

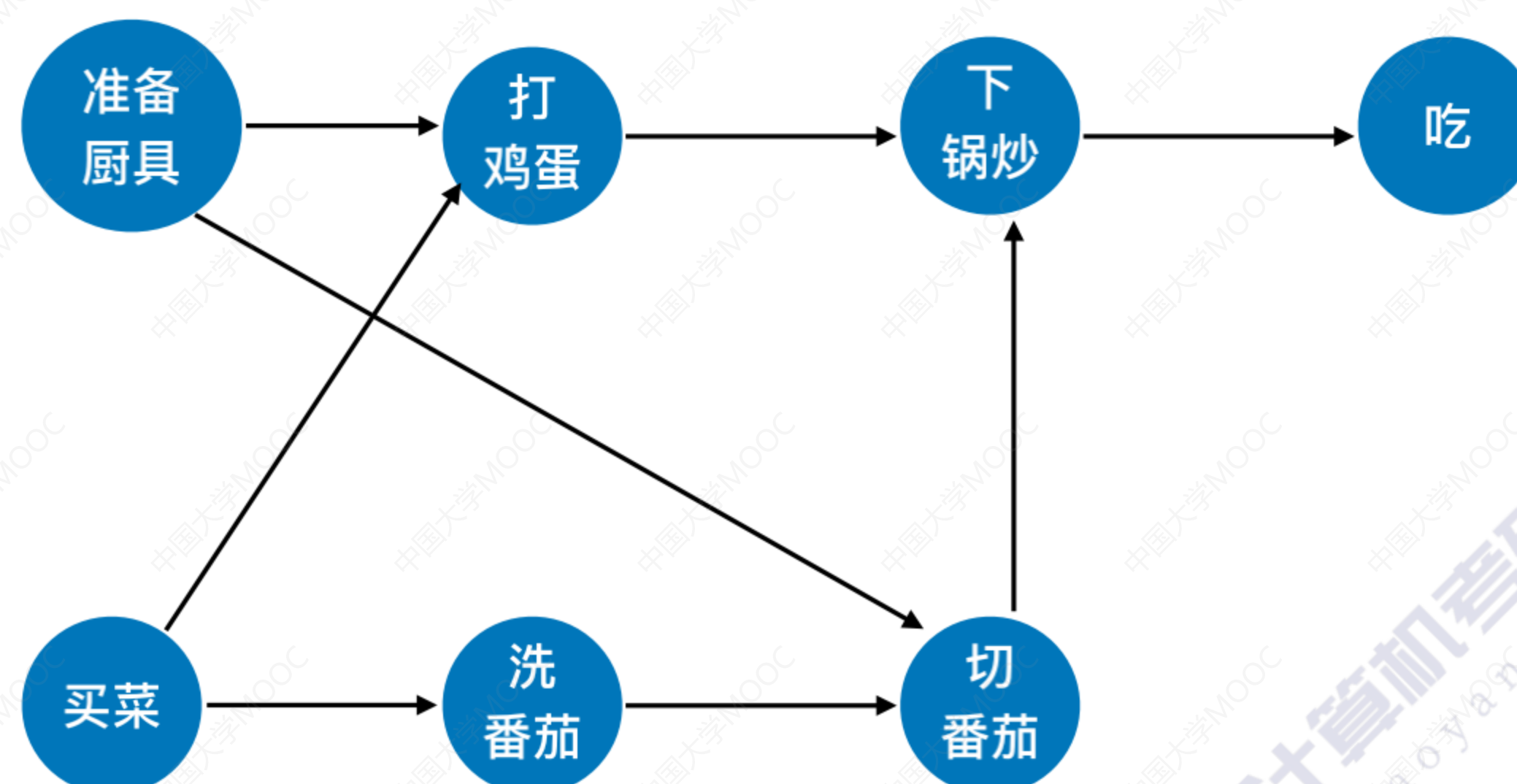
## 本节内容

# 拓扑排序

王道考研/CSKAOYAN.COM

## AOV网

AOV网(Activity On Vertex NetWork, 用顶点表示活动的网):  
用DAG图(有向无环图)表示一个工程。顶点表示活动, 有向边 $\langle V_i, V_j \rangle$ 表示活动 $V_i$ 必须先于活动 $V_j$ 进行

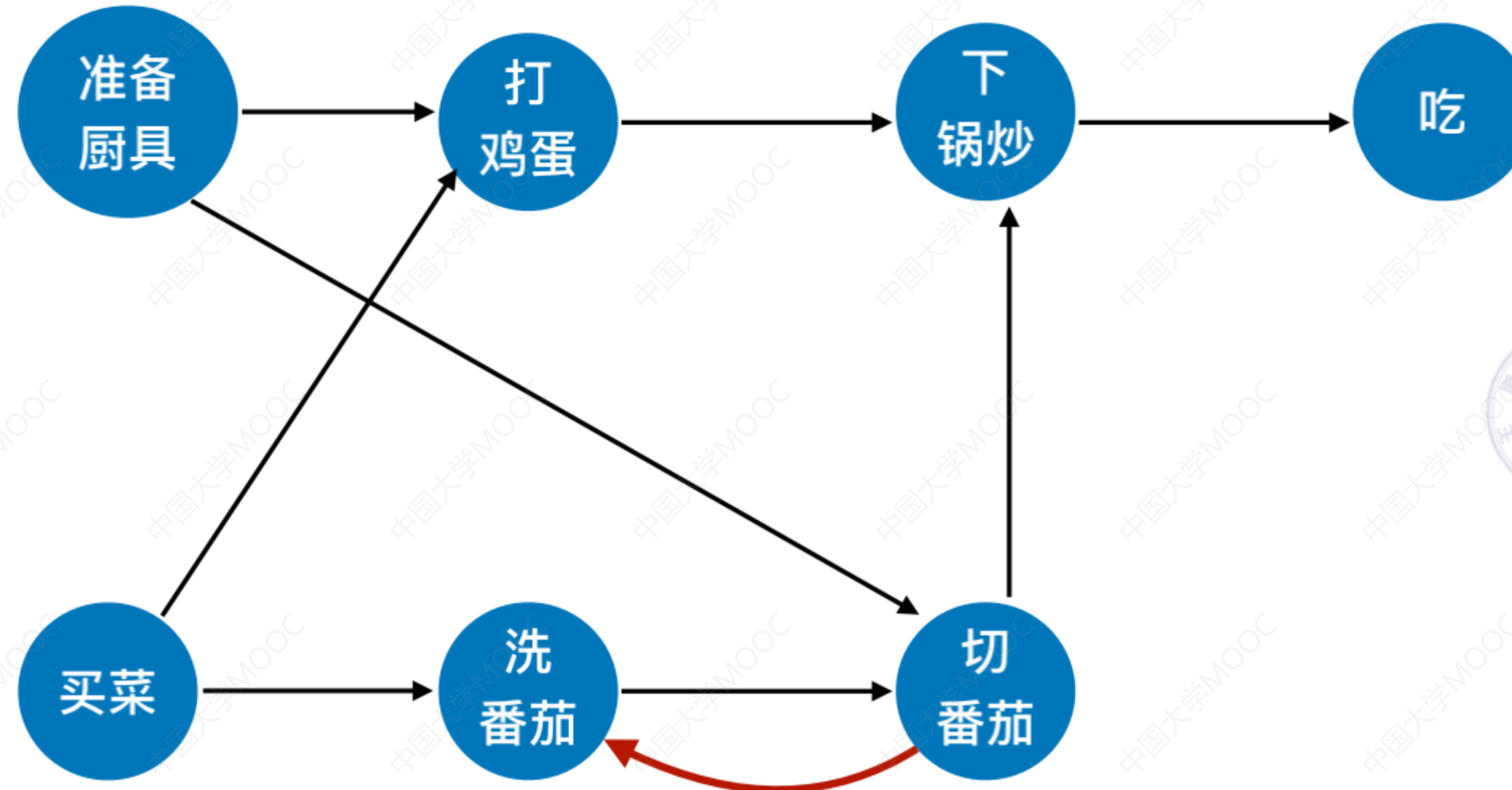


表示“番茄炒蛋工程”的AOV网

王道考研/CSKAOYAN.COM

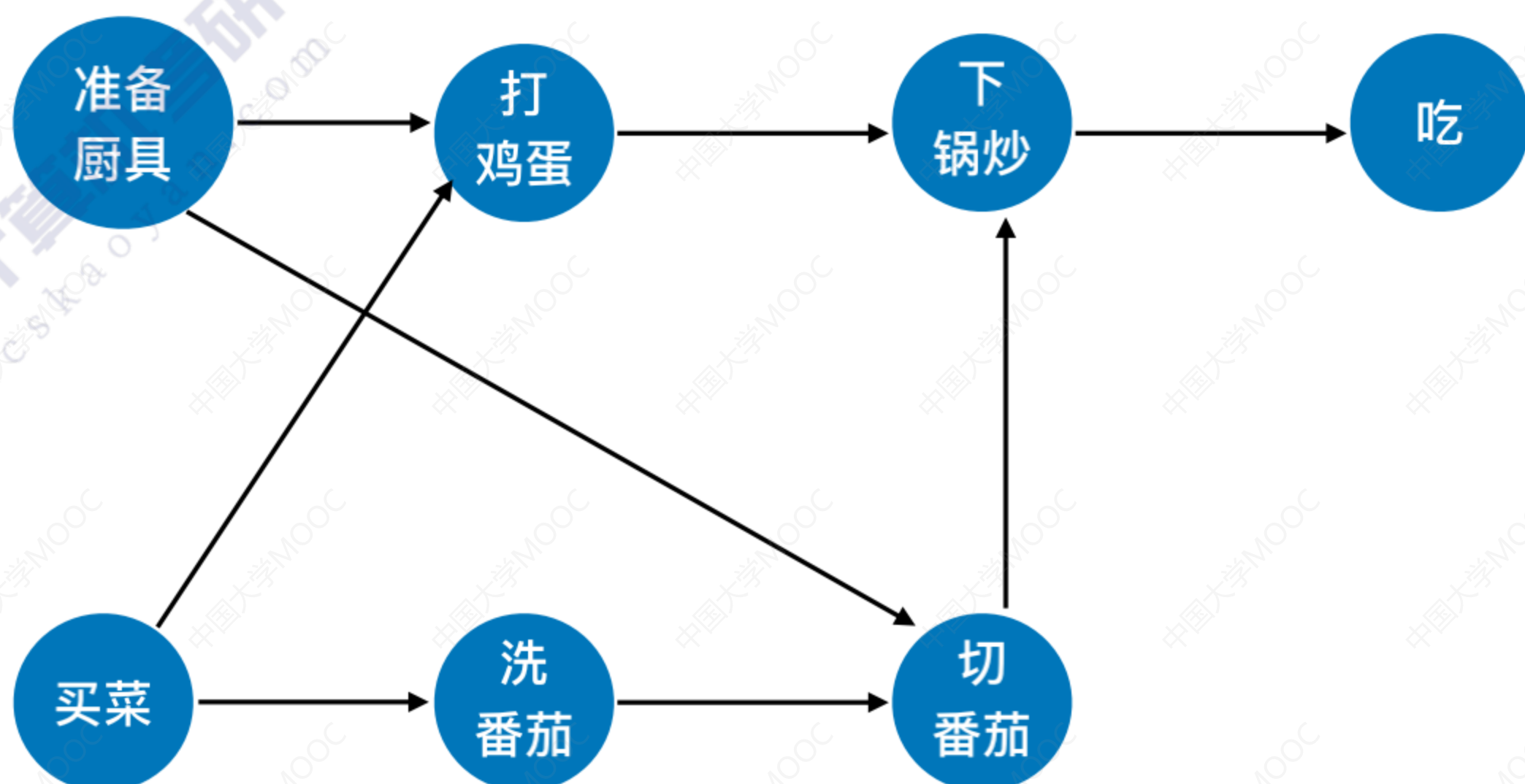
## AOV网

AOV网(Activity On Vertex NetWork, 用顶点表示活动的网):  
用DAG图(有向无环图)表示一个工程。顶点表示活动, 有向边 $\langle V_i, V_j \rangle$ 表示活动 $V_i$ 必须先于活动 $V_j$ 进行



王道考研/CSKAOYAN.COM

## 拓扑排序



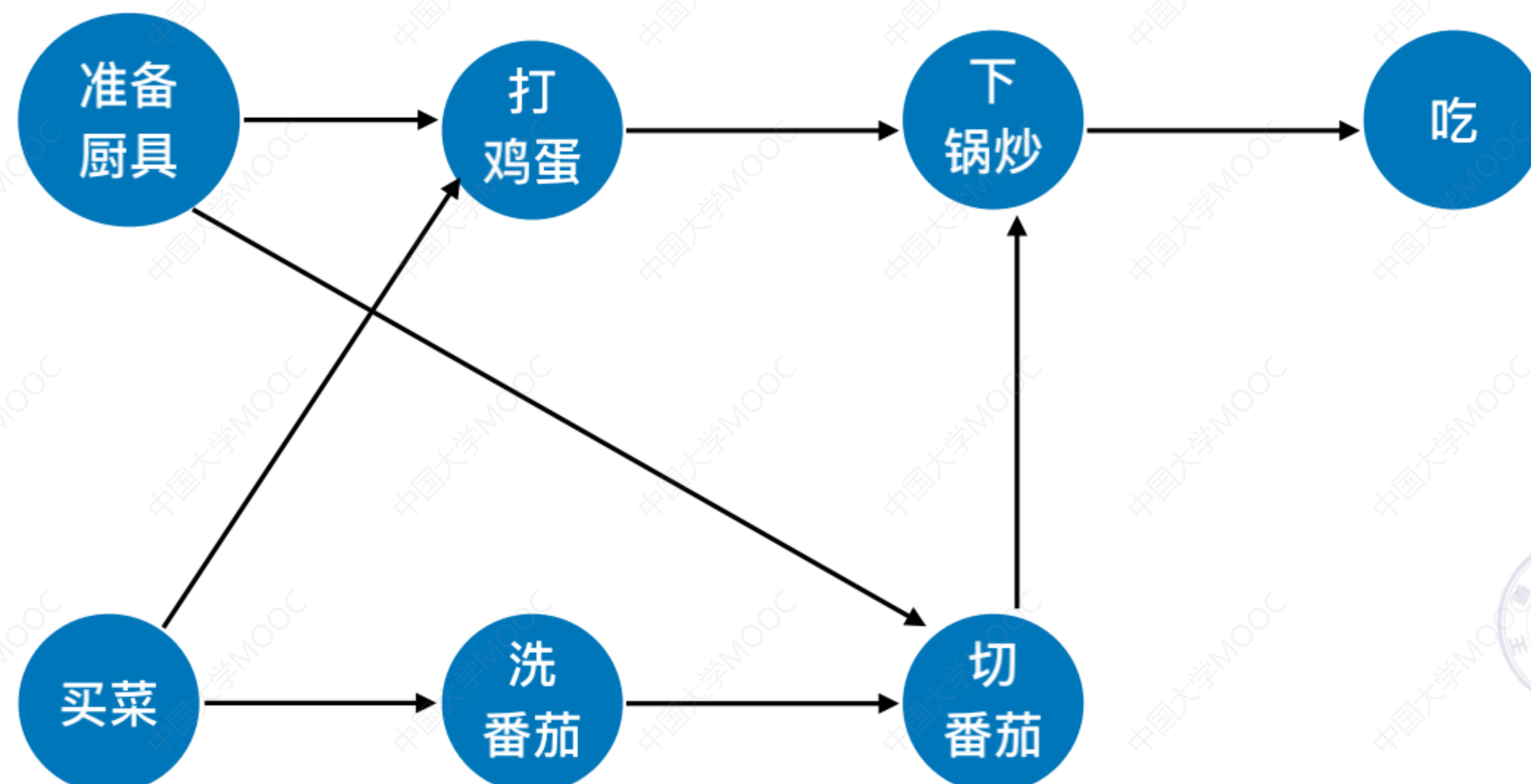
**拓扑排序:** 在图论中, 由一个有向无环图的顶点组成的序列, 当且仅当满足下列条件时, 称为该图的一个拓扑排序:

- ① 每个顶点出现且只出现一次。
- ② 若顶点A在序列中排在顶点B的前面, 则在图中不存在从顶点B到顶点A的路径。

或定义为: 拓扑排序是对有向无环图的顶点的一种排序, 它使得若存在一条从顶点A到顶点B的路径, 则在排序中顶点B出现在顶点A的后面。每个AOV网都有一个或多个拓扑排序序列。

王道考研/CSKAOYAN.COM

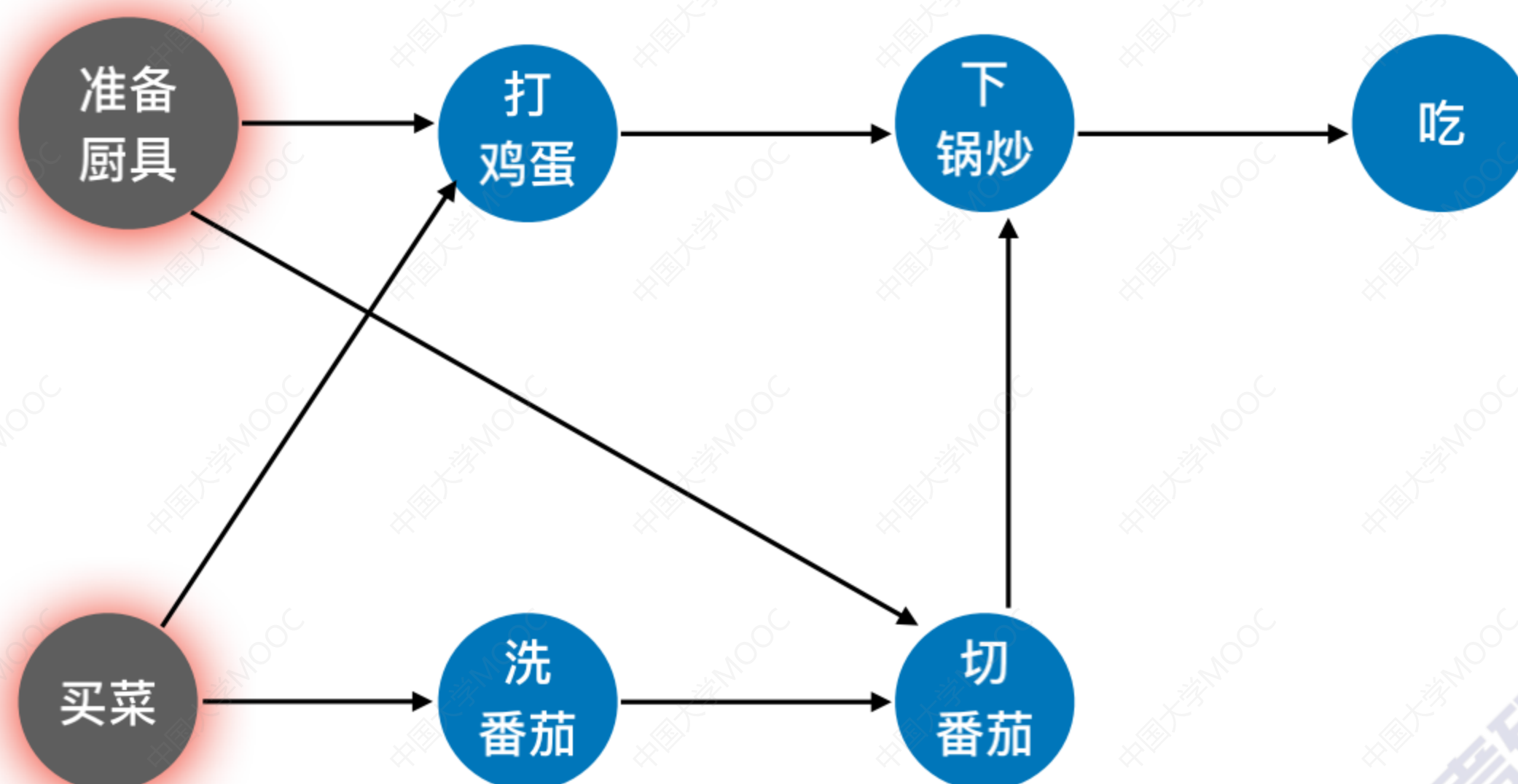
## 拓扑排序



拓扑排序：找到做事的先后顺序

王道考研/CSKAOYAN.COM

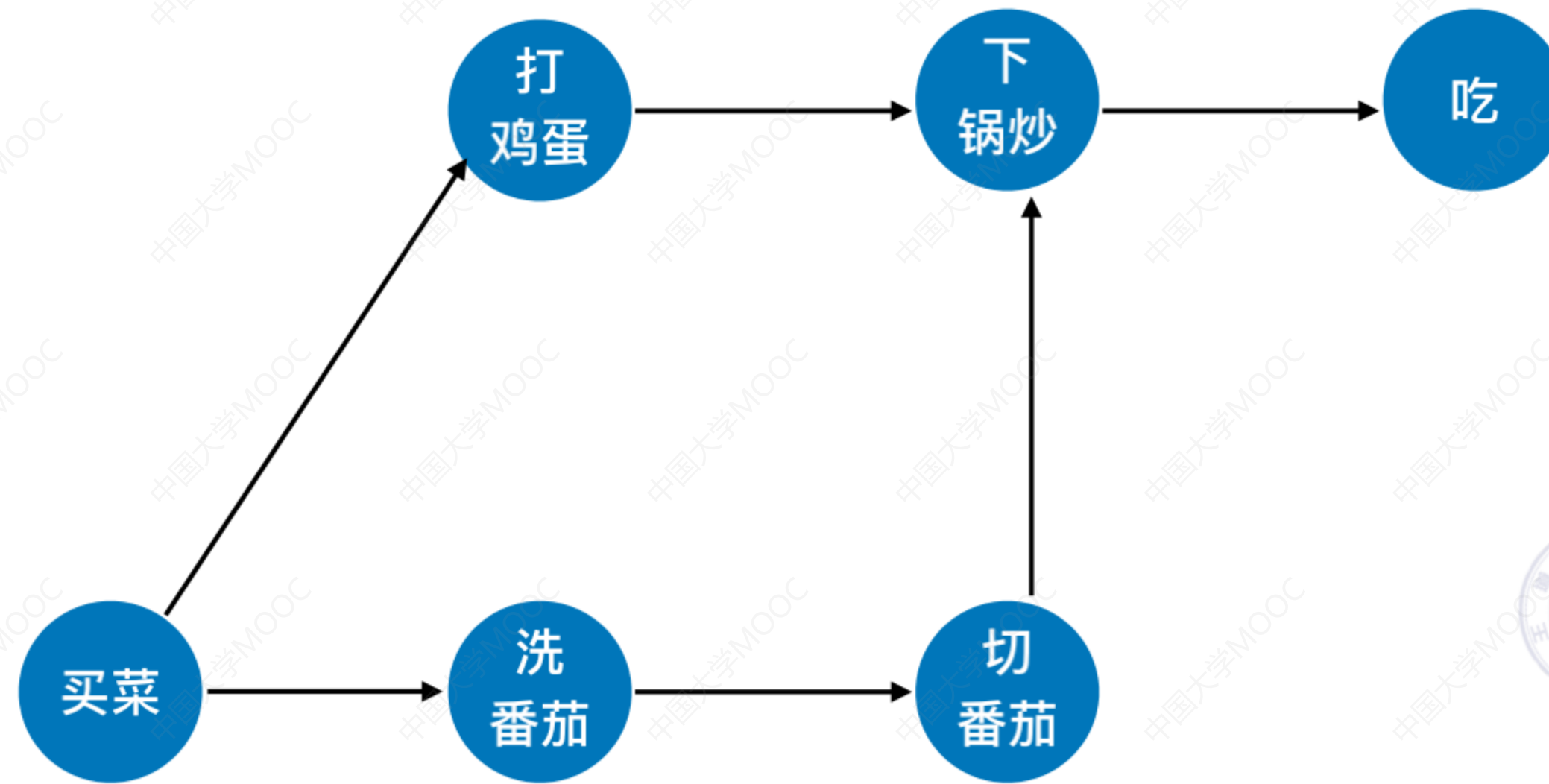
## 拓扑排序



拓扑排序：找到做事的先后顺序

王道考研/CSKAOYAN.COM

## 拓扑排序

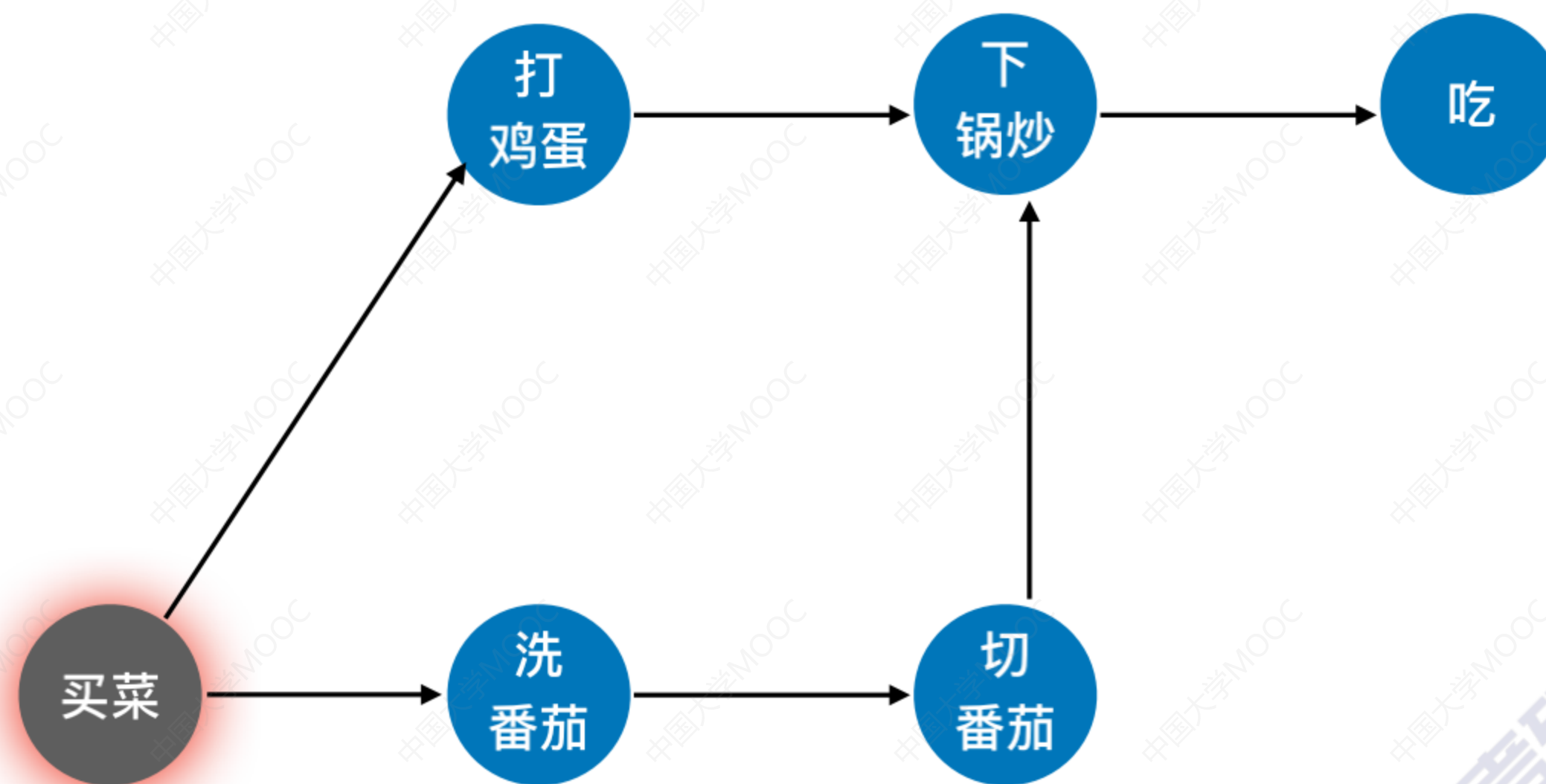


拓扑排序：找到做事的先后顺序

准备  
厨具

王道考研/CSKAOYAN.COM

## 拓扑排序

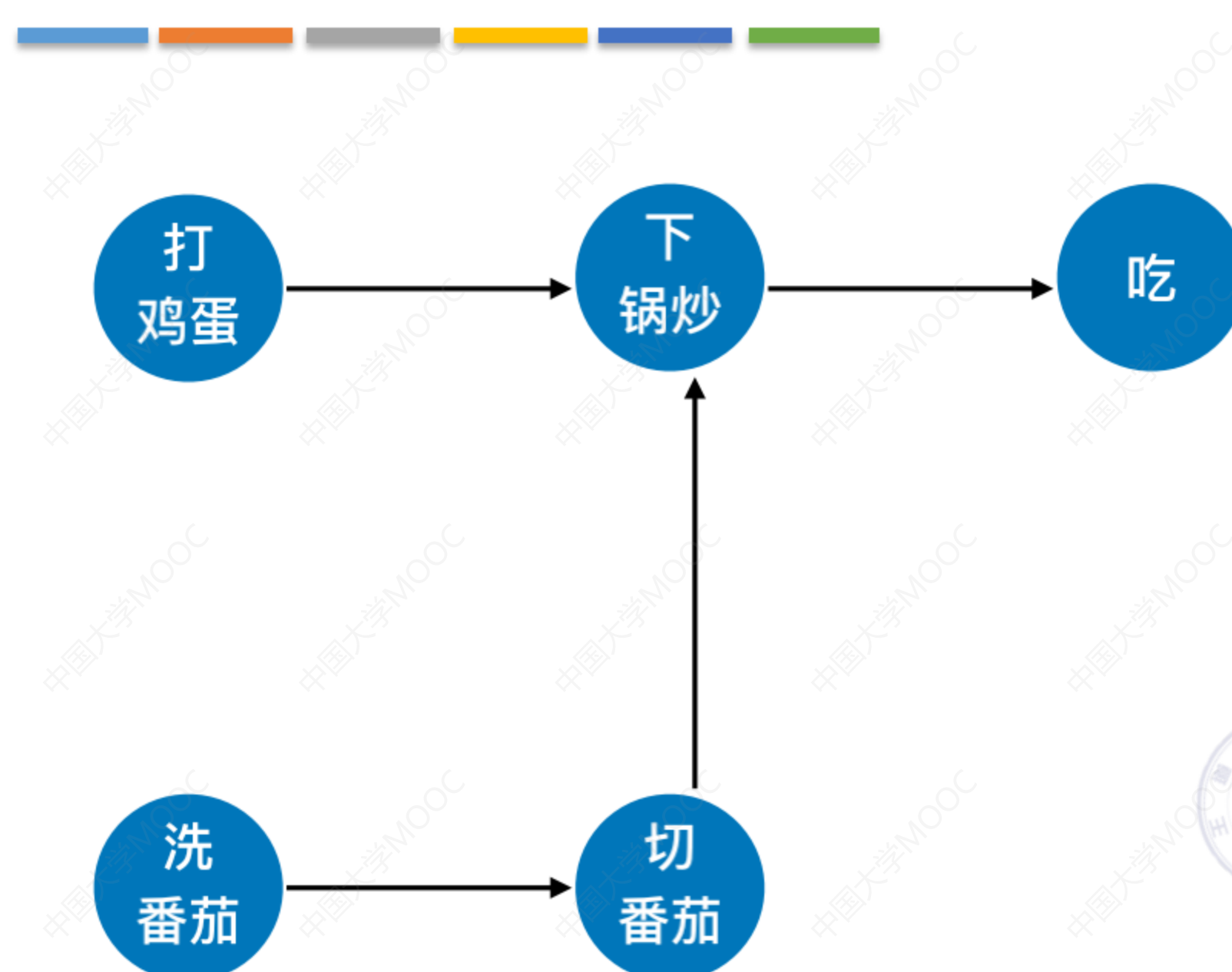


拓扑排序：找到做事的先后顺序

准备  
厨具

王道考研/CSKAOYAN.COM

## 拓扑排序

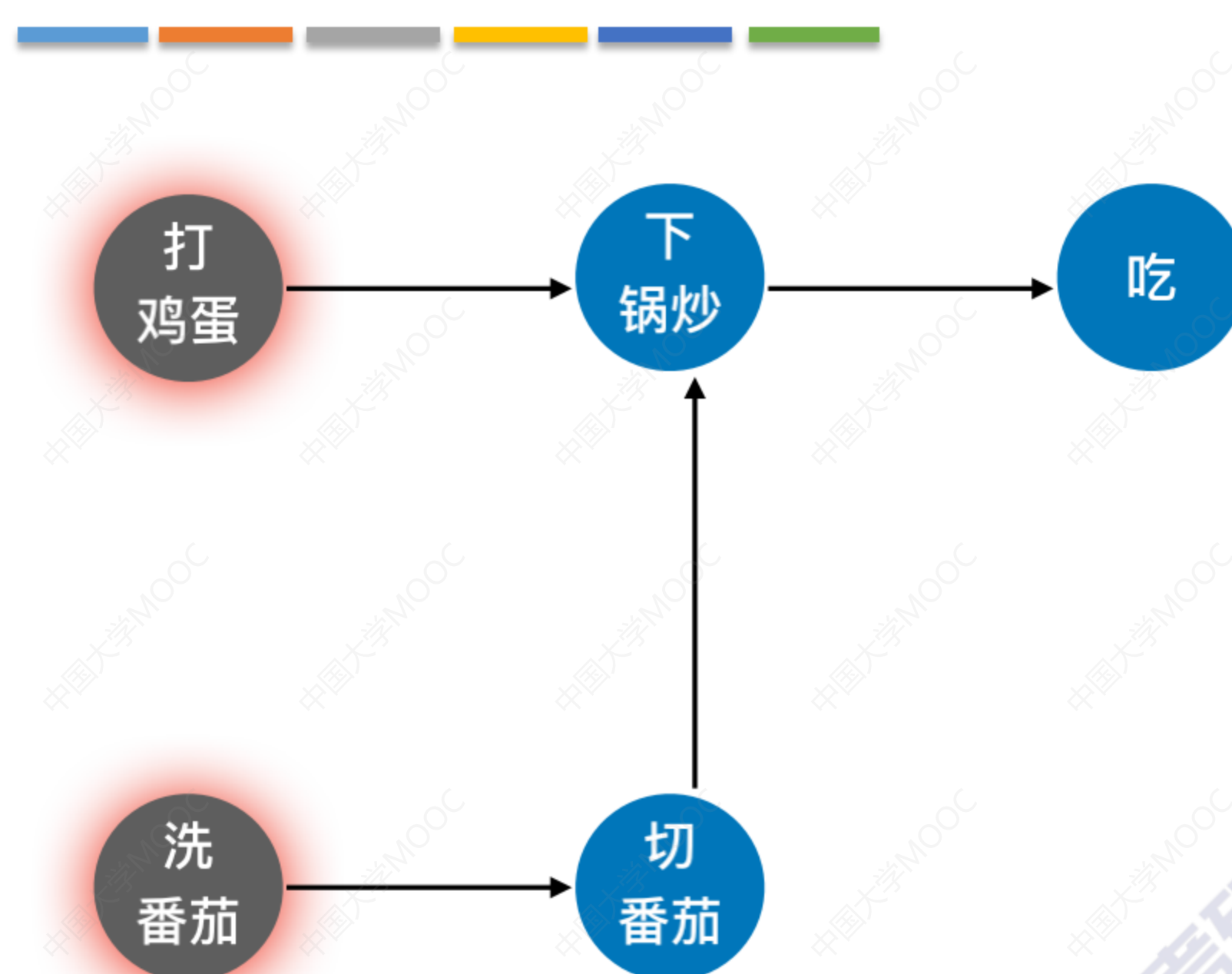


拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序

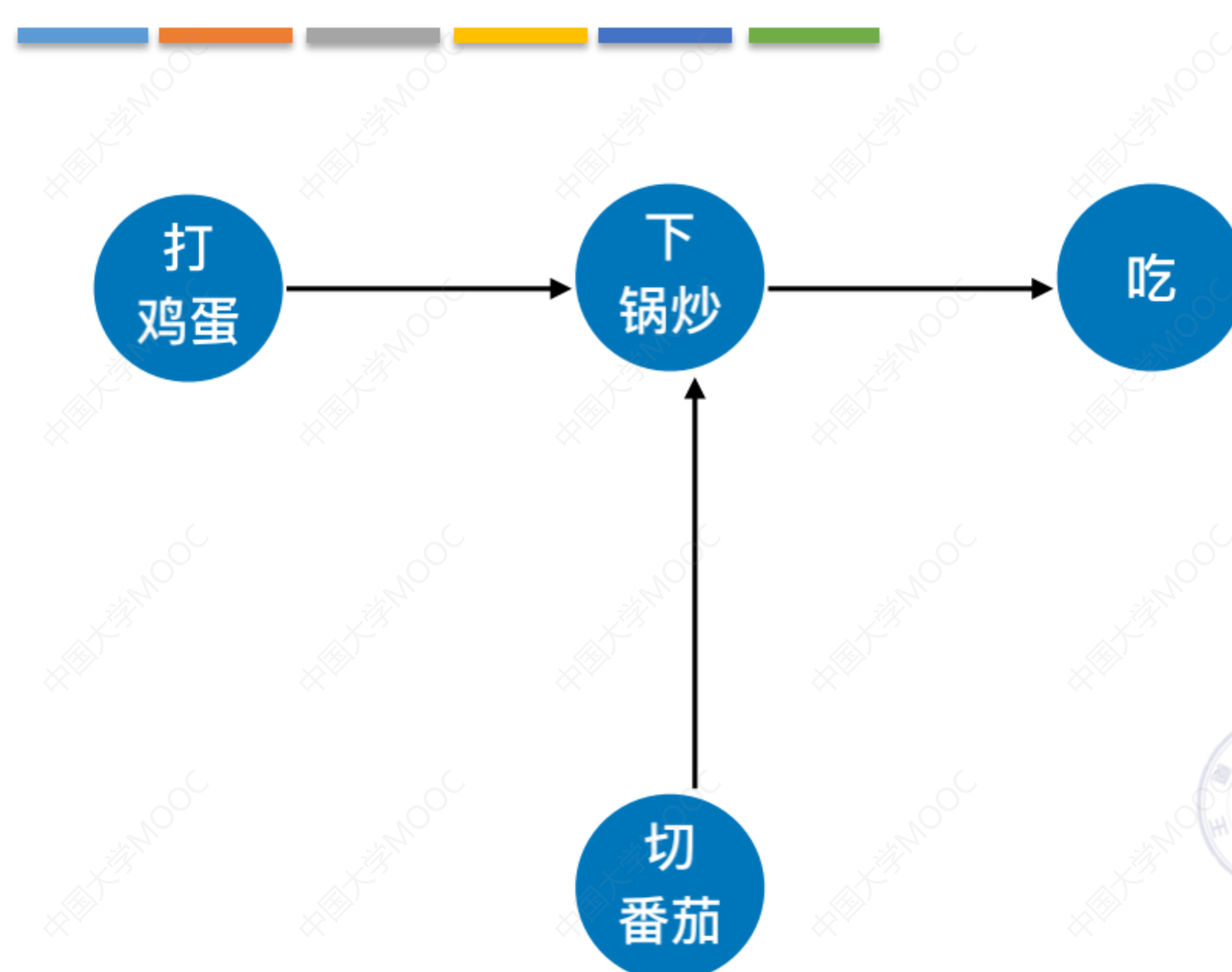


拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序

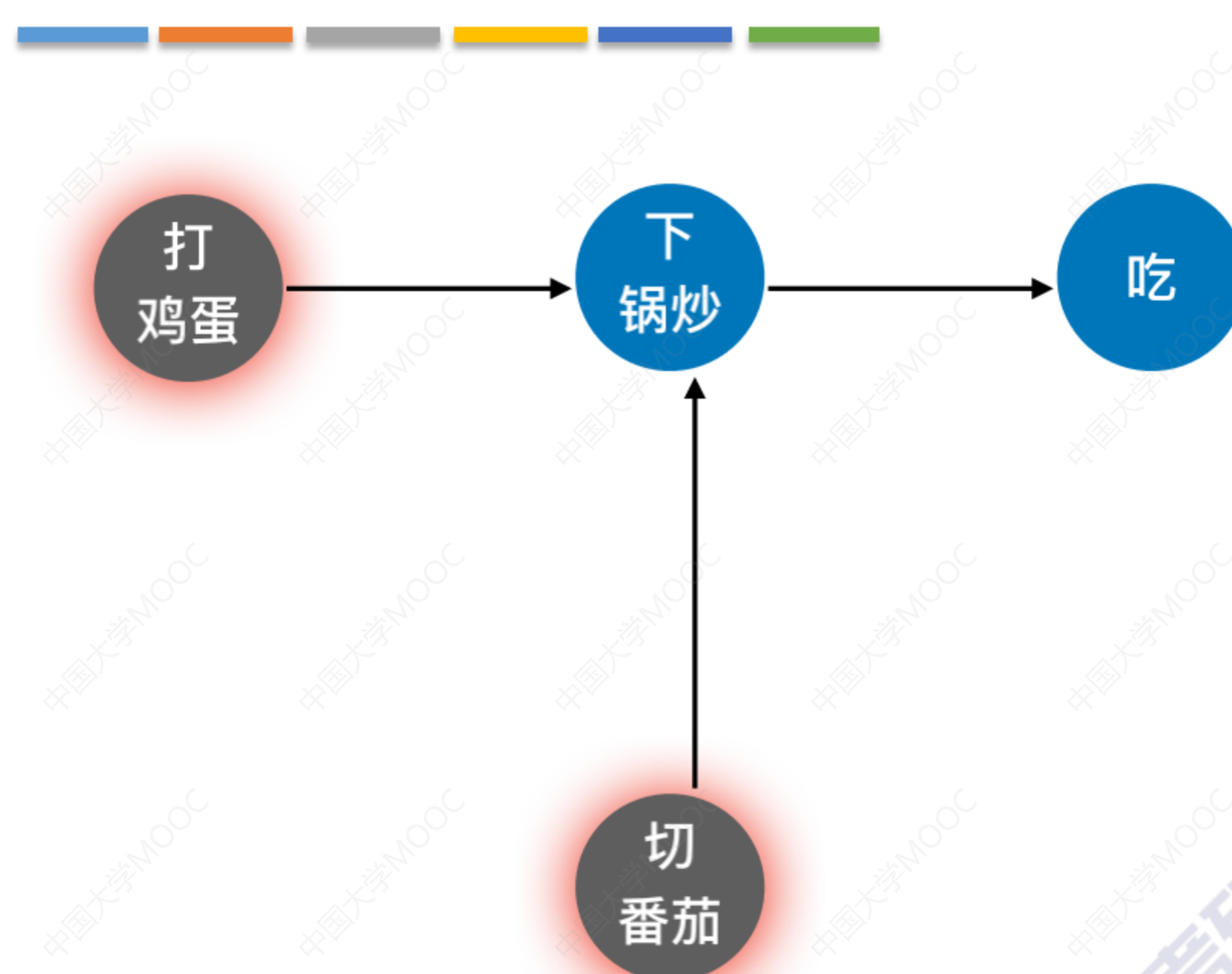


拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序

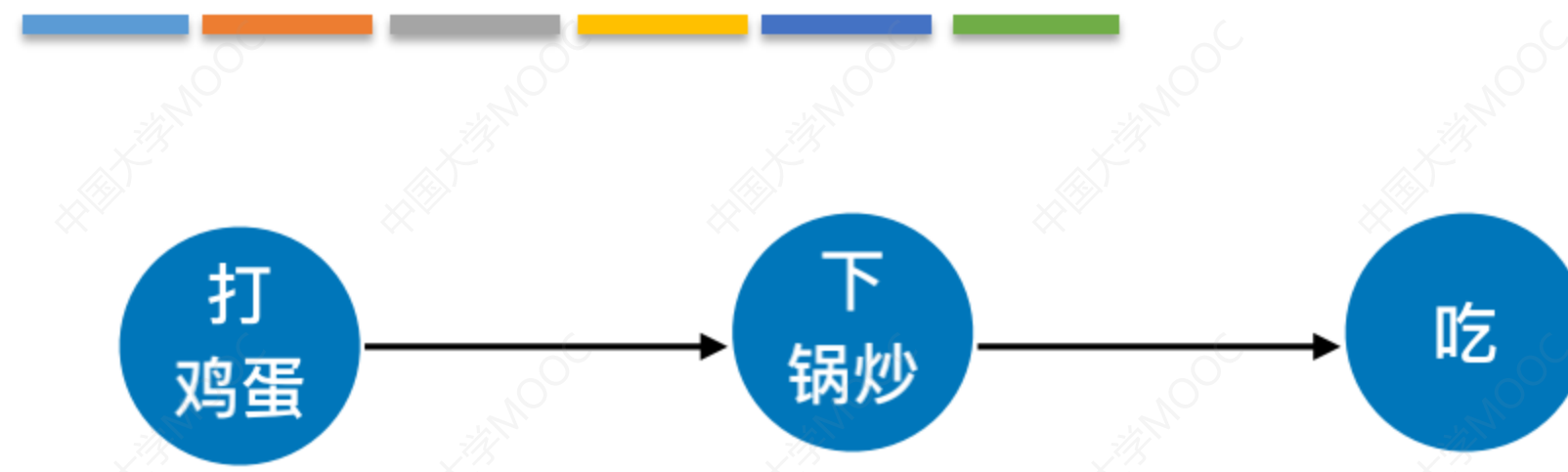


拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序

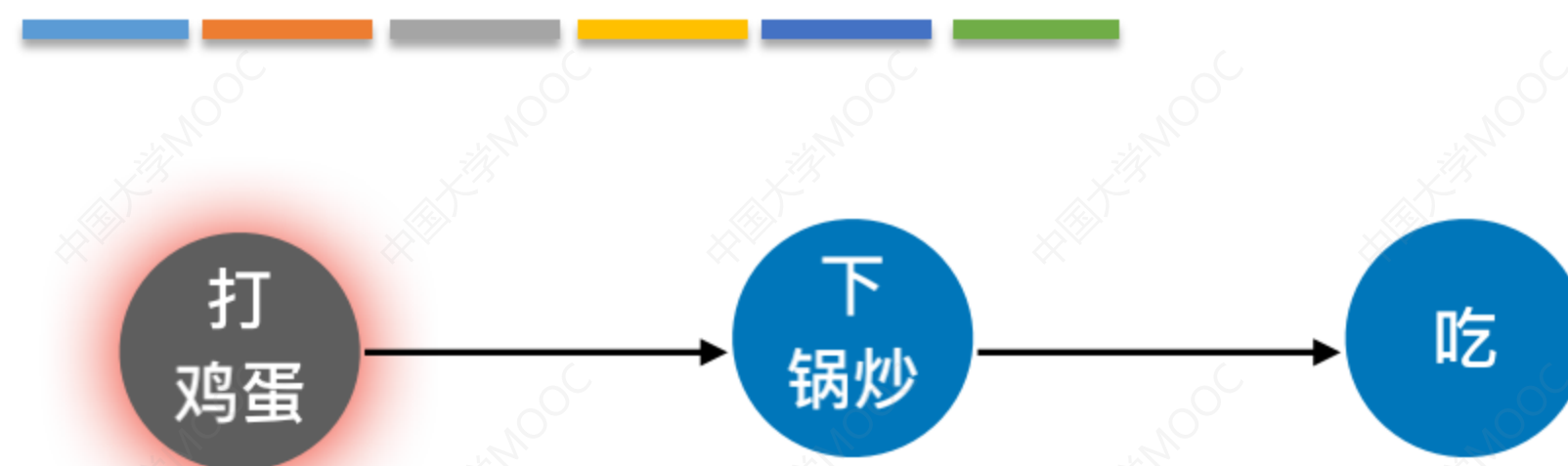


拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序



拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序



拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序



拓扑排序：找到做事的先后顺序



王道考研/CSKAOYAN.COM

## 拓扑排序



吃

拓扑排序：找到做事的先后顺序

准备  
厨具

买菜

洗  
番茄

切  
番茄

打  
鸡蛋

下  
锅炒

王道考研/CSKAOYAN.COM

## 拓扑排序



吃

拓扑排序：找到做事的先后顺序

准备  
厨具

买菜

洗  
番茄

切  
番茄

打  
鸡蛋

下  
锅炒

王道考研/CSKAOYAN.COM

## 拓扑排序

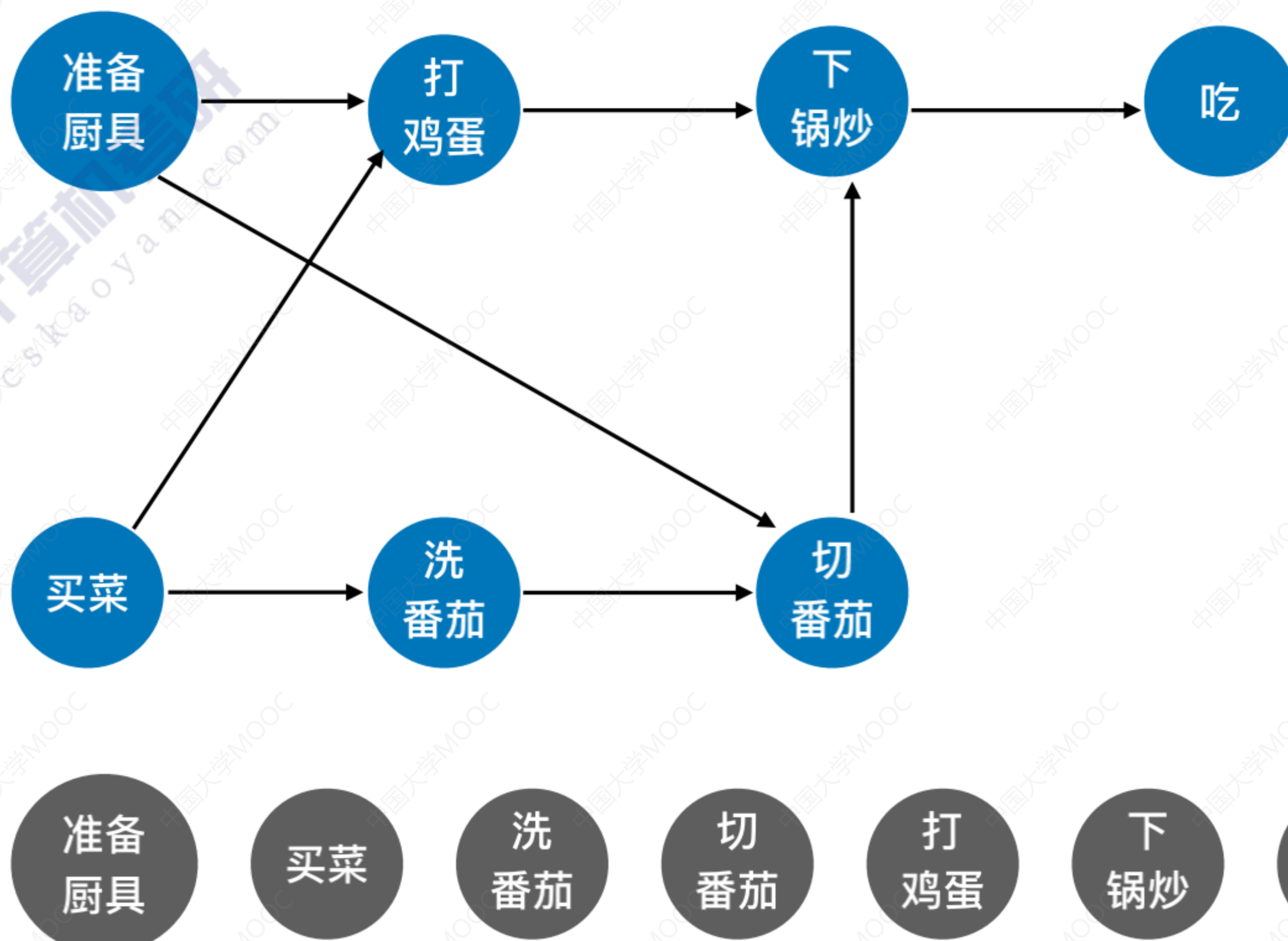
拓扑排序的实现：

- ① 从AOV网中选择一个没有前驱（入度为0）的顶点并输出。
- ② 从网中删除该顶点和所有以它为起点的有向边。
- ③ 重复①和②直到当前的AOV网为空或当前网中不存在无前驱的顶点为止。



王道考研/CSKAOYAN.COM

## 拓扑排序



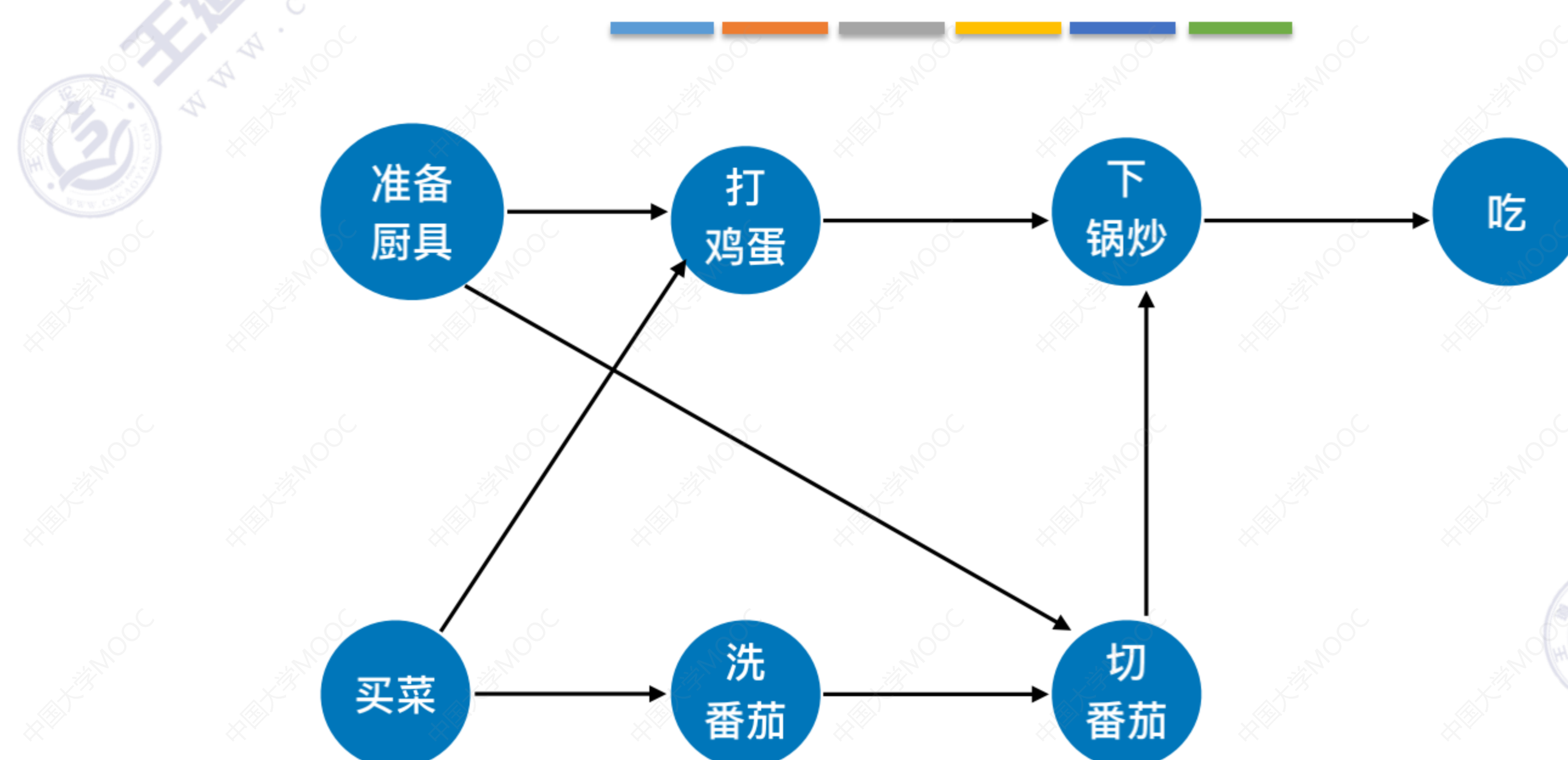
**拓扑排序**：在图论中，由一个有向无环图的顶点组成的序列，当且仅当满足下列条件时，称为该图的一个拓扑排序：

- ① 每个顶点出现且只出现一次。
- ② 若顶点A在序列中排在顶点B的前面，则在图中不存在从顶点B到顶点A的路径。

或定义为：拓扑排序是对有向无环图的顶点的一种排序，它使得若存在一条从顶点A到顶点B的路径，则在排序中顶点B出现在顶点A的后面。每个AOV网都有一个或多个**拓扑排序序列**。

王道考研/CSKAOYAN.COM

## 拓扑排序



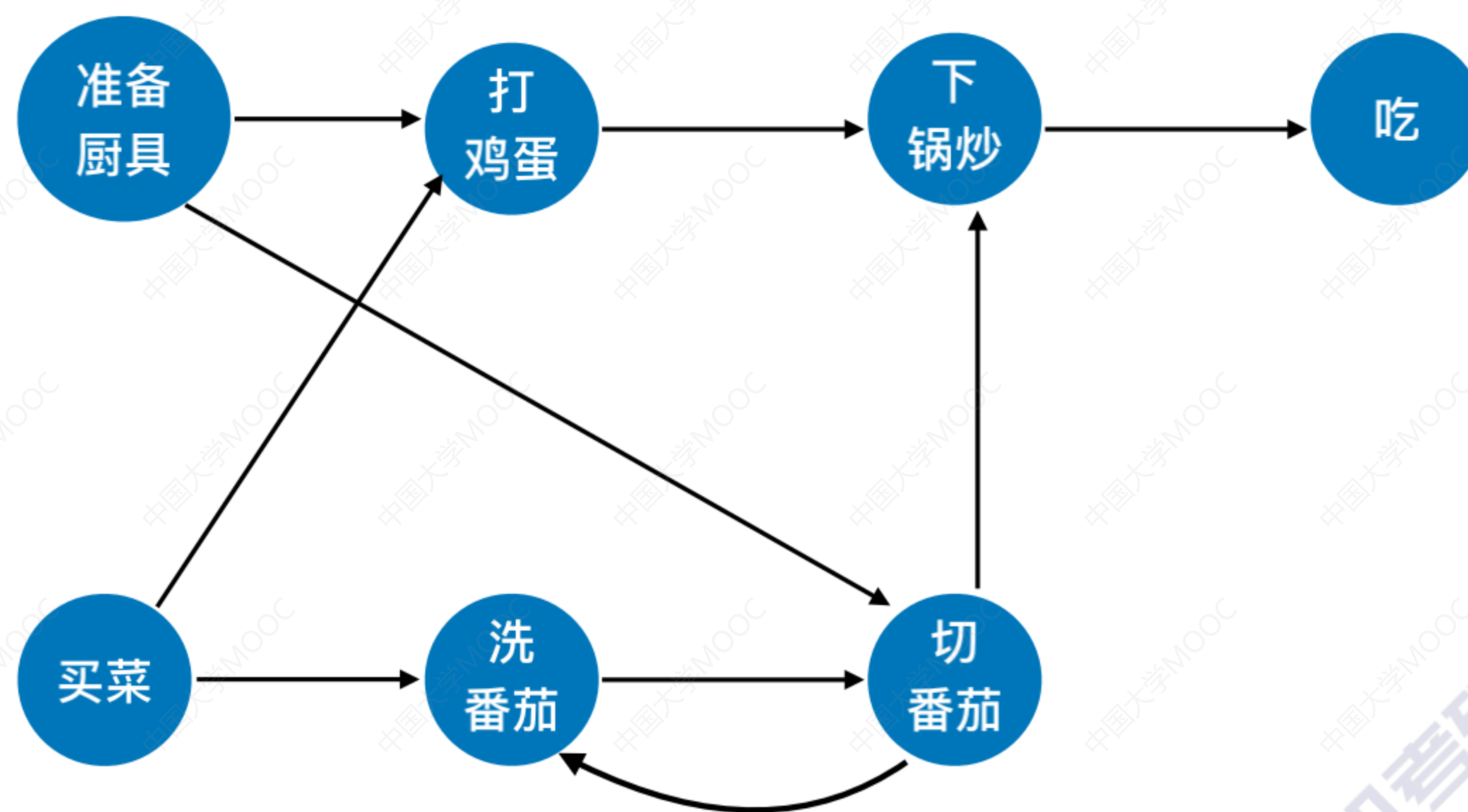
拓扑排序的实现:

- ① 从AOV网中选择一个没有前驱的顶点并输出。
- ② 从网中删除该顶点和所有以它为起点的有向边。
- ③ 重复①和②直到当前的AOV网为空或当前网中不存在无前驱的顶点为止。

说明有回路

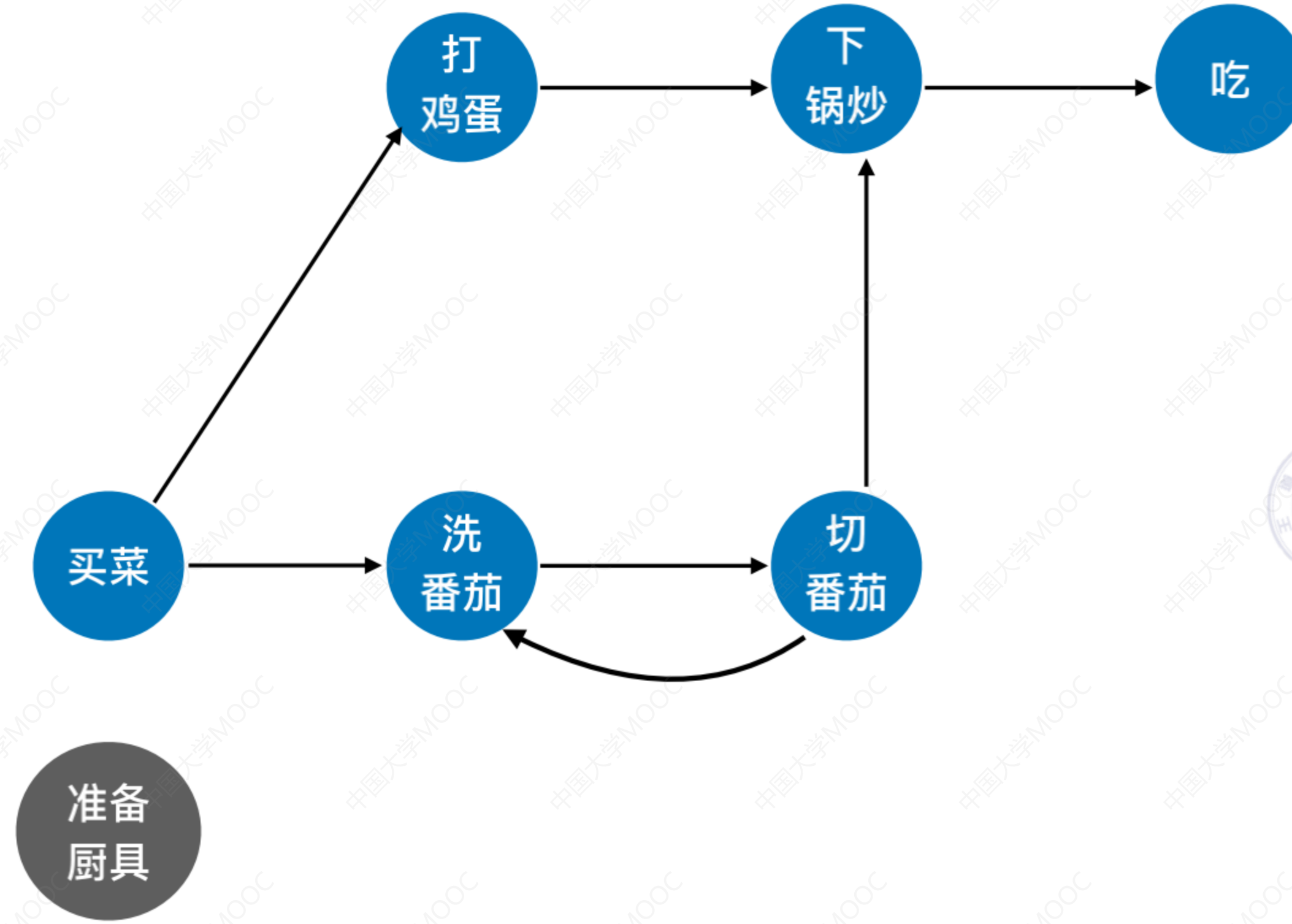
王道考研/CSKAOYAN.COM

## 对有回路的图进行拓扑排序



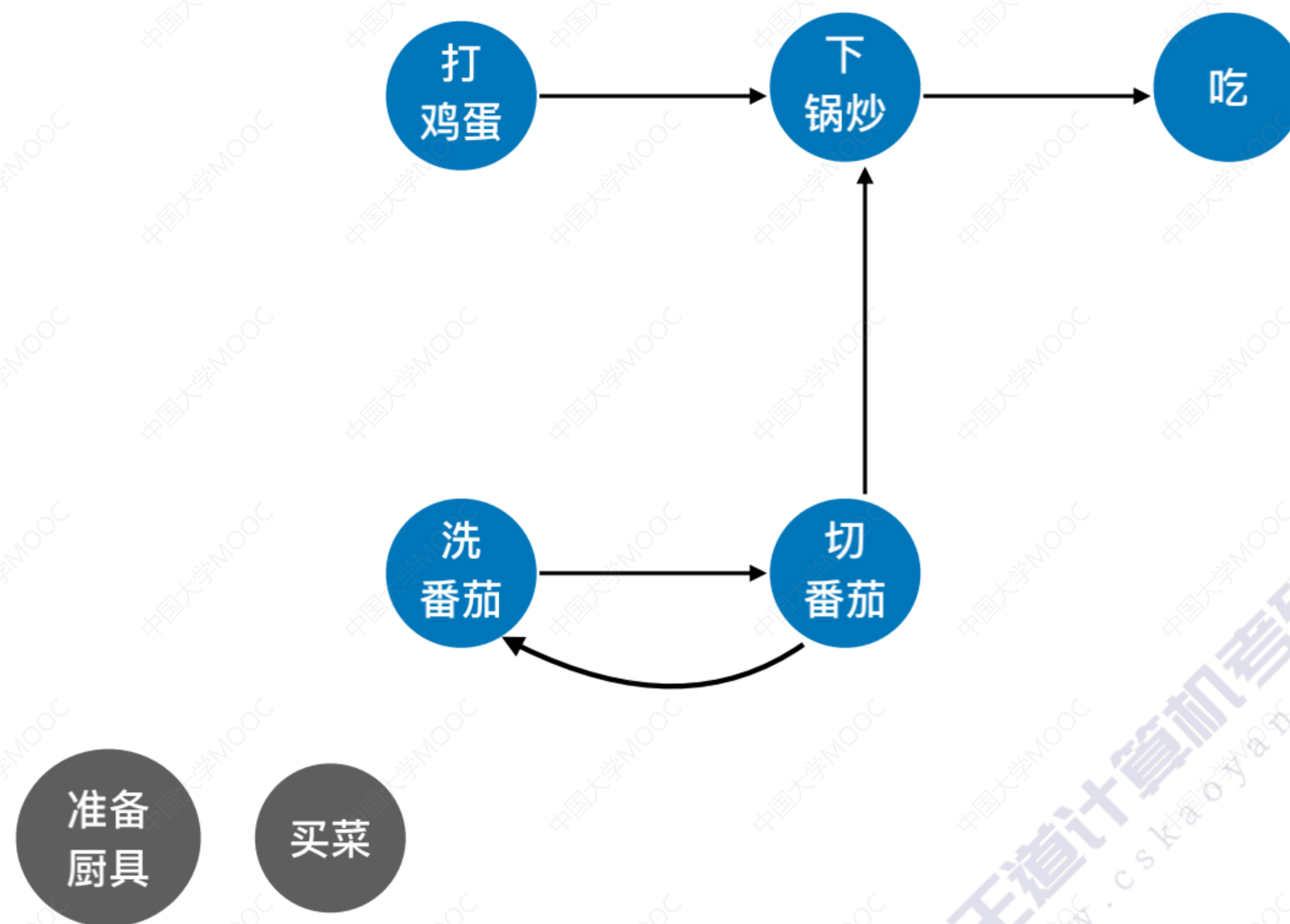
王道考研/CSKAOYAN.COM

## 对有回路的图进行拓扑排序



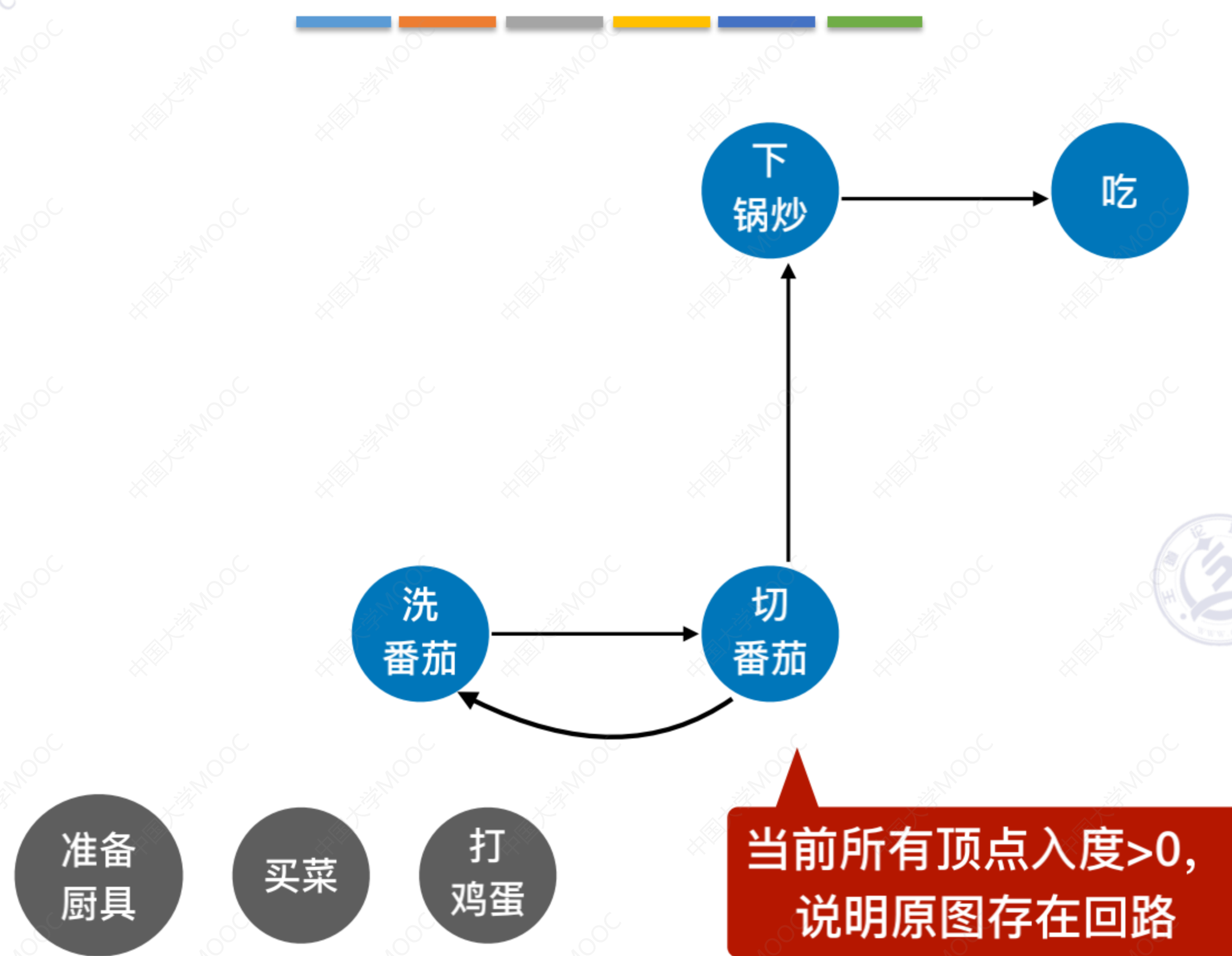
王道考研/CSKAOYAN.COM

## 对有回路的图进行拓扑排序



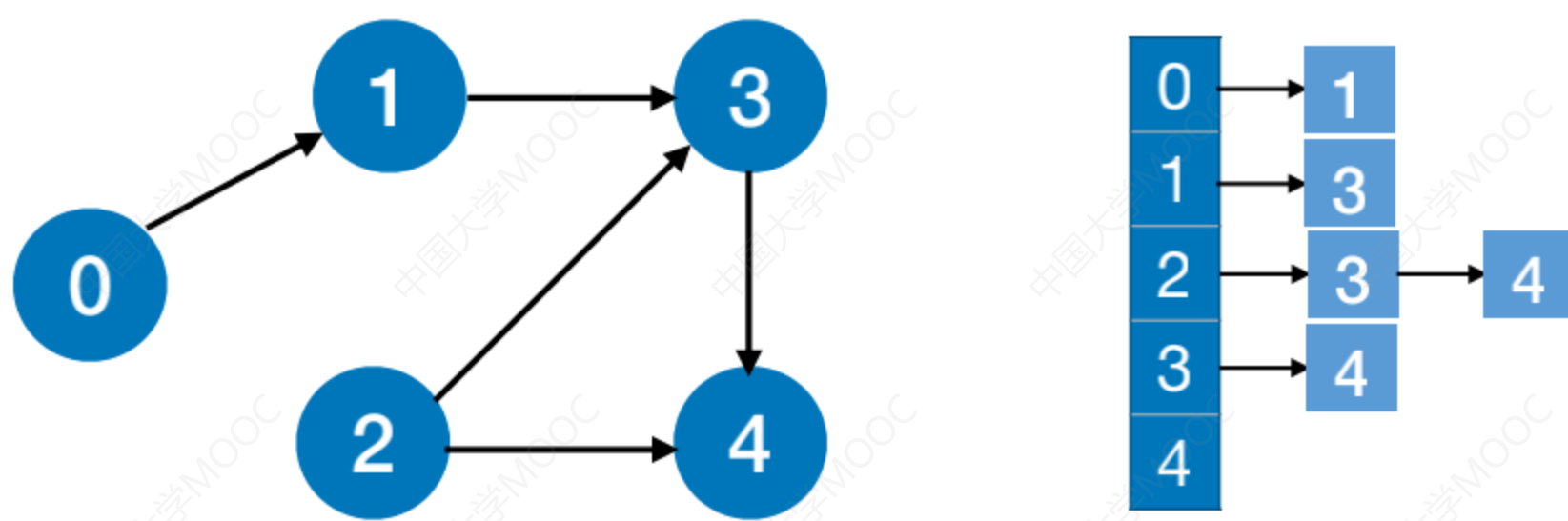
王道考研/CSKAOYAN.COM

## 对有回路的图进行拓扑排序



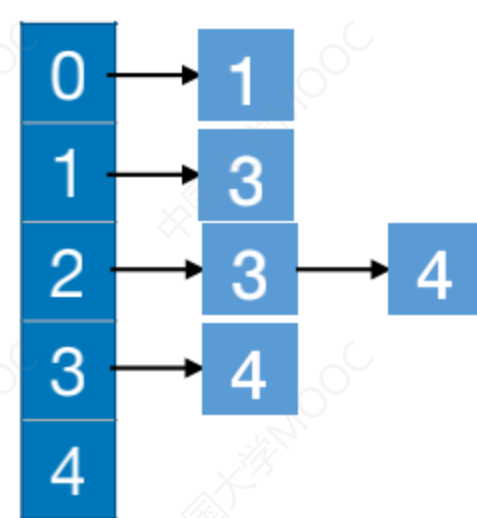
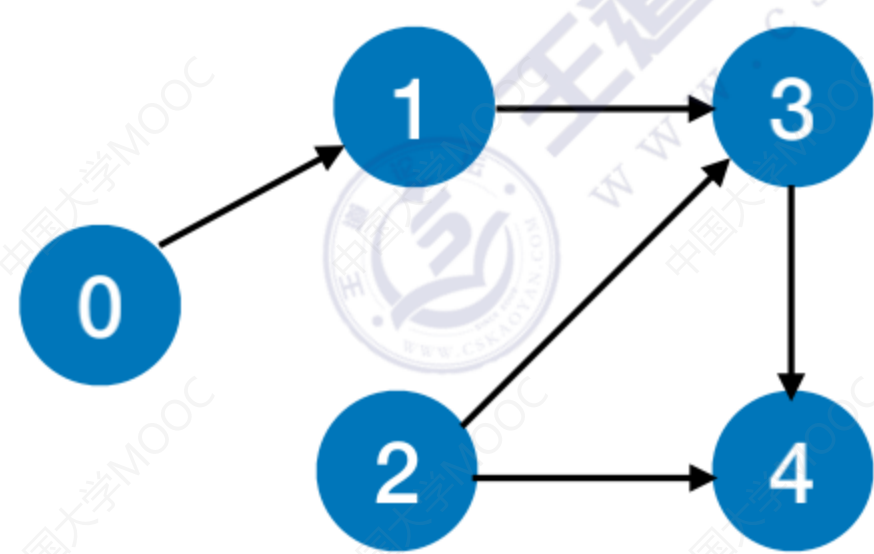
王道考研/CSKAOYAN.COM

```
#define MaxVertexNum 100 //图中顶点数目的最大值
typedef struct ArcNode{ //边表结点
    int adjvex; //该弧所指向的顶点的位置
    struct ArcNode *nextarc; //指向下一条弧的指针
    //InfoType info; //网的边权值
}ArcNode;
typedef struct VNode{ //顶点表结点
    VertexType data; //顶点信息
    ArcNode *firstarc; //指向第一条依附该顶点的弧的指针
}VNode, AdjList[MaxVertexNum];
typedef struct{
    AdjList vertices; //邻接表
    int vexnum, arcnum; //图的顶点数和弧数
} Graph; //Graph是以邻接表存储的图类型
```



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0; i<G.vexnum; i++){
        if(indegree[i]==0)
            Push(S, i); //将所有入度为0的顶点进栈
    }
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S, i); //栈顶元素出栈
        print[count++] = i; //输出顶点i
        for(p=G.vertices[i].firstarc; p; p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈S
            v=p->adjvex;
            if(--indegree[v]==0)
                Push(S, v); //入度为0, 则入栈
        }
    }
    if(count < G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

0
1
0
2
2

当前顶点入度

print[]

-1
-1
-1
-1
-1

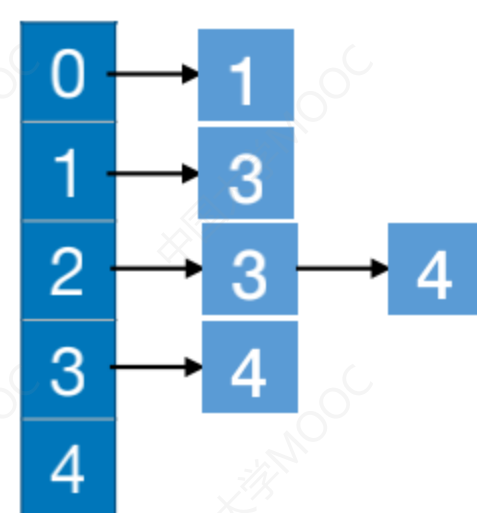
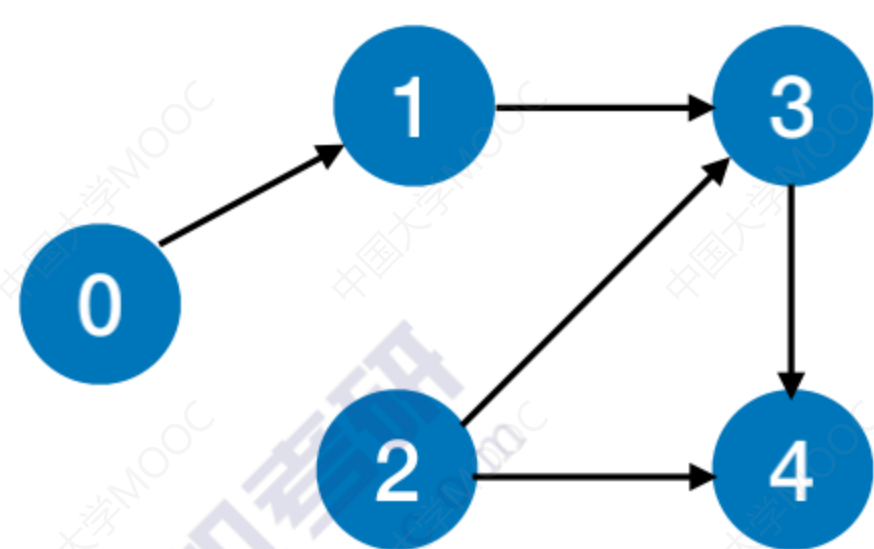
记录拓扑序列



保存度为0的顶点 (也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

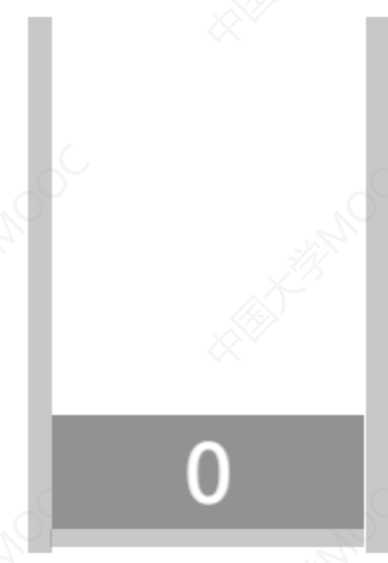
0
1
0
2
2

当前顶点入度

print[]

-1
-1
-1
-1
-1

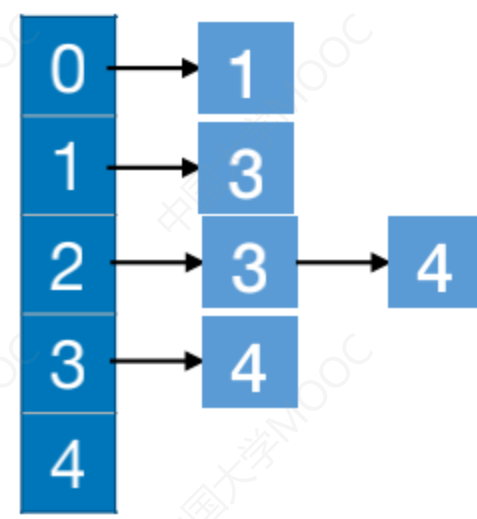
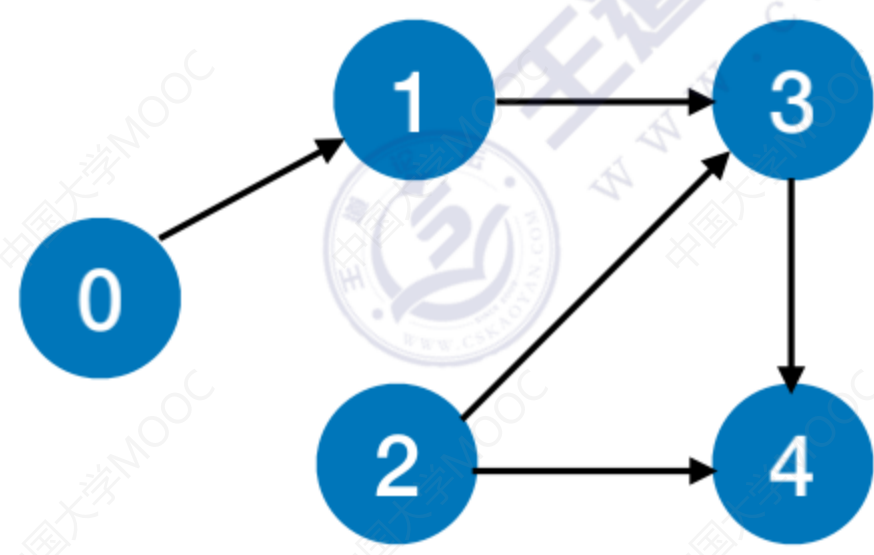
记录拓扑序列



保存度为0的顶点 (也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

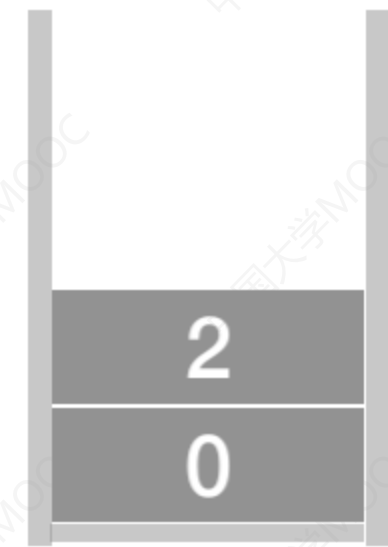
0
1
0
2
2

当前顶点入度

print[ ]

-1
-1
-1
-1
-1

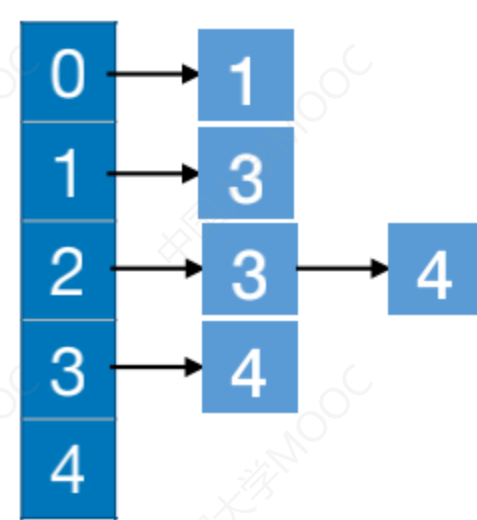
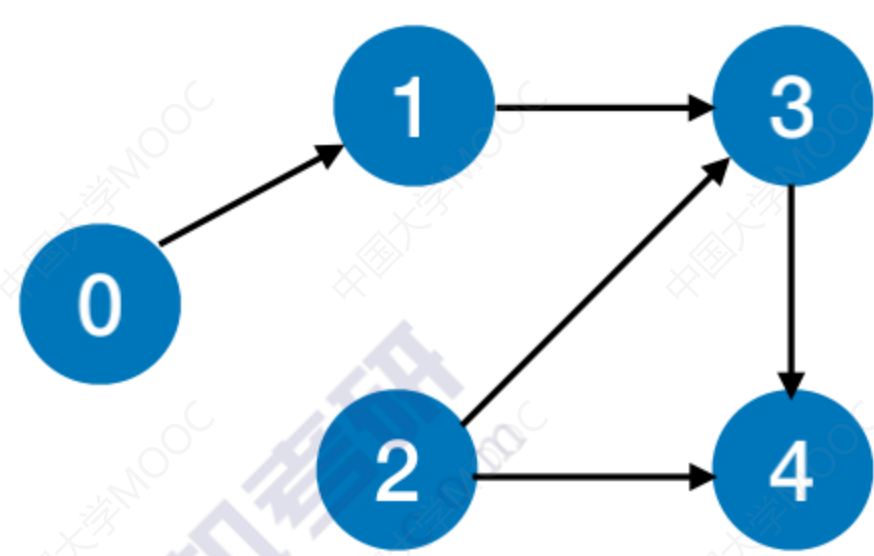
记录拓扑序列



保存度为0的顶点 (也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

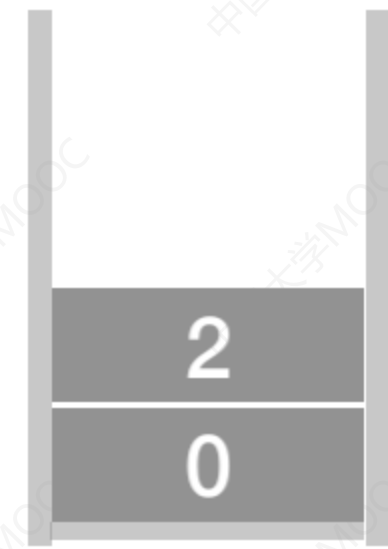
0
1
0
2
2

当前顶点入度

print[ ]

-1
-1
-1
-1
-1

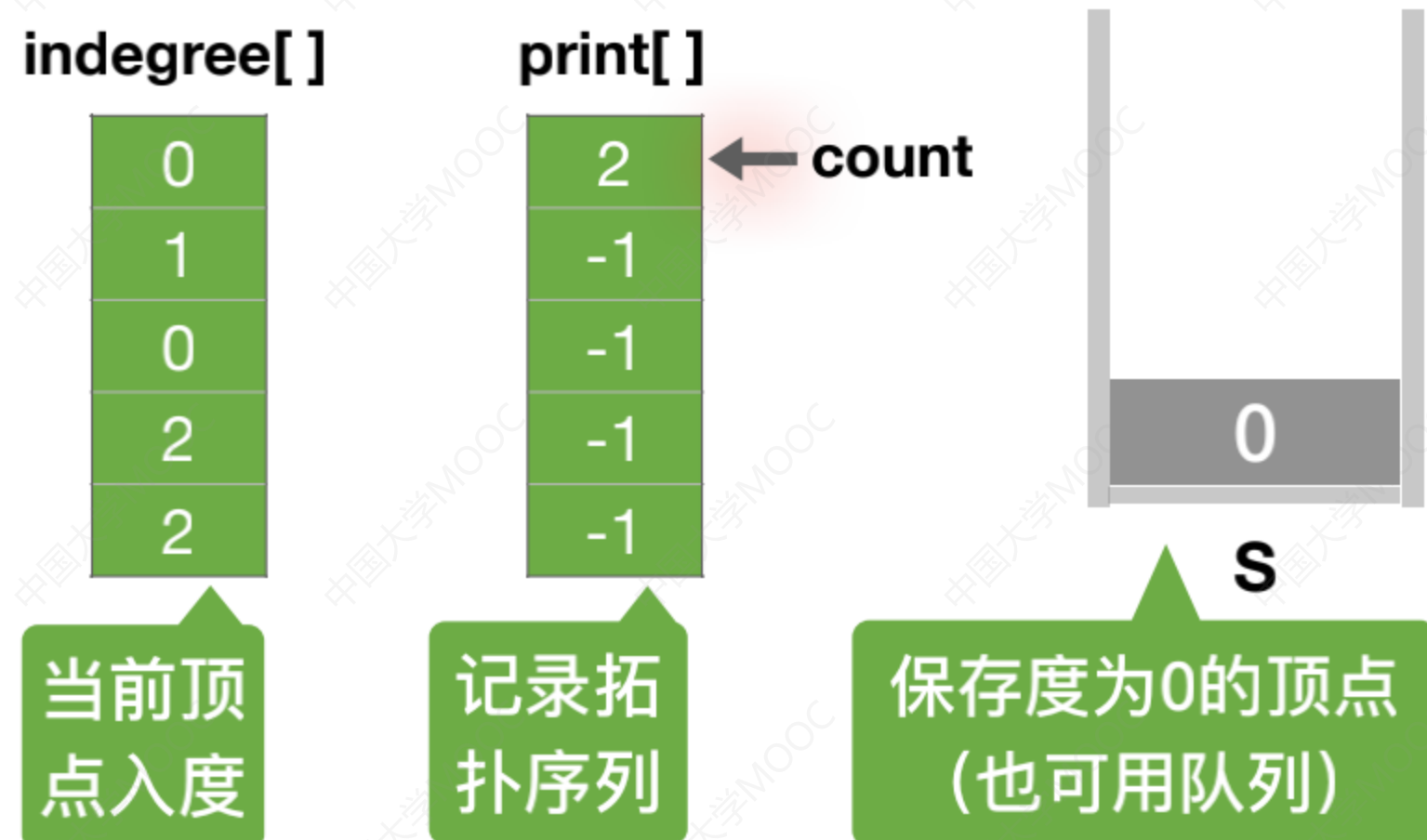
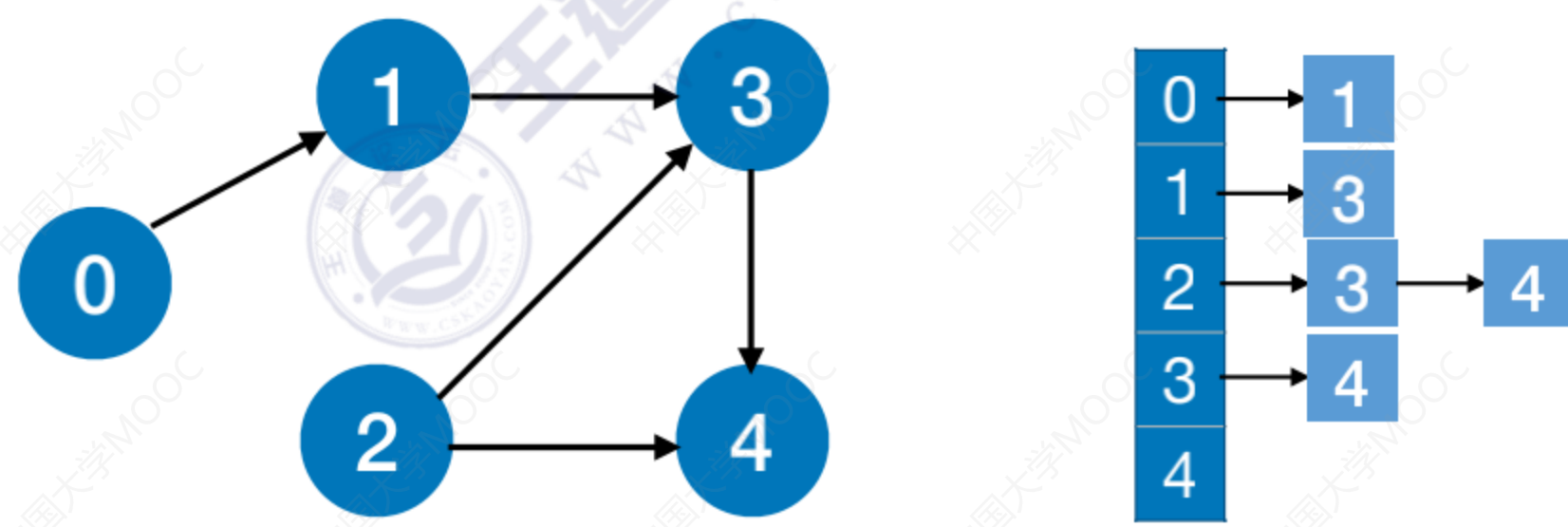
记录拓扑序列



保存度为0的顶点 (也可用队列)

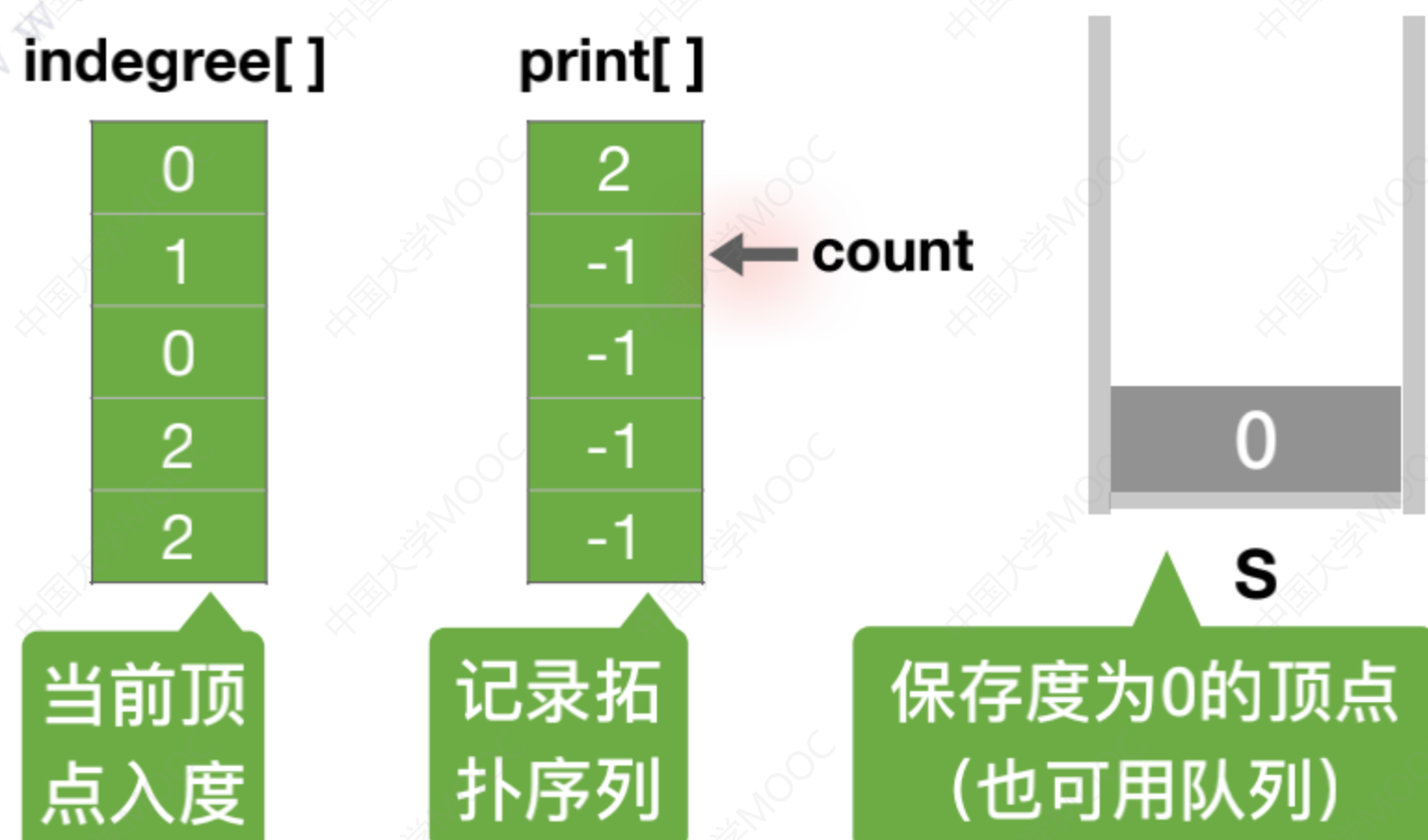
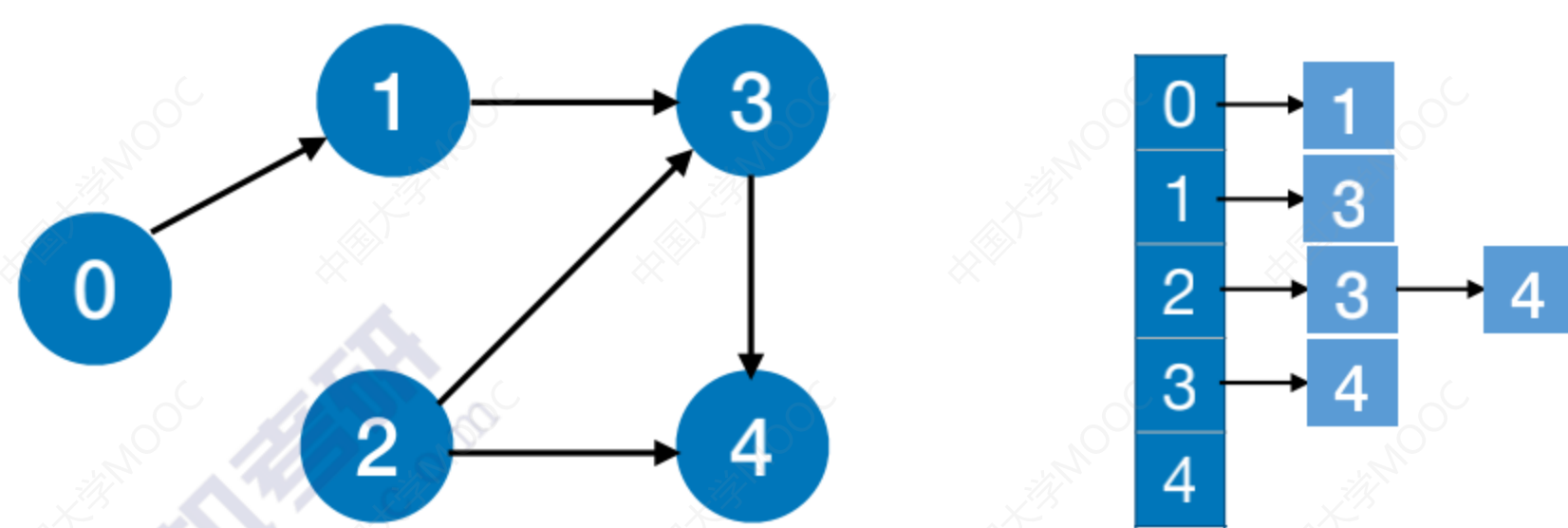
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



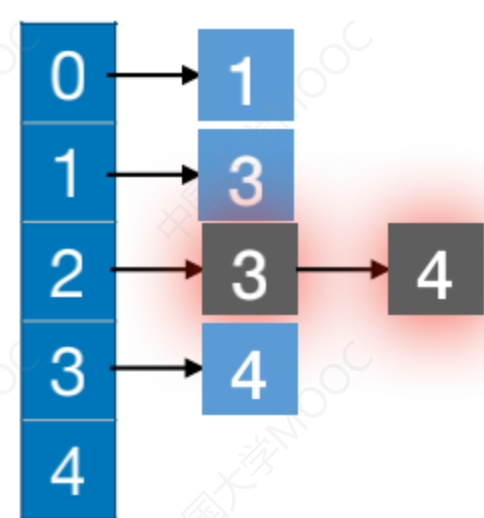
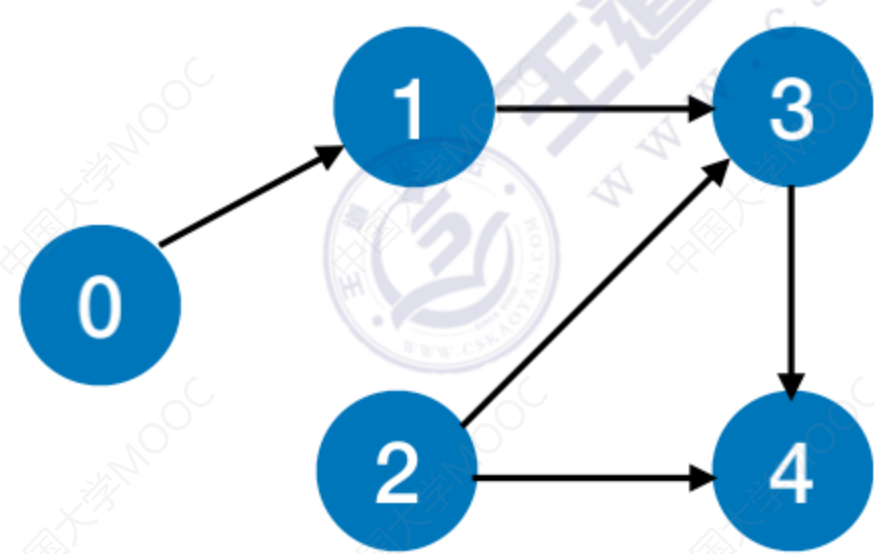
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

0
1
0
2
2

当前顶点入度

print[]

2
-1
-1
-1
-1

记录拓扑序列

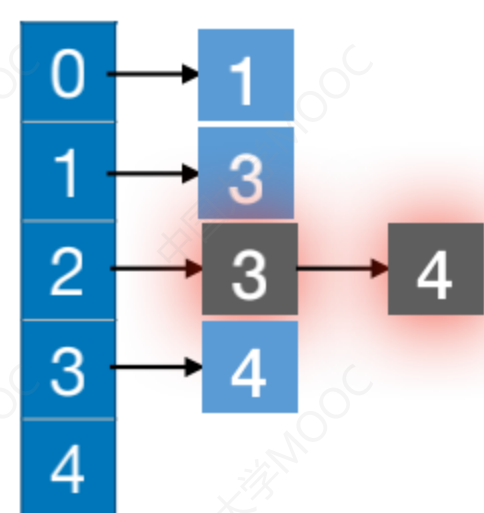
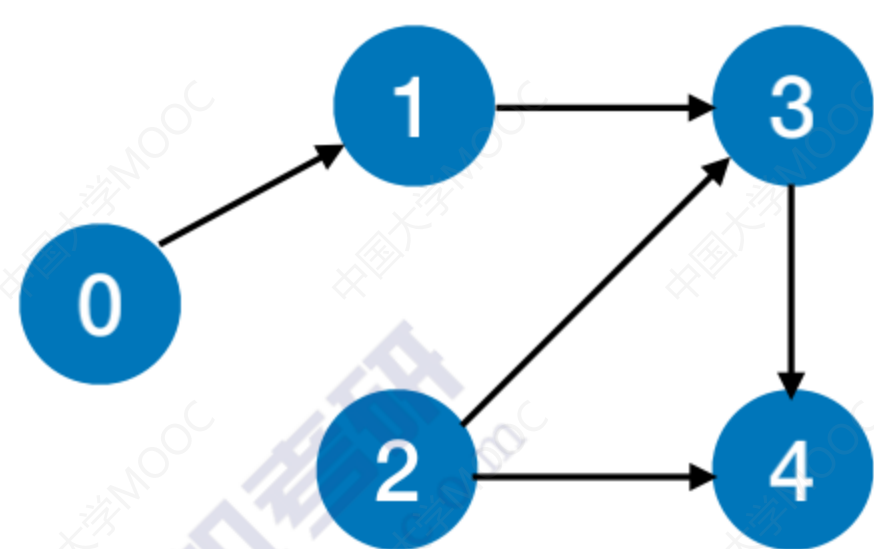
← count



保存度为0的顶点 (也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

0
1
0
1
1

当前顶点入度

print[]

2
-1
-1
-1
-1

记录拓扑序列

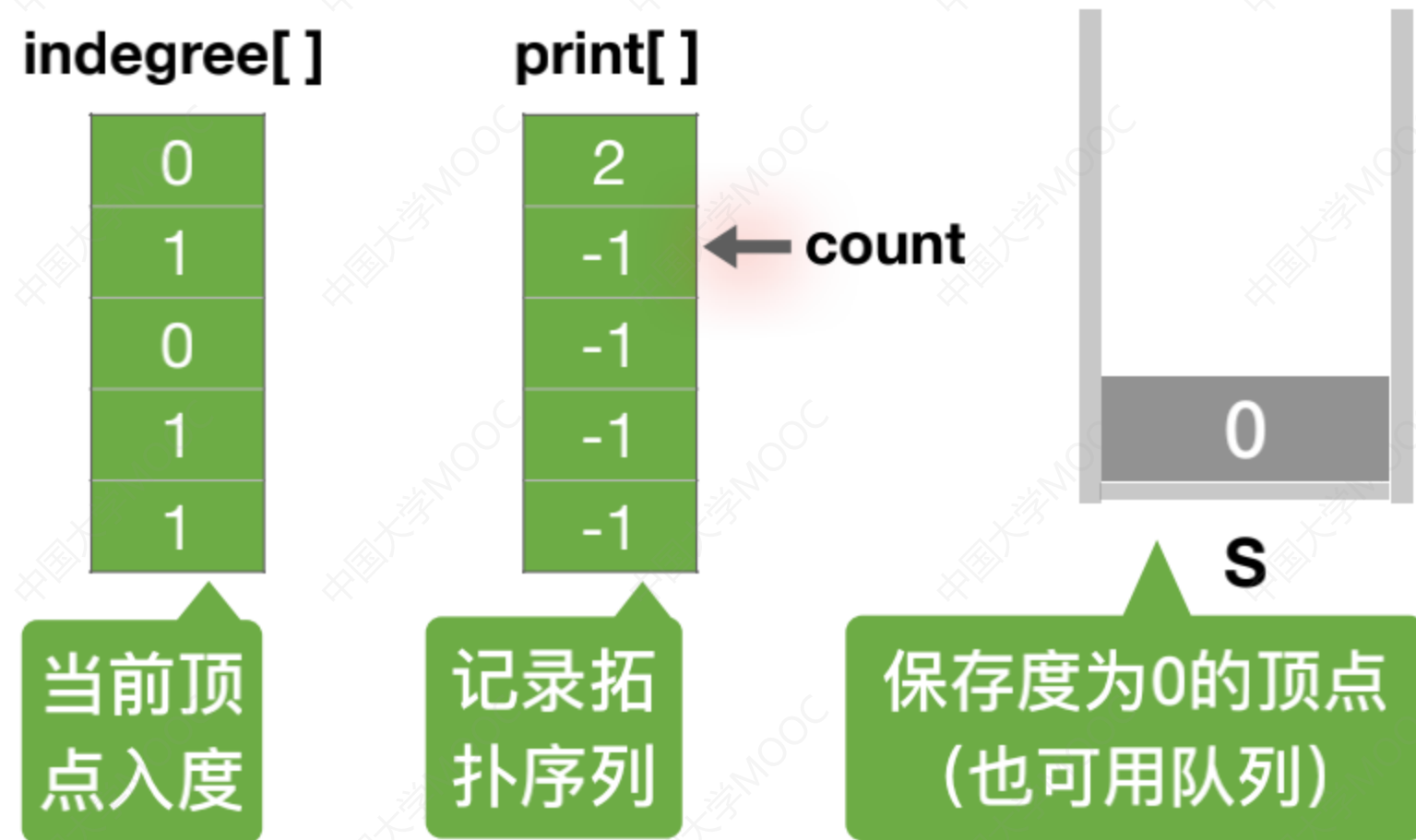
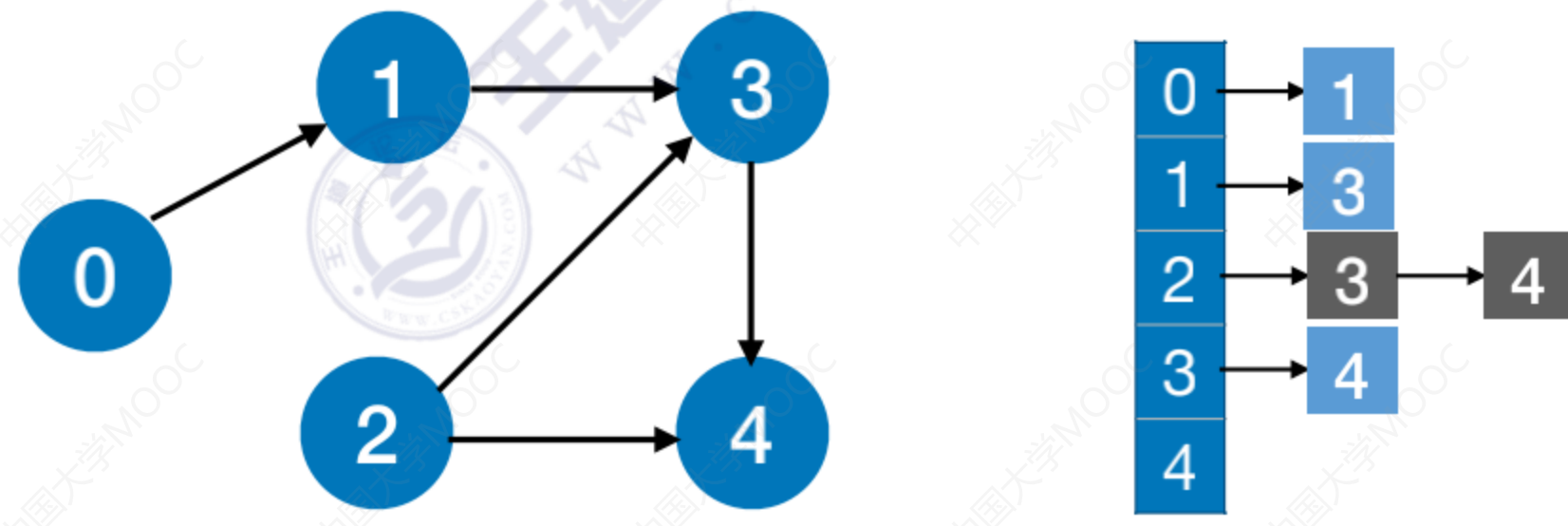
← count



保存度为0的顶点 (也可用队列)

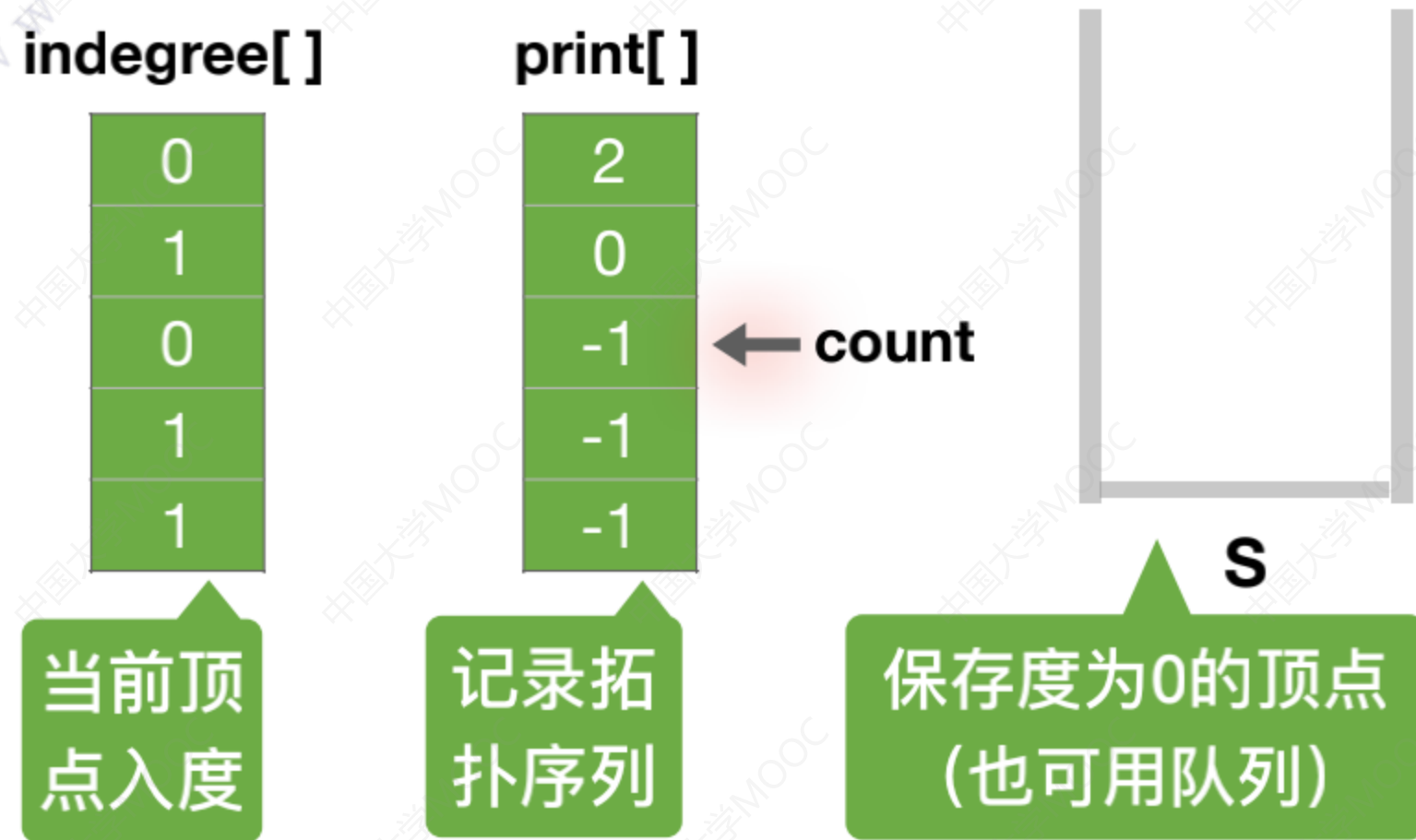
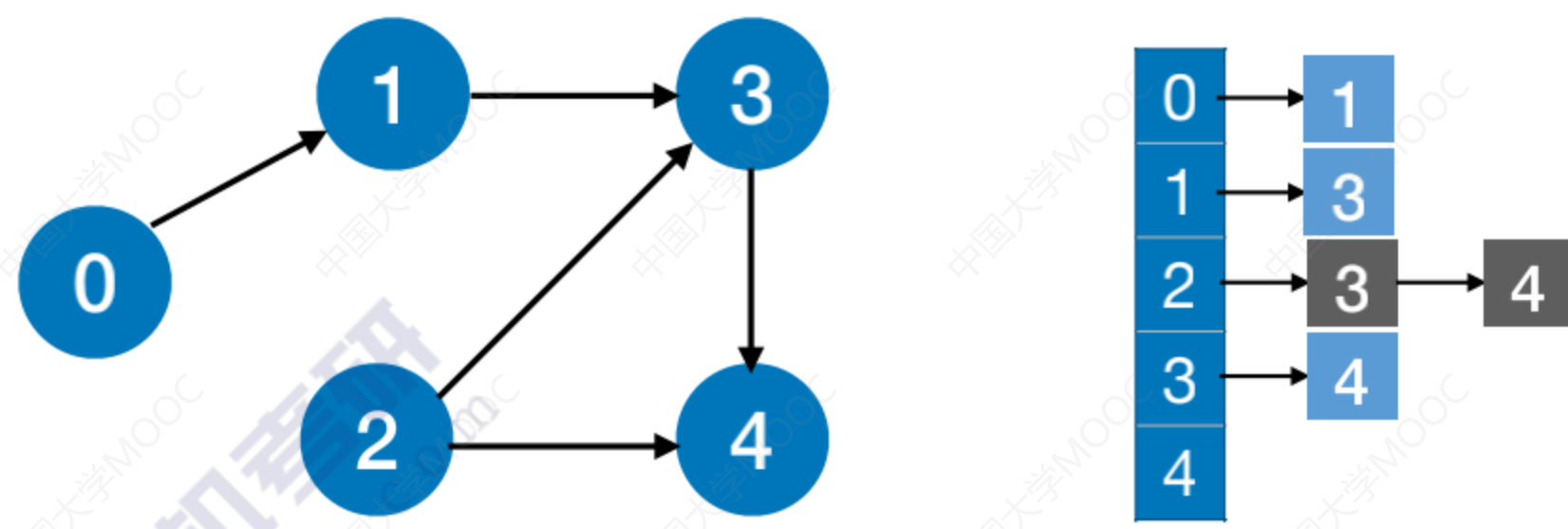
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



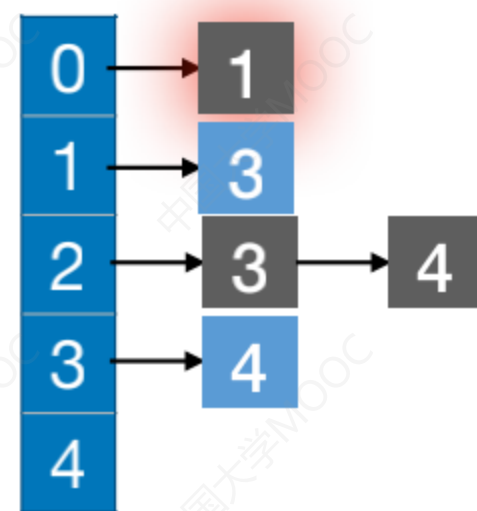
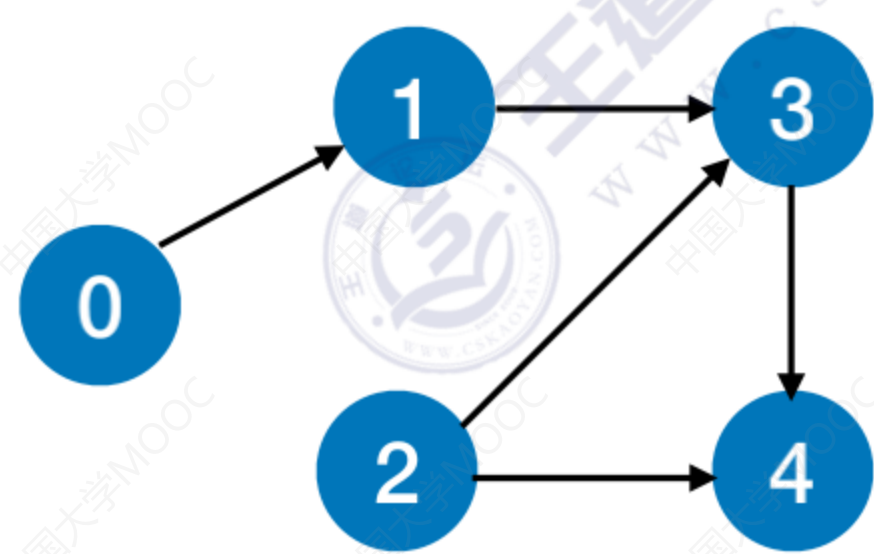
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

0
1
0
1
1

当前顶点入度

print[]

2
0
-1
-1
-1

记录拓扑序列

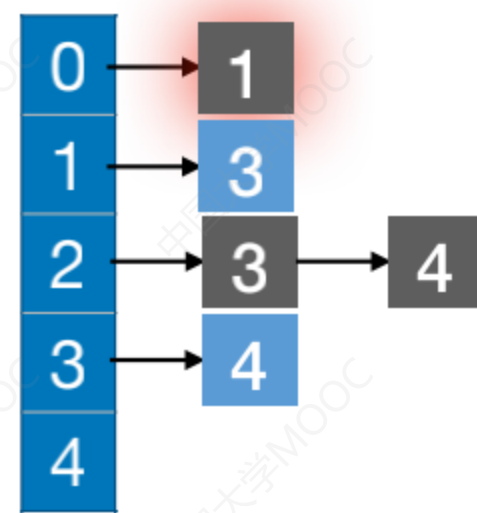
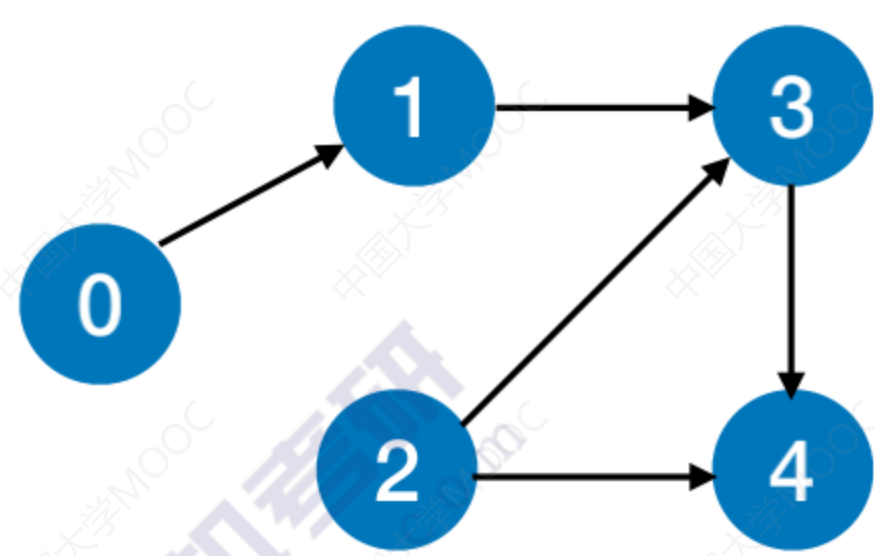
← count



保存度为0的顶点 (也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[]

0
0
0
1
1

当前顶点入度

print[]

2
0
-1
-1
-1

记录拓扑序列

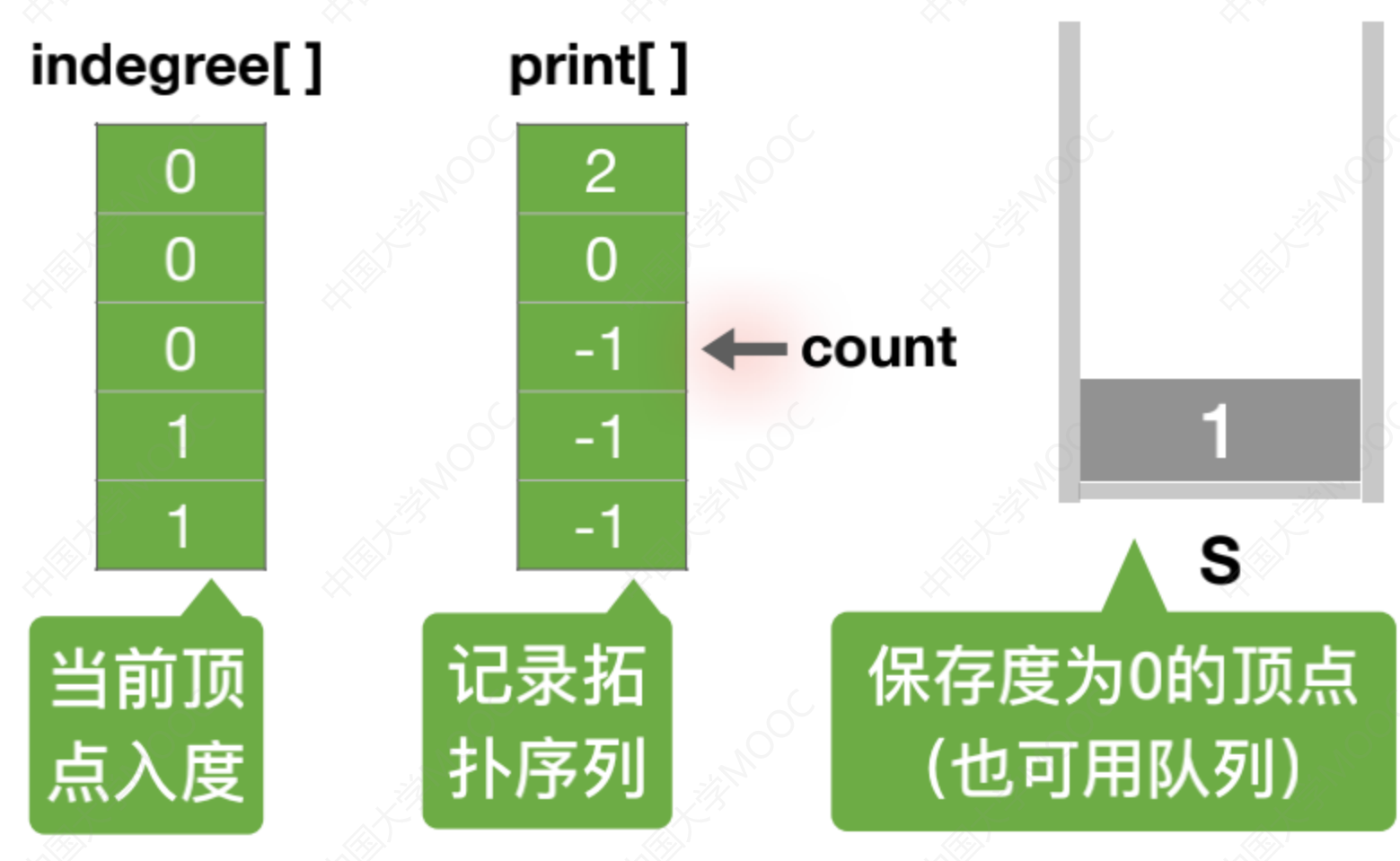
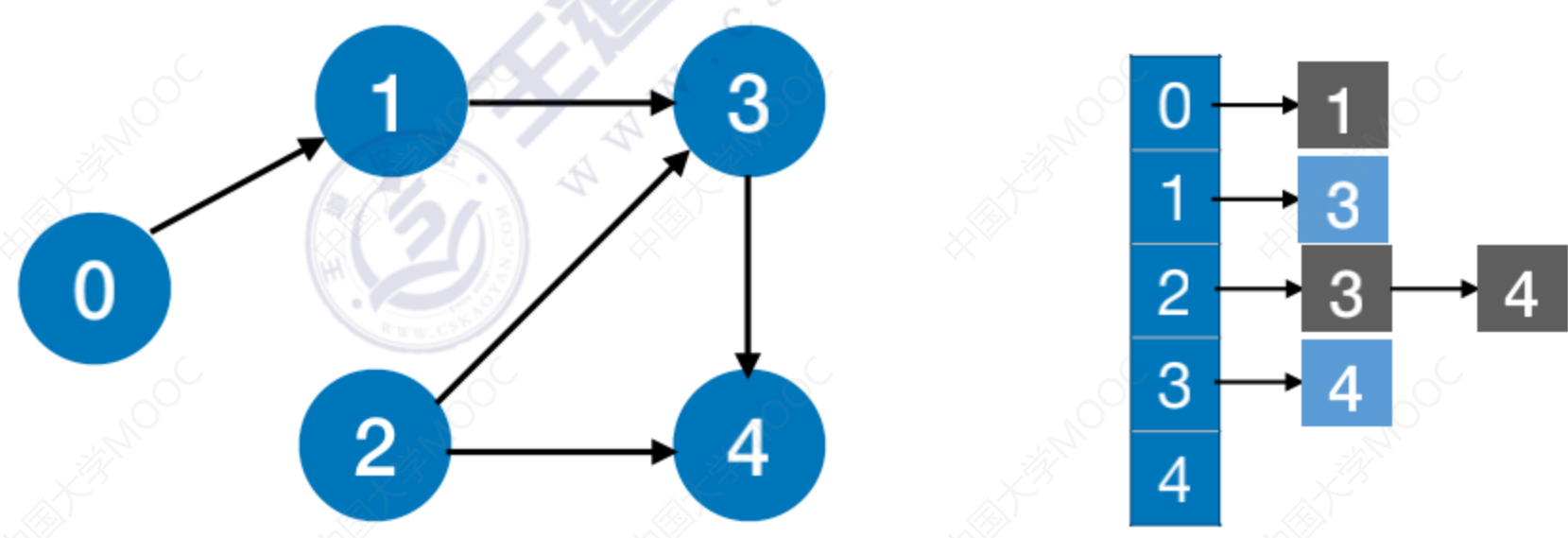
← count



保存度为0的顶点 (也可用队列)

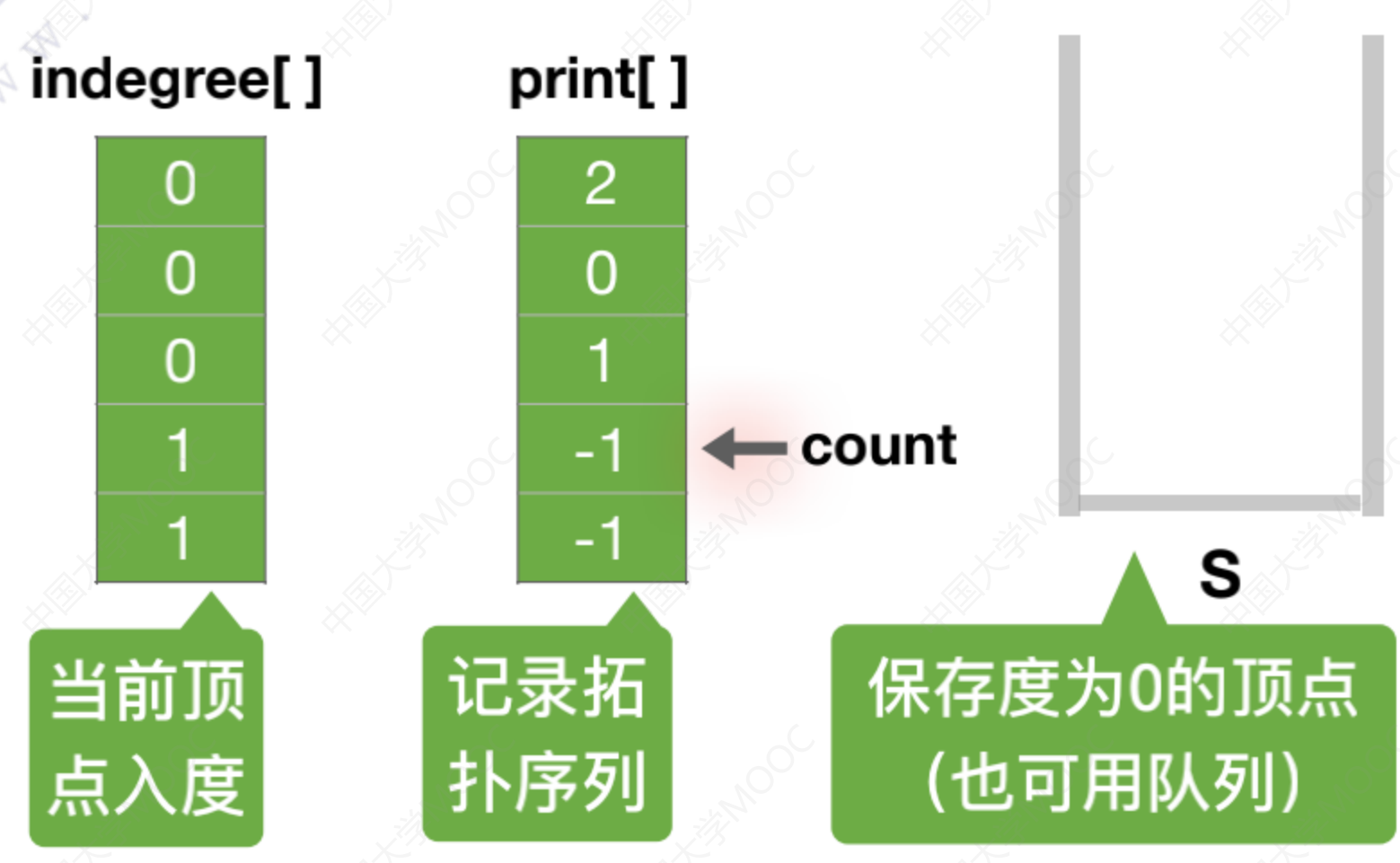
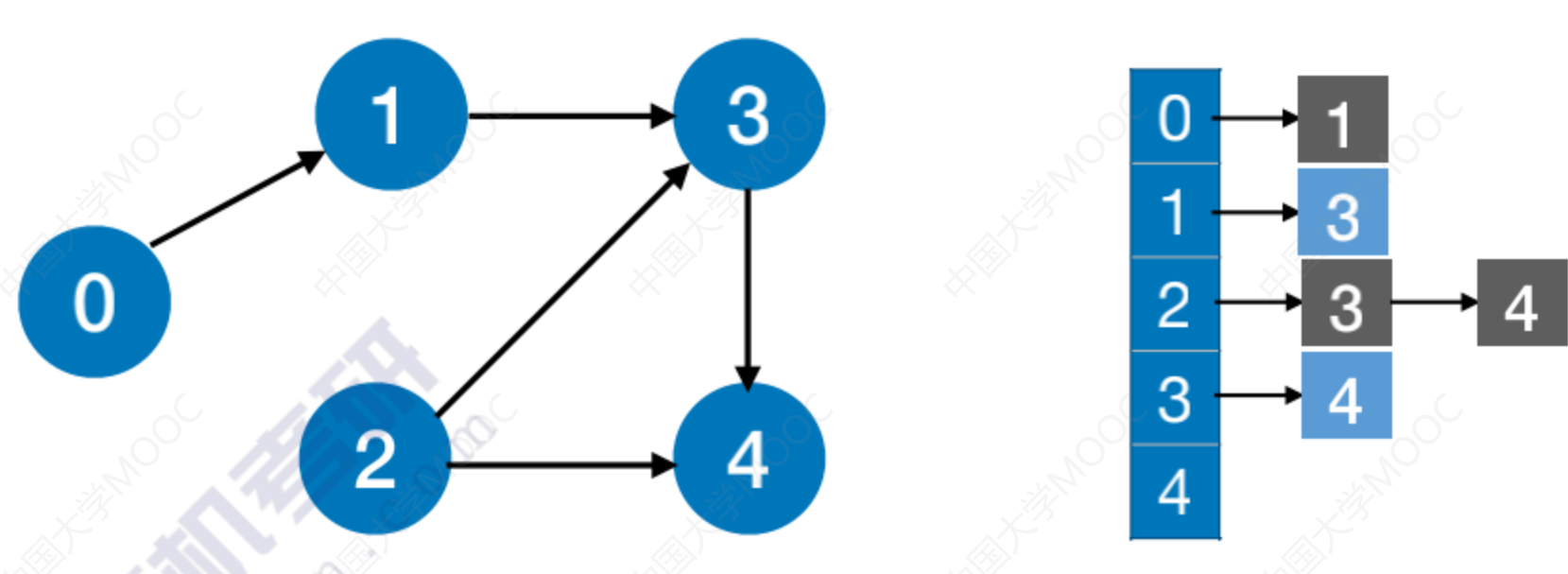
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



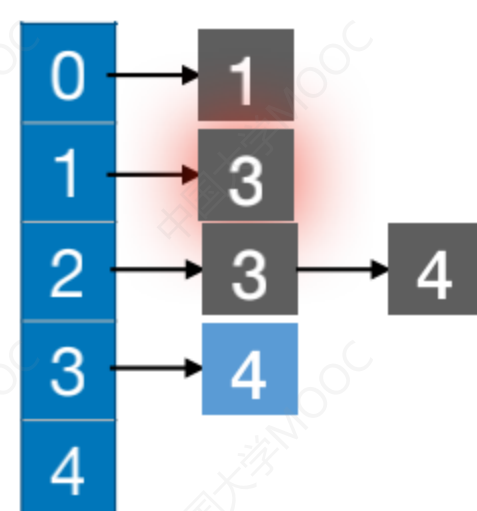
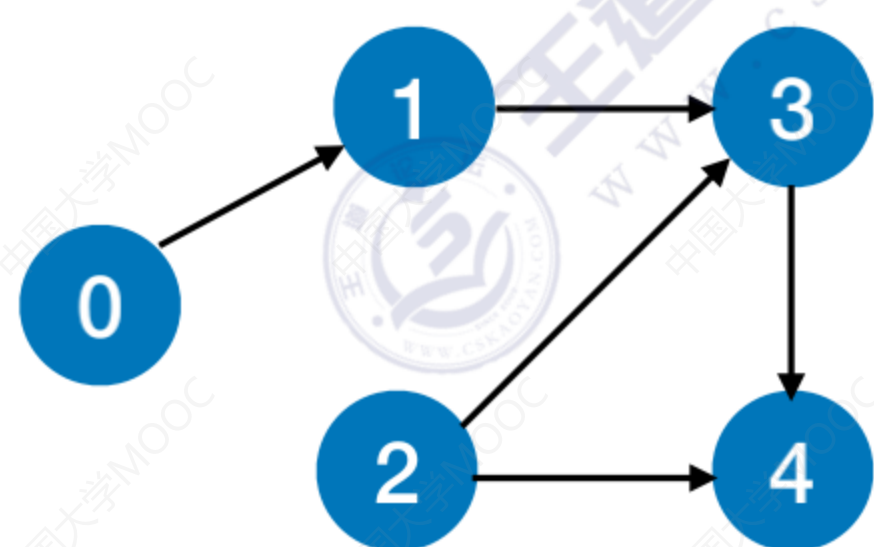
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    } //while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    } //while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

0  
0  
0  
1  
1

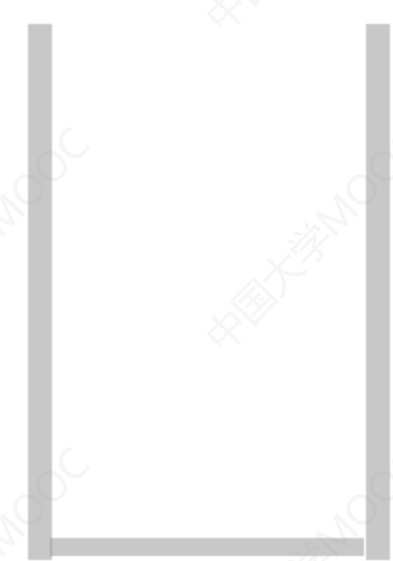
当前顶  
点入度

print[ ]

2  
0  
1  
-1  
-1

记录拓  
扑序列

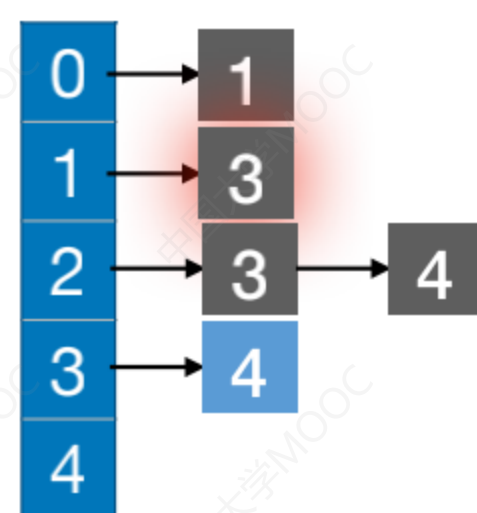
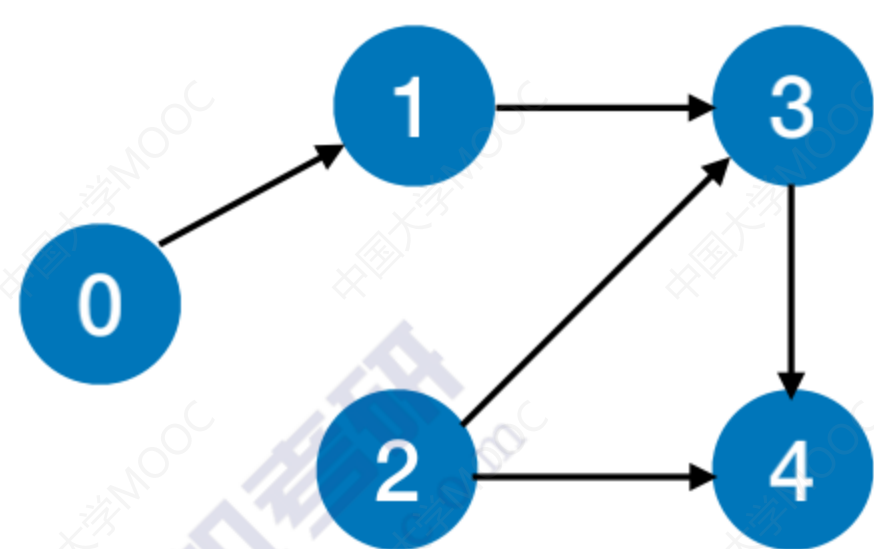
← count



保存度为0的顶点  
(也可用队列)

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

0  
0  
0  
0  
1

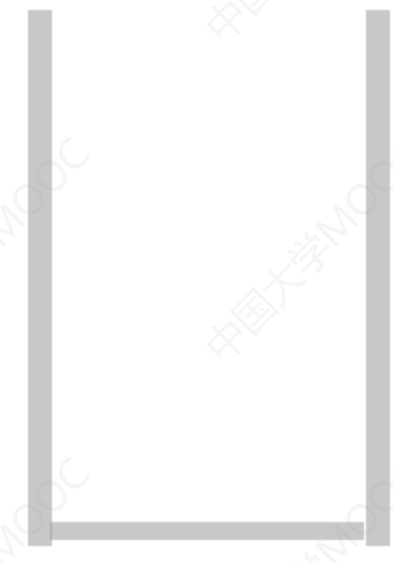
当前顶  
点入度

print[ ]

2  
0  
1  
-1  
-1

记录拓  
扑序列

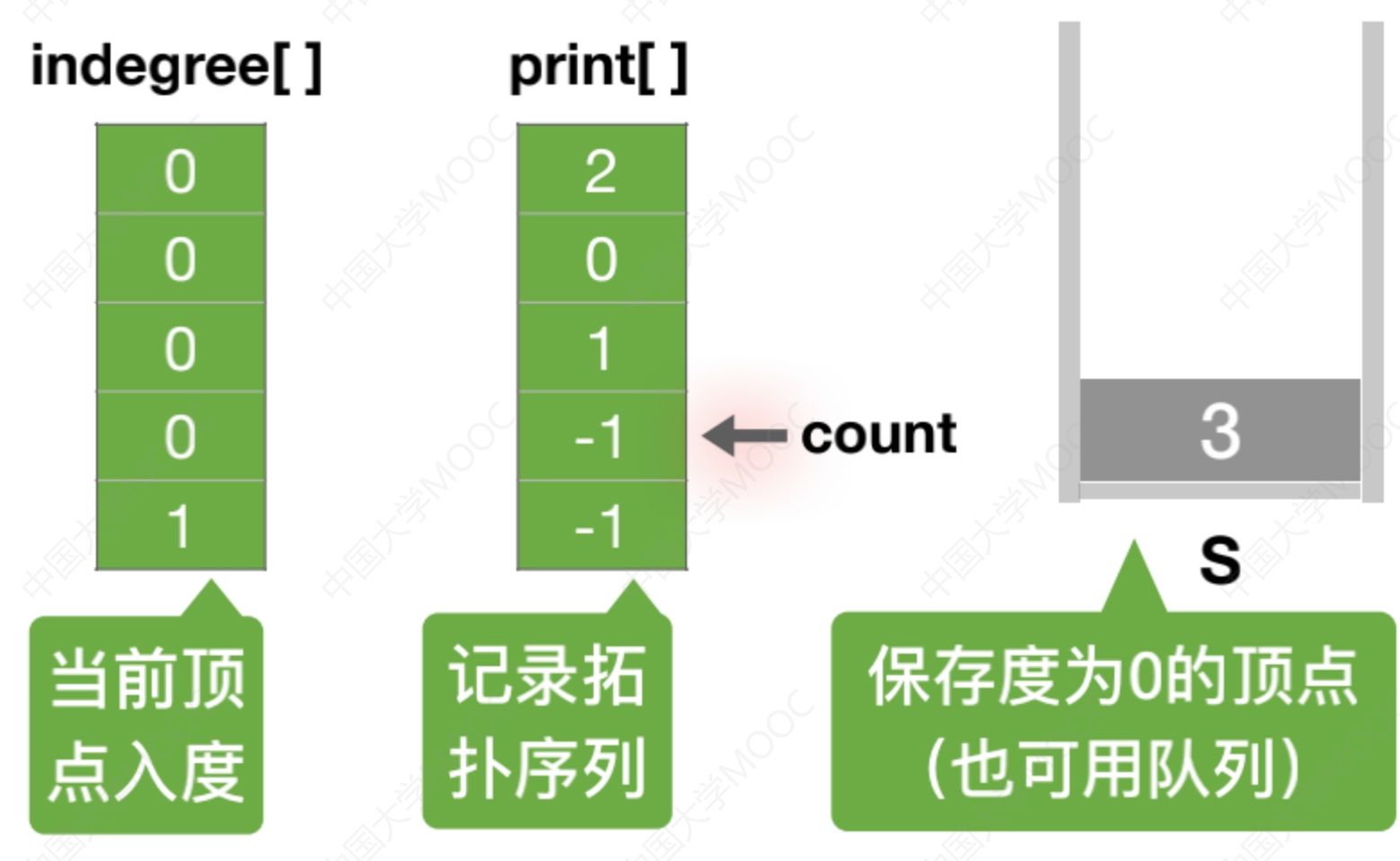
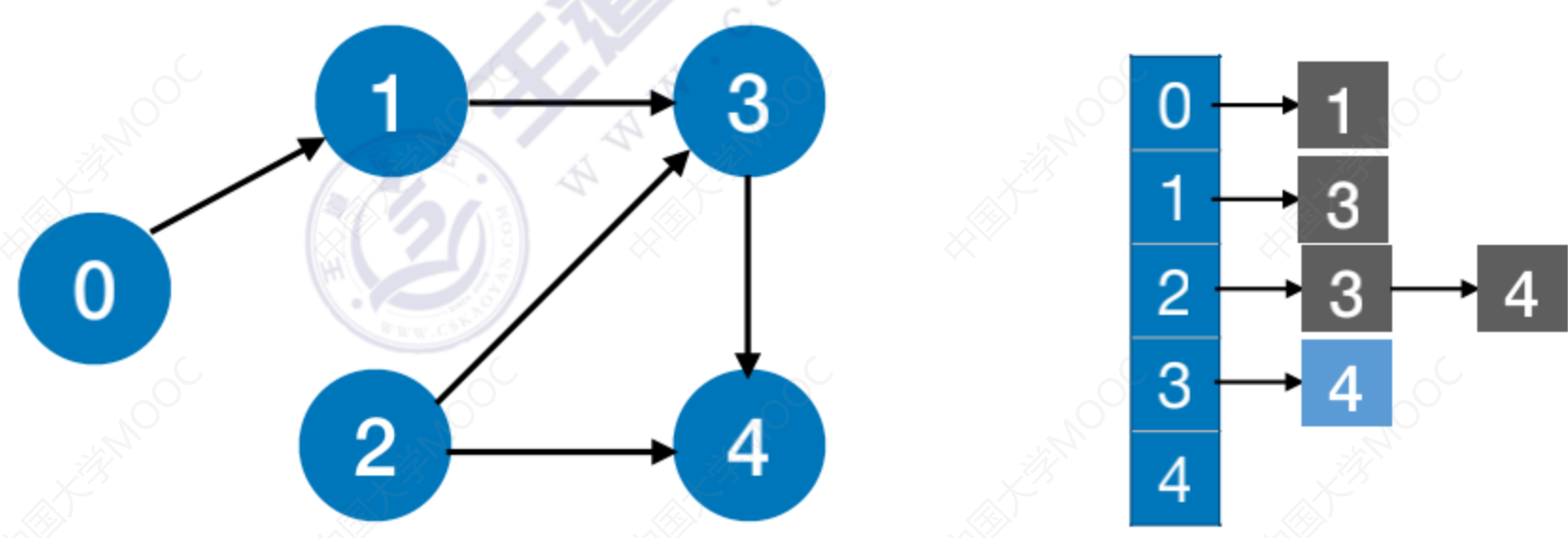
← count



保存度为0的顶点  
(也可用队列)

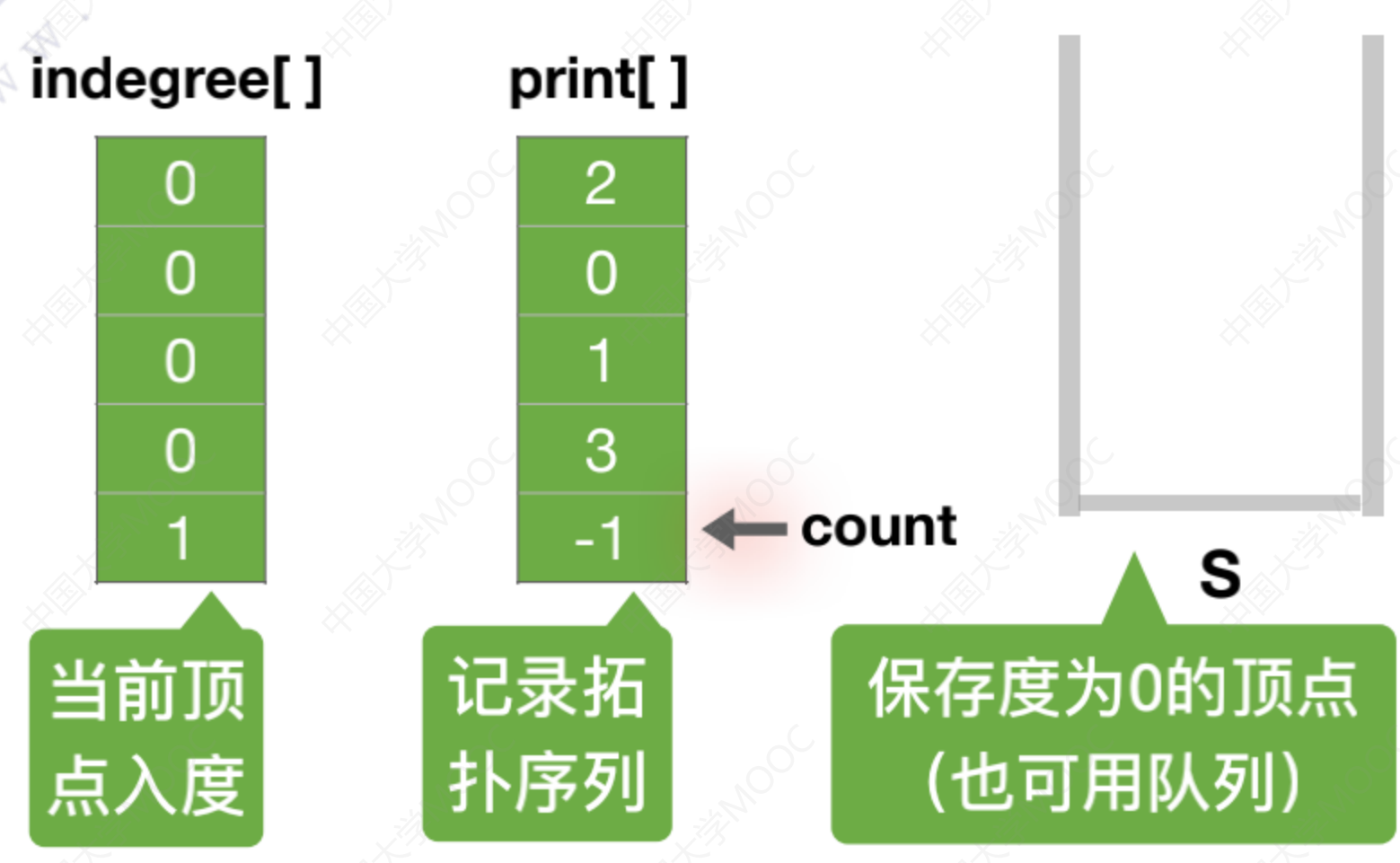
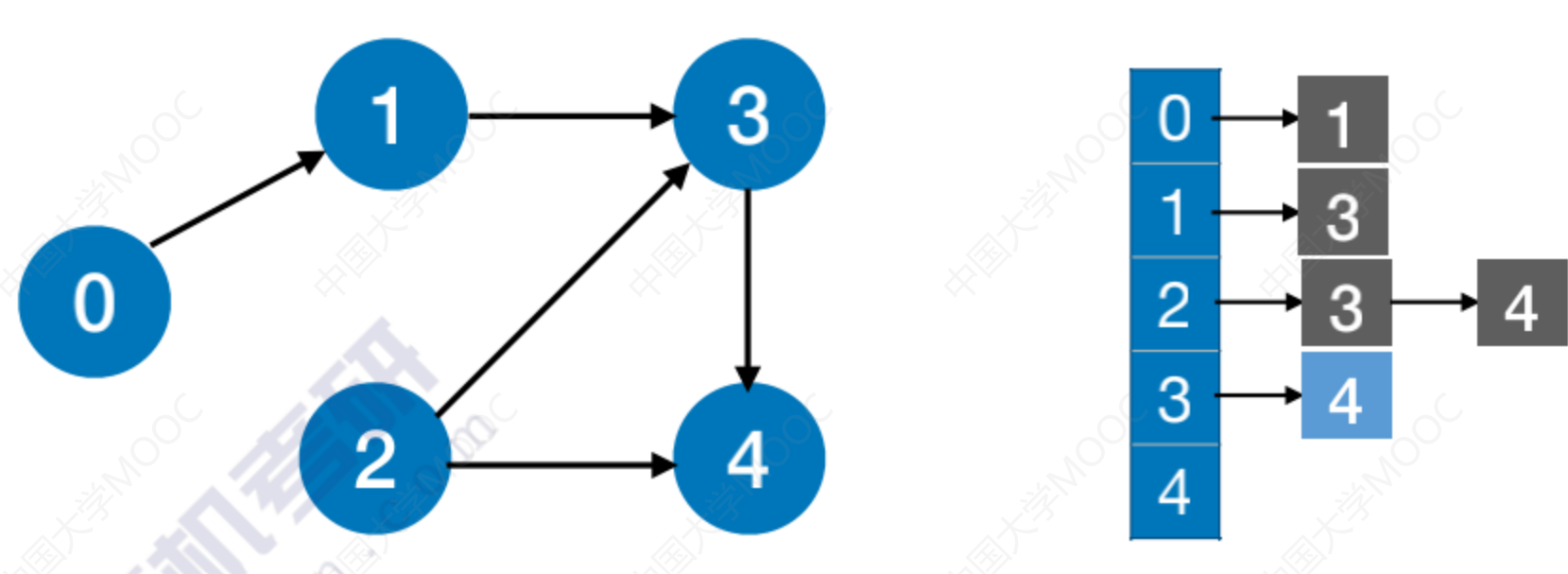
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



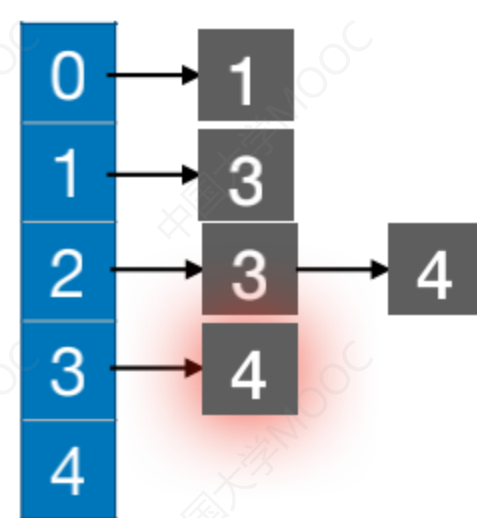
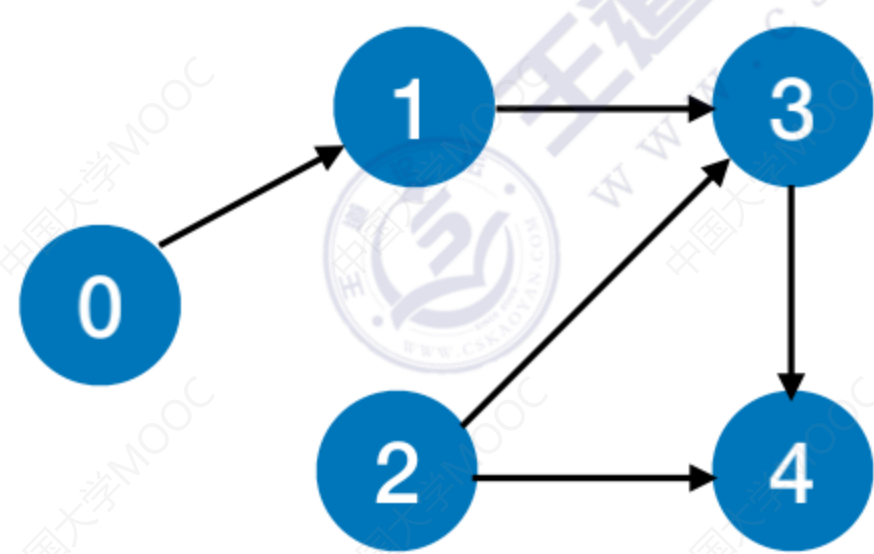
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

0  
0  
0  
0  
1

当前顶  
点入度

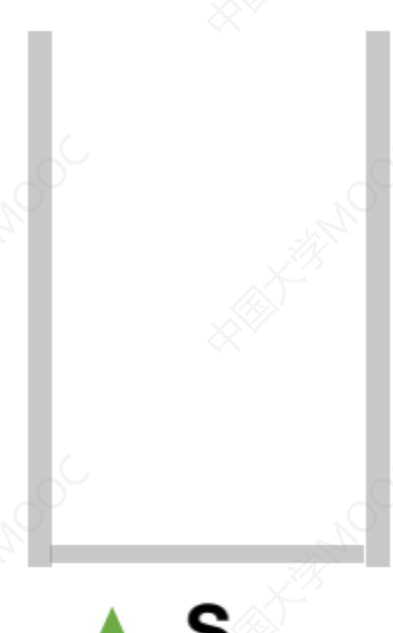
print[ ]

2  
0  
1  
3  
-1

记录拓  
扑序列

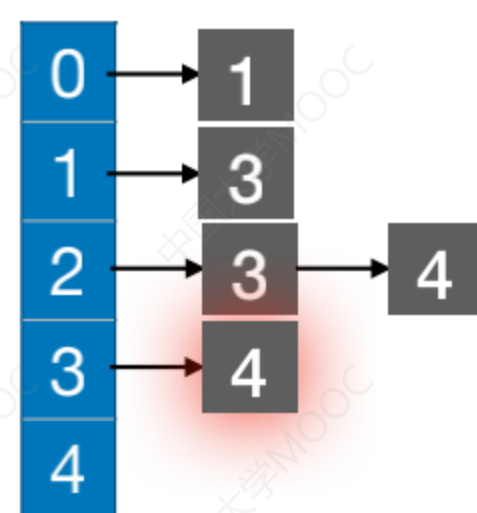
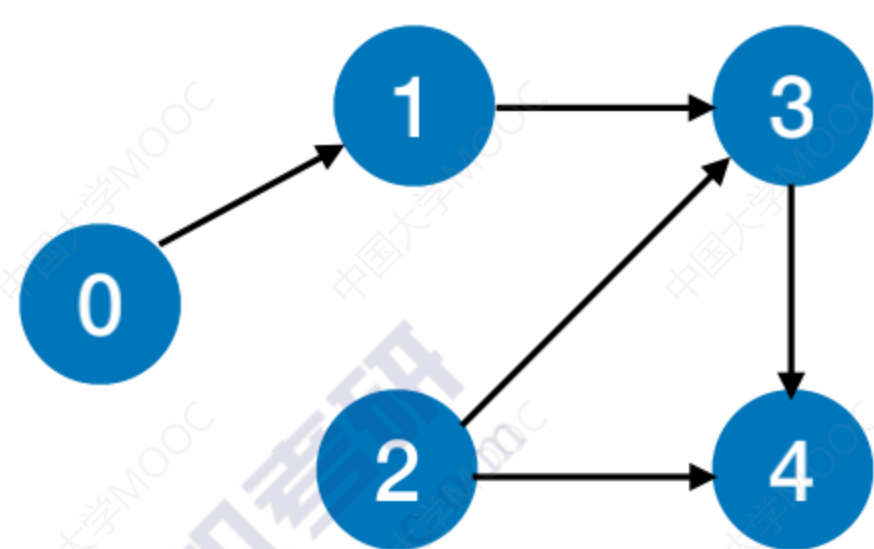
← count

保存度为0的顶点  
(也可用队列)



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



indegree[ ]

0  
0  
0  
0  
0

当前顶  
点入度

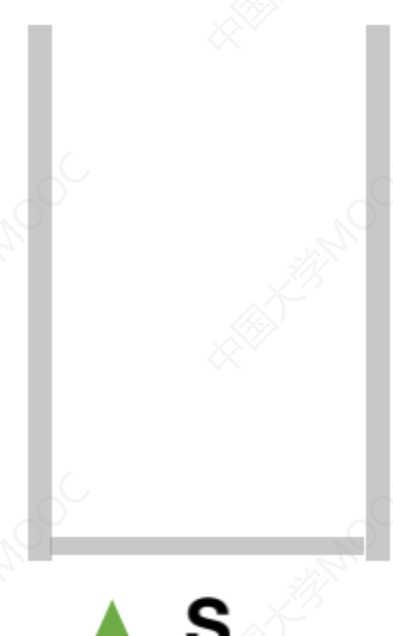
print[ ]

2  
0  
1  
3  
-1

记录拓  
扑序列

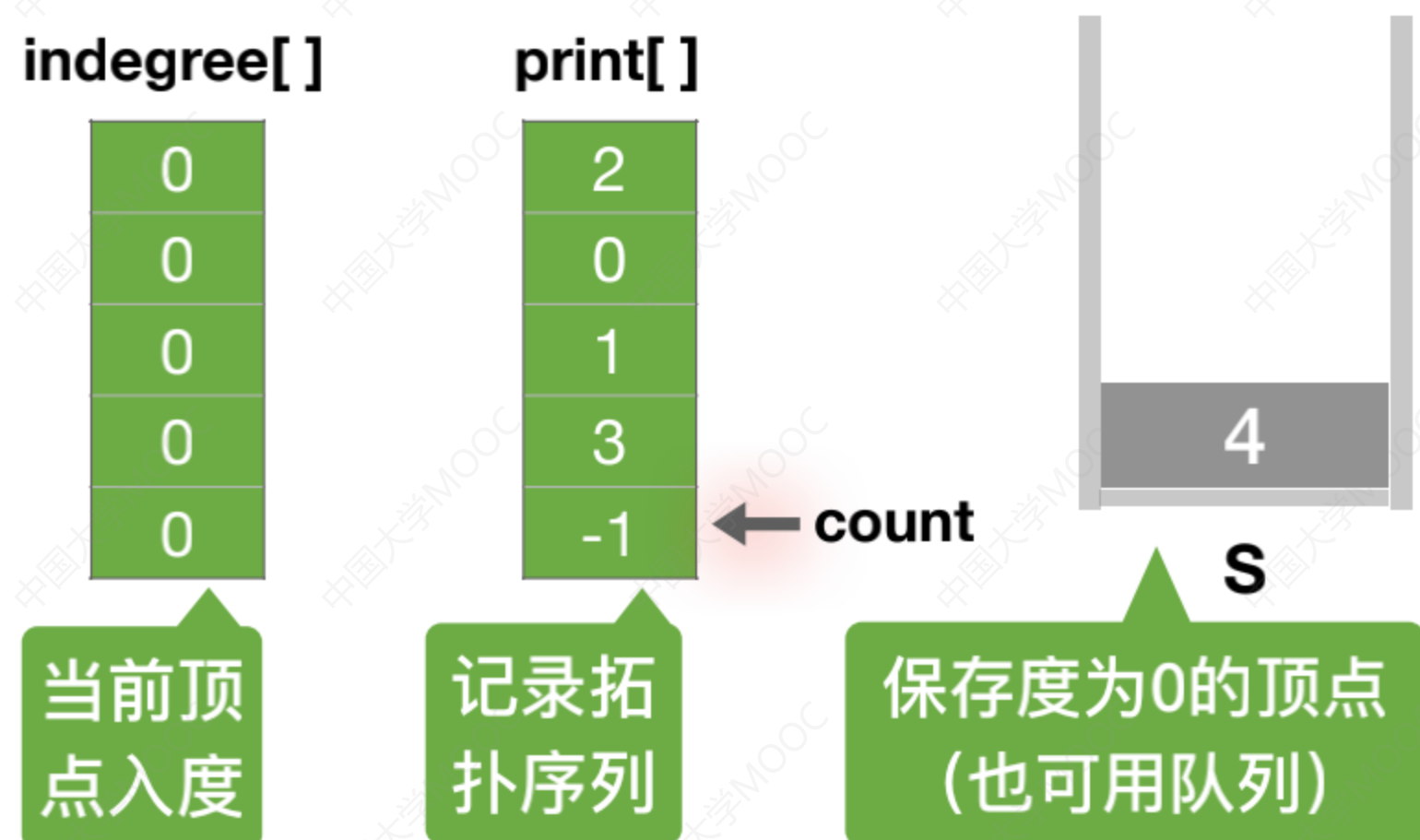
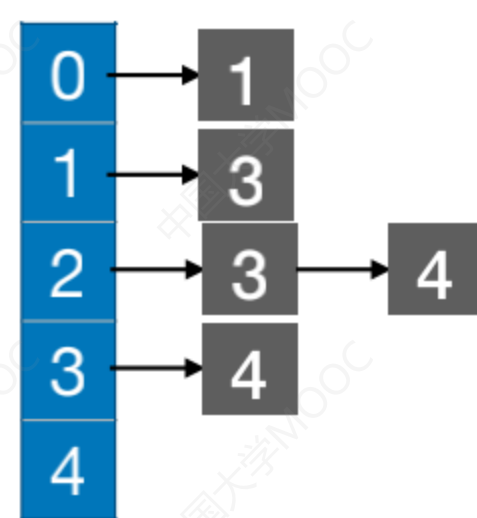
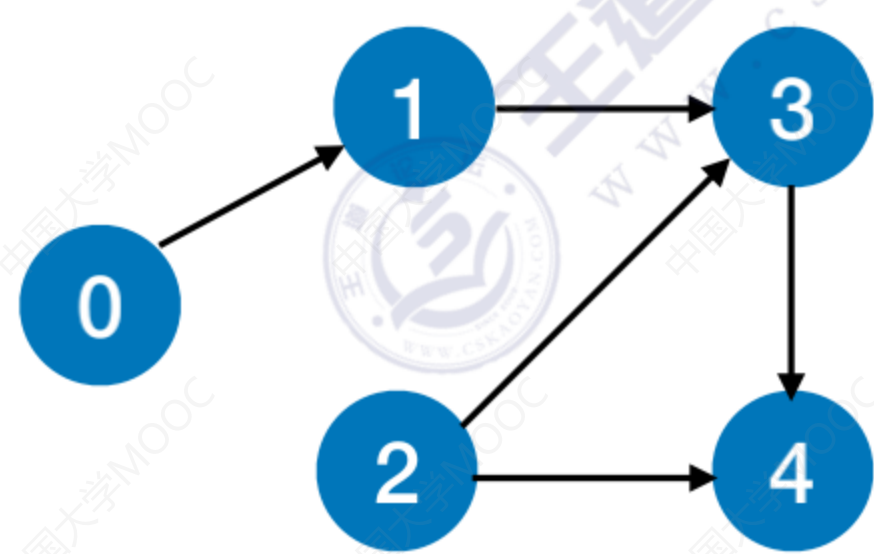
← count

保存度为0的顶点  
(也可用队列)



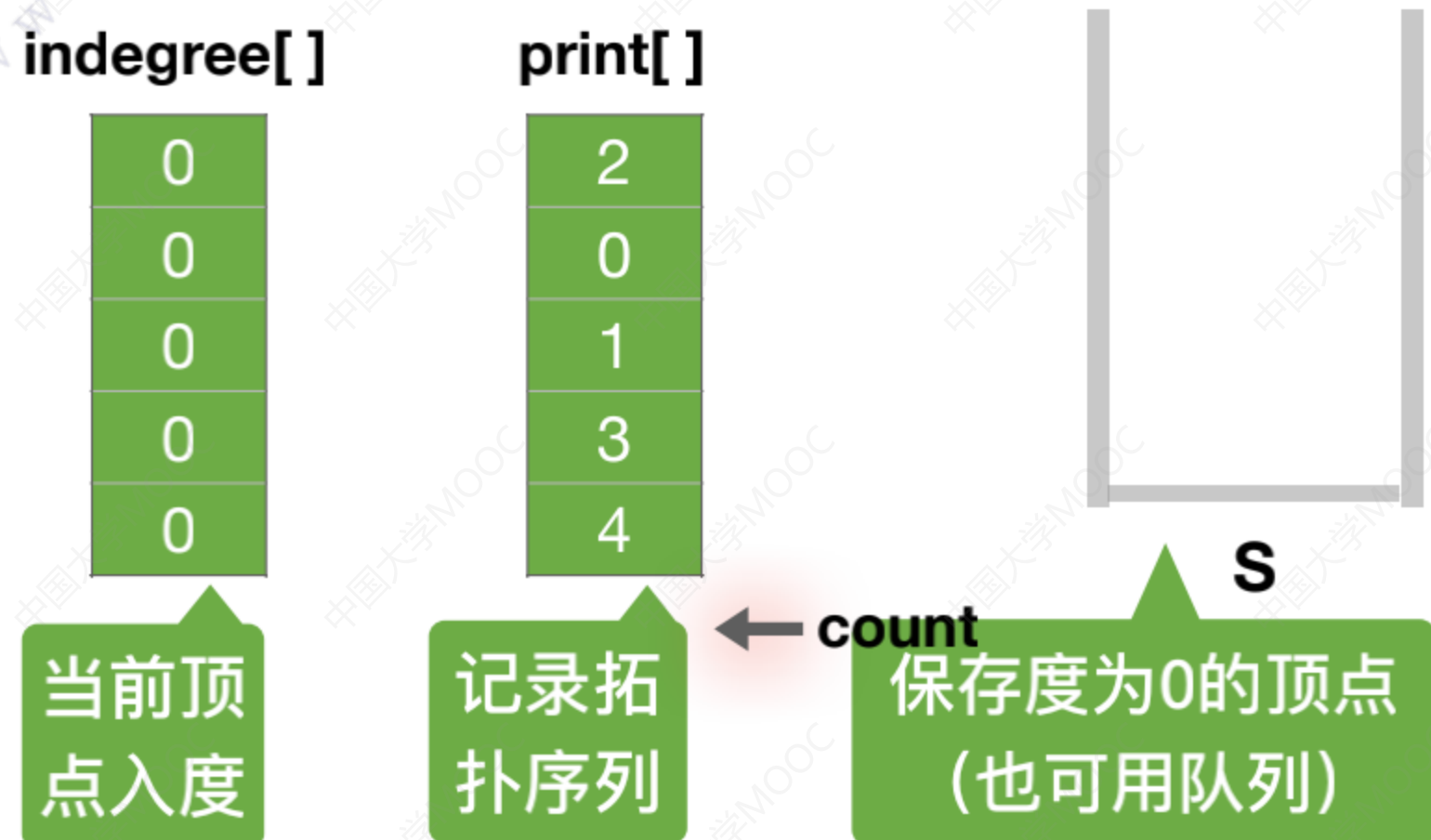
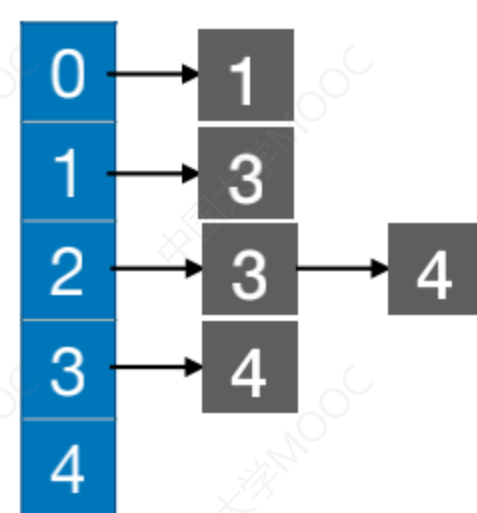
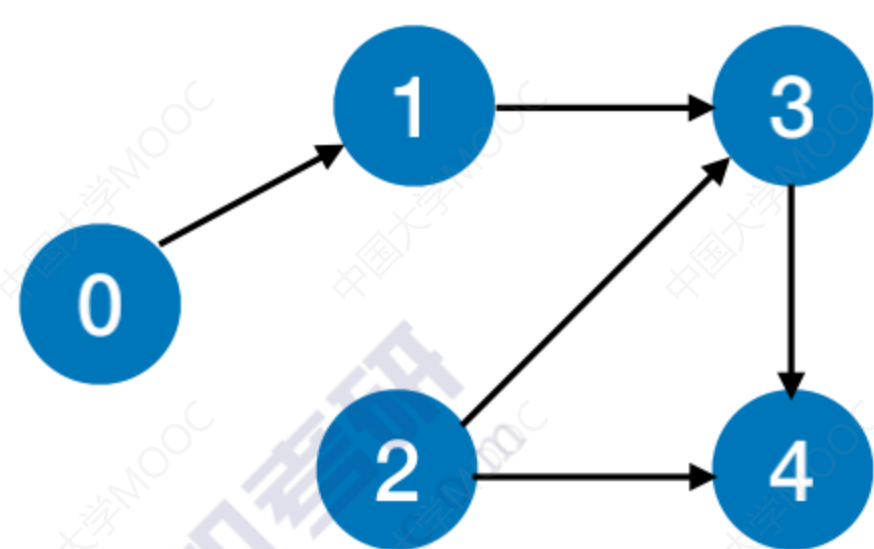
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



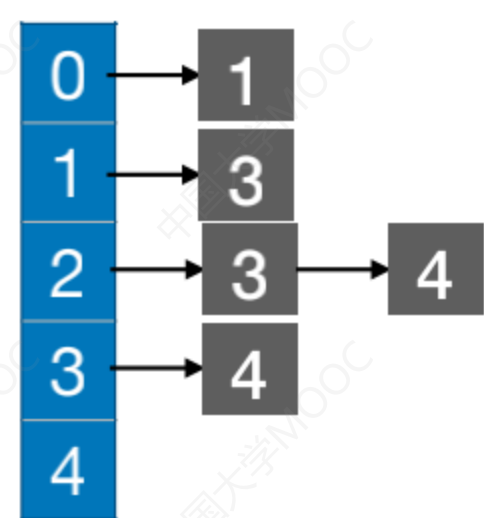
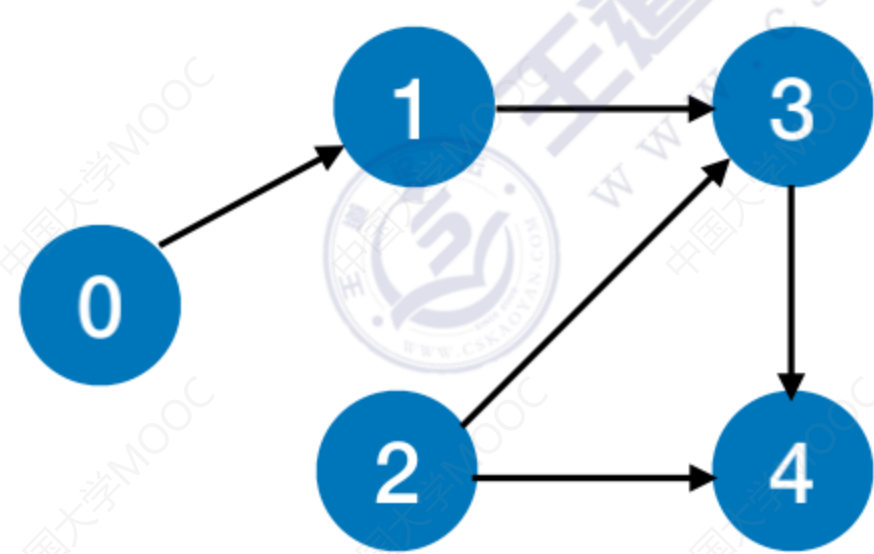
```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    }//while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM



每个顶点都需要处理一次

每条边都需要处理一次

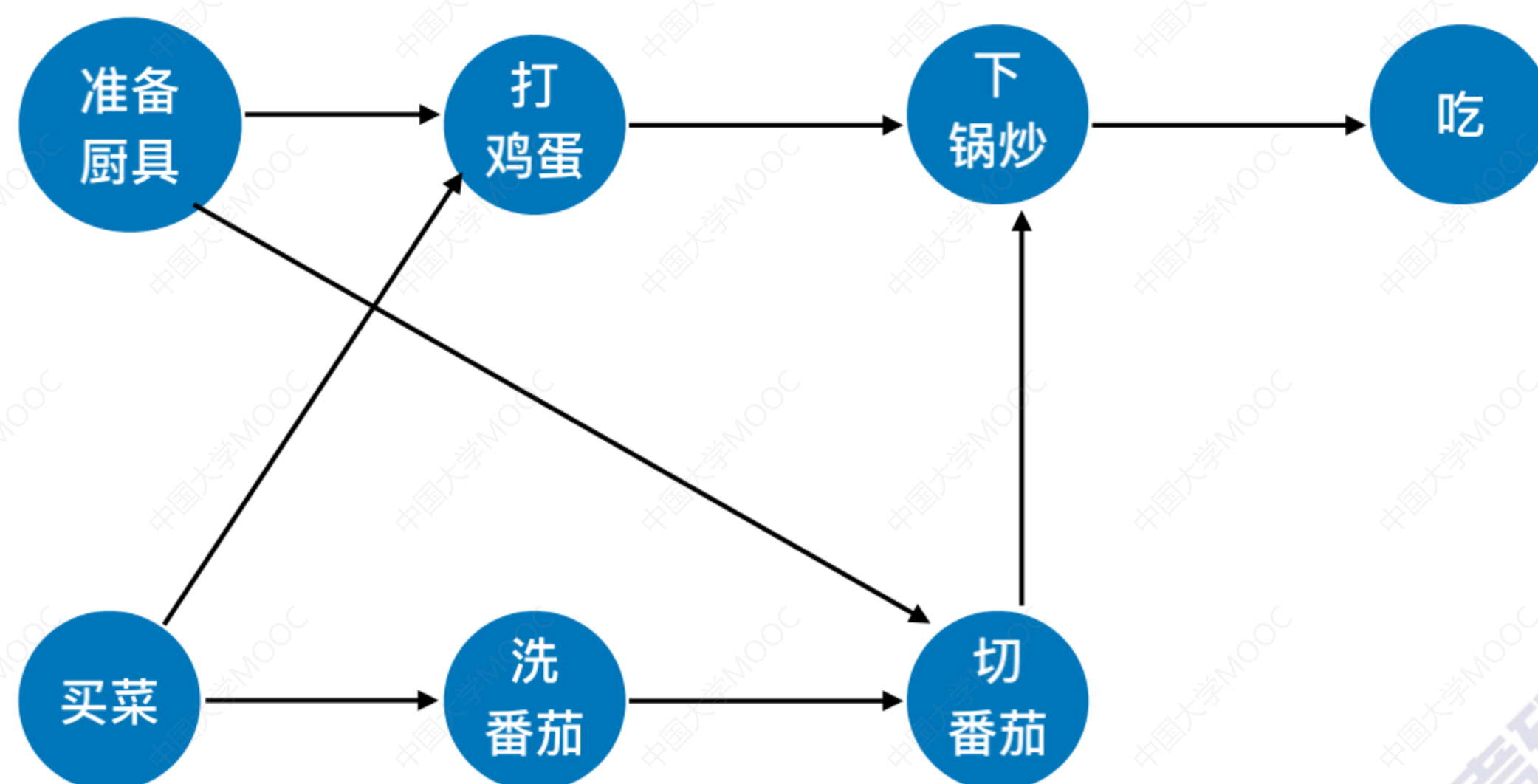
时间复杂度:  $O(|V|+|E|)$

若采用邻接矩阵, 则需 $O(|V|^2)$

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(!(--indegree[v]))
                Push(S,v); //入度为0, 则入栈
        }
    } //while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序

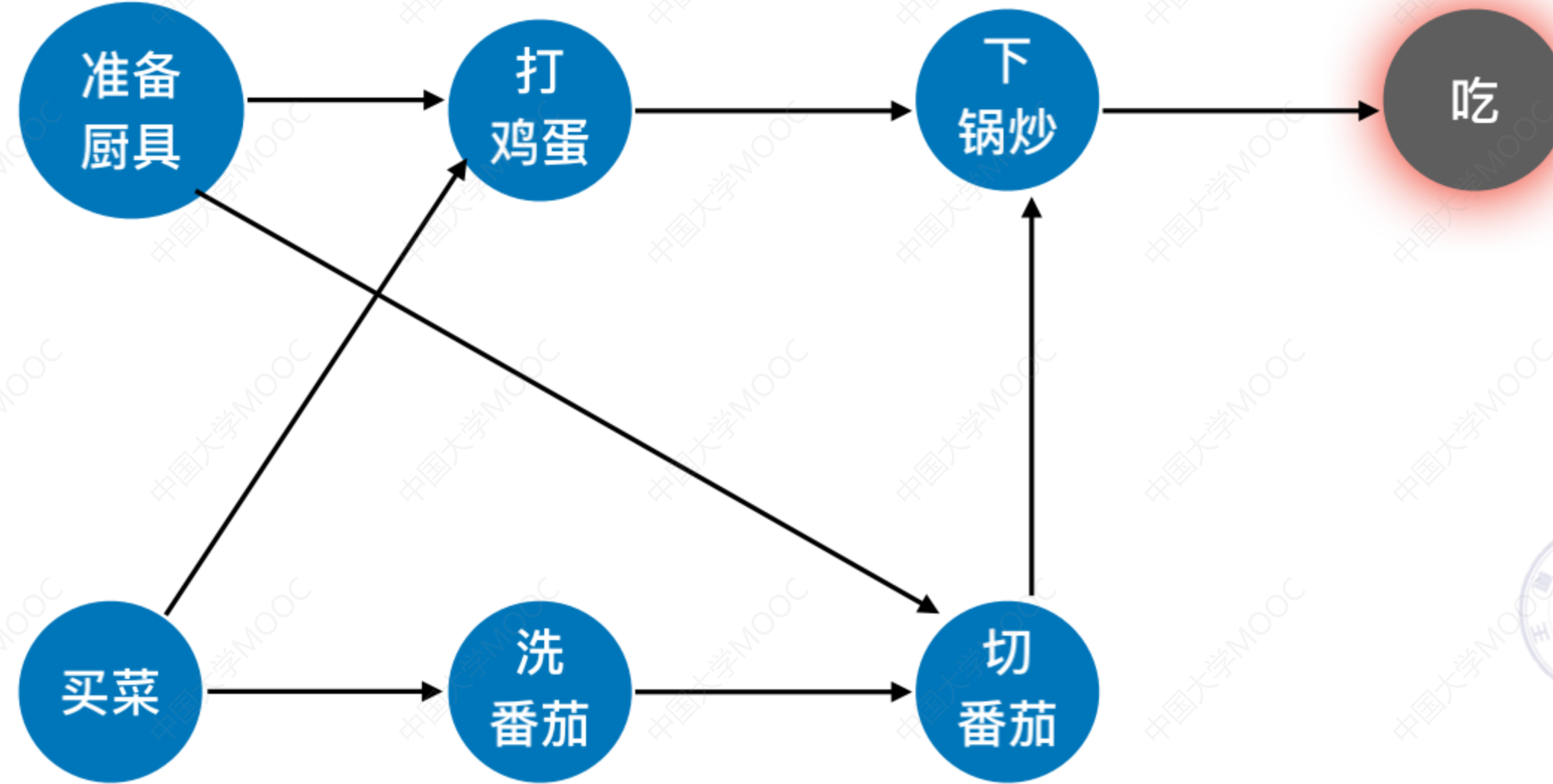


对一个AOV网, 如果采用下列步骤进行排序, 则称之为**逆拓扑排序**:

- ① 从AOV网中选择一个没有后继 (出度为0) 的顶点并输出。
- ② 从网中删除该顶点和所有以它为终点的有向边。
- ③ 重复①和②直到当前的AOV网为空。

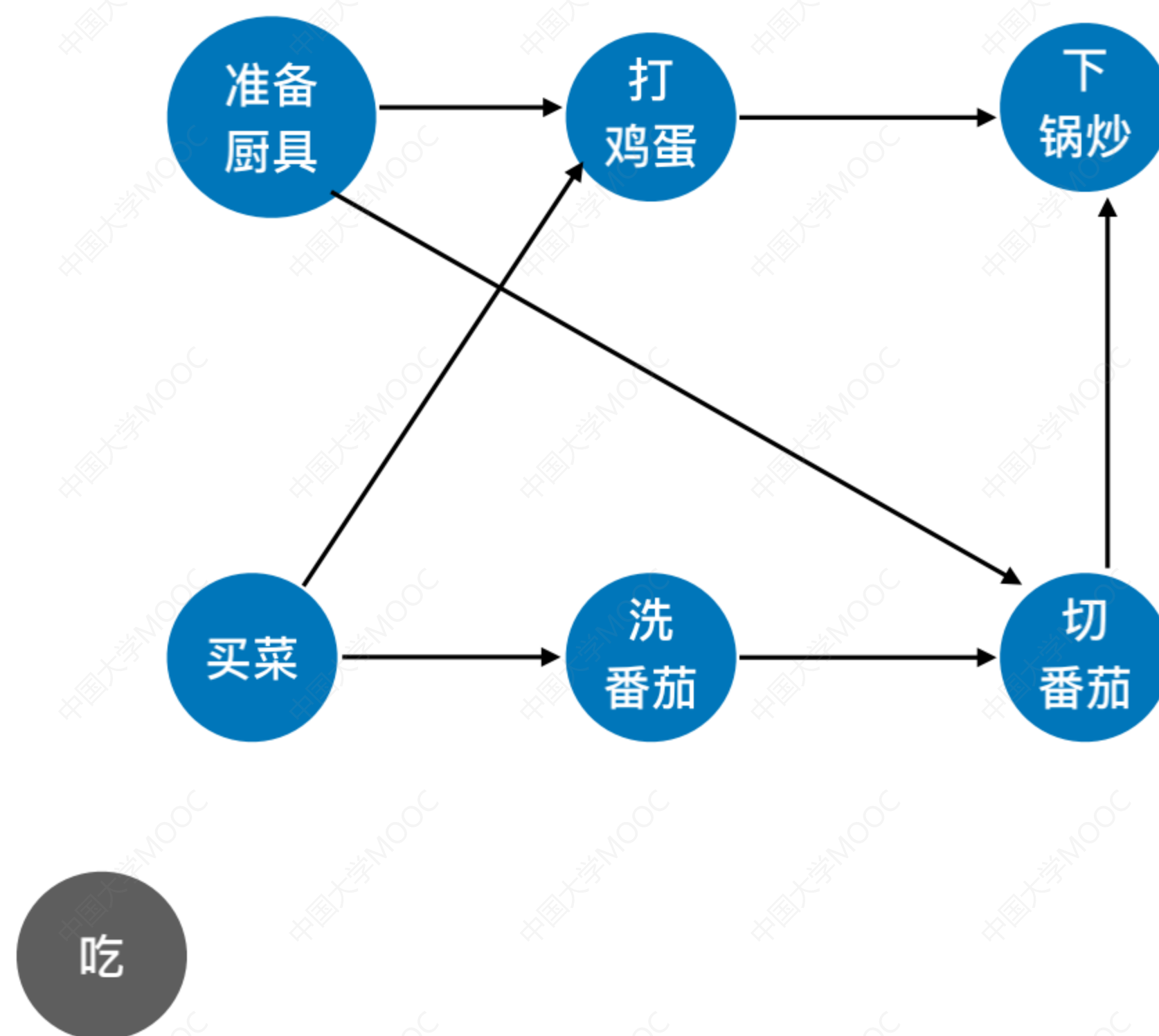
王道考研/CSKAOYAN.COM

### 逆拓扑排序



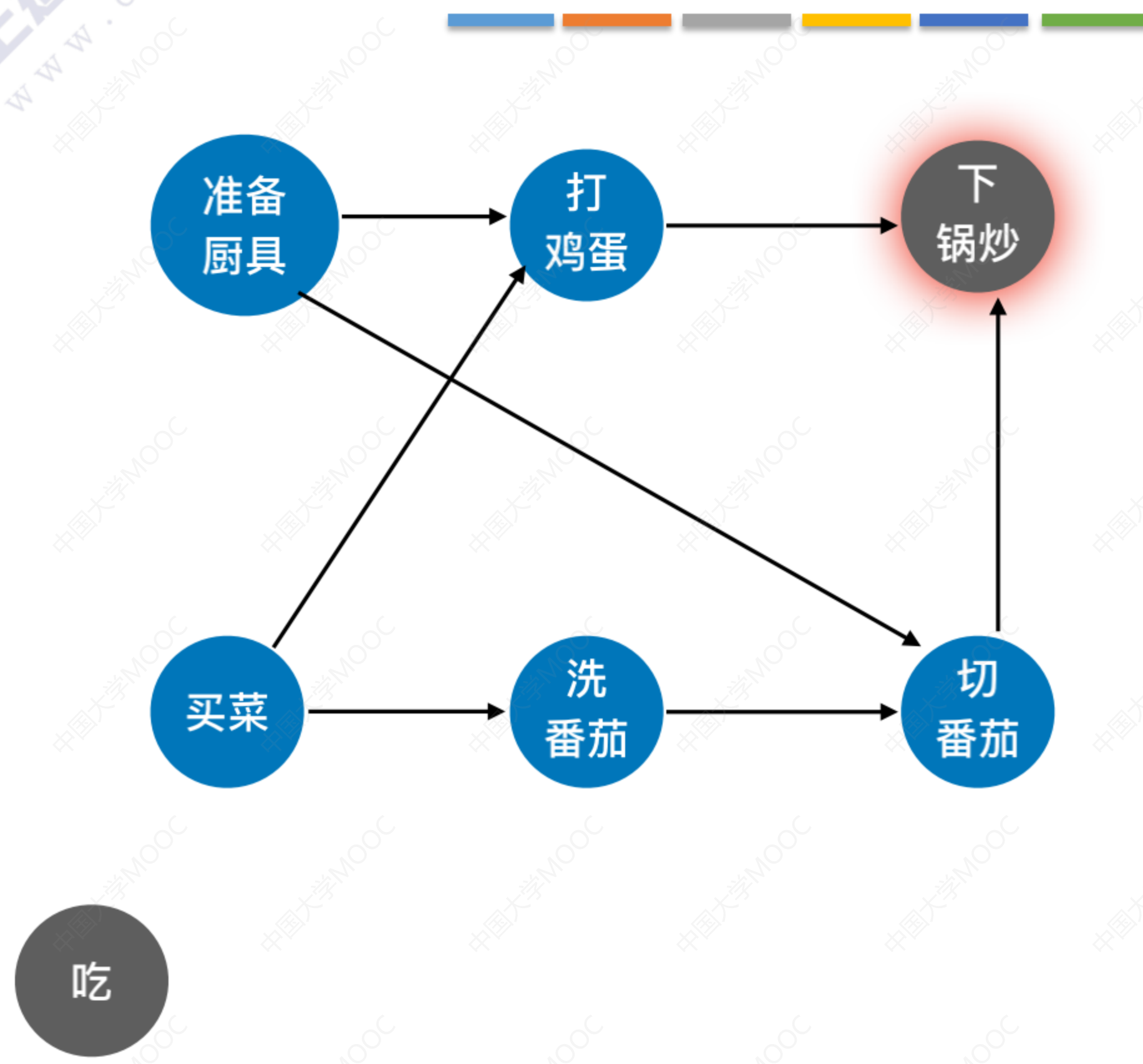
王道考研/CSKAOYAN.COM

### 逆拓扑排序



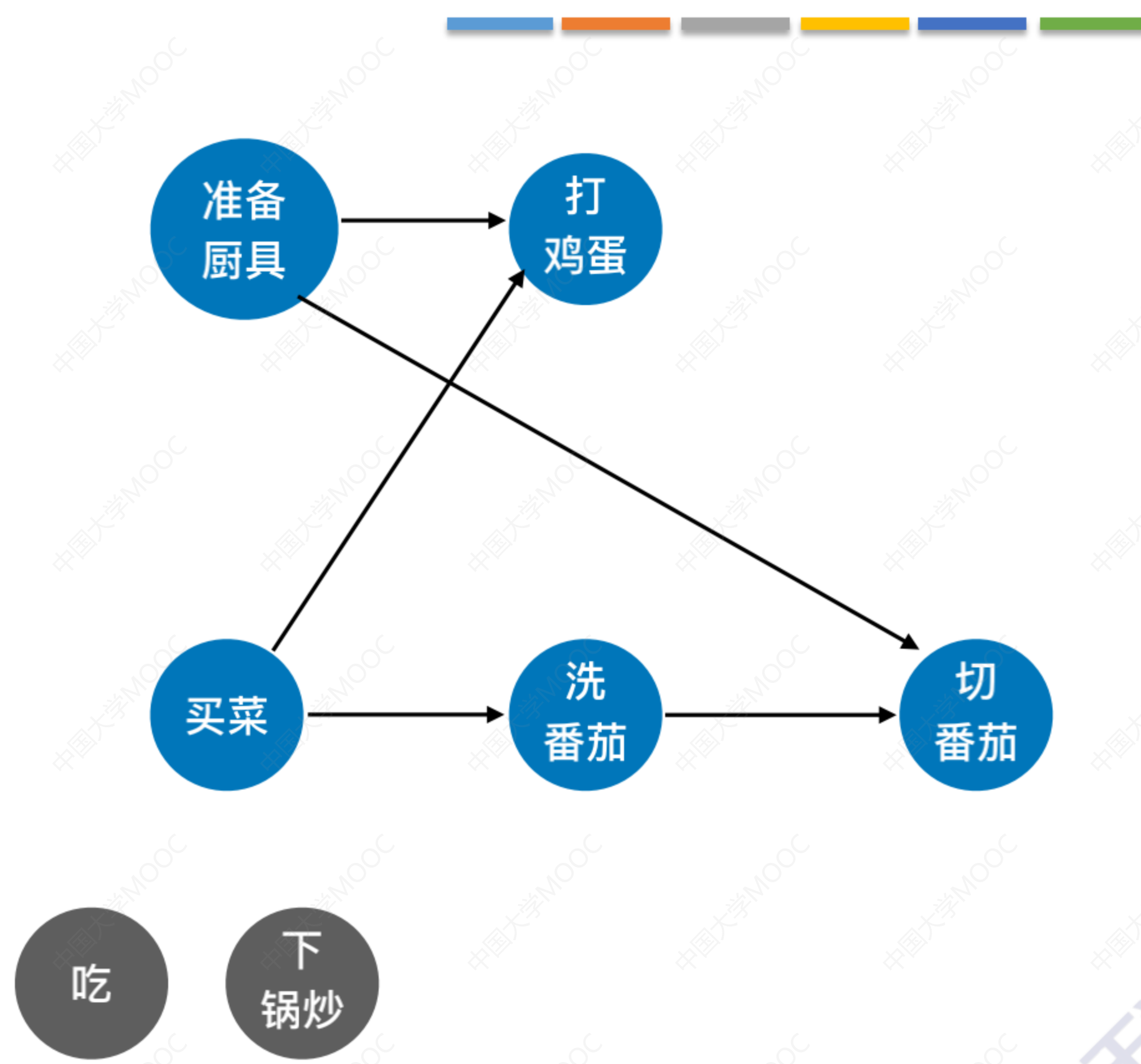
王道考研/CSKAOYAN.COM

## 逆拓扑排序



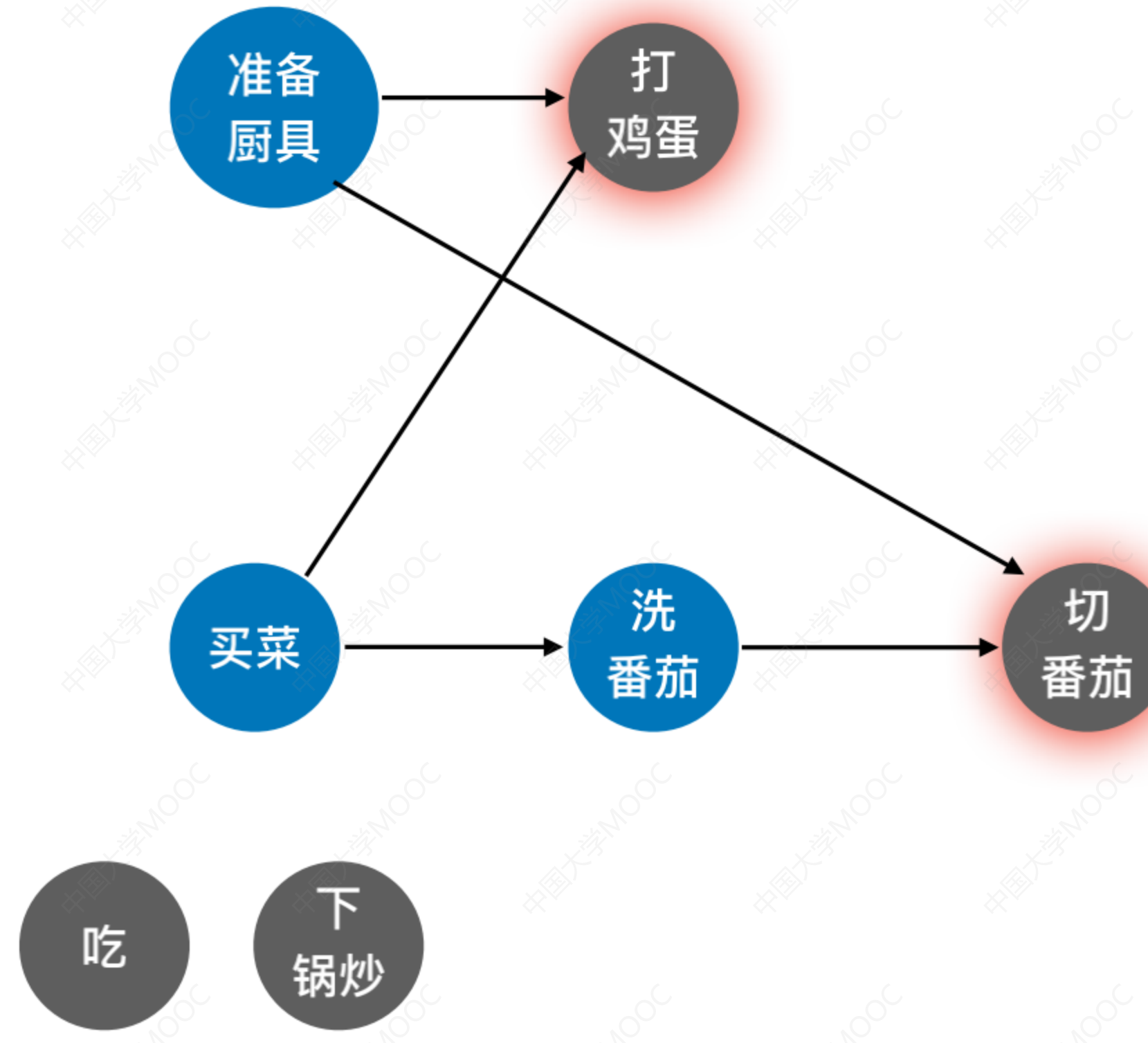
王道考研/CSKAOYAN.COM

## 逆拓扑排序



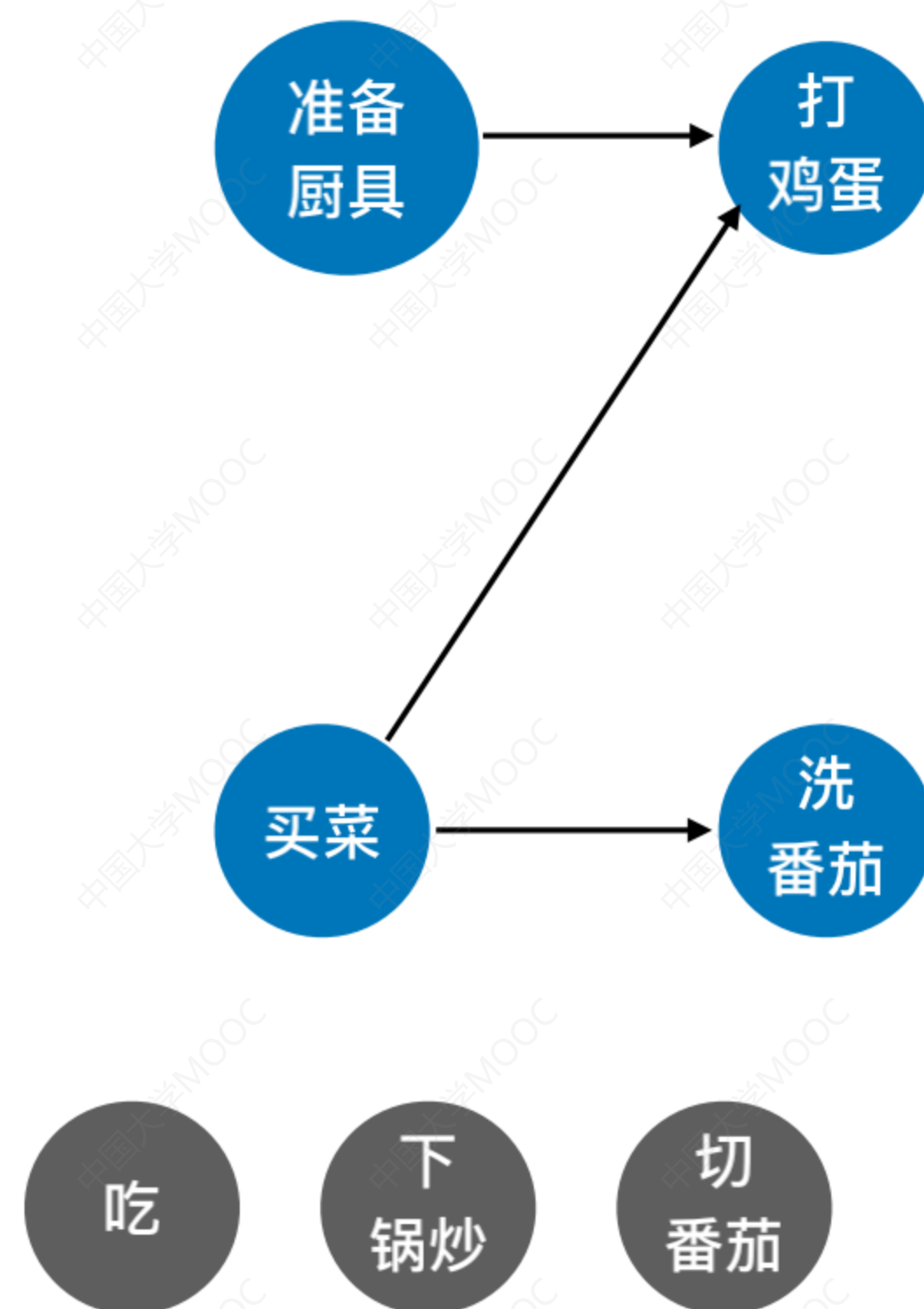
王道考研/CSKAOYAN.COM

## 逆拓扑排序



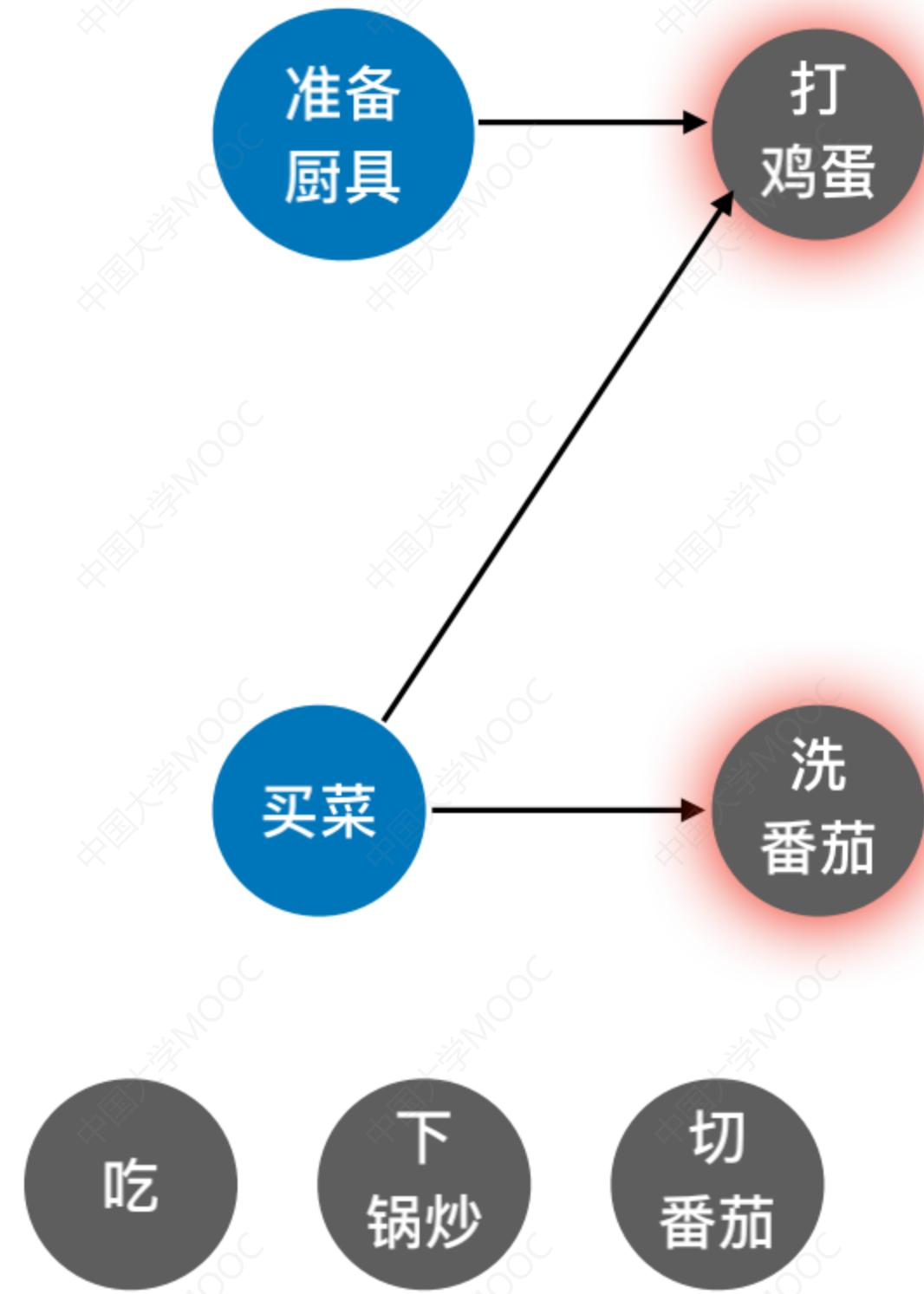
王道考研/CSKAOYAN.COM

## 逆拓扑排序



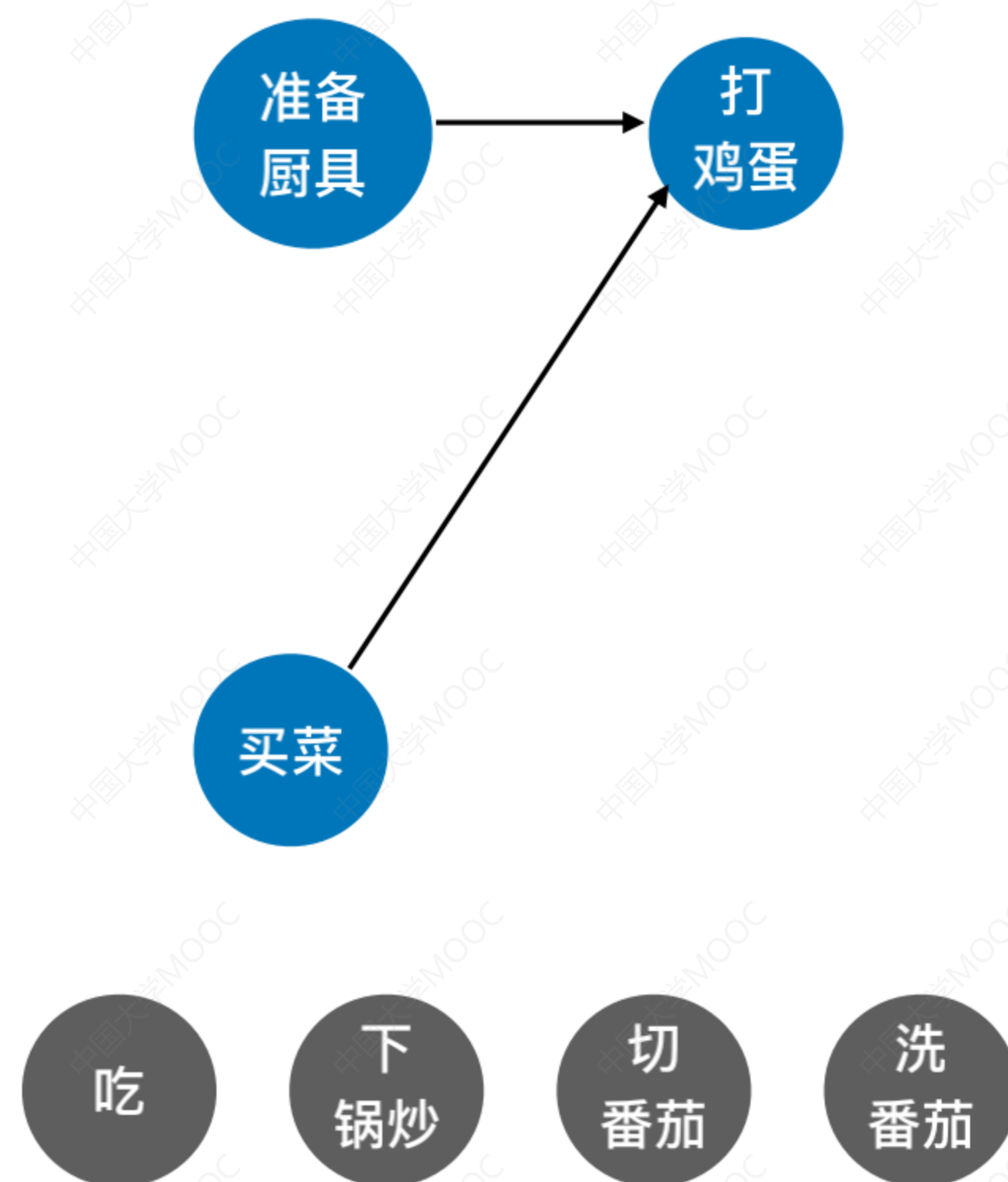
王道考研/CSKAOYAN.COM

## 逆拓扑排序



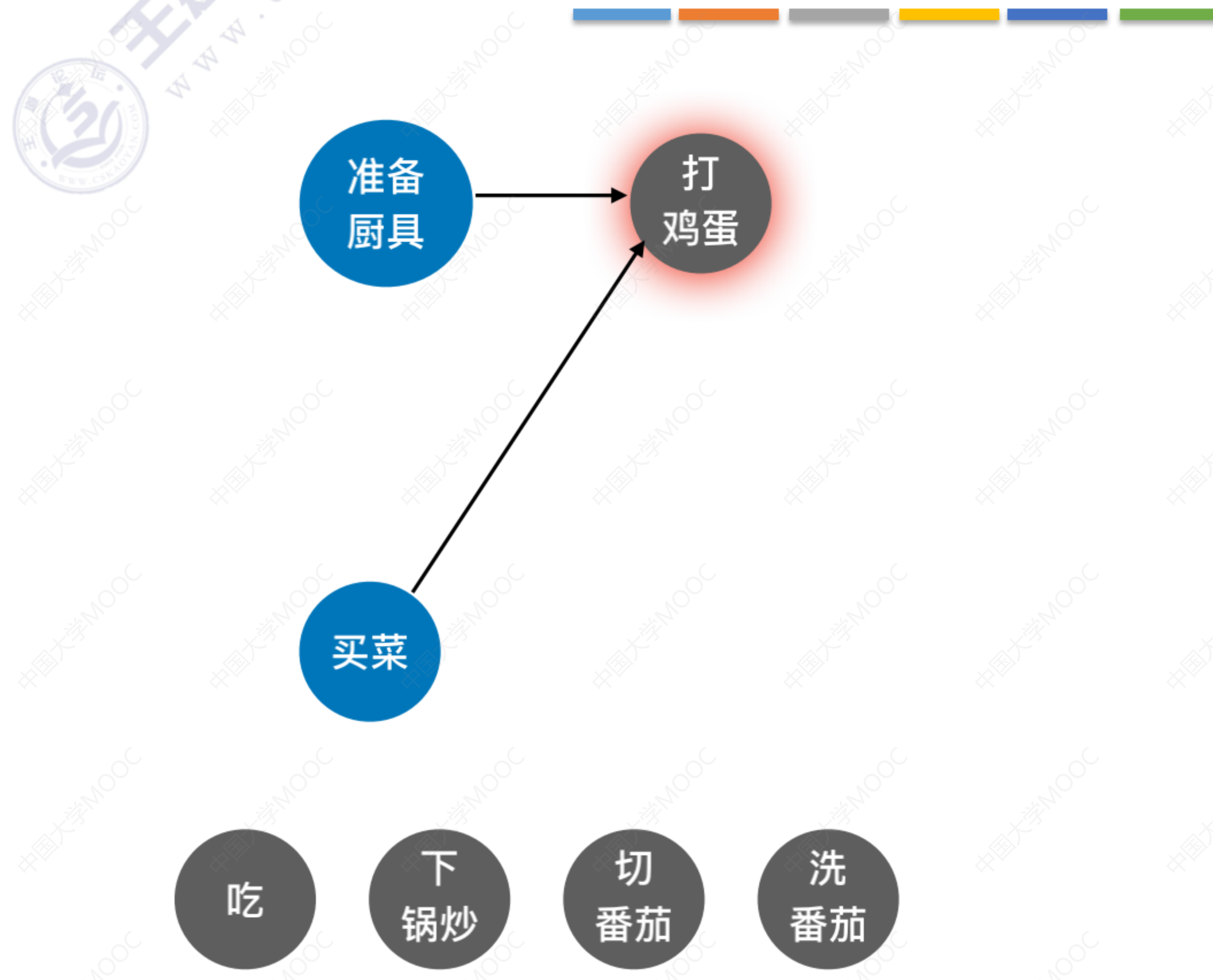
王道考研/CSKAOYAN.COM

## 逆拓扑排序



王道考研/CSKAOYAN.COM

## 逆拓扑排序



王道考研/CSKAOYAN.COM

## 逆拓扑排序



王道考研/CSKAOYAN.COM

## 逆拓扑排序

准备  
厨具

买菜

吃

下  
锅炒

切  
番茄

洗  
番茄

打  
鸡蛋

王道考研/CSKAOYAN.COM

## 逆拓扑排序

对一个AOV网逆拓扑排序：

- ① 从AOV网中选择一个没有后继（出度为0）的顶点并输出。
- ② 从网中删除该顶点和所有以它为终点的有向边。
- ③ 重复①和②直到当前的AOV网为空。

吃

下  
锅炒

切  
番茄

洗  
番茄

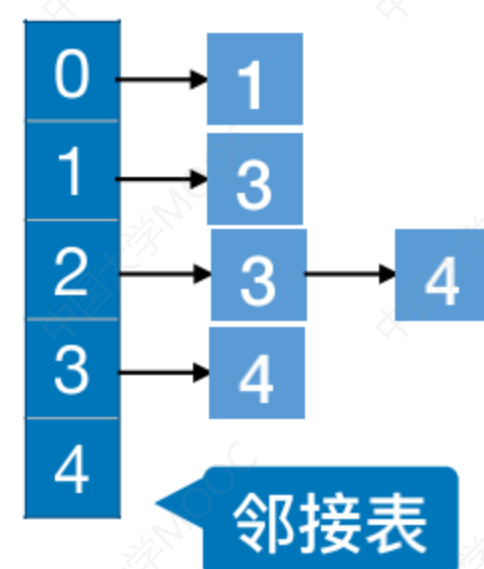
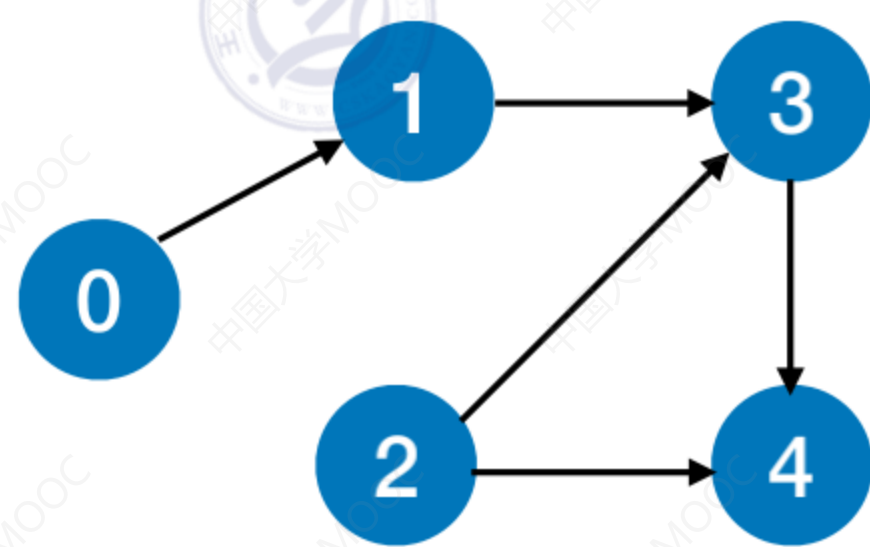
打  
鸡蛋

准备  
厨具

买菜

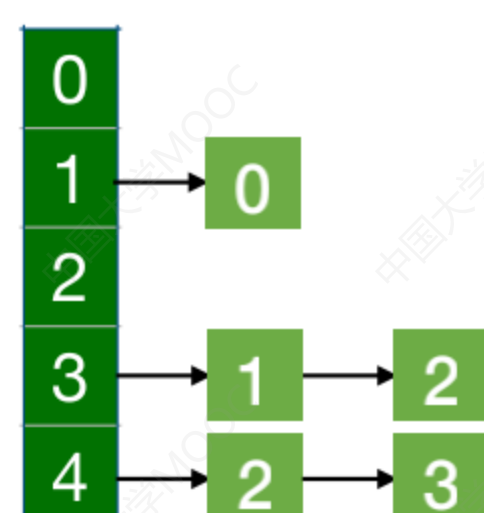
王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现



练习：模仿拓扑排序的思想实现**逆拓扑排序**  
 思考：使用不同的存储结构来对时间复杂度的影响

	0	1	2	3	4
0	0	1	0	0	0
1	0	0	0	1	0
2	0	0	0	1	1
3	0	0	0	0	1
4	0	0	0	0	0

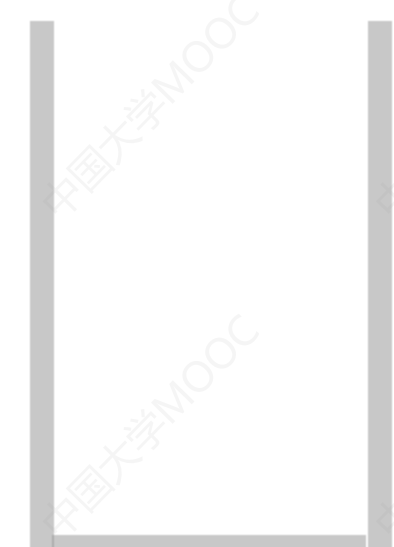
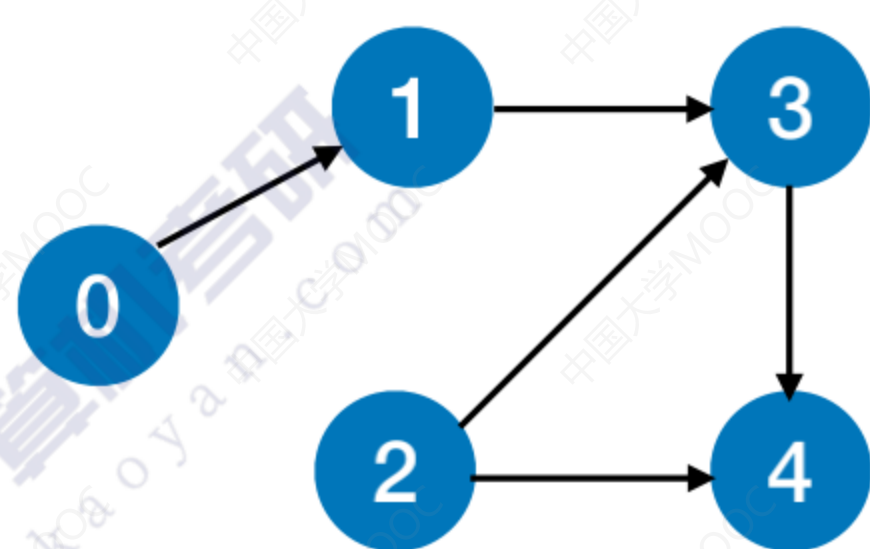


### 拓扑排序的实现

```
bool TopologicalSort(Graph G){
    InitStack(S); //初始化栈, 存储入度为0的顶点
    for(int i=0;i<G.vexnum;i++)
        if(indegree[i]==0)
            Push(S,i); //将所有入度为0的顶点进栈
    int count=0; //计数, 记录当前已经输出的顶点数
    while(!IsEmpty(S)){ //栈不为空, 则存在入度为0的顶点
        Pop(S,i); //栈顶元素出栈
        print[count++]=i; //输出顶点i
        for(p=G.vertices[i].firstarc;p;p=p->nextarc){
            //将所有i指向的顶点的入度减1, 并且将入度减为0的顶点压入栈s
            v=p->adjvex;
            if(--indegree[v]==0)
                Push(S,v); //入度为0, 则入栈
        }
    } //while
    if(count<G.vexnum)
        return false; //排序失败, 有向图中有回路
    else
        return true; //拓扑排序成功
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



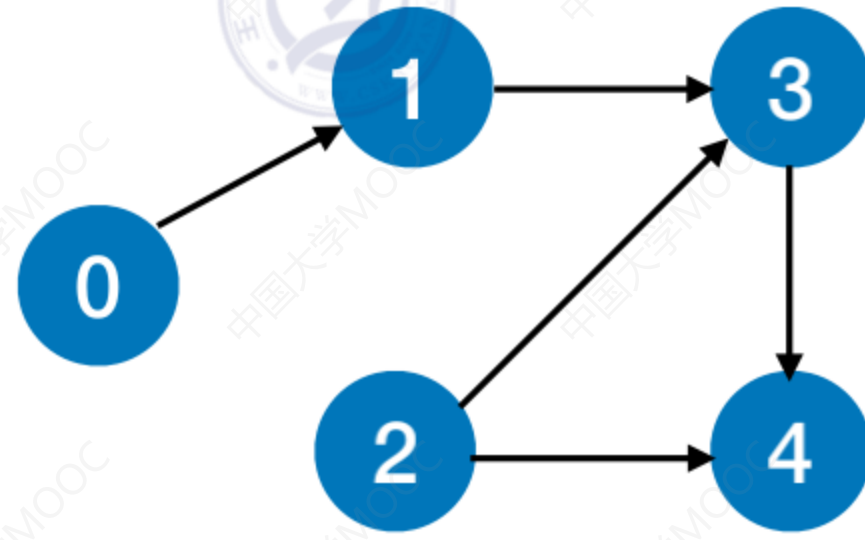
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v)
        if(!visited[v])
            DFS(G,v); //本代码中是从v=0开始遍历
}

void DFS(Graph G,int v){ //从顶点v出发, 深度优先遍历图G
    visit(v); //访问顶点v
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

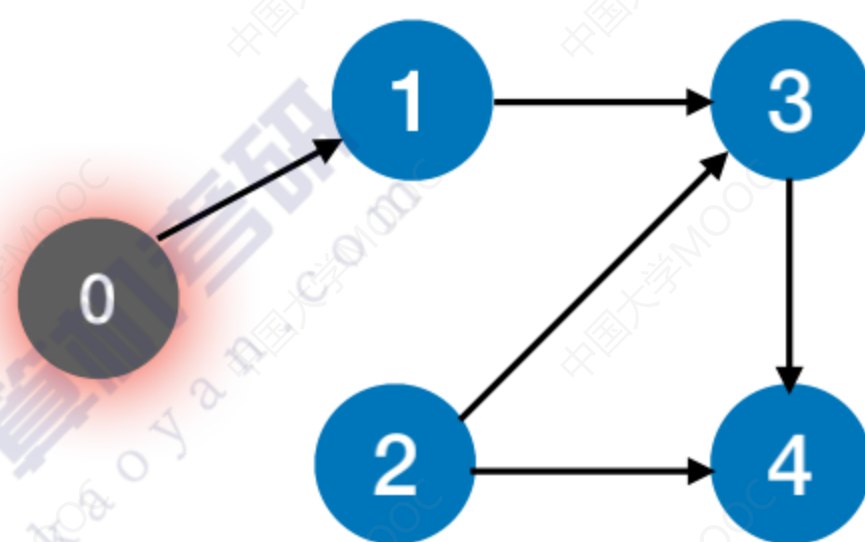


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

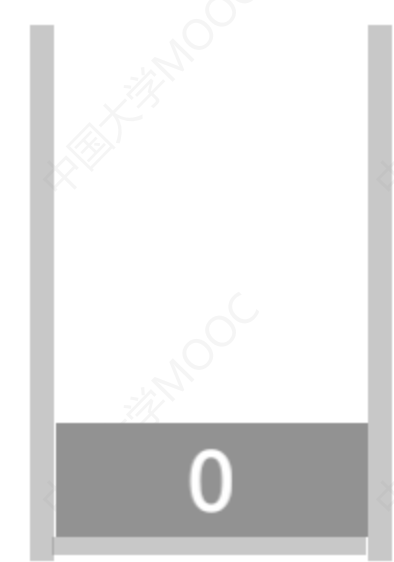
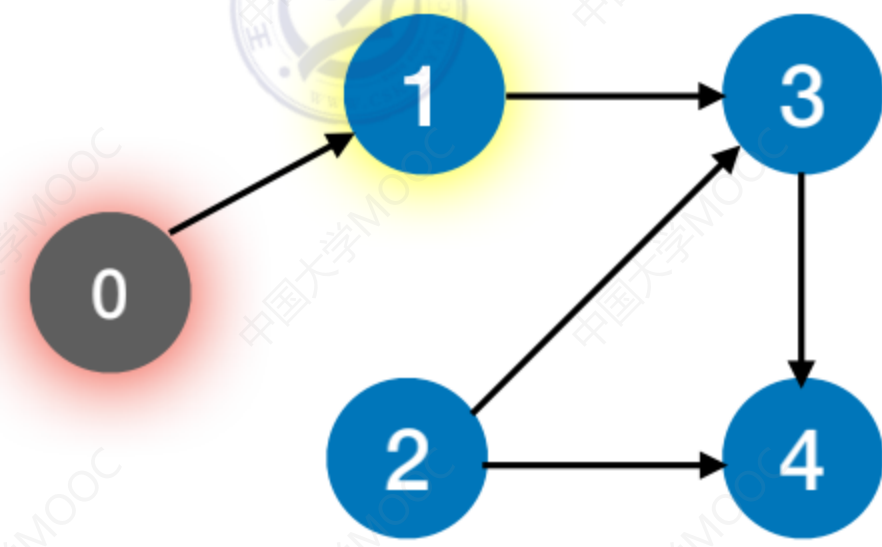


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

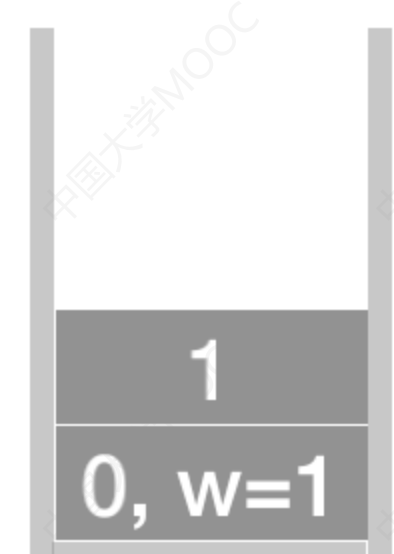
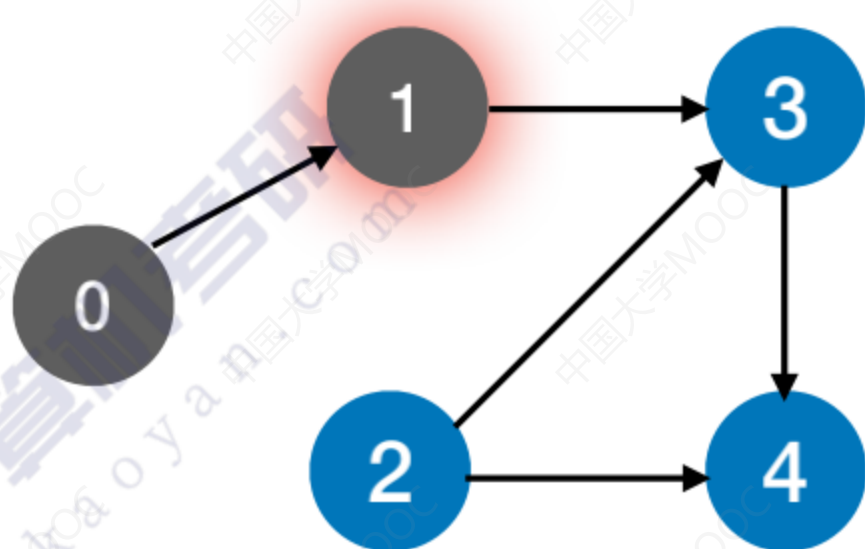


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

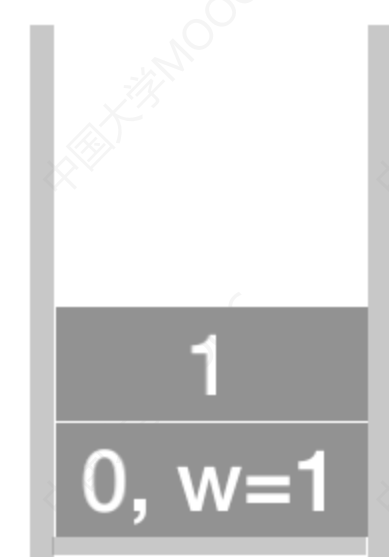
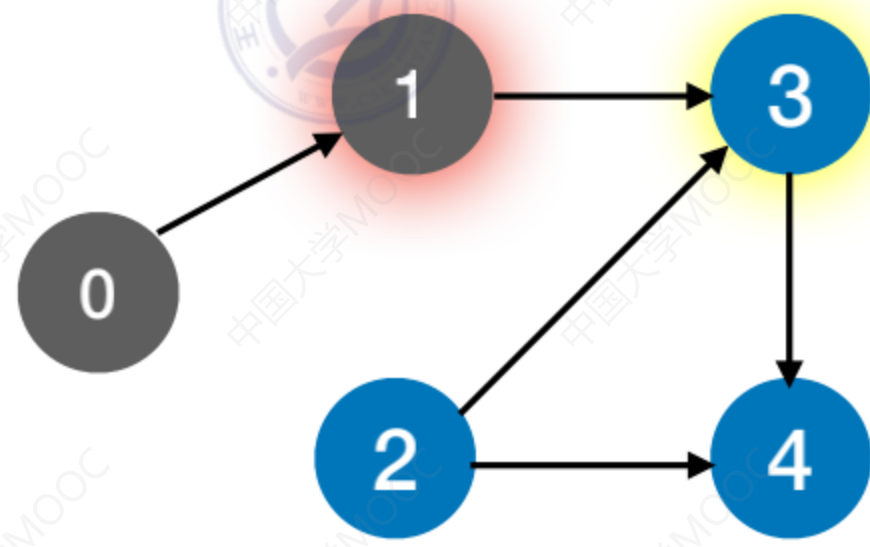


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

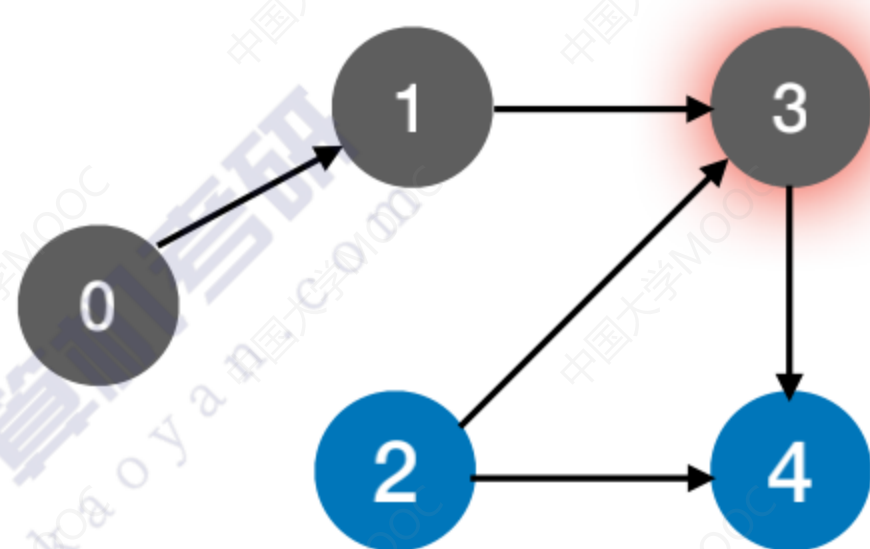


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)

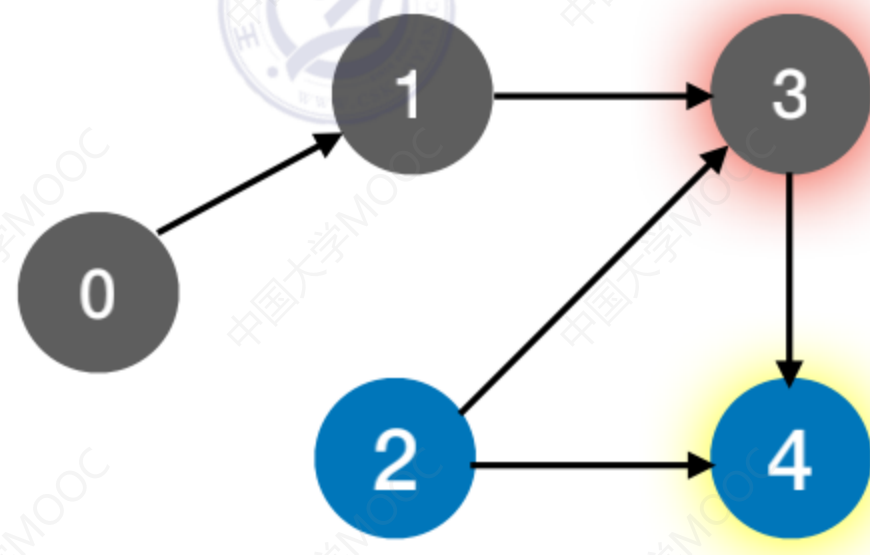


递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



递归栈

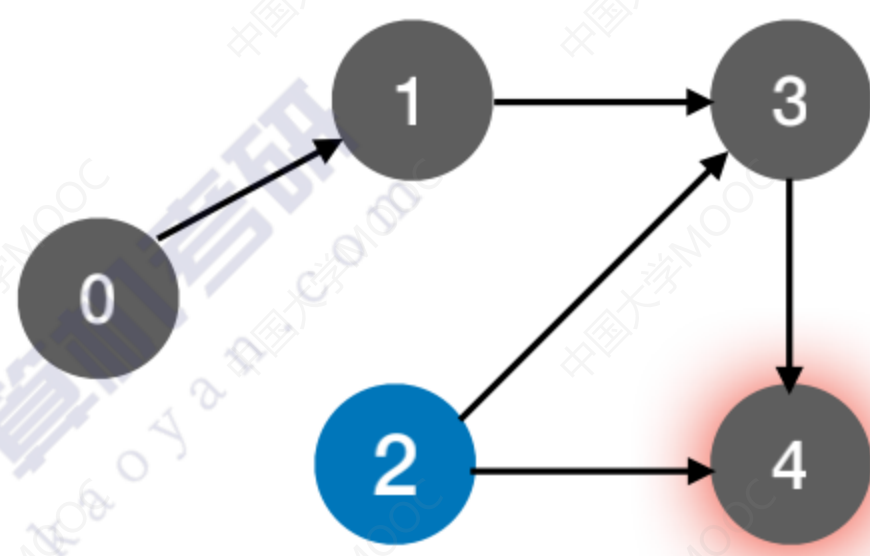
```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if

    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



递归栈

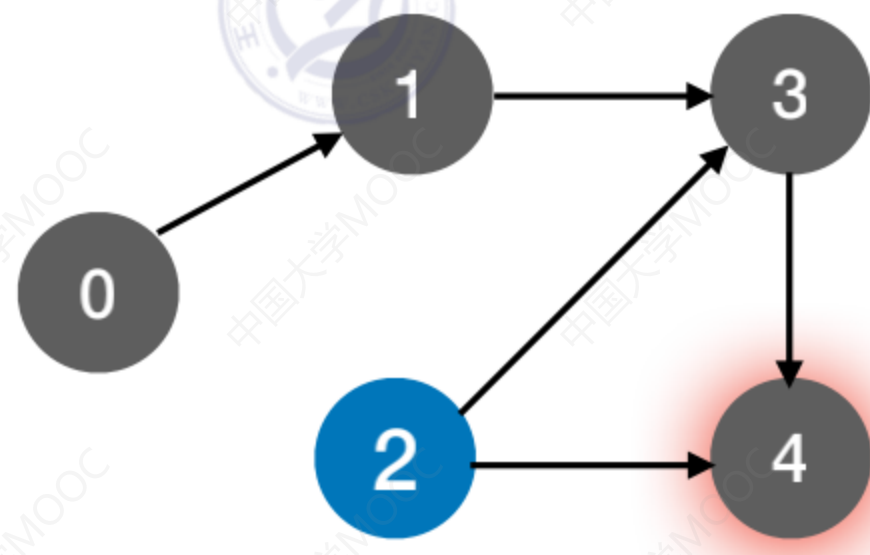
```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if

    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



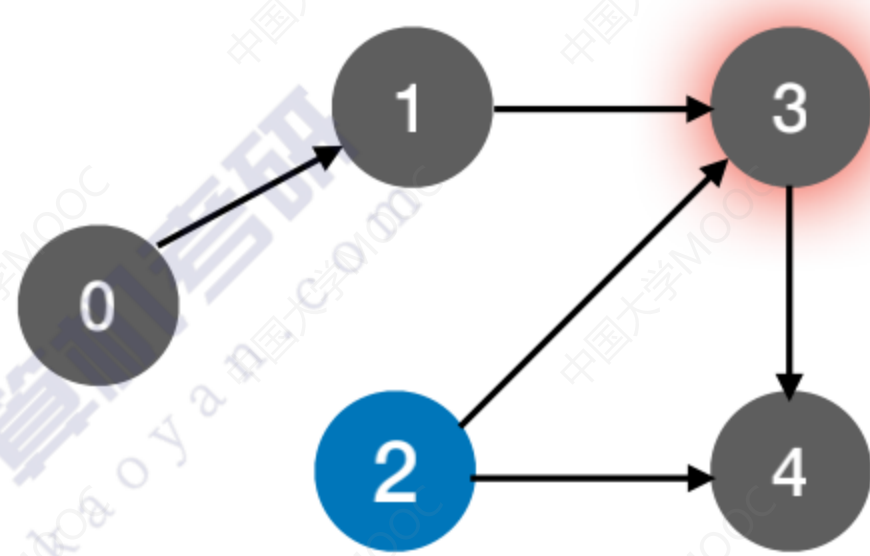
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



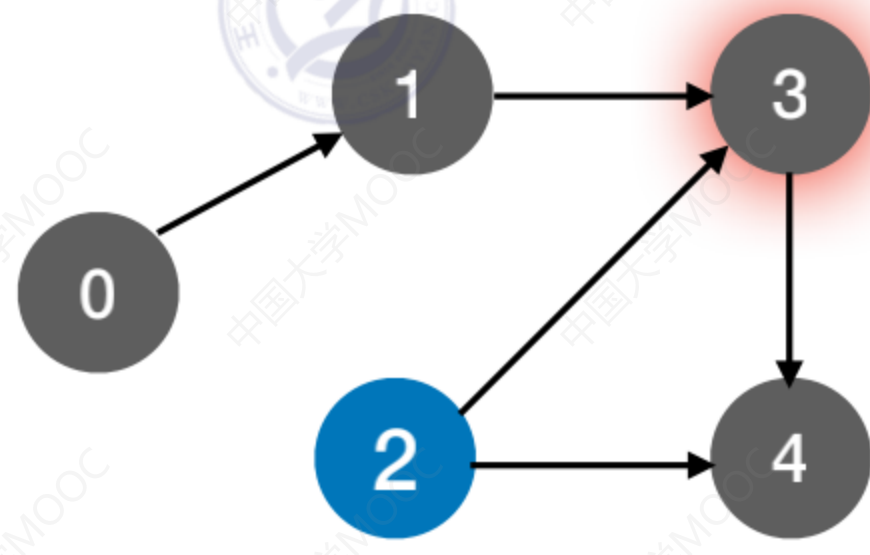
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



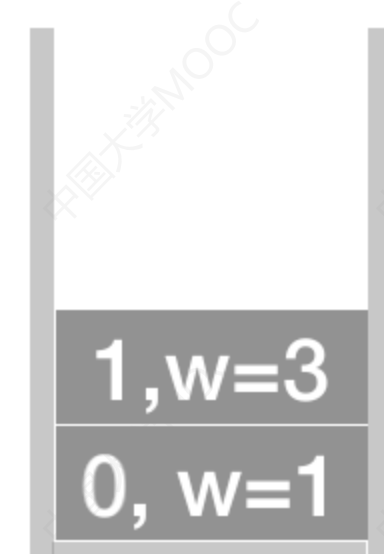
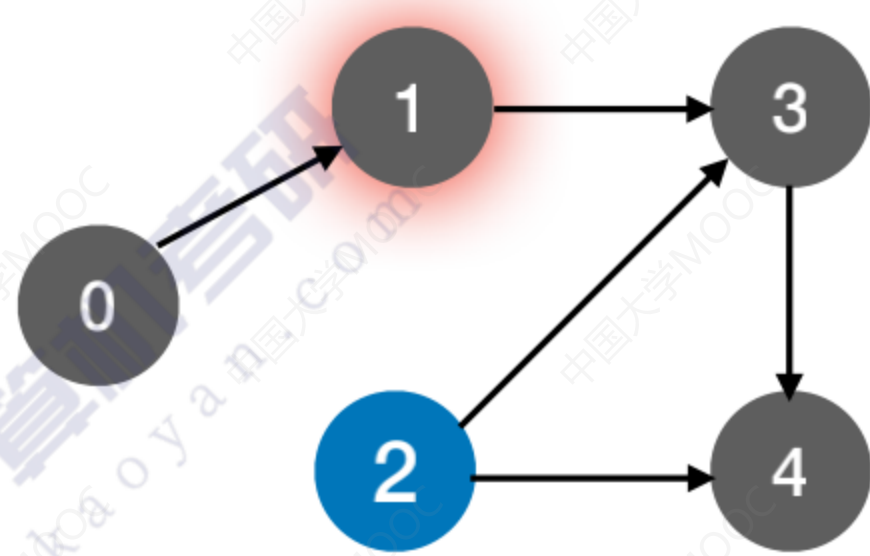
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



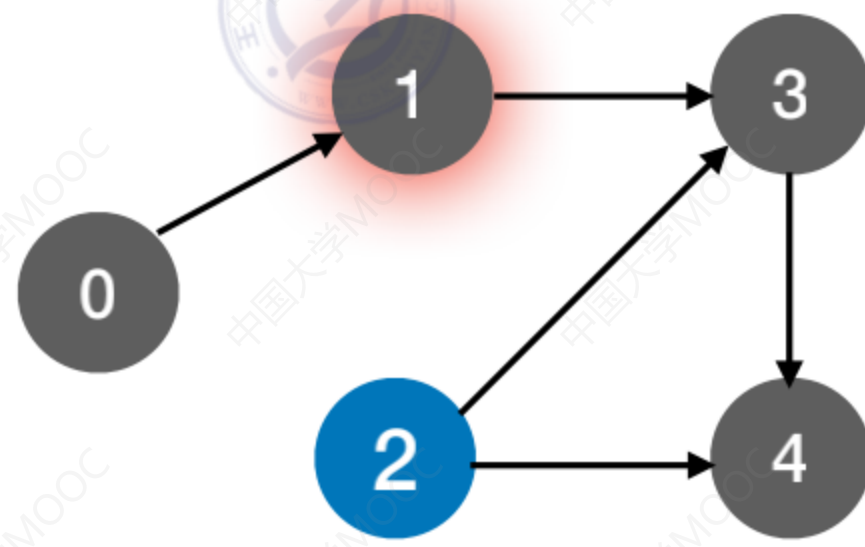
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



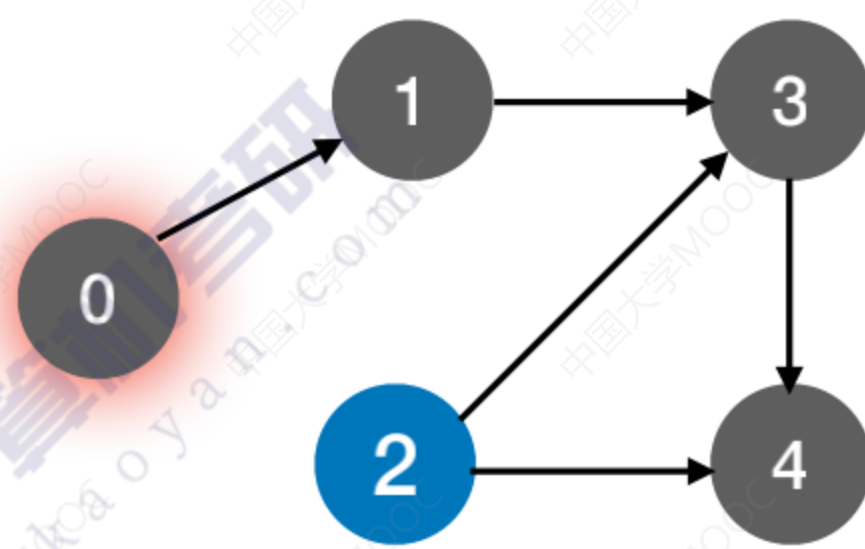
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



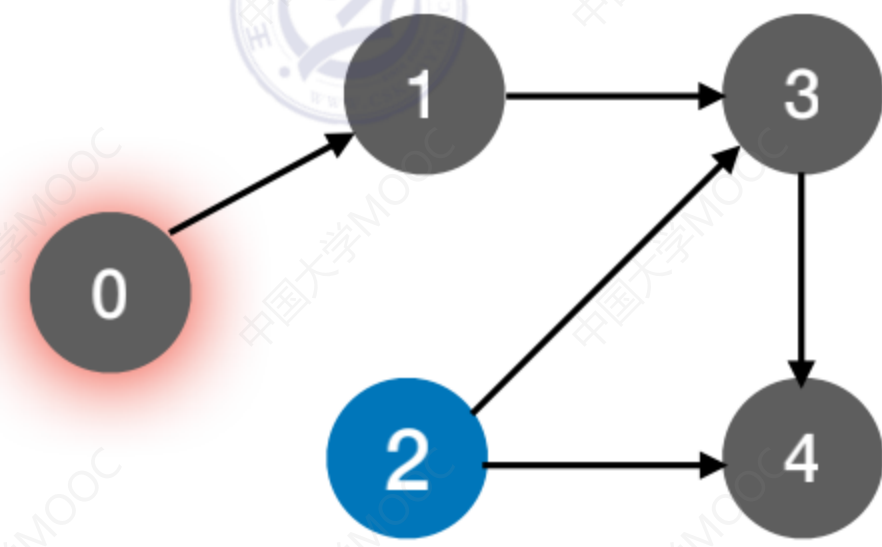
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



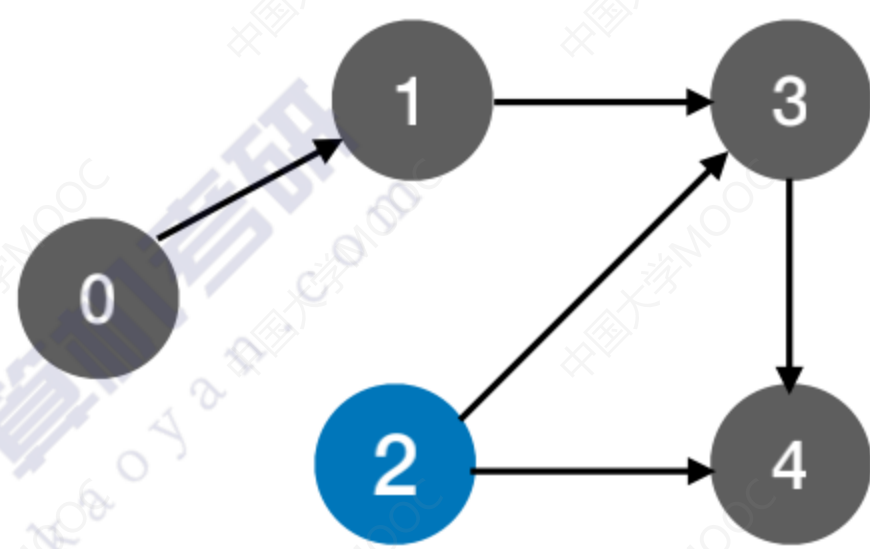
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1 0

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



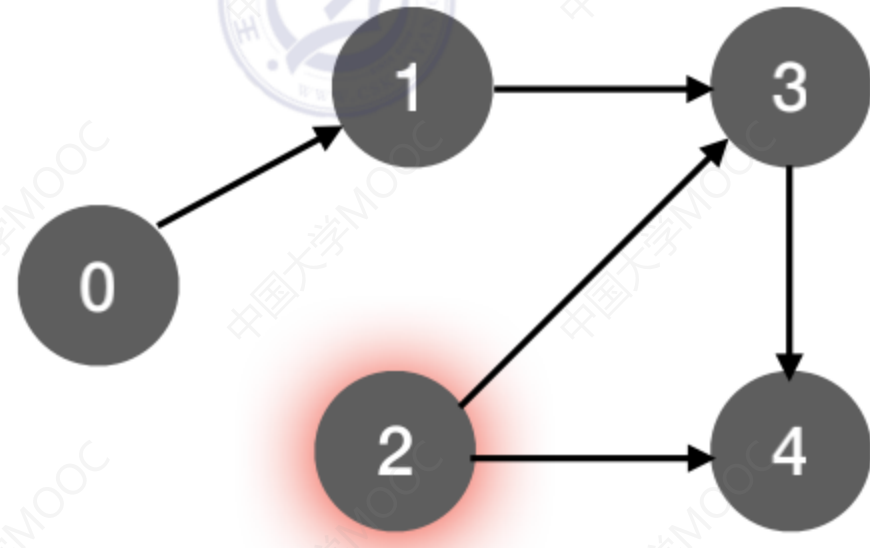
递归栈

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发,深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1 0

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

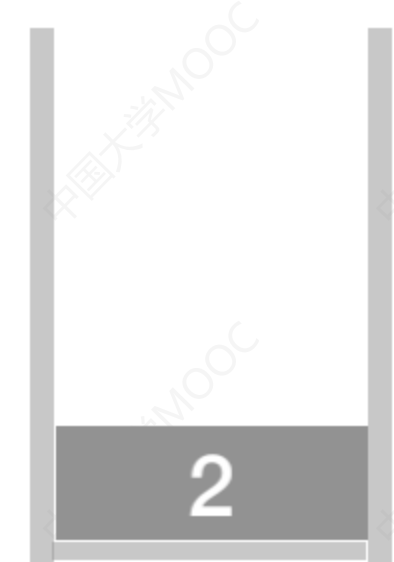
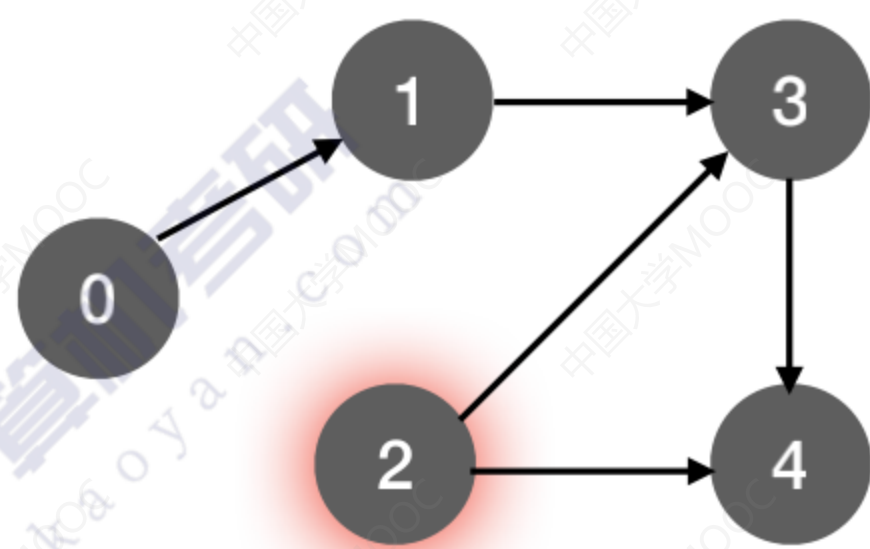
void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if

    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1 0

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}

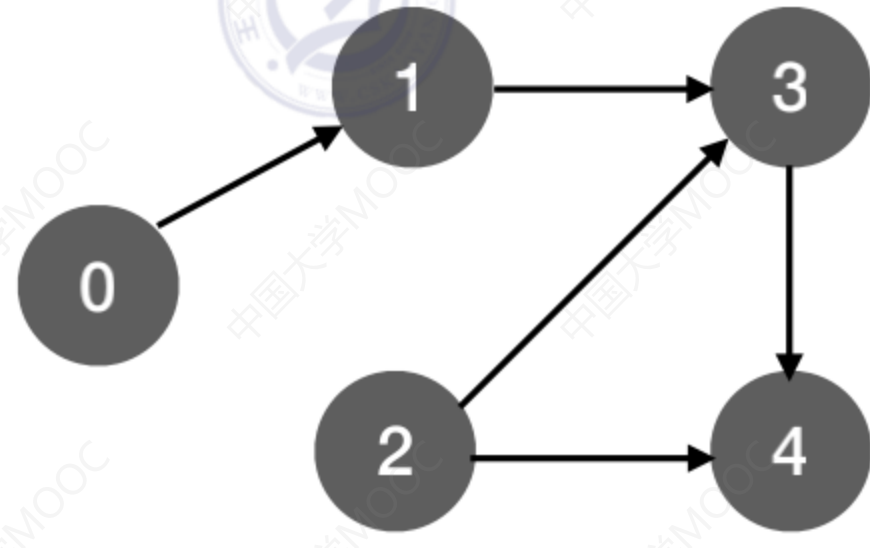
void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if

    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1 0 2

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



递归栈

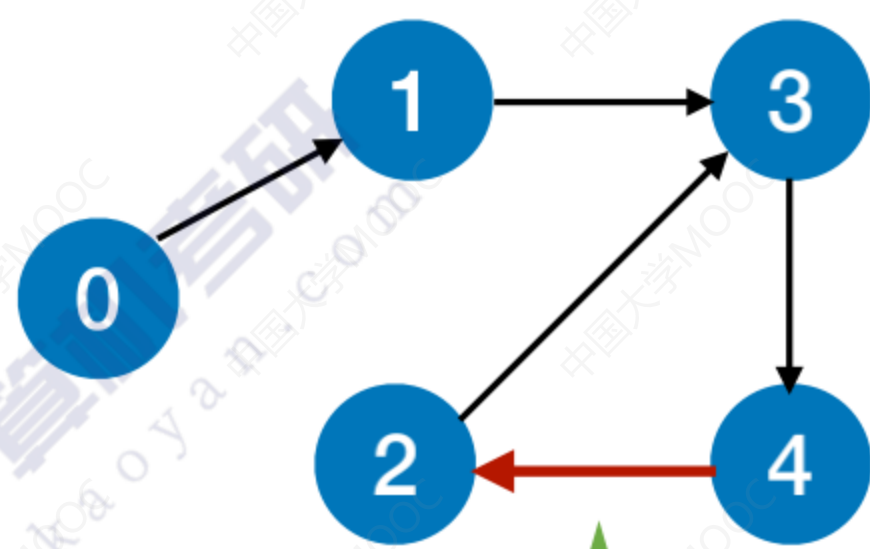
DFS实现逆拓扑排序：  
在顶点退栈前输出

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

逆拓扑排序序列: 4 3 1 0 2

王道考研/CSKAOYAN.COM

## 逆拓扑排序的实现 (DFS算法)



思考: 如果存在回路, 则不存在逆拓扑排序序列, 如何判断回路?



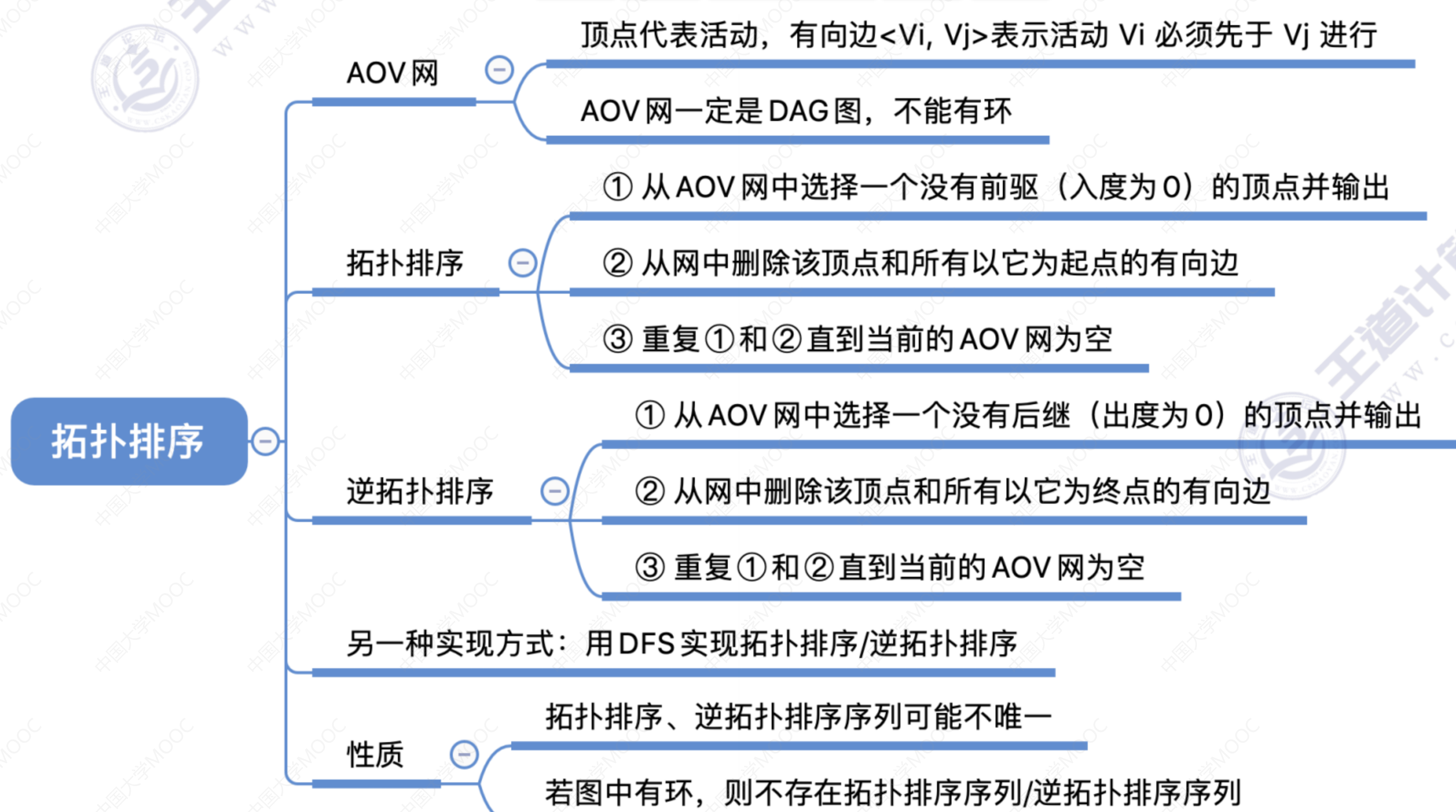
递归栈

DFS实现逆拓扑排序：  
在顶点退栈前输出

```
void DFSTraverse(Graph G){ //对图G进行深度优先遍历
    for(v=0;v<G.vexnum;++v)
        visited[v]=FALSE; //初始化已访问标记数据
    for(v=0;v<G.vexnum;++v) //本代码中是从v=0开始遍历
        if(!visited[v])
            DFS(G,v);
}
void DFS(Graph G,int v){ //从顶点v出发，深度优先遍历图G
    visited[v]=TRUE; //设已访问标记
    for(w=FirstNeighbor(G,v);w>=0;w=NextNeighbor(G,v,w))
        if(!visited[w]){ //w为u的尚未访问的邻接顶点
            DFS(G,w);
        } //if
    print(v); //输出顶点
}
```

王道考研/CSKAOYAN.COM

## 知识点回顾与重要考点



王道考研/CSKAOYAN.COM



@王道论坛



@王道计算机考研备考



@王道咸鱼老师-计算机考研

@王道楼楼老师-计算机考研



@王道计算机考研



知乎

@王道计算机考研

微信视频号

@王道计算机考研

微信公众平台

@王道在线