期末复习

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

渐近表示法

■$f(n)=\Theta(g(n))$ 渐近紧确界

$\Theta(g(n))$ = {$f(n)$: 存在正常量c_1,c_2,n_0, 使得对所有$n \geq n_0$有$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$}

■$f(n)=O(g(n))$ 渐近上界

$O(g(n))$ = {$f(n)$: 存在正常量c,n_0, 使得对所有$n \geq n_0$有$0 \leq f(n) \leq cg(n)$}

■$f(n)=\Omega(g(n))$ 渐近下界

$\Omega(g(n))$ = {$f(n)$: 存在正常量c,n_0, 使得对所有$n \geq n_0$有$0 \leq cg(n) \leq f(n)$}

渐近表示法 (续)

■$f(n)=o(g(n))$ 渐近非紧上界

$o(g(n))$ = {$f(n)$: 对任意常数$c>0$, 存在常数$n_0 > 0$,
使得对所有$n \geq n_0$有$0 \leq f(n) < cg(n)$}

■$f(n)=\omega(g(n))$ 渐近非紧下界

$\omega(g(n))$ = {$f(n)$: 对任意常数$c>0$, 存在常数$n_0 > 0$,
使得对所有$n \geq n_0$有$0 \leq cg(n) < f(n)$}

■ $$\lim_{n\to\infty}\frac{f(n)}{g(n)}=\begin{cases}0 & f(n)=o(g(n)), f(n)=O(g(n))\\ a, a>0 & f(n)=\Theta(g(n)), f(n)=O(g(n)), f(n)=\Omega(g(n))\\ \infty & f(n)=\omega(g(n)), f(n)=\Omega(g(n))\end{cases}$$

递归式求解——主方法

■定理4.1（教材p53-54，主定理） 令$a \geq 1$和$b>1$是常数，$f(n)$是一个函数，$T(n)$是定义在非负整数上的递归式：

$$T(n) = aT(n/b) + f(n)$$

其中我们将n/b解释为$\lceil n/b \rceil$或$\lfloor n/b \rfloor$，有如下渐近界：

1. 若对某个常数$\varepsilon>0$有$f(n) = O(n^{\log_b a - \varepsilon})$，则$T(n) = \Theta(n^{\log_b a})$
2. 若$f(n) = \Theta(n^{\log_b a})$，则$T(n) = \Theta(n^{\log_b a} \lg n)$
3. 若对某个常数$\varepsilon>0$有$f(n) = \Omega(n^{\log_b a + \varepsilon})$，且对某个常数$c<1$和所有足够大的$n$有$af(n/b) \leq cf(n)$，则$T(n) = \Theta(f(n))$

递归式求解——主方法 (续)

- 定理意义：比较$f(n)$和$n^{\log_b a}$，直观上两函数较大者决定$T(n)$
  1. $n^{\log_b a}$比$f(n)$大一个多项式因子n^{ε}：$T(n) = \Theta(n^{\log_b a})$
  2. 两者相同，乘以对数因子$\lg n$：
     $T(n) = \Theta(n^{\log_b a} \lg n) = \Theta(\lg n f(n))$
  3. $f(n)$比$n^{\log_b a}$大一个多项式因子n^{ε}，以及满足“正则”条件：$T(n) = \Theta(f(n))$
- 注：三种情况并未覆盖所有可能的$f(n)$，存在间隙

分治法求解——二分搜索

```
BINARY_SEARCH(A, x, low, high)
1  if low ≤ high
2      mid ← ⌊(low + high)/2⌋
3      if x = A[mid]
4          return mid
5      if x < A[mid]
6          return BINARY_SEARCH(A, x, low, mid-1)
7      else return BINARY_SEARCH(A, x, mid+1, high)
8  return 0
```

- 最坏情况运行时间：$T(n)=T(n/2)+\Theta(1)$，由主方法易得$T(n)=\Theta(\lg n)$

引入1

- 假设有100个砝码外观完全相同，已知其中有一个是劣质的质量稍轻，而其他99个质量相同，如何用一个天平快速得出哪一个是劣质的砝码？
  - 想法一：随机挑一个砝码作为基准，放在天平一侧，剩下的砝码依次放在天平另一侧进行称重
    - 最坏情况：前98次称重天平都平衡
    - 最好情况：第1次称重就天平不平衡

引入1 (续)

想法二：

- 第1次：天平一边50个，重的那一侧排除
- 第2次：天平一边25个，重的那一侧排除
- 第3次：天平一边12个，剩下1个，一样重则剩下的一个劣质；不一样重则重的一侧和剩下的一个排除
- 第4次：天平一边6个，重的那一侧排除
- 第5次：天平一边3个，重的那一侧排除
- 第6次：天平一边1个，剩下1个，一样重则剩下的一个劣质；否则哪边轻哪边劣质

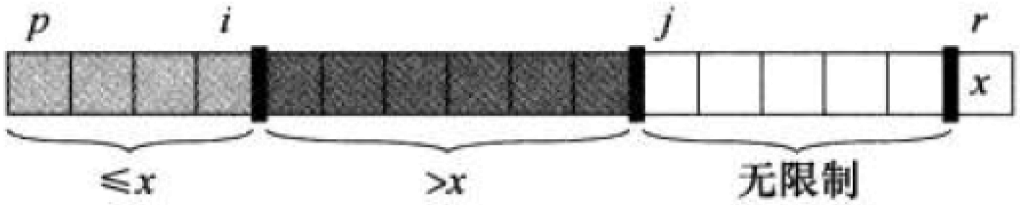
二分法

$\lfloor \log_2 100 \rfloor = 6$

分治法求解——快速排序 (续)

■算法描述

```
QUICKSORT(A, p, r)
1  if p < r
2      q ← PARTITION(A, p, r)
3      QUICKSORT(A, p, q-1)
4      QUICKSORT(A, q+1, r)
```




```
PARTITION(A, p, r)
1  x ← A[r]  //划分元/主元(pivot element)
2  i ← p - 1
3  for j ← p to r-1 do
4      if A[j] ≤ x
5          i ← i + 1,  exchange A[i] with A[j]
6  exchange A[i+1] with A[r]
7  return i+1
```

循环不变式：

1. $A[p..i] \le x$
2. $A[i+1..j-1] > x$
3. $A[r] = x$

该划分使得：

$A[p..q-1] \le A[q] < A[q+1..r]$

分治法求解——快速排序 (续)

■随机化版本（教材p100）

- 快速排序的平均性能假定：输入的所有排列是等可能的
- 算法随机化是指：算法行为不仅由输入确定，而且与随机数发生器产生的值有关，强迫输入分布是随机的

```
RANDOMIZED_PARTITION(A, p, r)
1  i ← RANDOM(p, r)
2  exchange A[r] with A[i]
3  return PARTITION(A, p, r)
```

- 分析较困难
- 算法非常有效，排序过程中，某次随机选择最坏不会影响总体效果

Sherwood算法——快速排序

■例：快速排序

- ➢平均时间复杂度$O(n\lg n)$
- ➢有序数列排序：时间复杂度$O(n^2)$

初始序列：30 25 19 12 6 ⟶ 一次划分：[6 25 19 12] 30 []

划分不均衡

- ➢如何提升最坏情况性能？
  - 想法一：每次随机选择划分元 (pivot)

初始序列：30 25 19 12 6 ➡ 位置调整：19 25 30 12 6 ➡ 一次划分：[6 12] 19 [25 30]

随机选择

划分较均衡

  - 想法二：把初始序列打乱

初始序列：30 25 19 12 6 ➡ 打乱顺序：12 30 6 19 25 ➡ 一次划分：[6] 12 [30 19 25]

划分较均衡

矩阵链乘法

- 给定n个矩阵的序列（矩阵链）$<A_1, A_2, ..., A_n>$，计算乘积$A_1A_2 \cdots A_n$
- 计算多个矩阵连乘积可用括号来决定计算次序，每一个括号内的矩阵相乘调用标准的矩阵乘法
- 矩阵积的完全括号化
  - 它是单一矩阵
  - 或者是两个完全括号化的矩阵链的积

递归定义
计算次序无二义性

$(A_1(A_2(A_3A_4)))$　　$((A_1(A_2A_3))A_4)$

$(A_1((A_2A_3)A_4))$　　$(((A_1A_2)A_3)A_4)$

$((A_1A_2)(A_3A_4))$

矩阵链乘法 (续)

■不同的括号化方式产生不同的计算成本

```
MATRIX_MULTIPLY(A, B)
1  if A.columns ≠ B.rows
2     error "incompatible dimensions"
3  else let C be a new A.rows×B.columns matrix
4     for i ← 1 to A.rows do
5        for j ← 1 to B.columns do
6           c_ij ← 0
7           for k ← 1 to A.columns do
8              c_ij ← c_ij + a_ik·b_kj
9  return C
```

- 第8行执行次数：A.*rows*×B.*columns*×A.*columns*(或B.*rows*)
- 设A是$p\times q$矩阵、B是$q\times r$矩阵，则计算C=A·B共需pqr次标量乘法

矩阵链乘法 (续)

■矩阵链乘法问题实质上是一个最优括号化问题：

- 给定n个矩阵的链$<A_1, A_2, ..., A_n>$，矩阵A_i的规模为$p_{i-1} \times p_i$ $(1 \le i \le n)$，求完全括号化方案，使得计算乘积$A_1 A_2 \cdots A_n$所需标量乘法次数最少
- 计算括号化方案数量：设$P(n)$表示一个n个矩阵的链中可选括号化方案数量，则穷举法产生数量：

$$P(n) = \begin{cases} 1, & n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k), & n \ge 2. \end{cases}$$

Catalan数，指数阶：$\Omega(4^n/n^{1.5})$，不如直接求解矩阵乘积！

矩阵链乘法 (续)

■$m[i, j]$：计算$A_{i..j}$所需标量乘法次数的最小值

- 若$i<j$，利用步骤1最优子结构计算代价（k是最优分割点）：
  $A_{i..k}$计算代价 + $A_{k+1..j}$计算代价 + 两者乘积计算代价
  即：$m[i, j] = m[i, k] + m[k+1, j] + p_{i-1}p_k p_j$
- 以上公式成立需要k是最优分割点
- 检查所有$j-i$种可能的k即可：

$$m[i,j] = \begin{cases} 0, & i = j, \\ \min\limits_{i \le k < j} \{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\}, & i < j. \end{cases}$$

- 定义$s[i, j]$保存$A_{i..j}$最优括号化方案分割点位置k

矩阵链乘法 (续)

■例：计算矩阵链<A_1, A_2, A_3, A_4>乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$

- $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
- $i < j$：$m[1, 2]$：k=1时为$m[1, 1] + m[2, 2] + p_0p_1p_2$

  $m[2, 3]$：k=2时为$m[2, 2] + m[3, 3] + p_1p_2p_3$

  $m[3, 4]$：k=3时为$m[3, 3] + m[4, 4] + p_2p_3p_4$

  $m[1, 3]$：k=1时为$m[1, 1] + m[2, 3] + p_0p_1p_3$

  k=2时为$m[1, 2] + m[3, 3] + p_0p_2p_3$

  $m[2, 4]$：k=2时为$m[2, 2] + m[3, 4] + p_1p_2p_4$

  k=3时为$m[2, 3] + m[4, 4] + p_1p_3p_4$

矩阵链乘法 (续)

■例：计算矩阵链<A_1, A_2, A_3, A_4>乘积。以下计算所有$m[i, j]$的值，其中$1 \le i \le j \le n$

- $i = j$：$m[1, 1] = m[2, 2] = m[3, 3] = m[4, 4] = 0$
- $i < j$：$m[1, 2], m[2, 3], m[3, 4]$

  $m[1, 3], m[2, 4]$

  $m[1, 4]$：k=1时为$m[1, 1] + m[2, 4] + p_0p_1p_4$

  k=2时为$m[1, 2] + m[3, 4] + p_0p_2p_4$

  k=3时为$m[1, 3] + m[4, 4] + p_0p_3p_4$

矩阵链乘法 (续)

```
MATRIX_CHAIN_ORDER(p)
1   n ← p.length - 1
2   let m[1..n, 1..n] and s[1..n-1, 2..n] be new tables
3   for i ← 1 to n do
4       m[i, i] ← 0    // i = j
5   for l ← 2 to n do  // l：矩阵链长度，i ≠ j 时 1 ≤ j-i = l-1 ≤ n-1
6       for i ← 1 to n-l+1 do
7           j ← i + l - 1
8           m[i, j] ← ∞
9           for k ← i to j-1 do
10              q ← m[i, k] + m[k+1, j] + p_{i-1} p_k p_j
11              if q < m[i, j]
12                  m[i, j] ← q
13                  s[i, j] ← k
14  return m and s
```

矩阵链乘法 (续)

- 时间复杂度：$O(n^3)$，三层循环
- 空间复杂度：$O(n^2)$，保存表m和s
- 较穷举方法指数阶高效得多

矩阵链乘法 (续)

4. 构造最优解

- MATRIX_CHAIN_ORDER求出了计算矩阵链乘积所需的最少标量乘法次数，但并未指出如何进行这种最优代价矩阵链乘法计算
- 表s记录了构造最优解的最优分割信息，可递归求出其中最外层划分位置：$k = s[1, n]$，则进一步求$s[1, k]$和$s[k+1, n]$，直到$s[i, j]$中$i=j$为止

```
PRINT_OPTIMAL_PARENS(s, i, j)
1  if i = j
2      print "A"_i
3  else print "("
4      PRINT_OPTIMAL_PARENS(s, i, s[i, j])
5      PRINT_OPTIMAL_PARENS(s, s[i, j]+1, j)
6      print ")"
```

矩阵链乘法 (续)

■按照最优括号化计算矩阵链乘积

```
MATRIX_CHAIN_MULTIPLY(A, s, i, j)
1  if i = j
2      return A_i
3  else
4      X ← MATRIX_CHAIN_MULTIPLY(A, s, i, s[i, j])
5      Y ← MATRIX_CHAIN_MULTIPLY(A, s, s[i, j]+1, j)
6      return MATRIX_MULTIPLY(X, Y)
```

最长公共子序列LCS

- 子序列：将给定序列中零个或多个元素去掉之后得到的结果
  - 形式化定义：给定一个序列$X=<x_1, x_2, ..., x_m>$，另一个序列$Z=<z_1, z_2, ..., z_k>$满足如下条件时称为X的子序列：存在一个严格递增的X的下标序列$<i_1, i_2, ..., i_k>$，对所有$j=1, 2, ..., k$，满足$x_{i_j}=z_j$
  - 例：X=<A, B, C, B, D, A, B>
    Z=< B, C, D, B>为X的子序列，对应下标序列为<2, 3, 5, 7>

子序列不一定是由原序列连续元素构成的序列！

最长公共子序列LCS (续)

- 公共子序列 (common subsequence)：给定两个序列X和Y，如果Z既是X的子序列，也是Y的子序列，则称Z是X和Y的公共子序列
- 最长公共子序列问题 (longest-common-subsequence problem)：求两个序列公共子序列中最长的一个
- 求解两个给定序列的LCS
  1. 刻画LCS结构特征
  2. 递归解
  3. 计算LCS长度
  4. 构造LCS

最长公共子序列LCS (续)

1. 刻画LCS特征

- 穷举法：穷举X的所有子序列，检查是否也是Y的子序列。若$|X|=m$，则子序列共2^m个，穷举法为指数阶下界
- LCS具有最优子结构性质
  前缀：给定一个序列$X=<x_1, x_2, ..., x_m>$，对$i=0, 1, ..., m$，定义X的第i前缀为$X_i=<x_1, x_2, ..., x_i>$
- **定理15.1** 令$X=<x_1, x_2, ..., x_m>$和$Y=<y_1, y_2, ..., y_n>$为两个序列，$Z=<z_1, z_2, ..., z_k>$为X和Y的任意LCS
  1. 若$x_m=y_n$，则$z_k=x_m=y_n$且Z_{k-1}是X_{m-1}和Y_{n-1}的一个LCS；
  2. 若$x_m\neq y_n$，则$z_k\neq x_m$意味着Z是X_{m-1}和Y的一个LCS；
  3. 若$x_m\neq y_n$，则$z_k\neq y_n$意味着Z是X和Y_{n-1}的一个LCS。

最长公共子序列LCS (续)

2. 递归解

- 由定理15.1：求$X=<x_1, x_2, ..., x_m>$和$Y=<y_1, y_2, ..., y_n>$的一个LCS时，需求解一个或两个子问题：
  - 若$x_m=y_n$，则求解X_{m-1}和Y_{n-1}的一个LCS，将$x_m=y_n$追加到这个LCS的末尾
  - 若$x_m \neq y_n$，(1) 求解X_{m-1}和Y的LCS；(2) 求解X和Y_{n-1}的LCS。求两者长度最大者

最长公共子序列LCS (续)

2. 递归解

➢$c[i, j]$：X_i和Y_j的LCS长度（$0 \le i \le m, 0 \le j \le n$）

$$c[i,j] = \begin{cases} 0, & i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1, & i, j > 0 \text{ and } x_i = y_j, \\ \max(c[i, j-1], c[i-1, j]), & i, j > 0 \text{ and } x_i \neq y_j. \end{cases}$$

➢限定了需求解哪些子问题，并非所有子问题都要求解

最长公共子序列LCS (续)

3. 计算LCS长度

- 不同子问题个数：$\Theta(mn)$
- 输入：序列$X=<x_1, x_2, ..., x_m>$和$Y=<y_1, y_2, ..., y_n>$
- 输出：$c[0..m, 0..n]$：按行主次序计算LCS长度
  $b[1..m, 1..n]$：辅助构造最优解子序列

$$b[i,j]=\begin{cases}\nwarrow, & c[i,j]=c[i-1,j-1]+1,\\ \uparrow, & c[i,j]=c[i-1,j],\\ \leftarrow, & c[i,j]=c[i,j-1].\end{cases}$$

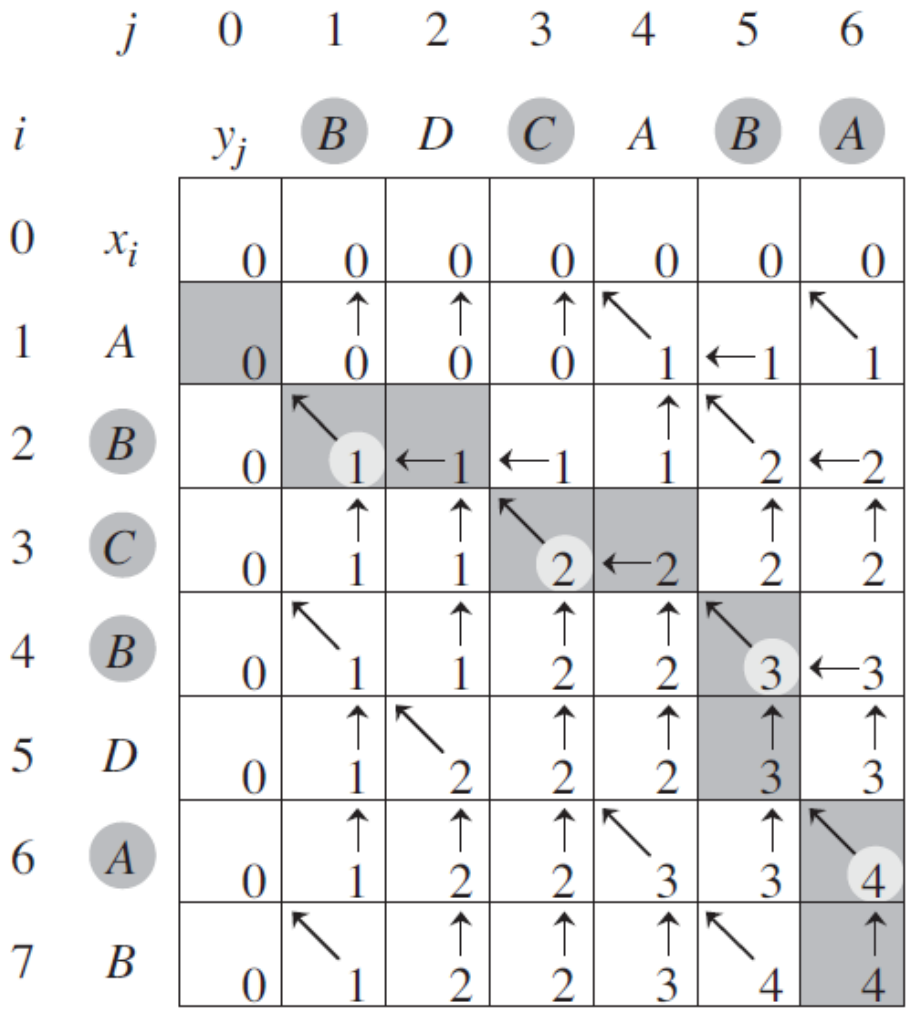
- 构造解时，从$b[m, n]$出发，根据箭头方向上溯至$i=0$或$j=0$为止，当$b[i, j]$包含“↖”时打印出x_i即可

最长公共子序列LCS (续)

```
LCS_LENGTH(X, Y)
1   m ← X.length
2   n ← Y.length
3   let b[1..m, 1..n] and c[0..m, 0..n] be new tables
4   for i ← 1 to m do
5       c[i, 0] ← 0
6   for j ← 0 to n do
7       c[0, j] ← 0
8   for i ← 1 to m do      // 依次考虑X1, X2, ..., Xm的前缀子列
9       for j ← 1 to n do  // 依次考虑Y1, Y2, ..., Yn的前缀子列
10          if xi = yj          // 两个前缀子列的最后一位相同
11              c[i, j] ← c[i-1, j-1] + 1
12              b[i, j] ← "↖"
13          elseif c[i-1, j] ≥ c[i, j-1] // 两个前缀子列的最后一位不同
14              c[i, j] ← c[i-1, j]
15              b[i, j] ← "↑"
16          else c[i, j] ← c[i, j-1]
17              b[i, j] ← "←"
18  return c and b
```

$\Theta(mn)$

最长公共子序列LCS (续)



i		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
0	x_i		0	0	0	0	0	0	0
1	A		0	↑ 0	↑ 0	↑ 0	↖ 1	← 1	↖ 1
2	B		0	↖ 1	← 1	← 1	↑ 1	↖ 2	← 2
3	C		0	↑ 1	↑ 1	↖ 2	← 2	↑ 2	↑ 2
4	B		0	↖ 1	↑ 1	↑ 2	↑ 2	↖ 3	← 3
5	D		0	↑ 1	↖ 2	↑ 2	↑ 2	↑ 3	↑ 3
6	A		0	↑ 1	↑ 2	↑ 2	↖ 3	↑ 3	↖ 4
7	B		0	↖ 1	↑ 2	↑ 2	↑ 3	↖ 4	↑ 4

X=<A, B, C, B, D, A, B>

Y=<B, D, C, A, B, A>

最长公共子序列LCS (续)

4. 构造LCS

➢从$b[m, n]$开始根据箭头上溯至$i=0$或$j=0$即可

- 当$b[i, j]$= “↖”时，有$x_i=y_j$是LCS的一个元素
- 逆序构造出LCS，可用递归算法顺序打印

```
PRINT_LCS(b, X, i, j)
1  if i = 0 or j = 0
2      return
3  if b[i, j] = "↖"
4      PRINT_LCS(b, X, i-1, j-1)
5      print x_i
6  elseif b[i, j] = "↑"
7      PRINT_LCS(b, X, i-1, j)
8  else PRINT_LCS(b, X, i, j-1)
```

$O(m+n)$

每次递归调用
i和j至少一个会减少1

贪心算法——活动选择问题

3. 贪心算法

- 直观上，我们应该选择这样一个活动，选出它后剩下的资源应能被尽量多的其他任务所用
- 每次选择候选集中最早结束的活动
- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k: a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集

活动选择问题 (续)

3. 贪心算法

- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k : a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集
- 证明：（第2部分，反证法）假定S_{im}的解非空，则存在$a_k \in S_{im}$，使得$f_i \le s_k < f_k \le s_m$。由此得到$a_k \in S_{ij}$的完成时间先于$a_m$，与$a_m$是$S_{ij}$最早完成的活动矛盾

活动选择问题 (续)

3. 贪心算法

- **定理16.1** 考虑任意非空子问题S_{ij}，令a_m是S_{ij}中结束时间最早的活动，即：$f_m = \min\{f_k: a_k \in S_{ij}\}$，则
  1. a_m在S_{ij}的某个最大兼容活动子集中
  2. 子问题S_{im}的解是空集
- 证明：（第1部分）设A_{ij}是S_{ij}的某个最优解，假设A_{ij}中的活动已按完成时间单调递增排序，且a_k是A_{ij}中最早结束的活动：

  1、若$a_k = a_m$，则问题已得证，即最优解包含a_m；

  2、若$a_k \neq a_m$，构造子集$A'_{ij} = (A_{ij} - \{a_k\}) \cup \{a_m\}$，即将最优解中的$a_k$替换为$a_m$，则需证明$A'_{ij}$也是最优解

  因为$f_m \leq f_k$，因此A'_{ij}中的活动也不冲突，且$|A'_{ij}| = |A_{ij}|$

  A'_{ij}也是S_{ij}的一个最优解，包含a_m

活动选择问题 (续)

3. 贪心算法

- 动态规划求解时，原问题S_{ij}可分解为两个子问题S_{ik}和S_{kj}求解，且这种分解有$|S_{ij}|$种可能
- 定理16.1可简化问题求解过程：
  - 求S_{ij}最优解时只用到一个子问题，另一个子问题为空
  - 只需考虑一种选择，即选择S_{ij}中最早完成的活动
- 定理16.1可以自顶向下的方式解每一个子问题

活动选择问题 (续)

3. 贪心算法

- 当某个a_m加入解集合后，我们总是在剩余活动中选择第一个不与a_m冲突的活动加入解集，该活动是能够最早完成且与a_m兼容的
- 这种选择为剩余活动的调度留下了尽可能多的机会，即：留出尽可能多的时间给剩余的尚未调度的活动，以使解集合中包含的活动最多

每次选一个最早完成并与刚加入解集元素兼容的活动

活动选择问题 (续)

3. 贪心算法

➢例：

i	1	2	3	4	5	6	7	8	9	10	11
s_i	1	3	0	5	3	5	6	8	8	2	12
f_i	4	5	6	7	8	9	10	11	12	13	14

1. 选择a_1，则下一活动开始时间须大于等于4，排除a_2, a_3
2. 选择a_4，则下一活动开始时间须大于等于7，排除a_5, a_6, a_7
3. 选择a_8，则下一活动开始时间须大于等于11，排除a_9, a_{10}
4. 选择a_{11}，则下一活动开始时间须大于等于14
5. 无可选活动，结束，解集为$\{a_1, a_4, a_8, a_{11}\}$

活动选择问题 (续)

5. 迭代贪心算法

- RECURSIVE_ACTIVITY_SELECTOR几乎就是尾递归：以一个对自身的递归调用再接一次并集操作结尾
- 尾递归过程改为迭代形式通常很直接，某些特定语言的编译器可以自动完成这一工作

```
GREEDY_ACTIVITY_SELECTOR(s, f)
1   n ← s.length
2   A ← {a_1}
3   k ← 1
4   for m ← 2 to n do
5       if s[m] ≥ f[k]
6           A ← A ∪ {a_m}
7           k ← m
8   return A
```

时间复杂度：$\Theta(n)$
（已排序情况）

排序：$\Theta(n\lg n)$

算法正确性证明？
循环不变式及证明→定理16.1证明→算法正确

近似算法概述

- 许多具有实际意义的问题都是NPC问题：P≠NP时无法在多项式时间内求最优解
- 解决NPC问题方法：
  - 输入规模小：指数级运行时间算法解决
  - 多项式时间内解决特殊情况
  - 多项式时间内得到近似最优解
- 近似算法：返回近似最优解的算法

近似算法概述 (续)

■最优化问题Π（最小化问题/最大化问题）：

➢算法A的近似比α：

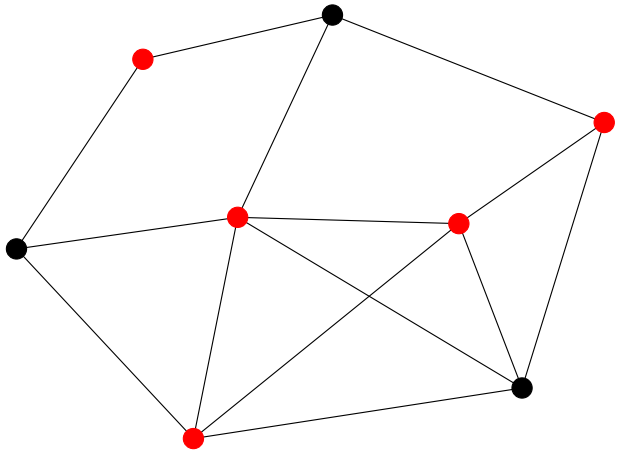
$$\forall I \in D: \begin{cases} 1 \leq \dfrac{\mathrm{SOL}_A(I)}{\mathrm{OPT}(I)} \leq \alpha, & \text{Minimization} \\ \alpha \leq \dfrac{\mathrm{SOL}_A(I)}{\mathrm{OPT}(I)} \leq 1, & \text{Maximization} \end{cases}$$

近似比限制了算法A所能达到的最坏情况

顶点覆盖判定问题

■顶点覆盖问题VERTEX-COVER

- 给定一个无向图$G=(V, E)$和一个正整数k，判定是否存在$V' \subseteq V$和$|V'|=k$，使得对任意$(u,v) \in E$有$u \in V'$或$v \in V'$，如果存在，就称V'为图G的一个大小为k的顶点覆盖
- 即E中每条边至少有一个顶点在V'中



存在一个规模k为5的顶点覆盖

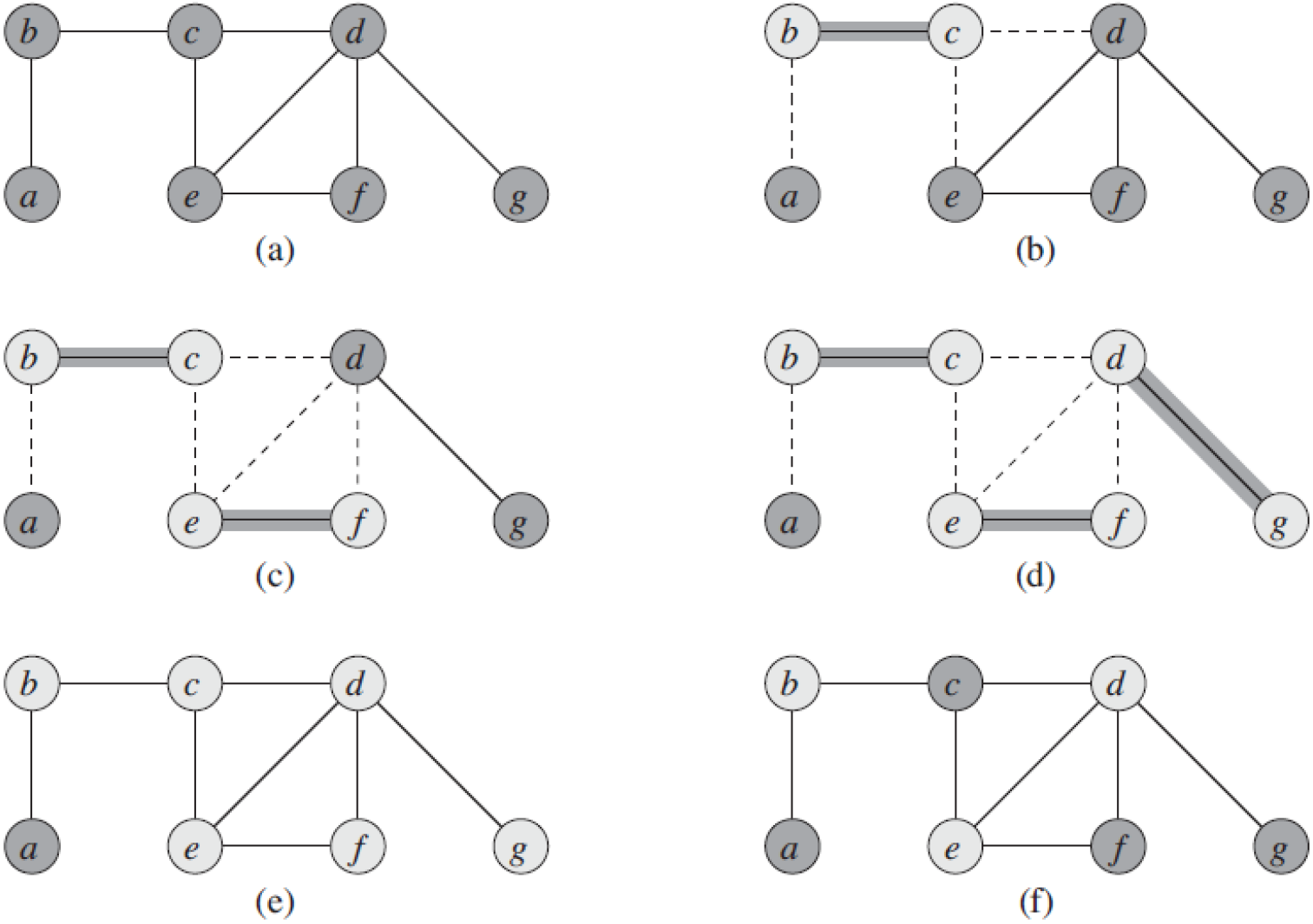
顶点覆盖问题

- 顶点覆盖的规模：V'包含的顶点数
- 顶点覆盖问题：在一个给定的无向图中找出一个具有最小规模的顶点覆盖（最优顶点覆盖）

```
APPROX_VERTEX_COVER(G)
1   C ← ∅
2   E' ← G.E
3   while E' ≠ ∅ do
4       选择E'中任一条边(u, v)
5       C ← C ∪ {u, v}
6       将E'中与u及v邻接的边全部删除
7   return C
```

$O(|V|+|E|)$

顶点覆盖问题 (续)



顶点覆盖问题 (续)

■APPROX_VERTEX_COVER算法近似比为2，即返回的规模最多是最优顶点覆盖规模的2倍

- 证明：（通过算法第4行选出的边的数量进行过渡）

  设算法第4行选出的边的数量为$|A|$，则$|C| = 2|A|$；

  设最优解为C^*，因为A中的边不共用顶点，则A中每条边至少有一个顶点要在C^*中，即 $|A| \leq |C^*|$；

  综合得到$|C| \leq 2|C^*|$，即 $\mathrm{SOL/OPT} \leq 2$

```
APPROX_VERTEX_COVER(G)
1   C ← ∅
2   E' ← G.E
3   while E' ≠ ∅ do
4       选择E'中任一条边(u, v)
5       C ← C ∪ {u, v}
6       将E'中与u及v邻接的边全部删除
7   return C
```

旅行商问题TSP

■旅行商问题TSP (Traveling Salesman Problem)

- 给定一个无向完全图$G=(V, E)$及定义在$V\times V$上的一个费用函数c和一个整数k，判定G是否存在经过V中各顶点恰好一次的回路，使得该回路的费用不超过k
- 一个售货员必须访问n个城市，售货员希望恰好访问每个城市一次，并最终回到出发城市。售货员从城市i到城市j的旅行费用为一个整数$c(i, j)$，旅行所需的全部费用是他旅行经过的各边费用之和，售货员希望整个旅行费用最低。判定问题为：旅行费用不超过k。

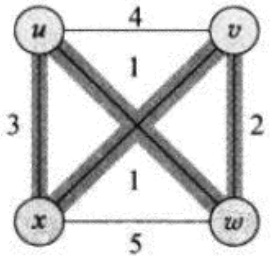


图 34-18 旅行商问题的一个实例。阴影覆盖的边表示费用最低的旅行路线，其费用为 7

旅行商问题TSP (续)

- 输入：无向完全图 $G=(V, E)$，每条边$(u, v)\in E$有一个非负整数代价$c(u, v)$
- 输出：G的一条具有最小代价的哈密顿回路（G中每个顶点只经过一次）
- 实际情况中，代价通常满足三角不等式，即：
  $c(u, w) \leq c(u, v) + c(v, w)$
  即：直达代价更小

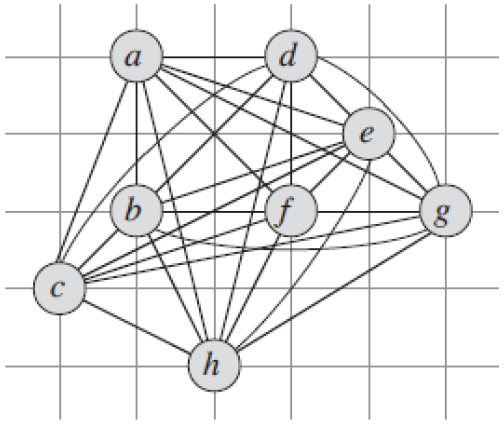
旅行商问题TSP (续)

■满足三角不等式的TSP：最小生成树法求近似解

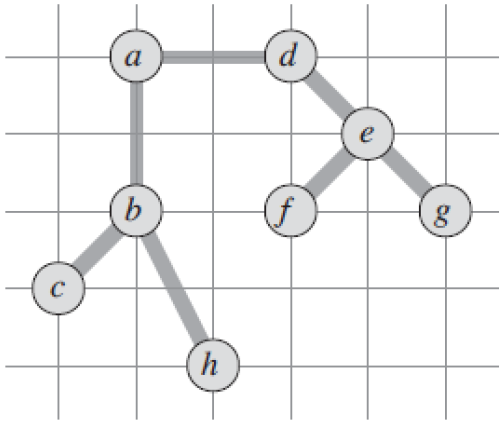
APPROX_TSP_TOUR(G, c)
1 选择一个顶点$r \in G.V$
2 使用Prim算法构造图G以r为根结点的最小生成树T
3 先序遍历T，生成顶点序列H
4 **return** 依次经过序列H中顶点、并最后回到第一个顶点的路径

- 运行时间：$\Theta(|V|^2)$
- 算法近似比：2
  - 证明：最小生成树的权值是最优旅行路线的下界：$c(T) \leq OPT$
    设仅能在最小生成树的边上进行旅行，那么先序遍历并回到根结点的代价：$c(W)=2c(T)$
    由于三角不等式，算法输出的代价$SOL \leq c(W)$
    由此得到：$SOL \leq 2OPT$，即$SOL/OPT \leq 2$

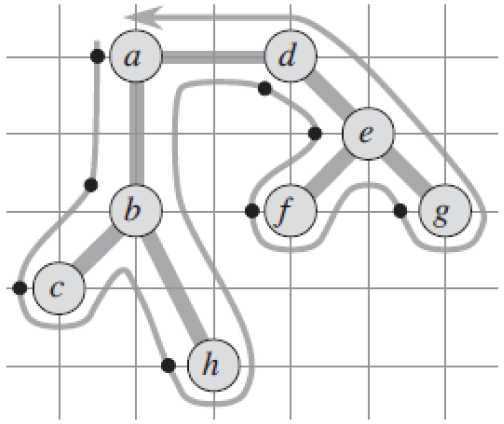
旅行商问题TSP (续)



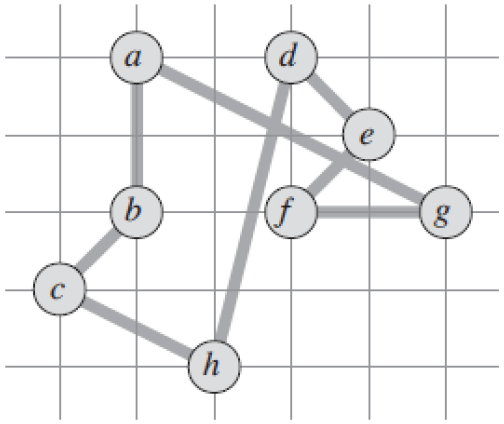
(a)



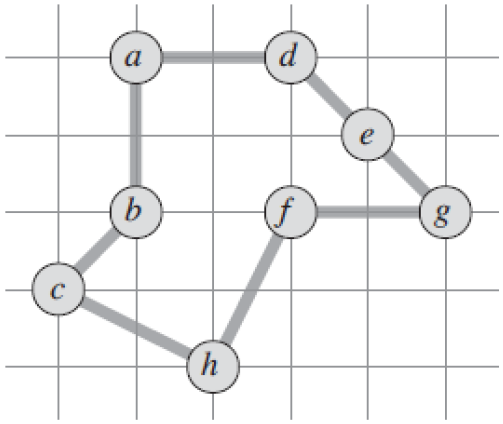
(b)



(c)



(d)

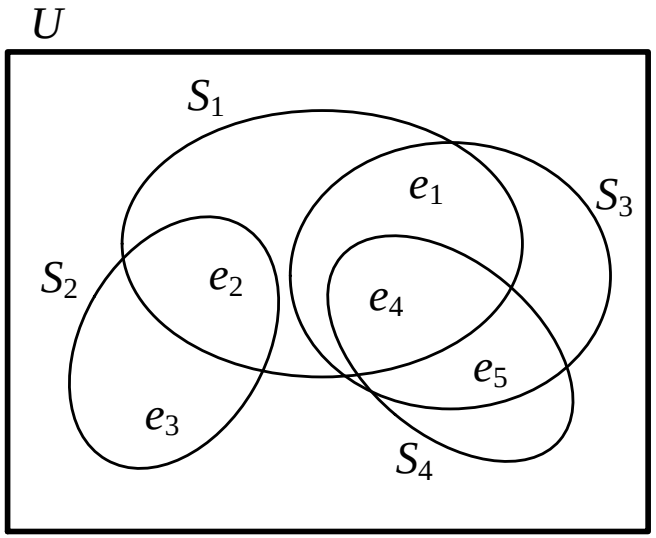


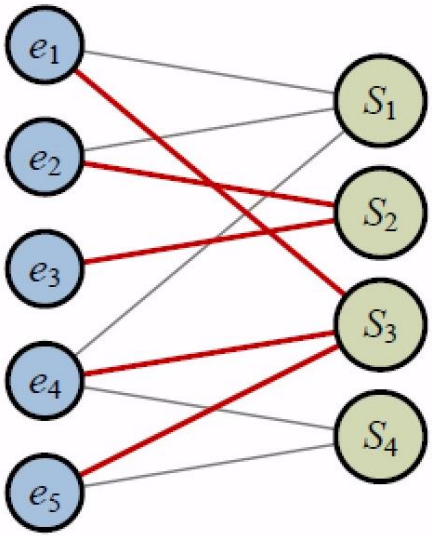
(e)

旅行商问题TSP (续)

- 一般TSP：P≠NP情况下，不存在多项式时间内具有常数近似比的近似算法（定理35.3）

集合覆盖问题





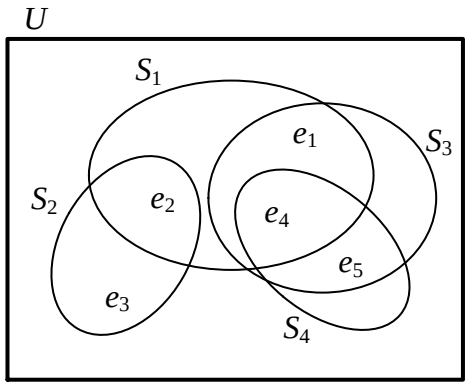
Set Cover

Given a universe U of n elements, a collection of subsets of U, $\mathcal{S} = \{S_1, ..., S_m\}$, and a cost function $c : \mathcal{S} \to \mathbf{Q}^+$, find a minimum cost subcollection of $\mathcal{S}$ that covers all elements of U.

集合覆盖问题 (续)

- 代价函数通常为解集合族中集合的数量，即用最少的集合覆盖所有的元素
- 贪心近似算法：每次选择覆盖元素最多的集合

```
GREEDY_SET_COVER(U, S)
1   X ← U
2   F ← ∅
3   while X ≠ ∅ do
4       选择使得|S_i∩X|最大的S_i∈S
5       X ← X - S_i
6       F ← F ∪ {S_i}
7   return F
```



多项式时间的
(ln|U|+1)近似算法

子集和问题

与0-1背包问题联系：
物品——正整数
物品价值——正整数的值
物品重量——正整数的值
背包容量——整数和t

■子集和问题SUBSET-SUM

- 给定一个正整数有限集S和一个整数目标$t>0$，判定是否存在S的一个子集$S' \subseteq S$，使得S'中的整数的和为t
- 例：S={1, 2, 7, 14, 49, 98, 343, 686, 2409, 2793, 16808, 17206, 117705, 117993}, t =138457，则子集S'={1, 2, 7, 98, 343, 686, 2409, 17206, 117705}是该问题的一个解
- 最优化问题：找到子集$S' \subseteq S$，使得S'中的整数的和尽可能大，但不能超过为t
- 存在$O(|S|t)$的伪多项式时间算法

子集和问题 (续)

■FPTAS算法：

以下设$n = |S|$，且$S=\{x_1, x_2, ..., x_n\}$

对任意$\varepsilon>0$，令$k = \left\lfloor \frac{\varepsilon x_{\max}}{n} \right\rfloor$，其中$x_{\max} = \max\limits_{1\le i\le n} x_i$

设置$x_i' = \lfloor x_i/k \rfloor, \quad i = 1, 2, ..., n$

返回以x_i'为物品价值、x_i为物品重量、t为背包容量的0-1背包动态规划算法的解

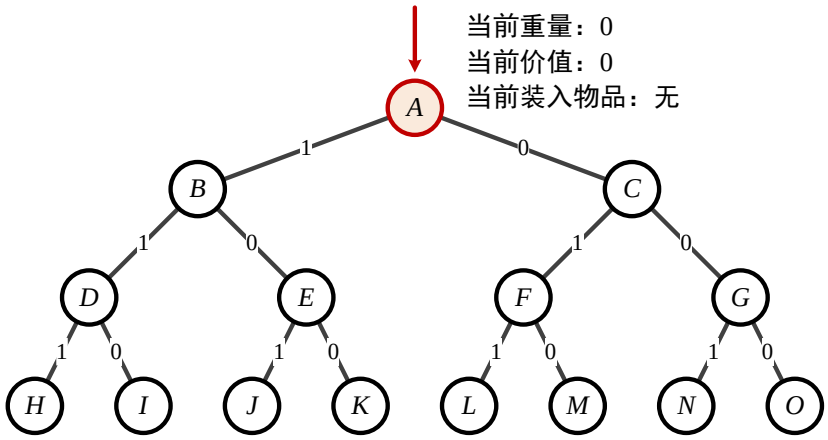
- 时间复杂度：$O(n^3/\varepsilon)$
- 近似比：SOL/OPT $\geq$ 1-ε

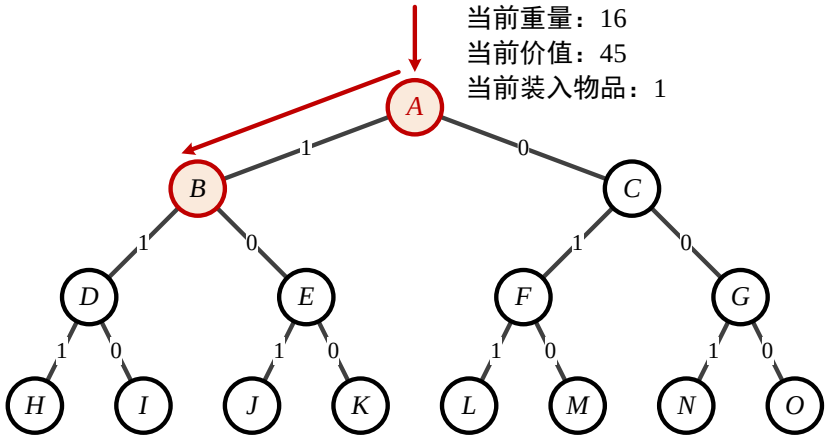
回溯法算法框架 (续)

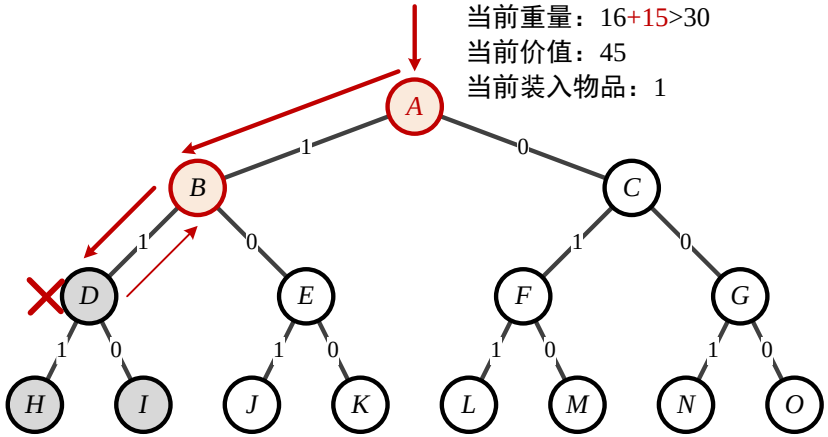
- 回溯法基本思想
  - 从开始结点（根结点）出发，以深度优先方式搜索整个解空间
  - 当前结点为扩展结点，可继续纵深搜索的结点为活结点，无法纵深搜索的为死结点
  - 扩展结点为死结点时，回溯至最近的一个活结点处，这个活结点作为扩展结点，递归求解，直至找到解或无活结点为止

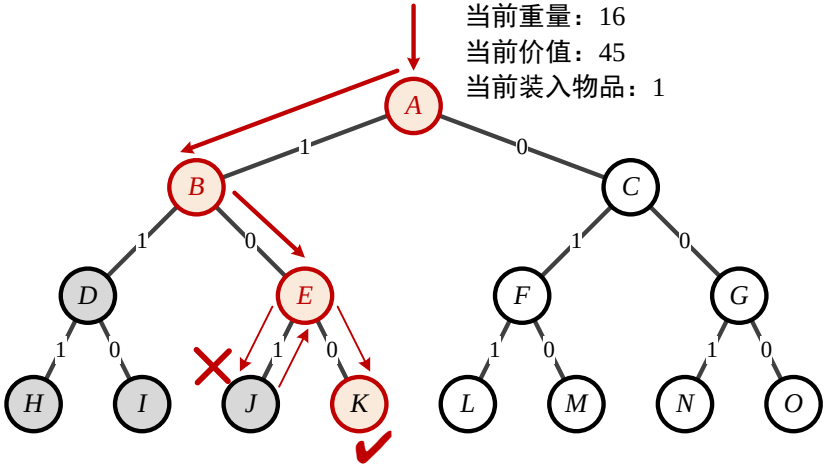
回溯法算法框架 (续)

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



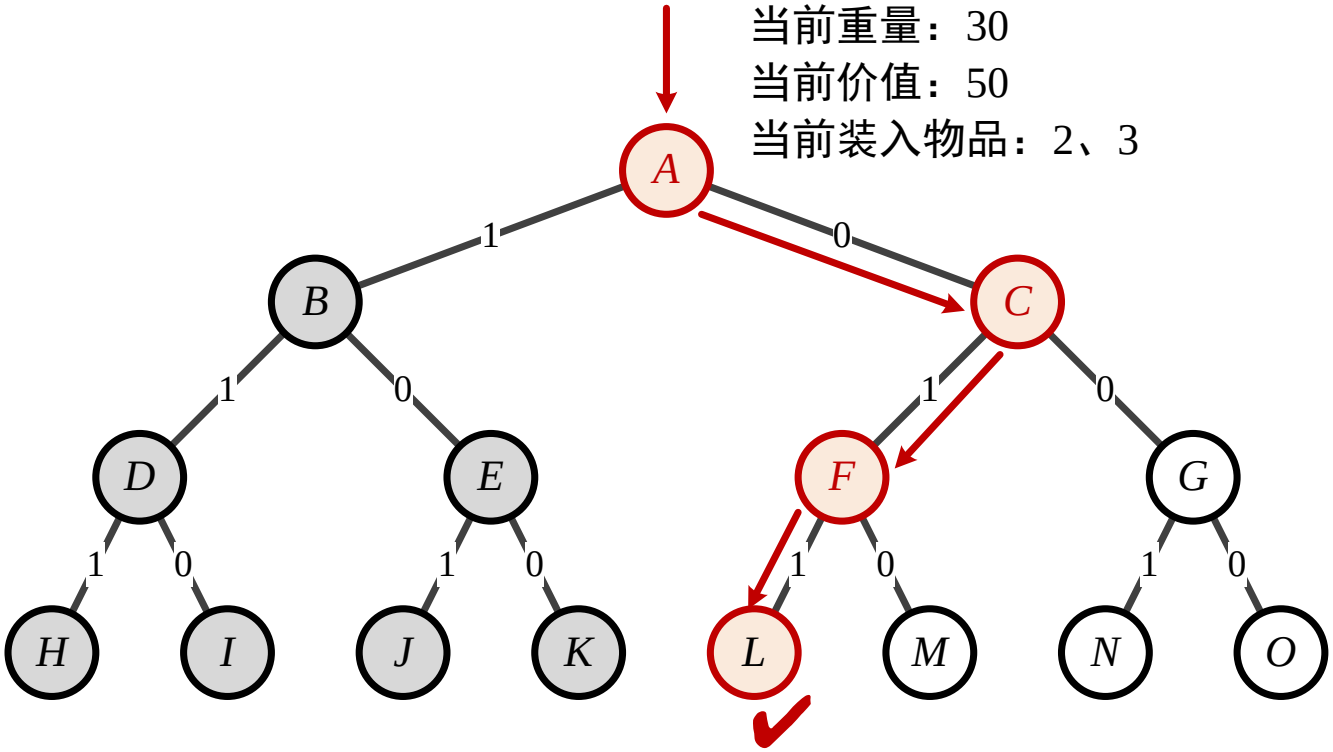






回溯法算法框架 (续)

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



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儿子结点。在这些儿子结点中，导致不可行解或导致非最优解的儿子结点被舍弃，其余儿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

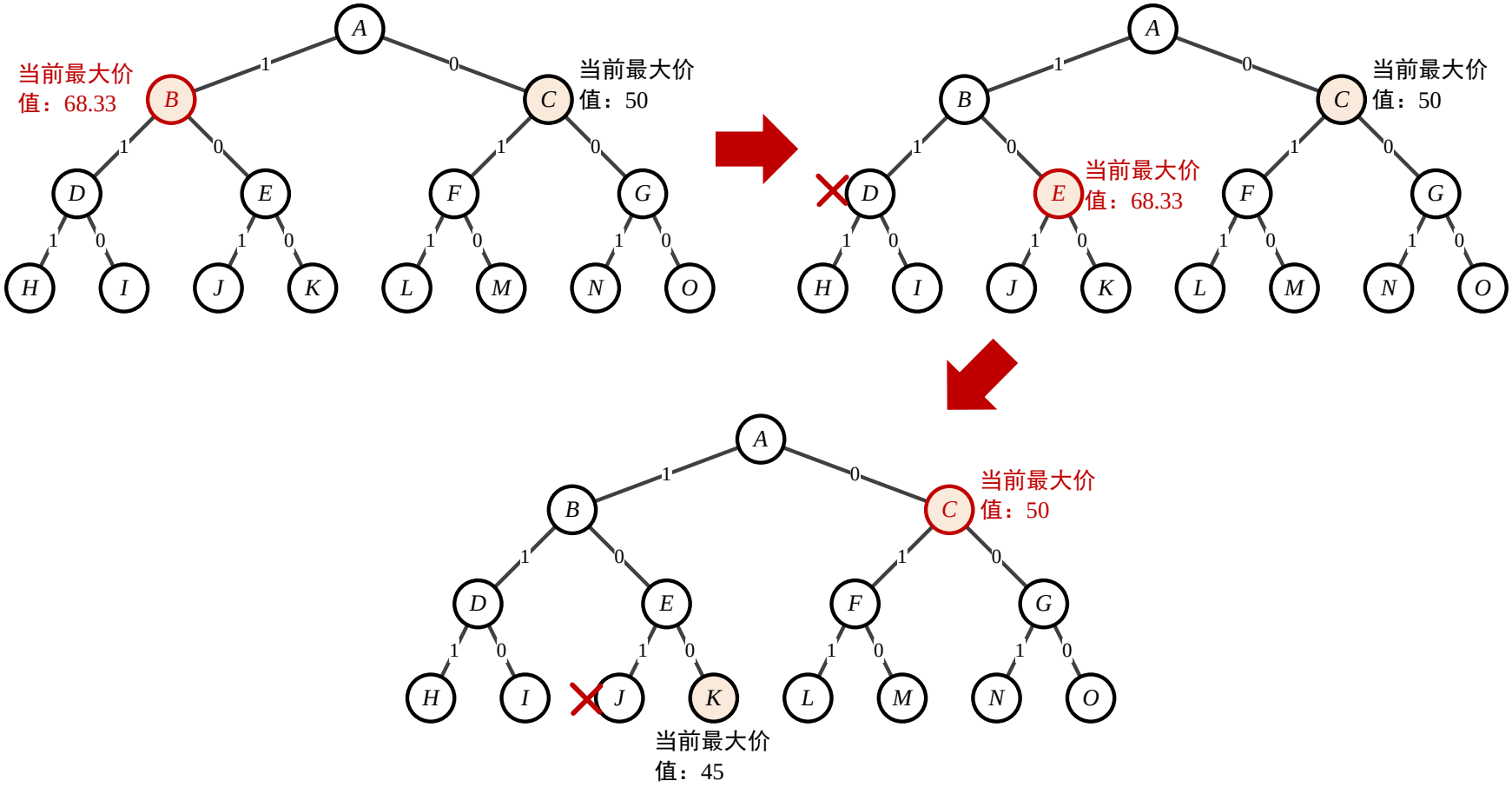
分支限界法求解举例——0-1背包问题

■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：已装袋的物品价值 + 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儿子结点的可行性
- 如果该左儿子结点是可行结点，则将它加入到子集树和活结点优先队列中
- 当前扩展结点的右儿子结点一定是可行结点，仅当右儿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

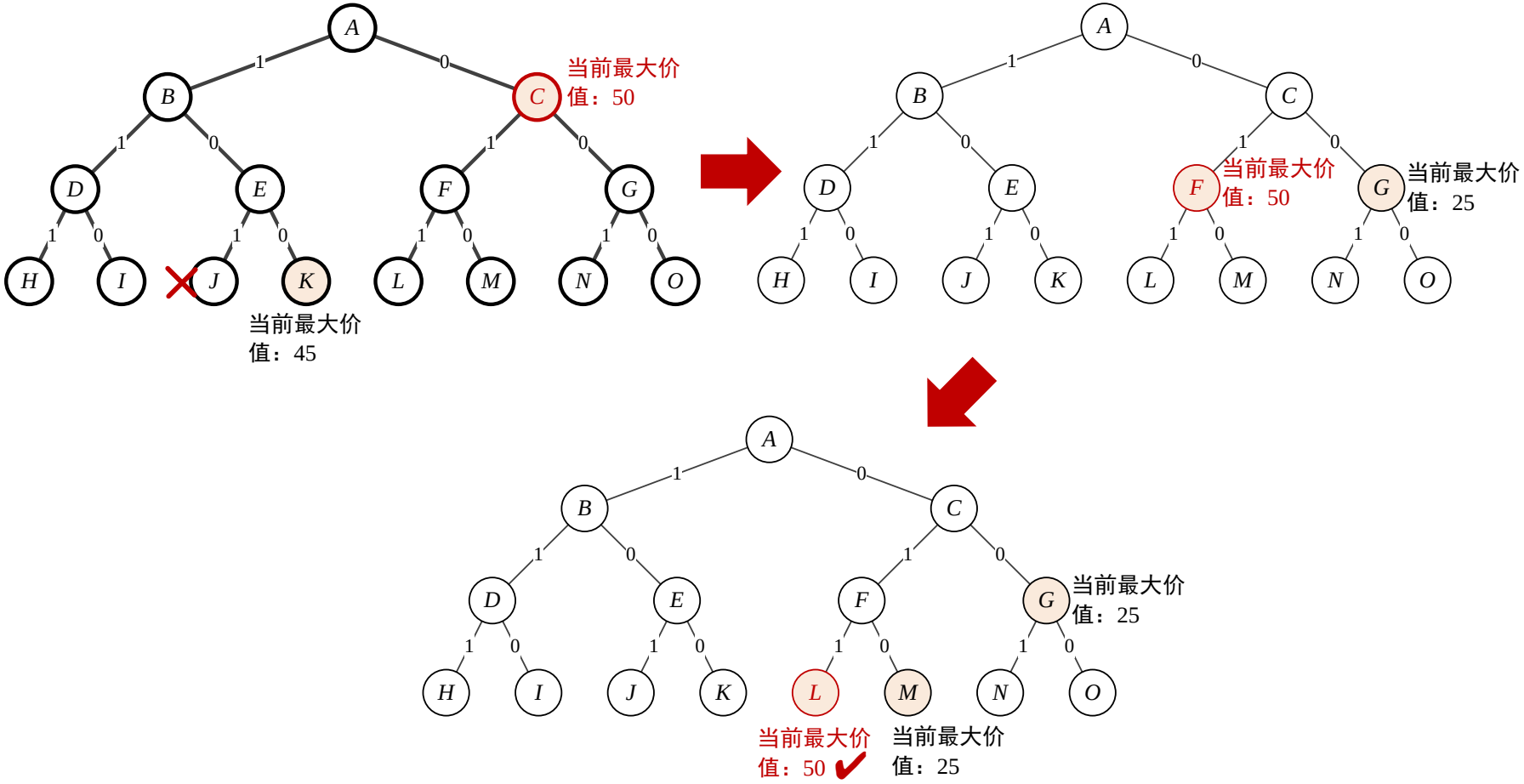
分支限界法求解举例——0-1背包问题(续)

■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]



Las Vegas算法

■特点

- ➢要么返回正确的解，要么随机决策导致一个僵局
- ➢若陷入僵局，使用同一实例运行同一算法，有独立的机会求出解
- ➢成功的概率随着执行时间的增加而增加

■算法的一般形式

```
OBSTINATE(x)
1   repeat
2       LV(x, y, success)
3   until success
4   return y
```

x：输入实例，y：返回值
success：布尔值指示执行成功/失败

Monte Carlo算法

■特点

- ➢偶尔会出错，但对任何实例均能以高概率找到正确解
- ➢算法运行次数越多，得到正确解的概率越高

■基本概念

- ➢设p是一个实数，且$1/2 < p < 1$，若一个Monte Carlo算法以不小于p的概率返回一个正确的解，则该MC算法称为 p 正确，算法的优势 (advantage) 是 $p - 1/2$
- ➢若一个Monte Carlo算法对同一实例决不给出两个不同的正确解，则该算法称是相容的 (consistent) 或一致的

为了增加一个一致的、p 正确算法成功的概率，只需多次调用同一算法，然后选择出现次数最多的解

Monte Carlo算法应用——主元素问题

■问题：设$A[1..n]$是含有n个元素的数组，若A中等于x的元素个数大于$n/2$，则称x是数组A的主元素

➢注：若存在，则只可能有1个主元素

■例：数组$A=\{3, 2, 3, 2, 3, 3, 5\}$，共7个元素，其中元素3出现4次，占一半以上，因此A存在主元素3

Monte Carlo算法应用——主元素问题(续)

```
MAJ(A)
1   i ← RANDOM(1, n)
2   x ← A[i]
3   k ← 0
4   for j ← 1 to n do
5       if A[j] = x
6           k ← k + 1
7   return (k > n/2)
```

- 返回true：A含有主元素x，算法一定正确
- 返回false：元素x不是A的主元素，算法可能错误（仅包含主元素时出错）
- A确实包含一个主元素时，x为主元素的概率大于1/2

MAJ是偏真1/2正确的算法

Monte Carlo算法应用——主元素问题(续)

■算法改进：通过重复调用技术降低错误概率

- 重复调用MAJ的结果是相互独立的
- 当A含有主元素时，k次重复调用MAJ均返回false的概率为$(1-p)^k = 2^{-k}$
- 在k次调用中，只要有一次MAJ返回true，即可判定A有主元素
- 当需要控制算法出错概率小于$\varepsilon > 0$时，相应算法调用MAJ的次数为：

$$\varepsilon = 2^{-k} \Rightarrow k = \left\lceil \lg \frac{1}{\varepsilon} \right\rceil$$

```
MAJMC(A, ε)
1  k ← ⌈lg(1/ε)⌉
2  for i ← 1 to k do
3      if MAJ(A)
4          return true
5  return false
```

时间复杂度为$O(n\lg(1/\varepsilon))$

注意，这里只是用此问题来说明MC算法，实际上对于判定主元素问题存在$O(n)$的确定性算法

Sherwood算法

- 分析确定性算法在平均情况下的时间复杂度时，通常假定算法的输入实例满足某一特定的概率分布
- 很多算法对于不同输入实例运行时间差别很大，可采用Sherwood概率算法消除时间复杂度与输入实例间的依赖关系
- 通常有两种方式：
  - 在确定性算法的某些步骤引入随机因素
  - 仅对输入实例随机处理，再执行确定性算法

Sherwood算法实例——快速排序

```
RAND_QUICKSORT(A, low, high)
1   //A: 待排序数组，low/high: 排序起始/终止下标
2   if low < high
3       i ← RANDOM(low, high); //low..high随机抽取一个下标
4       swap(A[low], A[i])
5       k ← PARTITION(A, low, high)
6       RAND_QUICKSORT(A, low, k-1)
7       RAND_QUICKSORT(A, k+1, high)
```

引入随机因素

```
SHUFFLE(A)
1   n ← A.length
2   for i ← 1 to n-1 do
3       //在A[i..n]中随机选一个元素放在A[i]上
4       j ← RANDOM(i, n)
5       swap(A[i], A[j])
6   执行原确定性算法
```

原算法较复杂，很难对其进行修改时可适用

输入实例随机处理