算法作业部分答案

2019

第2章

在此输入您的封面副标题

2.1-2 重写过程 INSERTION-SORT，使之按非升序(而不是按非降序)排序。

INSERTION-SORT(A)	*cost*	*times*
for $j \leftarrow 2$ **to** $length[A]$	c_1	n
do $key \leftarrow A[j]$	c_2	$n-1$
▷ Insert $A[j]$ into the sorted sequence $A[1..j-1]$.	0	$n-1$
$i \leftarrow j-1$	c_4	$n-1$
while $i>0$ and $\boxed{A[i]>key}$ <	c_5	$\sum_{j=2}^{n} t_j$
do $A[i+1] \leftarrow A[i]$	c_6	$\sum_{j=2}^{n} (t_j-1)$
$i \leftarrow i-1$	c_7	$\sum_{j=2}^{n} (t_j-1)$
$A[i+1] \leftarrow key$	c_8	$n-1$

2.1-3 考虑下面的查找问题：

输入：一列数 $A=\langle a_1, a_2, \cdots, a_n\rangle$ 和一个值 v。

输出：下标 i，使得 $v=A[i]$，或者当 v 不在 A 中出现时为 NIL。

写出针对这个问题的线性查找的伪代码，它顺序地扫描整个序列以查找 v。利用循环不变式证明算法的正确性。确保所给出的循环不变式满足三个必要的性质。

Algorithm 1 LINEAR-SEARCH(A, v)

```
Input: A = ⟨a1, a2, ... an⟩ and a value v.
Output: An index i such that v = A[i] or nil if v ∉ A
for i ← 1 to n do
    if A[i] = v then
        return i
    end if
end for
return nil
```

循环不变式：对于任意的x<i,A[x]!=v

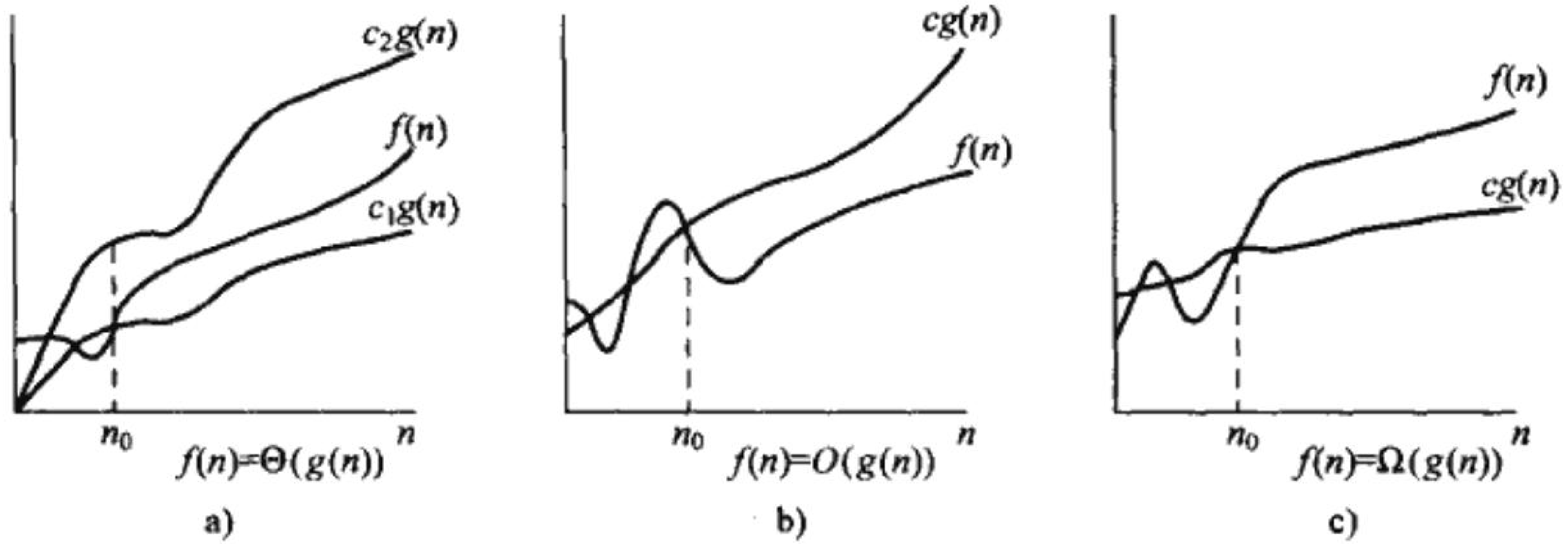
三条性质：初始化；保持；终止

2.2-1 用 Θ 形式表示函数 $n^3/1000-100n^2-100n+3$。

Θ 记号 限制一个函数在常数因子内

O 记号 给出一个函数在常数因子内的上限

Ω 符号 给出一个函数在常数因子内的下限



$\Theta(g(n))=\{f(n)$：存在正常数 c_1，c_2 和 n_0，使对所有的 $n \geqslant n_0$，有 $0 \leqslant c_1 g(n) \leqslant f(n) \leqslant c_2 g(n)\}$ 对任一个函数 $f(n)$，若存在正常数 c_1，c_2，使当 n 充分大时，$f(n)$ 能被夹在 $c_1 g(n)$ 和 $c_2 g(n)$ 中间，则 $f(n)$ 属于集合 $\Theta(g(n))$。因为 $\Theta(g(n))$ 是一个集合，可以写成"$f(n) \in \Theta(g(n))$"

$$c_1 n^3 \le n^3/1000 - 100n^2 - 100n + 3 \le c_2 n^3 \text{成立}$$

$$c_1 \le \frac{1}{1000} - \frac{100}{n} - \frac{100}{n^2} + \frac{3}{n^3} \le c_2$$

$c_2 \ge 1$,对所有 $n \ge 1$ 成立

$$c_1 \le \frac{1}{1000} - \frac{1}{10^4} - \frac{1}{10^{10}} \text{对所有} n \ge 10^6 \text{成立}$$

选择 $c_1 = 10^{-3} - 10^{-4} - 10^{-10}, c_2 = 1, n_0 = 10^6$ $n^3/1000 - 100n^2 - 100n + 3 = \Theta(n^3)$.

2.2-2 考虑对数组 A 中的 n 个数进行排序的问题：首先找出 A 中的最小元素，并将其与 A[1]中的元素进行交换。接着，找出 A 中的次最小元素，并将其与 A[2]中的元素进行交换。对 A 中头 $n-1$ 个元素继续这一过程。写出这个算法的伪代码，该算法称为选择排序(selection sort)。对这个算法来说，循环不变式是什么？为什么它仅需要在头 $n-1$ 个元素上运行，而不是在所有 n 个元素上运行？以 Θ 形式写出选择排序的最佳和最坏情况下的运行时间。

Algorithm 2 SELECTION-SORT(A)

```
Input: A = ⟨a1, a2, ... an⟩
Output: sorted A.
for i ← 1 to n − 1 do
    j ← FIND-MIN(A, i, n)   # 找出A[i……n]中的最小元素，并把下标赋给j
    A[j] ↔ A[i]
end for
```

循环不变式：A[1，……，i-1]已经排好序，且比其它n-i+1个数都小。

当数组A[1,……，n-1]已经排好序，A[n]必定比前面任意数都大，因此只需执行n-1次即可。

最佳和最坏情况下，都需要执行FIND-MIN函数，此函数决定了时间复杂度上的边界：

$$\Theta\left(\sum_{i=1}^{n} i\right) = \Theta(n^2)$$

第3章

在此输入您的封面副标题

3.1-1 设 $f(n)$ 与 $g(n)$ 都是渐近非负函数。利用 Θ 记号的基本定义来证明 $\max(f(n), g(n)) = \Theta(f(n)+g(n))$。

由第三章的Θ记号的定义：找到c_1，c_2，对于所有的$n \geq n_0$，下式都成立

$$0 \leqslant c_1(f(n)+g(n)) \leqslant \max(f(n), g(n)) \leqslant c_2(f(n)+g(n))$$

可以令$c_1=0.5$，$c_2=1$，对于所有的$n \geq n_0$，不等式都成立

3.1-2 证明对任意实常数 a 和 b，其中 $b>0$，有

$$(n+a)^b=\Theta(n^b)$$

$c_1n^b \le (n+\mathrm{a})^{\mathrm{b}} \le c_2n^b$ （$n, n+a$均大于零（即 $n>-a$），幂函数为增函数）

令$c_1=d_1{}^b, c_2=d_2{}^b$,即$(d_1n)^b \le (n+a)^b \le (d_2n)^b$

$$d_1n \le n+a \le d_2n$$

令 $d_1=1/2, d_2=1$,得 $n\geq -2a$，$n>a$

即 $n_0=\max(-2a, -a, a)$，$c_1=(1/2)^b$, $c_2=1$时，恒有 $c_1n^b \le (n+\mathrm{a})^{\mathrm{b}} \le c_2n^b$

$$n-|a| \le n+a \le n+|a|$$

$$\left.\begin{array}{l} |a| \le \dfrac{n}{2}\text{时，}\quad n+a \ge n-|a| \ge \dfrac{n}{2} \\ |a| \le n\text{时，}\quad n+a \le 2n \end{array}\right\} n \ge 2|a|\text{时，}\ \frac{n}{2} \le n+a \le 2n$$

当b>0,底数x为正实数时，为单调递增函数，

根据Θ定义，结论成立

第4章

在此输入您的封面副标题

4.3-1 用主方法来给出下列递归式紧确的渐进界：

a) T(n)=4T(n/2)+n

b) $T(n)=4T(n/2)+n^2$

c) $T(n)=4T(n/2)+n^3$

定理 4.1(主定理) 设 $a\geqslant 1$ 和 $b>1$ 为常数，设 $f(n)$ 为一函数，$T(n)$ 由递归式

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

对非负整数定义，其中 n/b 指 $\lfloor n/b \rfloor$ 或 $\lceil n/b \rceil$。那么 $T(n)$ 可能有如下的渐近界：

1)若对于某常数 $\epsilon>0$，有 $f(n)=O(n^{\log_b a-\epsilon})$，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)若对某常数 $\epsilon>0$，有 $f(n)=\Omega(n^{\log_b a+\epsilon})$，且对常数 $c<1$ 与所有足够大的 n，有 $af(n/b)\leqslant cf(n)$，则 $T(n)=\Theta(f(n))$。 ■

Use the master method to find bounds for the following recursions. Note that $a = 4, b = 4$ and $n^{\log_2 4} = n^2$

- $T(n) = 4T(n/2) + n$. Since $n = O(n^{2-\epsilon})$ case 1 applies and we get $T(n) = \Theta(n^2)$.
- $T(n) = 4T(n/2) + n^2$. Since $n^2 = \Theta(n^2)$ we have $T(n) = \Theta(n^2 \lg n)$.
- $T(n) = 4T(n/2) + n^3$. Since $n^3 = \Omega(n^{2+\epsilon})$ and $4(n/2)^3 = 1/2n^3 \leqslant cn^3$ for some $c < 1$ we have that $T(n) = \Theta(n^3)$.

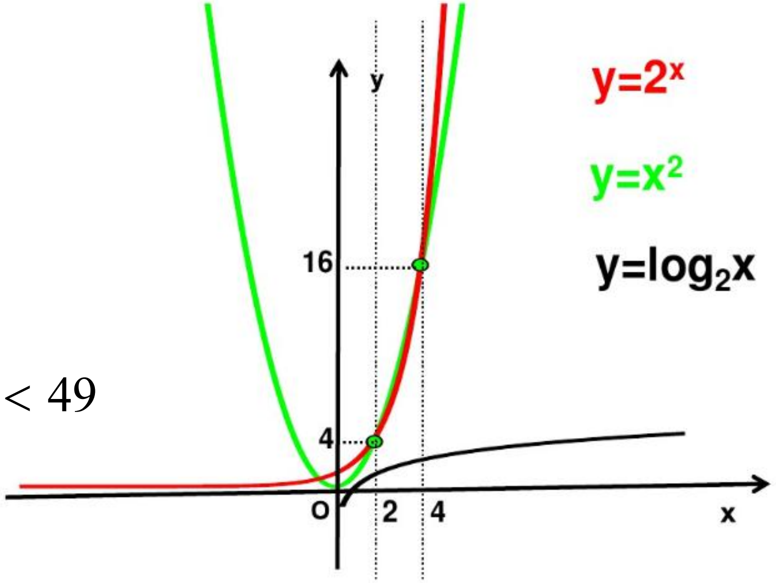
4.3-2 某个算法A的运行时间由递归式$T(n)=7T(n/2)+n^2$表示；另一个算法A'的运行时间为$T'(n)=aT'(n/4)+n^2$。若要A'比A更快，那么a的最大整数值是多少？

根据定理，A的时间复杂度：$T(n)=\Theta(n^{\lg 7})$

对于 A'，$n^{\log_b a} = n^{\log_4 a}$

$$\begin{cases} a < 16\text{时}，\quad T'(n)=\Theta(n^2),\ A'\text{比}A\text{快} \\ a = 16\text{时}，\quad T'(n)=\Theta(n^2 \lg n),\ A'\text{比}A\text{快} \\ a > 16\text{时}，\quad T'(n)=\Theta(n^{\lg\sqrt{a}}),\ \text{若}A'\text{比}A\text{快，则}\sqrt{a}<7,\text{即}\,a<49 \end{cases}$$

因此，$a = 48$



第7章

在此输入您的封面副标题

7.1-2 当数列A[p...r]中的元素都相同时，PARTITION返回的q值是多少？修改PARTITION，使得当数列A[p...r]中所有元素的值都相同时，$q=\lfloor (p+r)/2 \rfloor$。

```
PARTITION(A, p, r)
x ← A[r]
i ← p−1
for j ← p to r−1
        do if A[j]≤x
                then i ← i+1
                        exchange A[i]↔A[j]
exchange A[i+1]↔A[r]
return i+1
```

元素都相同时返回的值为r

```
PARTITION'(A, p, r)

x = A[r]
i = p - 1
for j = p to r - 1
        if A[j] <= x
                i = i + 1
                exchange A[i] with A[j]
i = i + 1
exchange A[i] with A[r]

if i = r
        return (p + r) / 2
return i
```

中国科学技术大学

7.2-1 用代入法证明：正如本节开头所提到的那样，返回式 $T(n)=T(n-1)+\Theta(n)$ 的解为：$T(n)=\Theta(n^2)$。

快速排序的最坏情况划分: $T(n)=T(n-1)+T(0)+\Theta(n)=T(n-1)+\Theta(n)$

代入法:

1. 猜测解的形式。
2. 用数学归纳法求出解中的常数，并证明解是正确的。

令$\Theta(n)=c_2n$,猜测解为$T(n)=O(n^2)$,代入法要证明适当选择常数c_1,有$T(n)\le c_1n^2$

$$\begin{aligned}T(n)&=T(n-1)+c_2n\\&\leqslant c_1(n-1)^2+c_2n\\&=c_1n^2-2c_1n+c_1+c_2n \qquad (2c_1>c_2,n\geqslant c_1/(2c_1-c_2))\\&\leqslant c_1n^2\end{aligned}$$

令$\Theta(n) = c_2 n$,猜测解为$T(n) = \Omega(n^2)$,代入法要证明适当选择常数c_3,有$T(n) \ge c_3 n^2$

$$
\begin{aligned}
T(n) &= T(n-1) + c_2 n \\
&\ge c_3 (n-1)^2 + c_2 n \\
&= c_3 n^2 - 2c_3 n + c_3 + c_2 n \\
&\ge c_3 n^2
\end{aligned}
\qquad \left(2c_3 < c_2, n \ge -\frac{c_3}{c_2 - 2c_3} \right)
$$

因此，T(n)=Θ(n^2)

7.2-2 当数列A包含的元素都具有相同值时，QUICKSORT的时间复杂度是什么？

```
QUICKSORT(A, p, r)
  if p<r
     then q ← PARTITION(A, p, r)
          QUICKSORT(A, p, q−1)
          QUICKSORT(A, q+1, r)
```

A中元素都相等时，PARTITION划分极不平衡n-1:0，快速排序总的时间复杂度为$\Theta(n^2)$

7.4-1 证明：在递归式：

$$T(n)=\max_{0\leqslant q\leqslant n-1}(T(q)+T(n-q-1))+\Theta(n)$$

中，$T(n)=\Omega(n^2)$。

猜测 $T(n)\ge c_1n^2$

$$T(n)=\max_{0\le q\le n-1}(T(q)+T(n-q-1))+\Theta(n)$$

$$\ge \max_{0\le q\le n-1}(c_1q^2+c_1(n-q-1)^2)+cn$$

二阶导数为正，在端点处取得最大值

$$\ge c_1(n^2-2n+1)+cn$$

$$\ge c_1n^2$$

找到$c>2c_1$, $n\ge -c_1/(c-2c_1)$ 因此结论成立。

第14章

在此输入您的封面副标题

14.1-3 写出OS-SELECT的非递归形式。

```
OS-SELECT(x, i)
{  r = size[left[x]] + 1; //秩，size[left[x]] 是对树进行中序遍历，排在x之前的结点的个数
   while (r != i)
    {
      if (i < r)
      {
        x = left[x];
         r = size[left[x]] + 1;
      }
       else  //i>r时
       {
         x = right[x];
         i = i -r;   //共有r个元素排在右子树之前，
         r = size[left[x]] + 1;
       }
     }
    return x;
 }
```

左子树遍历

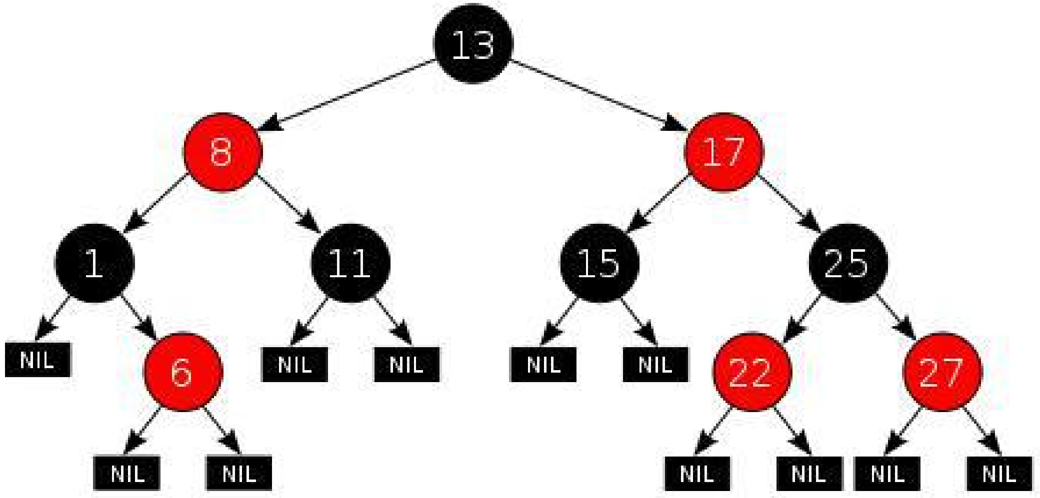
右子树遍历

14.2-2 能否在不影响任何红黑树操作性能的前提下，将结点的黑高度作为一个域来维护？如果可以的话，说明怎么做；如果不能，说明为什么。

黑高（bh(x)）: 从某结点出发，到达一个叶结点的任意一条简单路径上的黑色结点的个数。

定理14.1（红黑树的扩张）设域f对含n个结点的红黑树进行扩张的域，且假设某结点x的域f的内容可以仅用结点x,left[x]和right[x]中的信息计算，包括f[left[x]]和f[right[x]]。这样在插入和删除操作中，我们可以在不影响这两个操作O(lgn)渐进性能的情况下，对T的所有结点的f值进行维护。

红黑树



性质：
1. 每个结点或是红色的，或是黑色的。
2. 根节点是黑色的。
3. 每个叶结点（NIL）是黑色的。
4. 如果一个结点是红色的，则它的两个子节点都是黑色的。
5. 对每个结点，从该结点到其所有后代叶结点的简单路径上，均包含相同数目的黑色结点。

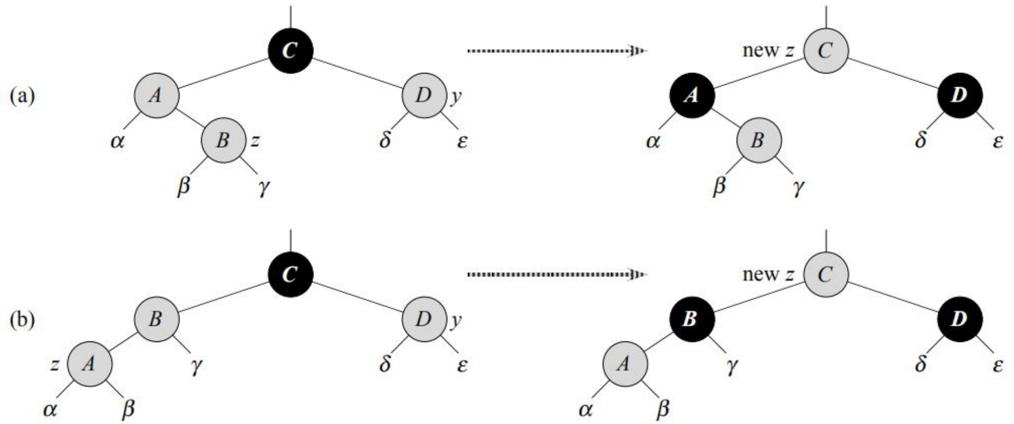
可以。

1. 从定理14.1知，f为黑高时，只与x、x.left、 x.right信息有关。可以在插入、删除操作期间对f进行维护，不影响O（lgn）的渐进性能。

2. 插入和删除操作只对根节点的某一叶节点的路径上的结点（或部分叔结点）进行修改，O(lgn)个结点（即树的高度）；每个结点黑高修改只需O(1)的时间。 因此总的时间为O(lgn)。

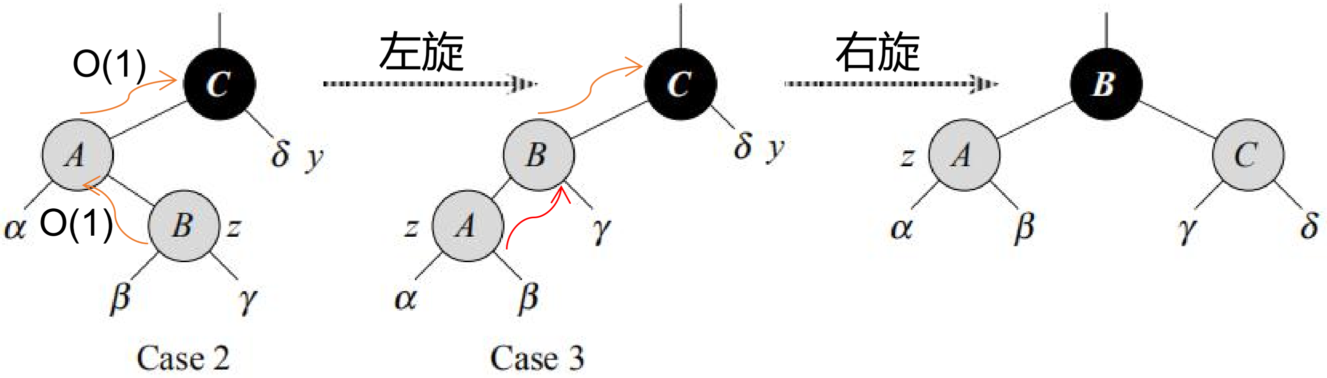
3. 以插入操作为例进行维护，分情况讨论。 附P181-182,第13章的三种情况示意图

情况1：z的叔结点为红色



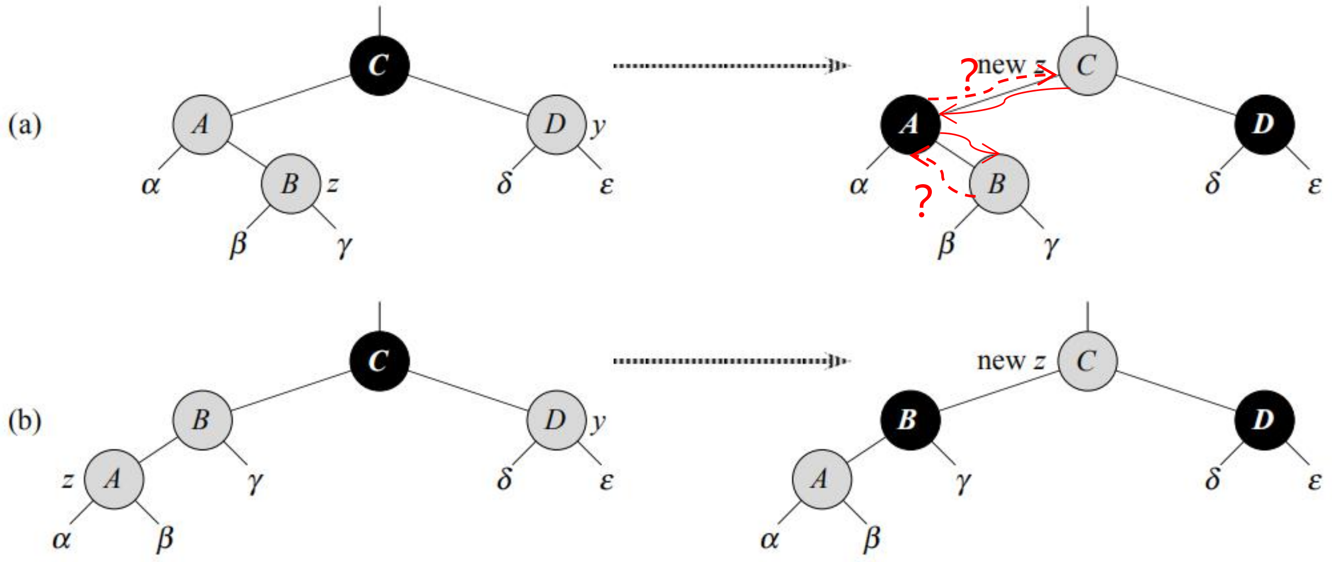
情况2：z的叔结点为黑色，z为右孩子

情况3：z的叔结点为黑色，z为左孩子



14.2-3 能否将红黑树中结点的深度做为一个域来进行有效的维护？如果可以的话，说明怎么做；如果不能，说明为什么。

不能，插入或删除一个结点时，内部结点的深度均改变，无法在O（lgn）内完成。



14.3-2 重写INTERVAL-SEARCH代码，使得当所有的区间都是开区间时，它也能正确的工作。

```
INTERVAL-SEARCH(T, i) // 找出树T中与区间i重叠的那个结点。
x ← root[T]
while x≠nil[T] and i does not overlap int[x]
    do if left[x]≠nil[T] and max[left[x]]≥low[i]
                                          >
        then x ← left[x]
        else x ← right[x]
return x
```

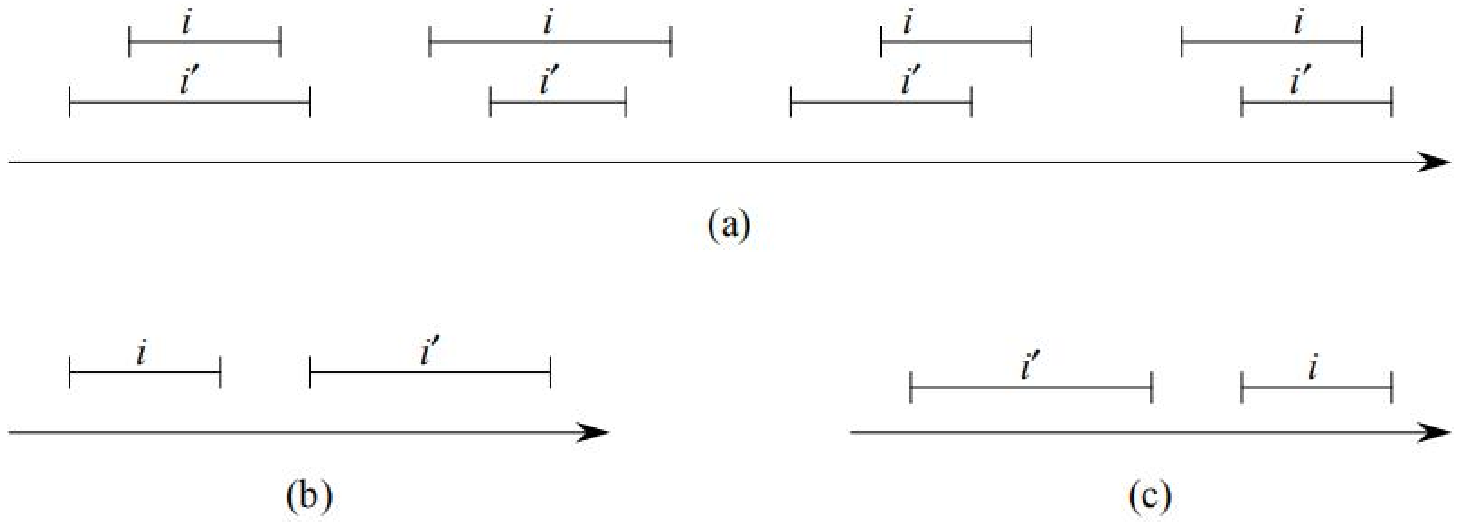


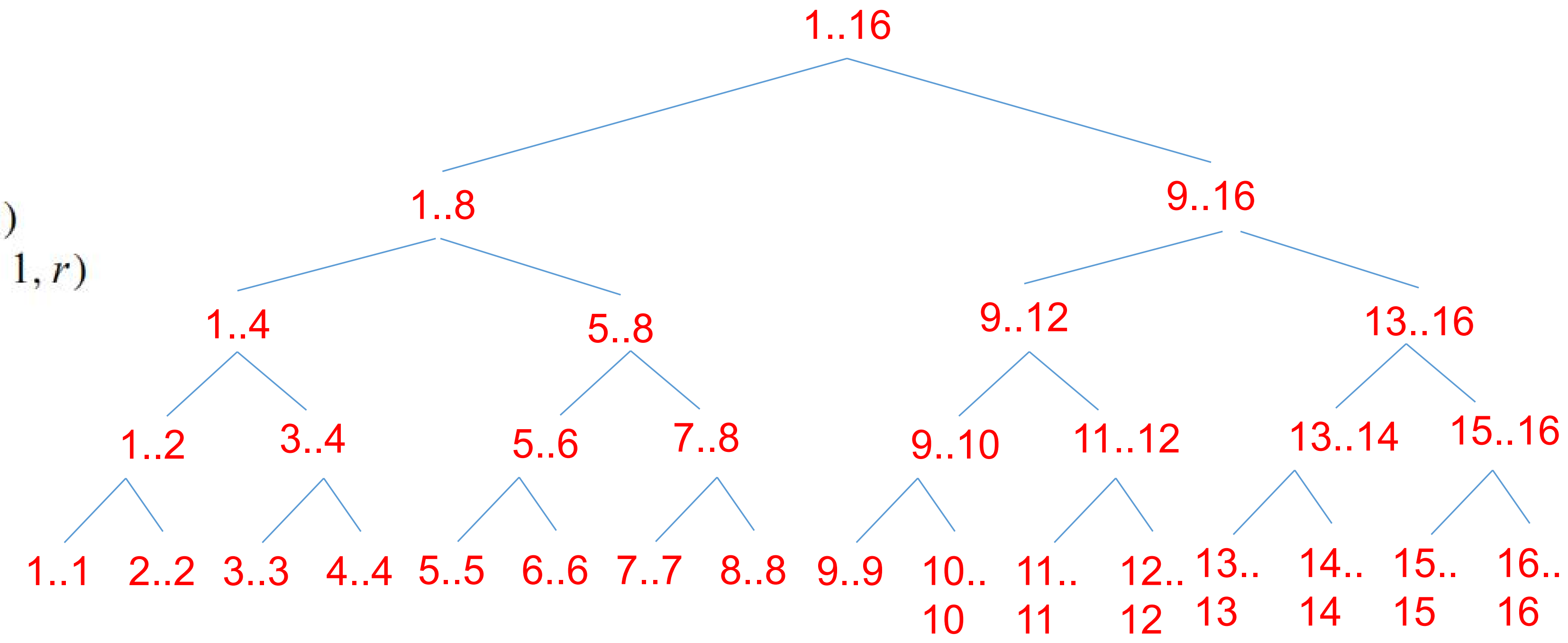
Figure 14.3 The interval trichotomy for two closed intervals i and i'. **(a)** If i and i' overlap, there are four situations; in each, $low[i] \le high[i']$ and $low[i'] \le high[i]$. **(b)** The intervals do not overlap, and $high[i] < low[i']$. **(c)** The intervals do not overlap, and $high[i'] < low[i]$.

第15章

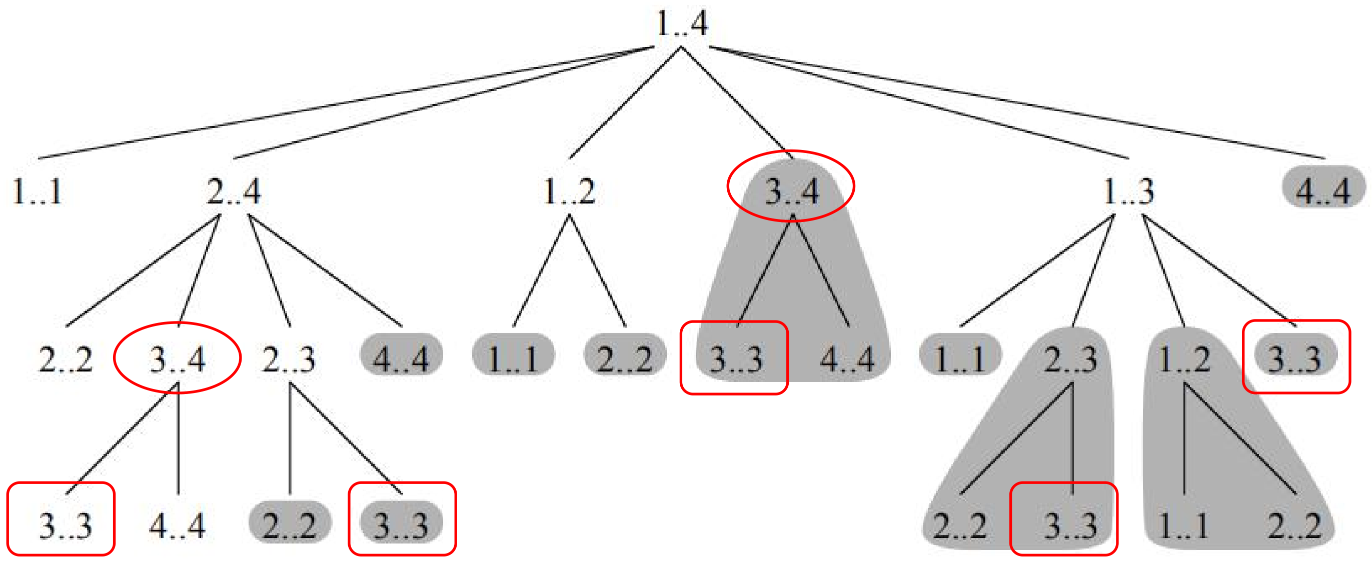
在此输入您的封面副标题

15.3-2 画出2.3.1节的过程MERGE-SORT作用于一个包含16个元素的数组上的递归树。请解释在加速一个好的分治算法如MERGE-SORT方面，做备忘录方法为什么没有什么效果？

```
MERGE-SORT(A, p, r)
if p < r
   then q ← ⌊(p + r)/2⌋
        MERGE-SORT(A, p, q)
        MERGE-SORT(A, q + 1, r)
        MERGE(A, p, q, r)
```



没有重叠子问题，不适合做备忘录方法



如果递归算法反复求解相同的子问题，我们称最优化问题具有重叠子问题

15.4-3 请给出一个LCS-LENGTH的运行时间为O(mn)的做备忘录版本。

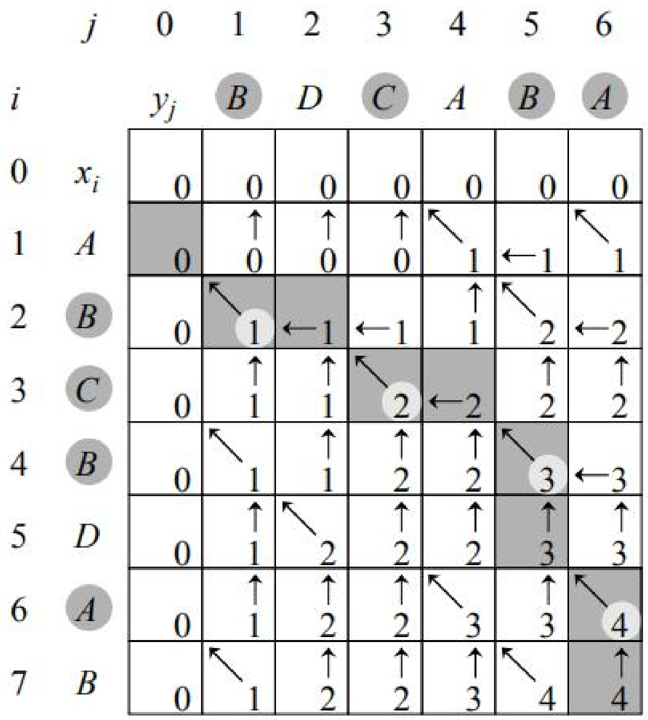
最长公共子序列问题：

$$c[i,j]=\begin{cases}0 & \text{如果 } i=0 \quad \text{或 } j=0\\ c[i-1,j-1]+1 & \text{如果 } i,j>0 \text{ 和 } x_i=y_j\\ \max(c[i,j-1],c[i-1,j]) & \text{如果 } i,j>0 \text{ 和 } x_i\neq y_j\end{cases}$$

MEMOIZED-LCS-LENGTH(X, Y, i, j) 假设：初始化c数组元素为无穷大

```
MEMOIZED-LCS-LENGTH(X, Y, i, j)
if i = 0 or j = 0
   then c[i, j] = 0

  else  if x_i = y_i
          then if c[i-1, j-1] = ∞
                 then MEMOIZED-LCS-LENGTH(X, Y, i-1, j-1)
               c[i, j] ← c[i-1, j-1] + 1
               b[i, j] ← '↖'
  else
        if c[i, j-1] = ∞
          then MEMOIZED-LCS-LENGTH(X, Y, i, j-1)
        if c[i-1, j] = ∞
          then MEMOIZED-LCS-LENGTH(X, Y, i-1, j)
        if c[i, j-1] ≥ c[i-1, j]
          then
               c[i, j] ← c[i, j-1]
               b[i, j] ← '←'
          else c[i, j] ← c[i-1, j]
               b[i, j] ← '↑'
```



15.5-2 对有n=7个关键字以及如下概率的集合，确定一棵最优二叉查找树的代价和结构：

	i	0	1	2	3	4	5	6	7
关键字：	p_i		0.04	0.06	0.08	0.02	0.10	0.12	0.14
伪关键字：	q_i	0.06	0.06	0.06	0.06	0.05	0.05	0.05	0.05

期望搜索代价的增加：

$$w(i,j)=\sum_{l=i}^{j} p_l+\sum_{l=i-1}^{j} q_l = w(i,r-1)+p_r+w(r+1,j)$$

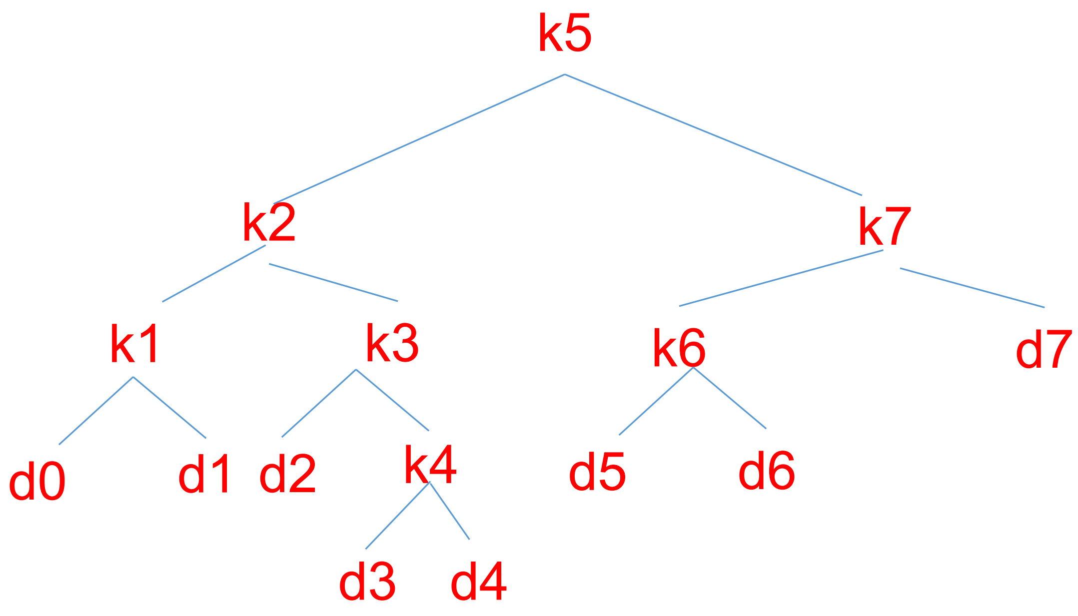
期望代价：

$$e[i,j]=\begin{cases} q_{i-1} & \text{如果 } j=i-1 \\ \min\limits_{i\leqslant r\leqslant j}\{e[i,r-1]+e[r+1,j]+w(i,j)\} & \text{如果 } i\leqslant j \end{cases}$$

```
OPTIMAL_BST(p,q,n)
    for i=1 to n+1     //初始化e和w的值
        do e[i,i-1] = qi-1;
           w[i,i-1] = qi-1;
     for l=1 to n
        do for i=1 to n-l+1
                do j=i+l-1;
                    e[i,j] = MAX;
                    w[i,j] = w[i,j-1]+pj+qj;
                    for r=i to j
                        do t=e[i,r-1]+e[r+1,j]+w[i,j]
                            if t<e[i,j]
                                then e[i,j] = t;
                                     root[i,j] = r;
return e and root;
```

r(e)表格如下：

j \ i	1	2	3	4	5	6	7	8
0	(0.06)							
1	1(0.28)	(0.06)						
2	2(0.62)	2(0.3)	(0.06)					
3	2(1.02)	3(0.68)	3(0.32)	(0.06)				
4	2(1.34)	3(0.93)	3(0.57)	4(0.24)	(0.05)			
5	3(1.83)	3(1.41)	4.5(1.04)	5(0.57)	5(0.3)	(0.05)		
6	3(2.44)	5(1.96)	5(1.48)	5(1.01)	6(0.72)	6(0.32)	(0.05)	
7	5(3.12)	5(2.61)	5(2.13)	6(1.55)	6(1.2)	7(0.78)	7(0.34)	(0.05)



期望代价（也可以直接查表得到）:

$$ET = 1 + \sum_{i=1}^{n} depth_i(k_i)p_i + \sum_{i=1}^{n} depth_i(d_i)q_i$$

$$= 1 + 2*0.04 + 1*0.06 + 2*0.08 + 3*0.02 + 0*0.10 + 2*0.12$$

$$+ 1*0.14 + 3*0.06 + 3*0.06 + 3*0.06 + 4*0.06 + 4*0.05 + 3*0.05$$

$$+ 3*0.05 + 2*0.05 = 3.12$$

15.5-3假设不维护表w[i,j],我们再OPTIMAL-BST的第8行直接从公式（15.17）计算w[i,j]的值,并在第10行用这个计算的值。这个改变对OPTIMAL-BST的渐进执行时间有什么影响?

$$w(i,j) = \sum_{l=i}^{j} p_l + \sum_{l=i-1}^{j} q_l \tag{15.17}$$

$O(n^3)=O(n)*O(n)*\left(O(n)+O(n)\right)$

两层循环

第16章

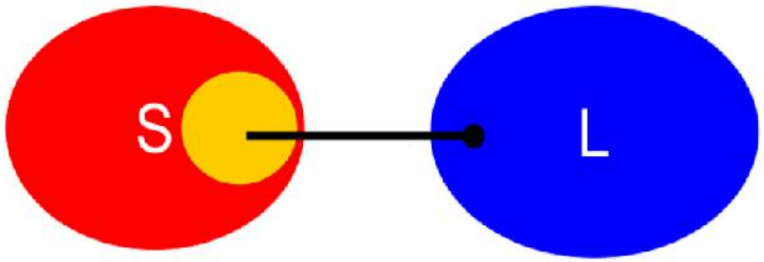
在此输入您的封面副标题

16.4-1 证明：（S，lk）为一个拟阵，其中S为任一有限集合，lk为S的所有大小至多为k的子集构成的集合，k≤|S|。

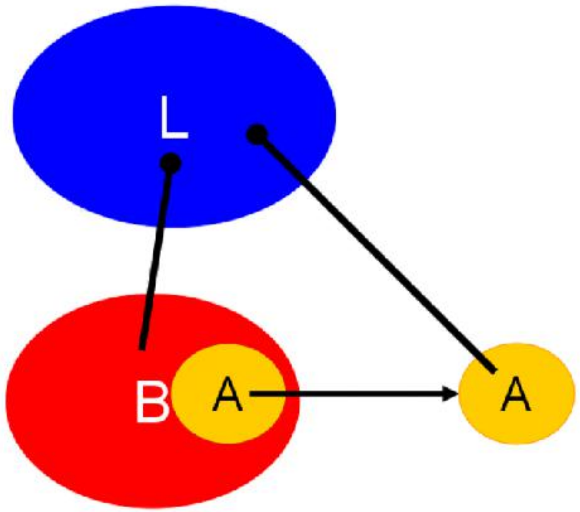
拟阵是一个二元组 M=(S,L)

1.S是一个有限集。

2. L是以集合作为元素的集合，且它的元素必须是S的子集。

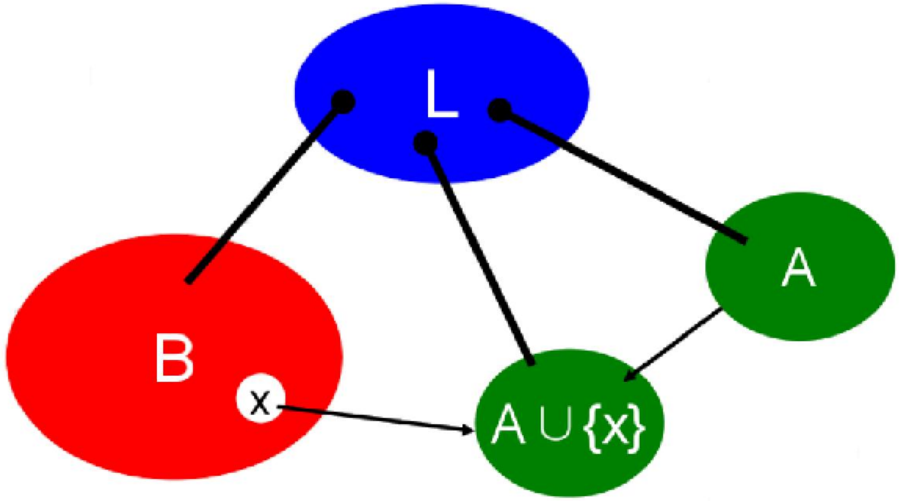


3.遗传性：对任意 $B \in L$ 任意 $A \subseteq B$ 有 $A \in L$。



4. 交换性：对任意 $A \in L, B \in L, |A|<|B|$

存在一个 $x \in B - A$，使 $A \cup \{x\} \in L$



证明满足3个特性，略

16.5-1 解图16-7中调度问题的实例，但要将每个惩罚W_i替换成$80-W_i$

单位时间任务最优调度问题：

M=(S单位时间任务集,L提前任务集)

	Task						
a_i	1	2	3	4	5	6	7
d_i	4	2	4	3	1	4	6
w_i	70	60	50	40	30	20	10
$80-W_i$	10	20	30	40	50	60	70
（$80-W_i$）$/d_i$	2.5	10	7.5	13.33	50	15	11.67

加权拟阵、贪心算法，求最大独立子集A:

```
GREEDY(M, w)
A ← ∅
sort S[M] into monotonically decreasing order by weight w
for each x ∈ S[M], taken in monotonically decreasing order by weight w(x)
        do if A ∪ {x} ∈ ℓ[M]
                then A ← A ∪ {x}
return A
```

最优调度： $<a_5,a_4,a_6,a_3,a_7,a_2,a_1>$ 总惩罚：$W_1+W_2=30$

16.5-2 说明如何利用引理16.12的性质2在O(|A|)时间内确定一个给定的任务集A是否是独立的。

对于一个任务集合A, 如果存在一个调度方案，使A中所有任务都不延迟，则称A是独立的。

引理 16.12 对任意的任务集合 A，下列命题是等价的：

1)集合 A 是独立的；

2)对 $t=0, 1, 2, \cdots, n$，有 $N_t(A)\leqslant t$。

3)如果对 A 中任务按期限的单调递增的顺序进行调度，则没有一个任务是迟的。

判断A是否独立：

① A中为单位时间任务。

② 16.12性质2　即，$N_t(A)$（在集合A中的任务个数）的deadline是t。

且：t≤d（任务集中的deadline）

③ A是独立集合，则$|A|=N_t(A)=O(|A|)$

第17章

在此输入您的封面副标题

17.1-1 如果一组栈操作中包括了一次MULTIPUSH操作，它一次把k个元素压入栈内，那么栈操作的平摊代价的界O(1)是否还能保持？

不能保持。

对于MULTIPUSH, n个操作的序列最坏情况下花费总时间T(n)=O(nk)

摊还代价为T(n)/n,为 O(k)。

17.1-2 证明：在k位计数器的例子中，如果包含一个DECREMENT操作，n个操作可能花费Θ（nk）时间

计数器初始状态为全0.假设有以下操作序列：

DECREMENT
INCREMENT
DECREMENT
INCREMENT
……

每次操作都会有k次的翻转，一共进行n次操作即可。

最坏情况例如：

$2^k - 1 + 1 - 1 + 1 - \ldots$。

17.2-1 对一个大小适中不超过k的栈上执行一系列的栈操作。在每k个操作后，复制整个栈的内容以留作备份。证明：在对各种栈操作赋予合适的平摊代价后，n个栈操作（包括复制栈的操作）的代价为O(n)。

核算法：

COPY	POP	PUSH
0	0	3

POP+PUSH (2元)

3元=消耗1元+预留2元

(1元)

17.3-1 假设有势函数φ，使得对所有的i都有φ(Di)⩾φ(D0),但是φ(D0)≠0。证明：存在一个势函数φ'使得φ'(D0)=0，φ'(Di)⩾0，对所有i⩾1，且用φ'表示的平摊代价与用φ表示的平摊代价相同。

令 $\Phi'(D_i)=\Phi(D_i)-\Phi(D_0)$

平摊代价: $\sum_{i=1}^{n}\hat{c}_i=\sum_{i=1}^{n}c_i+\Phi'(D_n)-\Phi'(D_0)$

$=\sum_{i=1}^{n}c_i+\Phi(D_n)-\Phi(D_0)$

17.3-4 假设某个栈在执行 n 个栈操作 PUSH、POP 和 MULTIPOP 之前包含 s_0 个对象，结束后包含 s_n 个对象，则这 n 个栈操作的总代价是多少？

据P263计算，第i次操作，PUSH的摊还代价为2，MULTIPOP 和 POP的摊还代价为0

对于1≤i≤n, 有 $\hat{c}_i \le 2$

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0)$$

$$\le 2n + s_0 - s_n$$

17.4-2 证明：如果作用于一动态表上的第 i 个操作是 TABLE-DELETE，且 $\alpha_{i-1} \geq 1/2$，则以势函数(17.6)表示的每个操作的平摊代价由一个常数从上方限界。

$$\Phi(T) = \begin{cases} 2 \cdot T.num - T.size & 若\ \alpha(T) \geq 1/2 \\ T.size/2 - T.num & 若\ \alpha(T) < 1/2 \end{cases}$$

α_{i-1}>=1/2,因此第i个操作TABLE-DETLETE不会引起收缩，c_i=1,s_i=s_{i-1}

如果 $\alpha_i \geq 1/2$:

$$\begin{aligned} \hat{c}_i &= c_i + \Phi_i - \Phi_{i-1} \\ &= 1 + (2n_i - s_i) - (2n_{i-1} - s_{i-1}) \\ &= 1 + (2(n_{i-1} - 1) - s_{i-1}) - (2n_{i-1} - s_{i-1}) \\ &= -1 \end{aligned}$$

如果 $\alpha_i < 1/2$:

$$
\begin{aligned}
\hat{c_i} &= c_i + \Phi_i - \Phi_{i-1} \\
&= 1 + (s_i/2 - n_i) - (2n_{i-1} - s_{i-1}) \\
&= 1 + (s_{i-1}/2 - (n_{i-1} - 1)) - (2n_{i-1} - s_{i-1}) \\
&= 2 + \frac{3}{2}s_{i-1} - 3n_{i-1} \\
&= 2 + \frac{3}{2}s_{i-1} - 3\alpha_{i-1}s_{i-1} \\
&\leqslant 2 + \frac{3}{2}s_{i-1} - \frac{3}{2}s_{i-1} \\
&= 2
\end{aligned}
$$

17.4-3 假设当某个表的装载因子降至 1/4 以下时，我们不是通过将其大小缩小一半来收缩，而是在表的装载因子低于 1/3 时，通过将其大小乘以 2/3 来进行收缩。利用势函数

$$\Phi(T) = |2 \cdot num[T] - size[T]|$$

来证明采用这种策略的 TABLE-DELETE 操作的平摊代价由一个常数从上方限界。

$$\text{load factor} = \frac{\text{number of the element in a dynamic table}}{\text{size of the dynamic table}}$$

$$\phi(T) = \begin{cases} 2 \times T.num - T.size & \text{if } \alpha(T) \geq 1/2 \\ T.size/2 - T.num & \text{if } \alpha(T) < 1/2 \end{cases}$$

$$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1}$$

- case1：$\alpha_{i-1}>=1/2,\alpha_i>=1/2$,不收缩，平摊代价-1
- case2：$\alpha_{i-1}>=1/2,\alpha_i<1/2$,不收缩，平摊代价$<=3$
- case3：$1/3<\alpha_{i-1}<1/2,1/3<=\alpha_i<1/2$,不收缩，平摊代价3
- case4：$1/3<=\alpha_{i-1}<1/2,\alpha_i><1/3$,收缩，平摊代价$<3$

- 综上，平摊代价的上界为常数3

第20章

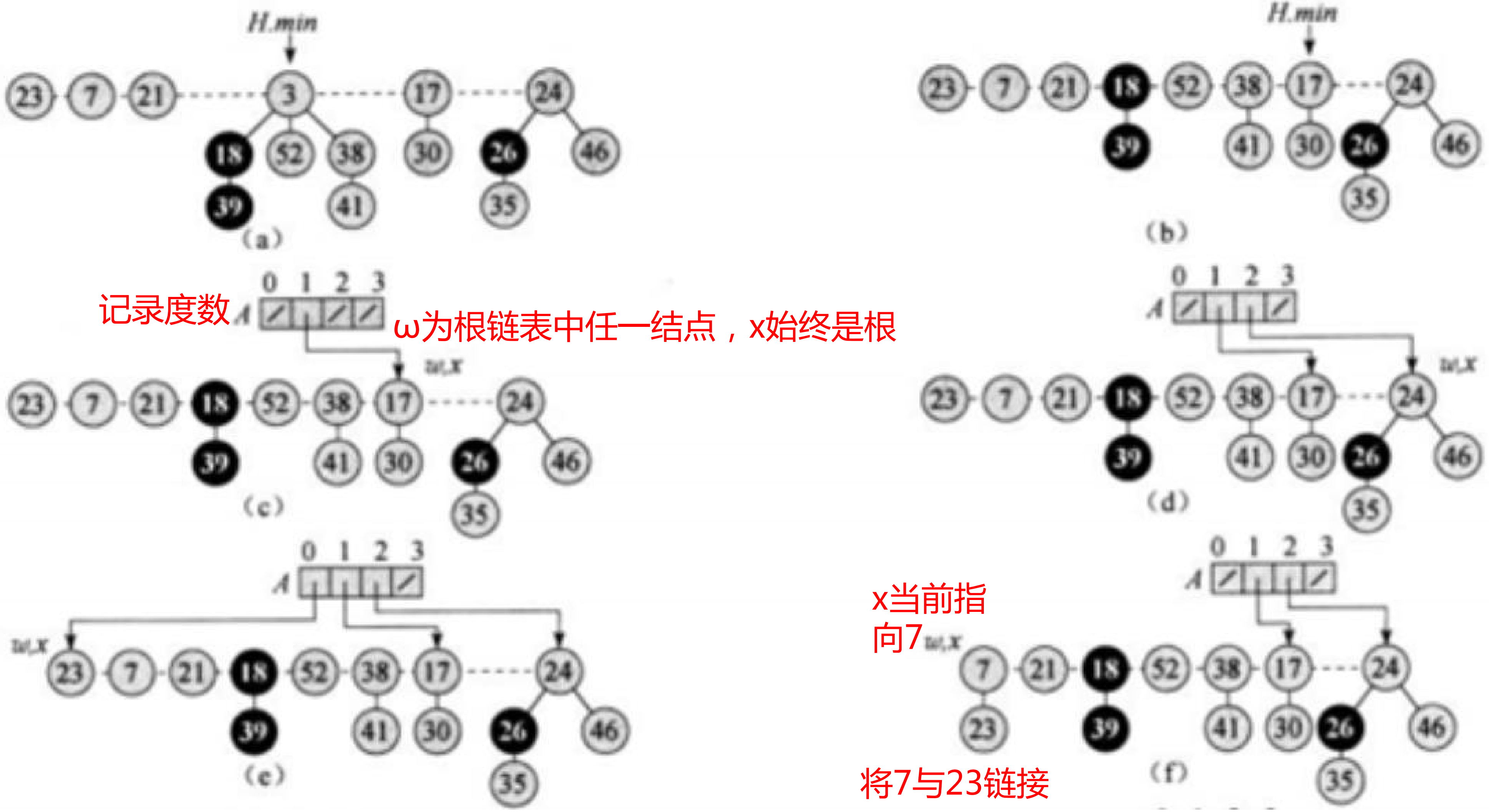
在此输入您的封面副标题

20.2-1 请给出对图 20-3m 中所示的斐波那契堆调用 FIB-HEAP-EXTRACT-MIN 后得到的斐波那契堆。

```
FIB-HEAP-EXTRACT-MIN(H)
z ← min[H]
if z ≠ NIL
   then for each child x of z
           do add x to the root list of H
              p[x] ← NIL
        remove z from the root list of H
        if z = right[z]
           then min[H] ← NIL
           else min[H] ← right[z]
                CONSOLIDATE(H)
        n[H] ← n[H] − 1
return z
```

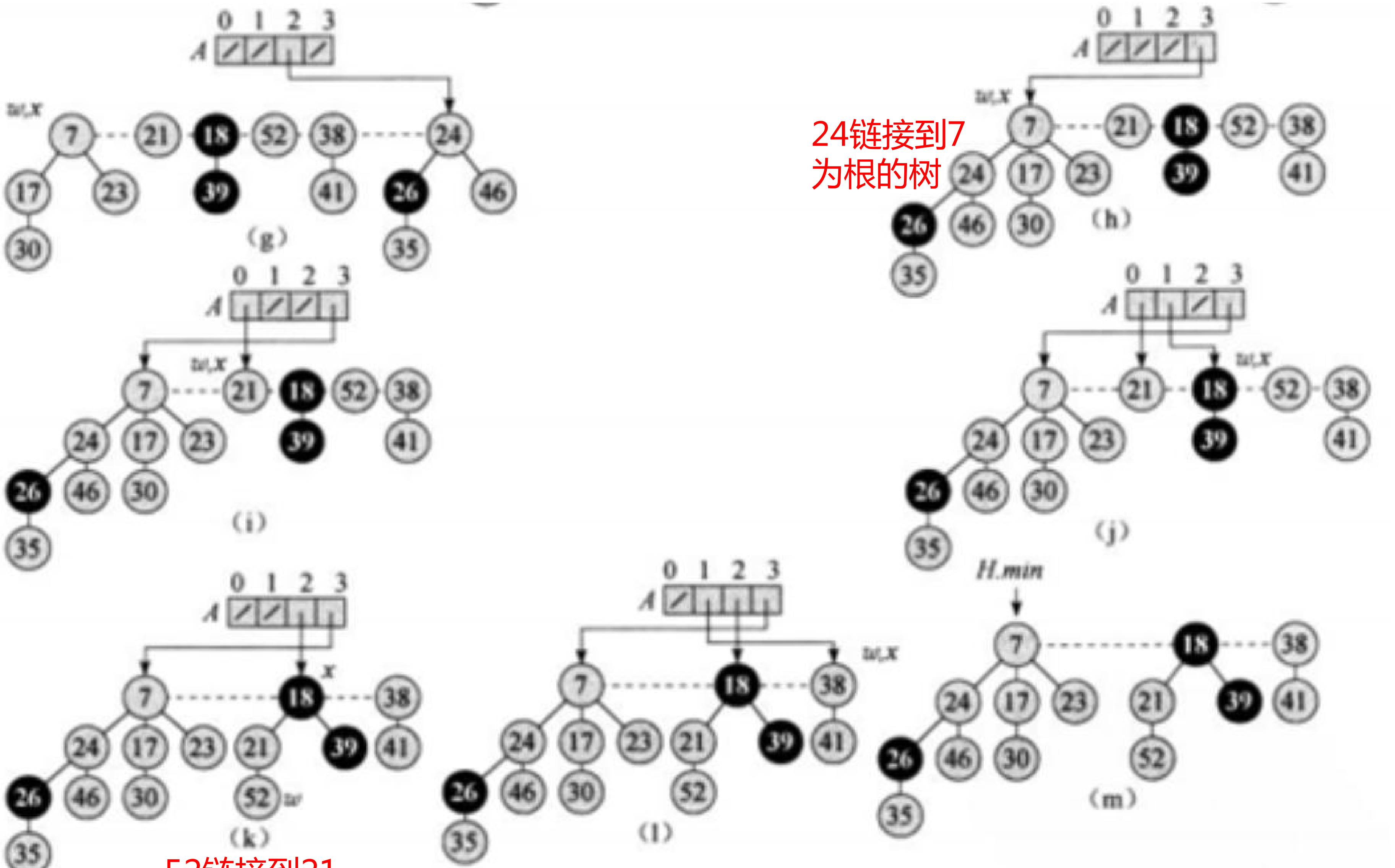
CONSOLIDATE(*H*) 合并根链表

```
for i ← 0 to D(n[H])
    do A[i] ← NIL
for each node w in the root list of H
    do x ← w
       d ← degree[x]
       while A[d] ≠ NIL
           do y ← A[d]          ▷ Another node with the same degree as x.
              if key[x] > key[y]
                 then exchange x ↔ y
              FIB-HEAP-LINK(H, y, x)
              A[d] ← NIL
              d ← d+1
         A[d] ← x
min[H] ← NIL
for i ← 0 to D(n[H])
    do if A[i] ≠ NIL
          then add A[i] to the root list of H
               if min[H] = NIL or key[A[i]] < key[min[H]]
                  then min[H] ← A[i]
```

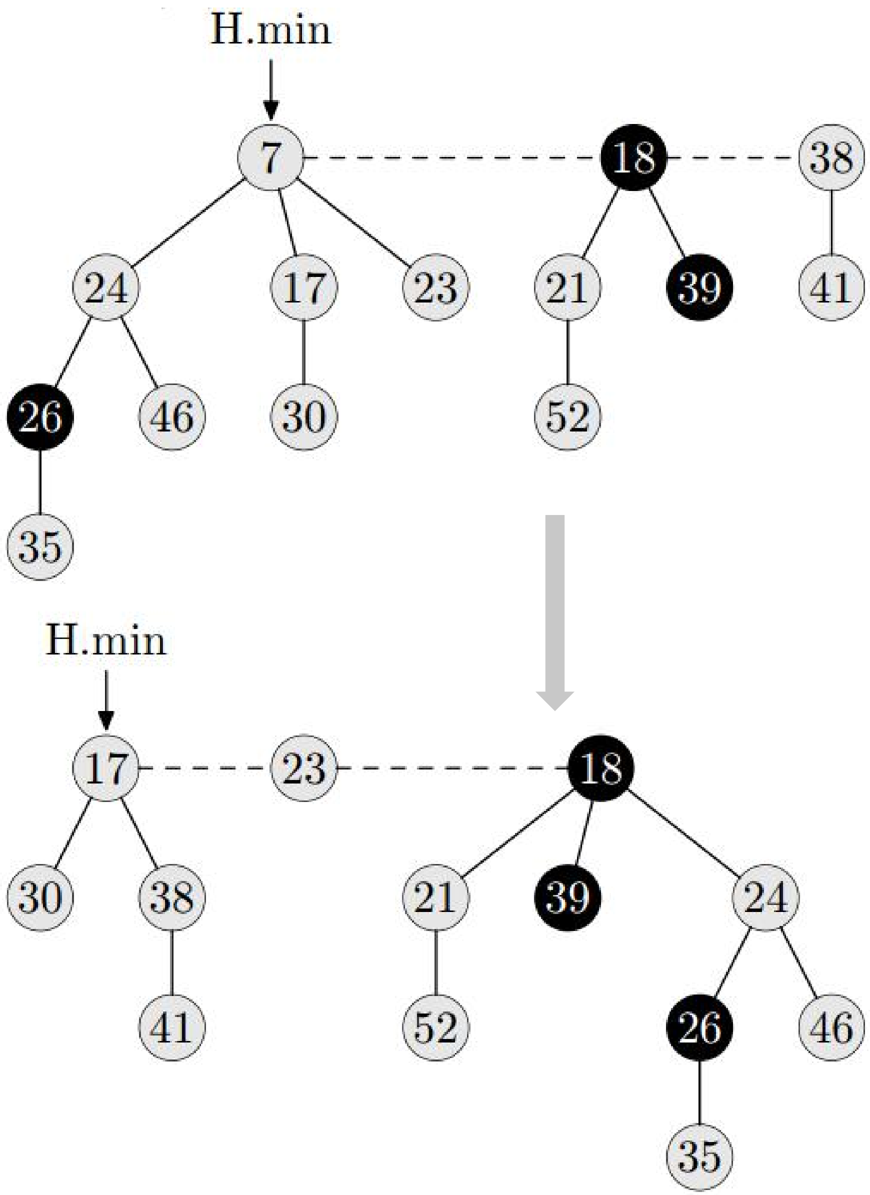

H.min
(a)
(b)
记录度数
ω为根链表中任一结点，x始终是根
(c)
(d)
(e)
x当前指
向7
将7与23链接
(f)

执行while
循环，将
17链接到
7为根的树

24链接到7
为根的树



52链接到21，
然后接到18


H.min
7
18
38
24
17
23
21
39
41
26
46
30
52
35
H.min
17
23
18
30
38
21
39
24
41
52
26
46
35

20.2-3 证明：如果仅需支持可合并堆操作，则在包含 n 个结点的斐波那契堆中结点的最大度数 $D(n)$ 至多为 $\lfloor \lg n \rfloor$。

如果仅需支持可合并堆操作，Fibonacci heap是一组无序的二项树，因而根据二项树的性质可以容易得证

引理 19.1(二项树的性质) 二项树 B_k 具有以下的性质：

1)共有 2^k 个结点，

2)树的高度为 k，

3)在深度 i 处恰有 $\binom{k}{i}$ 个结点，其中 $i=0, 1, 2, \cdots, k$，

4)根的度数为 k，它大于任何其他结点的度数；并且，如果根的子女从左到右编号为 $k-1, k-2, \cdots, 0$，子女 i 是子树 B_i 的根。

推论 19.2 在一棵包含 n 个结点的二项树中，任意结点的最大度数为 $\lg n$。

假设 $degree[x]=k$

$$F_k = \begin{cases} 0 & \text{如果 } k=0 \\ 1 & \text{如果 } k=1 \\ F_{k-1}+F_{k-2} & \text{如果 } k\geqslant 2 \end{cases}$$

在练习 3.2-7 中 $F_{k+2} \geqslant \phi^k$

设 s_k 为所有满足 $degree[z]=k$ 的结点 z 中，size(z)的最小可能值。$\phi=(1+\sqrt{5})/2$,

引理 20.3 size(x) $\geqslant s_k \geqslant F_{k+2} \geqslant \phi^k$。

n个结点的斐波那契堆，任意结点x的度数为k，则

$n \ge \mathrm{size}(x) \ge \Phi^k$

$k \le \log_\Phi n$,又因为k为整数，$k \le \lfloor \log_\Phi n \rfloor$

20.3-1 假设一个斐波那契堆中的某个根 x 是有标记的。请解释 x 是如何成为有标记的根。另说明 x 有无标记对分析来说没有影响，即使它不是先被链接到另一个结点，然后又失去一个子结点的根。

x失去一个孩子，则mark=TRUE

势的改变至多为：

$$((t(H)+c)+2(m(H)-c+2))-(t(H)+2m(H))=4-c$$

这样，FIB-HEAP-DECREASE-KEY 的平摊代价至多为

$$O(c)+4-c=O(1)$$

20.3-2 使用聚集分析来证明 FIB-HEAP-DECREASE-KEY 的平摊时间 $O(1)$ 是每个操作的平均代价。

据P300（第三版）:

FIB-HEAP-DECREASE-KEY需要的摊还代价为O(1)

假设其中调用了c次CASCADING-CUT，每次需要O(1)的时间

因此，c+1次操作中，实际代价为O(C)

平均代价为O(C)/(C+1)=O(1)