算法分析与设计

Analysis and Design of Algorithm

第13次课

课程回顾

- 适用对象：求解搜索问题和优化问题
- 搜索空间：树，结点对应部分解向量，可行解在树叶上
- 搜索过程：采用系统的方法隐含遍历搜索树
- 搜索策略：深度/宽度优先、函数优先、宽深结合
- 剪枝方法：约束函数、限界函数
- 结点分支判定条件：
  - 不满足剪枝条件—分支扩张解向量
  - 满足剪枝条件—回溯到该结点的父结点
- 存储：当前路径

课程回顾

问题	解性质	解描述向量	搜索空间	搜索方式	约束条件
n皇后	可行解	$\langle x_1, x_2, \cdots, x_n \rangle$ x_i：第i行列号	n叉树	深度优先搜索	彼此不攻击
0-1背包	最优解	$\langle x_1, x_2, \cdots, x_n \rangle$ $x_i \in \{0,1\}$	子集树	深度优先搜索	不超过总重量
旅行商	最优解	$\langle k_1, k_2, \cdots, k_n \rangle$ $1,2,\cdots,n$的排列	排列树	深度优先搜索	选没有经过的城市

特点	搜索解	向量，不断扩张部分向量	树	跳跃式遍历	约束条件，回溯判定

回溯法的几个应用

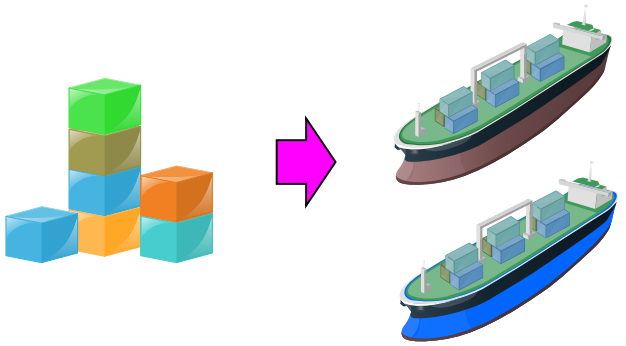
装载问题

装载问题及其应用

- 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。



装载问题的求解思路

- 输入：物品重量W，旅行箱载重c_1, c_2
  - 首先将第一个旅行箱尽可能装满；
  - 将剩余的物品装上第二个旅行箱。
  - 将第一个旅行箱尽可能装满等价于选取全体物品的一个子集，使该子集中物品重量之和最接近。

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

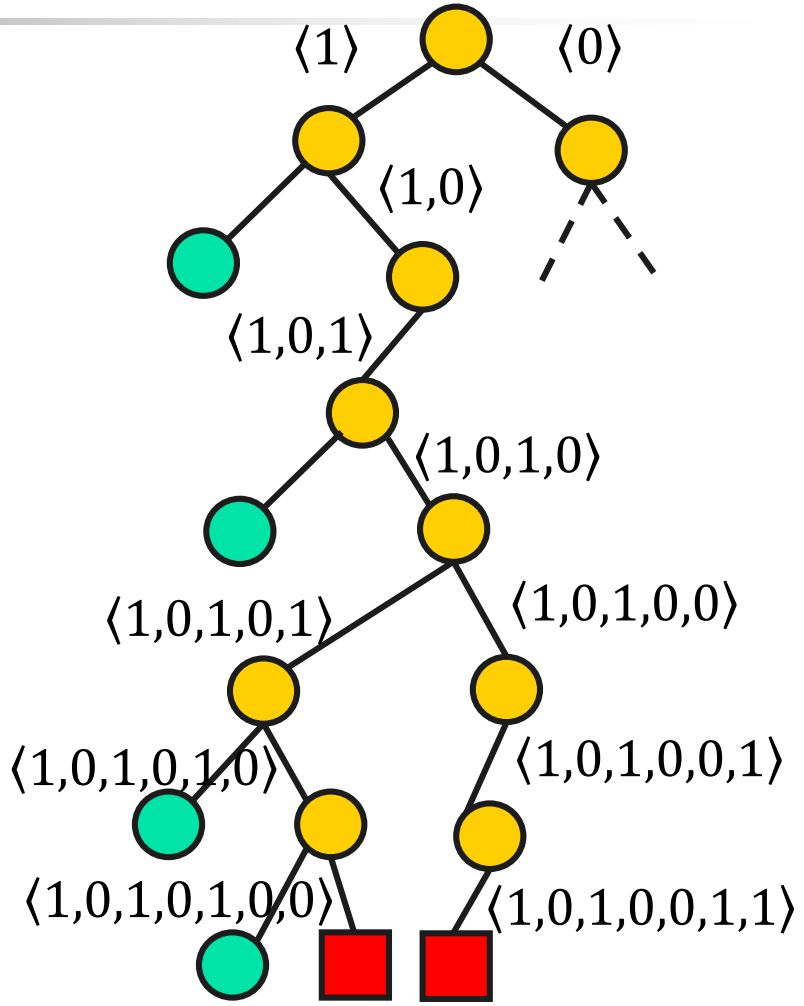
$$\text{s.t.} \sum_{i=1}^{n} w_i x_i \le c_1$$

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

装载问题的解空间树

- 一棵二叉树（子集树）

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

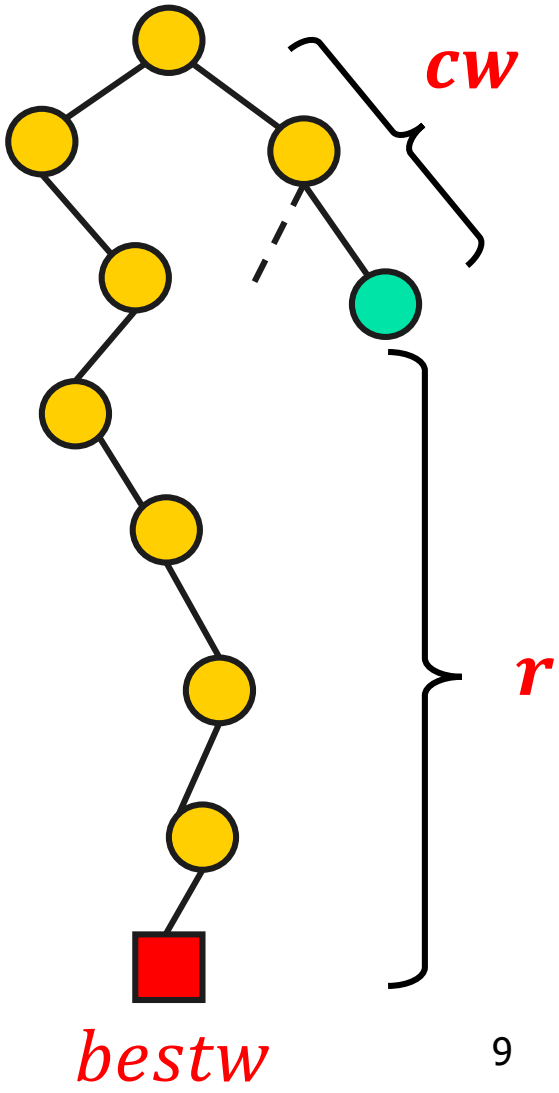


装载问题的剪枝函数

- 可行性约束函数
- 限界函数
  - 有用的变量
    - 当前旅行箱内重量：cw
    - 当前最优解：$bestw$
  - 上界函数：剩余物品的重量

$$r = w_{i+1} + w_{i+2} + \cdots + w_n$$

  - 剪枝条件：
  - 若$cw + r \leq bestw$，则剪枝



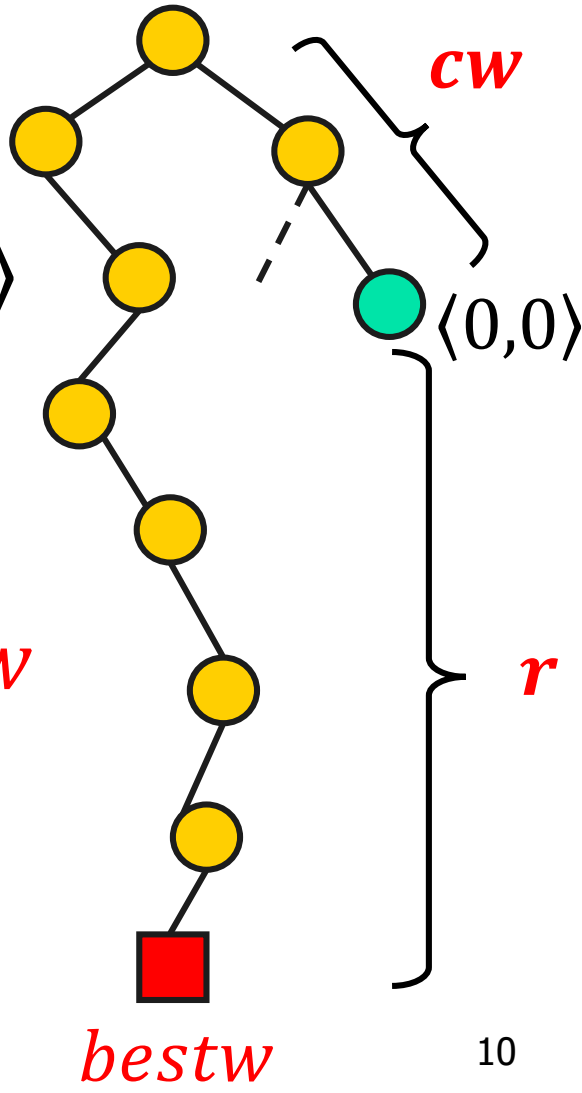
装载问题的剪枝函数

- 实例
  - 物品重量

    $W = \langle 90,65,40,30,20,12,10 \rangle$
  - $c_1 = 152,\ c_2 = 130$
  - 最优解：$\langle 1,0,1,0,0,1,1 \rangle$
  - $bestw=152$
  - $cw=0$
  - $r=40+30+20+12+10=112$

$cw + r \leq bestw$

剪枝！



装载问题的算法实现

```
void backtrack (int i)
{// 搜索第i层结点
     if (i > n)
          更新最优解bestx,bestw;return;
     r -= w[i];
     if (cw + w[i] <= c) {
          x[i] = 1;
          cw += w[i];
          backtrack(i + 1);
          cw -= w[i];          }
     if (cw + r > bestw)   {
          x[i] = 0;
          backtrack(i + 1);          }
     r += w[i];
}
```

到达叶结点

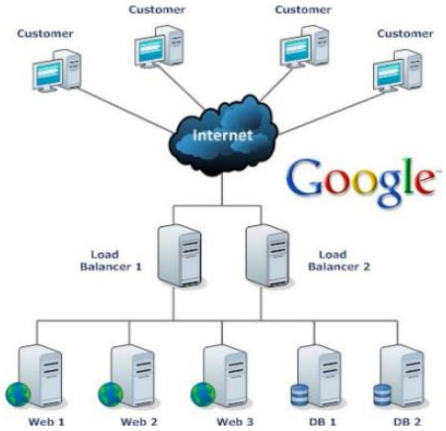
搜索左子树

搜索右子树

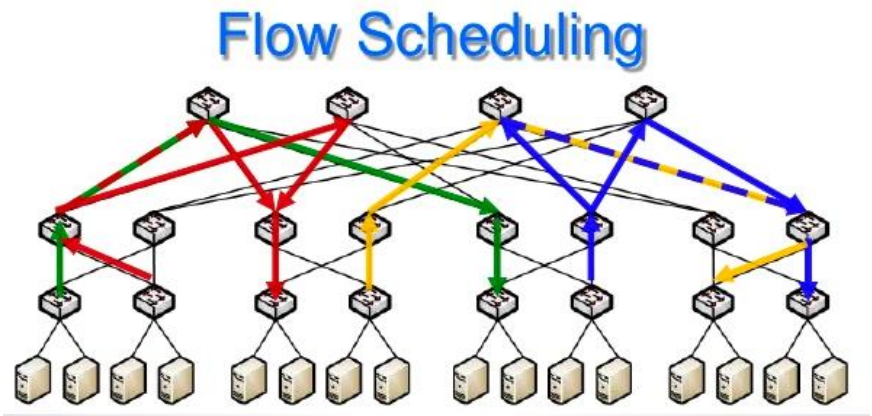
批处理作业调度问题

批处理作业调度及其应用

- 给定n个作业的集合$\{J_1,J_2,\ldots,J_n\}$。每个作业必须先由机器1处理，然后由机器2处理。
  - t_{ji}是作业J_i需要机器j的处理时间。
  - F_i是作业i的完成时间。
- 目标：求最佳作业调度方案使作业完成时间和最小



Web服务器调度



网络交换机的流调度

批处理作业调度问题的解空间

- 实例：

梅园打印店每天要处理很多打印任务，分两步：

- 将文件拷贝至电脑
- 在打印机打印

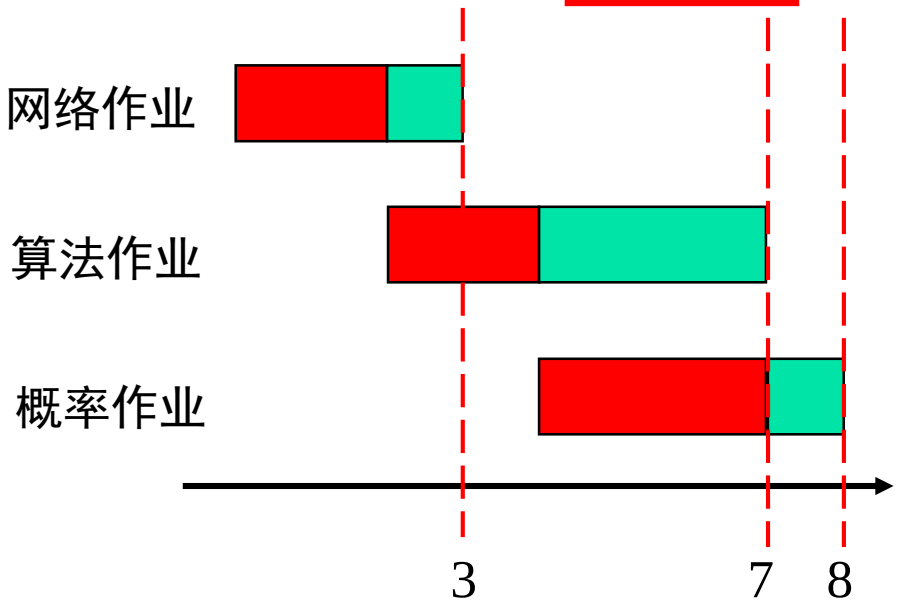
问题：给定一系列任务，且任务的拷贝和打印时间已知，请找出这些任务先后顺序，使所有任务的总完成时间最短。

t_{ji}	拷贝	打印
网络作业	2	1
概率作业	3	1
算法作业	2	3

批处理作业调度问题的解空间

- 问题的解

可行解	⟨1,2,3⟩	⟨1,3,2⟩	⟨2,1,3⟩	⟨2,3,1⟩	⟨3,1,2⟩	⟨3,2,1⟩
完成时间	19	18	20	21	19	19

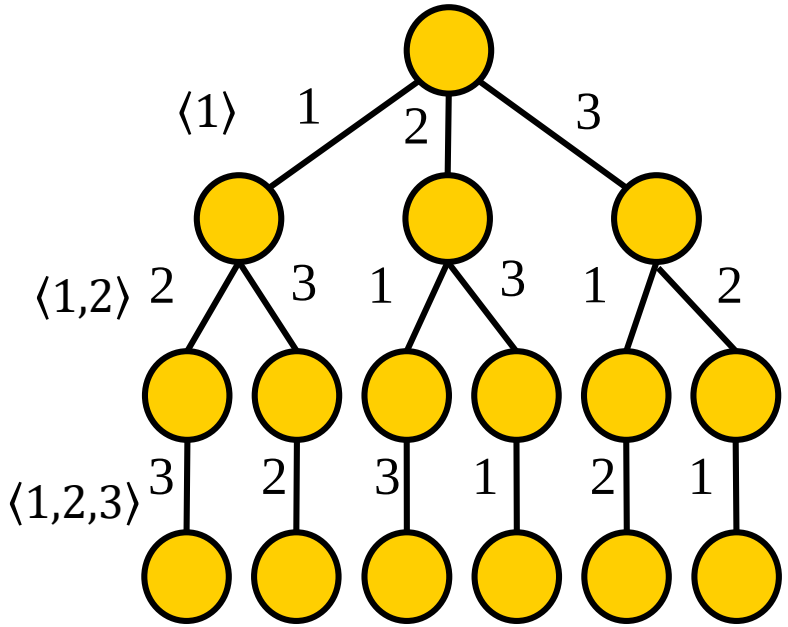


	任务	拷贝	打印
1	网络作业	2	1
2	概率作业	3	1
3	算法作业	2	3

批处理作业调度问题的解空间树

- 实例

	任务	拷贝	打印
1	网络作业	2	1
2	概率作业	3	1
3	算法作业	2	3



- 解空间
  - 排列树（n=3）
- 剪枝函数
  - 当前方案的执行时间>最优解

批处理作业调度问题的算法实现

```
void Backtrack(int i)
{
    if (i > n) {
        for (int j = 1; j <= n; j++)
            bestx[j] = x[j];
        bestf = f;}
    else
        for (int j = i; j <= n; j++) {
            f1+=M[x[j]][1];
            f2[i]=((f2[i-1]>f1)?f2[i-1]:f1)+M[x[j]][2];
            f+=f2[i];
            if (f < bestf) {
                Swap(x[i], x[j]);
                Backtrack(i+1);
                Swap(x[i], x[j]); }
            f1- =M[x[j]][1];
            f- =f2[i];
        }
}
```

到达叶子结点

当前方案的执行时间

若不被剪枝

```
M,      // 各作业所需的处理时间
x,      // 当前作业调度
bestx,  // 当前最优作业调度
f2,     // 机器2完成处理时间
f1,     // 机器1完成处理时间
f,      // 完成时间和
bestf,  // 当前最优值
N       // 作业数
```

两个核心问题小结

- 定义解空间
  - 解向量为$\langle x_1, x_2, \ldots, x_n \rangle$
  - 确定x_i的取值集合为X_i
  - 子集树、排列树、n叉树

- 定义剪枝函数
  - 可行性约束函数
  - 限界函数

回溯法的剪枝技巧

两种剪枝函数

1. **可行性约束函数**

2. **限界函数**

两种剪枝函数

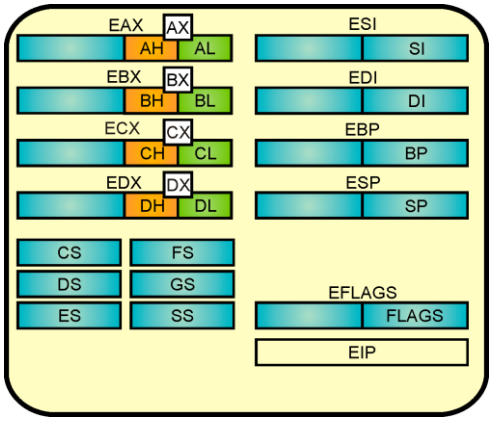
1. **可行性约束函数**

2. **限界函数**

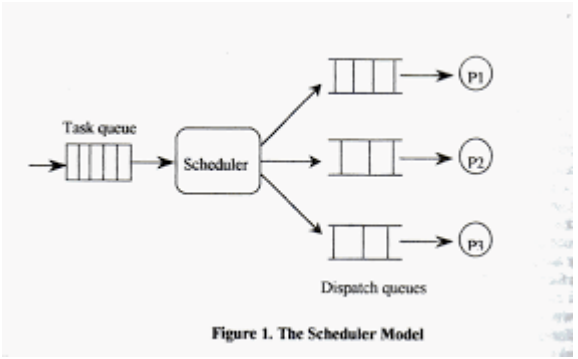
图着色问题及其应用（回顾）

- 给定无向连通图$G=(V,E)$，是否可k种颜色对G中顶点着色，可使任意两个顶点着色不同。
  - 是与否的判定问题
  - 解向量：$\langle x_1, x_2, \cdots, x_n \rangle$, x_i表示顶点i所着颜色

地图着色

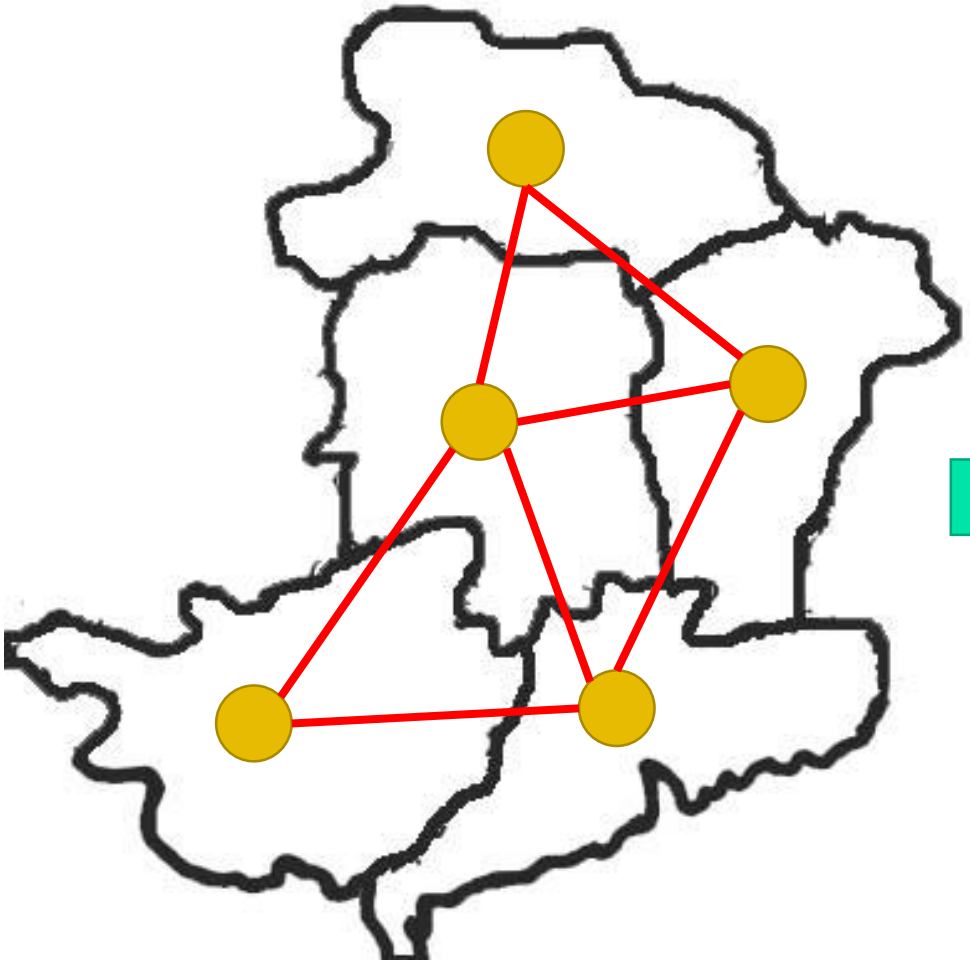


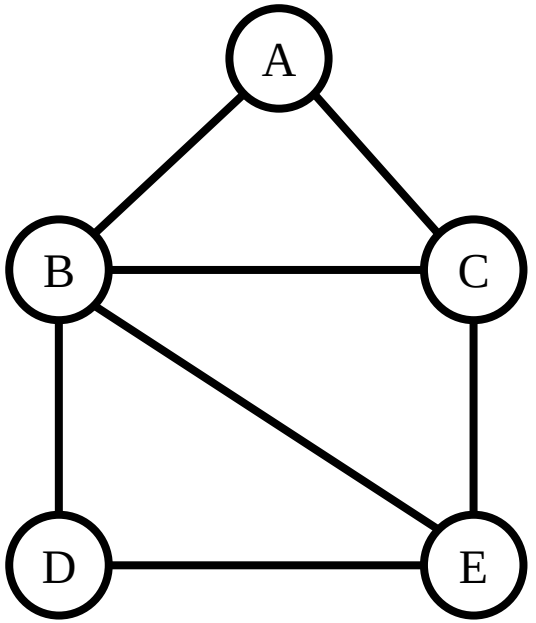
程序编译器的
寄存器分配算法



任务调度

实例：给中国地图着色



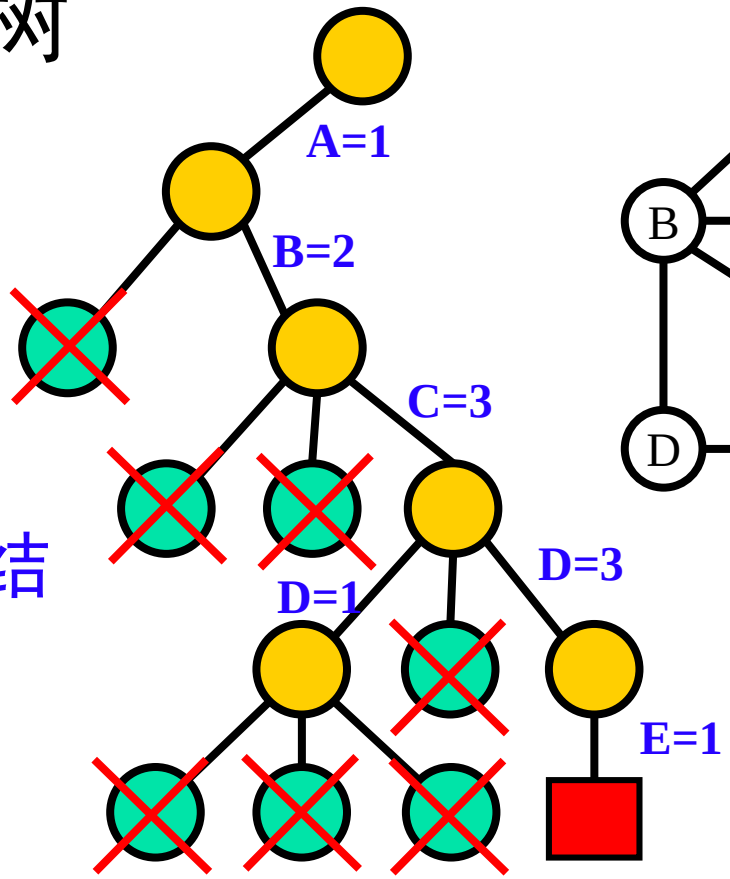


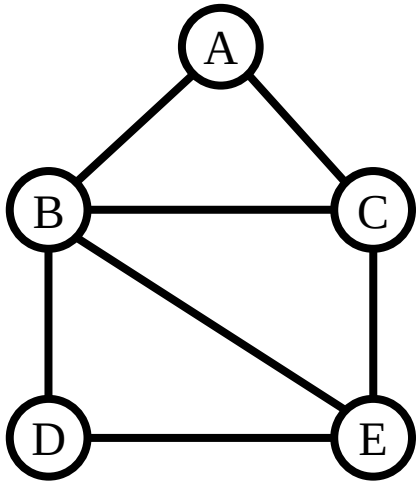
是否可以用三种不同颜色给这个地图着色？

图的m着色问题的解空间树

- 3着色问题—三叉树
- 剪枝函数
  - 可行性约束函数

解空间有$3^5=243$个结点，而回溯只搜了其中14个结点就找到了解。





两种剪枝函数

1. **可行性约束函数**

2. **限界函数**

回溯法与组合优化问题（回顾）

例如：0-1背包问题

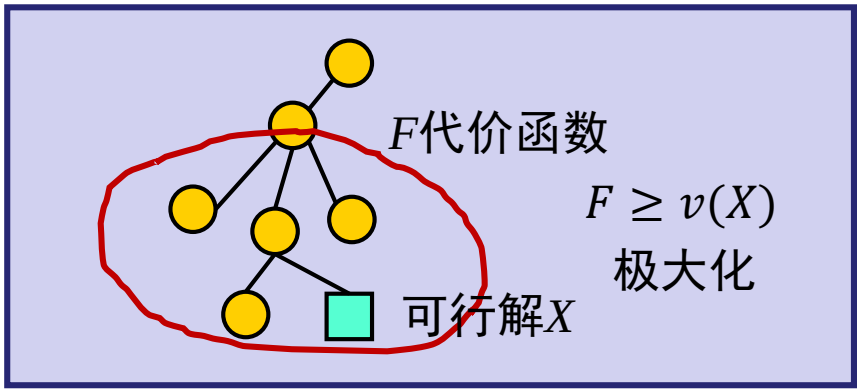
最大化 $x_1 + 3x_2 + 5x_3 + 10x_4$ 最大化价值

满足约束条件 $\begin{cases} 2x_1 + 3x_2 + 6x_3 + 7x_4 \leq 10 & \text{重量约束} \\ x_i \in \{0,1\}, \quad i = 1,2,3,4 & \text{定义域约束} \end{cases}$

- 组合优化问题
  - 目标函数（极大化或极小化）
  - 约束条件（解满足的条件）
  - 可行解：搜索空间满足约束条件的解
  - 最优解：使目标函数达极大（或极小）的可行解

回溯法的代价函数

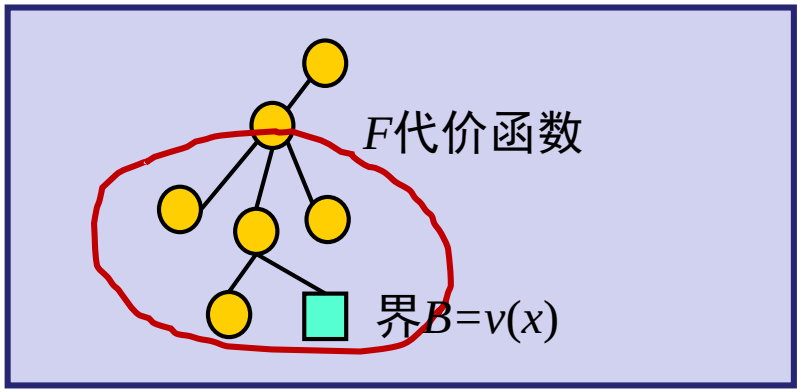
- 代价函数的计算位置
  - 搜索树的结点



- 代价函数的值
  - 对于极大化问题，以该点为根的子树所有可行解的值的上界（极小化问题为下界）
- 代价函数的性质
  - 对于极大化问题，父结点代价不小于子结点的代价（极小化问题相反）

回溯法的界

- 界的含义
  - 当前得到可行解的目标函数的最大值（极小化问题相反）
- 界的初值
  - 极大化问题初值为0（极小化问题为最大值）
- 界的更新
  - 得到更好的可行解时



回溯法的剪枝函数

- 剪枝函数
  - 不满足约束条件（可行性约束函数）
  - 代价函数值不优于当前的界（限界函数）
- 界的更新
  - 对于极大化问题，如果一个新的可行解的优化函数值大于（极小化问题为小于）当前的界，则把界更新为该可行解的值

回溯法剪枝的实例

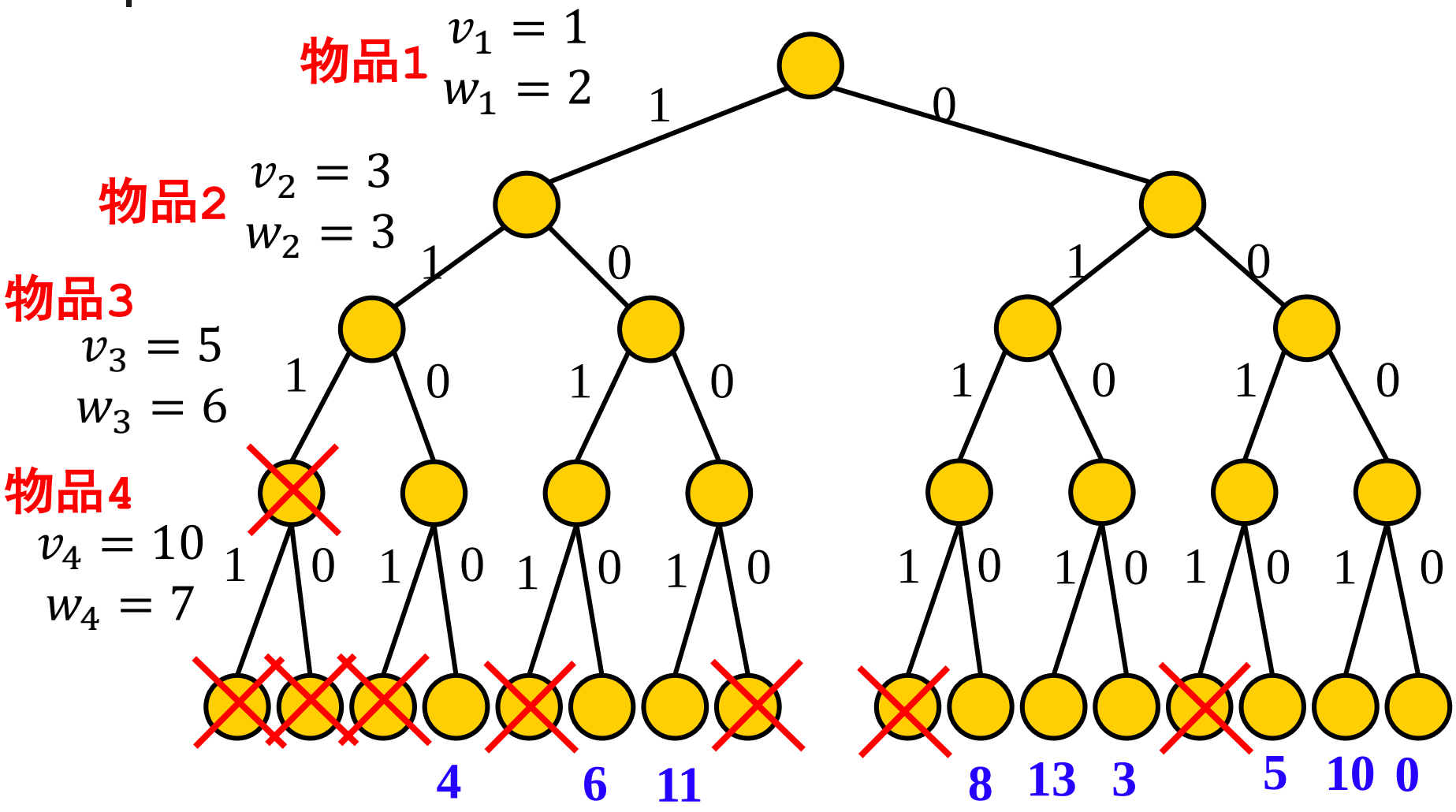
- 0-1背包问题
  - 4种物品，重量w_i和价值v_i分别为
  - $v_1 = 1, v_2 = 3, v_3 = 5,\ v_4 = 10$
  - $w_1 = 2, w_2 = 3, w_3 = 6,\ w_4 = 7$
  - 背包重量限制为10

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

通常的回溯法做法



代价函数的设定

- 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）

实例：0-1背包问题

最大化 $x_1 + 3x_2 + 5x_3 + 10x_4$

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

- 对元素重新排序使得$\frac{v_i}{w_i} \geq \frac{v_{i+1}}{w_{i+1}}$
- 排序后

最大化 $10x_1 + 3x_2 + 5x_3 + x_4$

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

实例：代价函数

- 结点$\langle x_1, x_2, \cdots, x_k \rangle$的代价函数
  - 代价函数=已装入价值+Δ
  - Δ：还可继续装入最大价值的上界

- $F(X) = \sum_{i=1}^{k} v_i x_i + \left(b - \sum_{i=1}^{k} w_i x_i\right) v_{k+1} / w_{k+1}$

  若对某个$j > k$有$b - \sum_{i=1}^{k} w_i x_i \geq w_j$（即放得下某个物品）

- $F(X) = \sum_{i=1}^{k} v_i x_i$

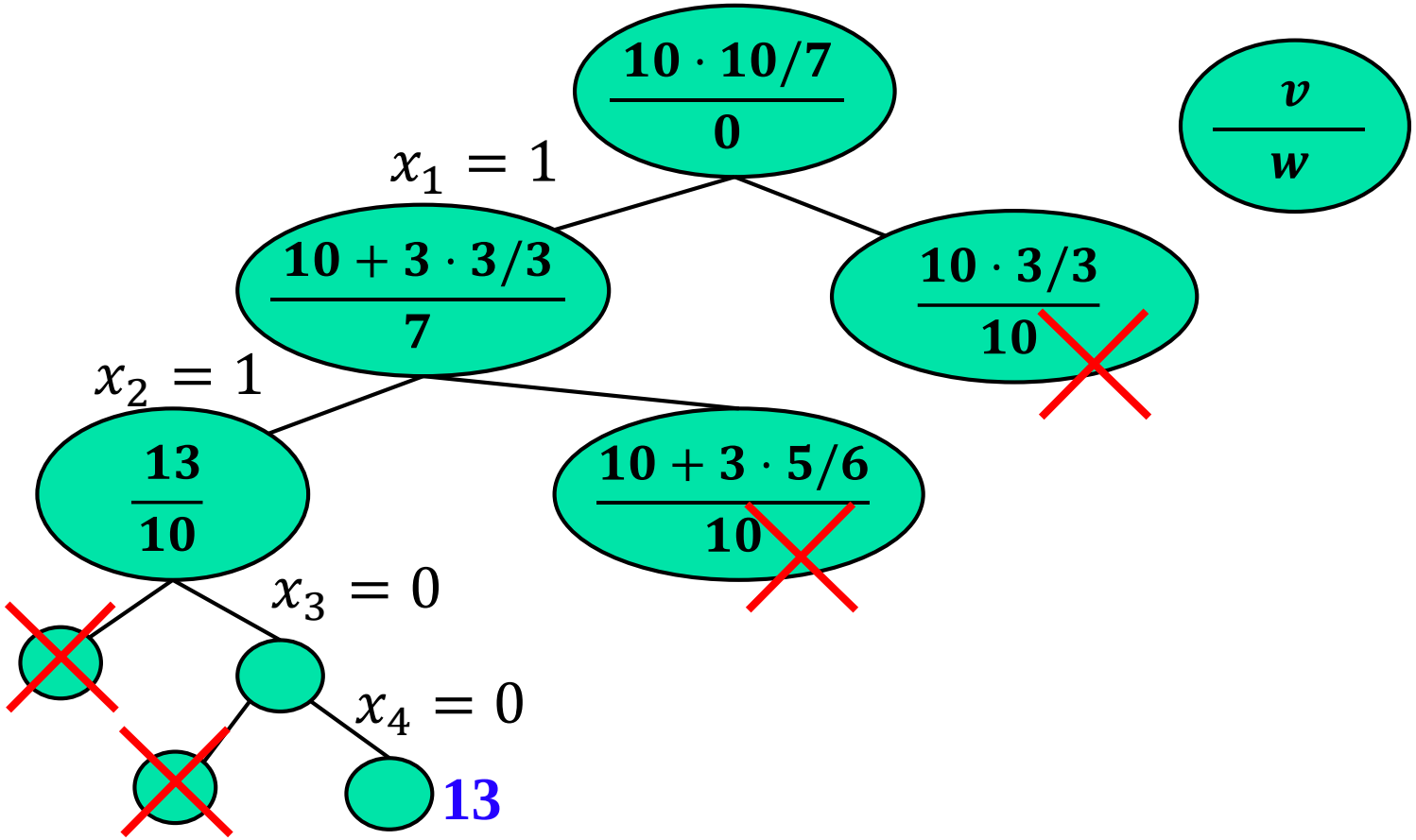
  否则（即放不下剩下的任一物品）

b：背包容量
w_i：物品i重量
v_i：物品i价值

实例：改进的回溯法

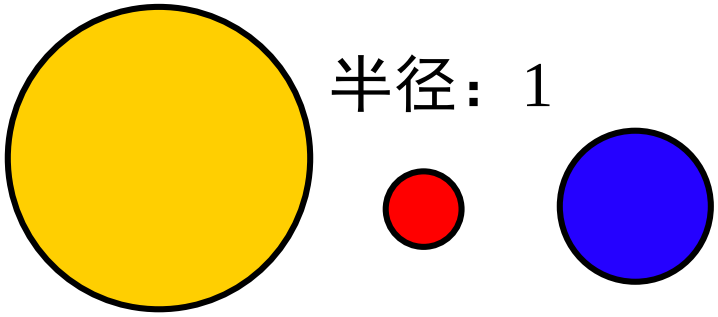
最大化　$10x_1 + 3x_2 + 5x_3 + x_4$

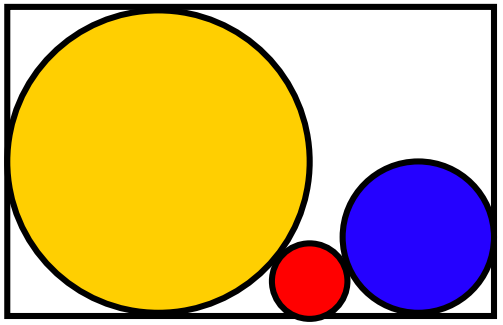
满足 $7x_1 + 3x_2 + 6x_3 + 2x_4 \leq 10; x_i \in \{0,1\},\ i = 1,2,3,4$



圆排列问题

- 问题定义
  - 给定n个大小不等的圆
  - 排进一个矩形框中，各圆与矩形框的底边相切
  - 目标：求有最小长度的圆排列

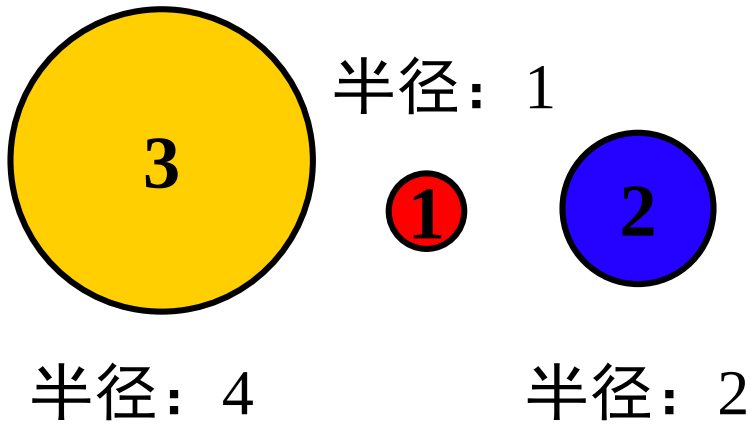




长度：$2\sqrt{2}+10$

圆排列问题的解空间

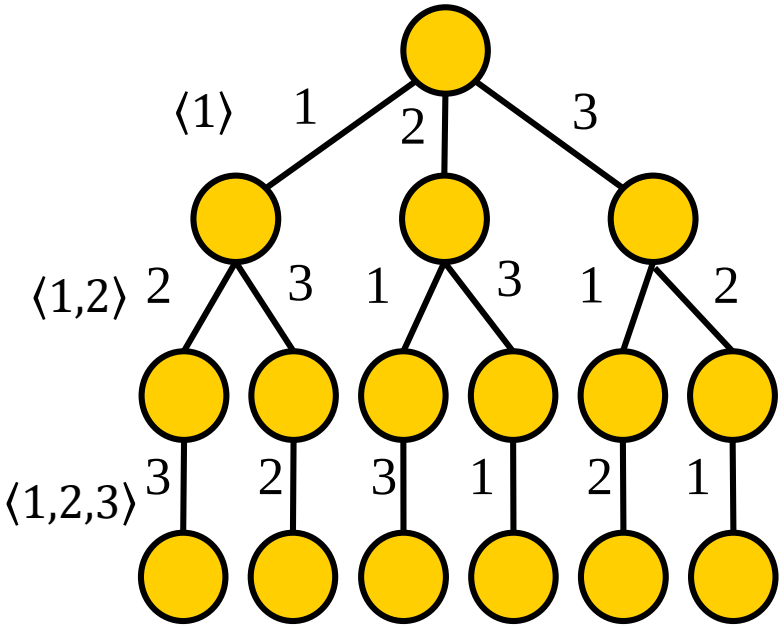
- 实例
  - 给出三个圆1,2,3
  - 半径为1,2,4



- 问题的可行解
  - 圆的所有排列
  - 如⟨1,2,3⟩，⟨1,3,2⟩

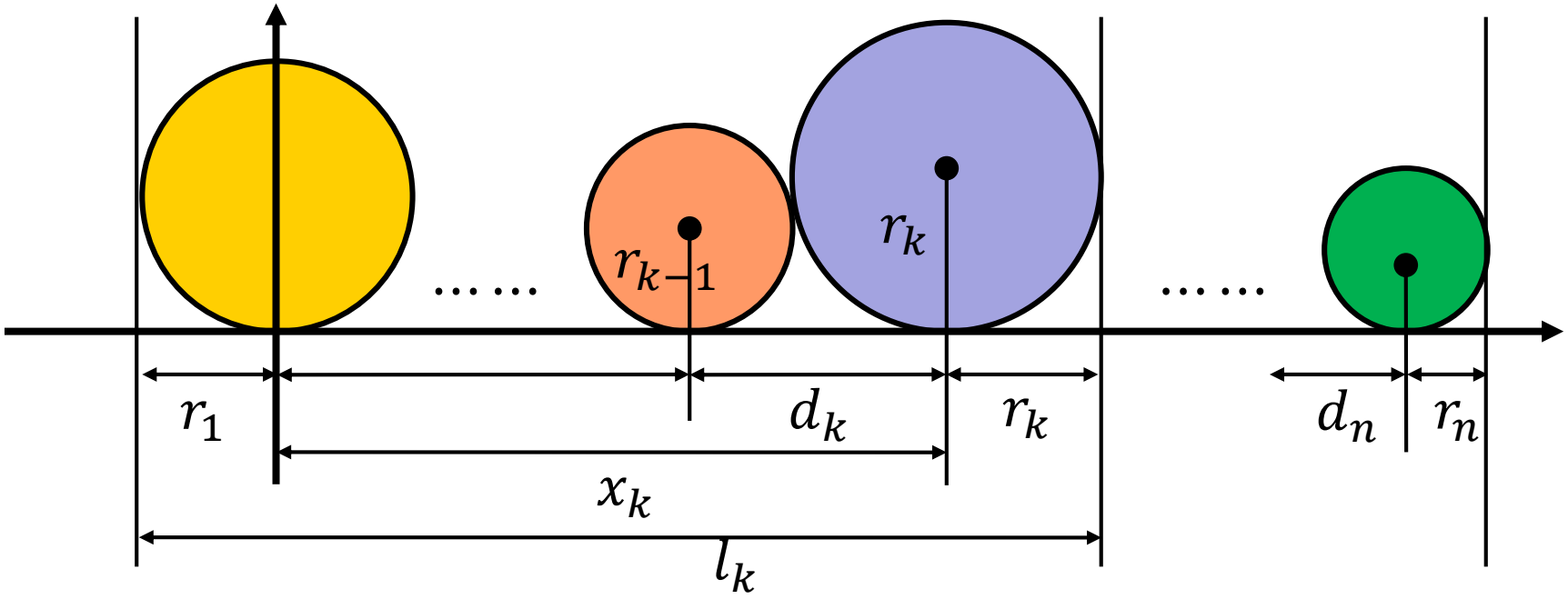
圆排列问题的解空间树

- 实例
  - 给出三个圆1,2,3
  - 半径为1,2,4
- 限界函数
  - 界：当前最短长度的圆排列
  - 代价函数？



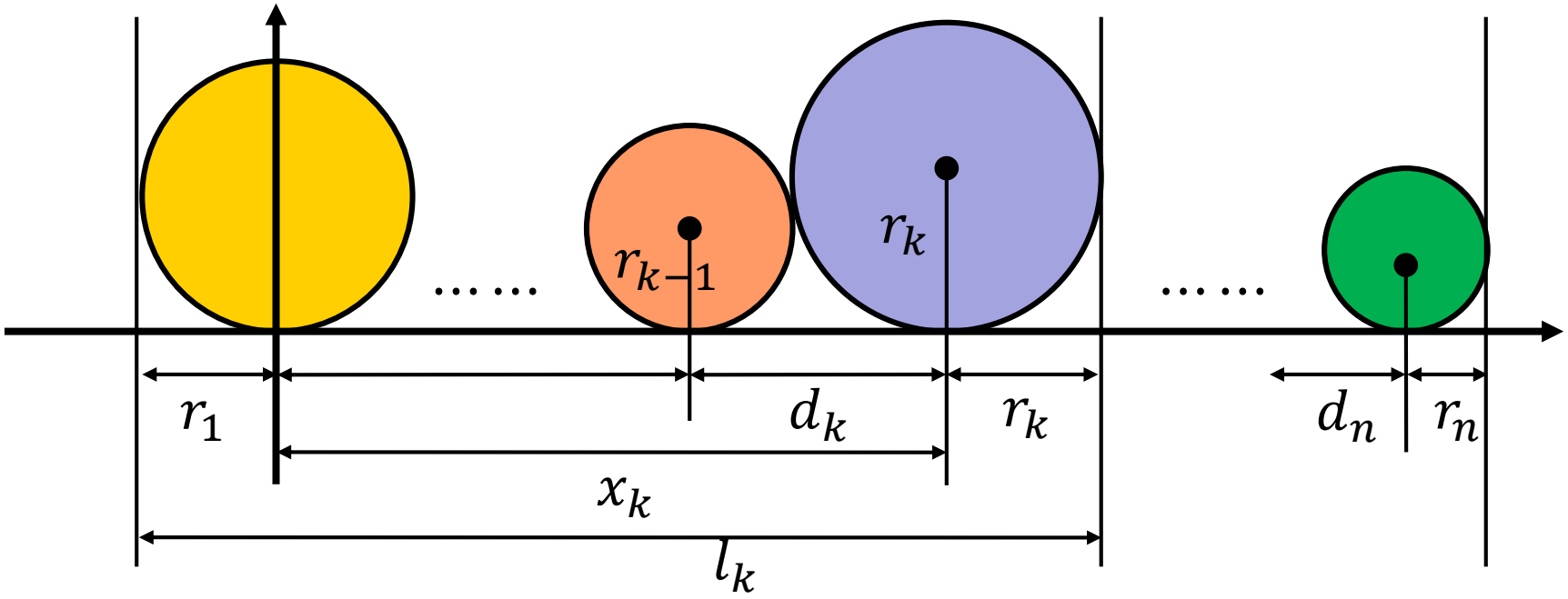
圆排列问题的代价函数

- r_i：第i个圆的半径
- d_k：第k-1个圆和第k个圆的圆心距离
- x_k：原点到第k个圆的圆心距离
- l_k：前k个圆排列的长度



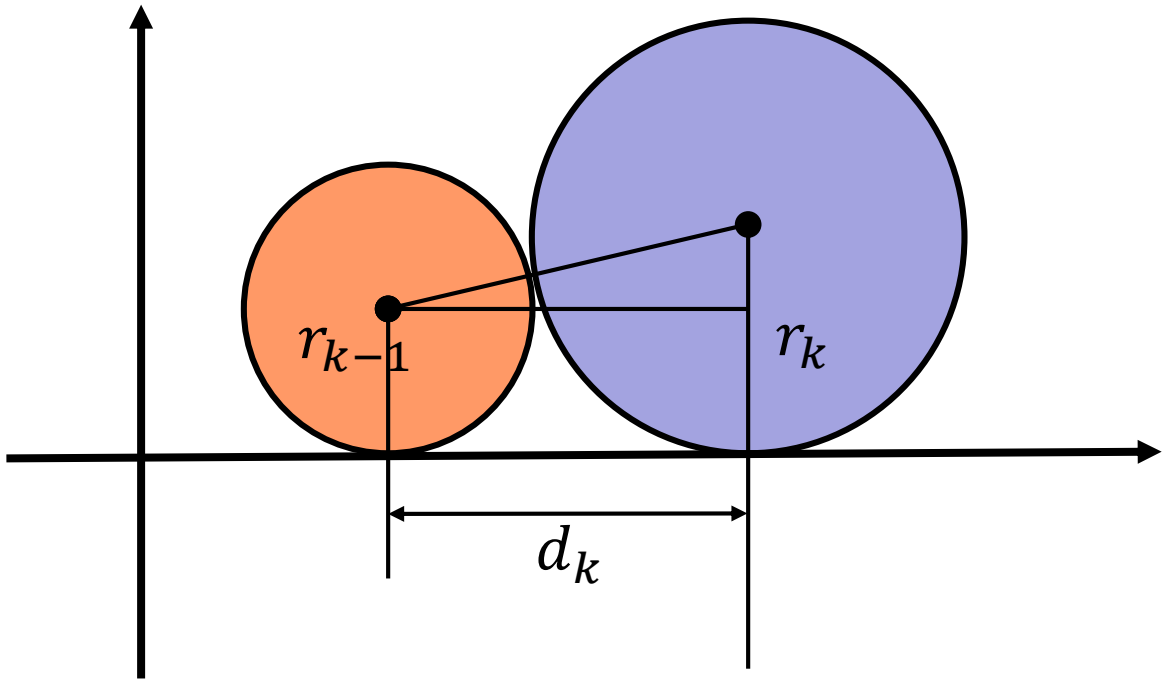
圆排列问题的代价函数

- 前k个圆排列的长度：$l_k = x_k + r_k + r_1$
- 排列长度：$l_n = x_k + d_{k+1} + d_{k+2} + \cdots + d_n + r_1 + r_n$



圆排列问题的代价函数

- $d_k = \sqrt{(r_{k-1} + r_k)^2 - (r_{k-1} - r_k)^2} = 2\sqrt{r_k r_{k-1}}$



圆排列问题的代价函数

估算排列长度：

$$l_n = x_k + d_{k+1} + d_{k+2} + \cdots + d_n + r_1 + r_n$$
$$= x_k + 2\sqrt{r_k r_{k+1}} + 2\sqrt{r_{k+1} r_{k+2}} + \cdots + 2\sqrt{r_{n-1} r_n} + r_1 + r_n$$
$$\geq x_k + 2(n-k)r_{min} + r_{min} + r_1$$

r_{min}：$r_k, r_{k+1}, \cdots r_n$中的最小值　$d_k = 2\sqrt{r_k r_{k-1}}$

代价函数：

$$x_k = x_{k-1} + d_k$$
$$F(X) = x_k + 2(n-k)r_{min} + r_{min} + r_1$$

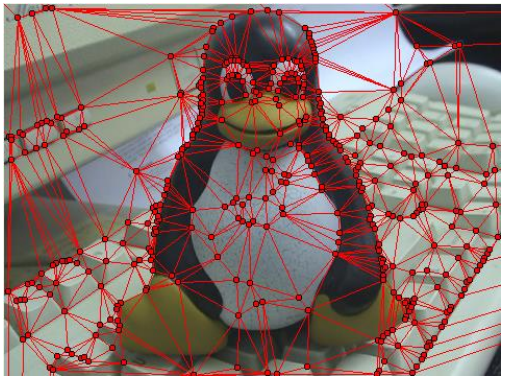
剪枝条件：

$$F(X) \geq \text{bestLength}$$

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

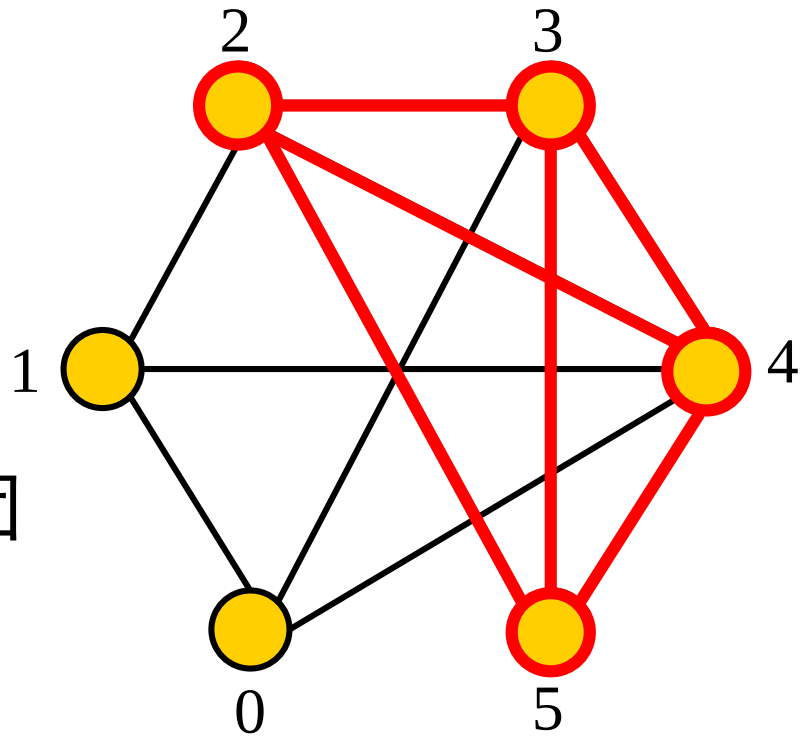
社会网络中的聚类分析



计算机视觉

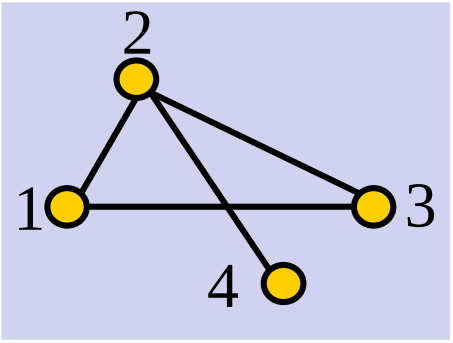
最大团问题的解空间

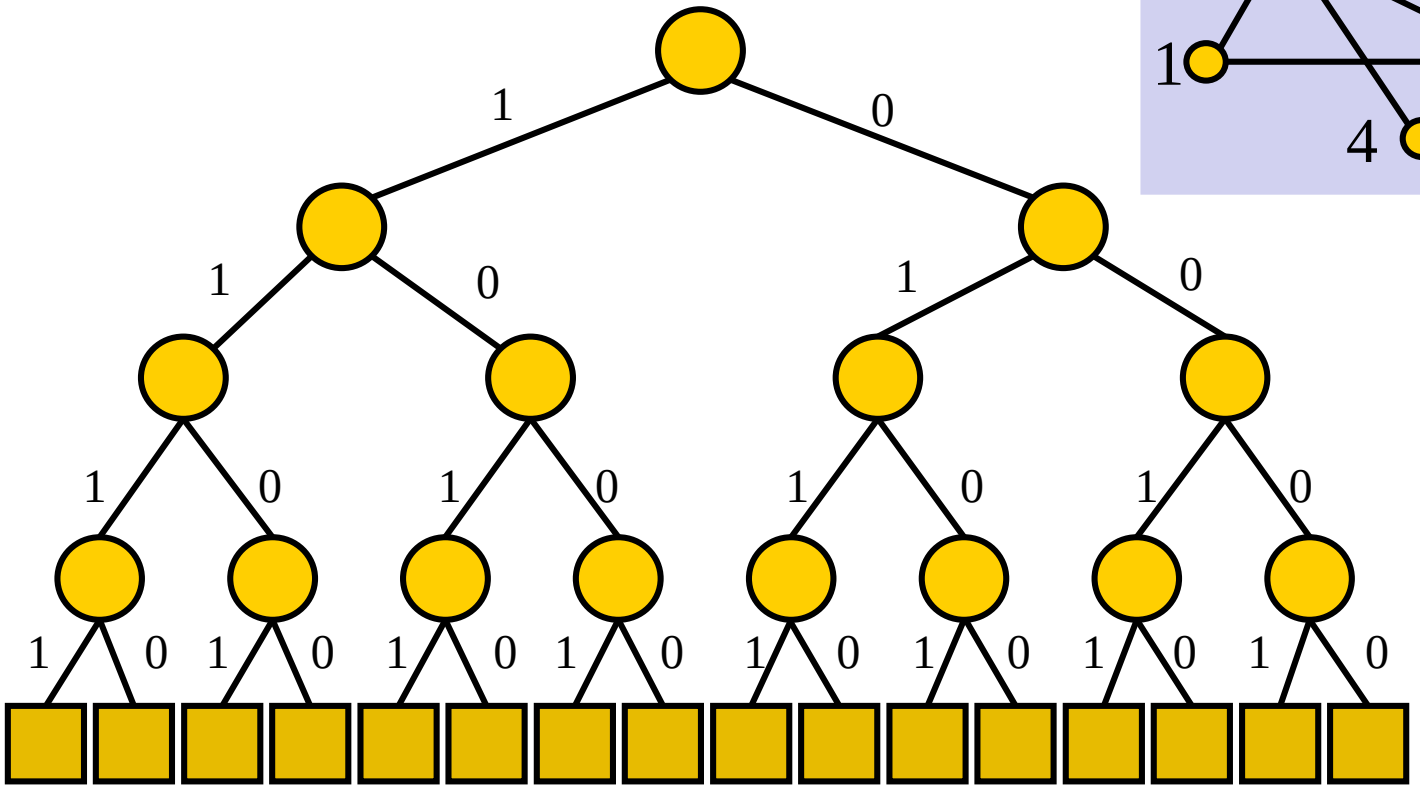
- 实例
  - 6个顶点，11条边
- 问题的解
  - 顶点2,3,4,5构成最大团
  - 解的表示⟨0,0,1,1,1,1⟩



最大团问题的解空间树

- 一棵二叉树





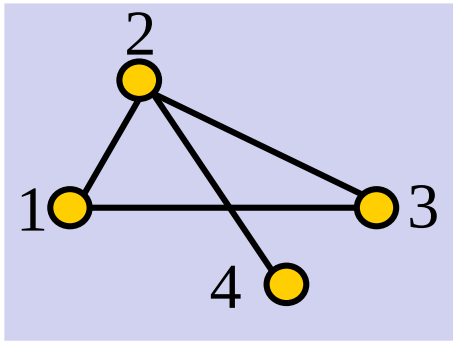
最大团问题的剪枝函数

- 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：

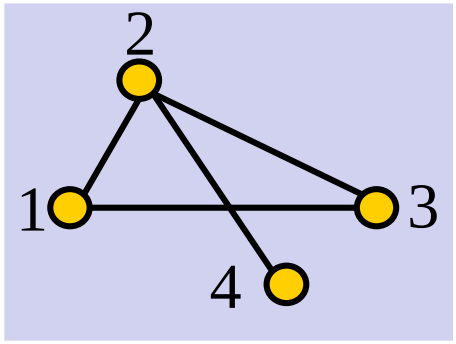
当前解的顶点数（cn）+未检查的顶点的数目（$n-i$）

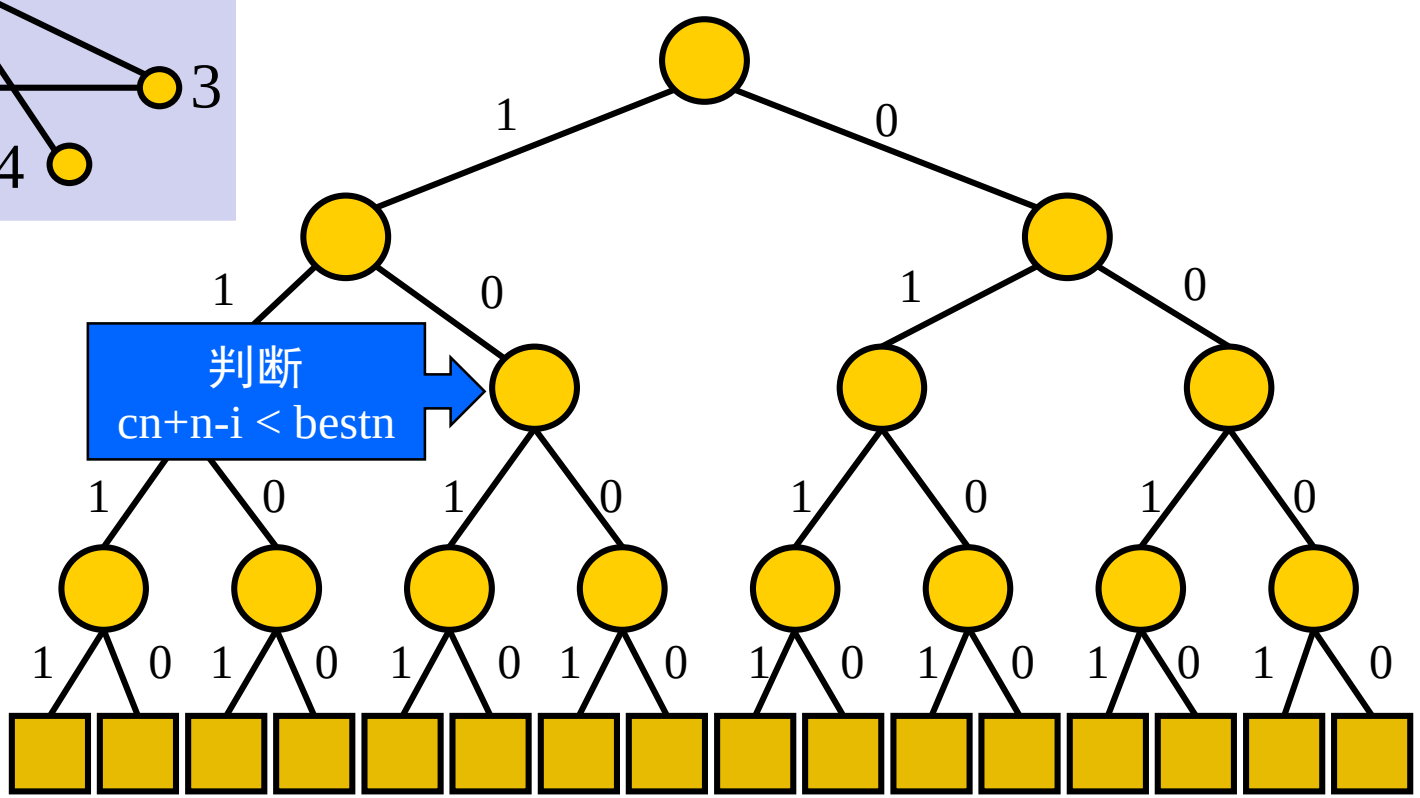
  - 剪枝条件：$cn+n-i < bestn$

剪枝之可行性约束函数

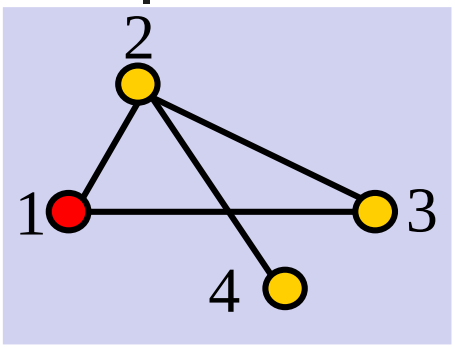


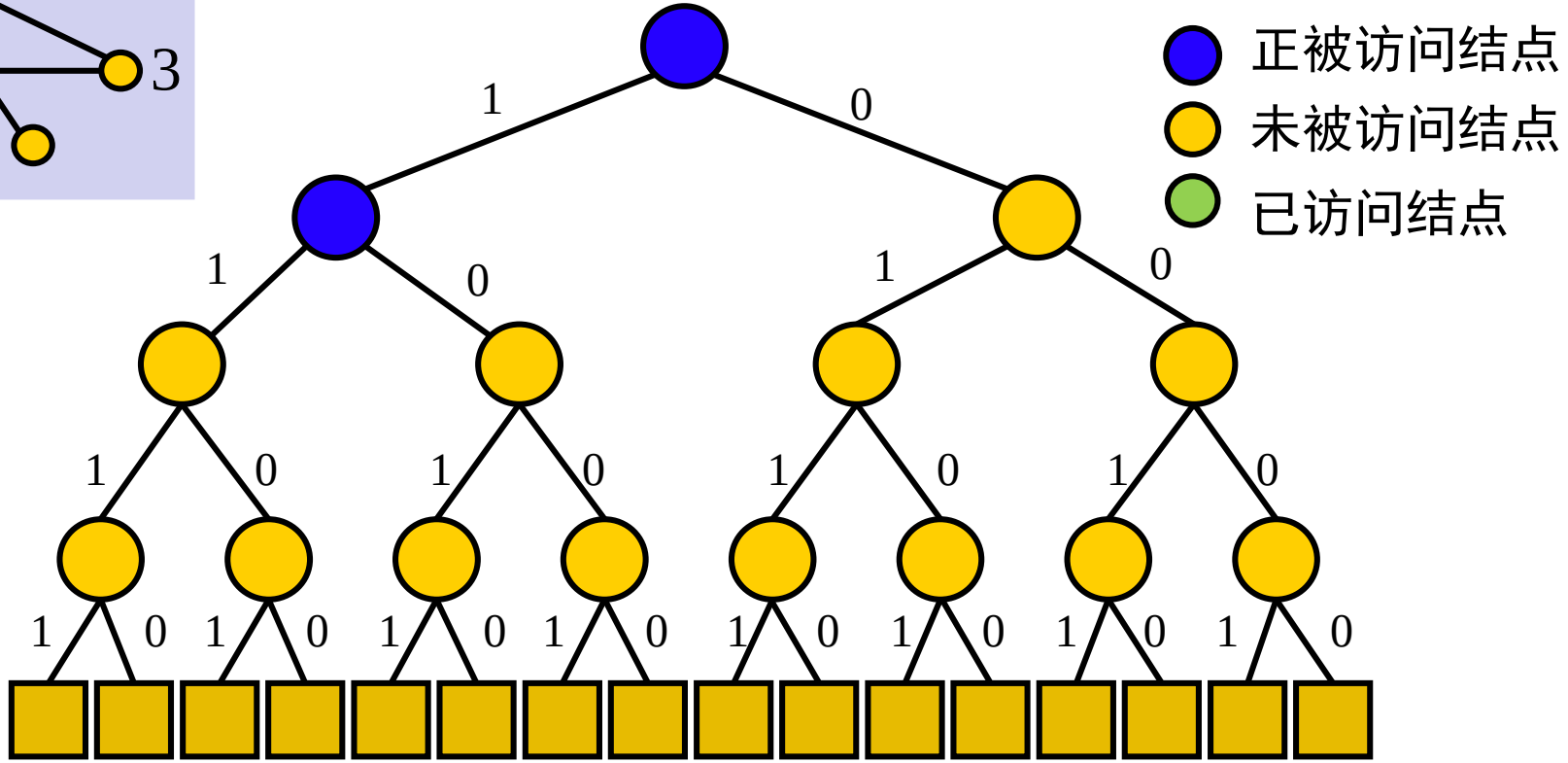
剪枝之限界函数



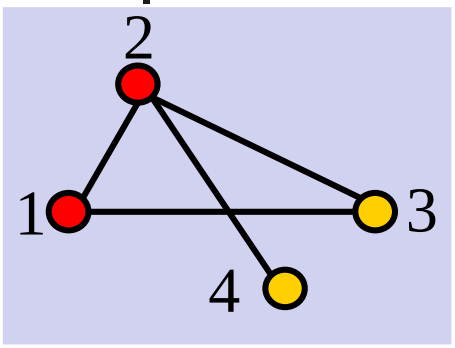


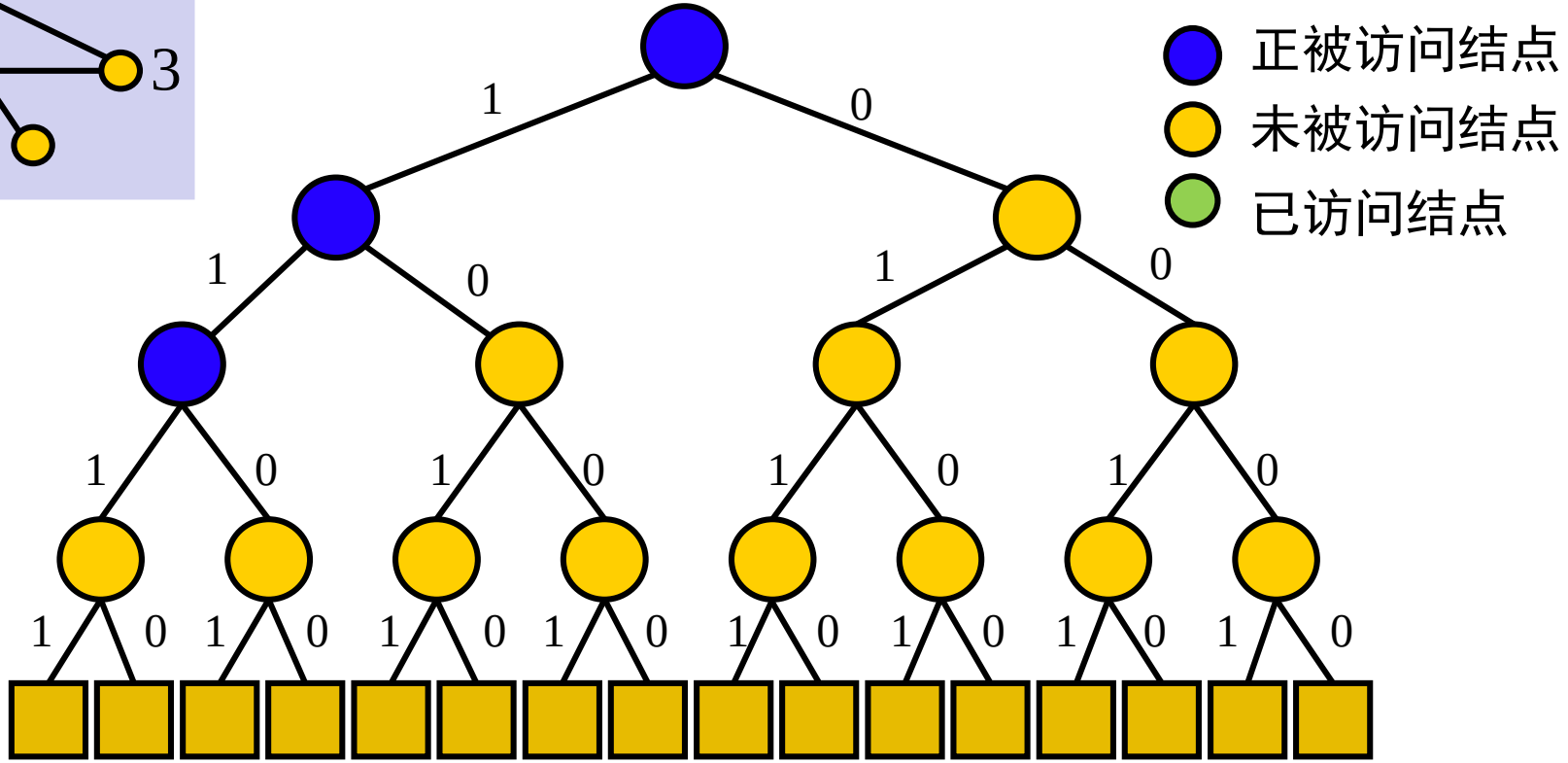
最大团问题的运行实例（1）



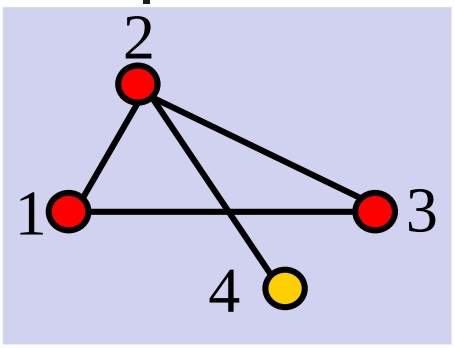


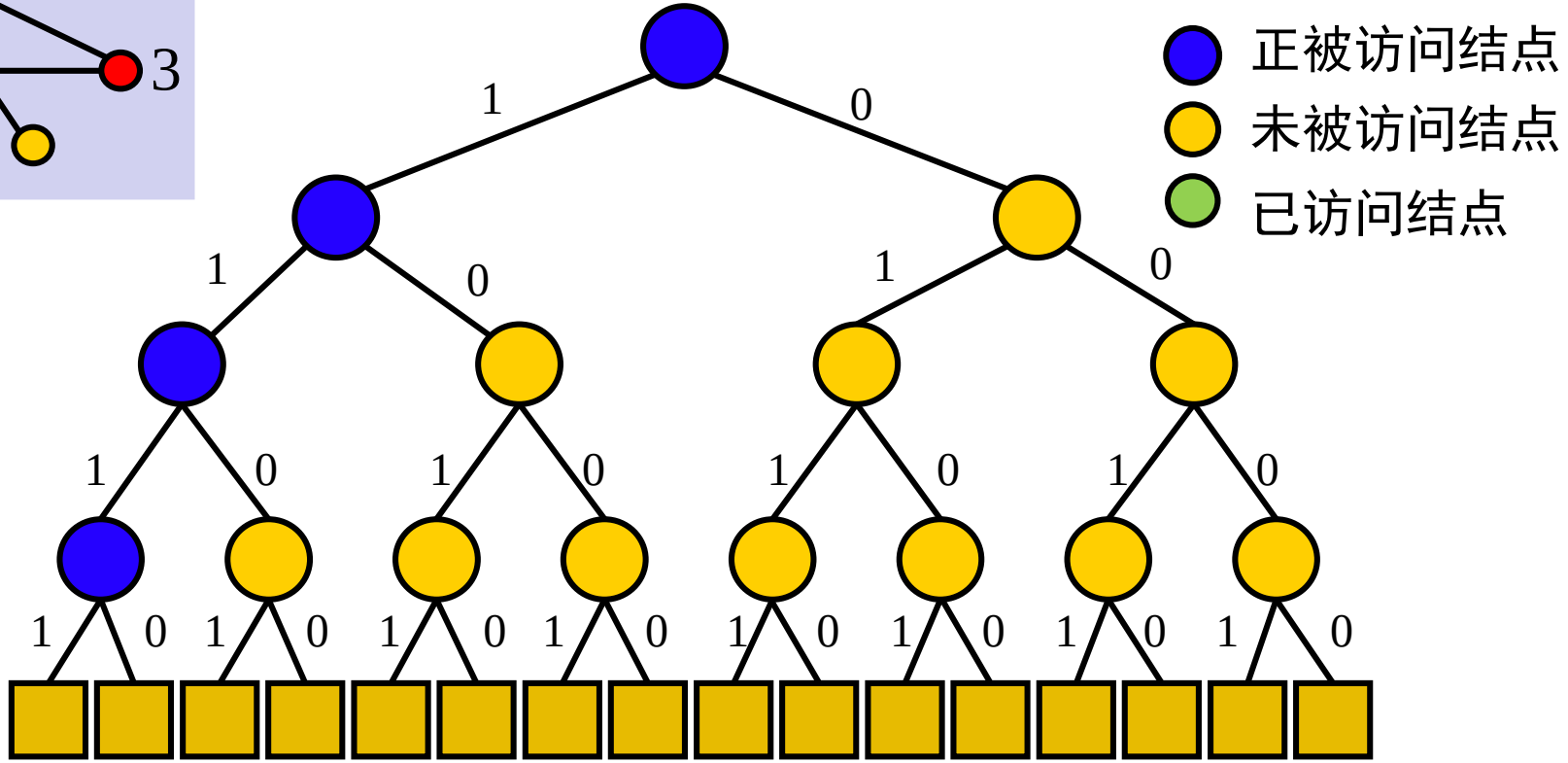
最大团问题的运行实例（2）



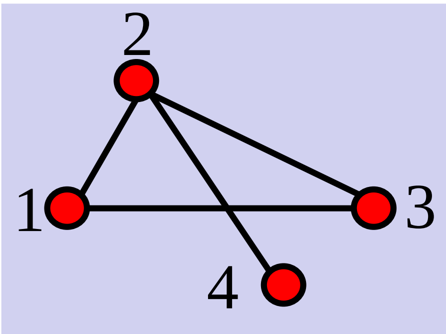


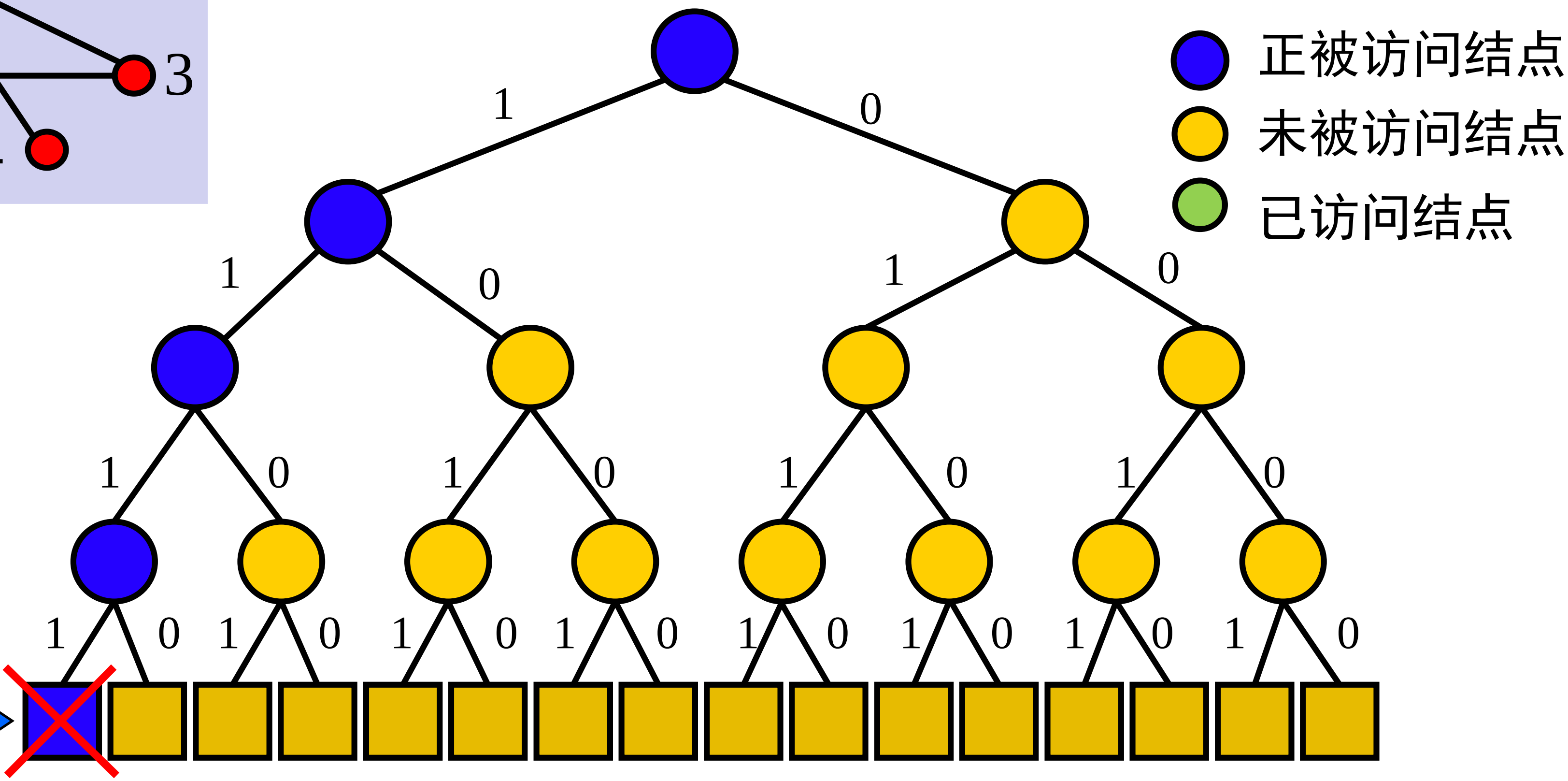
最大团问题的运行实例（3）



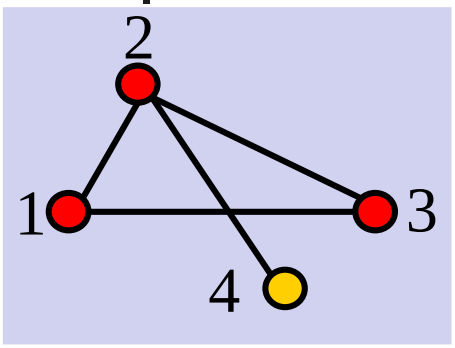


最大团问题的运行实例（4）





最大团问题的运行实例（5）

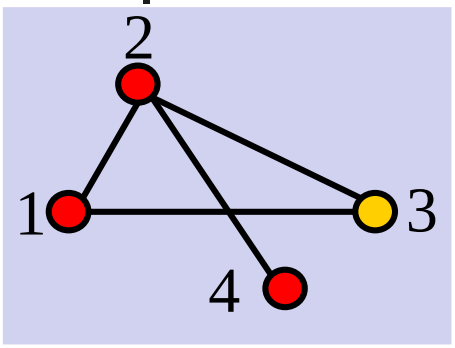


当前最大团规模为3

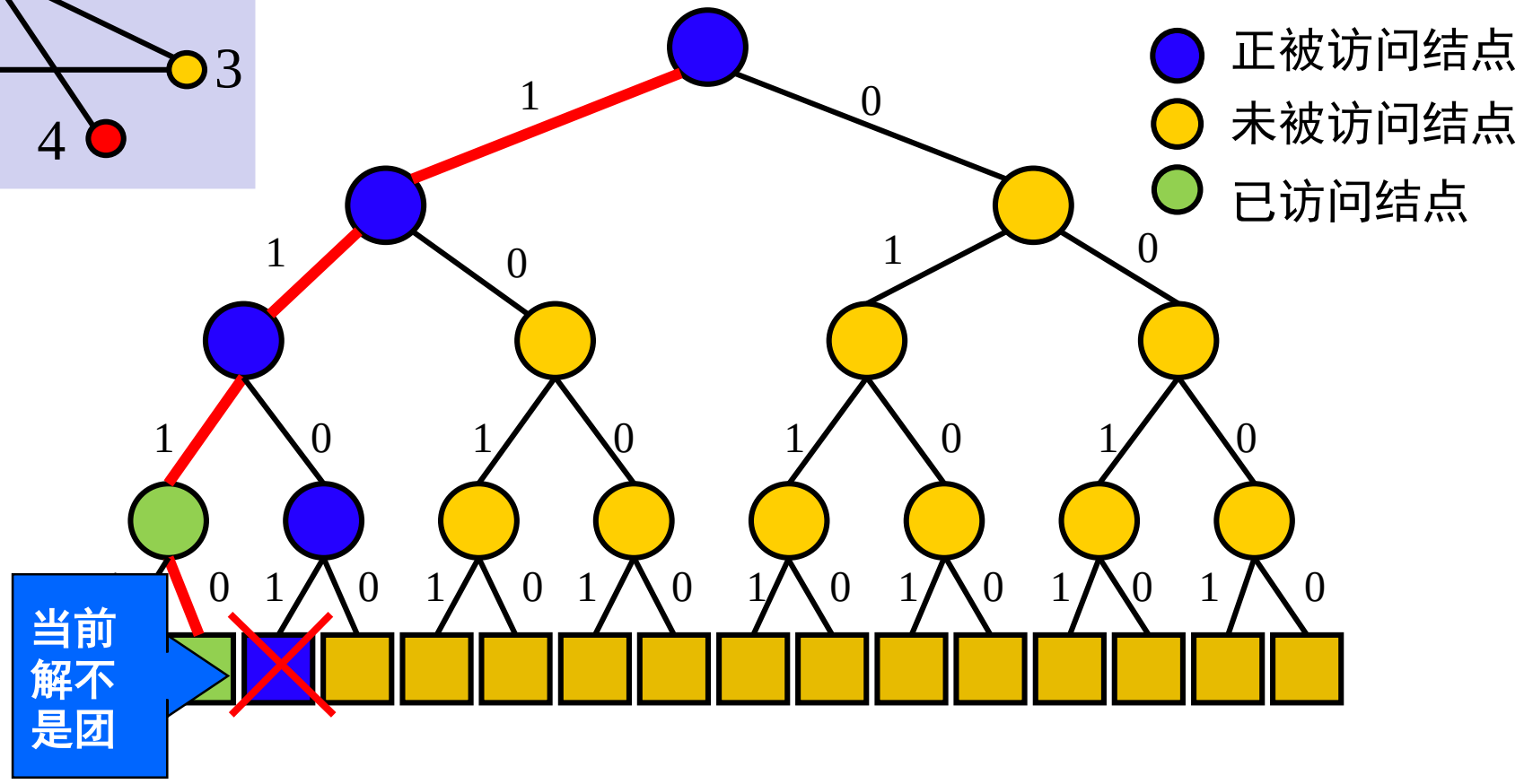
正被访问结点
未被访问结点
已访问结点

找到一个解！

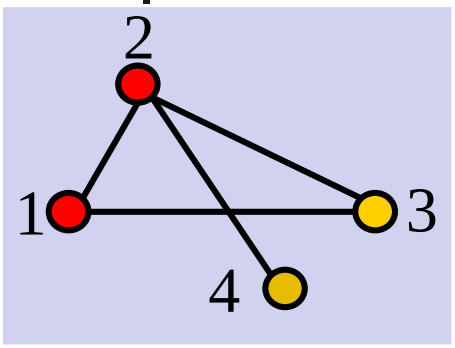
最大团问题的运行实例（6）



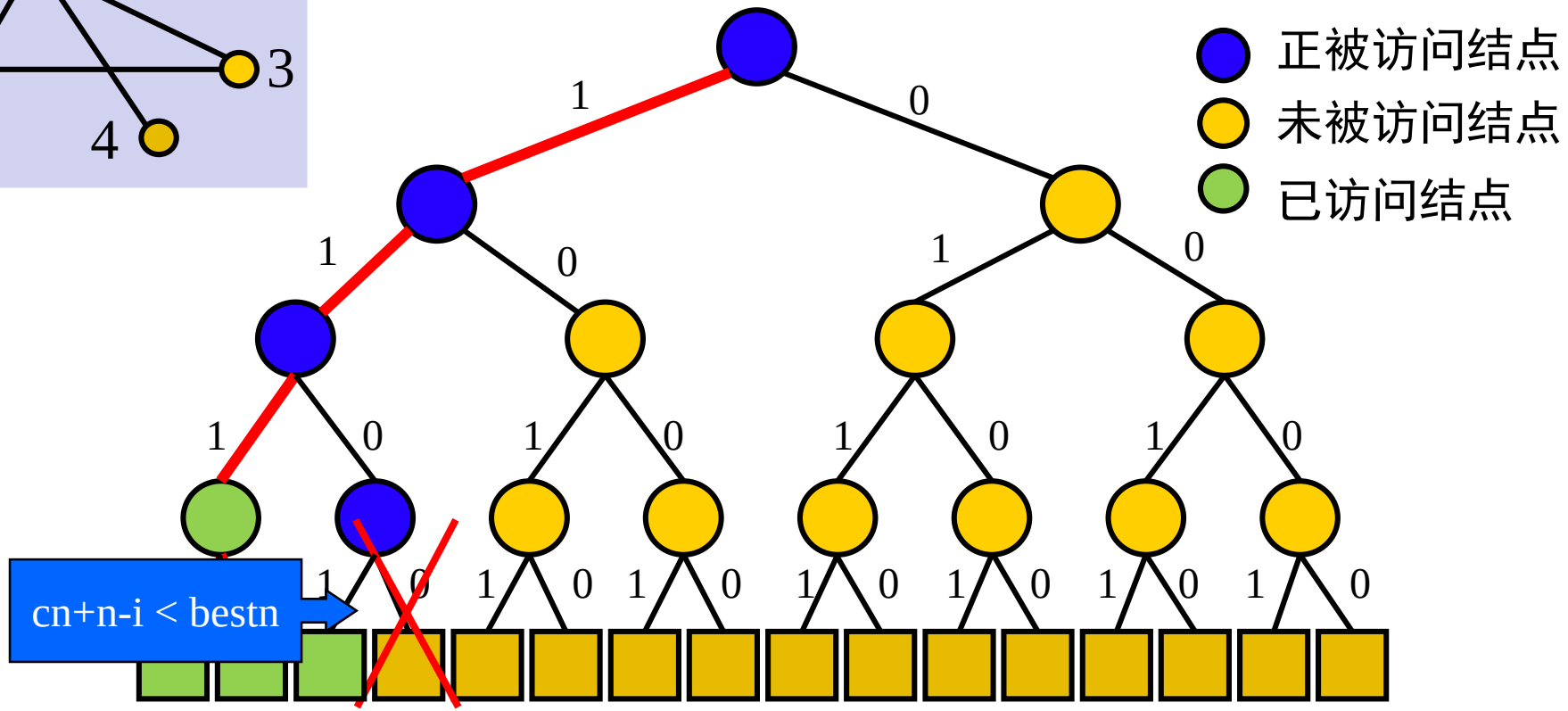
当前最大团规模为3



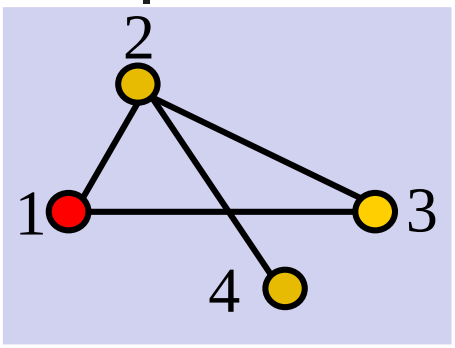
最大团问题的运行实例（7）



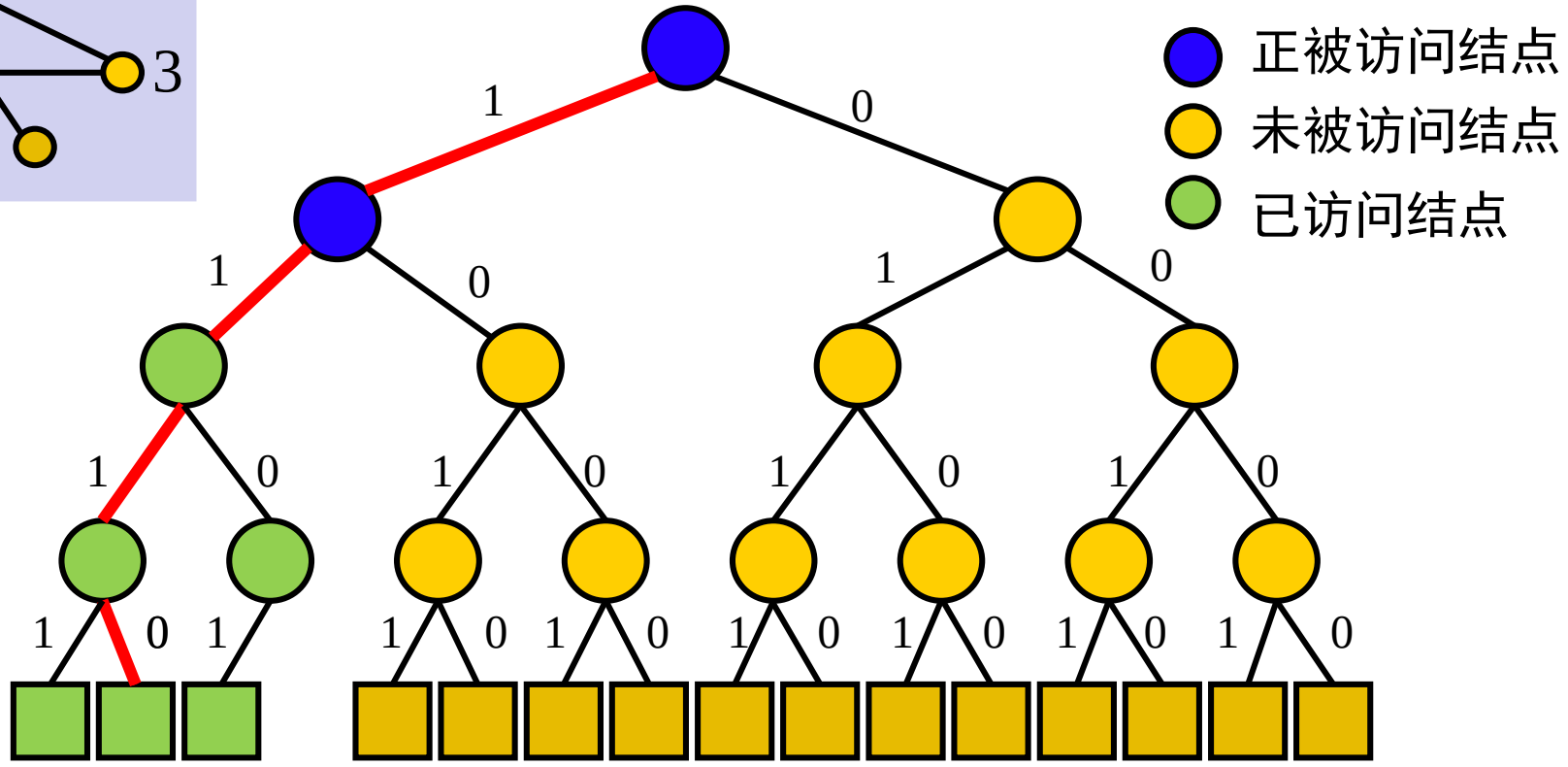
当前最大团规模为3



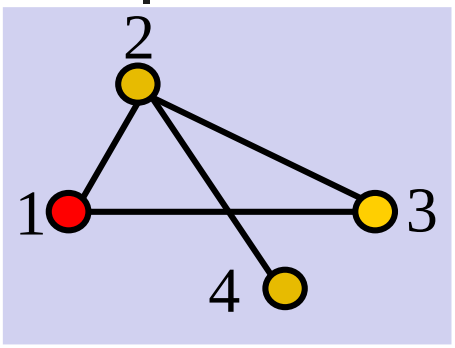
最大团问题的运行实例（8）



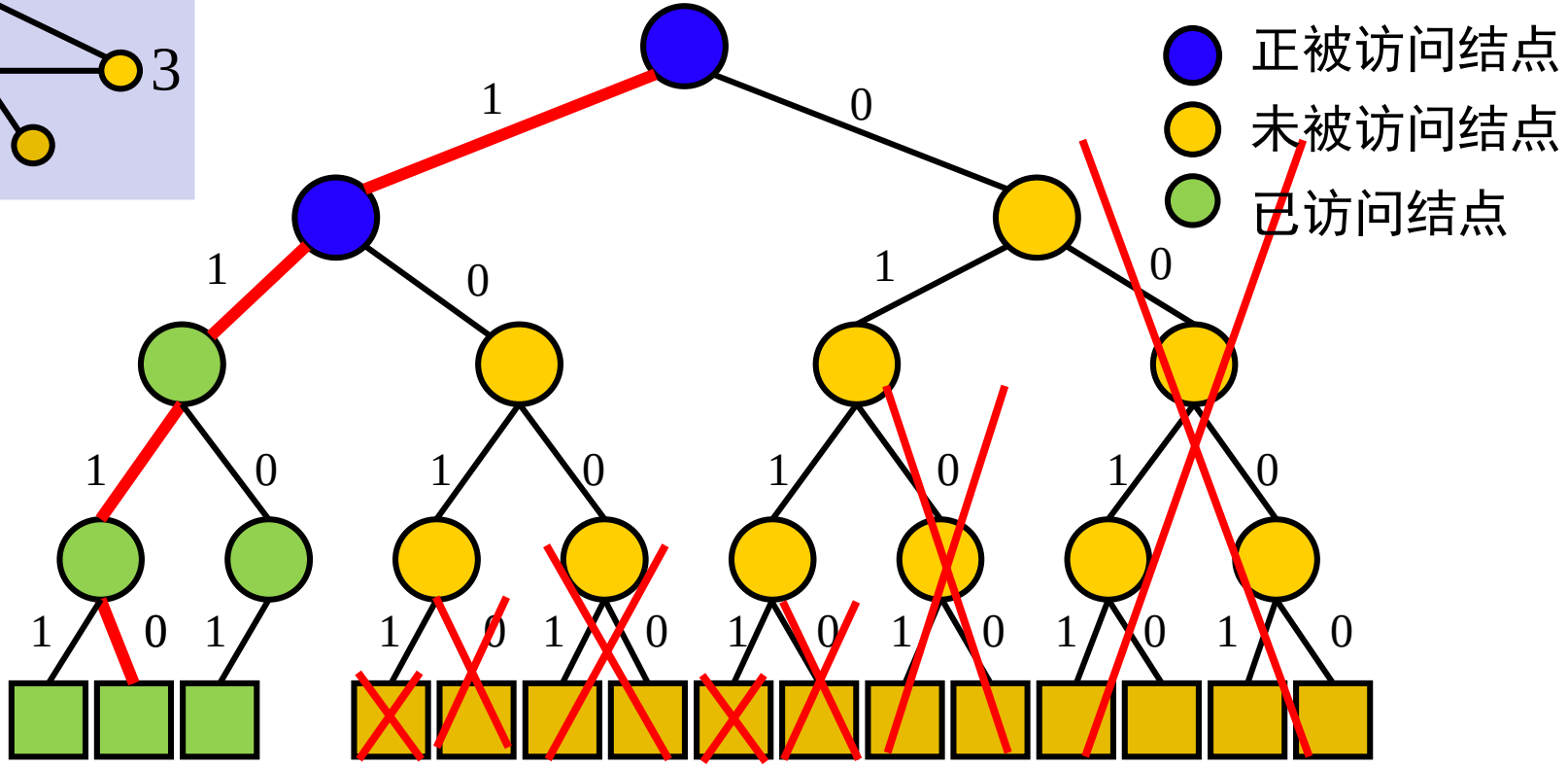
当前最大团规模为3



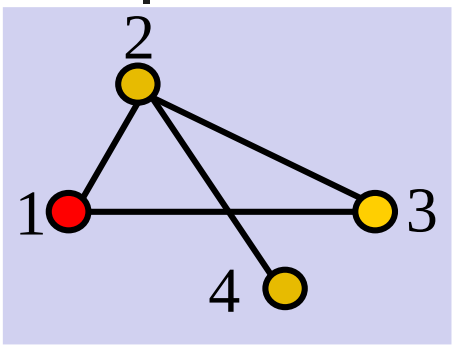
最大团问题的运行实例（9）



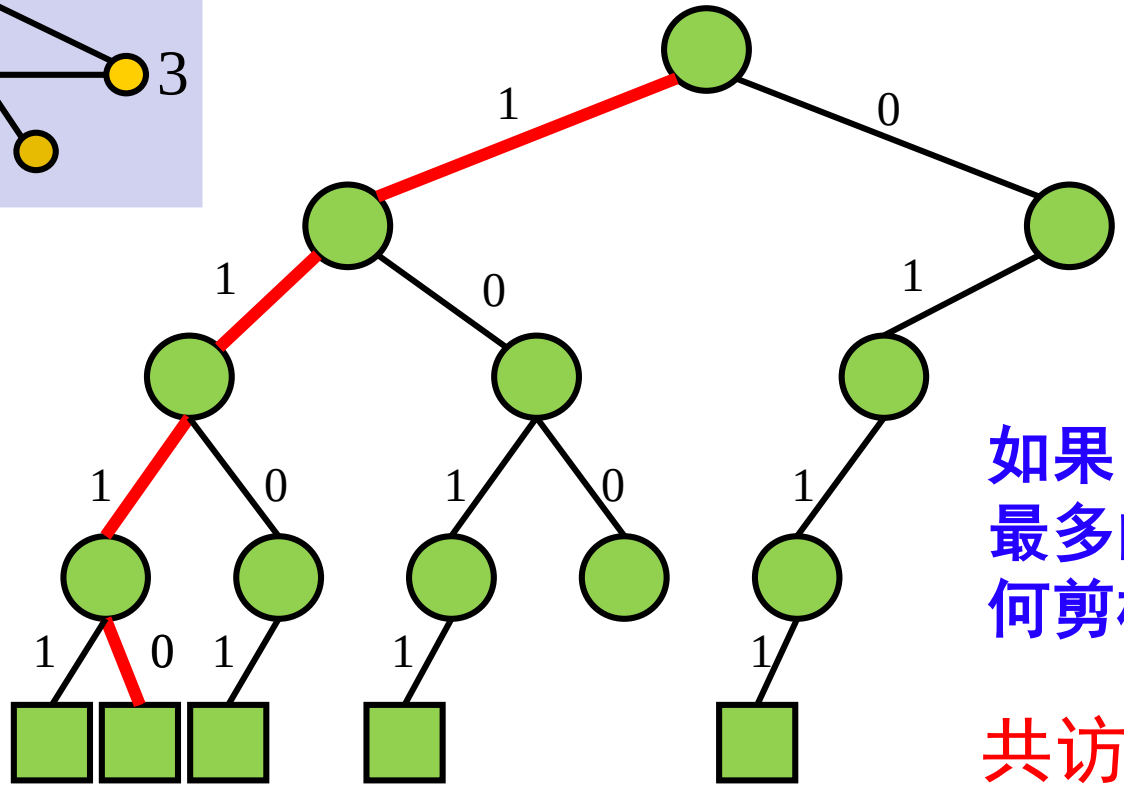
当前最大团规模为3



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
}
```

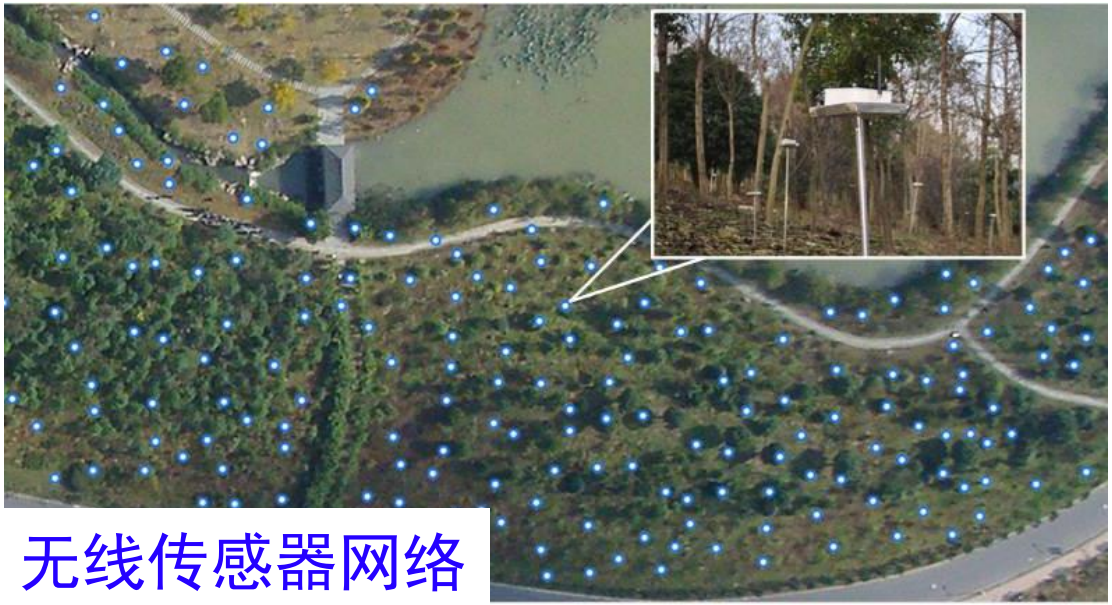
是否还有更好的剪枝策略？

复杂度分析

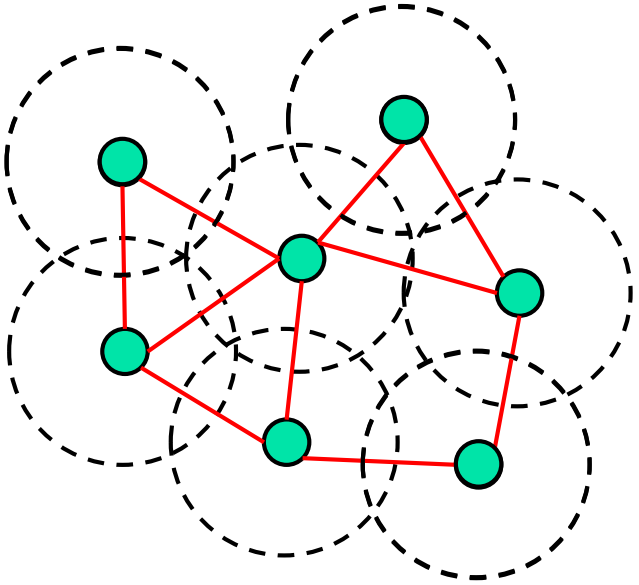
最大团问题的回溯算法**backtrack**所需的计算时间为$O(n2^n)$。

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

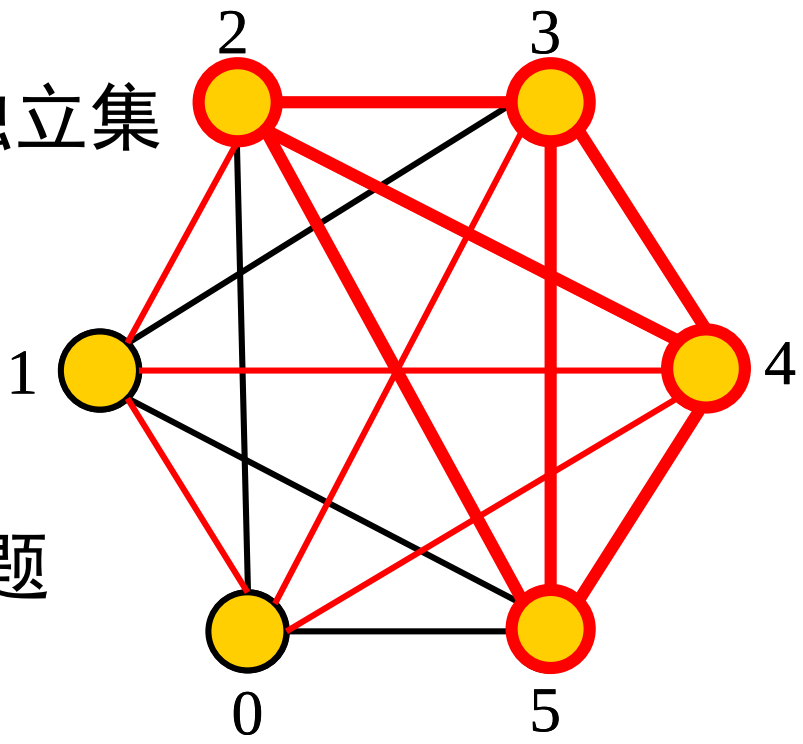


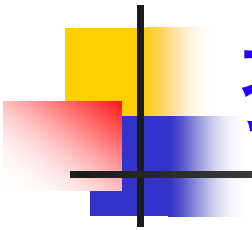
无线传感器网络



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）

本章小结

- 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