散列表

提到散列表能想到什么?

11.2-2（2'）

> 对于一个用链地址法解决冲突的散列表，说明将关键字5，28，16，15，20，33，12，17，10插入到该表中的过程，该表有9个槽位，并设其散列函数为h(k) = k mod 9

slots 0,1,...,8

h(k)	keys
0 mod 9	
1 mod 9	10->19->28
2 mod 9	20
3 mod 9	12
4 mod 9	
5 mod 9	5
6 mod 9	33->15
7 mod 9	
8 mod 9	17

11.4-1

> 考虑用开放寻址法，将关键字10，22，31，4，15，28，17，88，59插入到一长度为m = 11的散列表中，辅助散列函数h'(k) = k。试说明分别用线性探查、二次探查$(c_1 = 1, c_2 = 3)$和双重散列$(h_1(k) = k, h_2(k) = 1 + (k\,mod\,(m-1)))$将这些关键字插入散列表的过程。

线性探查

T_t表示时间从i=0开始，如果遇到冲突，依次向后遍历直到不产生冲突。

h(k,i) = (k+i) mod 11	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22							
1 mod 11								88	
2 mod 11									
3 mod 11									
4 mod 11				4					
5 mod 11					15				
6 mod 11						28			
7 mod 11							17		
8 mod 11									59
9 mod 11			31						
10 mod 11	10								

二次探查

$h(k,i) = (k + i + 3i^2) mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22							
1 mod 11									
2 mod 11(i=8)								88	
3 mod 11(i=3)							17		
4 mod 11				4					
5 mod 11									
6 mod 11						28			
7 mod 11(i=2)									59
8 mod 11					15				
9 mod 11			31						
10 mod 11	10								

双重散列

10,22,31,4,15,28,17,88,59

$h(k,i) = (k + i(1 + k mod 10)) mod 11$	T_0	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
0 mod 11		22							
1 mod 11									
2 mod 11(59+2*(1+59mod10)) mod 11									59
3 mod 11(17+1(1+17mod10)) mod 11							17		
4 mod 11				4					
5 mod 11(15+2(1+15 mod 10)) mod 11					15				
6 mod 11(28 mod 11)						28			
7 mod 11(88+2*(1+88mod10))mod11								88	
8 mod 11									
9 mod 11			31						
10 mod 11	10								

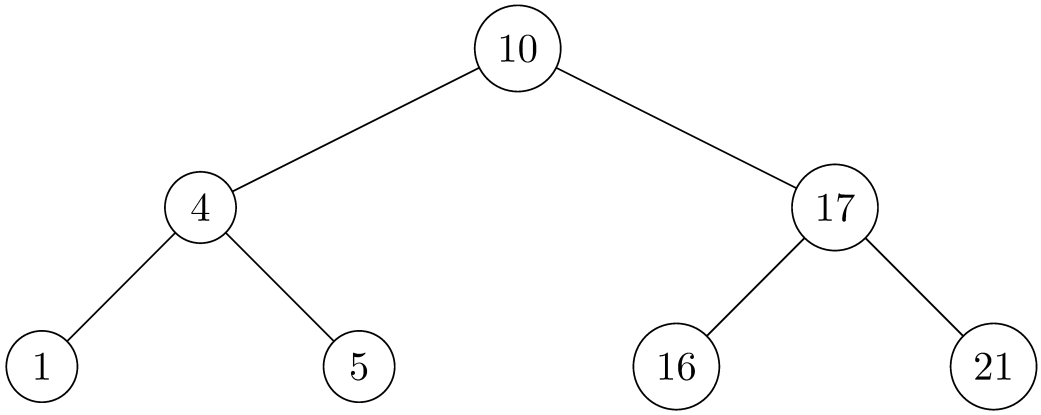
二叉搜索树

什么是二叉搜索树？二叉搜索树包括哪些操作？插入和删除操作？

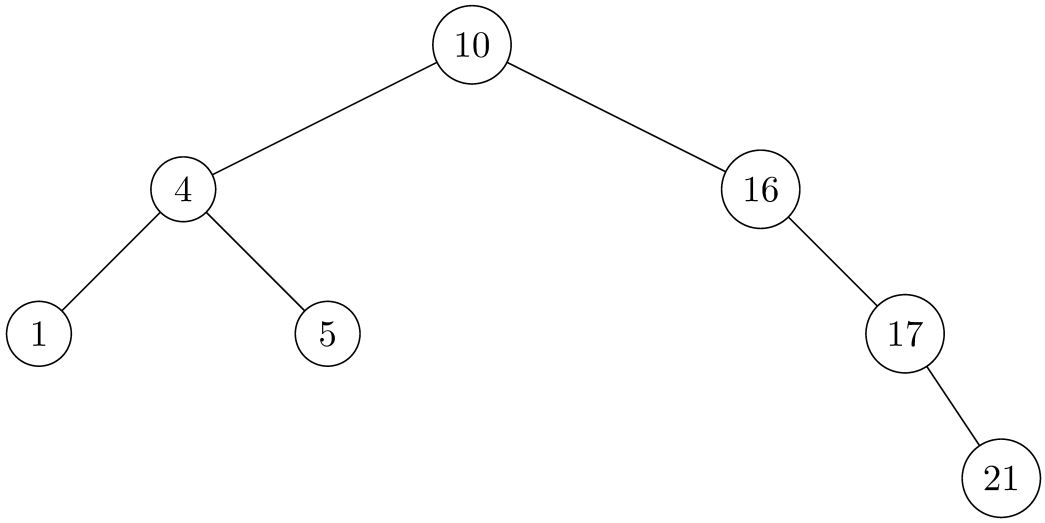
12.1-1（2’）

> 对于关键字集合{1，4，5，10，16，17，21}，分别画出高度为2，3，4，5和6的二叉搜索树

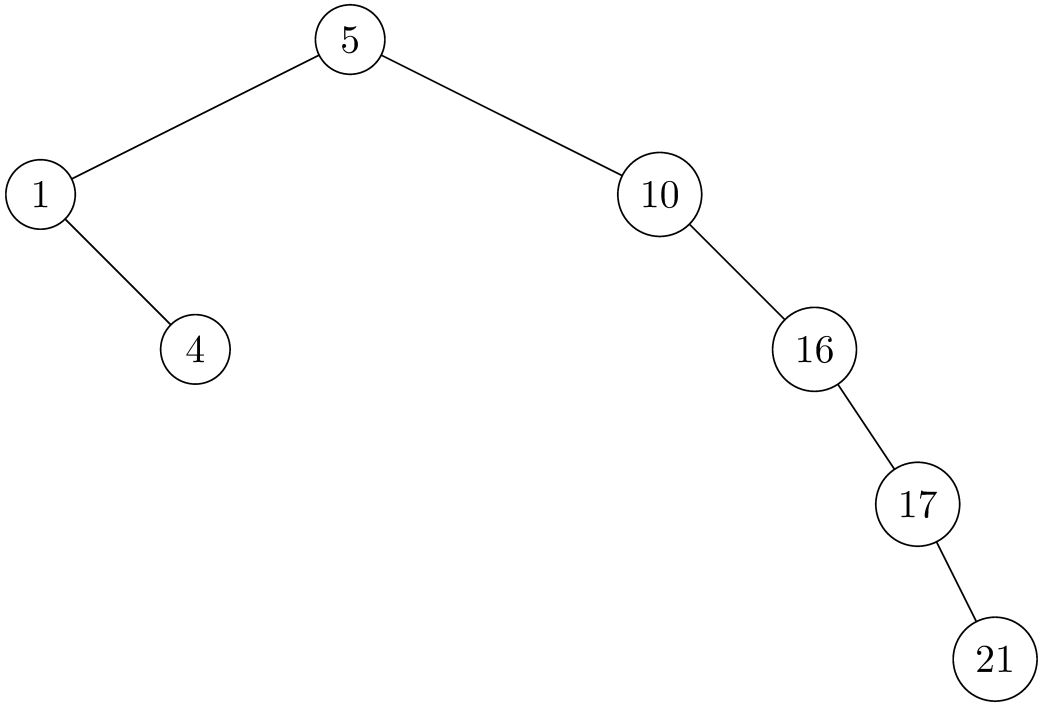
高度为2



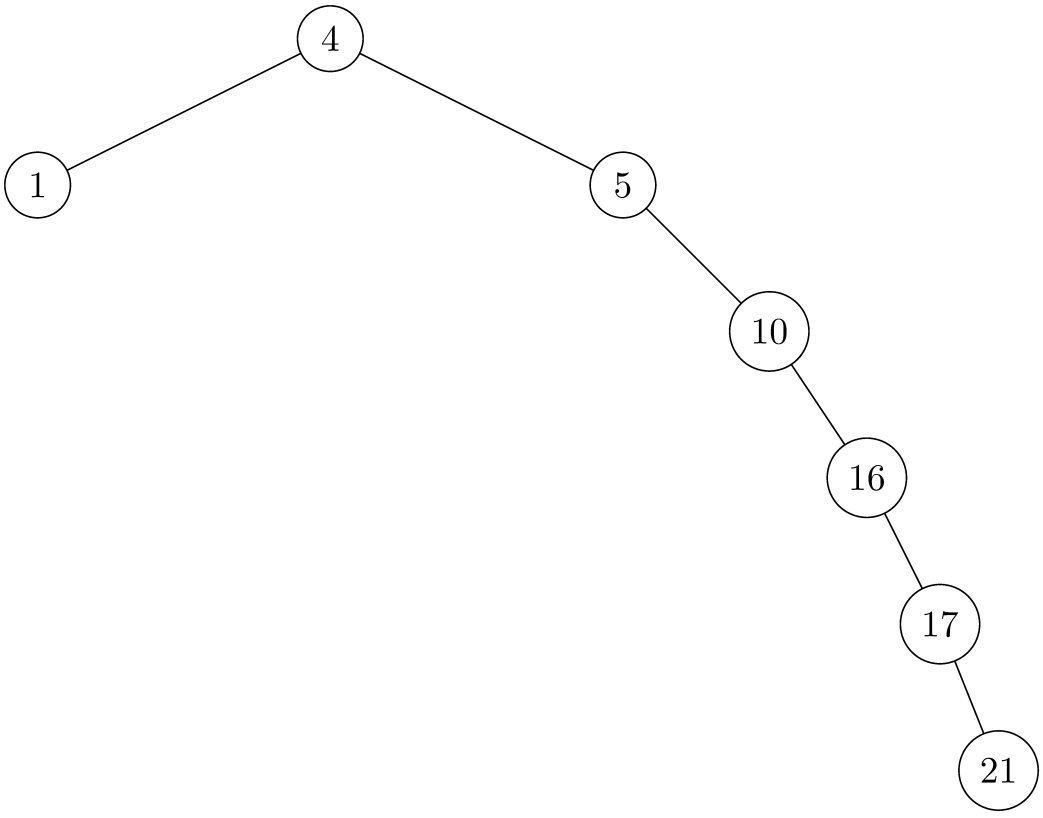
高度为3



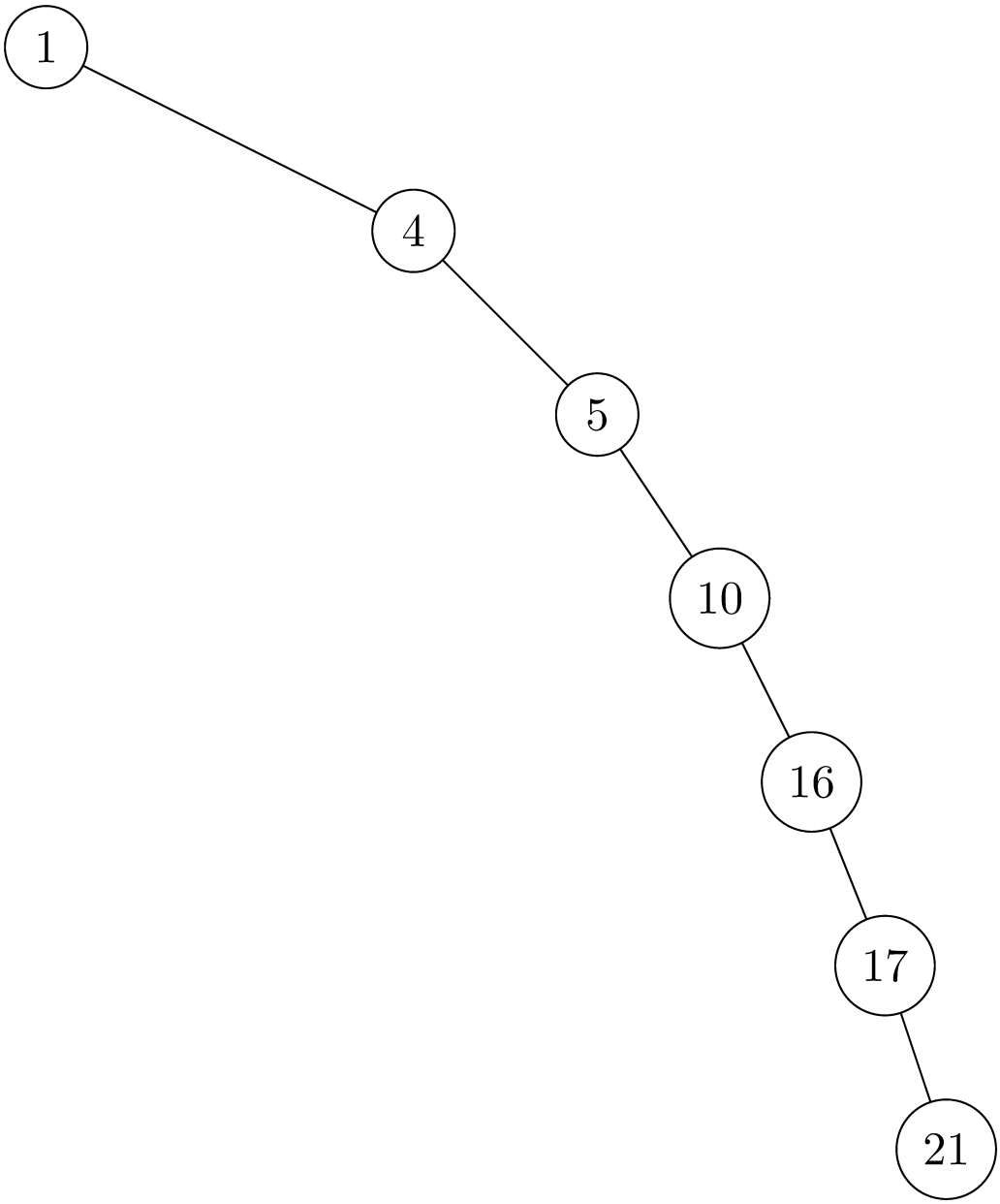
高度为4



高度为5



高度为6



12.1-2

> 二叉搜索树性质与最小堆性质之间有什么不同？能使用最小堆性质在 O(n) 时间内按序输出一棵有 n 个结点的树的关键字吗？可以的话，请说明如何做，否则解释说明理由。

二叉搜索树保证左子树节点的值小于等于根节点的值，右子树节点的值大于等于根节点的值。min-heap 属性只保证父节点小于子节点。

因此，min-heap 性质不能用于在 O(n) 时间内按序输出关键字，因为无法知道哪个子树包含下一个最小元素。

12.2-1

> 假设一棵二叉搜索树中的结点在1到1000之间，现在想要查找数值为363的结点，下面序列中哪个不是查找过的序列？
>
> **a.** 2, 252, 401, 398, 330, 344, 397, 363
>
> **b.** 924, 220, 911, 244, 898, 258, 362, 363
>
> **c.** 925, 202, 911, 240, 912, 245, 363

> **d.** 2, 399, 387, 219, 266, 382, 381, 278, 363
>
> **e.** 935, 278, 347, 621, 299, 392, 358, 363

c. 不可能是探索的节点序列，因为从911节点，但无法到达912节点，912大于911,912不能属于911的左子树。

e. 299不能出现在347的右子树。

12.2-5

> 证明：如果二叉搜索树中的一个结点有两个孩子，那么它的后继没有左孩子，它的前驱没有右孩子。

在二叉搜索树（Binary Search Tree，BST）中，后继指的是某个节点在中序遍历顺序中的下一个节点。更具体地说，对于给定的节点x，如果存在一个节点y，它的值大于x的值且在BST中是最小的（即没有比y更接近x的节点），那么y被称为x的后继。

如果其后继节点有左孩子，那么该左孩子会成为该节点的后继节点。

同理，如果前驱节点有右孩子，那么该右孩子会成为该节点的前驱节点。

综上，如果二叉搜索树中某结点有两个孩子，那么它的后继没有左孩子，前驱没有右孩子。

12.3-3

> 对于给定的 n 个数的集合，可以通过先构造包含这些数据的一棵二叉搜索树（反复使用TREE-INSERT逐个插入这些数），然后按中序遍历输出这些数的方法，来对它们排序。这个排序算法的最坏情况运行时间和最好情况运行时间各是多少？

`insert`过程时间复杂度是$O(h)$的，h代表树的高度。

设一共有n个节点，构建一棵树需要调用n次$insert$，这个过程耗时$O(nh)$，后续的中序遍历耗时是$O(n)$。所以排序过程的时间复杂度是由构建过程决定的。

最坏的情况集合按照正序或者倒序排列，则$h = n$，总时间是$O(n^2)$。

最好情况是集合按照层次遍历顺序排列，最后构建成一棵完全二叉树，$h = lg(n)$,总时间为$O(n\,lg(n))$。

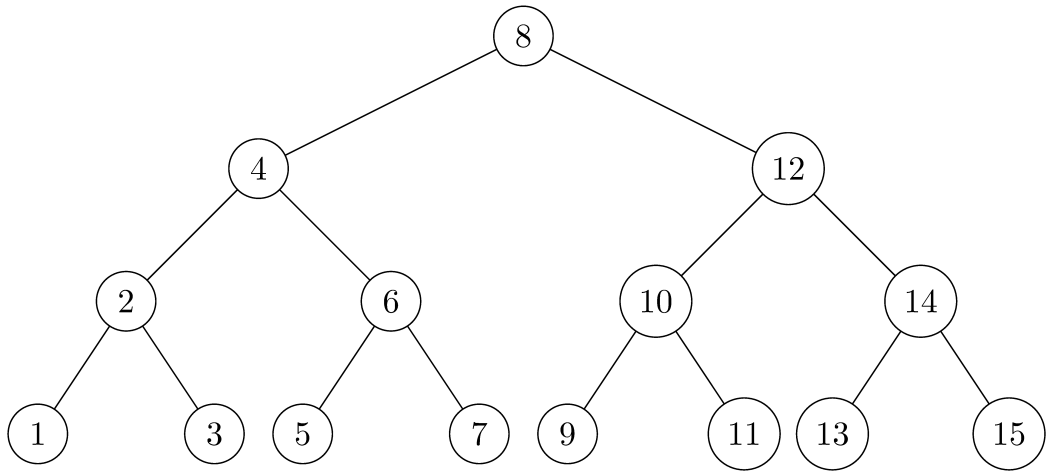
红黑树

红黑树的性质？插入、旋转、删除？

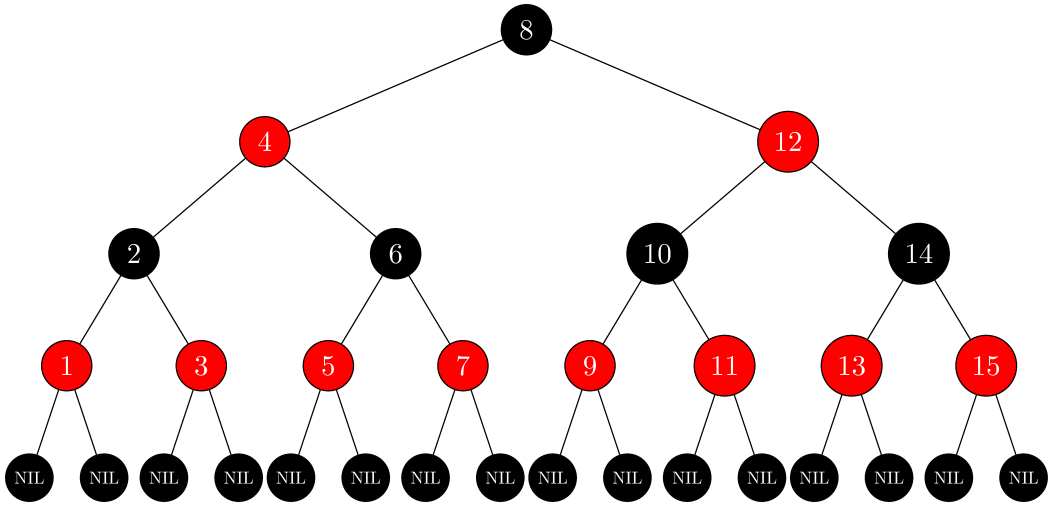
13.1-1

> 按照图13-1(a)的方式，画出关键字集合 {1,2,...,15} 上高度为3的完全二叉搜索树。以三种不同方式向图中加入NIL叶节点并对各结点着色，使所得的红黑树的黑高分别为2，3和4。

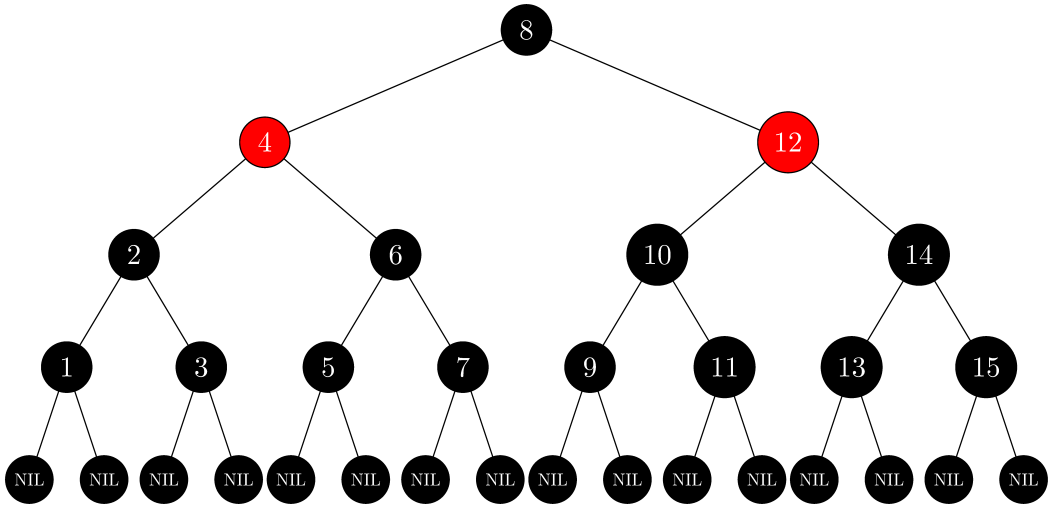
高度为3的二叉搜索树



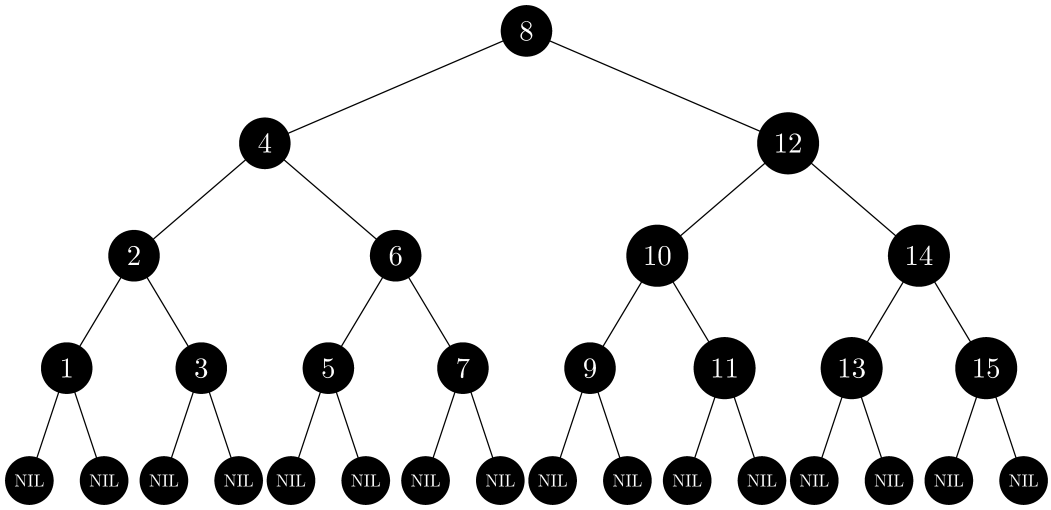
黑高为2



黑高为3



黑高为4



13.1-2

> 对图13-1中的红黑树，画出对其调用TREE-INSERT操作插入关键字36后的结果。如果插入结点被标为红色，所得的树是否还是一棵红黑树？如果插入结点被标为黑色呢，所得的树是否还是一棵红黑树？

回顾红黑树的性质

> 一棵红黑树是满足以下**红黑性质(red-black properties)**的二叉搜索树：
>
> 1. 每个结点要不是红色，要不是黑色。
> 2. 根结点是黑色的。
> 3. 每个叶结点（NIL结点）是黑色的。
> 4. 如果一个结点是红色的，那么它的两个孩子结点都是黑色的。
> 5. 对于每个结点，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。

如果插入的节点为红色，则树不满足属性4（如果一个节点是红色的，则它的两个叶子节点都是黑色的）。
如果插入的节点为黑色，则树不满足属性5（对每个节点，从该节点到其所有后代叶节点的简单路径上，句包含相同数目的黑色节点）

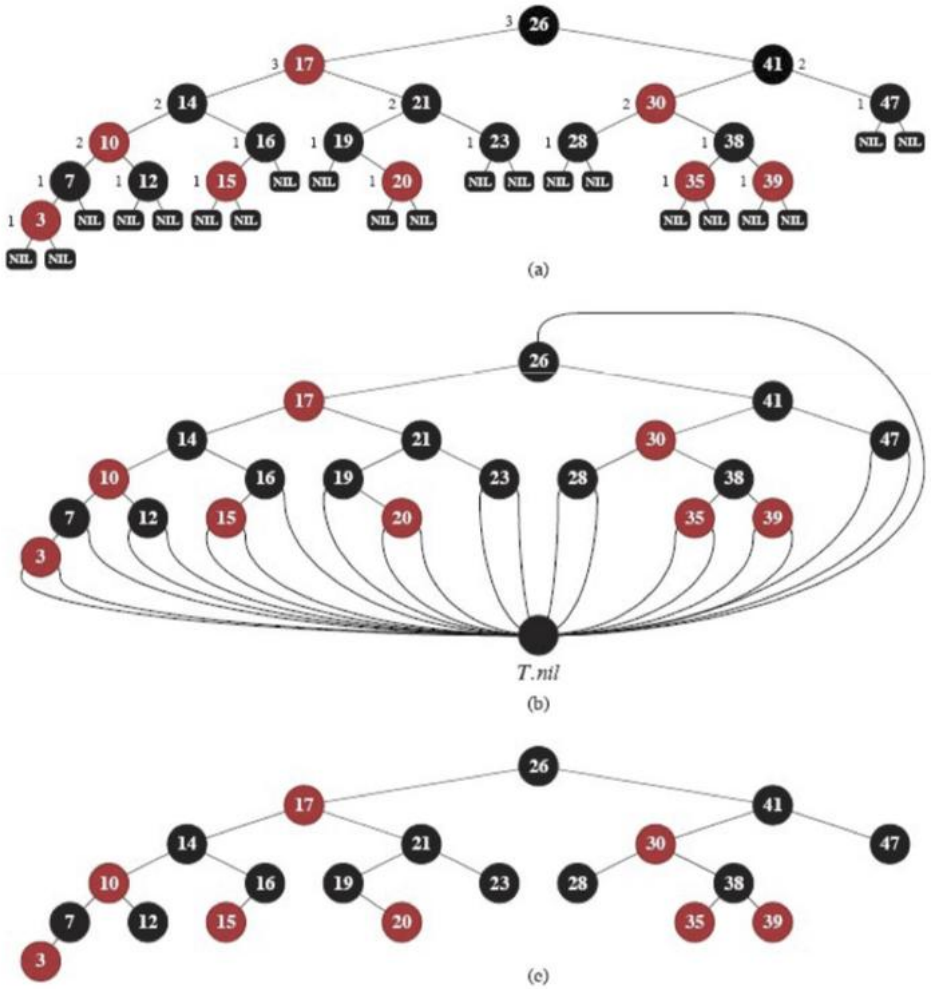
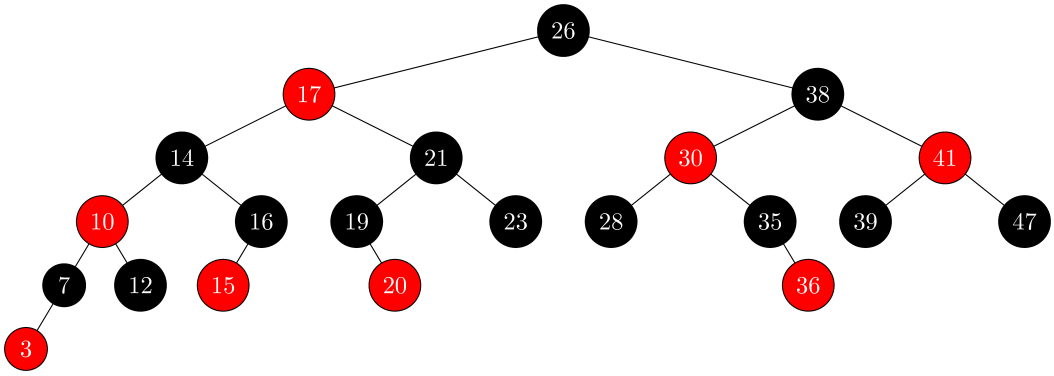


Figure 13.1 A red-black tree. Every node in a red-black tree is either red or black, the children of a red node are both black, and every simple path from a node to a descendant leaf contains the same number of black nodes. **(a)** Every leaf, shown as a NIL, is black. Each non-NIL node is marked with its black-height, where NILs have black-height 0. **(b)** The same red-black tree but with each NIL replaced by the single sentinel *T.nil*, which is always black, and with black-heights omitted. The root's parent is also the sentinel. **(c)** The same red-black tree but with leaves and the root's parent omitted entirely. The remainder of this chapter uses this drawing style.



13.1-6

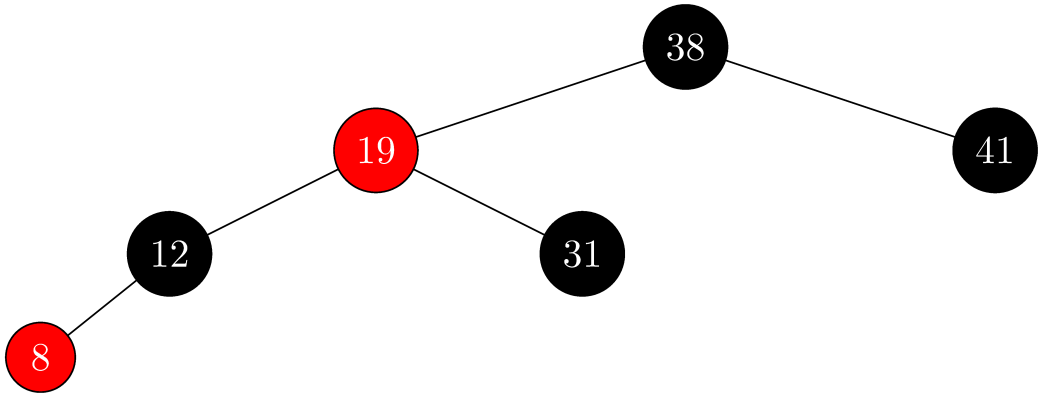
> 在一棵黑高为 k 的红黑树中，内部结点最多可能有多少个？最少可能有多少个？

最长的是一条半是黑色节点，一半是红色节点的路径，有$2^{2k}-1$个内部节点

最短的是全黑节点的路径，有$2^{k}-1$个内部节点

13.3-2

> 将关键字 41, 38, 31, 12, 19, 8 连续地插入一棵初始为空的红黑树，画出这个过程。



动态规划

代表性问题?

15.1-3

> 我们对钢条切割问题进行一点修改，除了切割下的钢条段具有不同的价格p_i外，每次切割需要付出固定的成本 c 。这样，切割方案的收益就等于钢条段的价格之和减去切割的成本。设计一个动态规划算法解决修改后的钢条切割问题。

```
BOTTOM-UP-CUT-ROD(p, n)
    let r[0..n] be a new array
    r[0] = 0    //长度为0的钢条没有收益
    for j = 1 to n  //对j=1,2,...,n按升序求解每个规模为j的子问题
        q = -∞
        for i = 1 to j
            q = max(q, p[i] + r[j - i])
        r[j] = q
    return r[n]
```

对于原始版本增加一些修改

```
MODIFIED-CUT-ROD(p, n, c)
    let r[0..n] be a new array
    r[0] = 0
    for j = 1 to n
        q = p[j]
        for i = 1 to j - 1
            q = max(q, p[i] + r[j - i] - c)
        r[j] = q
    return r[n]
```

我们需要考虑成本c在第 5-6 行循环的每次迭代中，但最后一次，当i=j（没有削减）。

15.1-5

> 斐波那契数列可以用递归式(3.22)定义，设计一个运行时间为 O(n) 动态规划算法计算第 n 个斐波那契数，画出子问题图，图中有多少顶点和边?

```
FIBONACCI(n)
    let fib[0..n] be a new array
    fib[0] = 1
    fib[1] = 1
    for i = 2 to n
        fib[i] = fib[i - 1] + fib[i - 2]
    return fib[n]
```

Figure 14.1-6

在子问题图中一共有n+1个节点，$v_0, v_1, \ldots, v_n$

对于v_0, v_1,只有0条边；而对于$v_2, v_3, \ldots, v_n$则有2条边；因此，在子问题图中一共有2n-2条边。

15.2-1

> 对于矩阵维度序列 ⟨5,10,3,12,5,50,6⟩ ， 求其矩阵链乘的最优括号化方案。

$$(A_{5\times10}A_{10\times3})((A_{3\times12}A_{12\times5})(A_{5\times50}A_{50\times6}))$$

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

具体计算过程：

$$m[1,1]=m[2,2]=m[3,3]=\ldots=m[6,6]=0$$

$$m[1,2]=m[1,1]+m[2,2]+5\times10\times3=150$$

$$m[2,3]=m[2,2]+m[3,3]+10\times3\times12=360$$

$$m[3,4]=m[3,3]+m[4,4]+3\times12\times5=180$$

$$m[4,5]=m[4,4]+m[5,5]+12\times5\times50=3000$$

$$m[5,6]=m[5,5]+m[6,6]+5\times50\times6=1500$$

$$m[1,3]=min\begin{cases}m[1,1]+m[2,3]+5\times10\times12\\ m[1,2]+m[3,3]+5\times3\times12\end{cases}=330$$

$$m[2,4]=min\begin{cases}m[2,2]+m[3,4]+10\times3\times5\\ m[1,2]+m[3,3]+10\times12\times5\end{cases}=330$$

$$m[3,5]=min\begin{cases}m[3,3]+m[4,5]+3\times12\times50\\ m[3,4]+m[5,5]+3\times5\times50\end{cases}=930$$

$$m[4,6]=min\begin{cases}m[4,4]+m[5,6]+12\times5\times6\\m[4,5]+m[6,6]+12\times50\times6\end{cases}=1860$$

$$m[1,4]=min\begin{cases}m[1,1]+m[2,4]+5\times10\times5\\m[1,2]+m[3,4]+5\times3\times5\\m[1,3]+m[4,4]+5\times12\times5\end{cases}=405$$

$$m[2,5]=min\begin{cases}m[2,2]+m[3,5]+10\times3\times50\\m[2,3]+m[4,5]+10\times12\times50\\m[2,4]+m[4,5]+10\times5\times50\end{cases}=2043$$

$$m[3,6]=min\begin{cases}m[3,3]+m[4,6]+3\times12\times6\\m[3,4]+m[5,6]+3\times5\times6\\m[3,5]+m[6,6]+3\times50\times6\end{cases}=1770$$

$$m[1,5]=min\begin{cases}m[1,1]+m[2,5]+5\times10\times50\\m[1,2]+m[3,5]+5\times3\times50\\m[1,3]+m[4,5]+5\times12\times50\\m[1,4]+m[5,5]+5\times5\times50\end{cases}=1655$$

$$m[2,6]=min\begin{cases}m[2,2]+m[3,6]+10\times3\times6\\m[2,3]+m[4,6]+10\times12\times6\\m[2,4]+m[5,6]+10\times5\times6\\m[2,5]+m[6,6]+10\times50\times6\end{cases}=1950$$

$$m[1,6]=min\begin{cases}m[1,1]+m[2,6]+5\times10\times6\\m[1,2]+m[3,6]+5\times3\times6\\m[1,3]+m[4,6]+5\times12\times6\\m[1,4]+m[5,6]+5\times5\times6\\m[1,5]+m[6,6]+5\times50\times6\end{cases}=2010$$

贪心算法

局部最优的选择 代表性问题?

16.1-2

> 假定我们不再一直选择最早结束的活动，而是选择最晚开始的活动，前提仍然是与之前选出的所有活动均兼容。描述如何利用这一方法设计的贪心算法，并证明算法会产生最优解。

这个策略和选择最早结束的方案是完全对称的，假设时间倒流，所有活动倒过来执行。

有 n 个活动构成的集合 $S=\{a_1,a_2,\dots,a_n\}$，假设这些活动已经按照结束时间从小到大排序，即： $s_1\geq s_2\geq s_3\geq\cdots\geq s_n$ 。

贪心选择一个能够最晚开始活动，这样留下尽可能多的资源供前面的活动使用。

由于所有活动已经按照开始时间从大到小排序，贪心选择 a_1 ，接下来只有一个子问题要处理，选择一个在 a_1 开始前结束的活动。为什么没必要考察在 a_1 结束后的活动呢？因为 $s_1<f_1$ ， s_1 是最晚开始的活动，所以不会有活动开始时间晚于 s_1 ，可以贪心选择活动 a_1 。

令 $S_k = \{a_i \in S : f_k \leq s_i\}$ 为在 a_k 开始前结束的任务集合。当贪心选择 a_1 后， S_1 是唯一需要求解的子问题。

考虑任意非空子问题 S_k ，如果 a_m 是 S_k 中开始时间最晚的活动，那么 a_m 在 S_k 的某个最大兼容活动子集中。

证明：定义 A_k 是 S_k 的最大兼容活动子集，且 a_j 是 A_k 中开始时间最晚的活动。

若$a_j = a_m$，则已证明$a_m \in A_k$。

若$a_j \neq a_m$，令$A'_k = (A_k - \{a_j\}) \cup \{a_m\}$，则$|A_k| = |A'_k|$。又$a_j$是$A_k$中开始时间最晚的活动，$a_m$是$A'_k$中开始时间最晚的活动，$s_m \geq s_j$。所以$A'_k$是$S_k$最大兼容活动子集，$a_m \in A'_k$。

我们每次选择开始时间最晚的且与当前活动兼容的活动，重复执行这个过程直到没有剩余活动可以选择。所选择的活动的开始时间必然是严格递减的。所以我们只需要按照开始时间单调递减的顺序处理所有活动，每个活动仅需要考察一次。

http://m.blog.chinaunix.net/uid-29527314-id-5149558.html

16.1-3

> 对于活动选择问题，并不是所有贪心方法都能得到最大兼容的活动子集。请举例说明，在剩余兼容活动中选择持续时间最短的活动不能得到最大集。类似地，说明在剩余兼容活动中选择与其它剩余活动重叠最少者，以及选择最早开始者均不能得到最优解。

在剩余兼容活动中选择持续时间最短者不能得到最大集，选择持续时间最短，我们只能选择一个；如果不选择持续时间最短，则可以选择第一个和第三个两个活动。

$\{(1,9),(8,11),(10,20)\}$

在剩余兼容活动中选择与其他活动重叠最少者，通过这种策略，首先选择$(4,7)$，无法得到$\{(-1,1),(2,5),(6,9),(10,12)\}$

$\{(-1,1),(2,5),(0,3),(0,3),(0,3),(4,7),(6,9),(8,11),(8,11),(8,11),(10,12)\}$

在剩余兼容活动中选择最早开始者，如果选择最早开始时间则只能得到一个解$(1,10)$

$\{(1,10),(2,3),(4,5)\}$

16.2-5

> 设计一个高效算法，对实数线上给定的一个点集$\{x_1,x_2,\ldots,x_n\}$，求一个单位长度的闭区间的集合，包含所有给定的点，并要求此集合最小。证明你的算法的是正确的。

设$x_1,x_2,\ldots,x_n$ 已经按照坐标从小到大排序，我们先考虑x_1，不存在值比x_1更小了，我们选择闭区间 $[x_1,x_1+1]$，设第一个大于 x_1+1 的点为 x_i，剩下子问题的点集为 $\{x_i,x_{i+1},\ldots,x_n\}$。重复执行上述步骤直到剩下子问题的点集为$\emptyset$ 。集合 S 为单位长度的闭区间的集合。

方法一：线性查找 + 线性查找

```
GREEDY-INTERVAL-SELECTOR(x, n)
    if n = 0
        return ∅
    let S be a new set
    i = 1
    while (i ≤ n)
        S = S ∪ {[x[i], x[i] + 1]}
        prev = i
        while (i ≤ n && x[i] ≤ x[prev] + 1) //向后遍历找到第一个大于
x[prev]+1的点
            i = i + 1
    return S
```

方法二：线性查找 + 二分查找，通过二分查找找到第一个大于x[prev]+1的点

```
GREEDY-INTERVAL-SELECTOR(x, n)
    if n = 0
        return ∅
    let S be a new set
    i = 1
    while (i ≤ n)
        S = S ∪ {[x[i], x[i] + 1]}
        prev = i
        l = i
        r = n
        while (l ≤ n)
            mid = ⌊(l + r) / 2⌋
            if (x[mid] ≤ x[prev] + 1)
                l = mid + 1
            else
                r = mid - 1
        i = l
    return S
```

16.3-2

> 证明一棵**非满二叉树(non-full binary tree)**不可能对应一个最优前缀编码。

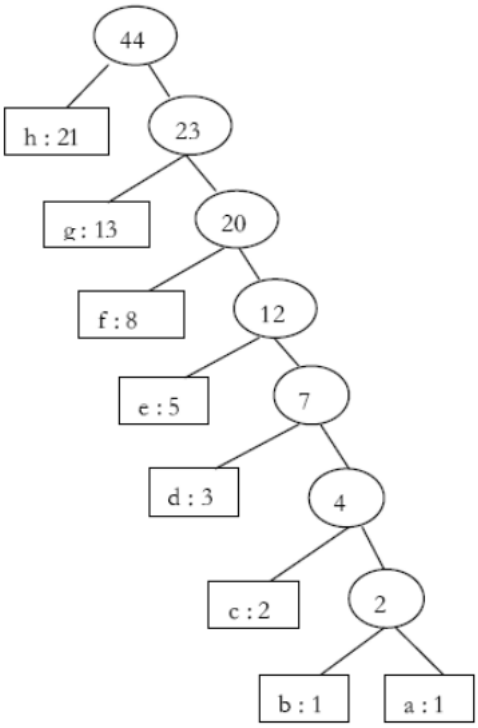
假设一棵非满二叉树对应一个最优前缀编码。可以采用反证法。

一棵非满二叉树至少存在一个度为 1 的内部结点，该内部结点只有一个孩子结点，且该孩子结点为叶结点。所以该非满二叉树对应的前缀无关编码可以进行优化，该内部结点可以调整为新的叶结点，记录其孩子结点的字符，并删除其孩子结点，此时该叶结点的深度可以减少 1 ，也就是存在比原先前缀无关编码更优的前缀无关编码，矛盾。

所以一棵非满二叉树不可能对应一个最优前缀无关编码。

16.3-3

> 如下所示，8个字符对应的出现的频率是斐波那契数列前 8 个数，此频率集合的赫夫曼编码是怎样的?
>
> $a:1 \quad b:1 \quad c:2 \quad d:3 \quad e:5 \quad f:8 \quad g:13 \quad h:21$
>
> 你能否推广你的结论，求频率集为前 n 个斐波那契数的最优前缀码?



a:1111111
b:1111110
c:111110
d:11110
e:1110
f:110
g:10
h:0

推广结论，求频率集为前n个斐波那契数的最优编码？

假设有n个字符，则第i个字符的赫夫曼编码对应于

若$i = 1$，有$n - 1$位的1

若$i > 1$，前$n - 1$位是1，最后一位是0

只需要证明：对任意的$n \geq 1$，总有下列式子成立：

(1) $f(n+1) \leq \sum_{i=1}^{n} f(i) \leq f(n+2)$

(2) $\sum_{i=1}^{n} f(i) = f(n+2) - 1$

用数学归纳法证明，对$n = 1$时，显然$f(2) \leq f(1) \leq f(3)$

假设，当$n = k(k > 1)$，(1)式成立，即

$f(k+1) \leq \sum_{i=1}^{n} f(i) \leq f(k+2)$成立，那么当$n = k + 1$时

$$f(k+2) = f(k) + f(k+1) \leq f(k+1) + f(k+1) \leq \sum_{i=1}^{k} f(i) + f(k+1) \leq f(k+2) + f(k+$$

由此，证明了$f(k+2) \leq \sum_{i=1}^{k+1} f(i) \leq f(k+3)$

第二个式子同理可以证明，证明也可以参考https://www.zhihu.com/question/22319265

摊还分析

包括哪几种？

用数据结构的一个操作序列中所执行的所有操作的平均时间来评价操作的代价

17.1-1

> 如果栈操作包含 MULTIPUSH 操作，将k个对象压入栈中，那么栈操作的摊还开销的界还是 O(1) 吗

不是，考虑最坏的情况，比如`MULTIPUSH`和`MULTIPOP`操作交替执行，每次开销均为$O(k)$，此时栈操作的摊还开销为$O(k)$，所以加入`MULTIPUSH`操作后，栈操作的摊还开销取决于具体的操作序列。

这一系列操作的时间复杂度取决于可以进行的push次数，一次MULTIPUSH需要$\Theta(k)$，n次MULTIPUSH需要$\Theta(kn)$，栈操作的摊还代价的界是$\Theta(k)$

17.1-2

> 证明：如果 k 位计数器例子中允许 DECREMENT 操作，那么 n个操作的运行时间能够达到 Θ(nk) 。

考虑对2^{k-1} 执行 DECREMENT操作，变为$2^{k-1}-1$，k 个二进制位均发生翻转，再对 $2^{k-1}-1$ 执行 INCREMENT操作，变为 2^{k-1}，k 个二进制位均发生翻转。所以最坏情况下，每个操作的开销均为 $\Theta(k)$ ，那么 n 个操作的运行时间能够达到$\Theta(nk)$。

17.1-3

> 对一个数据结构执行一个由 n 个操作组成的操作序列，当 i 严格为 2 的幂时，第 i 个操作的开销为 i ，否则代价为 1 ，使用聚合分析计算每个操作的摊还开销。

总的实际开销为:

$$\begin{aligned}\sum_{i=1}^{n} c(i) &= \sum_{j=0}^{\lfloor lg\, n\rfloor} 2^j + (n - \lfloor lg\, n\rfloor - 1)\cdot 1 \\ &= (2^{\lfloor lg\, n\rfloor} - 1) + n - \lfloor lg\, n\rfloor - 1 \\ &< (2\cdot 2^{lg\, n} - 1) + n - (lg\, n - 1) - 1 \\ &= 3n - lg\, n - 1 \\ &= O(n)\end{aligned}$$

因为总的摊还开销与总的实际开销渐近相等，所以每个操作的摊还开销为:

$$O(n)/n = O(1)$$

17.2-2

> 用核算法重做练习17.1-3

每次操作的摊还开销为3，保证$credit remaining \geq 0$恒成立

operation	actual cost	amortized cost	credit remaining
1	1	3	2
2	2	3	3
3	1	3	5
4	4	3	4
5	1	3	6
6	1	3	8
7	1	3	10
8	8	3	5
9	1	3	7
10	1	3	9
...	...	...	...

从17.1-3中得知$\sum_{i=1}^{n} c_i < 3n$，又$\sum_{i=1}^{n} \hat{c}_i = 3n$，故$\sum_{i=1}^{n} \hat{c}_i \geq \sum_{i=1}^{n} c_i$

执行 n 个操作后，总的摊还开销至多是 $3 \cdot n = O(n)$，操作序列中一个操作的摊还开销为 $O(n)/n = O(1)$。

17.3-2

> 使用势能法重做练习17.1-3

怎么选择合适的势函数?

栈操作的势函数定义为其中对象的数量

二进制计数器递增问题：将计数器执行 i 次$INCREMENT$操作的势定义为b_i—— i 次操作后计数器中1的个数

> 对一个数据结构执行一个由 n 个操作组成的操作序列，当 i 严格为 2 的幂时，第 i 个操作的开销为 i ，否则代价为 1 ，使用势能法计算每个操作的摊还开销。

由于第 i 个操作的开销不可能为负，

$\Phi(D_0) = 0$，对$i > 0$，$\Phi(D_i) = 2i - 2^{1+\lfloor lg\, i \rfloor}$

对$i = 1$，$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2i - 2^{1+\lfloor lg\, i \rfloor} - 0 = 1$

对$i > 1$，并且不是2的幂次，
$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = 1 + 2i - 2^{1+\lfloor lg\, i \rfloor} - (2(i-1) - 2^{1+\lfloor lg\,(i-1) \rfloor}) = 3$

对$i = 2^j$，
$\hat{c}_i = c_i + \Phi(D_i) - \Phi(D_{i-1}) = i + 2i - 2^{1+j} - (2(i-1) - 2^{1+j-1}) = 2$

一个操作的摊还代价是$O(1)$

17.3-4

> 执行 n 个 PUSH、POP和 MULTIPOP 栈操作的总代价是多少？假定初始时栈中包含 s_0 个对象，结束后包含 s_n 个对象。

设势函数为栈中包含的对象，由题意，$D_0 = s_0$，$D_n = s_n$

书上P263有关于PUSH操作的摊还代价 $\hat{c}_i = 2$，POP操作的摊还代价$\hat{c}_i = 0$，MULTIPOP操作的摊还代价$\hat{c}_i = 0$。

$$\sum_{i=1}^{n} c_i = \sum_{i=1}^{n} \hat{c}_i - \Phi(D_n) + \Phi(D_0) \leq \sum_{i=1}^{n} 2 - s_n + s_0 = 2n - s_n + s_0$$

基本的图算法

图的表示方式？深度优先，广度优先，拓扑排序

22.1-1

> 给定有向图的邻接链表，需要多长时间才能计算出每个顶点的出度(发出的边的条数)？多长时间才能计算出每个顶点的出度(进入的边的条数)？以P342 图22-2为例

设有向图为 $G = (V, E)$ 。

计算每个顶点的出度需要遍历所有的顶点和边，需要时间为 $\Theta(|V| + |E|)$。

计算每个顶点的入度需要遍历所有的顶点和边，需要时间为 $\Theta(|V| + |E|)$。

22.1-2

邻接链表

$$\begin{aligned}
&1 \to 2 \to 3\\
&2 \to 1 \to 4 \to 5\\
&3 \to 1 \to 6 \to 7\\
&4 \to 2\\
&5 \to 2\\
&6 \to 3\\
&7 \to 3
\end{aligned}$$

邻接矩阵

	1	2	3	4	5	6	7
1	0	1	1	0	0	0	0
2	1	0	0	1	1	0	0
3	1	0	0	0	0	1	1
4	0	1	0	0	0	0	0
5	0	1	0	0	0	0	0
6	0	0	1	0	0	0	0
7	0	0	1	0	0	0	0

最小生成树

Kruskal算法、Prim算法

23.1-1

> 设 (u,v) 是连通图 G 的一条权重最小的边，证明：边 (u,v) 为图 G 的某棵最小生成树中的一条边。

设(u,v)为横跨切割$(S,V-S)$的一条边，由于(u,v)是连通图G中一条权重最小的边，因此(u,v)为横跨切割$(S,V-S)$的一条轻量边，根据定理23.1，对于A，(u,v)是安全的。(u,v)为G的某棵最小生成树中的一条边。

23.1-5

> 设 e 为连通图 G(V,E) 的某一条环路上权重最大的边，证明 G'=(V,E−{e}) 中存在一棵最小生成树，该生成树同时也是 G 的最小生成树，也就是说， G 中存在一棵不包含 e 的最小生成树。

采用反证法，设T为G的一棵最小生成树，且$e\in T$，(u,v)是横跨G的某一个切割$(S,V-S)$的一条边，但不是轻量边。若从T中移除(u,v)，则T变得不连通。由于e位于某一条环路上，因此$(S,V-S)$必然切割环路另一条边$e\prime$，又由于e为该环路上权重最大的边，有$w(e\prime)\leq w(e)$。根据定理23.1的证明，我们可以构造生成树$T\prime$，由T移除(u,v)并添加(x,y)构成，有：

$$w(T')=w(T)-w(e)+w(e')\leq w(T)$$

由于T为最小生成树，因此$T\prime$一定为最小生成树，当且仅当$w(e\prime)=w(e)$时成立，也就是说，G中存在一棵不包含e的最小生成树。得证。

23.2-1

> 对于同一个输人图，Kruskal算法返回的最小生成树可以不同，这种不同来源于对边进行排序时，对权重相同的边的进行不同的处理。证明：对于图 G 的每棵最小生成树 T，都存在一种办法来对 G 的边的进行排序，使得Kruskal算法所返回最小生成树就是 T。

假设我们想选择T作为我们的最小生成树。然后，为了使用 Kruskal 算法获得这棵树，我们将首先根据边的权重对边进行排序，然后通过首先选择一条边（如果它包含在最小生成树中），并将所有不包含的在最小生成树T的边权重视作比在最小生成树中的边权重更大。

通过这种排序，我们仍然会找到一棵与所有最小生成树具有相同权重的树$w(T)$. 然而，由于我们优先考虑T中的边，我们知道我们将在可能位于其他最小生成树中的任何其他边上选择它们。

或者新增属性以区分权重相同的不同边。

23.2-2

```
double calculateMST(int numNodes, double **arr){
    bool selected[numNodes];
    double minWeight[numNodes];
    double totalWeight = 0;       // 最小生成树的边权重之和

    for (int i = 0; i < numNodes; ++i) {
        selected[i] = false;
        minWeight[i] = INT_MAX;
    }
    minWeight[0] = 0;
```

```
    for (int i = 0; i < numNodes; ++i) {
        int minNode = -1;
        double minWeightValue = INT_MAX;
        for (int j = 0; j < numNodes; ++j) {
            if(!selected[j] && minWeight[j]<minWeightValue){
                minNode = j;
                minWeightValue = minWeight[j];
            }
        }
        if(minNode == -1){
            return INT_MAX;
        }
        selected[minNode] = true;
        totalWeight += minWeightValue;
        for (int j = 0; j < numNodes; ++j) {
            if(!selected[j] && arr[minNode][j] < minWeight[j]){
                minWeight[j] = arr[minNode][j];
            }
        }
    }
    return totalWeight;
}
```