2.3-3

数学归纳法证明即可，略（注：几乎所有人都对）

2.3-4

下面是最坏情况下的 T(n)

$$T(n)=\begin{cases}\theta(1), n\le c(c\text{为常量})\\ T(n-1)+\theta(n), \text{否则}\end{cases}$$

3.1-1

证明：只需找出 c_1, c_2, n_0,使得

$0 <= c_1 * (f(n) + g(n)) <= max(f(n), g(n)) <= c_2 * (f(n)+g(n))$

取 $c_1=0.5$, $c_2=1$，由于 f(n),g(n)是非负函数，所以在 n>=0 时恒成立，所以得证。

3.1-8 参照写定义即可，略（注：几乎所有人都对）

4.1-1 （注意：上面的证明用到了课本 p51 的公式（3.4~3.7））

因为有一个常数 1 在里面，所以在使用替换方法的时候，考虑可能要证明下面的式子：

$T(n)\le c\lg(n-b)$，假设该不等式对$\lceil n/2\rceil$成立，那么

$$T(n)=T(\lceil n/2\rceil)+1\le c\lg(\lceil n/2\rceil-b)+1\le c\lg((n+1)/2-b)+1$$
$$=c\lg((n-b+1-b)/2)+1$$
$$=c\lg(n-b+1-b)+1-c$$
$$\le c\lg(n-b)$$

最后一个不等式成立的条件是$1-c\le 0$并且$1-b\le 0$，即$c\ge 1$ and $b\ge 1$

4.1-2

使用替换方法，要证明$\Omega(n\lg n)$,即要证明$T(n)\ge c(n+b)\lg(n+b)$

假设上面的不等式对$\lfloor n/2\rfloor$成立，那么

$$T(n)\ge c(n+b)\lg(n+b)$$
$$T(n)\ge 2c(\lfloor n/2\rfloor+b)\lg(\lfloor n/2\rfloor+b)+n\ge c(n-1+2b)\lg((n-1)/2+b)+n$$
$$=c(n+b+b-1)\lg((n+b+b-1)/2)+n=c(n+b+b-1)\lg(n+b+b-1)+(1-c)n-c(1-2b)$$
$$\ge c(n+b)\lg(n+b)$$

$b-1\ge 0, 1-c\ge 0$且$(1-c)n-c(1-2b)\ge 0$最后一个不等式成立。最终可知$\theta(n\lg n)$成立

注：题目中的要求使用递归树的方法，最好是像书上画一颗递归树然后进行运算。

4.2-1 答案：$\Theta(n^{\lg 3})$

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/2 \rfloor) + n \\
&\leqslant n + (3/2)n + (3/2)^2 n + \cdots + (3/2)^{\lg n - 1} n + \Theta(n^{\lg 3}) \\
&= n \sum_{i=0}^{\lg n - 1} (3/2)^i + \Theta(n^{\lg 3}) \\
&= n \cdot \frac{(3/2)^{\lg n} - 1}{(3/2) - 1} + \Theta(n^{\lg 3}) \\
&= 2(n(3/2)^{\lg n} - n) + \Theta(n^{\lg 3}) \\
&= 2n\frac{3^{\lg n}}{2^{\lg n}} - 2n + \Theta(n^{\lg 3}) \\
&= 2 \cdot 3^{\lg n} - 2n + \Theta(n^{\lg 3}) \\
&= 2n^{\lg 3} - 2n + \Theta(n^{\lg 3}) \\
&= \Theta(n^{\lg 3})
\end{aligned}
$$

assumming that $T(\lfloor n/2 \rfloor) \leqslant c\lfloor n/2 \rfloor^{\lg 3} - c\lfloor n/2 \rfloor$

$$
\begin{aligned}
T(n) &= 3T(\lfloor n/2 \rfloor) + n \\
&\leqslant 3c\lfloor n/2 \rfloor^{\lg 3} - c\lfloor n/2 \rfloor + n \\
&\leqslant \frac{3cn^{\lg 3}}{2^{\lg 3}} - \frac{cn}{2} + n \\
&\leqslant cn^{\lg 3} - \frac{cn}{2} + n \\
&\leqslant cn^{\lg 3}
\end{aligned}
$$

$c \geqslant 2$.

注意题目中的要求使用递归树的方法，最好是像书上画一颗递归树然后进行运算。

4.2.2 证略

4.2.3

由 $2^i = n$ 得 i=lgn

$$T(n) = \sum_{i=0}^{\lg n} 2^i cn = cn\frac{2^{\lg n+1} - 1}{2 - 1} = 2cn^2 - cn = \theta(n^2)$$

4.3-1

a) n^2

b) $n^2 \lg n$

c) n^3

4.3-4

$m^2 \lg^2 n$

7.1-2

(1)使用 P146 的 PARTION 函数可以得到 q=r

注意每循环一次 i 加 1，i 的初始值为 $p-1$，循环总共运行 $(r-1)-p+1$ 次，最终返回的 $i+1=p-1+(r-1)-p+1+1=r$

(2)由题目要求 q=(p+r)/2 可知，PARTITION 函数中的 i，j 变量应该在循环中同时变化。

```
Partition(A, p, r)
    x = A[p];
    i = p - 1;
    j = r + 1;
    while (TRUE)
        repeat
            j--;
        until A[j] <= x;
        repeat
            i++;
        until A[i] >= x;
        if (i < j)
            Swap(A, i, j);
        else
            return j;
```

7.3-2

（1）由 QuickSort 算法最坏情况分析得知：n 个元素每次都划 n-1 和 1 个，因为是 p<r 的时候才调用，所以为Θ（n）

（2）最好情况是每次都在最中间的位置分，所以递推式是：

N(n)= 1+ 2*N(n/2)

不难得到：N(n) =Θ（n）

7.4-2

T(n)=2*T(n/2)+ Θ（n）

可以得到 T(n) =Θ（n *lg*n）

由 P46 Theorem3.1 可得：Ω (n lgn)

13.1-5

prove:

By property 5 the longest and shortest path must contain the same number of black nodes. By property 4 every other nodes in the longest path must be black and therefore the length is at most twice that of the shortest path.

13.1-6

2^k-1　$2^{2k}-1$

13.2-3

a 的深度增加 1，b 的深度不变, c 的深度减少 1

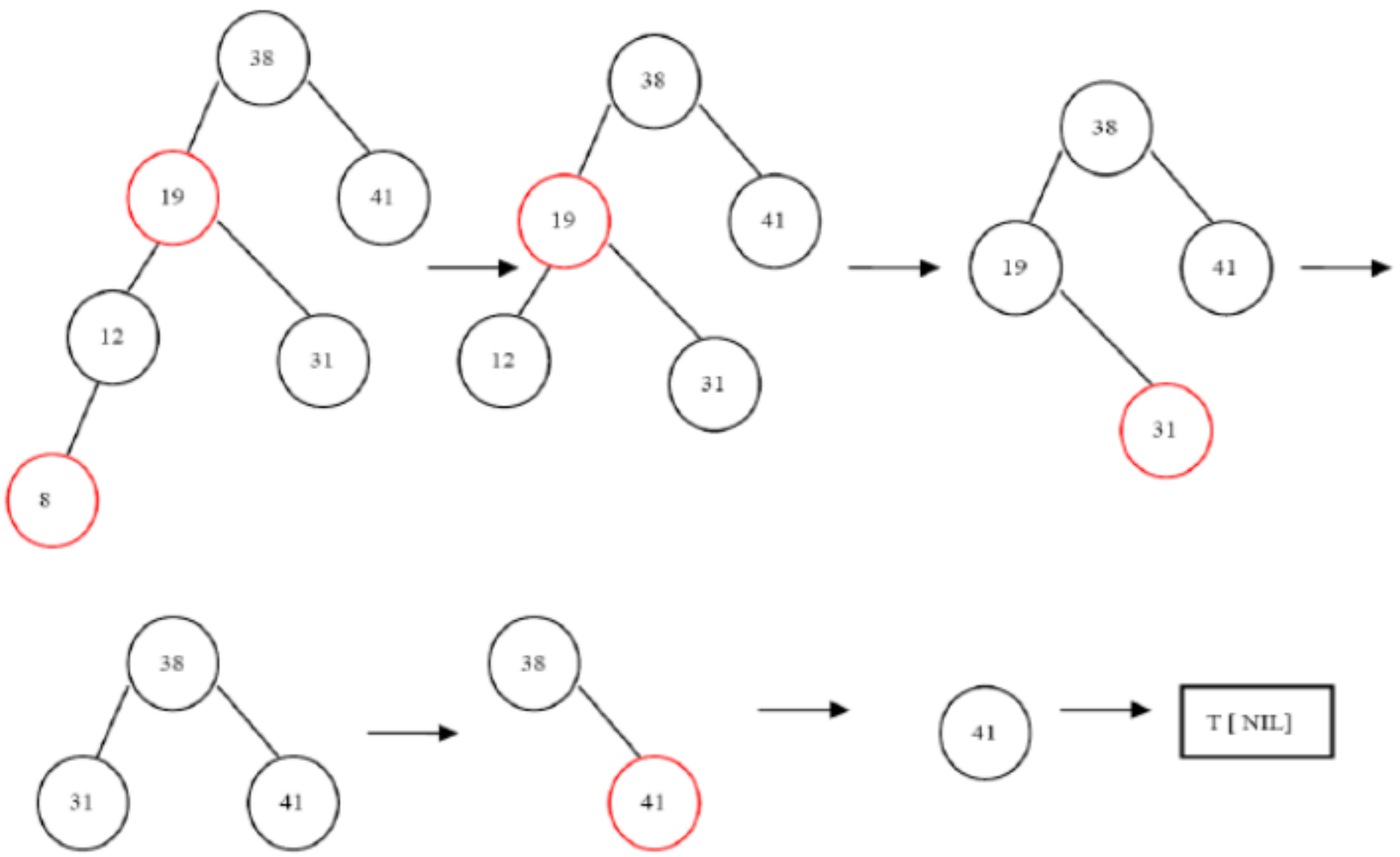
13.3-5

采用反证法

证明：假设 n>1 时，树中没有红色结点。

那么当树中有 n-1 个结点时，由假设可知仍没有红色结点。当用 RB-INSERT 算法插入第 n 个结点时，在 RB-INSERT 的 16 行，执行了 color[z]<- RED。也就是当前插入的第 n 个结点是红色的。而在 RB-INSERT-FIXUP 中的 while color[p[z]]= RED 这个循环不执行，因为由假设前 n-1 结点都是黑色的。所以算法直接执行 RB-INSERT-FIXUP 的第 16 行就结束了。所以算法结束的时候插入的第 n 个结点还是红色的。而这与假设相矛盾，所以可以证明当 n>1 的时候，树中至少有一个红色结点。

13.4-3



解释：箭头代表一步

第一步不执行 RB-DELETE-FIXUP

第二步符合 RB-DELETE-FIXUP 的 case 2

第三步不执行 RB-DELETE-FIXUP 的 while 循环，直接执行步骤 23

第四步符合 RB-DELETE-FIXUP 的 case 2

第五步不执行 RB-DELETE-FIXUP 的 while 循环，直接执行步骤 23

第六步得到空树

13.4-6 （NOTE：Page275 图 13.1(a)，一个节点可以有两个红色的子女（节点 35，39））
由 RB-DELETE 可以知道：y 为被删除的节点，x 为 y 的子女。
p[x]为 x 的父节点。p[x]可能为红色吗？
假设 p[x]为红色的，那么 RB-DELETE-FIXUP 中 x 的兄弟节点 w 一定不能是红色的。因为由红黑树的性质 4（Page 273），一个红色的节点只能有两个黑色的孩子。所以在 p[x]为红色的，x 的兄弟节点 w 为黑色的情况下，RB-DELETE-FIXUP 的 5——8 行（case1）不会执行。因此，一旦 RB-DELETE-FIXUP 的 5—6 行(case1)能够执行，p[x]肯定是黑色的。

14.1-4

题目要求：写一个 Page305，OS-RANK 算法效果差不多的递归版本
分析：一个元素的秩是它在集合的线性序中的位置。可以看作是在对树进行中序遍历中排在它之前的节点的个数再加 1。

```
OS-KEY-RANK (T, k)
if key [root [T]] = k
   then return size [left [root [T] ] ] +1
   else if key [root [T] ] > k
          then return OS-KEY-RANK (left[T], k)
          else return size [left [root [T] ] ] +1+OS-KEY-RANK (right[T], k)
```

14.2-2

题目要求：能否将红黑树中节点的黑高度作为一个域来进行有效的维护？
可以，因为一个节点的黑高度可以从该节点和它的两个孩子的信息计算得到。
做法：如果一个节点的孩子的黑高度是 n，并且该孩子的颜色为黑色，则该节点的黑高度为 n+1，否则为 n。
根据 Page309 的定理 14.1，插入和删除操作仍然可以在 O（lg n）的时间内实现。

14.2-3

不可以，性能改变 时间复杂度由 O(lgn) -> O(nlgn)

14.3-2

Note: 注意 Overlap 的定义稍有不同，需要重新定义。
算法：只要将 P314 页第三行的≥改成>就行。

14.3-3

INTERVAL-SEARCH-SUBTREE(*x*, *i*)
1 **while** *x* ≠ *nil*[*T*] **and** *i* **does not overlap** *int*[*x*]
2 **do if** *left*[*x*] ≠ *nil*[*T*] **and** *max*[*left*[*x*]] ≥ *low*[*i*]
3 **then** *x* ← *left*[*x*]
4 **else** *x* ← *right*[*x*]
5 **return x**

INTERVAL-SEARCH-MIN(*T*, *i*)
2 y←INTERVAL-SEARCH-SUBTREE(root[*T*], *i*) ▷先找第一个重叠区间
3 z←y
4 **while** y≠ *nil*[*T*] **and** *left*[*y*] ≠ *nil*[*T*] ▷在它的左子树上查找

```
5    do  z←y                                          ▷ 调用之前保存结果
6        y←INTERVAL-SEARCH-SUBTREE(y, i)
▷ 如果循环是由于y没有左子树，那我们返回y
▷ 否则我们返回z，这时意味着没有在z的左子树找到重叠区间
7 if y≠ nil[T] and i overlap int[y]
8    then return y
9    else return z
```

15.1-5

由 FASTEST-WAY 算法知：

$$f_1[j] = f_2[j-1] + t_{2,j-1} + a_{1,j}$$

$$f_2[j] = f_1[j-1] + t_{1,j-1} + a_{2,j}$$

所以有：

$$f_1[j] + f_2[j] = f_2[j-1] + t_{2,j-1} + a_{1,j} + f_1[j-1] + t_{1,j-1} + a_{2,j}$$

由课本 P328（15.6） (15.7)式代入，可得：

$$\min(f_1[j-1] + a_{1,j}, f_2[j-1] + t_{2,j-1} + a_{1,j}) + \min(f_2[j-1] + a_{2,j}, f_1[j-1] + t_{1,j-1} + a_{2,j})$$

$$= \min(f_1[j-1], f_2[j-1] + t_{2,j-1}) + a_{1,j} + \min(f_2[j-1], f_1[j-1] + t_{1,j-1}) + a_{2,j}$$

$$= f_2[j-1] + t_{2,j-1} + a_{1,j} + f_1[j-1] + t_{1,j-1} + a_{2,j}$$

化简得：

$$\min(f_1[j-1], f_2[j-1] + t_{2,j-1}) + \min(f_2[j-1], f_1[j-1] + t_{1,j-1})$$

$$= f_2[j-1] + t_{2,j-1} + f_1[j-1] + t_{1,j-1}$$

而：

$$\min(f_1[j-1], f_2[j-1] + t_{2,j-1}) \le f_2[j-1] + t_{2,j-1}$$

$$\min(f_2[j-1], f_1[j-1] + t_{1,j-1}) \le f_1[j-1] + t_{1,j-1}$$

等号在：

$$f_2[j-1] = f_1[j-1] + t_{1,j-1}$$

$$f_1[j-1] = f_2[j-1] + t_{2,j-1}$$

时成立，但是 $t_{2,j-1}$， $t_{1,j-1}$ 不为负数，矛盾。

15.2-1

参照 P336 算法，P337 例子就可以算得答案： 2010

Solve the matrix chain order for a specific problem. This can be done by computing MATRIX-CHAIN-ORDER(p) where $p = \langle 5, 10, 3, 12, 5, 50, 6 \rangle$ or simply using the equation:

$$m[i,j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \leqslant k < j}\{m[i,k] + m[k+1,j] + p_{i-1}p_k p_j\} & \text{if } i < j \end{cases}$$

The resulting table is the following:

i\j	1	2	3	4	5	6
1	0	150	330	405	1655	2010
2		0	360	330	2430	1950
3			0	180	930	1770
4				0	3000	1860
5					0	1500
6						0

The table is computed simply by the fact that $m[i,i] = 0$ for all i. This information is used to compute $m[i, i+1]$ for $i = 1, \ldots n-1$ an so on.

最终答案：$((A_1A_2)((A_3A_4)(A_5A_6)))$

15.2-2

```
MATRIX-CHAIN-MULTIPLY (A, s, i, j)
if (i = j)
        return A[i]
if (j =i+1)
        return MATRIX-MULTIPLY (A[i], A[j])
else
        B1 =MATRIX-CHAIN-MULTIPLY (A, s, i, S[i,j]);
        B2= MATRIX-CHAIN-MULTIPLY (A, s, S[i,j]+1, j);
        return MATRIX-MULTIPLY (B1, B2);
```

15.3-2 没有重叠的子问题存在

15.3-4

这题比较简单,由 P328 的(15.6)(15.7)展开两三步就可以看出来.

15.4-3

先初始化 c 数组元素为无穷大

MEMOIZED-LCS-LENGTH(X, Y, i, j)

```
1  if i = 0 or j = 0
2      then c[i, j] = 0
```

```
 3     else  if x_i = y_i
 4             then if c[i-1, j-1] = ∞
 5                     then MEMOIZED-LCS-LENGTH(X, Y, i-1, j-1)
 6                  c[i, j] ← c[i-1, j-1] + 1
 7                  b[i, j] ←'↖'
 8     else
 9          if c[i, j-1] = ∞
10             then MEMOIZED-LCS-LENGTH(X, Y, i, j-1)
11          if c[i-1, j] = ∞
12             then MEMOIZED-LCS-LENGTH(X, Y, i-1, j)
13          if c[i, j-1] ≥ c[i-1, j]
14             then
15                  c[i, j] ← c[i, j-1]
16                  b[i, j] ←'←'
17             else c[i, j] ← c[i-1, j]
18                  b[i, j] ←'↑'
```

15.5-2

e[i,j]的值如下：

i \ j	0	1	2	3	4	5	6	7	8
7	3.12	2.61	2.13	1.55	1.20	0.78	0.34	0.05	
6	2.44	1.96	1.48	1.01	0.72	0.32	0.05		
5	1.83	1.41	1.04	0.57	0.30	0.05			
4	1.34	0.93	0.57	0.24	0.05				
3	1.02	0.68	0.32	0.06					
2	0.62	0.30	0.06						
1	0.28	0.06							
0	0.06								

root的值为：

i \ j	1	2	3	4	5	6	7
7	5	5	5	6	6	7	7
6	3	3	4	4	6	6	
5	3	3	4	5	5		
4	2	3	3	4			
3	2	3	3				
2	2	2					
1	1						

由root的值很容易画出最优二叉查找树。

15.5-3

$O(n^3)$

P360 上面的（15.17）式花费的时间是 O(n)。注意算法 OPTIMAL-BST，虽然将 w[i,j]用 15.17 式计算，但是这个结果的 w[i,j]可以用于第 10 行中的计算，因此总的来说，时间复杂度仍然为 $O(n)* O(n)*(O(n)+O(n))=O(n^3)$.

16.1-2

仿照 page374 定理 16.1 的证明方法

$$f_m = \max\{s_k : a_k \in S_{ij}\}$$

直觉上，正好是书中 figure16.1 的逆过程，对应的式子 6.1 变为

$$s_0 \geq s_1 \geq s_2 \ldots \geq s_n \geq s_{n+1}$$

也就是把活动按照开始时间的从大到小的顺序先排列好

figure16.1 的结果集合中各元素的顺序变为

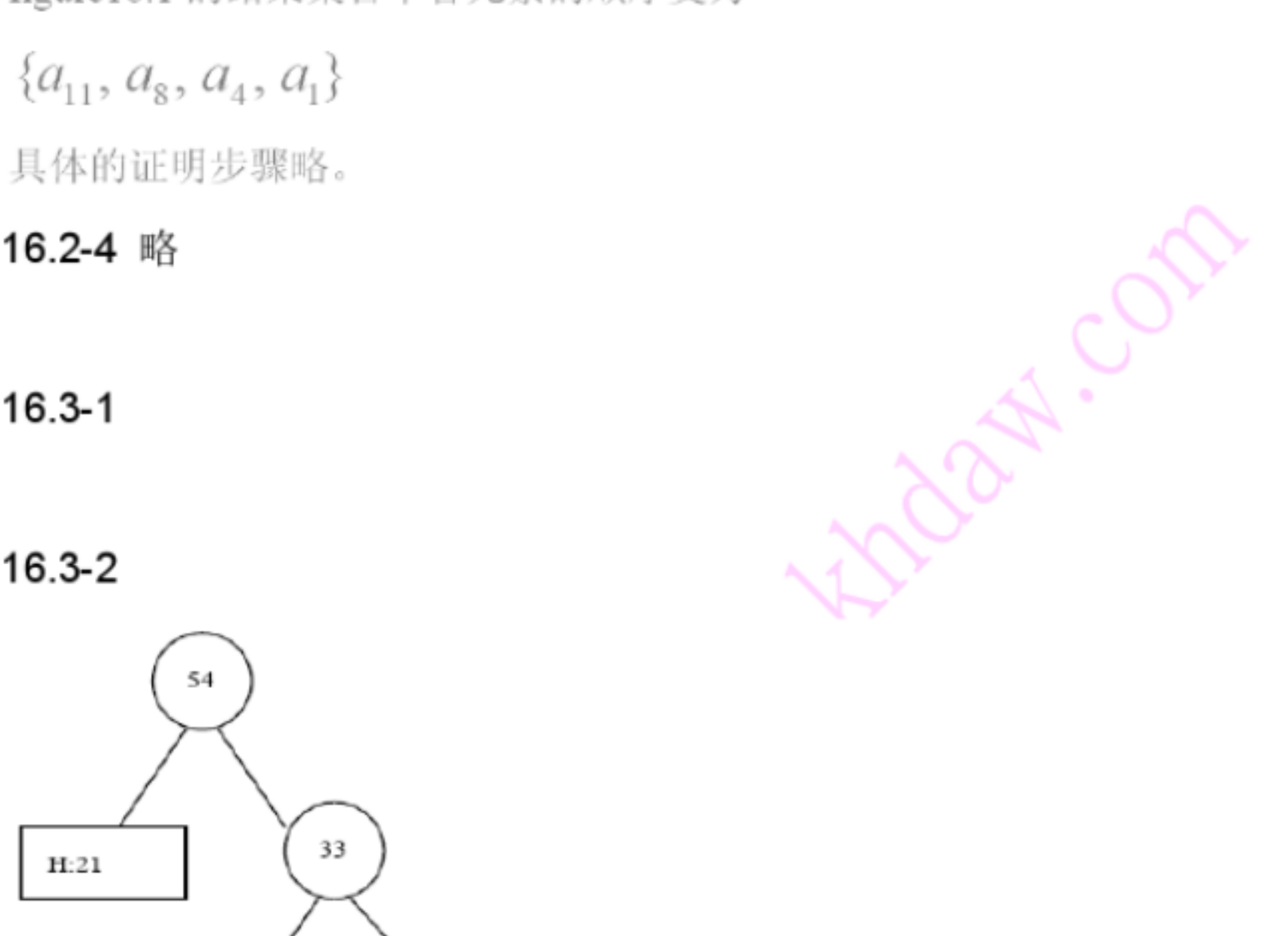
$$\{a_{11}, a_8, a_4, a_1\}$$

具体的证明步骤略。

16.2-4 略

16.3-1

16.3-2



前 n 个数的编码为：

$11...1,11...10,11...10,...,10,1$
$\quad n \qquad n-1 \qquad n-2$

16.3-4

反证

16.4-1

根据书上 page393 的 matroid 的定义，分别证明(S, I_k)中 S 为有穷非空集合，满足 hereditary 性质和 exchange 性质即可。

16.5-1

Answers By Zhifang Wang

17.1-1

这题有歧义

a) 如果 Stack Operations 包括 Push Pop MultiPush,答案是可以保持,解释和书上的 Push Pop MultiPop 差不多.

b) 如果是 Stack Operations 包括 Push Pop MultiPush MultiPop,答案就是不可以保持,因为 MultiPush,MultiPop 交替的话,平均就是 O(K).

17.1-2

Increment 操作每次最多可以使 k 位翻转,同样,Decrement 也是 k 位,所以不难得到结论.

17.2-1

使用会计方法，因为栈的大小始终不会超过 k，所以如果我们给每个栈操作都赋予多一些的代价，那么就可以有足够的余款来支付复制整个栈的代价。

对栈操作赋予以下的平摊代价：

PUSH	3
POP	0
MULTIPOP	0
COPY	0

17.3-1

因为 $\Phi(D_0) = k \neq 0$

定义 Φ' 为： $\Phi'(x) = \Phi(x) - k$

因为 $\Phi(D_i) \geq \Phi(D_0) = k$

所以 $\Phi'(D_i) = \Phi(D_i) - k \geq 0$

$\Phi'(D_0) = \Phi(D_0) - k = 0$

根据公式 17.3，用 Φ 和 Φ' 计算的平摊代价是相等的

17.3-4

定义势函数 Φ 栈中对象的个数，PUSH 的平摊代价为 2，MULTIPOP 和 POP 的平摊代价为 0

注意：这里是根据公式 17.4，而不是公式 17.3。考虑最坏情况，最终的结果是 $2n + s_0 - s_n$

17.4-3

假设第 i 个操作是 TABLE_DELETE，考虑装载因子 α: α_i =(第 i 次循环之后的表中的 entry 数)/(第 i 次循环后的表的大小)= $\mathrm{num}_i / \mathrm{size}_i$

case 1. If $\alpha_{i-1} = 1/2$, $\alpha_i < 1/2$

$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (\mathrm{size}_i - 2\mathrm{num}_i) - (2\mathrm{num}_{i-1} - \mathrm{size}_{i-1})$

$= 1 + (\mathrm{size}_{i-1} - 2(\mathrm{num}_{i-1} - 1)) - (2\mathrm{num}_{i-1} - \mathrm{size}_{i-1}) = 3 + 2\,\mathrm{size}_{i-1} - 4\,\mathrm{num}_{i-1}$

$= 3 + 2\,\mathrm{size}_{i-1} - 4\,\alpha_{i-1}\,\mathrm{size}_{i-1} \leq -1 + 2\,\mathrm{size}_{i-1} - 4/2\,\mathrm{size}_{i-1} = 3$

case 2. If $1/3 \leq \alpha_{i-1} < 1/2$, $\alpha_i \leq 1/2$,i-th operation would not cause a contraction

$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = 1 + (\mathrm{size}_i - 2\mathrm{num}_i) - (\mathrm{size}_{i-1} - 2\mathrm{num}_{i-1})$

$= 1 + (\mathrm{size}_{i-1} - 2(\mathrm{num}_{i-1} - 1)) - (\mathrm{size}_{i-1} - 2\mathrm{num}_{i-1}) = 3$

case 3. If $\alpha_{i-1} = 1/3$, $\alpha_i \leq 1/2$ thus i-th operation would cause a contraction

$\hat{c}_i = c_i + \Phi_i - \Phi_{i-1} = \mathrm{num}_i + 1 + (\mathrm{size}_i - 2\mathrm{num}_i) - (\mathrm{size}_{i-1} - 2\mathrm{num}_{i-1})$

$= \mathrm{num}_{i-1} - 1 + 1 + (2/3\,\mathrm{size}_{i-1} - 2(\mathrm{num}_{i-1} - 1)) - (\mathrm{size}_{i-1} - 2\mathrm{num}_{i-1})$

Thus the amortized cost of TABLE-DELETE is bounded by 3.

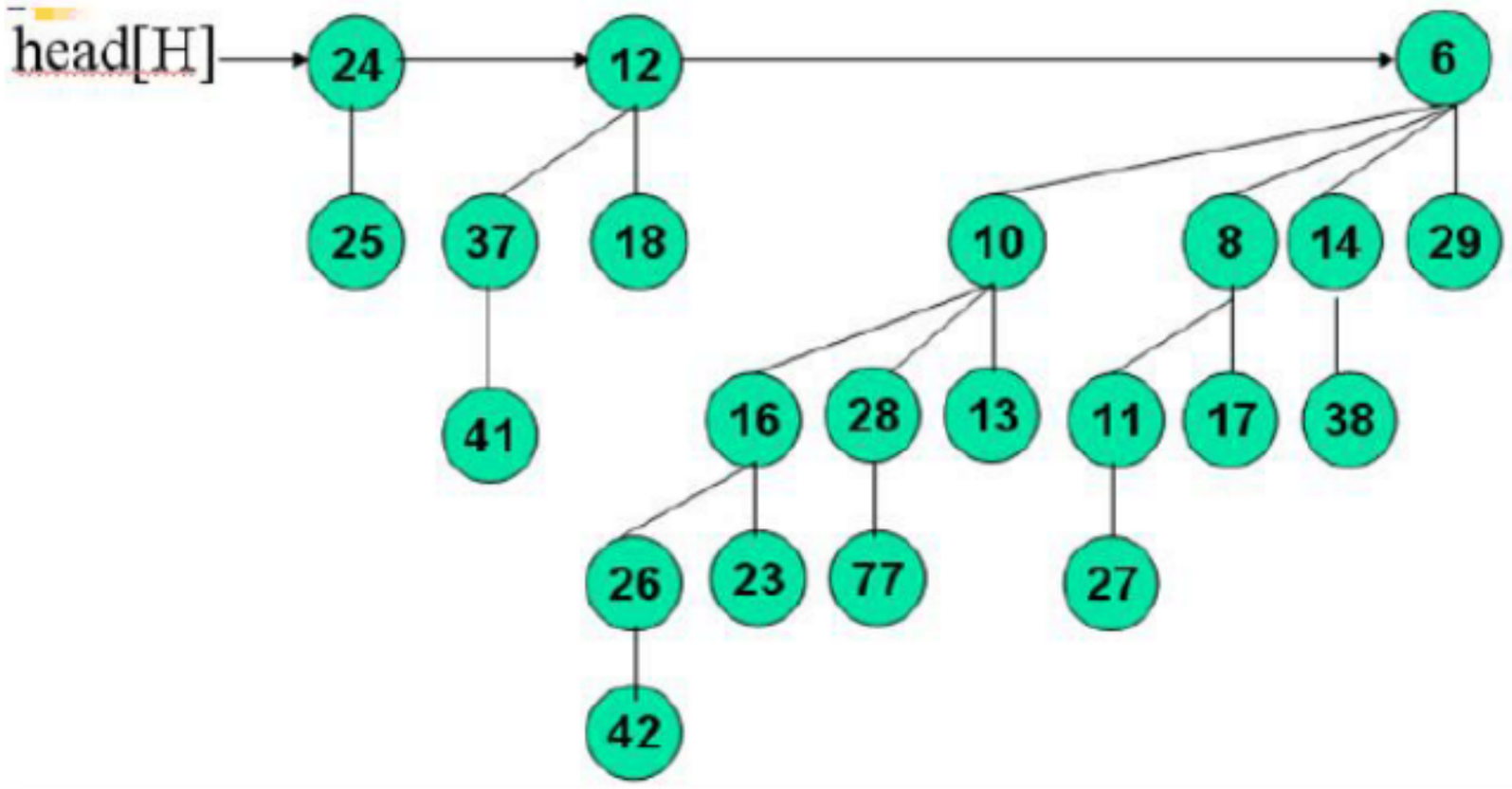
19.1-1.

If x is not a root node, then Degree[x]=Degree[sibling[x]]+1
If x is a root node, then Degree[x]<Degree[sibling[x]]

19.1-2

Degree[x]<degree[p[x]]

19.2-2

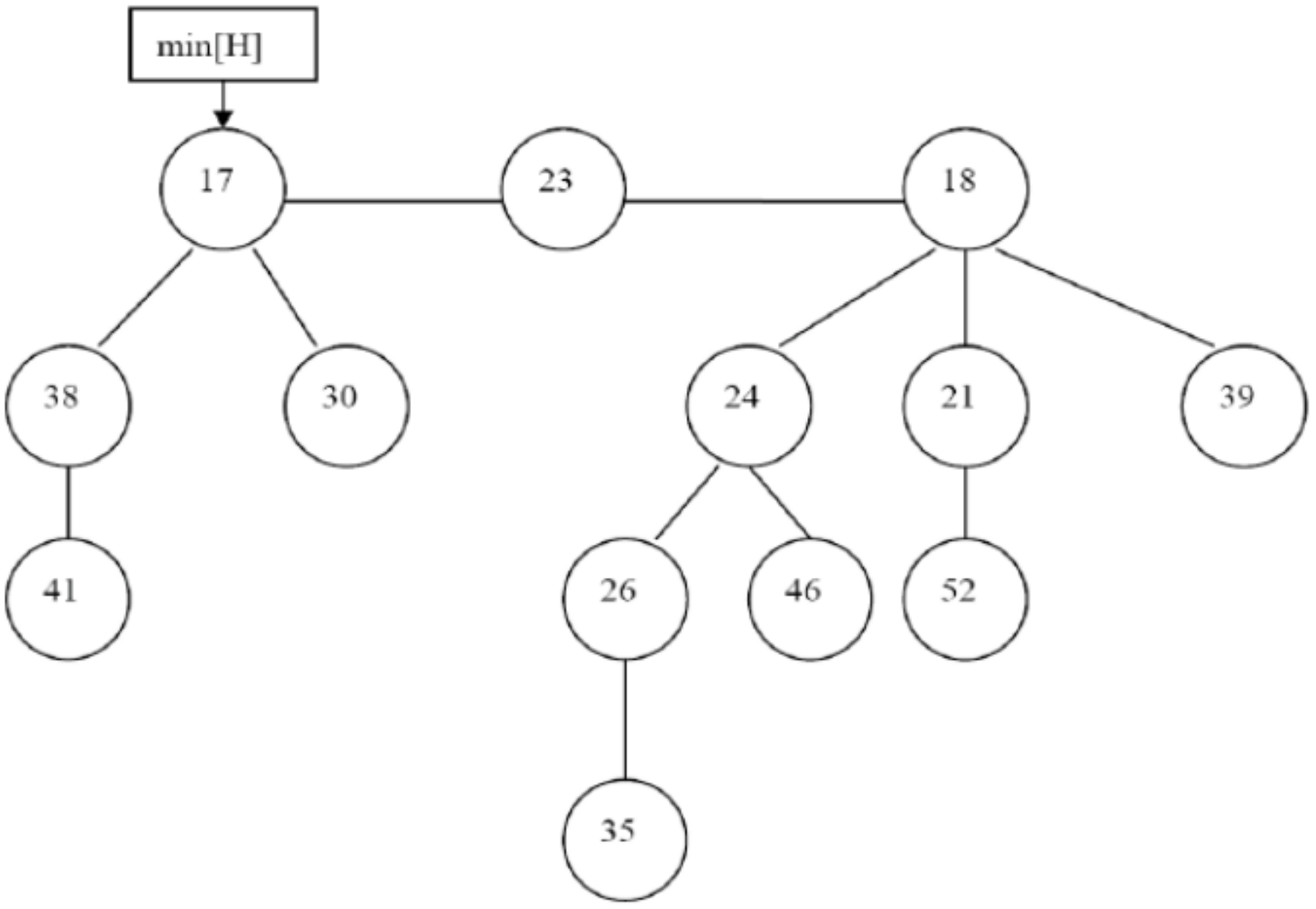


19.2-6

20.2-1

答案：参考书上 Page 484 上的图 20.3 以及 Page 486 的 CONSOLIDATE 算法

最终结果如下：（具体步骤略）



20.2-3

因为 **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，具体过程略。

20.3-1

答案：**1. x** 是如何成为有标记的根的？

在某个时刻，x 是一个根，然后 x 被链接到另一节点，且 x 失掉一个孩子，则 mark[x] 为 TRUE

2. 说明 **x** 有无标记对分析来说没有影响

根据书上 Page 492 的（t（H）+c）+2(m(H)-c+2)- (t(H)+2m(H))=4-c ，势的改变与 m(H)无关，因此，x 有无标记对分析来说都没有影响

26.1-1

如果 $(u,v)\notin E$ ， $(v,u)\notin E$ ，那么 $c(u,v)=0$， $c(v,u)=0$

由 capacity constraint, $f(u,v)\le c(u,v)=0$ (1)

由 skew symmetry， $f(u,v)=-f(v,u)\ge -c(v,u)=0$ (2)

由（1）（2）可得 $f(u,v)=0$，同理 $f(v,u)=0$

26.1-2

根据 flow conservation，对于所有的 $u\in V-\{s,t\}$，$\sum_{v\in V} f(u,v)=0$，以及（26.2）定义的 the total positive flow 不难得出证明，具体步骤略

26.2-1

根据书上 Page655 的 net flow 和 capacity 的定义以及图 26.4 的例子。不难得到

$$\begin{aligned} f(S,T) &= f(s,v_1)+f(v_2,v_1)+f(v_2,v_3)+f(v_4,v_3)+f(v_4,t) \\ &= 11+1+(-4)+7+4=19 \end{aligned}$$

$$\begin{aligned} c(S,T) &= c(s,v_1)+c(v_2,v_1)+c(v_4,v_3)+c(v_4,t) \\ &= 16+4+7+4=31 \end{aligned}$$

26.2-4

根据书上 651 页的式子（26.5） $c_f(u,v)=c(u,v)-f(u,v)$ 很容易证明这个结论

$$c_f(u,v)+c_f(v,u)=c(u,v)-f(u,v)+c(v,u)-f(v,u)$$

26.3-1
s – 1 – 6 – t
s – 2 – 8 – t
s – 3 – 7 – t

26.3-3

$$2\min\{|L|,|R|\}+1$$

26.4-2
根据 26.22 对每个顶点执行 relabel 操作的次数最多为 2|V|-1
所以对每条边检测次数最多为 2*（2|V|-1）次 O（V）
在剩余网络中最多 2|E|条边 O（E）
所以为 O（VE）