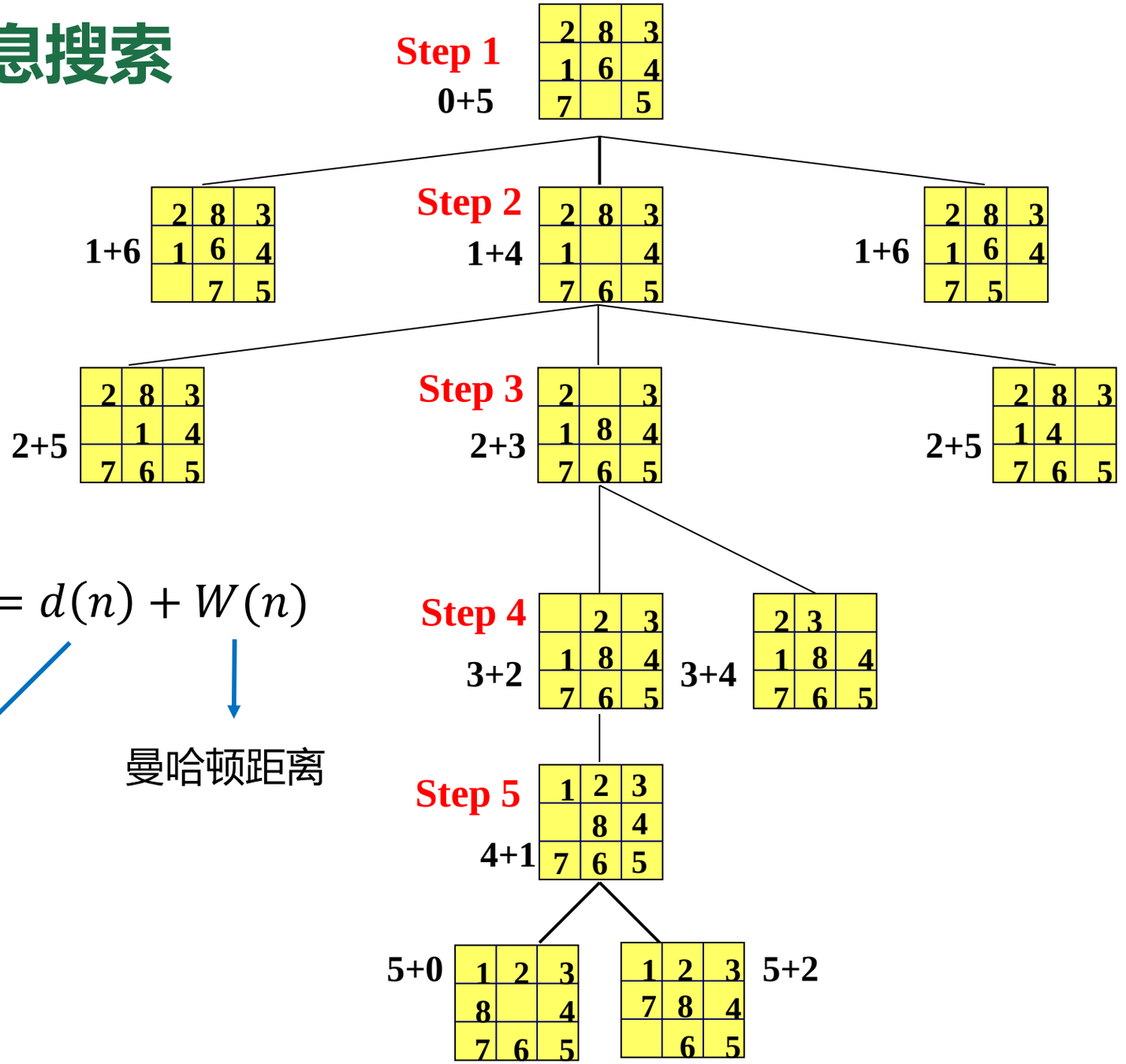
曼哈顿距离$W(n)$：两点之间水平距离和垂直距离之和





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

有信息搜索



估价函数 $f(n) = d(n) + W(n)$

结点的深度

曼哈顿距离

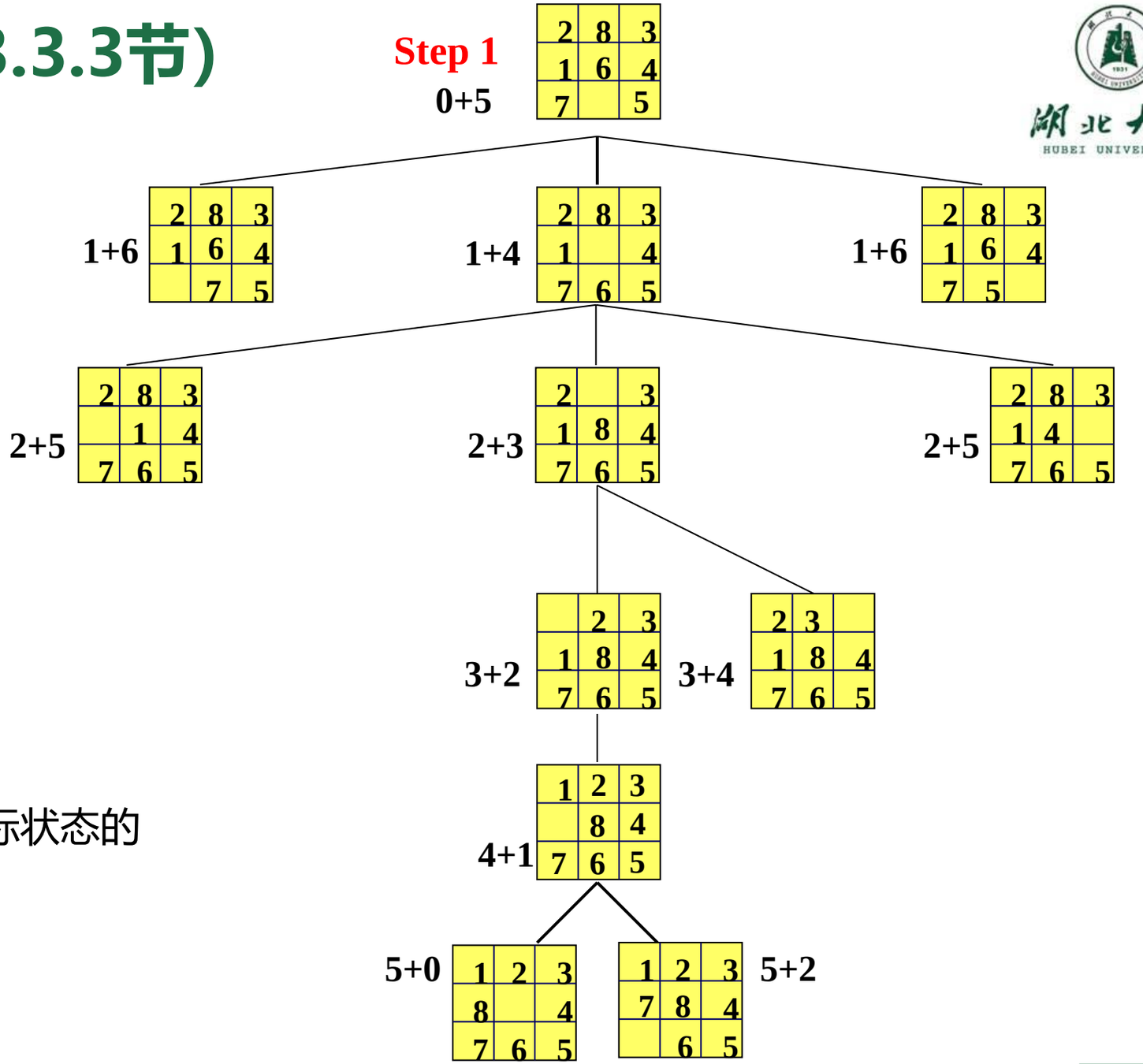
A*搜索（对应教材3.3.3节）

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



A*搜索

估价函数 $f(n) = g(n) + h(n)$

$g(n)$：到达结点n的代价

$h(n)$：结点n到达目标状态的估计代价

- A*算法是最优的，如果满足以下2个条件：
  - ✓ (可纳性) 对所有结点n， $h(n)$都比实际的最小代价$h^*(n)$更低，即$h(n) \leq h^*(n)$；
  - ✓ (一致性) 对所有的结点n，假设其后继为n'，从n到达n'的代价为c， $h(n) \leq h(n') + c$ 成立

先从结点n到结点n'，再从结点n'到达目标状态

A*搜索

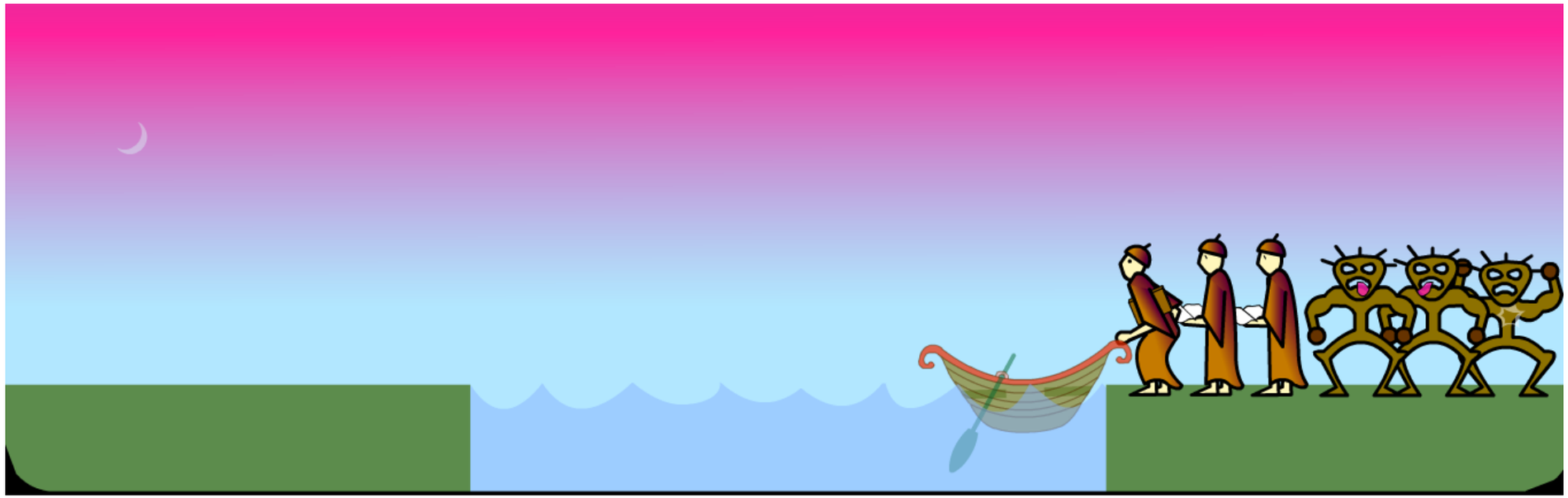
- A*算法是最优的，如果满足以下2个条件：
  - ✓ (可纳性) 对所有结点n， $h(n)$都比实际的最小代价$h^*(n)$更低，即$h(n) \leq h^*(n)$；
  - ✓ (一致性) 对所有的结点n，假设其后继为n'，从n到达n'的代价为c， $h(n) \leq h(n') + c$ 成立

- 反证法证明：
  1. 假设A*算法下一个拓展的结点为s，即$f(s) = g(s) + h(s)$在所有待扩展结点中最小；
  2. 假设结点s 不在最优的搜索路径上，即当前应该拓展结点s'；
  - ✓ 根据假设1， $g(s') + h(s') \geq g(s) + h(s)$
  - ✓ 设从结点s'到结点s的代价为c， $g(s') + c < g(s)$
  - ✓ 不等式两边同时加$h(s)$， $g(s') + h(s) + c < g(s) + h(s)$
  - ✓ 根据一致性，$g(s') + h(s') \leq g(s') + h(s) + c < g(s) + h(s)$
  - ✓ 得到$g(s') + h(s') < g(s) + h(s)$，与 $g(s') + h(s') \geq g(s) + h(s)$ 矛盾

状态空间表示

- **传教士野人问题 (Missionaries & Cannibals Problem)**

有三个传教士和三个野人过河，只有一条能装下两个人的船。如果在河的一方（或船上），野人的人数大于传教士的人数，那么传教士就会有危险。能否设计出一种安全的渡河方案?



状态空间表示

- **传教士野人问题 (Missionaries & Cannibals Problem)**

  ✓ 定义状态：(左岸传教士人数, 右岸传教士人数, 左岸野人数, 右岸野人数, 船的位置)

  ✓ 状态可以简化为：(左岸传教士人数, 左岸野人数, 船的位置)

初始状态：$(0, 0, R)$

目标状态：$(3\ , 3, L)$

状态空间表示

- **传教士野人问题 (Missionaries & Cannibals Problem)**

定义算符：坐船移动，Move(传教士人数，野人人数，方向)

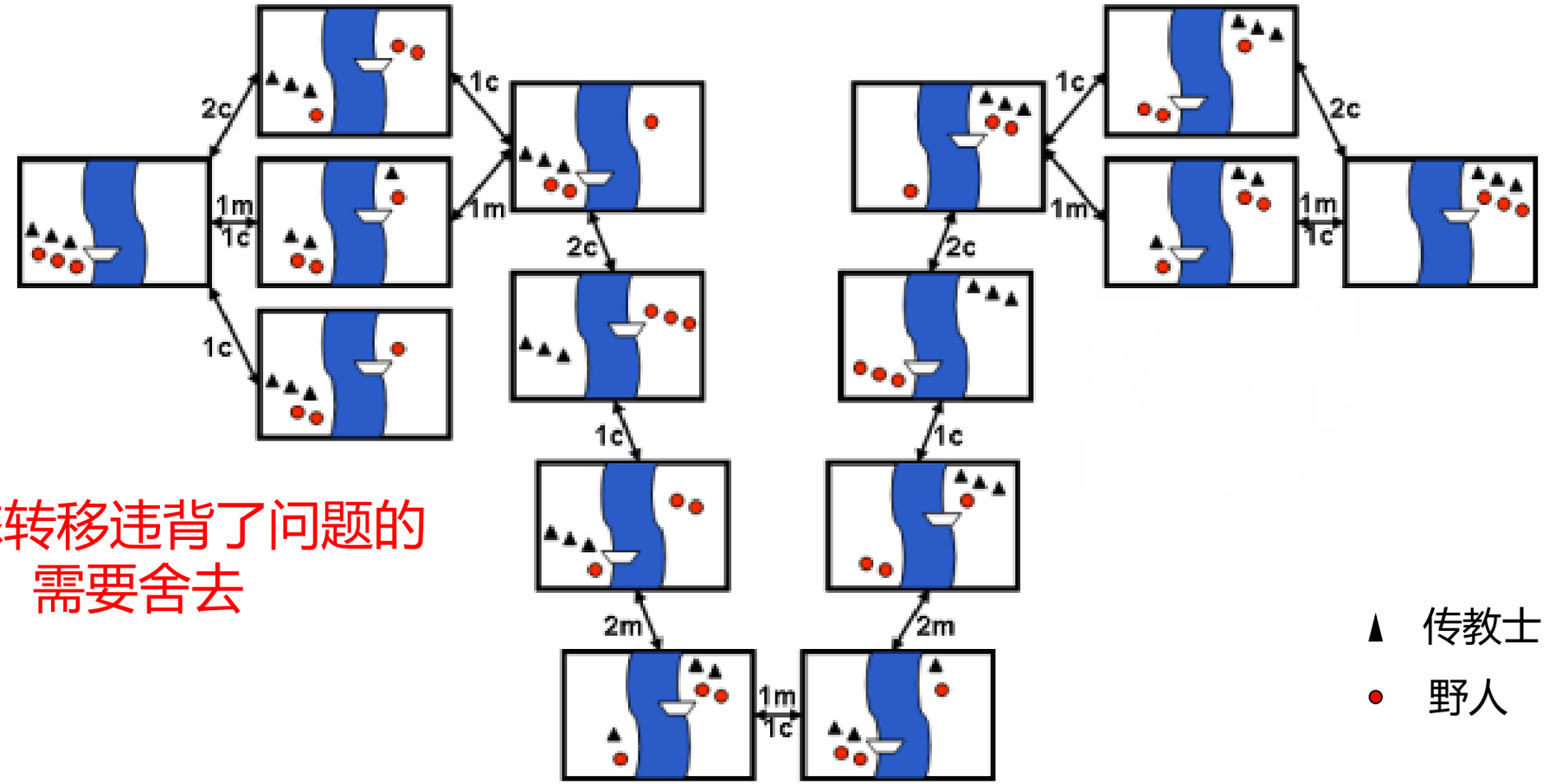
限定：船最多只能坐2人，且传教士人数需大于等于野人人数

$Move(1,1,L \to R)$	$Move(1,1,R \to L)$	1传教士 + 1野人
$Move(2,0,L \to R)$	$Move(2,0,R \to L)$	2传教士
$Move(0,2,L \to R)$	$Move(0,2,R \to L)$	2野人
$Move(1,0,L \to R)$	$Move(1,0,R \to L)$	1传教士
$Move(0,1,L \to R)$	$Move(0,1,R \to L)$	1野人

状态空间表示

- **传教士野人问题 (Missionaries & Cannibals Problem)**

画出状态空间：m代表传教士，c代表野人



*某些状态转移违背了问题的限制条件，需要舍去

作 业

- **理论课 第2次平时作业**（截止时间：3月19日上课前交给班长）

1. 简述宽度优先搜索、深度优先搜索和A*搜索三者之间的异同点。

2. 用A*搜索解决教材第130页，图3.30中的八数码难题。

结束语

湖北大学
HUBEI UNIVERSITY

谢 谢!