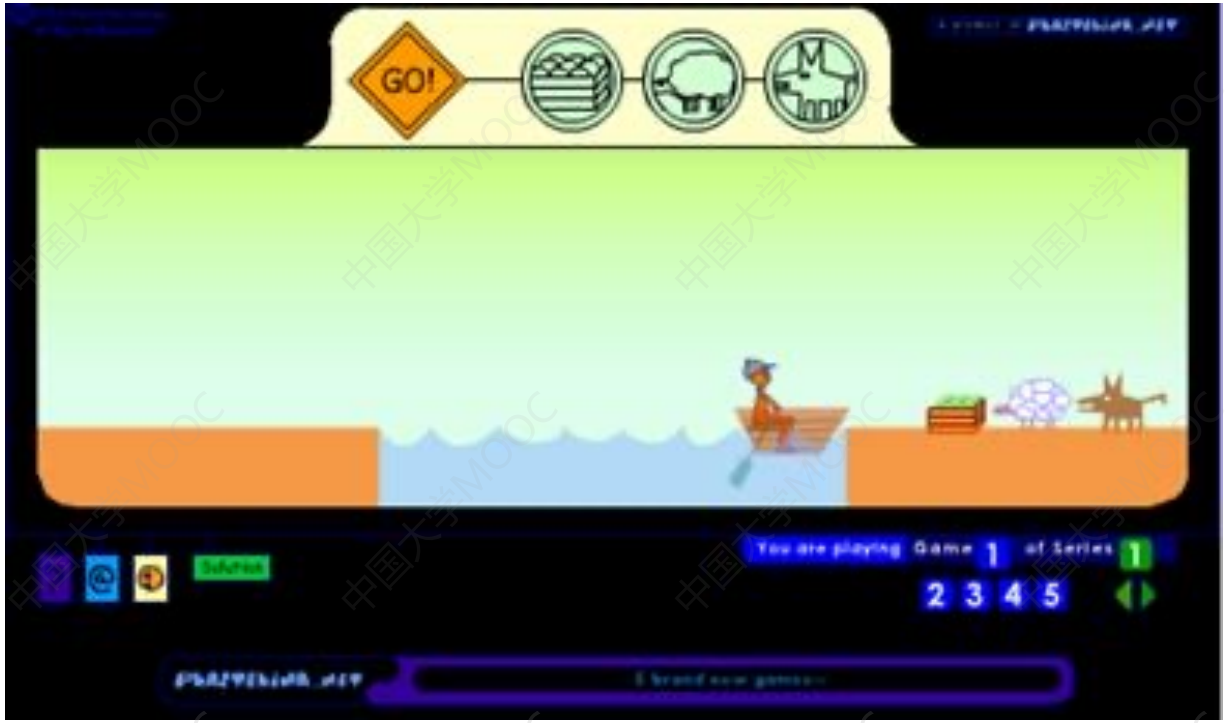
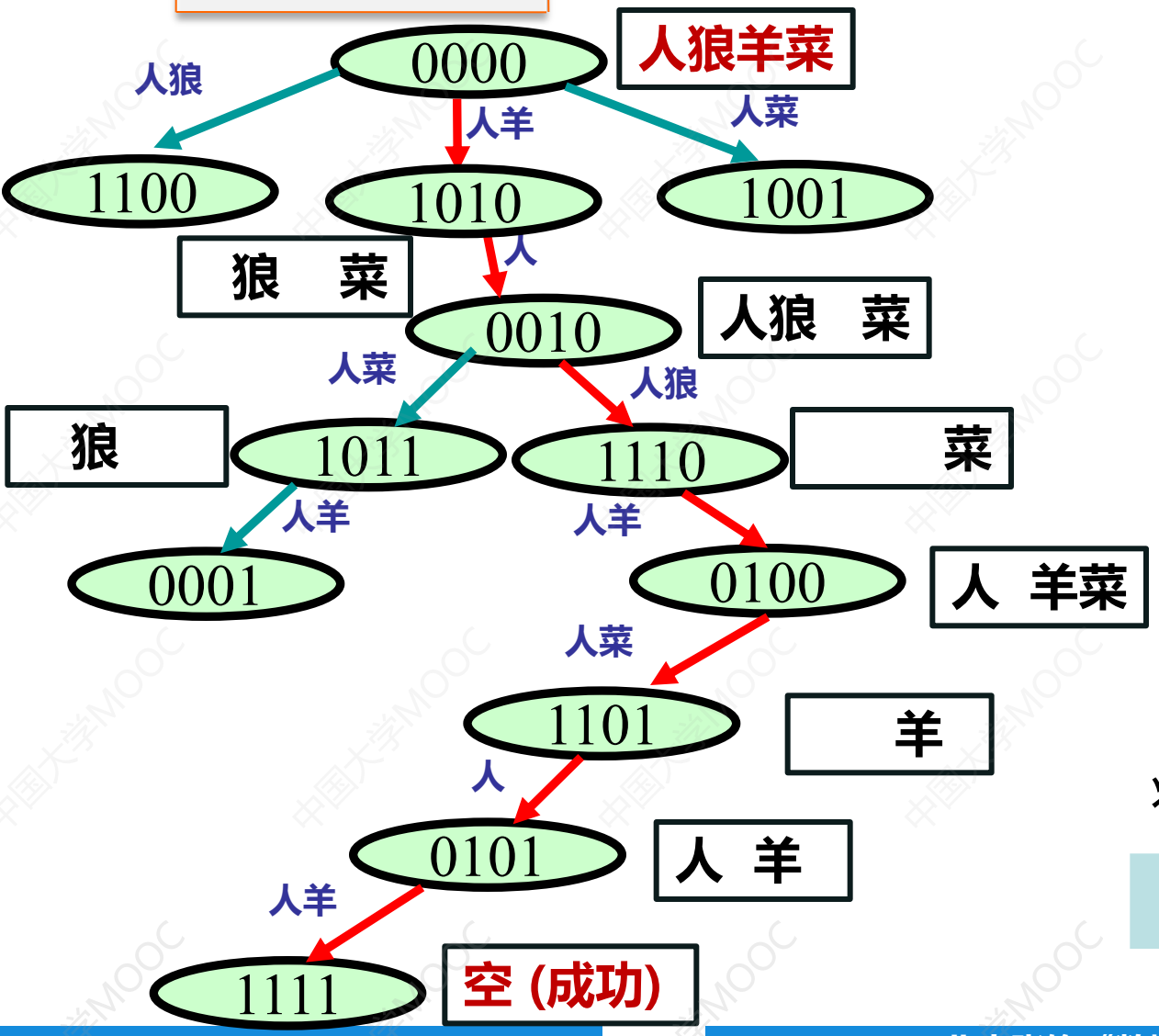
队列的应用

- 只要满足先来先服务特性的应用均可采用队列作为其数据组织方式或中间数据结构
- 调度或缓冲
  - 消息缓冲器
  - 邮件缓冲器
  - 计算机硬设备之间的通信也需要队列作为数据缓冲
  - 操作系统的资源管理
- 宽度优先搜索

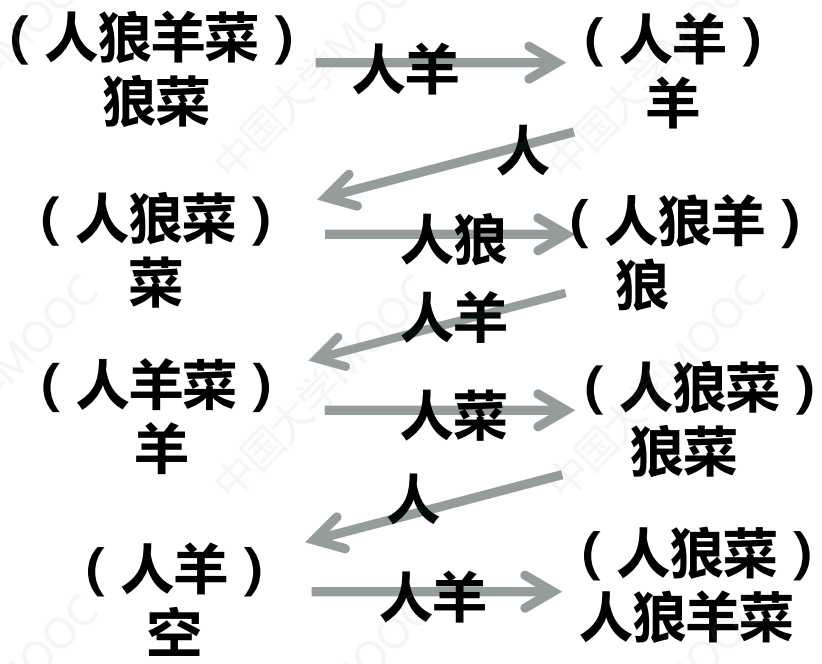
农夫过河问题

- **问题抽象**：“人狼羊菜”乘船过河
  - 只有人能撑船，船只有两个位置（包括人）
  - 狼羊、羊菜不能在没有人时共处





算法示意



状态位向量，物体在起始为0，否则为1

0	1	0	1
人	狼	羊	菜

问题分析

- 求解该问题最简单的方法是使用试探法，即
一步一步进行试探，每一步都搜索所有可能的选择，
对前一步合适的选择再考虑下一步的各种方案。

- 用计算机实现上述求解的搜索过程可以采用两种不同的策略：

广度优先搜索：搜索该步的所有可能状态，再进一步考虑后面的各种情况；（队列应用）

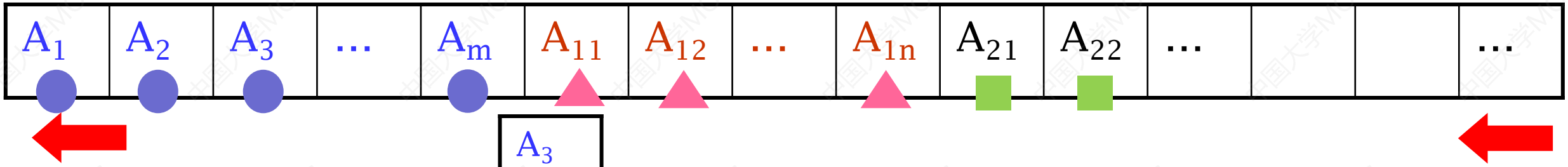
深度优先搜索：沿某一状态走下去，不行再回头。（栈应用）

问题分析

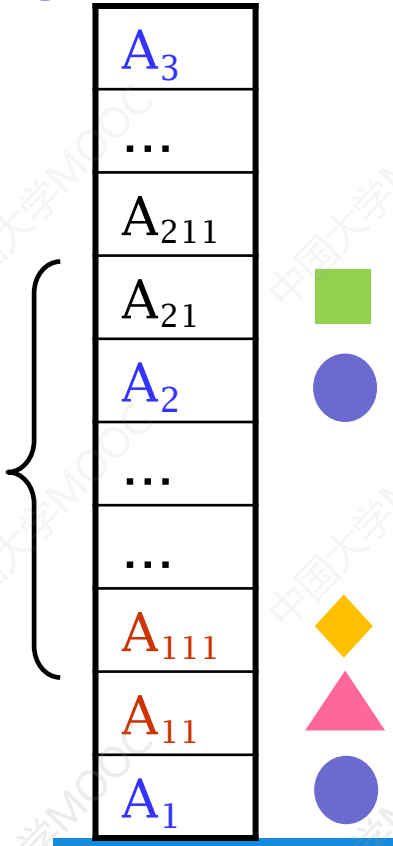
- 假定采用广度优先搜索解决农夫过河问题：
  - 采用队列做辅助结构，把下一步所有可能达到的状态都放在队列中，然后顺序取出对其分别处理，处理过程中再把下一步的状态放在队列中，……。
  - 由于队列的操作按照先进先出原则，因此只有前一步的所有情况都处理完后才能进入下一步。

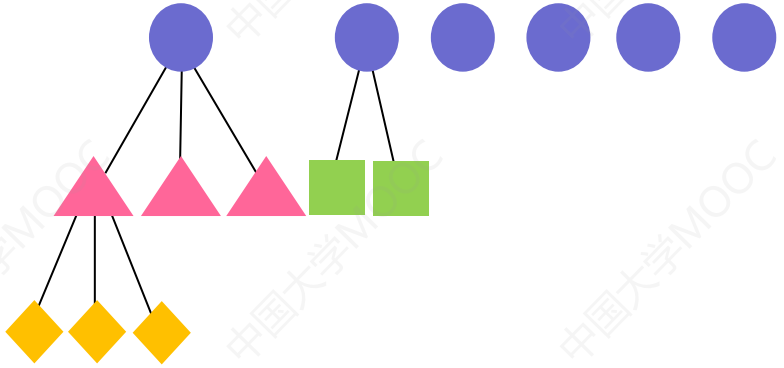
问题分析

广度优先：(m 种状态)



深度优先：(m 种状态)





数据抽象

- 每个角色的位置进行描述
  - 农夫、狼、羊和菜，四个目标依次各用一位，目标在起始岸位置：0，目标岸：1

0	1	1	0

  - 如 0110 表示农夫、白菜在起始岸，而狼、羊在目标岸（此状态为不安全状态）

数据的表示

- 用整数 status 表示上述四位二进制描述的状态
  - 整数 0x08 表示的状态

1	0	0	0

  - 整数 0x0F 表示的状态

1	1	1	1

- 如何从上述状态中得到每个角色所在位置？
  - 函数返回值为真（1），表示所考察人或物在目标岸
  - 否则，表示所考察人或物在起始岸

确定每个角色位置的函数

```
bool farmer(int status)
{  return ((status & 0x08)  != 0);     }

bool wolf(int status)
{  return ((status & 0x04)  != 0);     }

bool goat(int status)
{  return ((status & 0x02)  != 0);     }

bool cabbage(int status)
{  return ((status & 0x01)  != 0);     }
```

人	狼	羊	菜
1	x	x	x
x	1	x	x
x	x	1	x
x	x	x	1

人	狼	羊	菜
0	1	0	1

安全状态的判断

```
bool safe(int status)                    // 返回 true:安全，false:不安全
{
    if ((goat(status) == cabbage(status)) &&
        (goat(status) != farmer(status)))
            return(false);               // 羊吃白菜
    if ((goat(status) == wolf(status)) &&
        (goat(status) != farmer(status)))
            return(false);               // 狼吃羊
    return(true);                        // 其它状态为安全
}
```

算法抽象

- 问题变为

从状态0000（整数0）出发，寻找全部由安全状态构成的状态序列，以状态1111（整数15）为最终目标。

  - 状态序列中 每个 状态都可以从前一状态通过农夫（可以带一样东西）划船过河的动作到达。
  - 序列中不能出现 重复 状态

算法设计

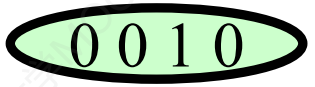
- 定义一个整数队列 moveTo，它的每个元素表示一个可以安全到达的中间状态
- 还需要定义一个数据结构 记录已被访问过的各个状态，以及已被发现的能够到达当前这个状态的路径
  - 用顺序表 route 的第 i 个元素记录状态i是否已被访问过
  - 若 route[i] 已被访问过，则在这个顺序表元素中记入前驱状态值； -1表示未被访问
  - route 的大小（长度）为 16

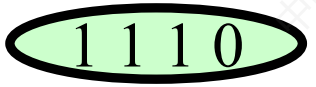
算法实现

```
void solve() {
    int movers, i, location, newlocation;
    vector<int> route(END+1, -1);
                        // 记录已考虑的状态路径
    queue<int> moveTo;
    // 准备初值
    moveTo.push(0x00);
    route[0]=0;
```

算法实现

人狼羊菜






```
while (!moveTo.empty() && route[15] == -1) {
    // 得到现在的状态
    status = moveTo.front();
    moveTo.pop();
    for (movers = 1; movers <= 8; movers <<= 1) {
    // 农夫总是在移动，随农夫移动的也只能是在农夫同侧的东西
        if (farmer(status)  == (bool)(status & movers)) {
            // 随农夫移动以后的状态
            newstatus = status ^ (0x08 | movers);
            // 安全的，并且未考虑过的走法
            if (safe(newstatus) && (route[newstatus] == -1)) {
                route[newstatus] = status;
                moveTo.push(newstatus); }
        }
    }
}
```

算法实现

```
    // 反向打印出路径
    if (route[15] != -1) {
        cout << "The reverse path is : “ << endl;
        for (int status = 15; status >= 0; status = route[status]) {
            cout << "The status is : “ << status << endl;
            if (status == 0) break;
        }
    }
    else
        cout << "No solution.“ << endl;
}
```