课程回顾

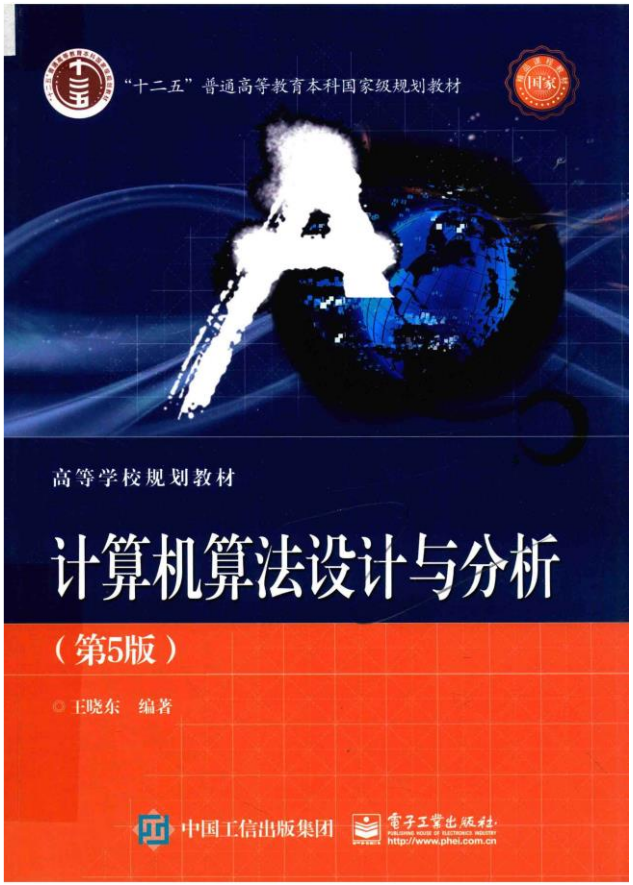
- 贪心算法
  - 贪心算法理论：加权拟阵中寻找最大权重的独立子集问题->贪心算法可求得最优解
  - 贪心算法应用：最优装载、单源最短路径、最小生成树
- 近似算法
  - 近似比
  - PTAS/FPTAS
  - 求解实例：顶点覆盖问题、TSP、集合覆盖问题、子集和问题

第6章 回溯法

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

参考书目

■接下来几章内容可参考以下书本：



王晓东 编著
《计算机算法设计与分析》(第5版)
电子工业出版社

本章内容

- 回溯法算法框架：问题解空间，基本思想，递归回溯，迭代回溯，子集树与排列树
- 实际问题：装载问题，批处理作业调度，符号三角形问题，n后问题，0-1背包问题，最大团问题，图的m着色问题，TSP，圆排列问题，电路板排列问题，连续邮资问题
- 回溯法效率分析

回溯法概述

- 回溯法有“通用的解题法”之称，可以系统地搜索一个问题的所有解或任一解
- 在问题解空间树中按照深度优先策略，从根结点出发搜索解空间树
- 适合解组合数较大的问题

回溯法算法框架

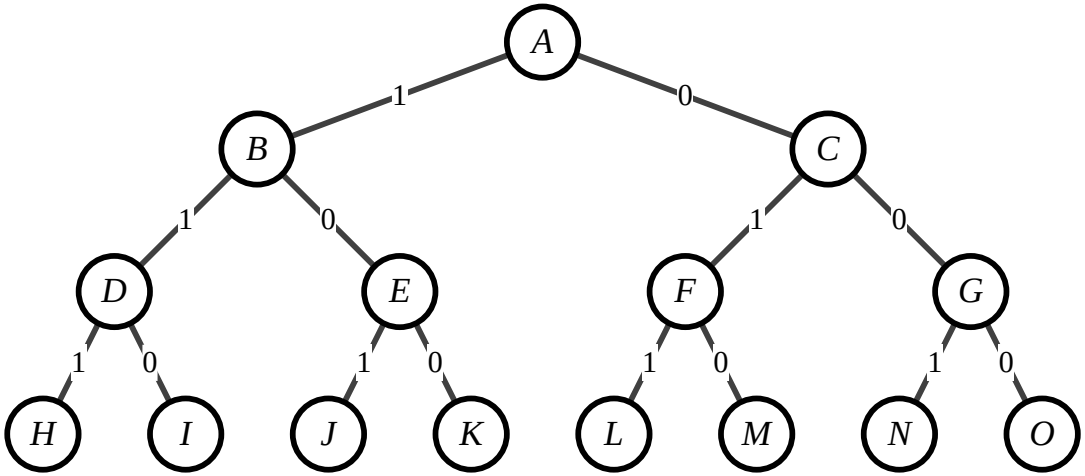
■问题的解空间

➢应明确定义，至少包含问题的一个（最优）解

➢例：0-1背包问题

解空间：长度为n的0-1向量，第i位为1表示选取物品i

如n=3时，解空间为{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}



回溯法算法框架 (续)

- 回溯法基本思想
  - 从开始结点（根结点）出发，以深度优先方式搜索整个解空间
  - 当前结点为扩展结点，可继续纵深搜索的结点为活结点，无法纵深搜索的为死结点
  - 扩展结点为死结点时，回溯至最近的一个活结点处，这个活结点作为扩展结点，递归求解，直至找到解或无活结点为止

回溯法算法框架

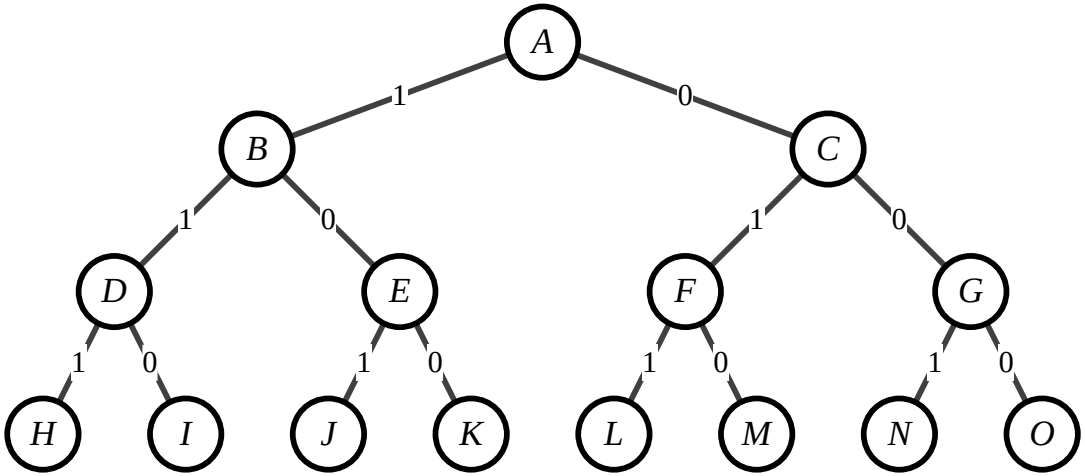
■问题的解空间

➢应明确定义，至少包含问题的一个（最优）解

➢例：0-1背包问题

解空间：长度为n的0-1向量，第i位为1表示选取物品i

如n=3时，解空间为{(0,0,0), (0,0,1), (0,1,0), (0,1,1), (1,0,0), (1,0,1), (1,1,0), (1,1,1)}

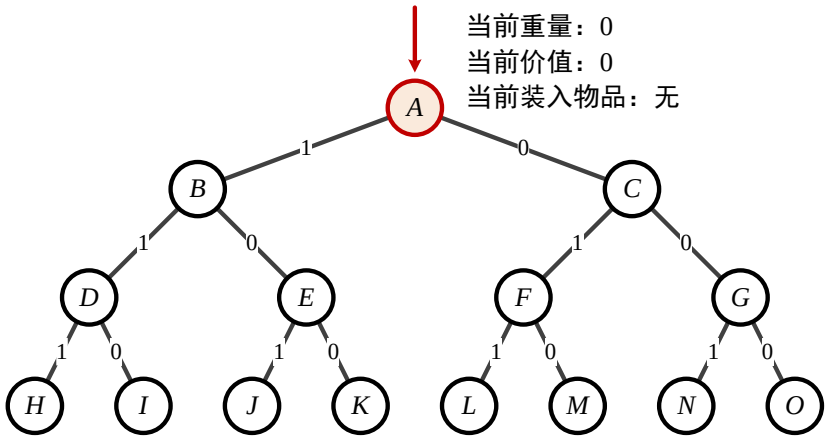


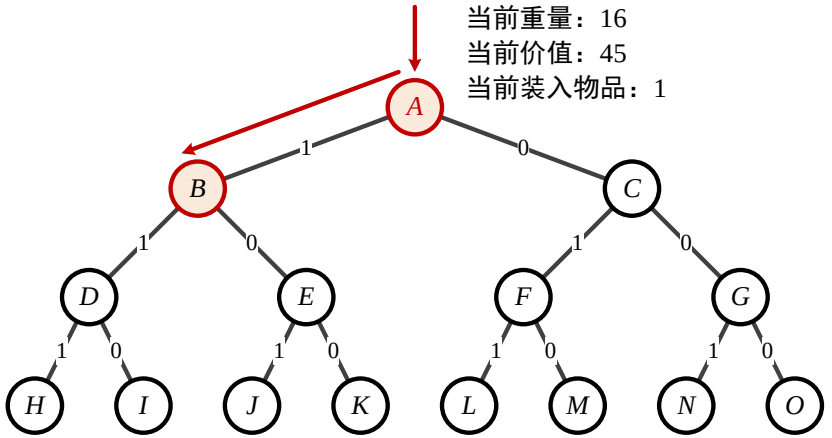
回溯法算法框架 (续)

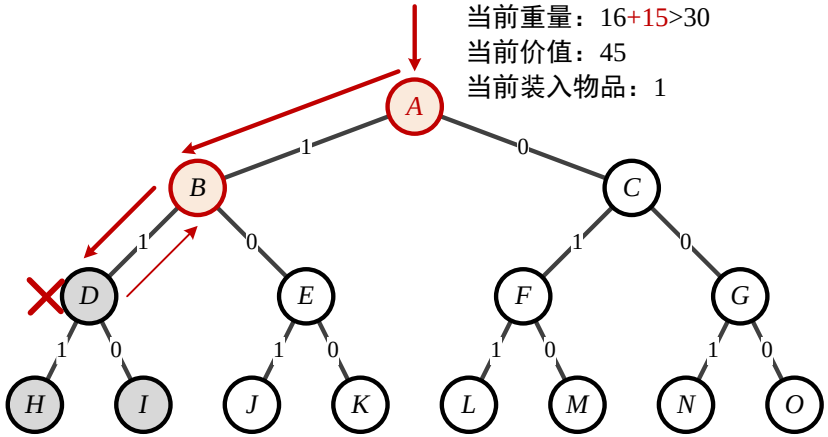
- 回溯法基本思想
  - 从开始结点（根结点）出发，以深度优先方式搜索整个解空间
  - 当前结点为扩展结点，可继续纵深搜索的结点为活结点，无法纵深搜索的为死结点
  - 扩展结点为死结点时，回溯至最近的一个活结点处，这个活结点作为扩展结点，递归求解，直至找到解或无活结点为止

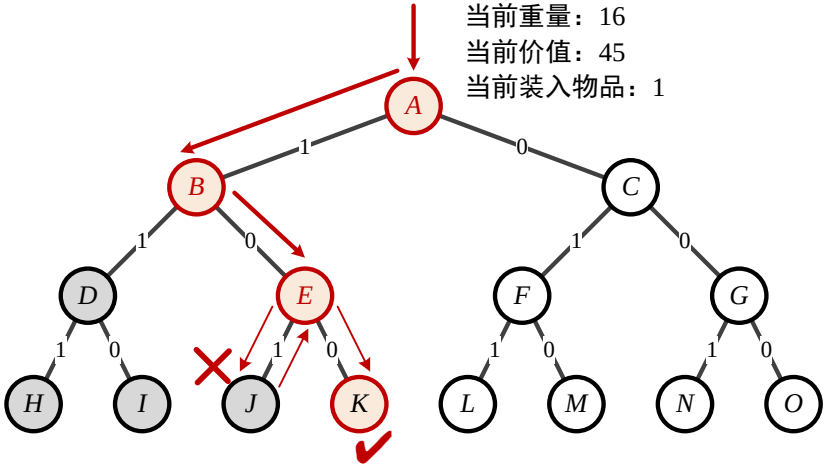
回溯法算法框架 (续)

- 例1：0-1背包问题：w=[16, 15, 15], v=[45, 25, 25], W=30



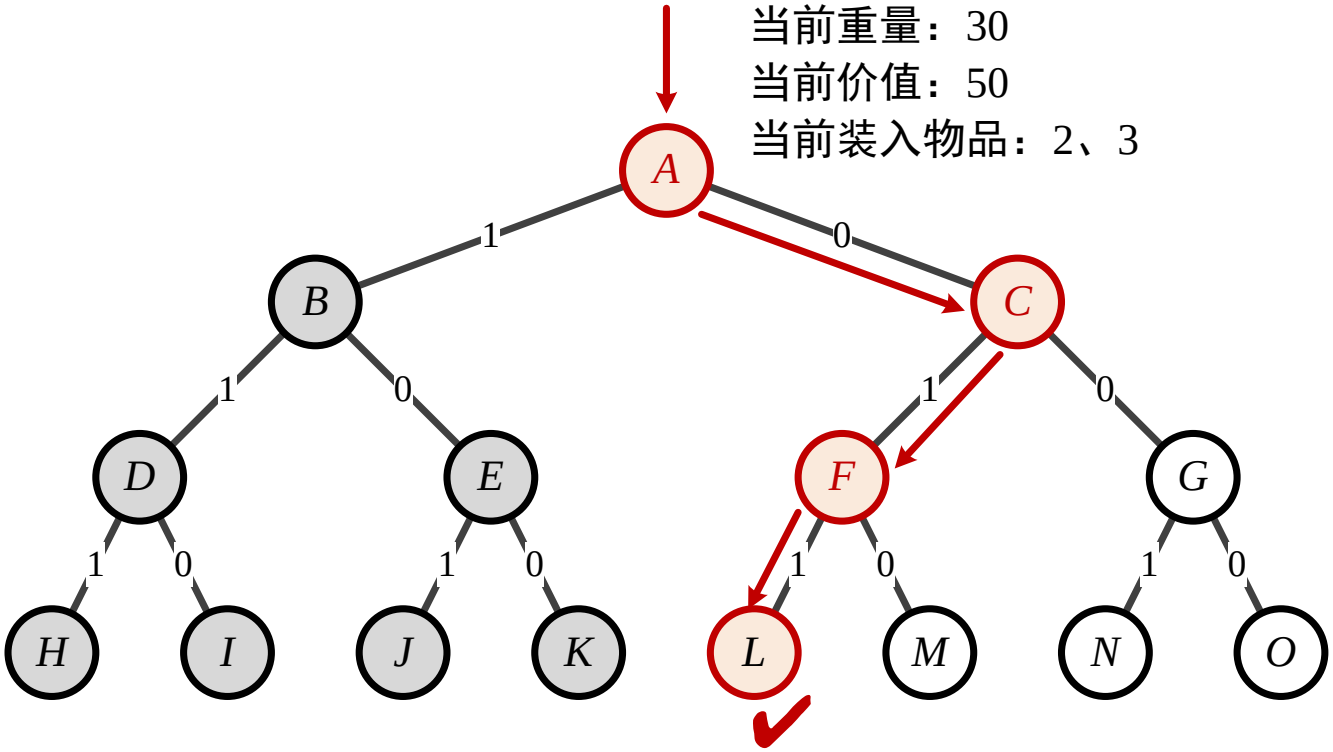






回溯法算法框架 (续)

- 例1：0-1背包问题：w=[16, 15, 15], v=[45, 25, 25], W=30



回溯法算法框架 (续)

- 例2：旅行售货员问题

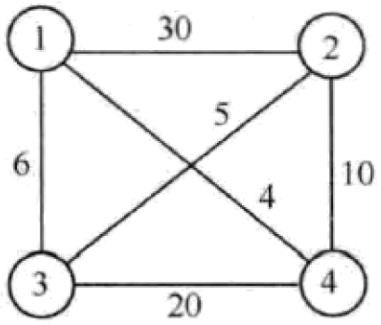


图 5-2　4 顶点带权图

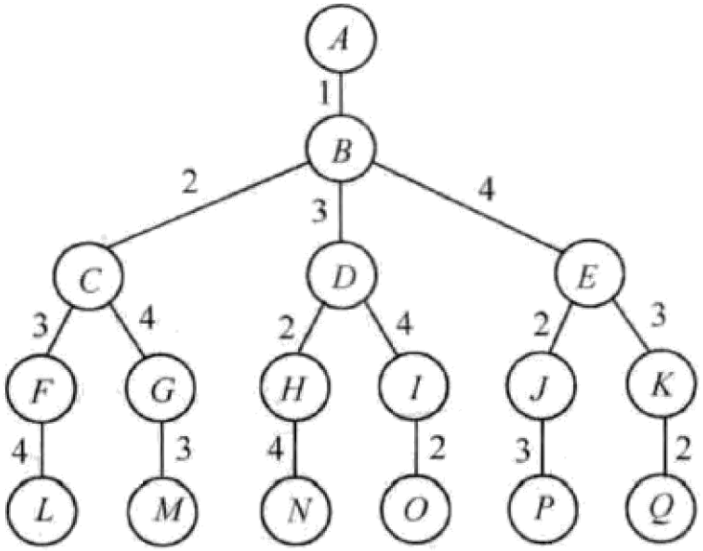


图 5-3　旅行售货员问题的解空间树

回溯法算法框架 (续)

■回溯法基本思想

➢采用剪枝函数避免无效搜索

- 约束函数在扩展结点处剪去不满足约束的子树
- 限界函数剪去得不到最优解的子树

■回溯法步骤：

1. 针对所给问题，定义问题的解空间
2. 确定易于搜索的解空间结构
3. 以深度优先方式搜索解空间，并在搜索过程中用剪枝函数避免无效搜索

回溯法算法框架 (续)

- 用回溯法解题的一个显著特征是在搜索过程中动态产生问题的解空间
  - 在任何时刻，算法只保存从根结点到当前扩展结点的路径
  - 如果解空间树中从根结点到叶结点的最长路径的长度为$h(n)$，则回溯法所需的计算空间通常为$O(h(n))$
  - 显式地存储整个解空间则需要$O(2^{h(n)})$或$O(h(n)!)$内存空间

回溯法算法框架 (续)

■递归回溯

```
BACKTRACK(x, t, n)  //n为总递归深度
1   if t > n
2       OUTPUT(x)   //x为可行解
3   elseif f(n, t) > g(n, t)  //当前无可搜索子树
4       return
5   else for i ← f(n, t) to g(n, t) do
6       x[t] ← h(i)
7       if CONSTRAINT(t) and BOUND(t)
8           BACKTRACK(x, t+1, n)
```

尾递归

- $f(n, t)$, $g(n, t)$：当前扩展结点处未搜索过的子树的起始编号和终止编号
- $h(i)$：当前扩展结点处$x[t]$的第i个可选值
- CONSTRAINT(t), BOUND(t)：当前扩展结点处的约束函数和限界函数

回溯法算法框架 (续)

■迭代回溯

```
ITERATIVE_BACKTRACK(n)
1   t ← 1
2   let x[1..n] be a new array
3   while t ≤ n do
4       if f(n, t) ≤ g(n, t)
5           for i ← f(n, t) to g(n, t) do
6               x[t] ← h(i)
7               if CONSTRAINT(t) and BOUND(t)
8                   t ← t + 1
9                   break
10      else t ← t - 1
11  OUTPUT(x)
```

回溯法算法框架 (续)

■子集树与排列树

- 当所给的问题是从n个元素的集合S中找出满足某种性质的子集时，相应的解空间树称为子集树
  - 结点数：2^n
  - 遍历时间：$\Omega(2^n)$
- 当所给的问题是确定n个元素满足某种性质的排列时，相应的解空间树称为排列树
  - 结点数：$n!$
  - 遍历时间：$\Omega(n!)$

回溯法求解举例——0-1背包问题

- 解空间：子集树
- 基本思想：在搜索解空间树时，只要其左儿子结点是一个可行结点，搜索就进入其左子树。当右子树中有可能包含最优解时才进入右子树搜索，否则将右子树剪去
- 可行性约束函数：背包容量限制 $\sum_{k=1}^{n} w_k x_k \leq W$
- 限界函数：剩余容量装入剩余物品的分数背包问题的最大价值

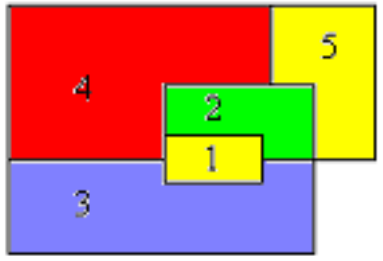
回溯法求解举例——0-1背包问题 (续)

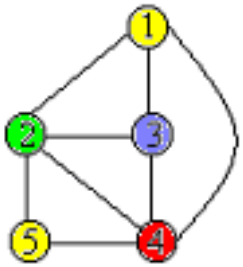
```
KNAPSACK_BACKTRACK(t, n, w, v, W, cw, cv)
1   if t > n
2       bestv ← cv
3       return
4   if cw + w[t] ≤ W   // 搜索左子树（装入当前物品）
5       cw ← cw + w[t]
6       cv ← cv + v[t]
7       KNAPSACK_BACKTRACK(t+1, n, w, v, W, cw, cv)
8       cw ← cw - w[t]
9       cv ← cv - v[t]
10  if BOUND(t+1) > bestv    // 搜索右子树（不装入当前物品）
11      KNAPSACK_BACKTRACK(t+1, n, w, v, W, cw, cv)
```

- BOUND：$O(n)$，最坏情况$O(2^n)$个结点都要计算BOUND函数
- 时间复杂度：$O(n2^n)$

回溯法求解举例——图的m着色问题

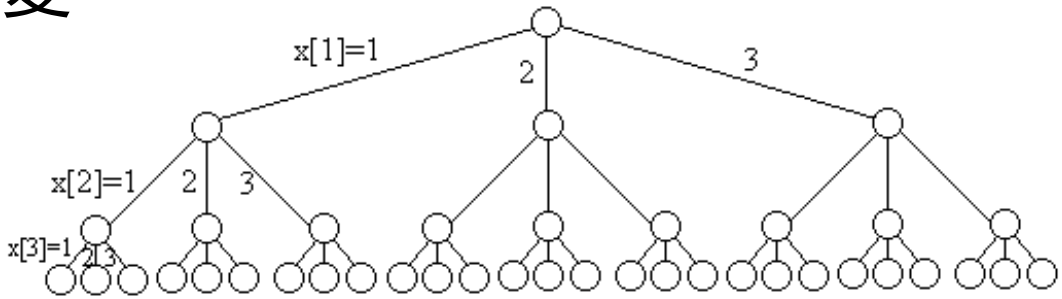
- 判定问题：给定无向连通图G和m种不同的颜色，用这些颜色为图G的各顶点着色，每个顶点着一种颜色，判断是否有一种着色法使G中每条边的2个顶点着不同颜色
- 最优化问题：若一个图最少需要m种颜色才能使图中每条边连接的2个顶点着不同颜色，则称这个数m为该图的色数。求一个图的色数m的问题称为图的m可着色优化问题
- 四色猜想





回溯法求解举例——图的m着色问题 (续)

- 解向量$x[1..n]$： $x[i]$表示顶点i所着颜色序号
- 可行性约束函数：顶点i与已着色的相邻顶点颜色不重复



- 时间复杂度：
  - 内结点个数：$\sum_{i=0}^{n-1} m^i$
  - 每个内结点检查子结点颜色可用性：$O(mn)$
  - 总耗时：$O(mn\sum_{i=0}^{n-1} m^i) = O(nm^n)$

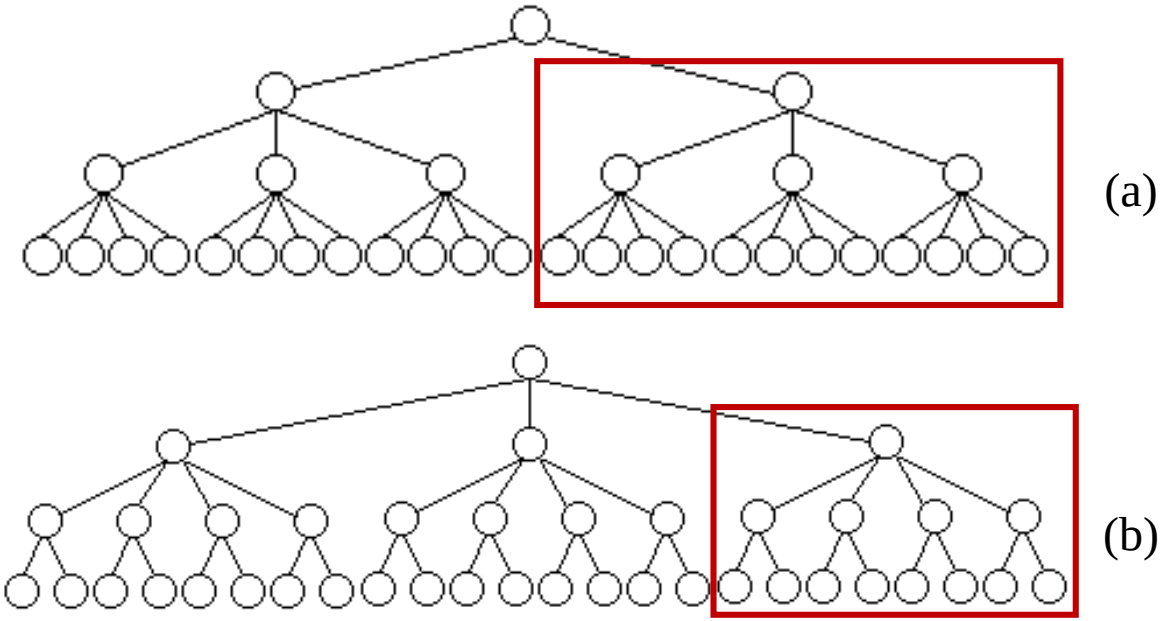
回溯法效率分析

- 回溯法效率依赖于：
  - 产生$x[k]$的时间
  - 满足显约束的$x[k]$值的个数
  - 计算约束函数CONSTRAINT的时间
  - 计算限界函数BOUND的时间
  - 满足约束函数和限界函数约束的所有$x[k]$的个数
- 好的约束函数能显著减少所生成的结点数，但这样的约束函数往往计算量较大

回溯法效率分析 (续)

■重排原理

➢对于许多问题而言，在搜索试探时选取$x[i]$的值顺序是任意的。在其它条件相当的前提下，让可取值最少的$x[i]$优先



第7章 分支限界法

苏州大学 计算机科学与技术学院
汪笑宇
Email: xywang21@suda.edu.cn

本章内容

- 分支限界法基本思想：队列式、优先队列式
- 实际问题：单源最短路径问题、装载问题、布线问题、0-1背包问题、最大团问题、旅行售货员问题、电路板排列问题、批处理作业调度问题

分支限界法 vs. 回溯法

- 求解目标：
  - 回溯法：找出解空间树中满足约束条件的所有解
  - 分支限界法：找出满足约束条件的一个解，或是在满足约束条件的解中找出在某种意义下的最优解
- 搜索方式：
  - 回溯法：深度优先的方式搜索解空间树
  - 分支限界法：广度优先或最小耗费优先的方式搜索解空间树

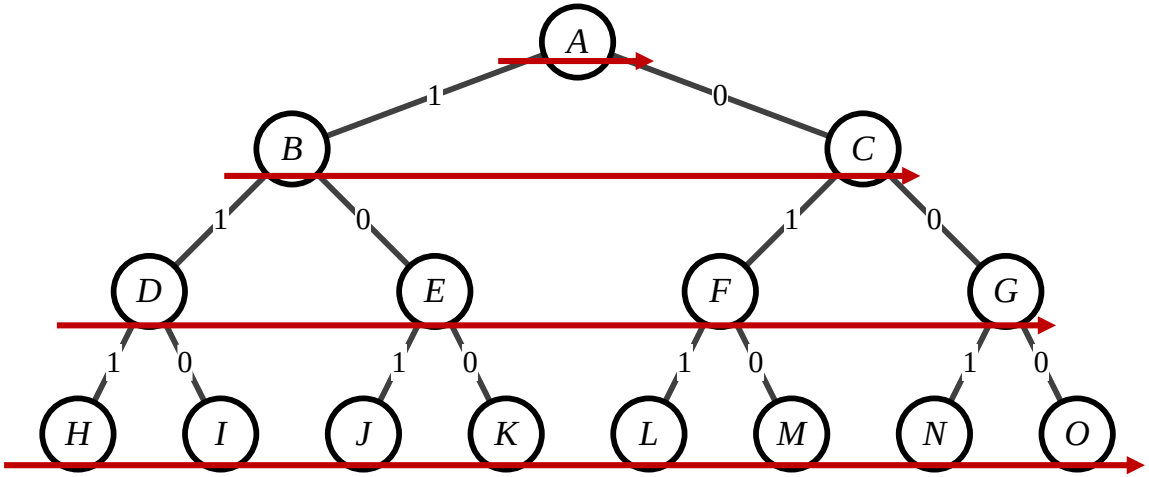
分支限界法基本思想

- 分支限界法常以广度优先或以最小耗费（最大效益）优先的方式搜索问题的解空间树
- 在分支限界法中，每一个活结点只有一次机会成为扩展结点。活结点一旦成为扩展结点，就一次性产生其所有子结点。在这些子结点中，导致不可行解或导致非最优解的子结点被舍弃，其余子结点被加入活结点表中
- 此后，从活结点表中取下一结点成为当前扩展结点，并重复上述结点扩展过程。这个过程一直持续到找到所需的解或活结点表为空时为止

分支限界法基本思想 (续)

■常见的两种分支限界法

- ➢队列式（FIFO）分支限界法：按照队列先进先出（FIFO）原则选取下一个节点为扩展节点
- ➢优先队列式分支限界法：按照优先队列中规定的优先级选取优先级最高的节点成为当前扩展节点



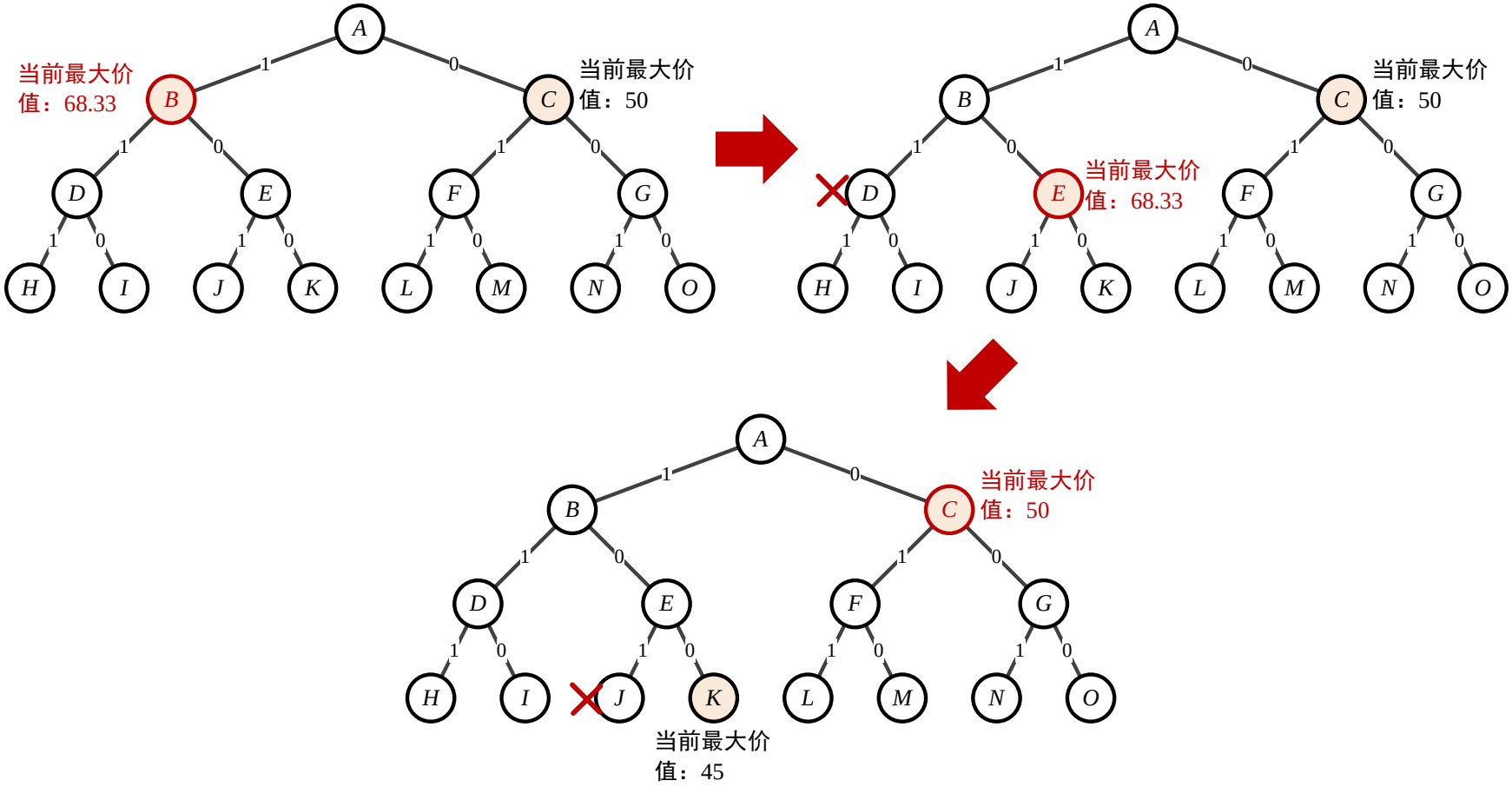
分支限界法求解举例——0-1背包问题

■算法思想：

- 将各物品依其单位重量价值从大到小进行排列
- 优先队列分支限界法：节点的优先级：已装包的物品价值 + 剩下的最大单位重量价值的物品装满剩余容量的价值和
- 首先检查当前扩展结点的左子结点的可行性
- 如果该左子结点是可行结点，则将它加入到子集树和活结点优先队列中
- 当前扩展结点的右子结点一定是可行结点，仅当右子结点满足上界约束时才将它加入子集树和活结点优先队列
- 当扩展到叶节点时为问题的最优值

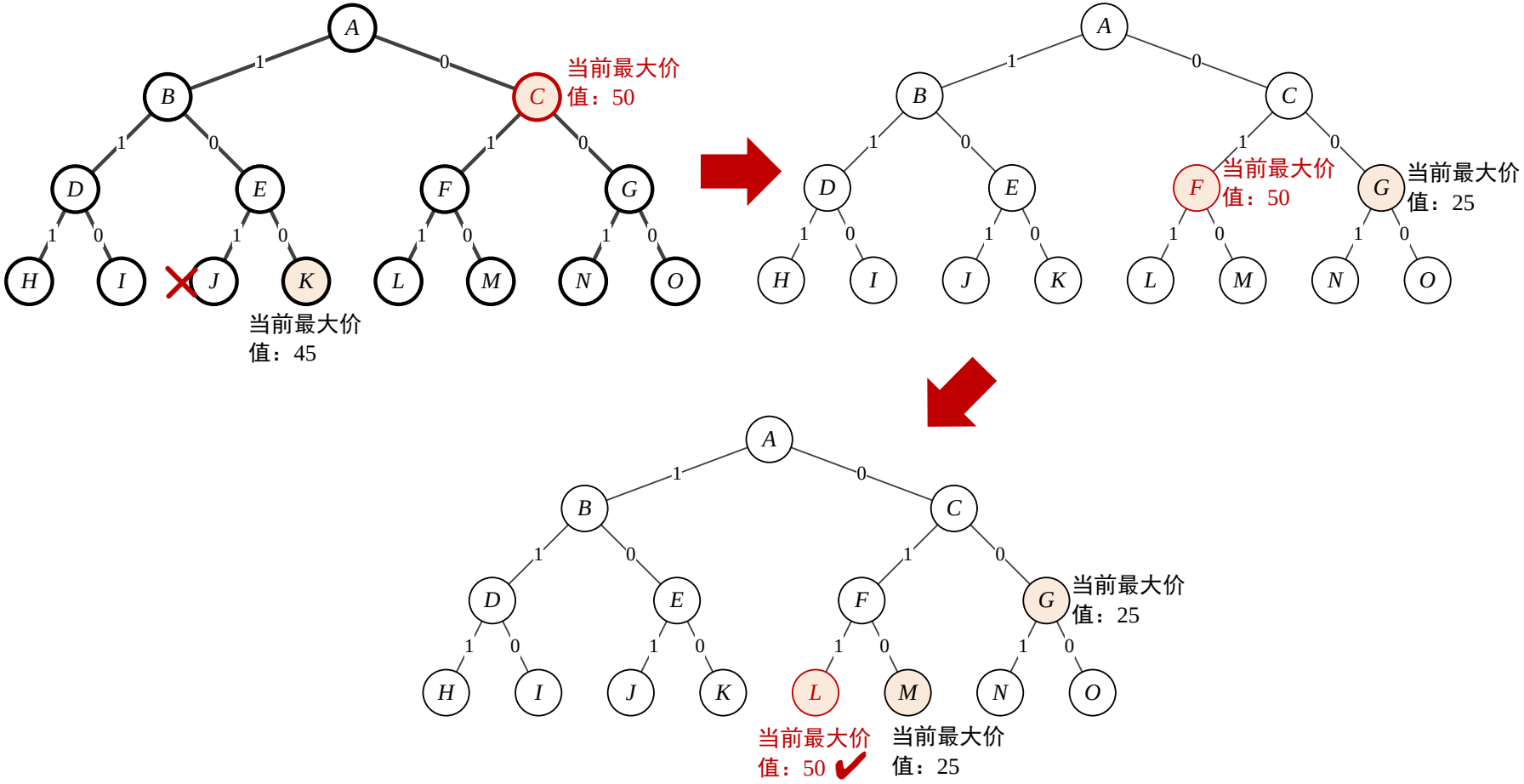
分支限界法求解举例——0-1背包问题(续)

■例：w=[16, 15, 15], v=[45, 25, 25], W=30, v/w≈[2.8125, 1.67, 1.67]



分支限界法求解举例——0-1背包问题(续)

■例：w=[16, 15, 15], v=[45, 25, 25], W=30, v/w≈[2.8125, 1.67, 1.67]



第8章 随机算法

苏州大学 计算机科学与技术学院

汪笑宇

Email: xywang21@suda.edu.cn

本章内容

- 随机化算法（参考书目Chapter 7）
- 概率分析和随机算法（教材Chapter 5）

本章目录

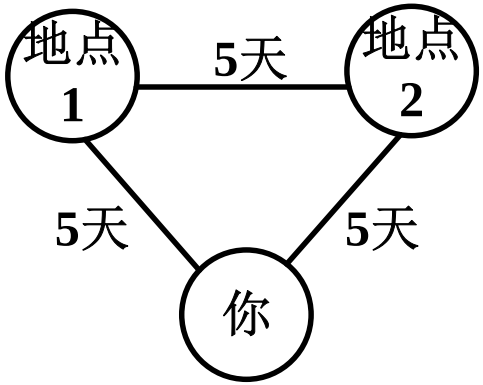
- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

本章目录

- ■随机算法概述
- ■概率分析相关知识
- ■数值随机化算法
- ■Sherwood算法
- ■Las Vegas算法
- ■Monte Carlo算法

引入

- 想象自己是神话故事的主人公，有一张不易懂的地图，上面描述了一处宝藏的藏宝地点。经分析你能确定最有可能的两个地点是藏宝地点，但二者相距甚远
- 假设你如果已到达其中一处，就立即知道该处是否为藏宝地点
- 你到达两处之一地点，以及从其中一处到另一处的距离是5天的行程
- 进一步假设有一条恶龙，每晚光顾宝藏并从中拿走一部分财宝
- 假设你取宝藏的方案有两种：
  - 方案1：花4天的时间计算出准确的藏宝地点，然后出发寻宝，一旦出发不能重新计算
  - 方案2：有一个小精灵告诉你地图的秘密，但你必须付给他报酬，相当于恶龙三晚拿走的财宝



假设宝藏价值x
恶龙一天取走y
方案1：x-9y
方案2：x-8y

引入 (续)

- 是否有其他方案？
  - 方案3：投硬币决定先到一处，失败后到另一处（冒险方案）
    - 一次成功所得：x-5y， 机会1/2
    - 二次成功所得：x-10y，机会1/2
    - 期望盈利：x-7.5y
- 当一个算法面临某种选择时，有时随机选择比耗时做最优选择更好，尤其是当最优选择所花的时间大于随机选择的平均时间的时候

随机算法只是期望的时间更有效，
但有可能遭受到最坏的可能性

随机算法概述

- 随机化算法允许算法在执行过程中随机地选择下一个计算步骤
- 在许多情况下， 当算法在执行过程中面临一个选择时， 随机性选择常比最优选择省时，因此，随机化算法可在很大程度上降低算法的复杂度
- 随机化算法的一个基本特征是对所求解问题的同一实例用同一随机化算法求解两次可能得到完全不同的效果，这两次求解所需的时间甚至得到的结果可能有相当大的差别

期望时间 vs. 平均时间

- 确定性算法的平均执行时间
  - 输入规模一定的所有输入实例是等概率出现时，算法的平均执行时间
- 随机算法的期望执行时间
  - 反复解同一个输入实例所花的平均执行时间
- 随机算法可讨论如下两种期望时间：
  - 平均的期望时间：所有输入实例上平均的期望执行时间
  - 最坏的期望时间：最坏的输入实例上的期望执行时间

随机算法特点

- 不可再现性
  - 在同一个输入实例上，每次执行结果不尽相同
    - n皇后问题：随机算法运行不同次将会找到不同的正确解
    - 找一给定合数的非平凡因子：每次运行的结果不尽相同，但确定性算法每次运行结果必定相同
- 分析困难
  - 要求有概率论、统计学和数论的知识

随机算法分类

■随机算法分类：

- 数值随机化算法常用于数值问题的求解，得到的往往是近似解， 且近似解的精度随计算时间的增加而不断提高
- 蒙特卡罗算法用于求问题的准确解，例如判定问题，其解为"是"或"否"， 二者必居其一，不存在任何近似解答。用蒙特卡罗算法能求得问题的一个解，但这个解未必是正确的
- 拉斯维加斯算法不会得到不正确的解。一旦用拉斯维加斯算法找到一个解，这个解就一定是正确解， 但有时找不到解
- 舍伍德算法总能求得问题的一个解，且求得的解总是正确的。当一个确定性算法在最坏情况下的计算复杂性与其在平均情况下的计算复杂性有较大差别时，可在这个确定性算法中引入随机性将它改造成一个舍伍德算法，消除或减少问题的好坏实例间的这种差别

本章目录

- 随机算法概述
- 概率分析相关知识
- 数值随机化算法
- Sherwood算法
- Las Vegas算法
- Monte Carlo算法

生活中的概率问题——购买盲盒

■购买盲盒

- 一套一共b款
- 每款买到的概率是$1/b$（假设商家很良心）

平均意义上需要买多少个盲盒才能集齐全套呢？

生活中的概率问题——购买盲盒 (续)

- 假设我们一次购买买到一个之前没买到的新的款式，则称作一次命中（hit）
  问题转化为：求有b次命中时，购买次数n的期望
- 将购买次数n分成若干个阶段，第i阶段表示从第$i-1$次命中到第i次命中之间的购买，购买次数为n_i，服从几何分布
- 第i阶段得到一次命中的概率：
  未买到的款式个数/总款式个数 $= (b - i + 1)/b$

几何分布：一系列伯努利试验，其中每次成功概率为p、失败概率为$1-p$，在获得一次成功前要进行的试验次数服从几何分布
几何分布期望为$1/p$

生活中的概率问题——购买盲盒 (续)

■第i阶段：

已有

购买1
重复

购买2
重复

购买3
重复

购买4
重复

购买5
重复

…

购买n_i
新款！

➢n_i服从$p = (b - i + 1)/b$的几何分布

生活中的概率问题——购买盲盒 (续)

■第i阶段购买次数为n_i，则得到b次命中的购买次数$n = \sum_{i=1}^{b} n_i$

$$E[n_i] = \frac{b}{b-i+1}$$

$$E[n] = E\left[\sum_{i=1}^{b} n_i\right] = \sum_{i=1}^{b} E[n_i] = \sum_{i=1}^{b} \frac{b}{b-i+1}$$

$$= b\sum_{i=1}^{b} \frac{1}{i} = b(\ln b + O(1))$$

大概需要购买$b\ln b$次才能集齐所有盲盒
（前提是各款式等概率出现）
假设b=30，则购买次数≈103