第14次课

算法分析与设计（2020）

Analysis and Design of Algorithm

任课教师：金嘉晖 副教授

办公室：东南大学九龙湖校区计算机楼368

Email：jjin@seu.edu.cn

助教：吴碧伟（220191682@seu.edu.cn）

回溯法的两种剪枝函数

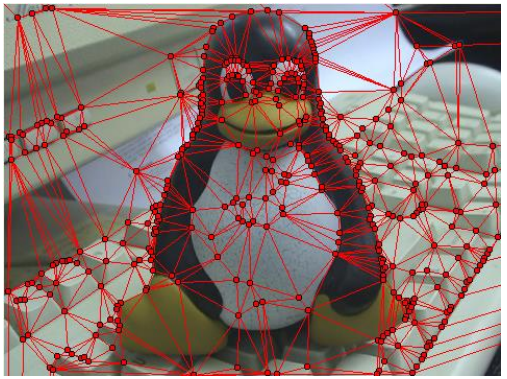
1. 可行性约束函数

2. **限界函数**

最大团问题及其应用

- 最大团问题（Maximum Clique Problem）
  - 图论中经典的组合优化问题
  - 完全图：图中任意两顶点都有边相连
  - 子图：若U是G的子图，则G含U中所有顶点和边
  - **目标：**找出图中顶点最多的所有完全子图

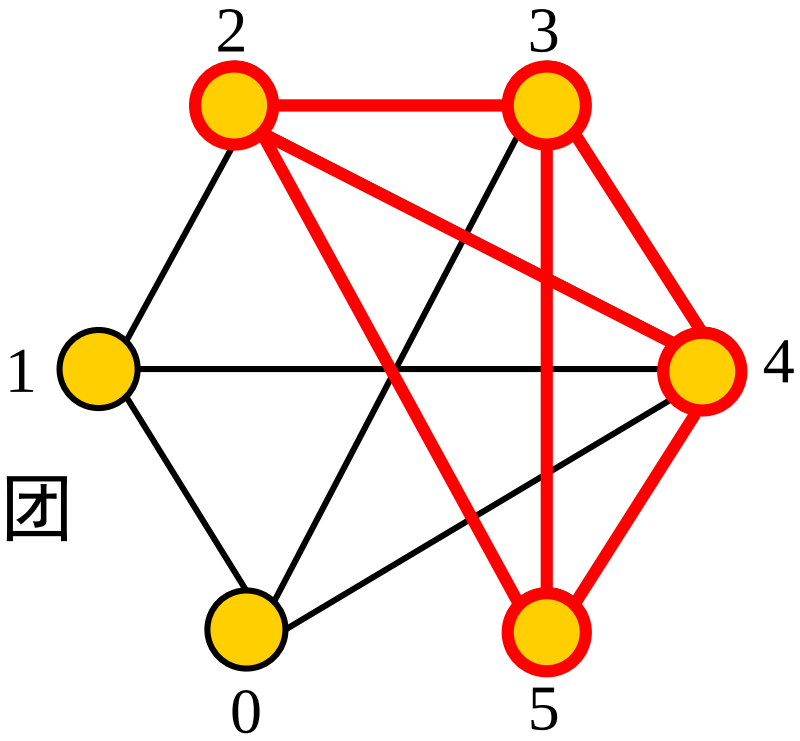
社会网络中的聚类分析



计算机视觉

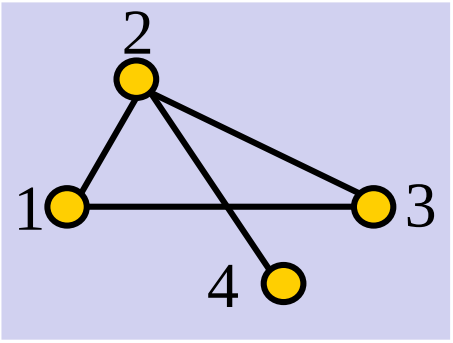
最大团问题的解空间

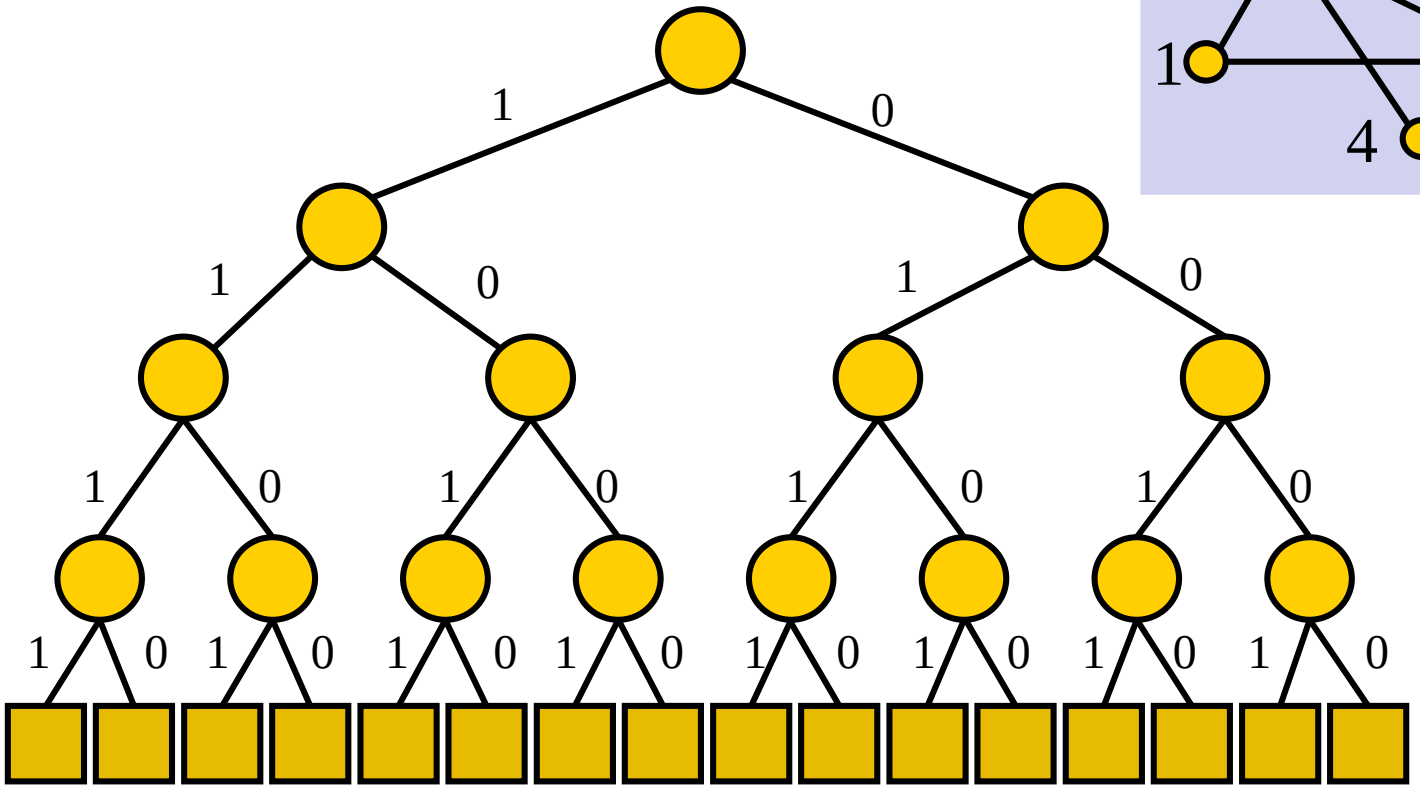
- 实例
  - 6个顶点，11条边
- 问题的解
  - 顶点2, 3, 4, 5构成最大团
  - 解的表示⟨0,0,1,1,1,1⟩



最大团问题的解空间树

- 一棵二叉树





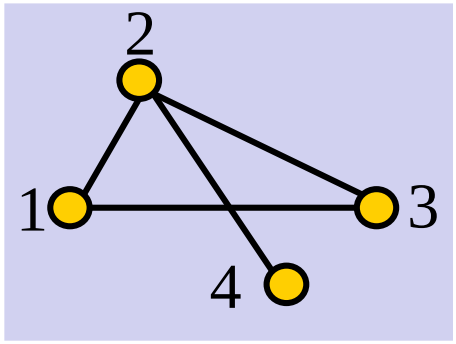
最大团问题的剪枝函数

- 可行性约束函数
  - 剪枝条件：当前的解不是团
- 假设顶点数为n，在第i层
- 限界函数
  - 界：已找到的最大团的顶点数（$bestn$）
  - 代价函数：

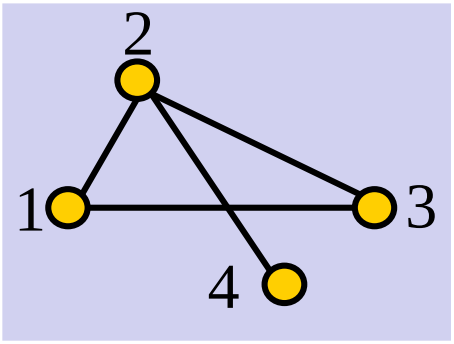
当前解的顶点数（cn）+未检查的顶点的数目（$n-i$）

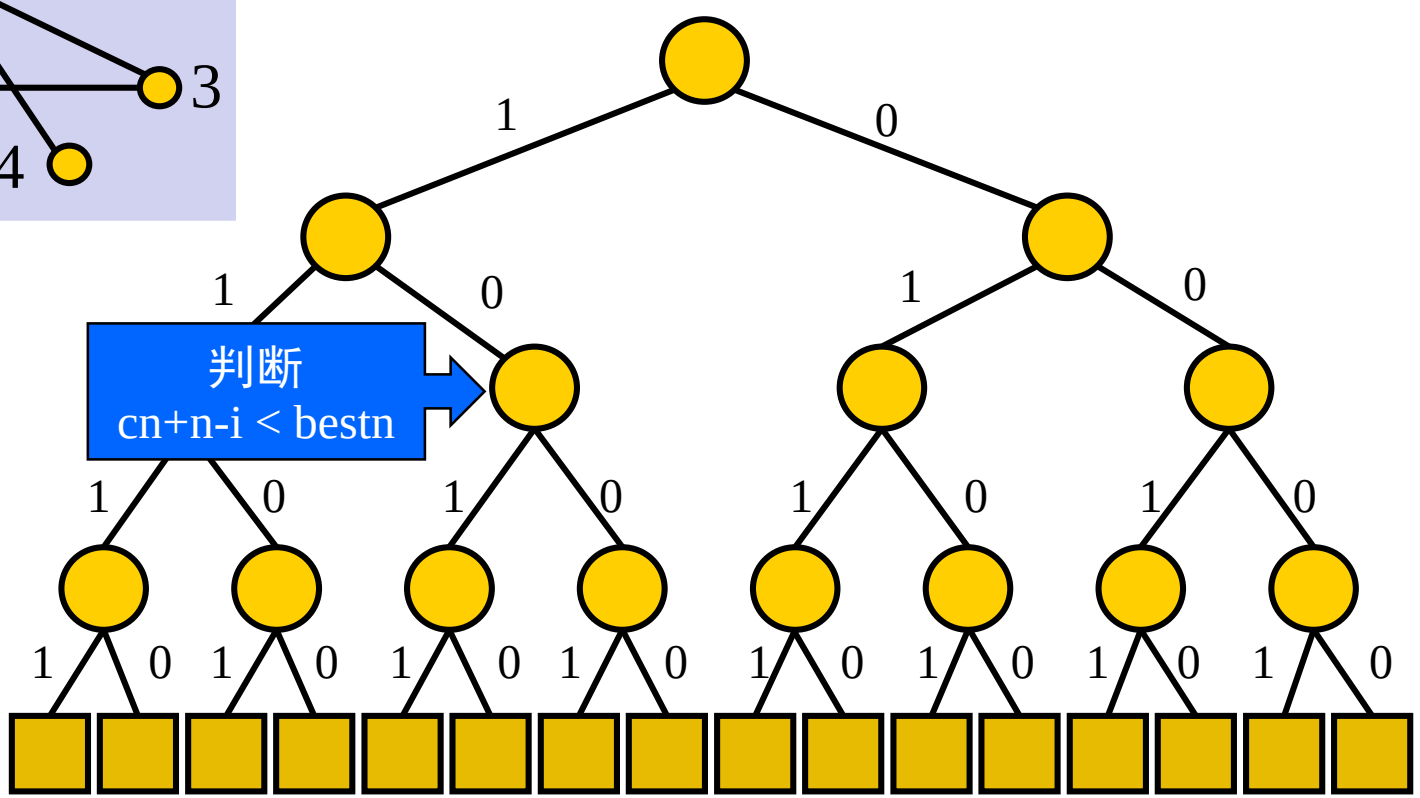
  - 剪枝条件：$cn+n-i < bestn$

剪枝之可行性约束函数

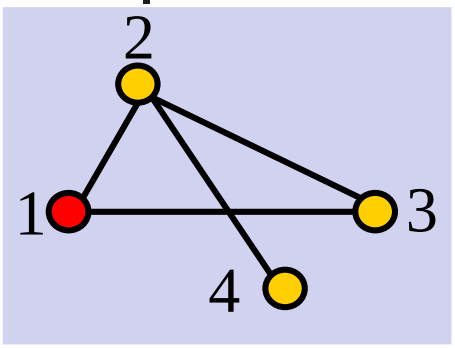


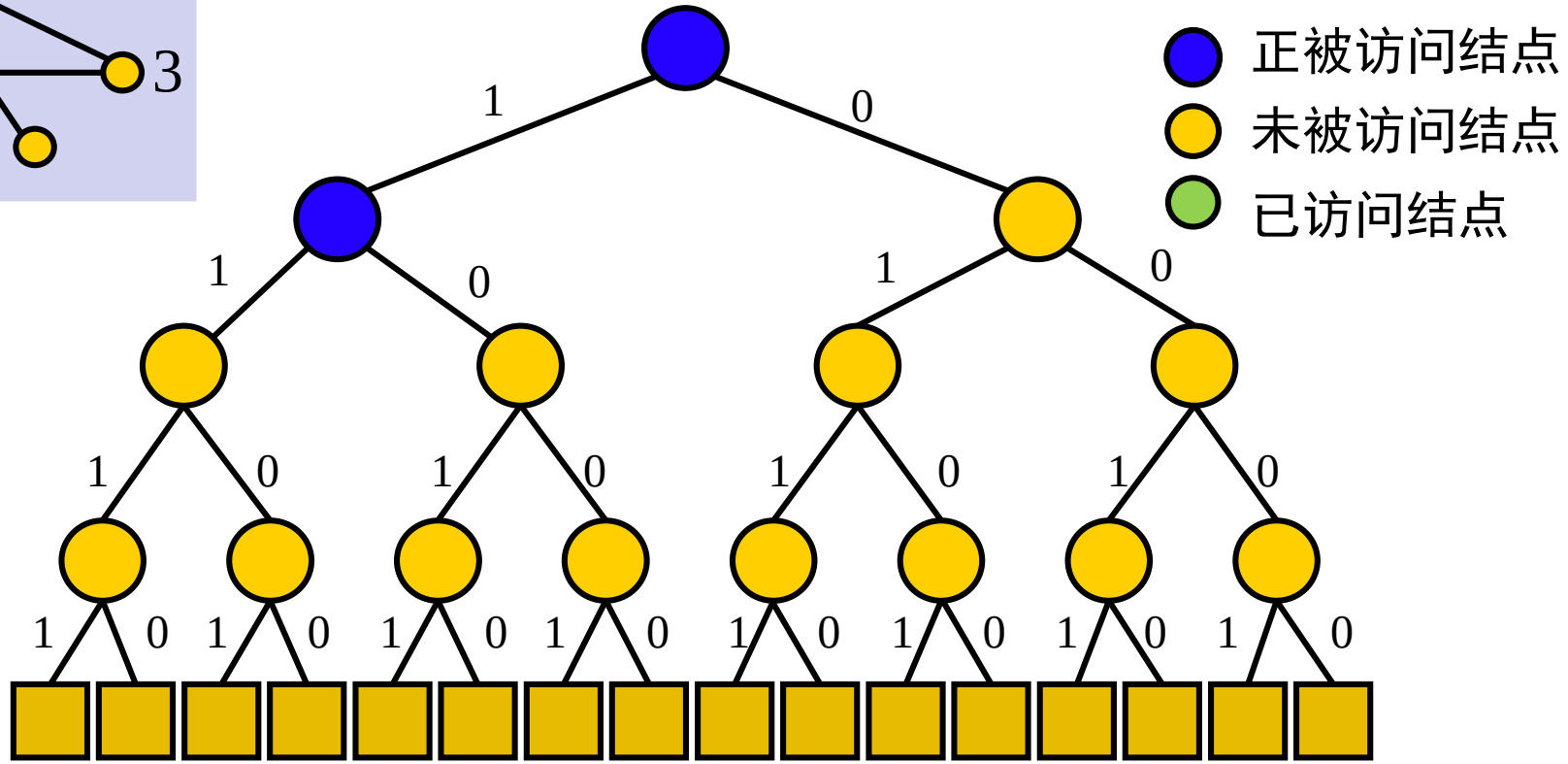
剪枝之限界函数



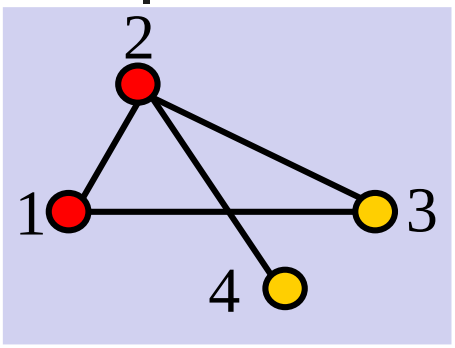


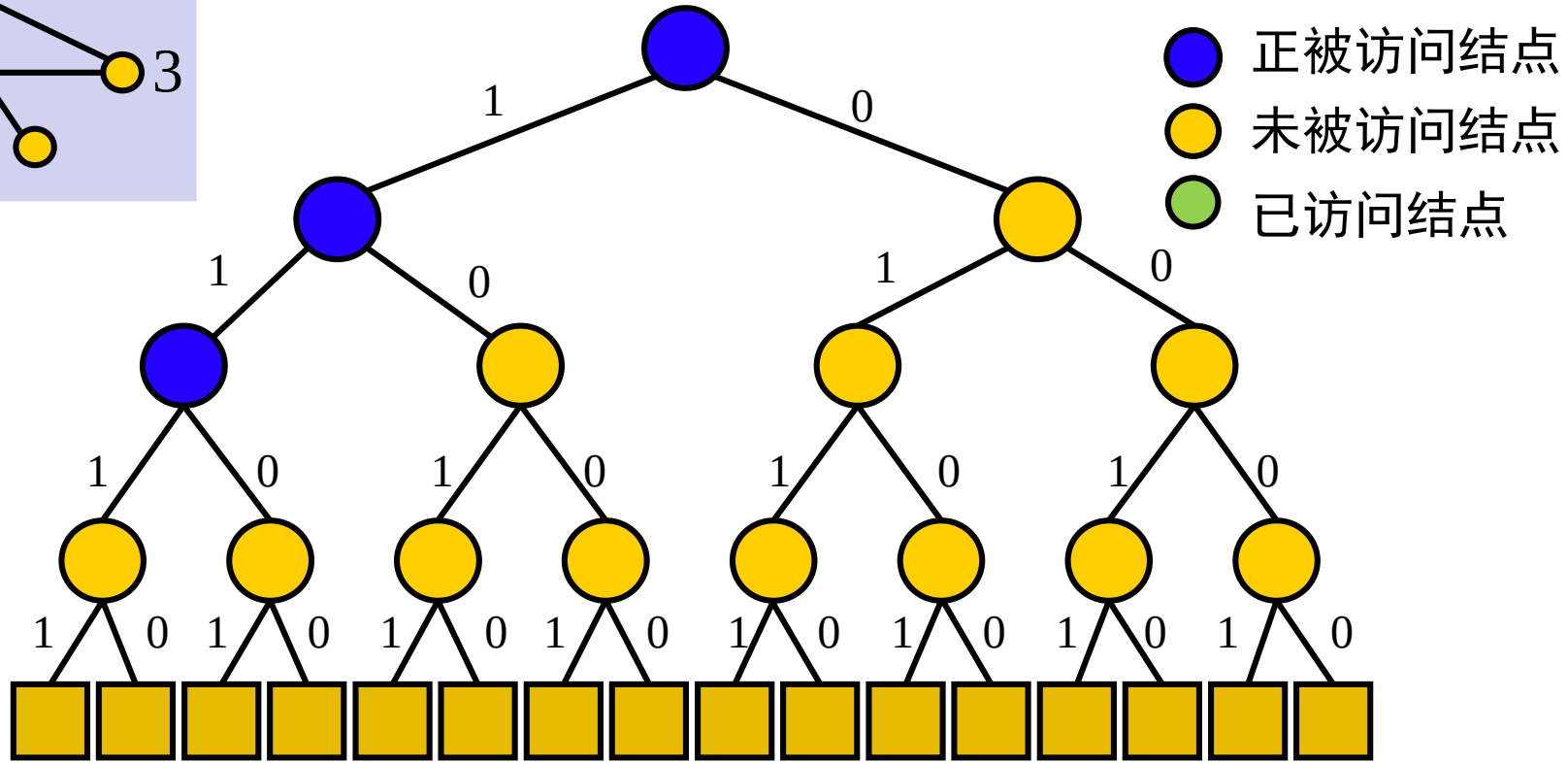
最大团问题的运行实例（1）



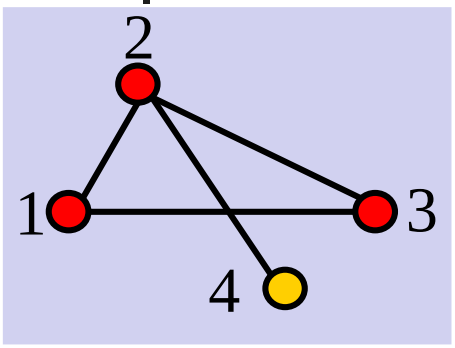


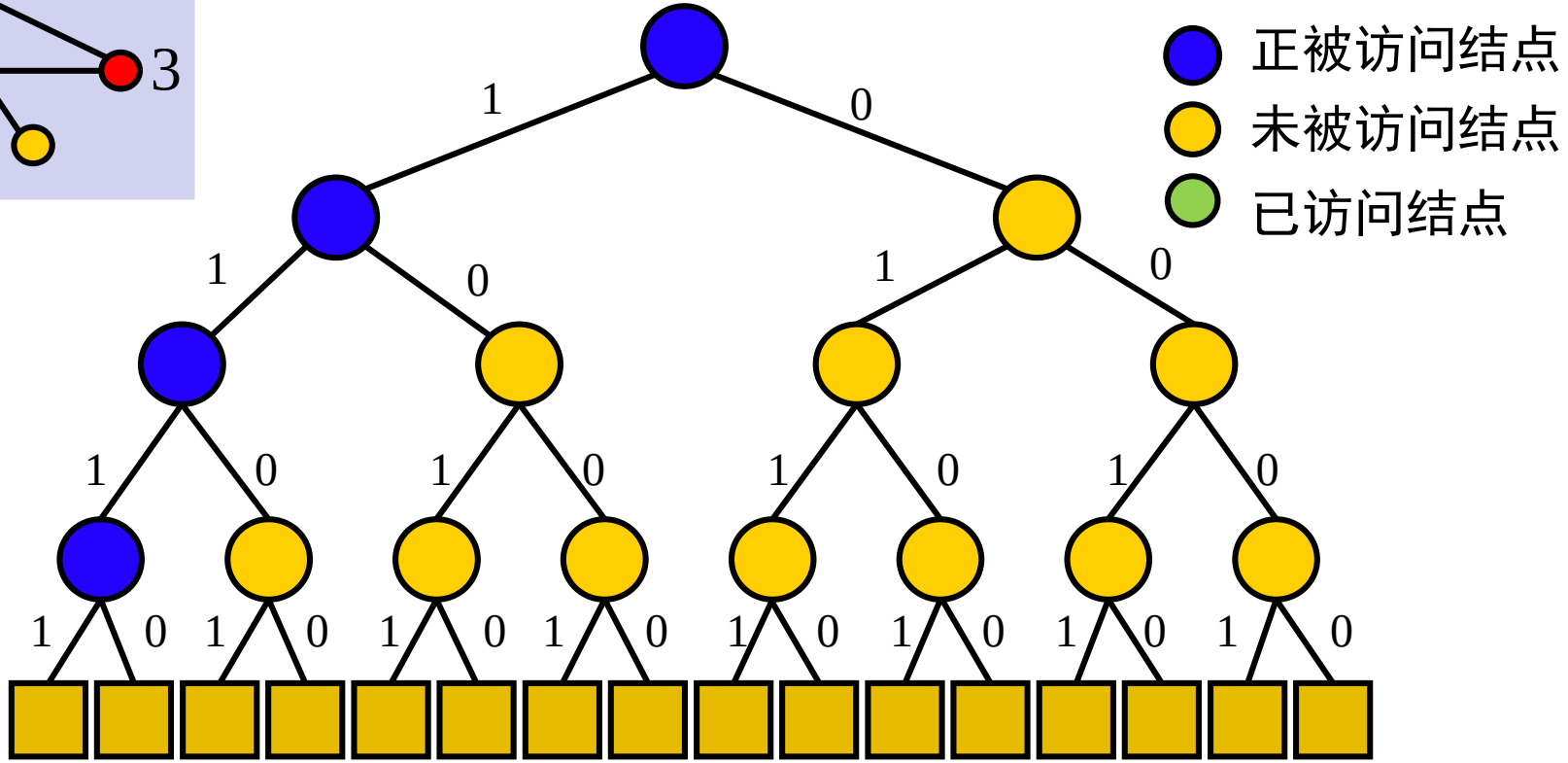
最大团问题的运行实例（2）



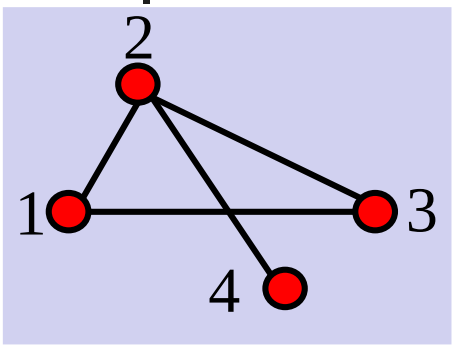


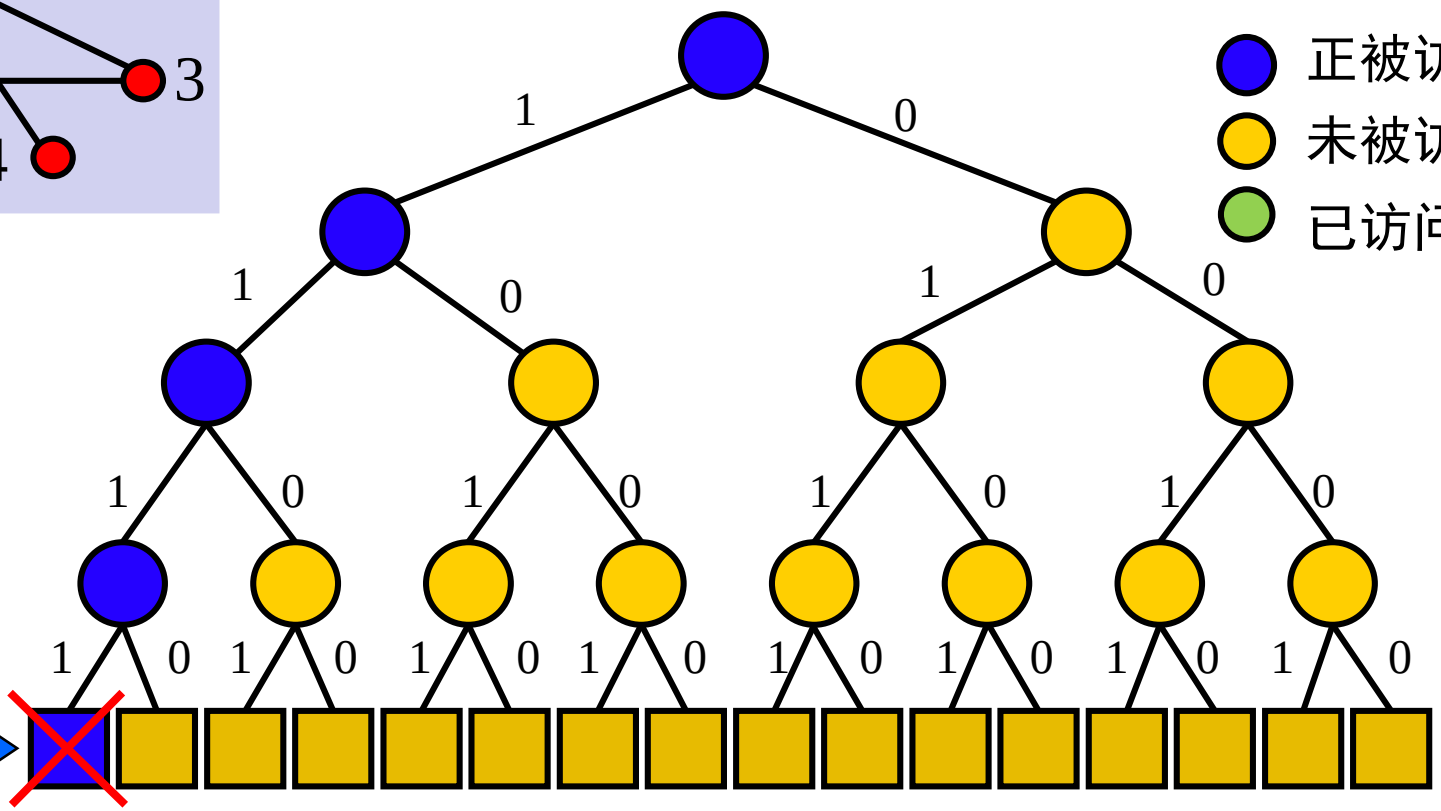
最大团问题的运行实例（3）





最大团问题的运行实例（4）





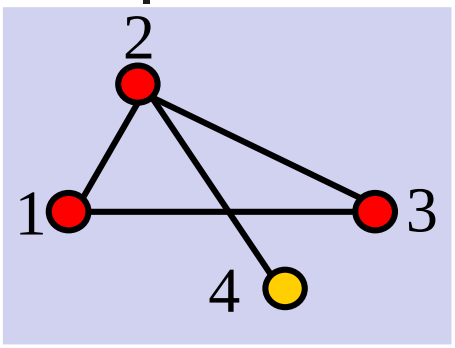
正被访问结点

未被访问结点

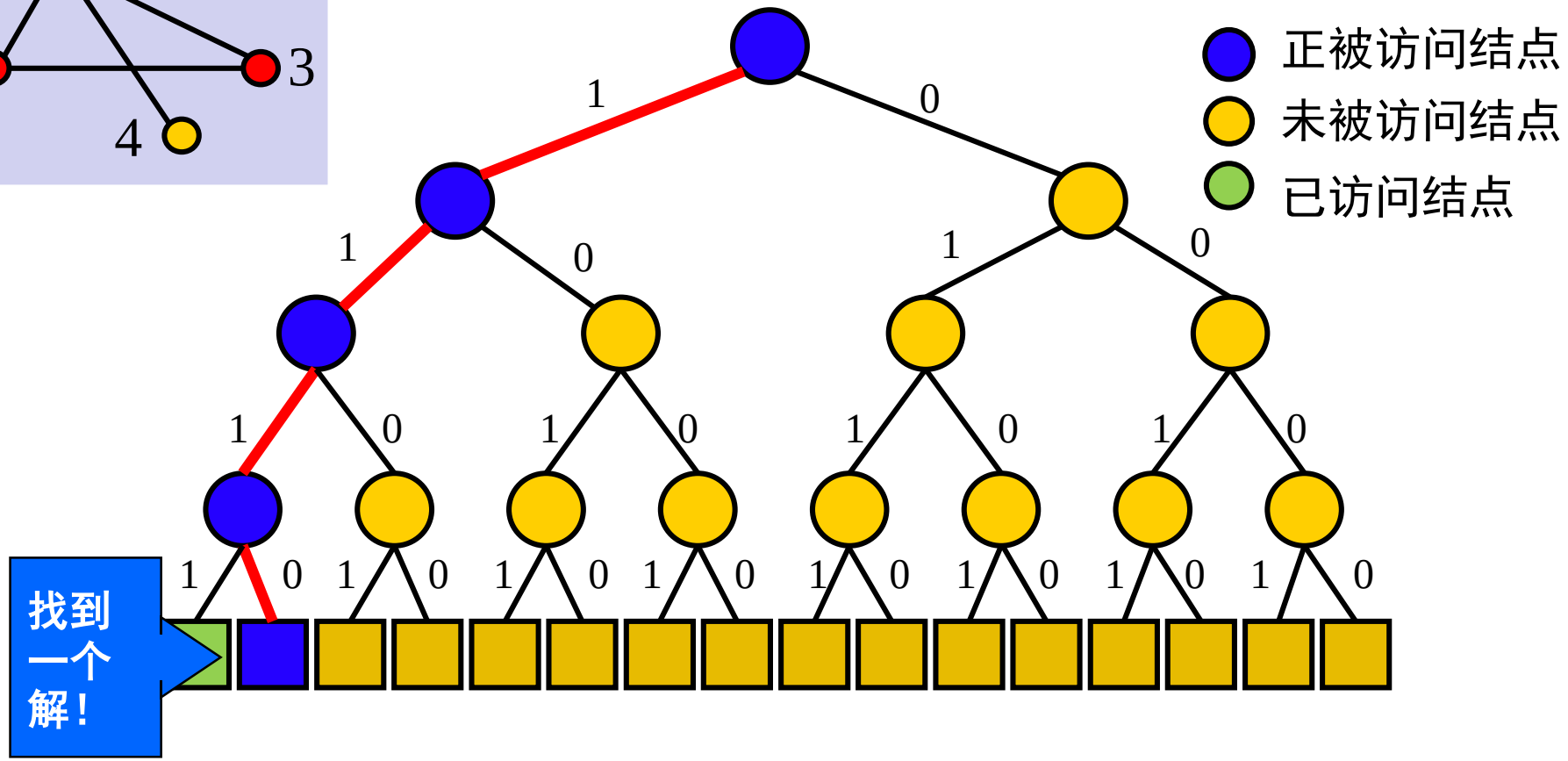
已访问结点

当前解不是团

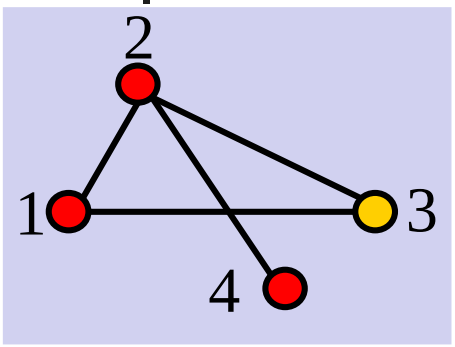
最大团问题的运行实例（5）



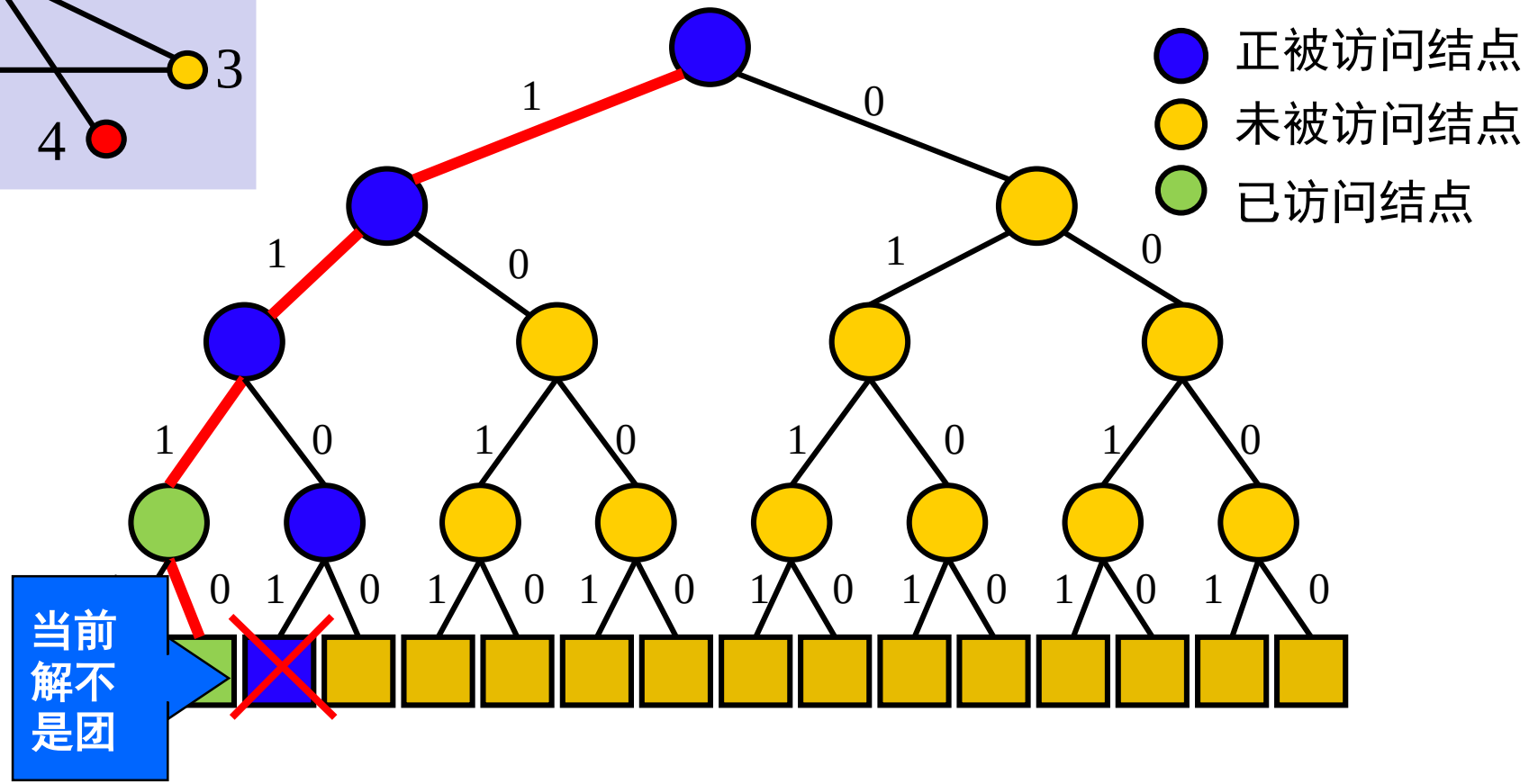
当前最大团规模为3



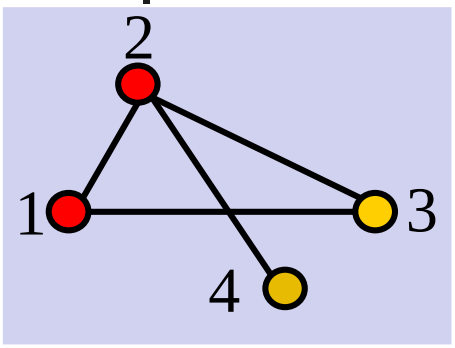
最大团问题的运行实例（6）



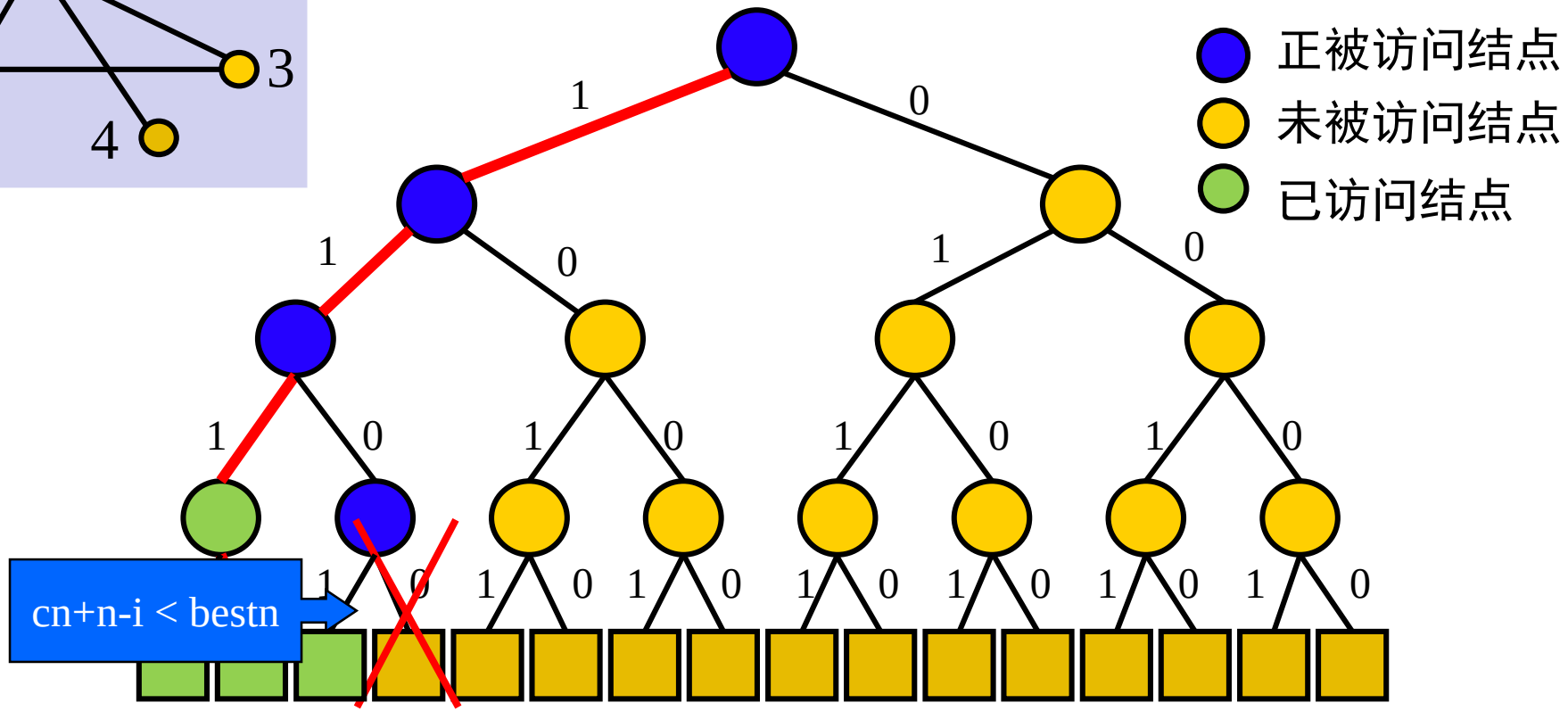
当前最大团规模为3



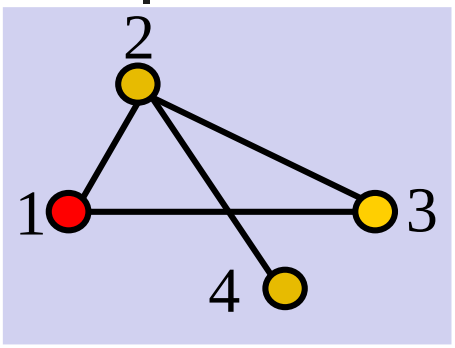
最大团问题的运行实例（7）



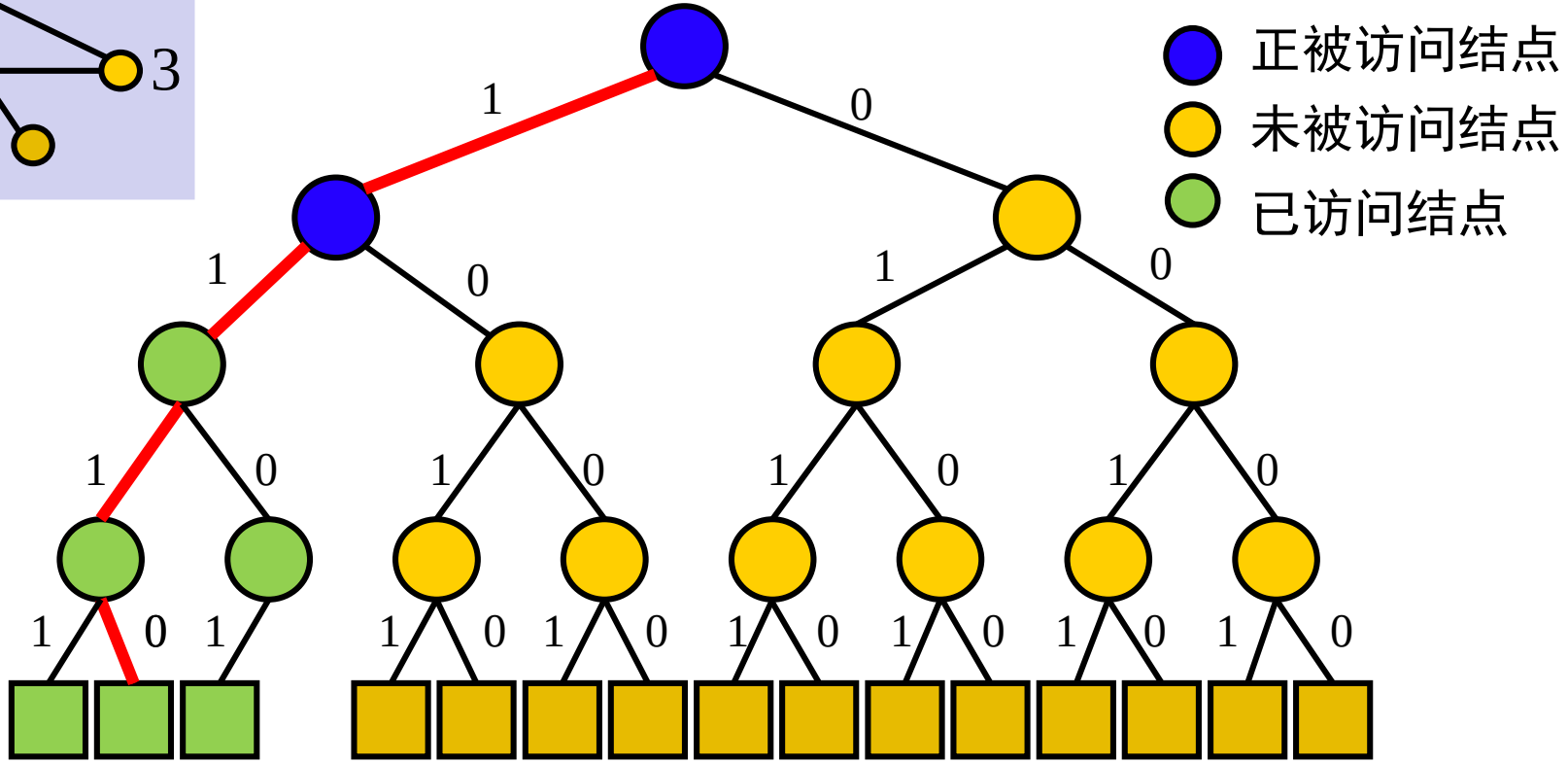
当前最大团规模为3



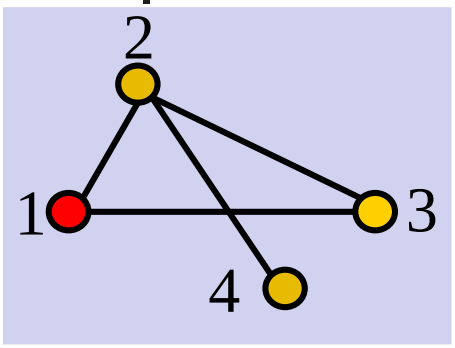
最大团问题的运行实例（8）



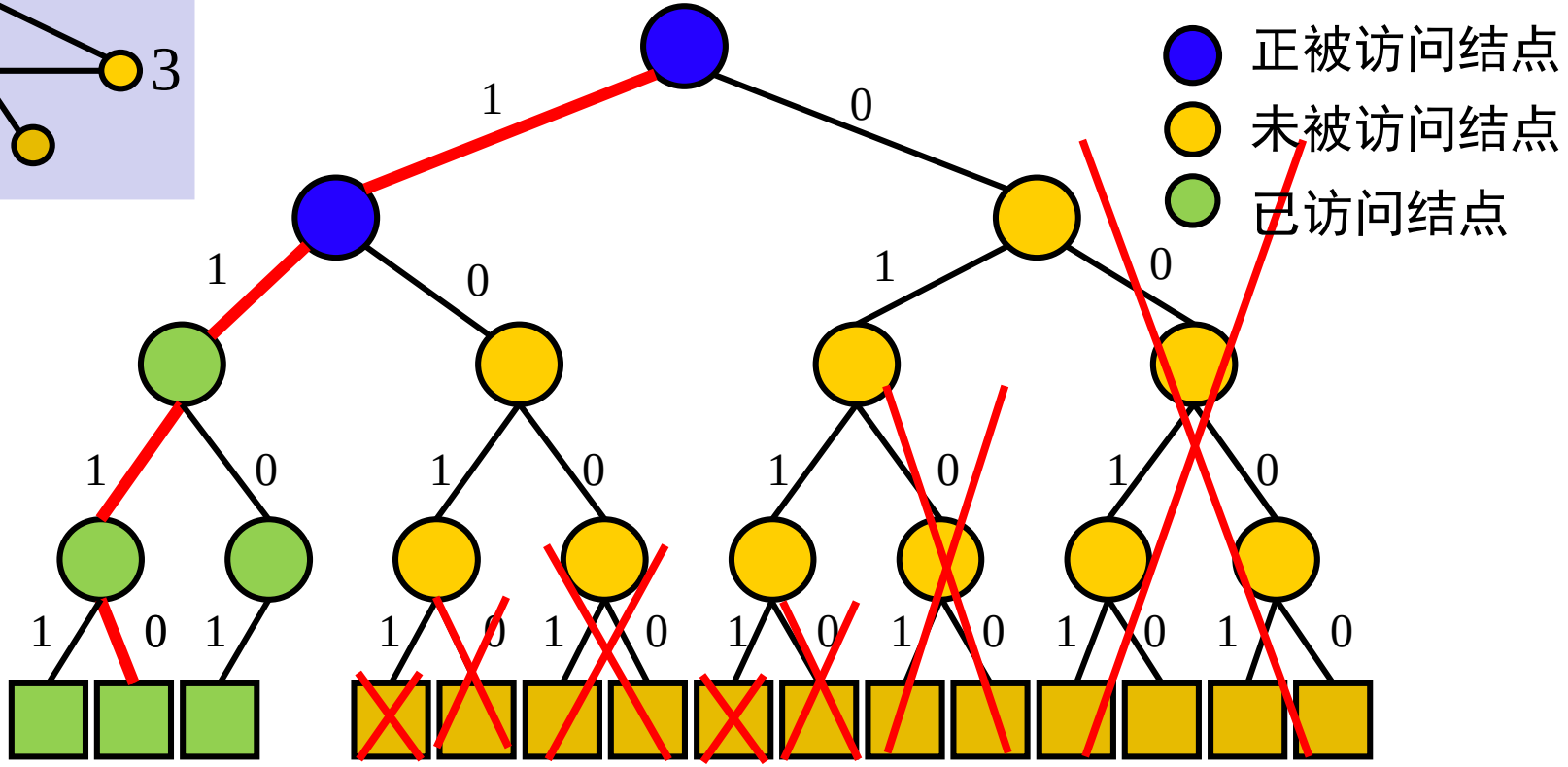
当前最大团规模为3



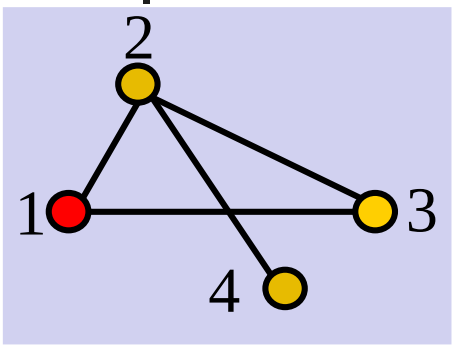
最大团问题的运行实例（9）



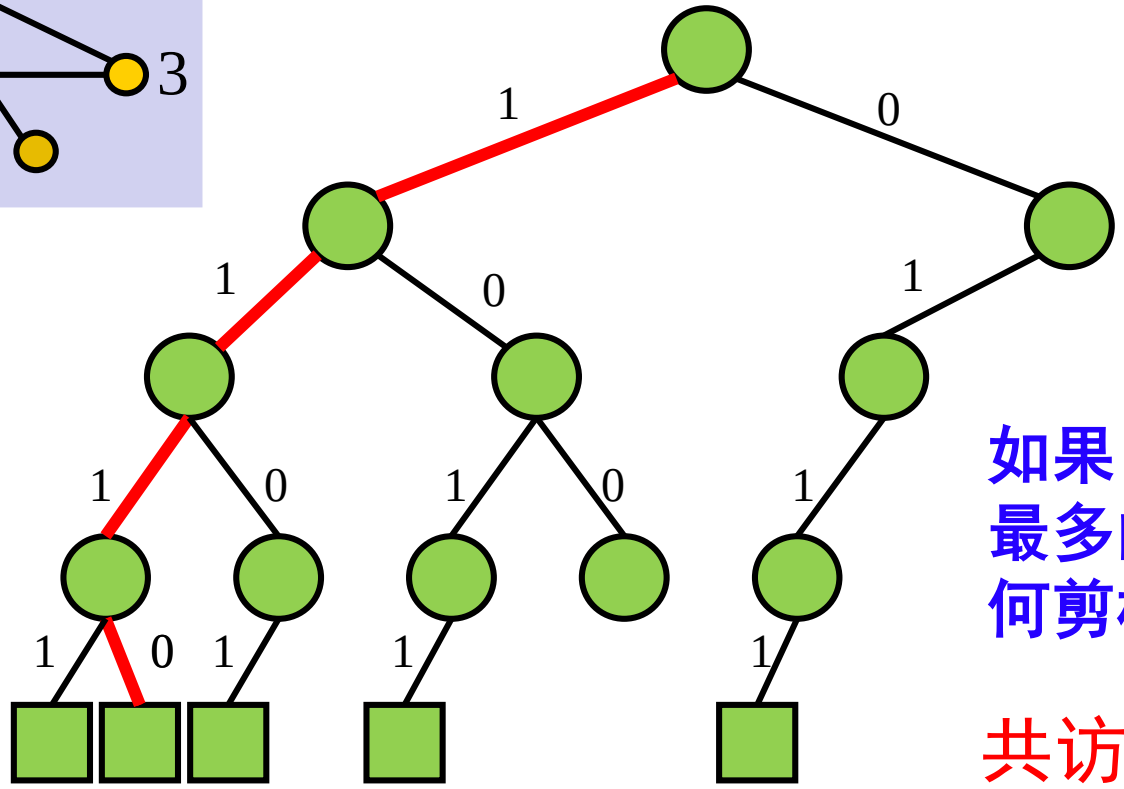
当前最大团规模为3



最大团问题的运行实例（9）



当前最大团规模为3



正被访问结点

未被访问结点

已访问结点

如果目标是找顶点最多的一个团，如何剪枝？

共访问了16个结点

最大团问题的算法实现

```
void Clique::Backtrack(int i){
    if (i > n) {
        for (int j = 1; j <= n; j++) bestx[j] = x[j];
        bestn = cn;
        return;
    }
    int OK = 1;
    for (int j = 1; j < i; j++)
        if (x[j] && a[i][j] == 0) {OK = 0;  break;}

    if
    if
}
```

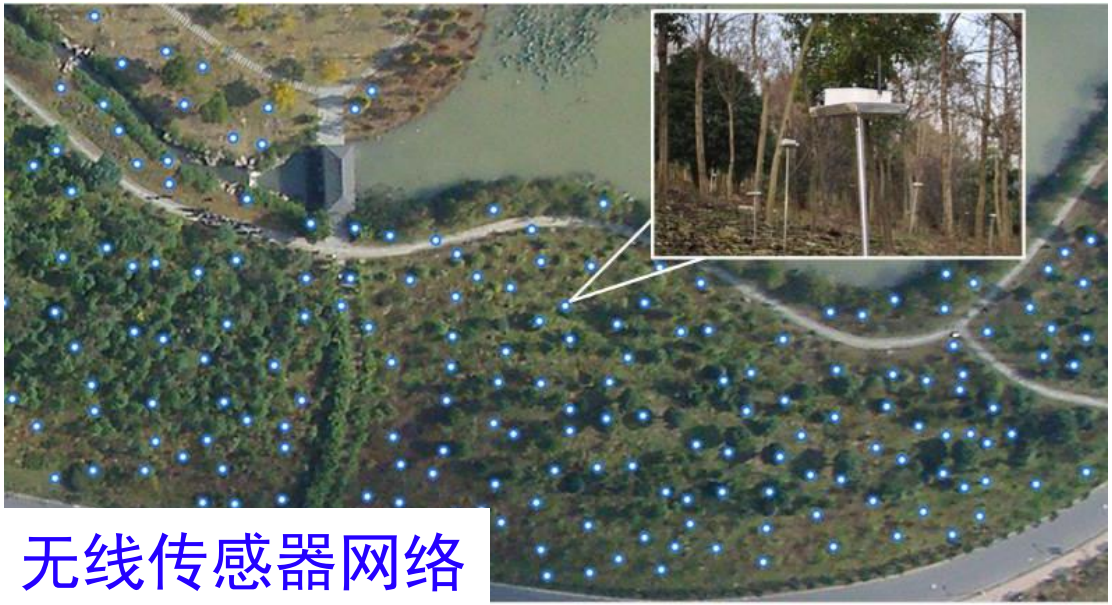
到达叶结点

复杂度分析

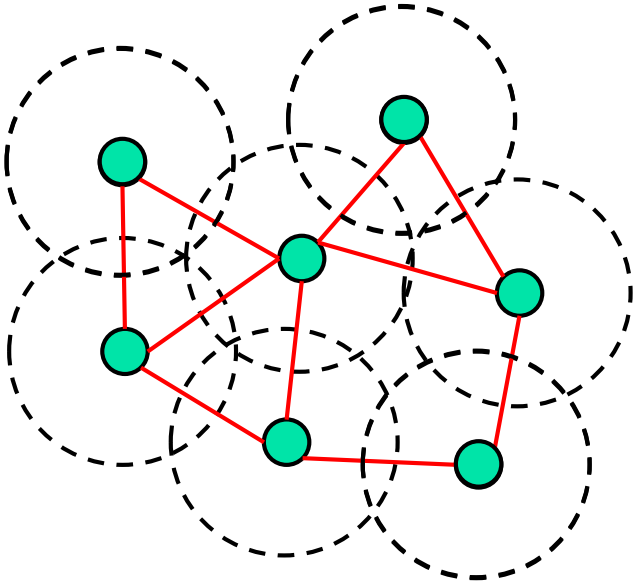
最大团问题的回溯算法**backtrack**所需的计算时间为$O(n2^n)$。

最大独立集问题（最大团问题的推广）

- 最大独立集问题（Maximal Independent Set Problem）
  - 独立集：任意两顶点都没边相连的顶点集合
  - **目标：**找出图中最大的独立集

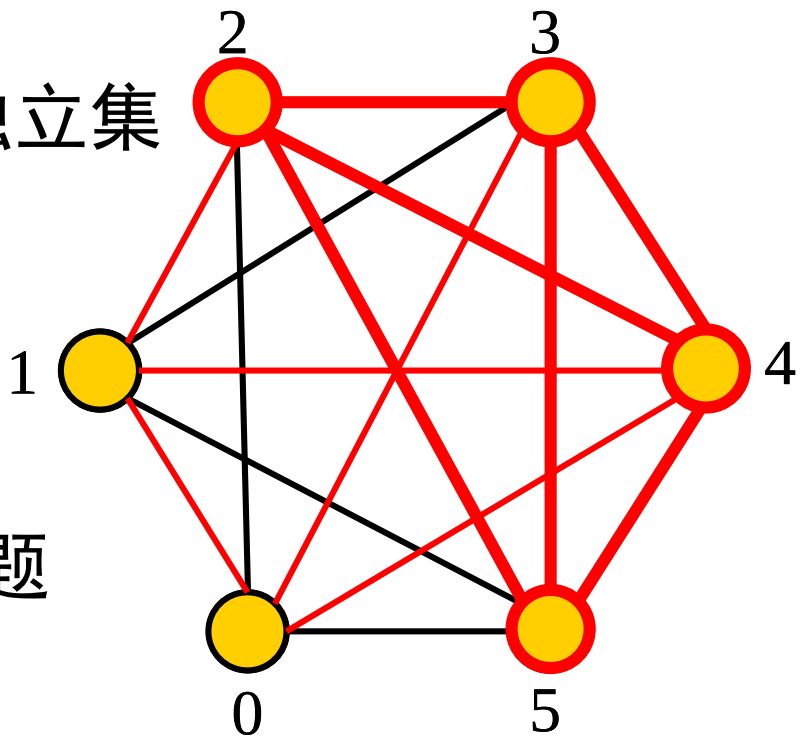


无线传感器网络



最大独立集问题的求解思路

- 问题的解
  - 顶点2, 3, 4, 5构成最大独立集
  - 解的表示⟨0,0,1,1,1,1⟩
- 求解思路
  - 等价于补图的最大团问题



提高回溯法效率的技巧

- 对输入的序列排序
  - 0-1背包（按单位价值由重自轻）
  - 最大团(按度数排）
- 设计精确的代价函数
  - 圆排列问题（数学推导）

课程回顾

- 回溯法的概念
  - 一种通用的求解解法
  - 具有剪枝函数的深度优先生成法
- 回溯法的核心问题
  - 构造解空间树（子集树、排列树、*n*叉树）
  - 设计剪枝函数（可行性约束函数、限界函数）
- 回溯法的应用
  - 装载问题；批处理作业调度；*n*后问题；0-1背包问题；最大团问题；图的*m*着色问题；旅行商问题、圆排列问题

第六章 分支限界法

学习要点

- 理解分支限界法的剪枝搜索策略
- 掌握用分支限界法的算法框架
  - 队列式（FIFO）分支限界法
  - 优先队列式分支限界法

- 应用范例
  - 旅行商问题、0-1背包问题、装载问题

从一个例子开始

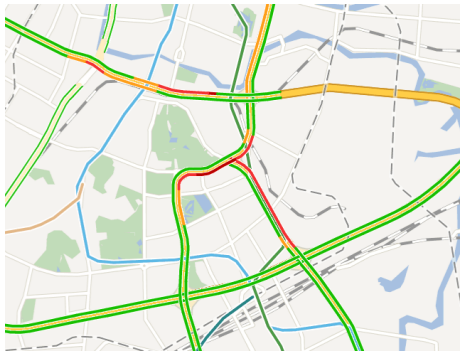
- 华同学的巡回演唱会
  - **地点：**深圳、武汉、北京、南京、杭州、泉州、广州重庆、长沙、合肥
  - **线路：**从深圳出发，跑遍各大城市，回到深圳

- 目标：考虑机票价格，确定票价最少的线路

票价	深圳	武汉	北京	……
深圳	0	500	600	……
武汉	100	0	800	……
北京	1000	200	0	……
……	……	……	……	……

非对称旅行商问题

- 问题定义
  - 城市集合：$C = \{c_1, c_2, \cdots c_n\}$
  - 城市距离：$d(c_i, c_j)$
  - 距离不对称：$d(c_i, c_j) \neq d(c_j, c_i)$
- 目标：求遍历所有城市（不重复）的最短路径



道路拥堵情况下
的送快递问题

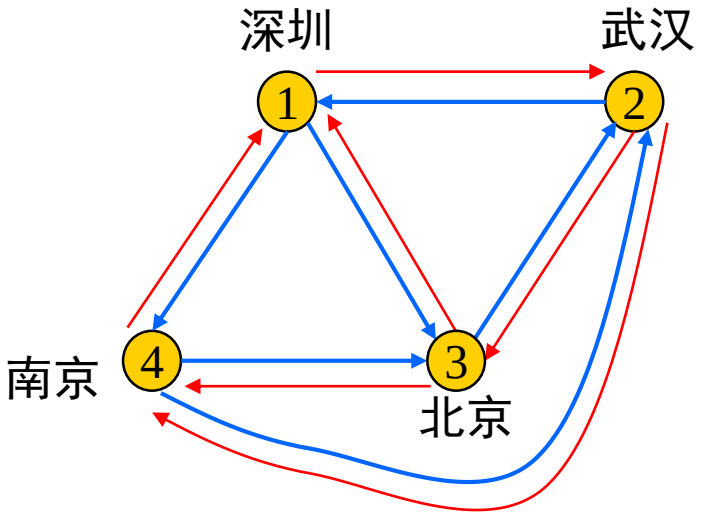
考虑城市单行线
的送快递问题

全国巡回演唱会
的路线安排问题

非对称旅行商问题的解空间

- 实例

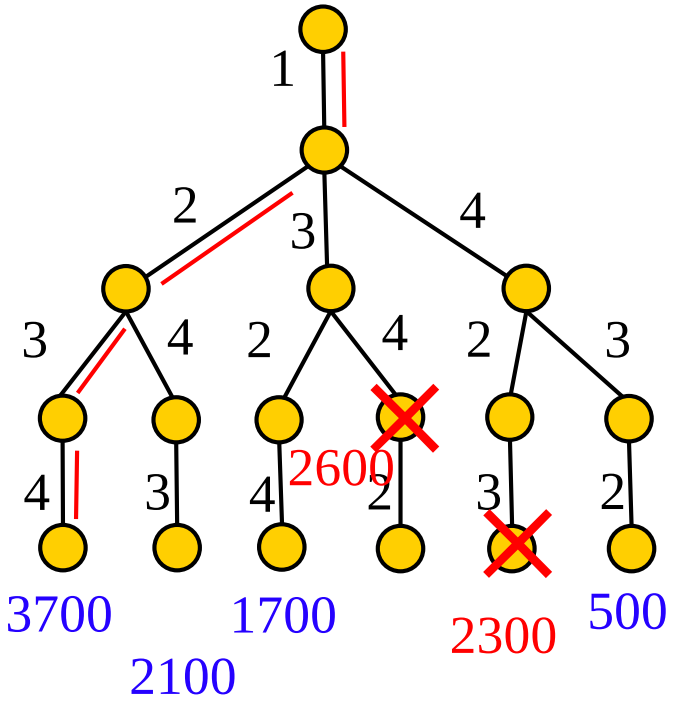
票价	深圳	武汉	北京	南京
深圳	0	500	600	100
武汉	100	0	800	500
北京	1000	200	0	2000
南京	400	400	100	0



- 最优解
  - 解的表示：⟨1,4,3,2⟩
  - 路线：深圳→南京→北京→武汉→深圳
  - 总票价：100+100+200+100=500

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0

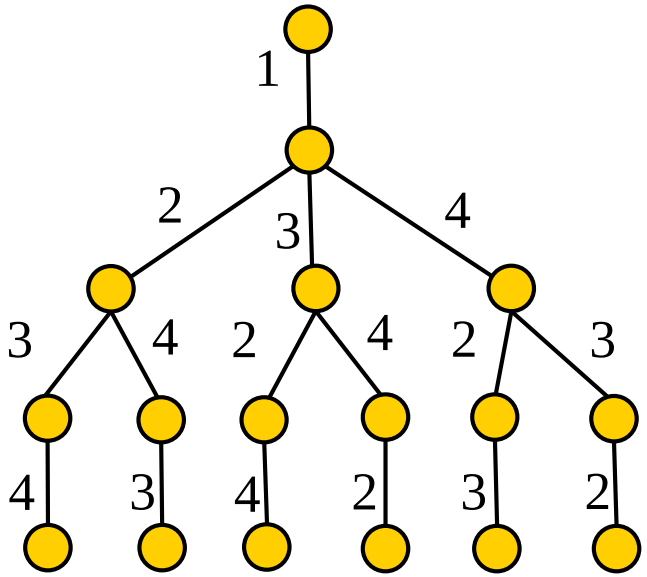


- 回溯法
  - 深度优先遍历解空间树
  - 剪枝函数：比较当前解与当前的界

是否可以知道该优先遍历哪些子树？

非对称旅行商问题的解空间树

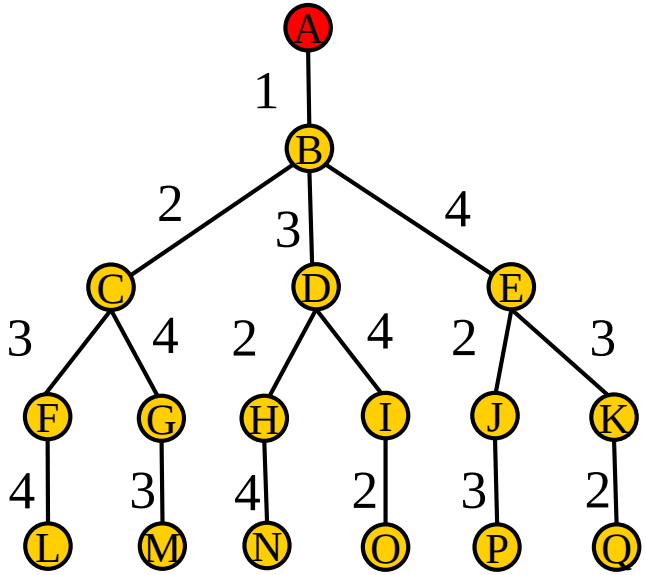
票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



- 如何尽快访问最优解
  - 不用深度优先搜索
  - 优先访问更靠近最优解的节点
  - 设置代价函数计算优先级
  - 利用优先级队列管理节点
- 如何设计优先级函数
  - 可以令优先级函数等于代价函数
  - 当前解的值 + 未来的最优解估计值

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

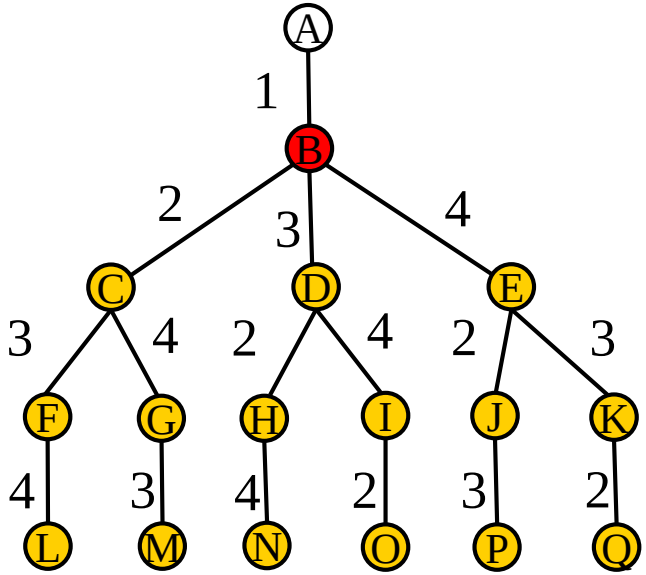
节点	当前值	未来最优值	总代价
A	0	500	500

优先级函数：=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

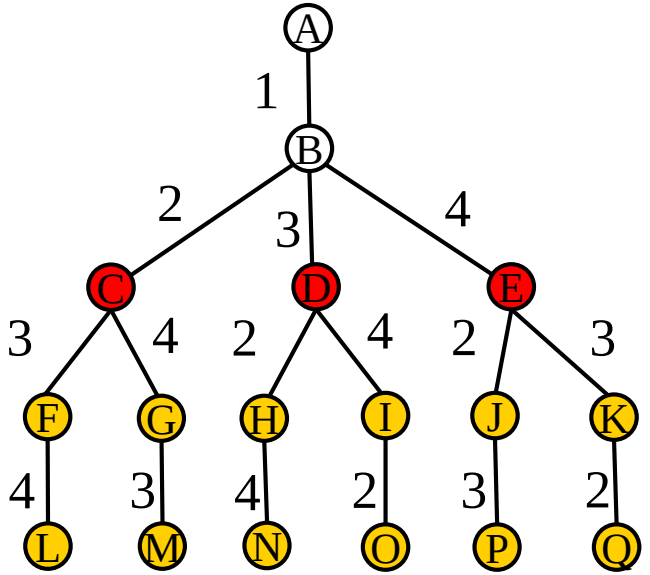
节点	当前值	未来最优值	总代价
B	0	500	500

优先级函数=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

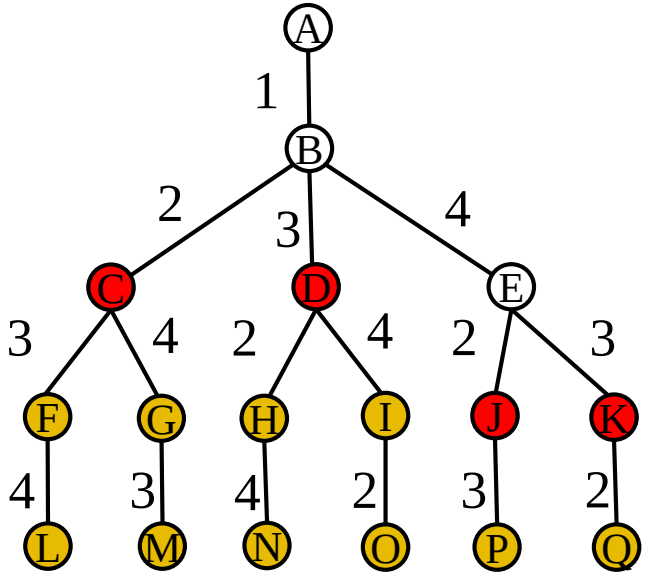
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
E	100	400	500

优先级函数=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

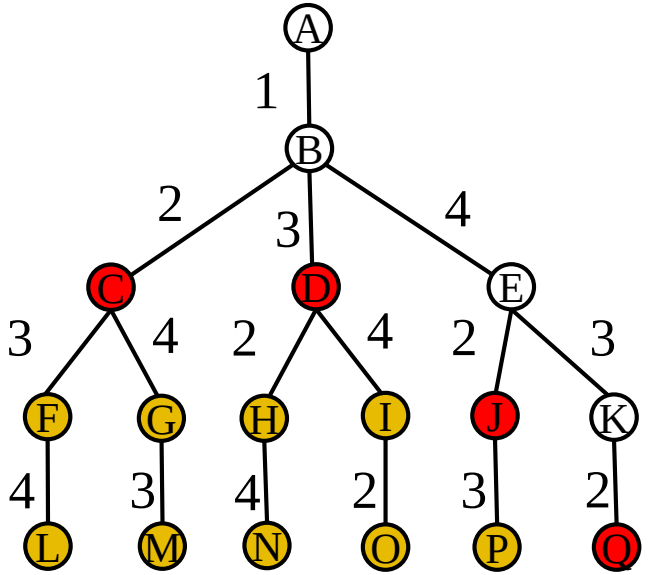
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
K	200	300	500

优先级函数=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

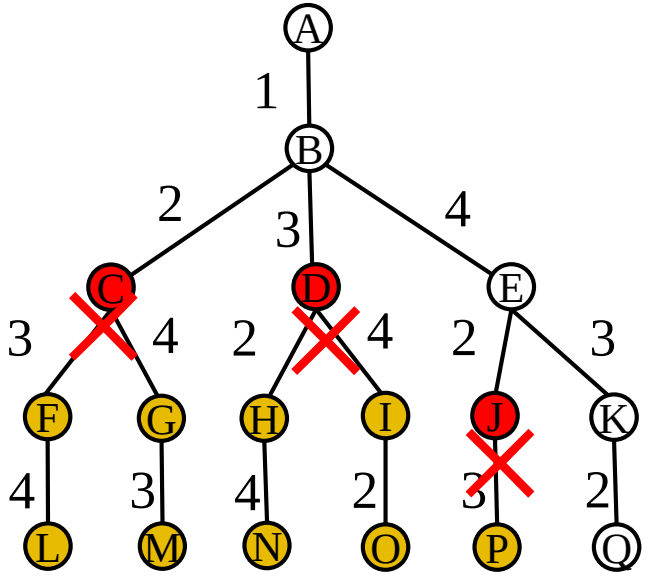
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
Q	400	100	当前 500 最优

优先级函数=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

非对称旅行商问题的解空间树

票价	1	2	3	4
1	0	500	600	100
2	100	0	800	500
3	1000	200	0	2000
4	400	400	100	0



节点颜色

- 红色：优先级队列中的节点
- 黄色：未被访问的节点
- 白色：已经完成访问的节点

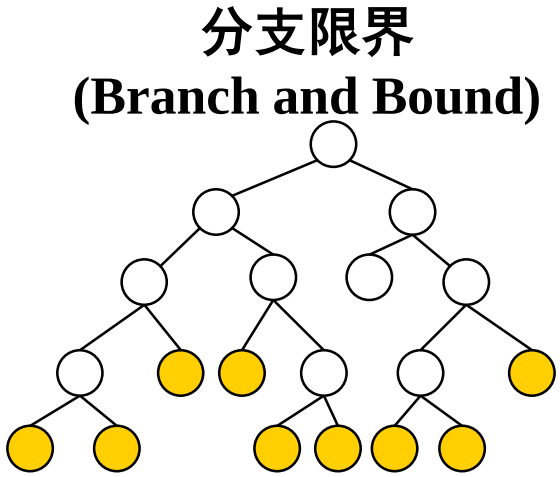
节点	当前值	未来最优值	总代价
C	500	400	900
D	600	400	1000
J	500	300	800
Q	400	100	当前 500 最优

优先级函数=当前解的值
+ 未来的最优解估计值

每一个未选城市最低出发票价之和 +
当前选中城市的最低出发票价

分支限界法的基本思想

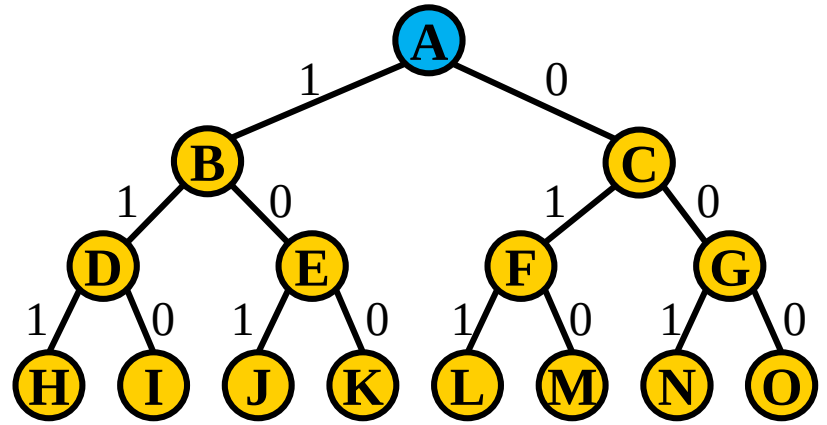
- 一种与回溯法类似的算法
  - 将问题建模为**解空间树**
  - 通常用**代价函数**估算每个分支的最优值
  - **优先选择**当前看来最好的分支
  - 通常用**广度**优先搜索
  - 搜索过程中**剪枝**
- 分支限界的剪枝函数
  - 不满足约束条件
  - 代价函数值不优于当前的界



分支限界法(算法框架-Step1)

```
void Branch-and-Bou
    Q.enqueue(x);
    Initialize bestx
    while(Q is not emp
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

根结点入队列；初始化界的值 *bestx*



正被访问结点　已访问结点

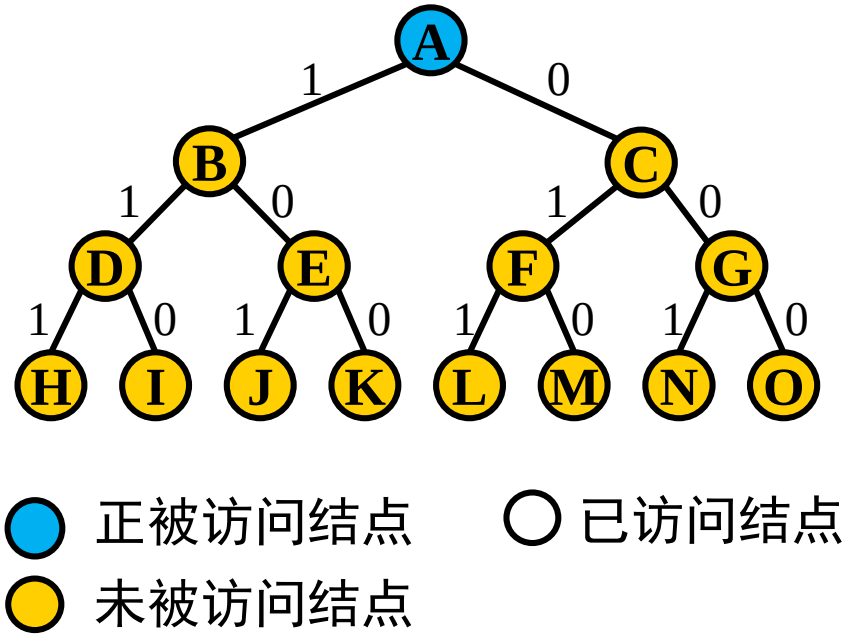
未被访问结点

变量	值
Q	<*A*>
S	

分支限界法(算法框架-Step2)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) &&
            Partition S into S1
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

根据优先级，选出一个代价函数值最优的结点

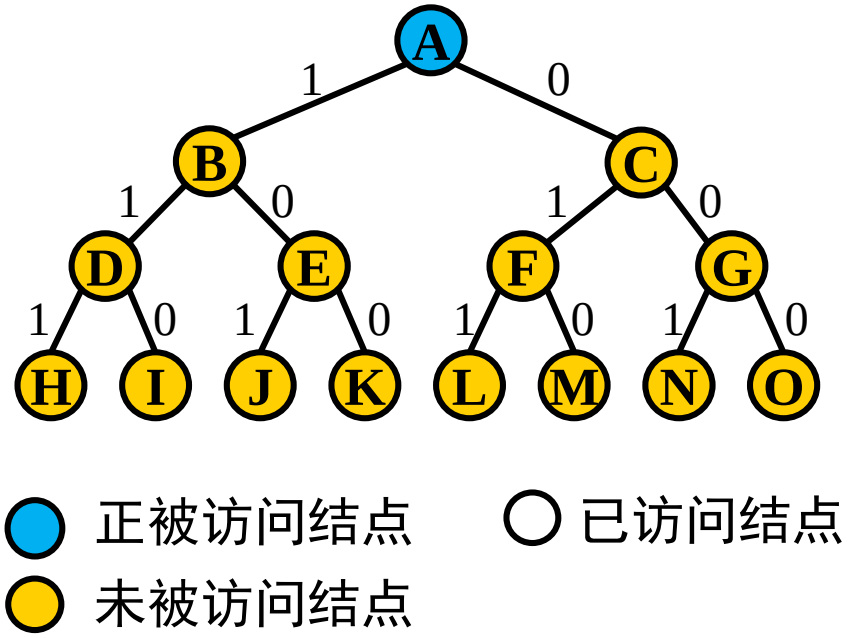


变量	值
Q	<>
S	*A*

分支限界法(算法框架-Step3)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i
                }
        }
    }
}
```

根据可行性约束函数和限界函数判断是不是对*S*剪枝

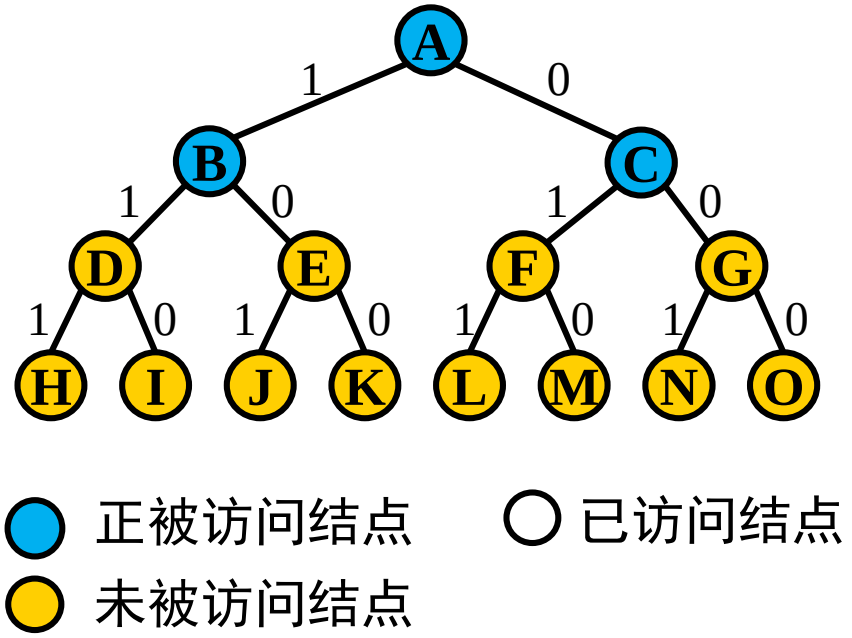


变量	值
Q	<>
S	*A*

分支限界法(算法框架-Step4)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node) update bestx
                else
            }
        }
    }
}
```

生成S的子结点

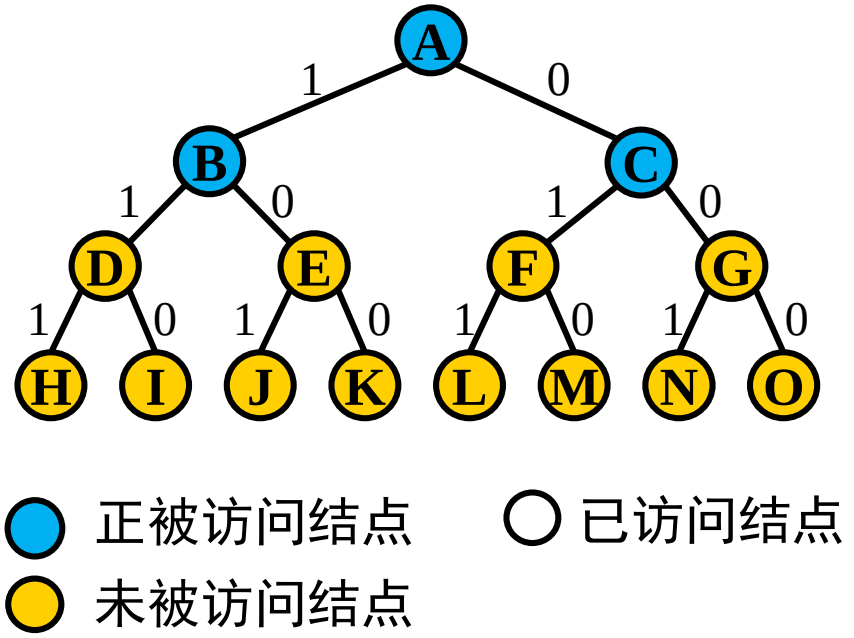


变量	值
Q	<>
S	*A*

分支限界法(算法框架-Step5)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

若子结点为叶子结点（即完整的解向量）则更新当前的界

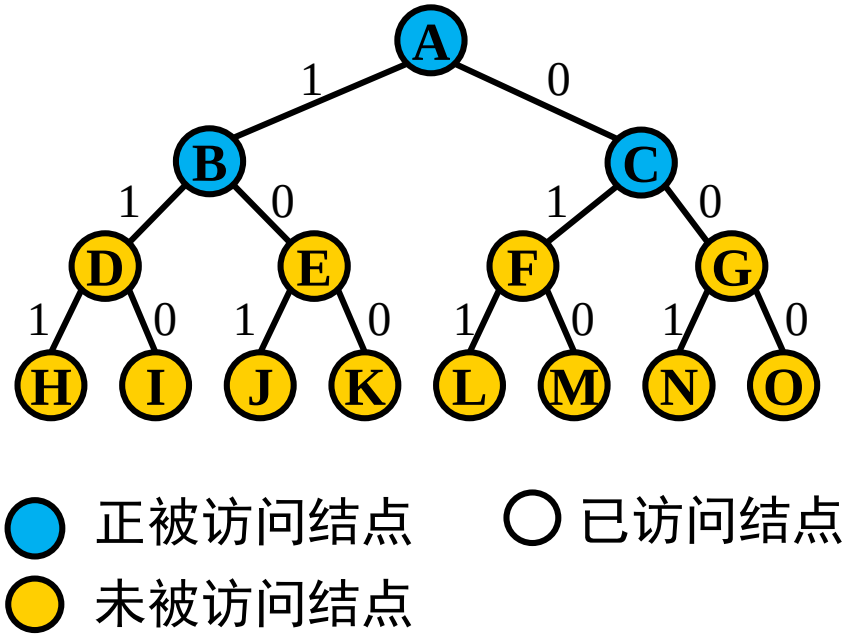


变量	值
Q	<>
S	*A*

分支限界法(算法框架-Step6)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

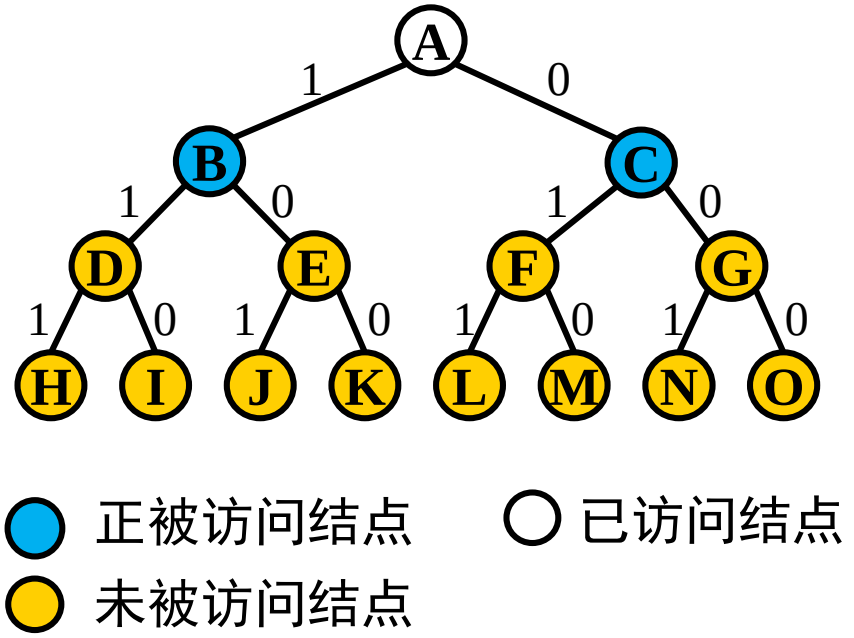
若子结点是非叶子结点（即部分解向量）则将子结点入队列



变量	值
Q	*<C, B>*
S	*A*

分支限界法(算法框架-Step7)

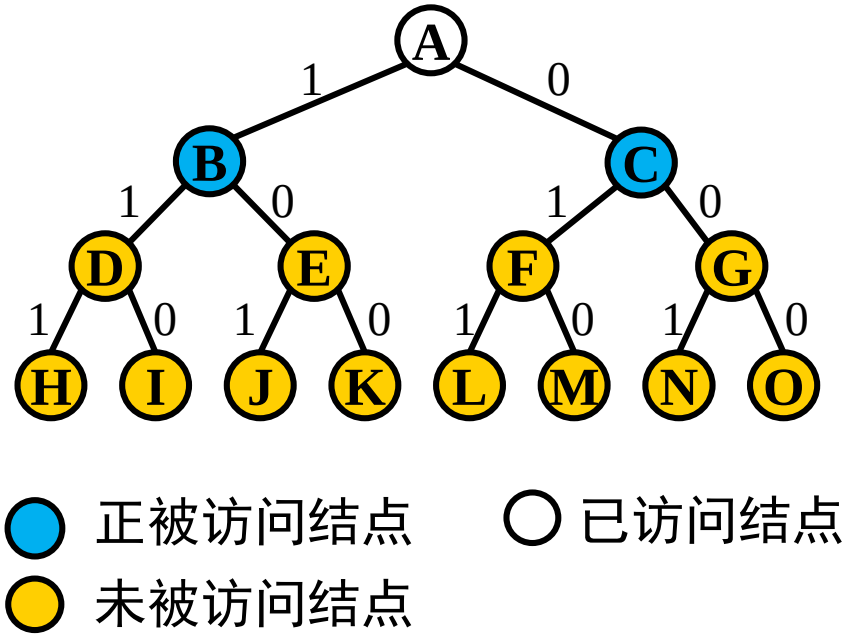
```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```



变量	值
Q	*<B>*
S	*C*

分支限界法(算法框架-Step8)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

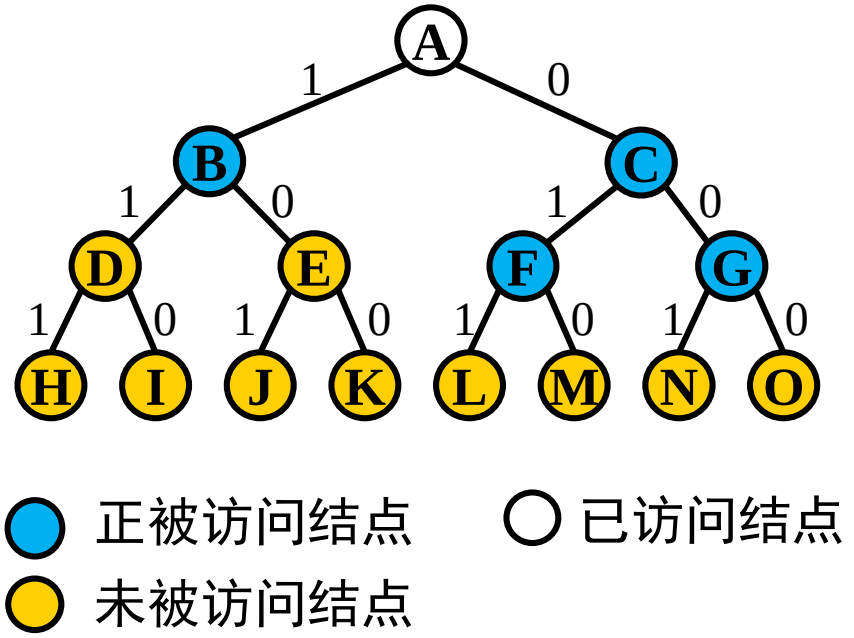


变量	值
Q	*<B>*
S	*C*

分支限界法(算法框架-Step9)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
            }
        }
    }
}
```

生成*C*的子结点*F*和*G*

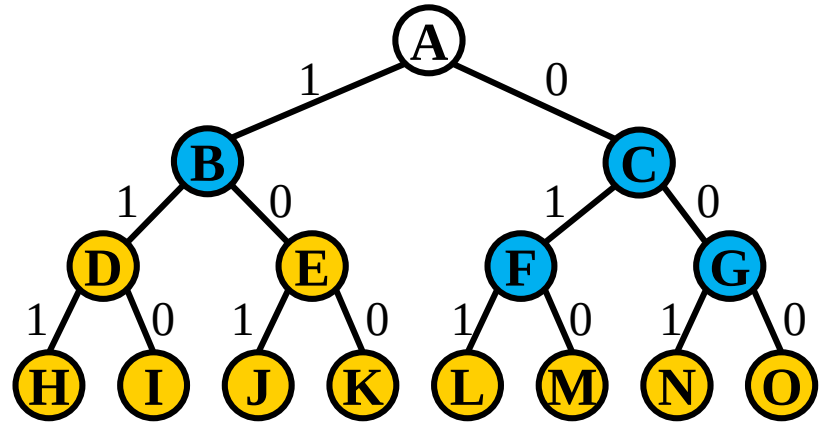


变量	值
Q	*<B>*
S	*C*

分支限界法(算法框架-Step10)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

判断*F*和*G*是否为叶子结点

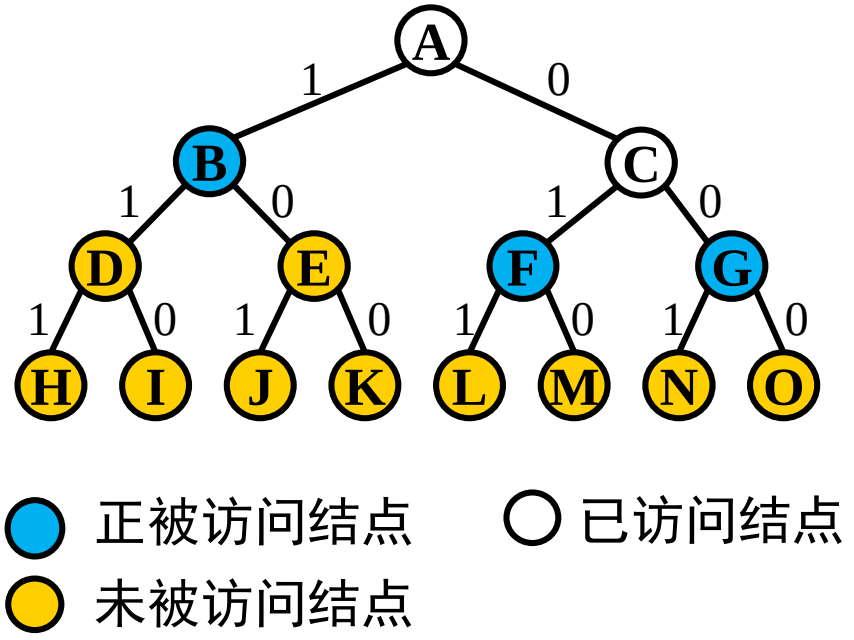


变量	值
Q	*<B>*
S	*C*

分支限界法(算法框架-Step11)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

将*F*和*G*入队列

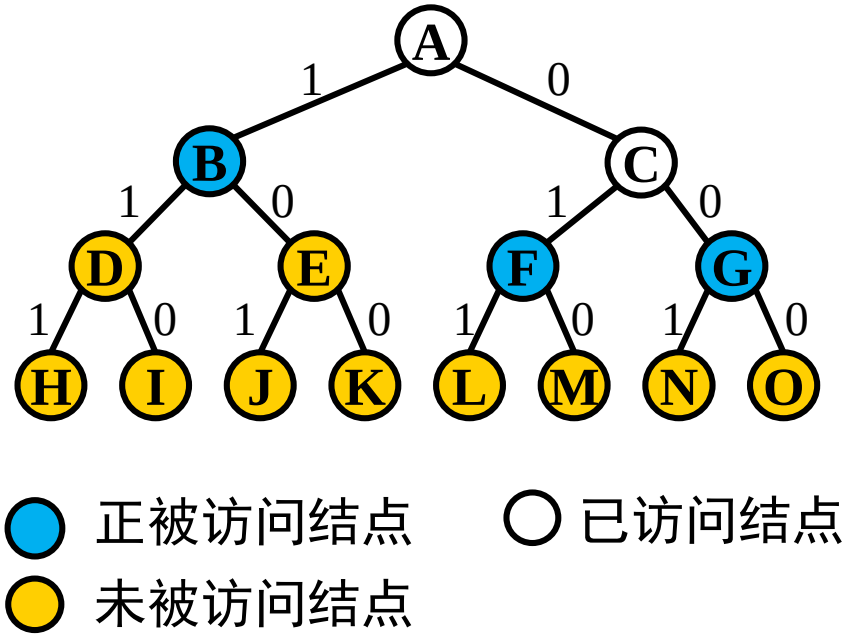


变量	值
Q	<G, *B*, *F*>
S	*C*

分支限界法(算法框架-Step12)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constrai(S) && bound(S, bestx)){
            Partitio into S1, S2, …, Sn
                                        update bestx
            }
        }
    }
}
```

假设当前访优先级最高的结点为*G*

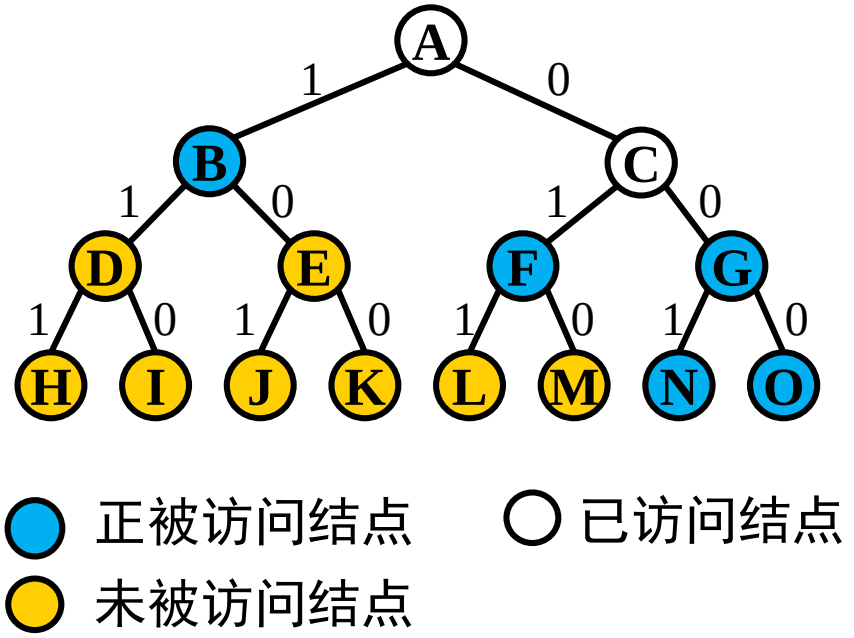


变量	值
Q	*<B, F>*
S	*G*

分支限界法(算法框架-Step13)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S) && bound(S, bestx)){
            Partition S into S1, S2, …, Sn
            for(i=1; i<=n; i++){
                if(Si is a leaf node)  update bestx
                else Q.enqueue(Si);
            }
        }
    }
}
```

子结点*N*和*O*是叶子结点，更新当前的界bestx

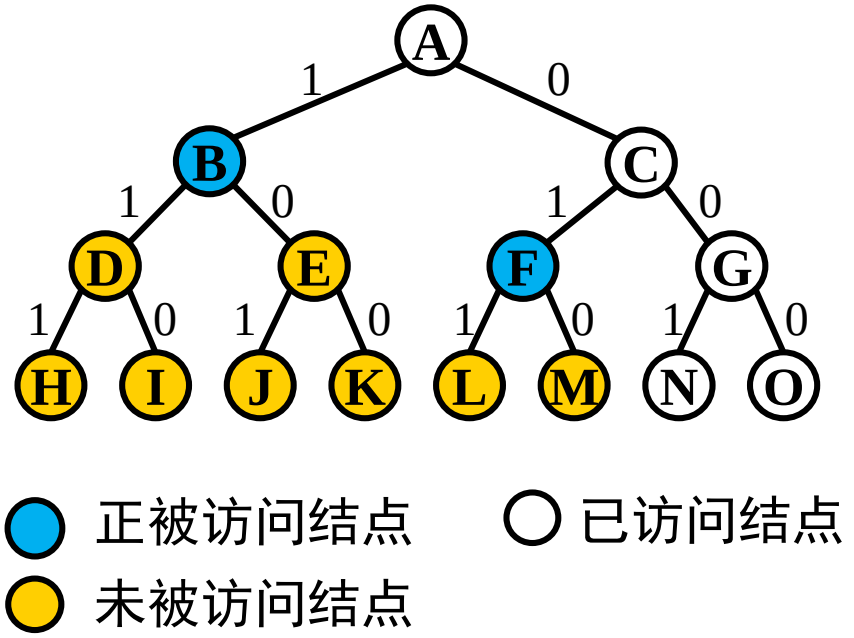


变量	值
Q	*<B, F>*
S	*G*

分支限界法(算法框架-Step14)

```
void Branch-and-Bound(x){
    Q.enqueue(x);
    Initialize bestx
    while(Q is not empty){
        S=Q.dequeue();
        if(constraint(S)      Bound(S, bestx)){
                                                  estx
            else Q.enqueue(Si);
        }
    }
}
```

不断进行出队和入队操作，直到队列Q为空



变量	值
Q	*<B, F>*
S	*G*

分支限界法与回溯法

- 搜索方式不同
  - 回溯法：深度优先
  - 分支限界法：FIFO队列式（广度优先）、优先级队列式
- 搜索目标不同
  - 回溯法：找所有解、可行解、最优解
  - 分支限界法：找最优解
- 搜索用到的函数不同
  - 回溯法：约束函数、限界函数
  - 分支限界：约束函数、限界函数、优先级函数

分支限界法与回溯法

- 遍历所需的空间不同
  - 回溯法：树的高度
  - 分支限界法：队列的长度
- 构造最优解的方式不同
  - 回溯法
    - 采用了深度优先搜索
    - 只需要$O(|X|)$空间，$|X|$为解向量长度
  - 分支限界法
    - 采用了广度优先搜索
    - 每个活的节点都需要保存解，空间开销大
    - 改进：用指针指向父节点，减少保存公共父节点的开销（书P169）

分支限界法的应用

0-1背包问题

- 实例
  - 4种物品，重量w_i和价值v_i分别为
  - $v_1 = 1, v_2 = 3, v_3 = 5,\ v_4 = 10$
  - $w_1 = 2, w_2 = 3, w_3 = 6,\ w_4 = 7$
  - 背包重量限制为10

例如：0-1背包问题

最大化 $x_1 + 3x_2 + 5x_3 + 10x_4$

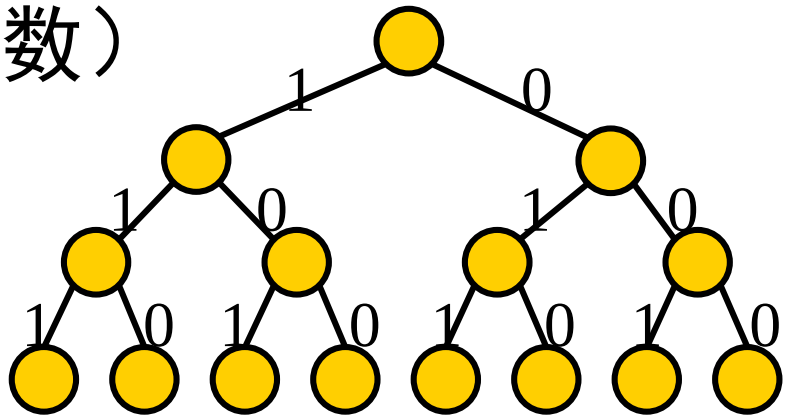
满足约束条件 $\begin{cases} 2x_1 + 3x_2 + 6x_3 + 7x_4 \leq 10 \\ x_i \in \{0,1\},\ \ i = 1,2,3,4 \end{cases}$

0-1背包问题—代价函数（回顾）

- 按v_i/w_i从大到小排序，$i = 1, 2, \cdots, n$
- 假设位于结点$\langle x_1, x_2, \cdots, x_k \rangle$
- 代价函数=已装入价值+Δ
  - Δ：还可继续装入最大价值的上界
  - Δ=背包剩余重量×v_{k+1}/w_{k+1}（可装）
  - Δ=0（不可装）

0-1背包问题—分支限界法

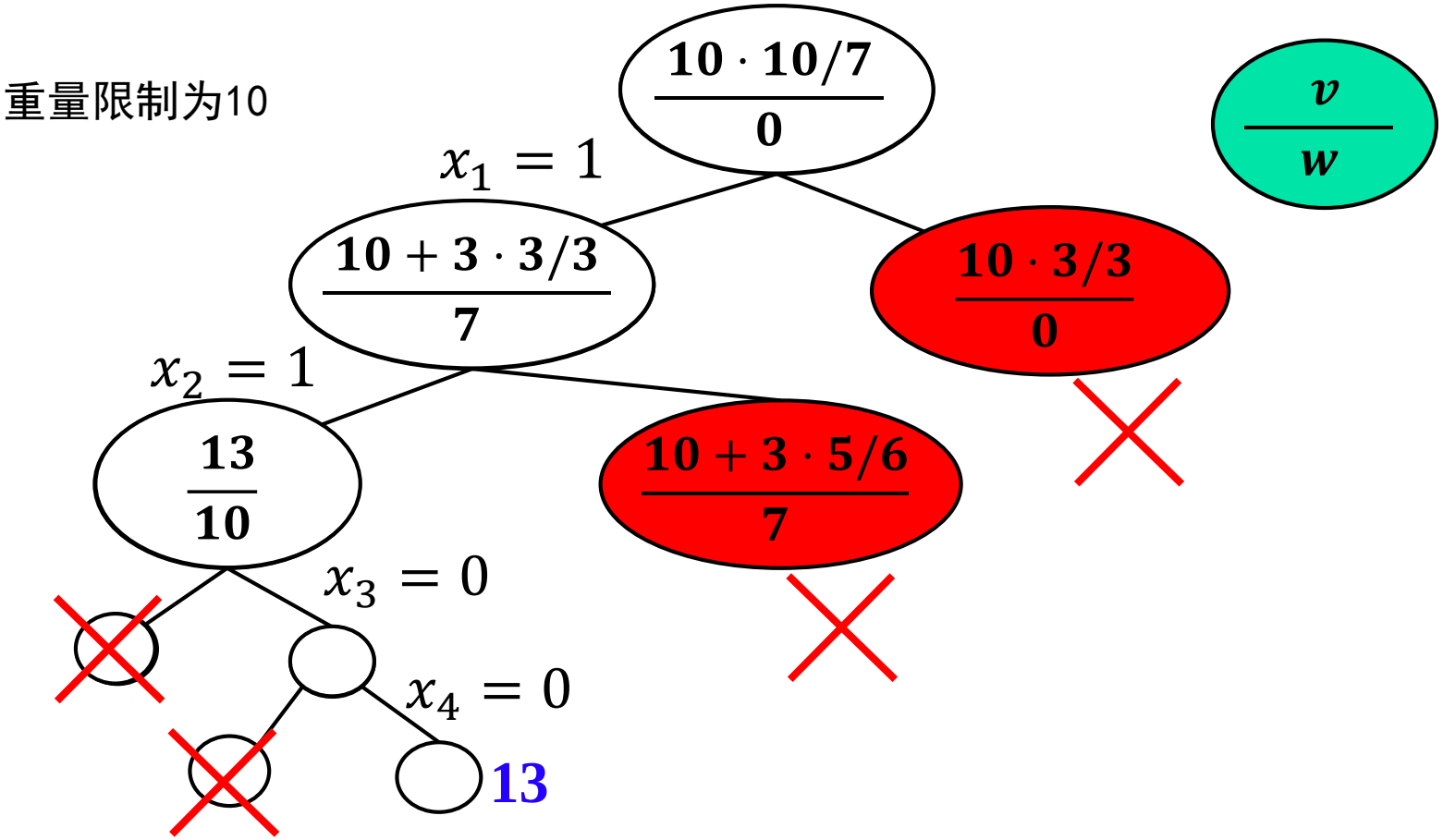
- 基本思想
  - 将物品按v_i/w_i从大到小排序，确定解空间树
  - 从空集∅和仅含空集∅的优先队列开始
  - 选择计算节点队列中代价值最高的节点并扩展
  - 若扩展出节点不被剪枝，将节点插入节点队列
  - 反复2~3步，直到优先队列为空时为止
- 代价函数（优先级函数）
  - 已装入价值+Δ
- 剪枝函数
  - 与回溯法相同



0-1背包问题—分支限界法

最大化　$10x_1 + 3x_2 + 5x_3 + x_4$

满足 $7x_1 + 3x_2 + 6x_3 + 2x_4 \leq 10; x_i \in \{0,1\}, \quad i = 1,2,3,4$

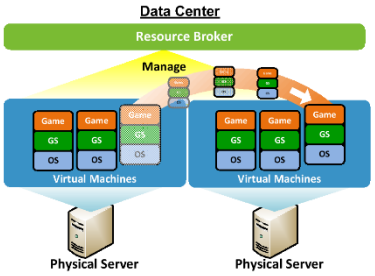


装载问题（回顾）

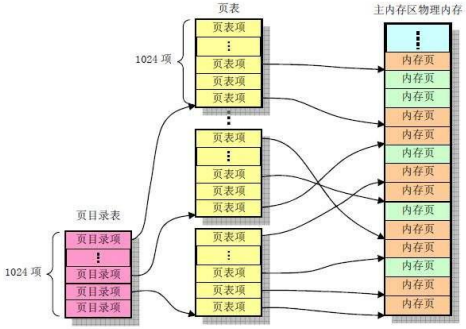
- 有一批共n个集装箱要装上2艘载重量分别为c_1和c_2的轮船，其中集装箱i的重量为w_i，且

$$w_1 + w_2 + \cdots + w_n \le c_1 + c_2$$

- 装载问题要求确定是否有一个合理的装载方案可将这个集装箱装上这2艘轮船。如果有，找出一种装载方案。



云计算虚拟机调度



操作系统内存管理

物料最优剪裁

集装箱最优装载

装载问题的求解思路（回顾）

- 输入：集装箱重量W，轮船载重c_1, c_2
  - 首先将第一艘轮船尽可能装满；
  - 将剩余的集装箱装上第二艘轮船。
  - 将第一艘轮船尽可能装满等价于选取全体集装箱的一个子集，使该子集中集装箱重量之和最接近。

- 实例：
  - $W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$
  - 最优解$\langle 1,0,0,1,1,1,0 \rangle$

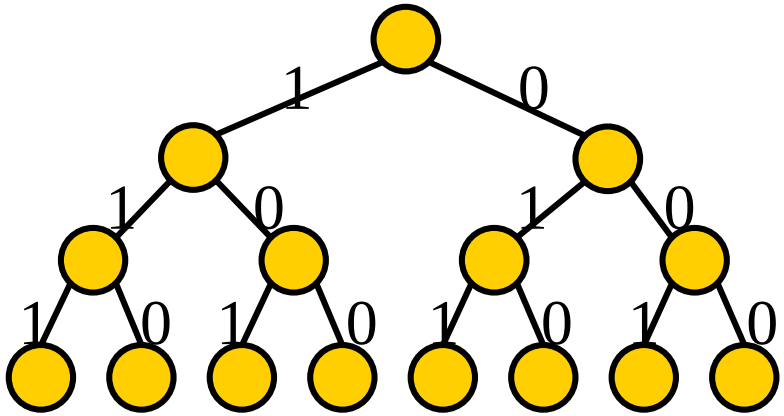
$$\max \sum_{i=1}^{n} w_i x_i$$

$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

$$x_i \in \{0,1\}, 1 \le i \le n$$

装载问题—分支限界法

- 基本思想（与0-1背包问题类似）
  - 从空集∅和仅含空集∅的优先队列开始
  - 选择计算节点队列中代价值最高的节点并扩展
  - 若扩展出节点不被剪枝，将节点插入节点队列
  - 反复2~3步，直到优先队列为空时为止
- 代价函数
  - 当前重量之和+未选中物品的重量之和
- 剪枝函数
  - 与回溯法相同



装载问题—分支限界法

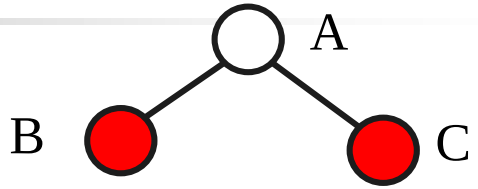
- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$

优先级队列

	A				
代价	267				

当前最优重量：0

装载问题—分支限界法



- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$

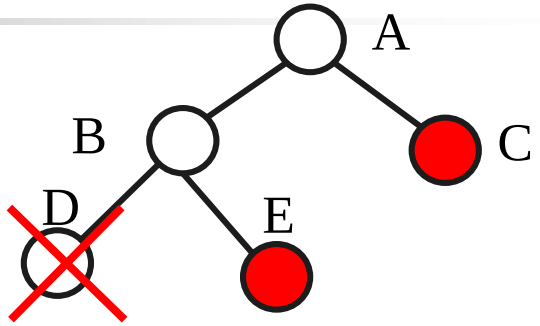
优先级队列

	B	**C**			
代价	**267**	177			

当前最优重量：90

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



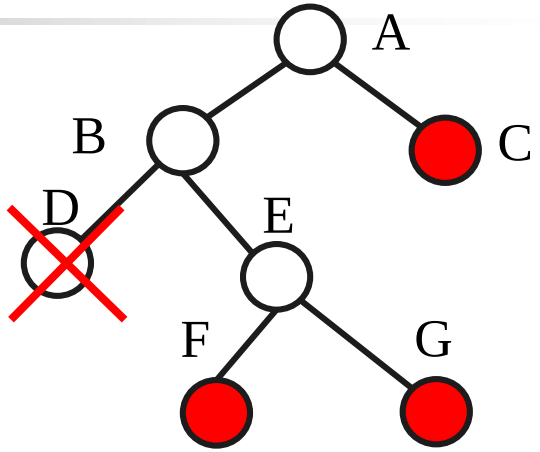
优先级队列

	E	C			
代价	202	177			

当前最优重量：90

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



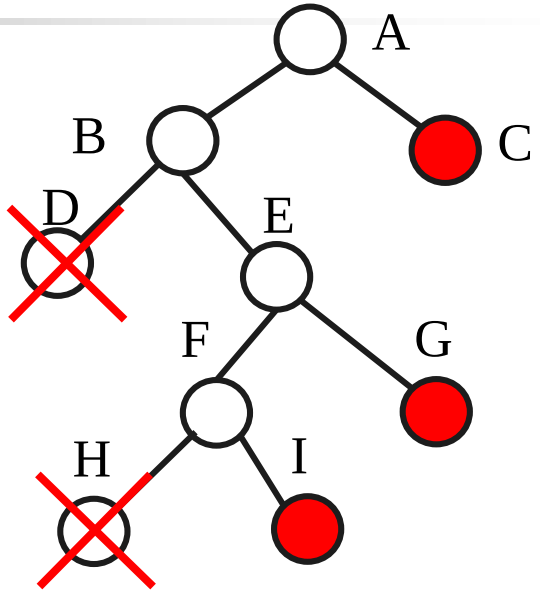
优先级队列

	F	C	G		
代价	202	177	162		

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



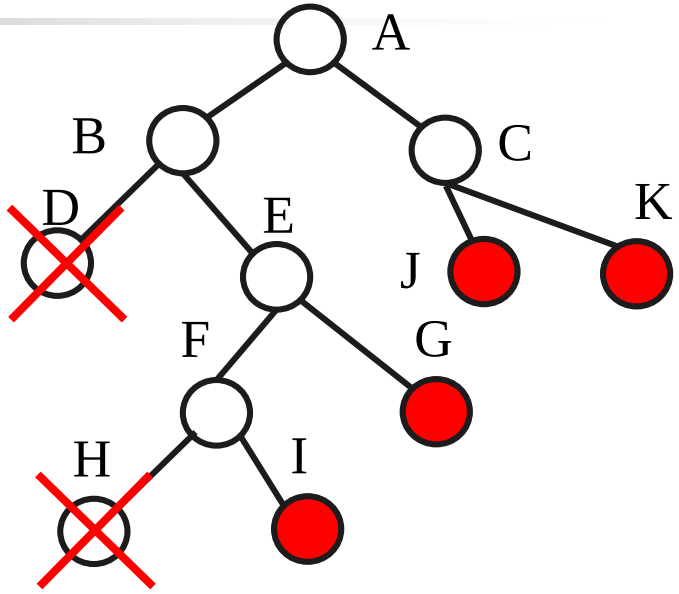
优先级队列

	C	I	G		
代价	177	172	162		

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



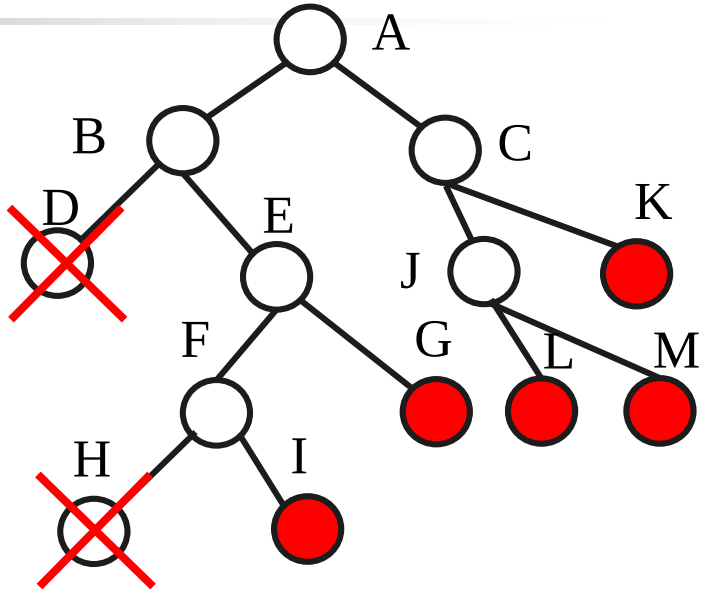
优先级队列

	J	I	G	K	
代价	177	172	162	112	

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



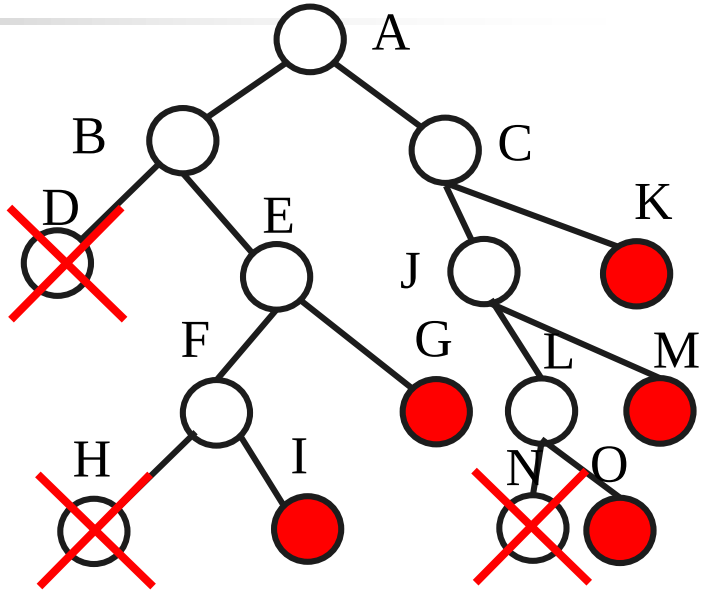
优先级队列

	L	I	G	M	K
代价	177	172	162	137	112

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



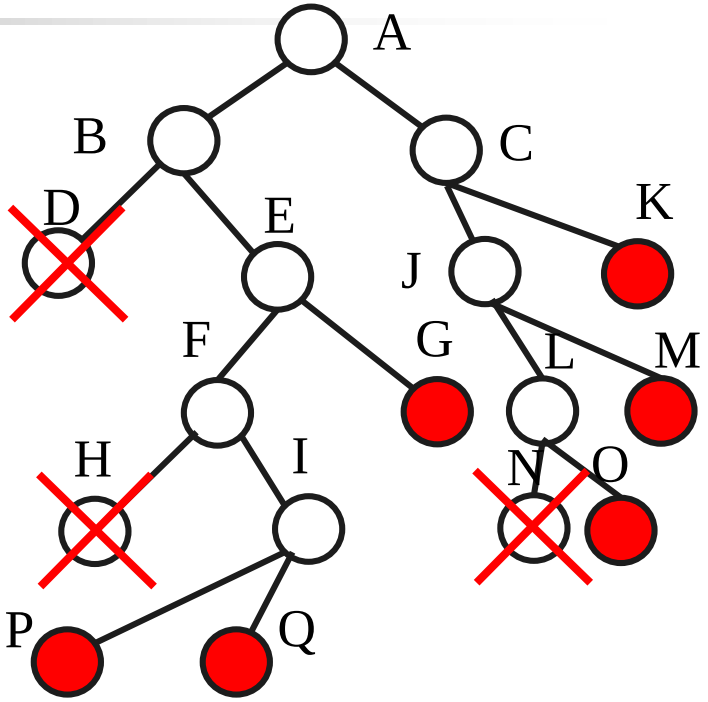
优先级队列

	I	G	O	M	K
代价	172	162	147	137	112

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



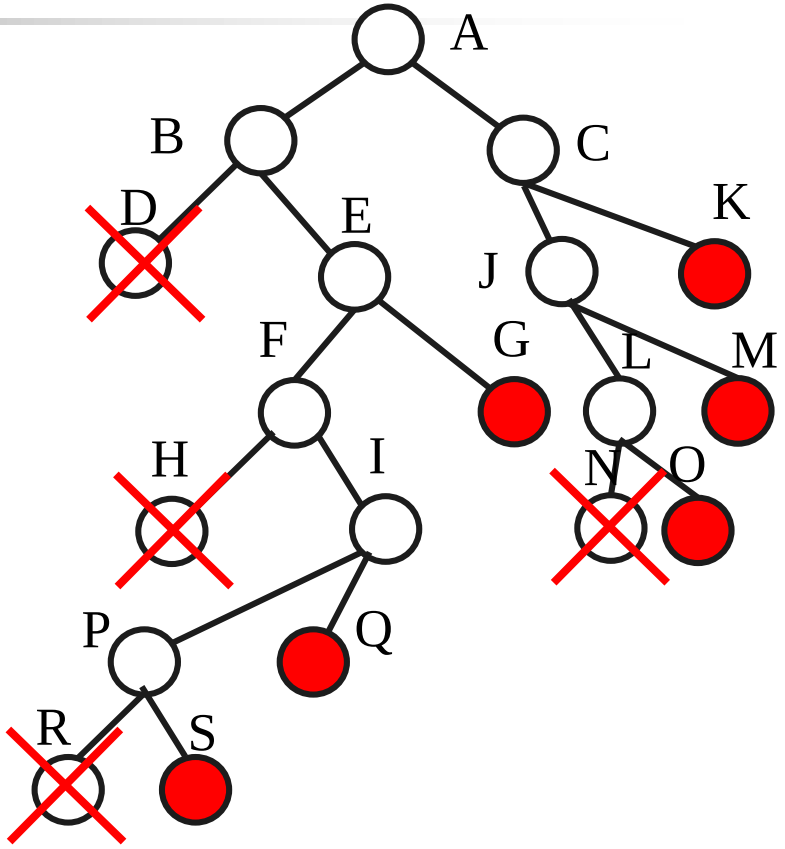
优先级队列

	P	G	O	Q	M	K
代价	**172**	162	157	152	137	112

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



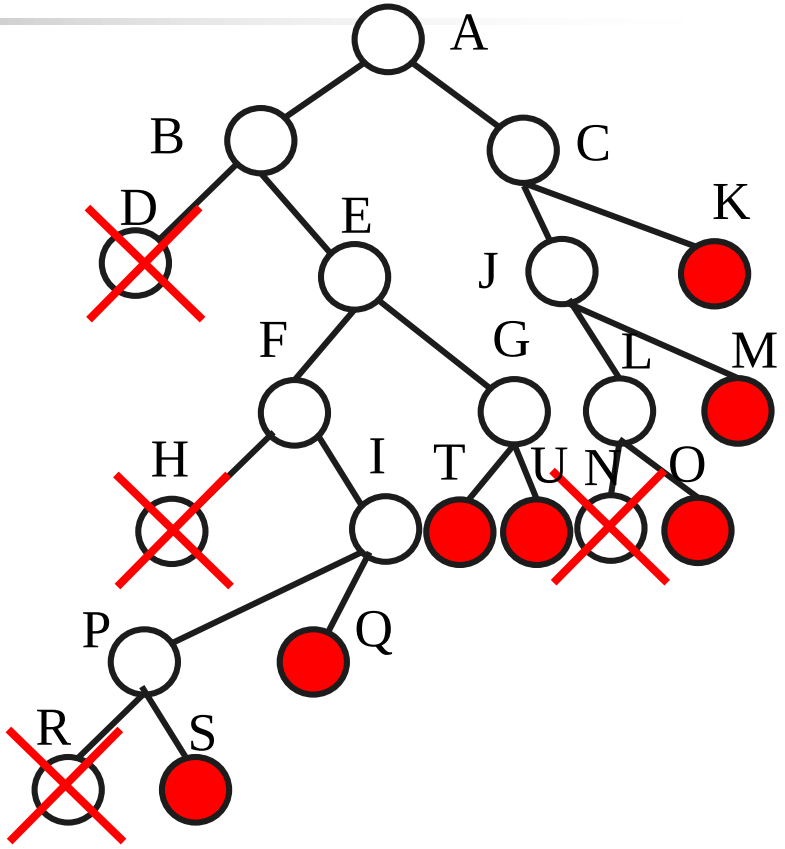
优先级队列

	G	S	O	Q	M	K
代价	162	160	157	152	137	112

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



优先级队列

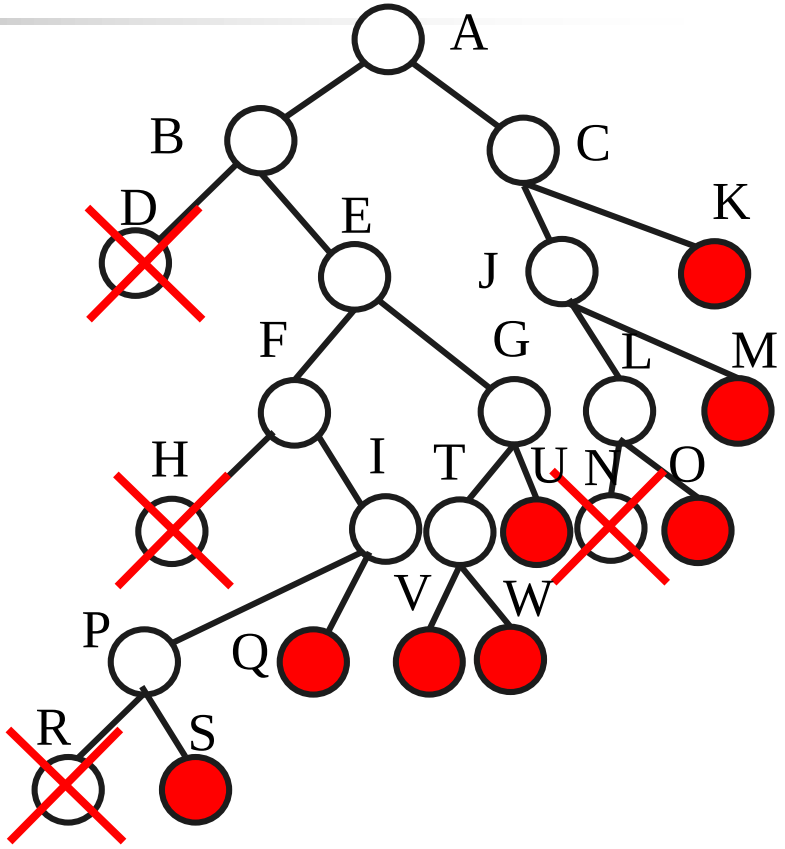
	T	S	O	Q	M	K
代价	162	160	157	152	147	137

	U					
代价	132					

当前最优重量：130

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



优先级队列

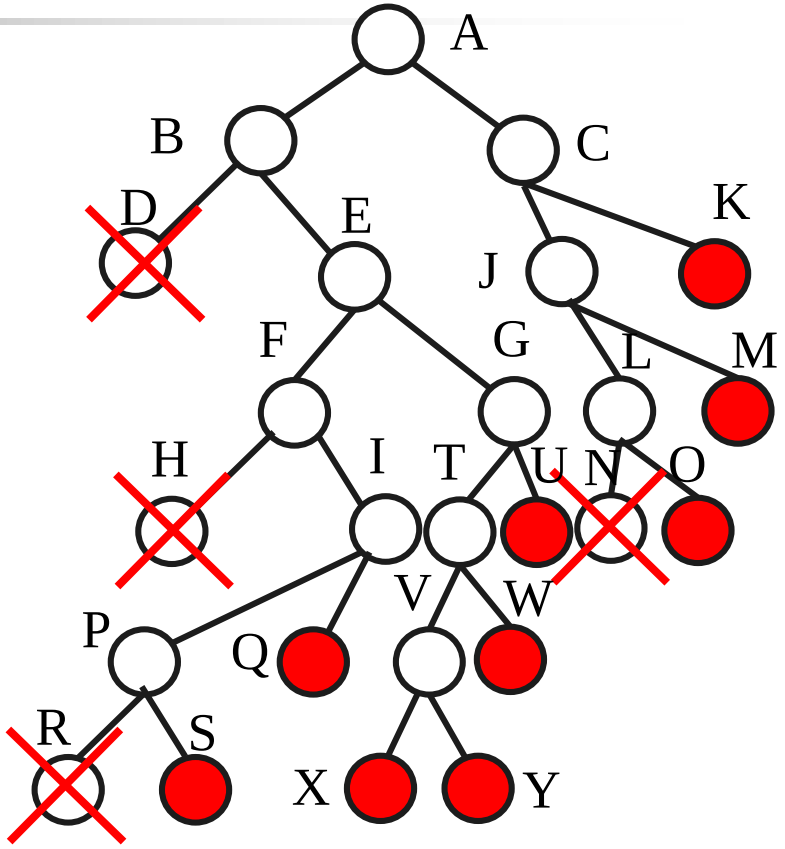
	V	S	O	Q	M	W
代价	162	160	157	152	147	142

	K	U				
代价	137	132				

当前最优重量：**140**

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



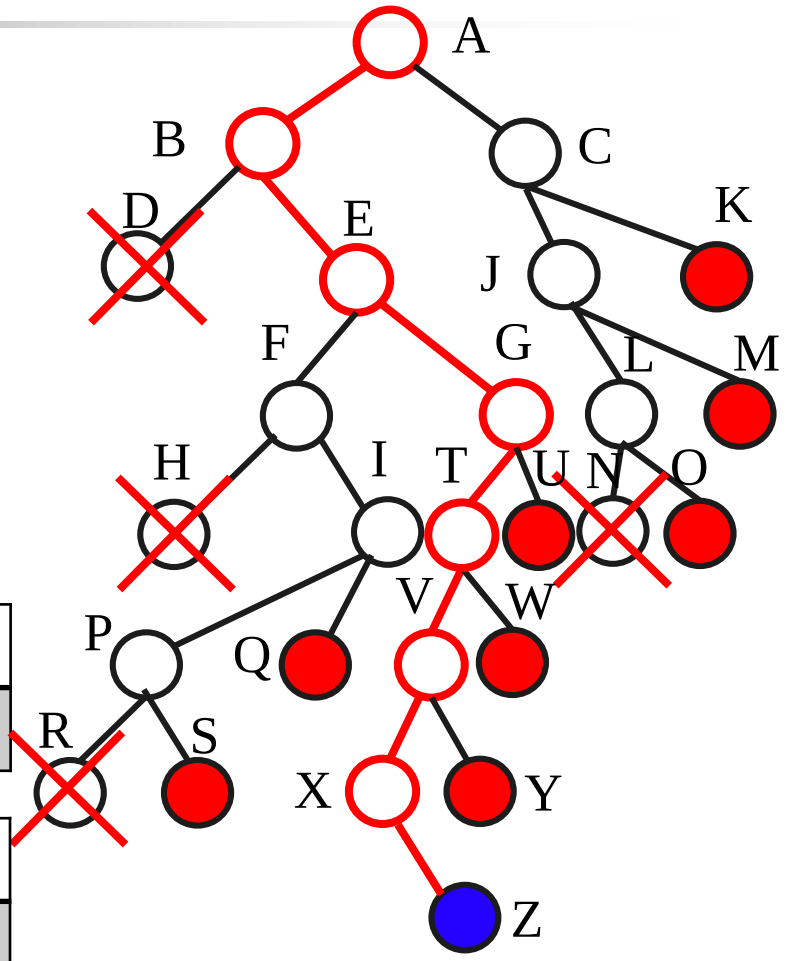
优先级队列

	X	S	O	Q	Y	M
代价	162	160	157	152	147	147
	W	K	U			
代价	142	137	132			

当前最优重量：152

装载问题—分支限界法

- 实例
  - 集装箱重量$W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$



优先级队列

	X	S	O	Q	Y	M
代价	162	160	157	152	147	147

	W	K	U			
代价	142	137	132			

当前最优重量：$152=c_1$ **得到了最优解！**

本章小结

- **分支限界法的剪枝和搜索策略**
  - 可行性约束函数
  - 限界函数
  - 优先级函数（代价函数）
- **分支限界法的算法框架**
  - FIFO队列式分支限界法（广度优先搜索）
  - 优先队列式分支限界法（优先级由代价函数确定）
- **分支限界法的应用**
  - 非对称旅行商问题
  - 0-1背包问题
  - 装载问题