第19章 二 项 堆

本章和第20章要介绍称为可合并堆(mergeable heap)的数据结构，这些数据结构支持下面五种操作：

MAKE-HEAP()：创建并返回一个不包含任何元素的新堆。

INSERT(H，x)：将结点 x(其关键字域中已填入了内容)插入堆 H 中。

MINIMUM(H)：返回一个指向堆 H 中包含最小关键字的结点的指针。

EXTRACT-MIN(H)：将堆 H 中包含最小关键字的结点删除，并返回一个指向该结点的指针。

UNION(H_1，H_2)：创建并返回一个包含堆 H_1 和 H_2 中所有结点的新堆。同时，H_1 和 H_2 被这个操作"删除"。

另外，这两章里的数据结构还支持下面两种操作：

DECREASE-KEY(H，x，k)：将新关键字值 k(假定它不大于当前的关键字值)赋给堆 H 中的结点 x。[⊖]

DELETE(H，x)：从堆 H 中删除结点 x。

如图19-1中的表格所示，如果不需要UNION操作，则普通的二叉堆(如第6章)堆排序中用到的性能就很好。在一个二叉堆上，非UNION操作的最坏情况运行时间为 $O(\lg n)$(或更好)。但是，如果某个应用中一定要用UNION操作，则二叉堆就不能令人满意了。UNION操作先将包含两个要合并的堆的数组并置，然后运行MIN-HEAPIFY(见练习6.2-2)；在最坏情况下，UNION操作的运行时间为 $\Theta(n)$。

过程	二叉堆(最坏情况)	二项堆(最坏情况)	斐波那契堆(平摊)
MAKE-HEAP	$\Theta(1)$	$\Theta(1)$	$\Theta(1)$
INSERT	$\Theta(\lg n)$	$\Omega(\lg n)$	$\Theta(1)$
MINIMUM	$\Theta(1)$	$\Omega(\lg n)$	$\Theta(1)$
EXTRACT-MIN	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$
UNION	$\Theta(n)$	$\Omega(\lg n)$	$\Theta(1)$
DECREASE-KEY	$\Theta(\lg n)$	$\Theta(\lg n)$	$\Theta(1)$
DELETE	$\Theta(\lg n)$	$\Theta(\lg n)$	$O(\lg n)$

图19-1 可合并堆的三种实现中各操作的运行时间。在执行某一操作时堆中的项目数用 n 表示

在这一章中，我们要讨论"二项堆"(binomial heap)，其最坏情况时间界也如图19-1所示。特别地，UNION操作只要 $O(\lg n)$ 时间就可完成包含 n 个元素的两个二项堆的合并。

在第20章中，我们将讨论斐波那契堆，对某些操作它具有更好的时间界。但请注意，图19-1中斐波那契堆的运行时间是平摊时间界，而不是每个操作的最坏情况时间界。

本章略去在插入前分配结点和删除后释放结点的问题。我们假定这些细节由调用堆过程的程序来处理。

⊖ 正如第五部分导言中所提到的，默认的可合并堆为可合并最小堆，因此，操作MINIMUM，EXTRACT-MIN和DECREASE-KEY都能够应用。或者，也可以定义一个具有操作MAXIMUM，EXTRACT-MAX和INCREASE-KEY的可合并最大堆。

从对 SEARCH 操作的支持方面来看，二叉堆、二项堆和斐波那契堆都是低效的。在这些结构中，要找到一个包含给定关键字的结点可能花一些时间。因为这个原因，在 DECREASE-KEY 和 DELETE 等涉及一个给定结点的操作需要一个指向该结点的指针作为输入的一部分。如 6.5 节关于优先队列的讨论中一样，当在一个应用中使用可合并堆时，我们通常将对应应用对象的柄(handle)存入每个可合并堆的元素中，同时也将对应可合并堆元素的柄存入每个应用对象。这些柄的确切性质取决于具体的应用及其实现。

19.1 节先定义二项树，再定义二项堆。另外，还要介绍二项堆的一种特别表示。19.2 节说
456 明如何以图 19-1 给出的时间界实现二项堆上的操作。

19.1　二项树与二项堆

一个二项堆由一组二项树所构成，故这一节先定义二项树并证明它们的一些关键性质，然后定义二项堆，并说明如何表示它们。

19.1.1　二项树

二项树 B_k 是一种递归定义的有序树(见 B.5.2 节)。如图 19-2a 所示，二项树 B_0 只包含一个结点。二项树 B_k 由两棵二项树 B_{k-1} 连接而成：其中一棵树的根是另一棵树的根的最左孩子。图 19.2b显示从 B_0 到 B_4 的二项树。

下面的引理给出二项树的一些性质。

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。

证明：对 k 进行归纳。对每一个性质，基都是二项树 B_0。验证每个性质对 B_0 成立是很容易的。

对归纳步骤，假设本引理对 B_{k-1} 成立。

1)二项树 B_k 包含两个 B_{k-1}，故 B_k 有 $2^{k-1}+2^{k-1}=2^k$ 个结点。

2)根据由两个 B_{k-1} 连接成 B_k 的方式，可知 B_k 结点的最大深度比 B_{k-1} 中的最大深度大 1。根据归纳假设，这个最大深度为 $(k-1)+1=k$。

3)设 $D(k, i)$ 表示二项树 B_k 在深度 i 处的结点数。因为 B_k 由两个 B_{k-1} 连接而成，故 B_{k-1}
457 中深度 i 处的某个结点在 B_k 中深度 i 和 $i+1$ 处各出现一次。换句话说，在 B_k 中，深度 i 处的结点个数为 B_{k-1} 中深度 i 处结点个数与 B_{k-1} 中深度 $i-1$ 处结点个数之和。这样，

$$\begin{aligned} D(k,i) &= D(k-1,i)+D(k-1,i-1) && \text{(根据递归假设)} \\ &= \binom{k-1}{i}+\binom{k-1}{i-1} && \text{(根据练习 C.1-7)} \\ &= \binom{k}{i} \end{aligned}$$

4)在 B_k 中，度数大于在 B_{k-1} 中的度数的唯一结点就是根，它在 B_k 中的子女比在 B_{k-1} 中多一个。因为 B_{k-1} 的根的度数为 $k-1$，故 B_k 的根的度数为 k。根据归纳假设，同时也像图 19-2c 所示的那样，从左至右看，B_{k-1} 的根的各子女分别为 B_{k-2}，B_{k-3}，…，B_0 的根。所以，当将 B_{k-1} 与另一 B_{k-1} 连接时，所得的根的子女为 B_{k-1}，B_{k-2}，…，B_0 的根。 ■

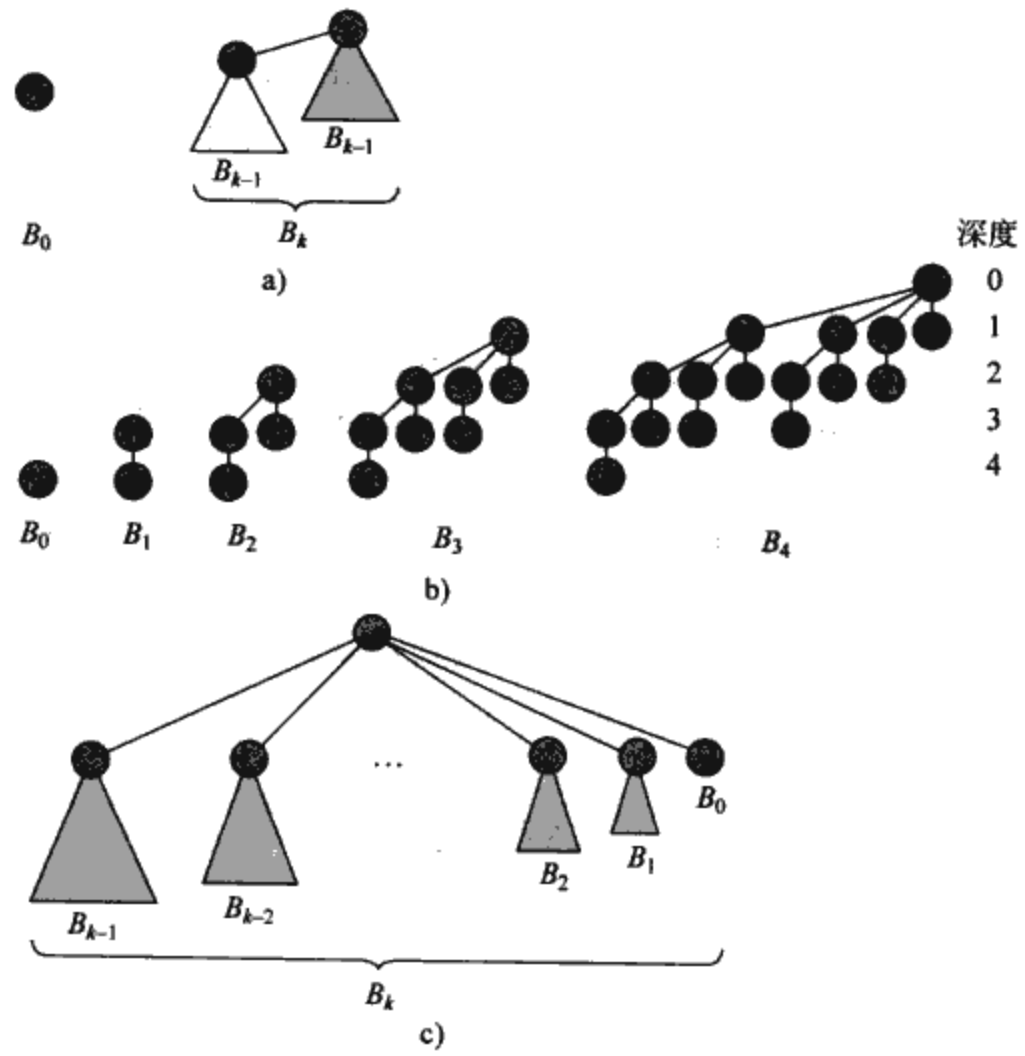


图 19-2　a)二项树 B_k 的递归定义，三角形表示有根的子树。b)二项树 B_0 至 B_4，B_4 中示出了各结点的深度。c)以另一种方式来看二项树 B_k

458

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 $\lg n$。

证明：由引理 19.1 的性质 1 和性质 4 直接可得。 ■

术语“二项树”是从引理 19.1 的性质 3 而来，因为项 $\binom{k}{i}$ 为二项系数。练习 19.1-3 对此术语作了进一步的说明。

19.1.2　二项堆

二项堆 H 由一组满足下面的二项堆性质的二项树组成。

1)H 中的每个二项树遵循最小堆性质：结点的关键字大于或等于其父结点的关键字。我们说这种树是最小堆有序的。

2)对任意非负整数 k，在 H 中至多有一棵二项树的根具有度数 k。

第一个性质告诉我们，在一棵最小堆有序的二项树中，其根包含了树中最小的关键字。

根据第二个性质可知，在包含 n 个结点的二项堆 H 中，包含至多 $\lfloor \lg n \rfloor + 1$ 棵二项树。为搞清楚这一点，注意 n 的二进制表示中有 $\lfloor \lg n \rfloor + 1$ 位，例如 $\langle b_{\lfloor \lg n \rfloor}, b_{\lfloor \lg n \rfloor - 1}, \cdots, b_0 \rangle$，从而 $n = \sum_{i=0}^{\lfloor \lg n \rfloor} b_i 2^i$。所以，根据引理 19.1 中的性质 1 可知，二项树 B_i 出现于 H 中，当且仅当位 $b_i = 1$。这样，二项堆 H 包含至多 $\lfloor \lg n \rfloor + 1$ 棵二项树。

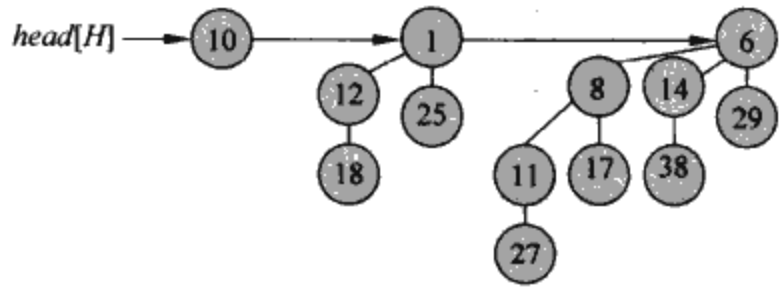
图 19-3a 示出了包含 13 个结点的二项堆 H。13 的二进制表示为 $\langle 1101 \rangle$，故 H 包含了最小堆有序二项树 B_3，B_2 和 B_0，它们分别有 8，4 和 1 个结点，即共有 13 个结点。

二项堆的表示

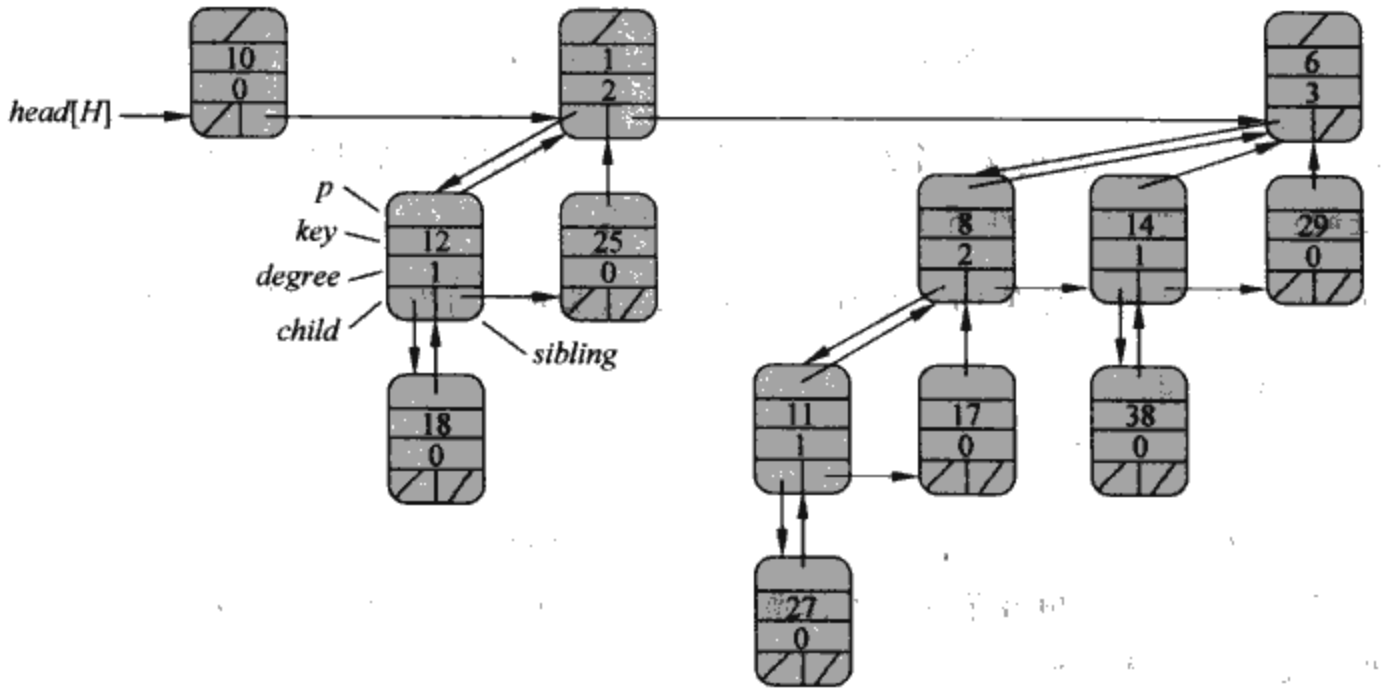
如图 19-3b 所示，二项堆中的每棵二项树都按 10.4 节中左孩子，右兄弟的表示方式存储。在每个结点中，都有一个关键字域及其他依应用要求而定的卫星数据。另外，每个结点 x 还包含了指向其父结点的指针 $p[x]$，指向其最左孩子的指针 $child[x]$，以及指向 x 的紧右兄弟的指针 $sibling[x]$。如果结点 x 是根，则 $p[x]=\text{NIL}$。如果结点 x 没有子女，则 $child[x]=\text{NIL}$。如果 x 是其父结点的最右孩子，则 $sibling[x]=\text{NIL}$。每个结点 x 都包含域 $degree[x]$，即 x 的子女个数。

459 如图 19-3 所示，一个二项堆中的各二项树的根被组织成一个链表，我们称之为根表。在遍历根表时，各根的度数是严格递增的。根据第二个二项堆性质，在一个 n 结点的二项堆中各根的度数构成了 $\{0, 1, \cdots, \lfloor \lg n \rfloor\}$ 的一个子集。对根结点与非根结点来说，*sibling* 域的含义是不同的。如果 x 为根，则 $sibling[x]$ 指向根表中下一个根（像通常一样，如果 x 为根表中最后一个根，则 $sibling[x]=\text{NIL}$）。

一个给定的二项堆 H 可通过域 $head[H]$ 来存取接；这个域即指向 H 的根表中第一个根的指针。如果二项堆 H 中没有元素，则 $head[H]=\text{NIL}$。



a)



b)

图 19-3　一个包含 13 个结点的二项堆。a)一个二项堆包含了二项树 B_0，B_2 和 B_3，它们分别有 8，4 和 1 个结点，即共有 13 个结点。由于每棵二项树都是最小堆有序的，所以任意结点的关键字都不小于其父结点的关键字。图中还示出了根表，它是一个按根的度数递增排序的链表。b)二项堆 H 的一个更具体的表示。每棵二项树按左孩子、右兄弟表示方式存储，每个结点存储自身的度数

460

练习

19.1-1　假设 x 为一个二项堆中，某棵二项树中的一个结点，并假定 $sibling[x]\neq\text{NIL}$。如果 x 不是根，则 $degree[sibling[x]]$ 与 $degree[x]$ 相比怎样？如果 x 是个根呢？

19.1-2　如果 x 是二项堆的某棵二项树的一个非根结点，$degree[p[x]]$与 $degree[x]$相比怎样？

19.1-3　如图 19-4 所示，假设按后序遍历顺序，将二项树 B_k 中的结点标为二进制形式。考虑深度 i 处标为 l 的一个结点 x，且设 $j=k-i$。证明：在 x 的二进制表示中共有 j 个 1。恰包含 j 个 1 的二进制 k 串共有多少？证明 x 的度数与 l 的二进制表示中，最右 0 的右边的 1 的个数相同。

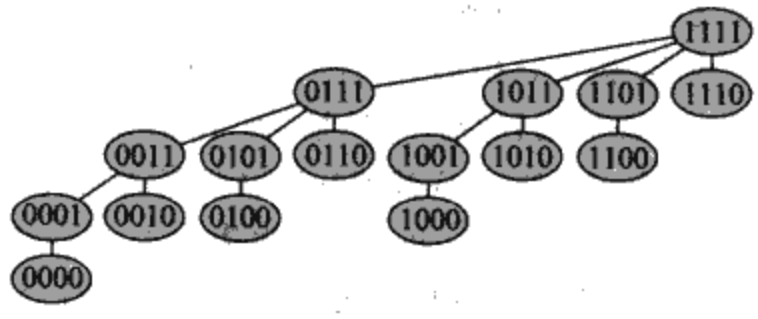


图 19-4　二项树 B_4，其各结点按后序遍历次序标以一个二进制数

19.2　对二项堆的操作

这一节里，要介绍如何在图 19-1 中给出的时间界内，执行对二项堆的各种操作。我们将只给出上界，下界留作练习 19.2-10。

创建一个新二项堆

为了构造一个空的二项堆，过程 MAKE-BINOMIAL-HEAP 分配并返回一个对象 H，且 $head[H]=$NIL。该过程的运行时间为 $\Theta(1)$。

461

寻找最小关键字

过程 BINOMIAL-HEAP-MINIMUM 返回一个指针，它指向包含 n 个结点的二项堆 H 中具有最小关键字的结点。这个实现假设没有一个关键字为∞(见练习 19.2-5)。

```
BINOMIAL-HEAP-MINIMUM(H)
1  y ← NIL
2  x ← head[H]
3  min ← ∞
4  while x≠NIL
5      do if key[x]<min
6           then min ← key[x]
7                y ← x
8         x ← sibling[x]
9  return y
```

因为一个二项堆是最小堆有序的，故最小关键字必在根结点中。过程 BINOMIAL-HEAP-MINIMUM 检查所有的根(至多有 $\lfloor \lg n \rfloor +1$)，将当前最小者存于 min 中，而将指向当前最小者的指针存于 y 之中。当对图 19-3 中二项堆调用时，BINOMIAL-HEAP-MINIMUM 返回一个指向具有关键字 1 的结点的指针。

因为至多要检查 $\lfloor \lg n \rfloor +1$ 个根，故 BINOMIAL-HEAP-MINIMUM 的运行时间为 $O(\lg n)$。

合并两个二项堆

合并两个二项堆的操作可用作后面大部分操作的一个子程序。过程 BINOMIAL-HEAP-UNION 反复连接根结点的度数相同的各二项树。在下面的过程，将以结点 y 为根的 B_{k-1} 树与以结点 z 为根的 B_{k-1} 树连接起来；亦即，它使得 z 成为 y 的父结点，并成为一棵 B_k 树的根。

```
BINOMIAL-LINK(y, z)
1  p[y] ← z
2  sibling[y] ← child[z]
3  child[z] ← y
4  degree[z] ← degree[z]+1
```

过程 BINOMIAL-LINK 在 $O(1)$时间内，使得结点 y 成为结点 z 的子女链表的新头。这个过程之所以能正确地运行，是因为每棵二项树的左孩子、右兄弟表示与树的排序性质正好匹配：在
462 一棵 B_k 树中，根的最左孩子是一棵 B_{k-1} 树的根。

下面的过程合并二项堆 H_1 和 H_2，并返回结果堆。在合并过程中，它也同时破坏了 H_1 和 H_2 的表示。除了 BINOMIAL-LINK 之外，这个过程还使用了一个辅助过程 BINOMIAL-HEAP-MERGE，来将 H_1 和 H_2 的根表合并成一个按度数的单调递增次序排列的链表。BINOMIAL-HEAP-MERGE(其伪代码留作练习 19.2-1)与 2.3.1 节中的 MERGE 过程是类似的。

```
BINOMIAL-HEAP-UNION(H1, H2)
 1  H ← MAKE-BINOMIAL-HEAP()
 2  head[H] ← BINOMIAL-HEAP-MERGE(H1, H2)
 3  free the objects H1 and H2 but not the lists they point to
 4  if head[H]=NIL
 5    then return H
 6  prev-x ← NIL
 7  x ← head[H]
 8  next-x ← sibling[x]
 9  while next-x≠NIL
10      do if(degree[x]≠degree[next-x])or
             (sibling[next-x]≠NIL and degree[sibling[next-x]]=degree[x])
11           then prev-x ← x                                  ▷ Cases 1 and 2
12                x ← next-x                                  ▷ Cases 1 and 2
13           else if key[x]≤key[next-x]
14                  then sibling[x] ← sibling[next-x]         ▷ Case 3
15                       BINOMIAL-LINK(next-x, x)             ▷ Case 3
16           else if prev-x=NIL                               ▷ Case 4
17                  then head[H] ← next-x                     ▷ Case 4
18                  else sibling[prev-x] ← next-x             ▷ Case 4
19                BINOMIAL-LINK(x, next-x)                    ▷ Case 4
20                x ← next-x                                  ▷ Case 4
21        next-x ← sibling[x]
22  return H
```

图 19-5 示出了 BINOMIAL-HEAP-UNION 的一个例子，其中出现了代码中给出的所有四种情况。

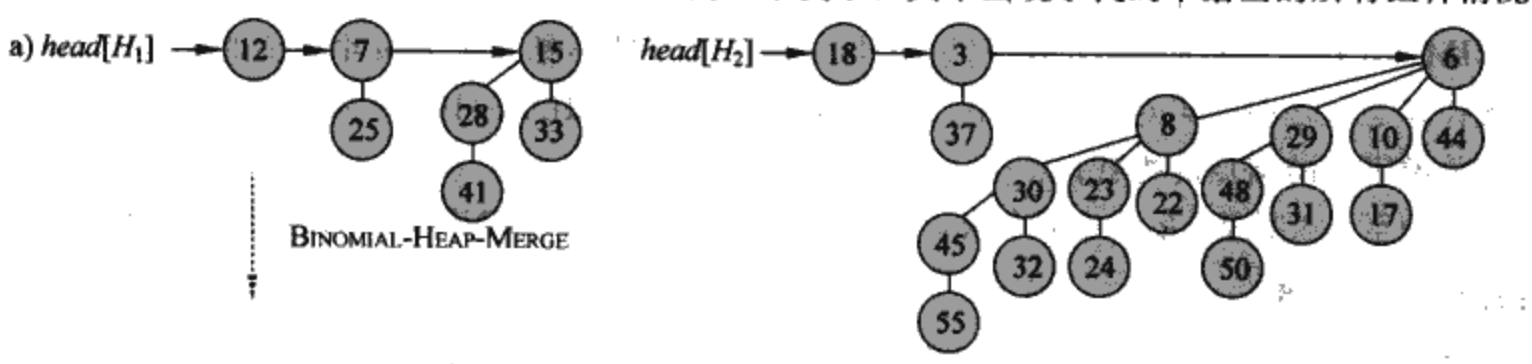


图 19-5　BINOMIAL-HEAP-UNION 的执行过程。a)二项堆 H_1 和 H_2。b)二项堆 H 是 BINOMIAL-HEAP-MERGE(H_1, H_2)的输出。开始时，x 是 H 的根表上的第一个根。因为 x 和 $next\text{-}x$ 的度数都为 0，且 $key[x]<key[next\text{-}x]$，这与情况 3 对应。c)在链接发生后，$x$ 是具有相同度数的三个根中的第一个，这与情况 2 对应。d)在根表中所有指针都向下移动一个位置后，情况 4 适用，因为 x 是两个具有相同度数的根中的第一个。e)在发生链接后，情况 3 适用。f)在另一次链接后，情况 1 适用，因为 x 的度数为 3，$next\text{-}x$ 的度数为 4。While 循环的这一次迭代也是最后一次，因为在根表中的各指针向下移动一个位置后，$next\text{-}x=$NIL

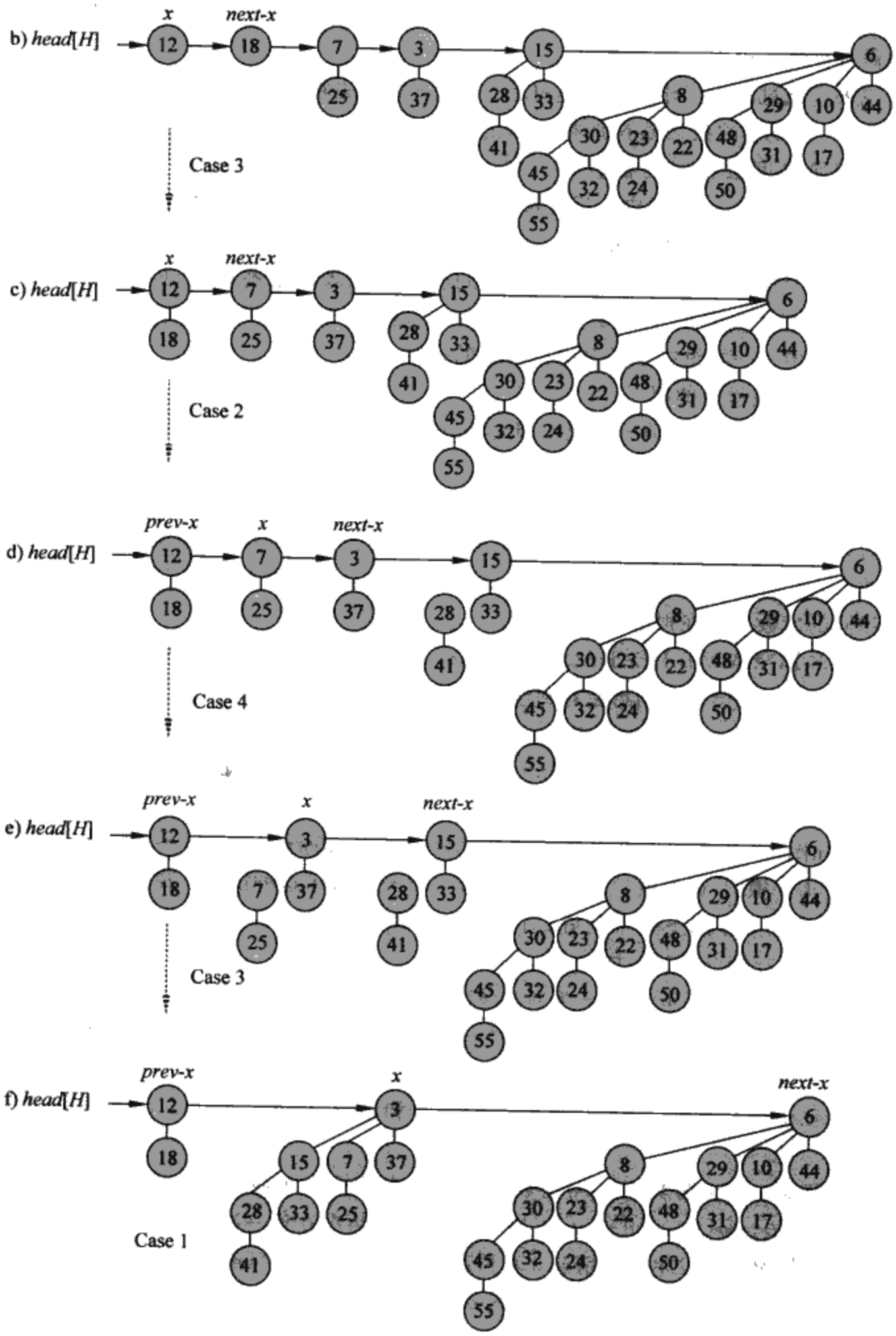


图 19-5　（续）

BINOMIL-HEAP-UNION 过程有两个阶段。第一个阶段中执行对 BINOMIAL-HEAP-MERGE 的调用，将二项堆 H_1 和 H_2 的根表合并成一个链表 H，它按度数排序成单调递增次序。然而，对于每一个度数值，可能有两个根(但不可能更多了)与其对应，所以第二阶段将相等

度数的根连接起来，直到每个度数至多有一个根时为止。因为链表 H 是按度数排序的，所以可以很快地做各种链表操作。

具体说来，该过程的工作过程是这样的。第 1～3 行先将二项堆 H_1 和 H_2 的根表合并成一个根表 H。H_1 和 H_2 的根表是按严格递增的度数来排序的，而 BINOMIAL-HEAP-MERGE 返回一个按单调递增度数排序的根表 H。如果 H_1 和 H_2 的根表共有 m 个根，则 BINOMIAL-HEAP-MERGE 可在 $O(m)$ 时间内，反复检查两个根表头上的根，并将度数较低的根添加至输出根表中，同时将其从输入根表中去掉。

过程 BINOMIAL-HEAP-UNION 接着对指向 H 的根表的某些指针进行初始化。首先，在第 4～5 行中，如果正好是在合并两个空二项堆，则返回。那么，从第 6 行开始，H 至少有一个根。在整个过程中，一共维护了三个指向根表的指针：

- x 指向目前被检查的根
- *prev-x* 指向根表中 x 前一个根，即 $sibling[prev\text{-}x]=x$（因为初始时 x 没有前驱，我们开始将 *prev-x* 置为 NIL）
- *next-x* 指向根表中 x 的后一个根，即 $sibling[x]=next\text{-}x$

463～465

开始时，对一个给定度数的根表 H 上至多有两个根：因为 H_1 和 H_2 为两个二项堆，对一个给定的度数，它们都各只含有一个根。另外，BINOMIAL-HEAP-MERGE 保证了如果 H 中的两个根具有相同的度数，则在根表中是邻居。

实际上，在 BINOMIAL-HEAP-UNION 的执行过程中，在有的时刻，H 根表可能会出现三个根具有相同的度数。待会儿我们将会看到这种情况是怎么发生的。在第 9～21 行中的 while 循环的每一次迭代中，要根据 x 和 *next-x*（甚至可能 $sibling[next\text{-}x]$）的度数，来确定是否把两者连接起来。该循环的一种不变式是每进入循环体时，x 和 *next-x* 都是非 NIL 的（具体的循环不变式可参见练习 19.2-4）。

情况 1，如图 19-6a 所示，发生的条件是 $degree[x]\neq degree[next\text{-}x]$；亦即，$x$ 为 B_k 树的根，而 *next-x* 为一棵 B_l 树的根，且 $l>k$。第 11～12 行处理了这种情况。我们并不连接 x 和 *next-x*，只是将指针指向表中的下一个位置。在第 21 行中更新 *next-x*，使之指向新结点 x 之后的结点。这个处理对所有情况都是一样的。

情况 2，如图 19-6b 所示，当 x 为具有相同度数的三个根中的第一个时发生；亦即，当 $degree[x]=degree[next\text{-}x]=degree[sibling[next\text{-}x]]$ 时发生。对这种情况的处理与情况 1 相同：将指针又一次移向表中下一个位置。下一次迭代将执行情况 3 或情况 4，把三个相等度数的根中第二个和第三个联合起来。第 10 行测试情况 1 和情况 2，第 11～12 行对两种情况都作了处理。

情况 3 和情况 4 当 x 为具有相同度数的两个根中的第一个时发生；亦即，当 $degree[x]=degree[next\text{-}x]\neq degree[sibling[next\text{-}x]]$ 时发生。这些情况在任一次迭代中都可能发生，但是其中之一总会紧随情况 2 而发生。在情况 3 和情况 4 中，我们将 x 与 *next-x* 相连接。这两种情况根据 x 和 *next-x* 哪一个具有更小的关键字来区分；这个区分条件决定在这两个根连接后哪一个结点将成为根。

在情况 3 中，如图 19-6c 所示，$key[x]\leqslant key[next\text{-}x]$，故将 *next-x* 连接到 x 上。第 14 行将 *next-x* 从根表中去掉，第 15 行使 *next-x* 成为 x 的最左孩子。

在情况 4 中，如图 19-6d 所示，*next-x* 具有更小的关键字，故 x 被连接到 *next-x* 上。第 16～18 行将 x 从根表中去掉；又有两种情况取决于 x 是（第 17 行）还是不是（第 18 行）根表中的第一个根。第 19 行使 x 成为 *next-x* 的最左孩子，第 20 行更新 x 以进入下一轮迭代。

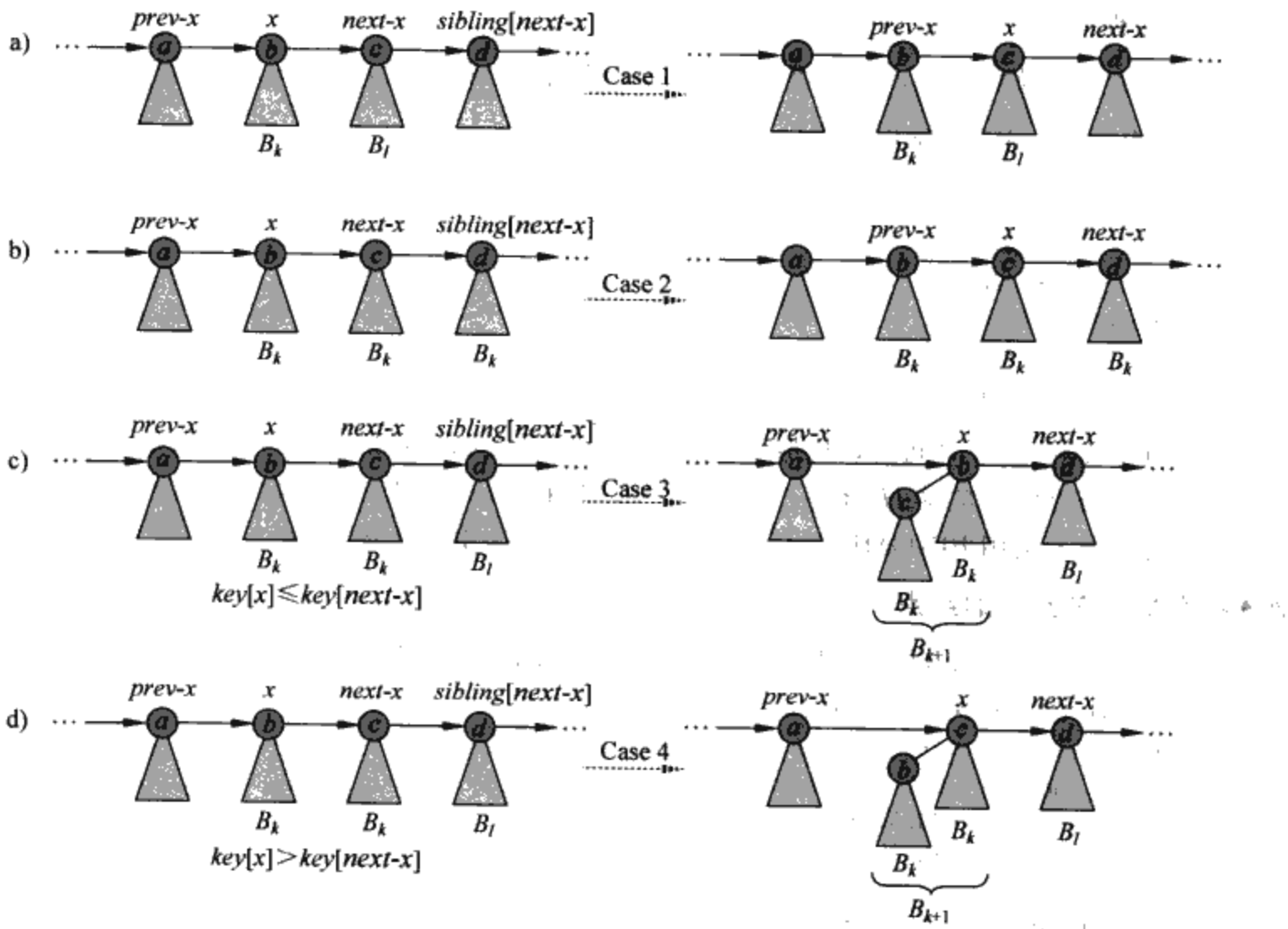


图 19-6　在 BINOMIAL-HEAP-UNION 中发生的四种情况，标记 a，b，c 和 d 仅用于区别所牵涉到的根；它们并不表示这些根的关键字或度数。在每种情况中，x 是一棵 B_k 树的根，且 $l>k$。a)情况 1：$degree[x]\neq degree[next\text{-}x]$。指针移向根表中的下一个位置。b)情况 2：$degree[x]=degree[next\text{-}x]=degree[sibling[next\text{-}x]]$。指针再一次移向根表中的下一个位置，而且下一次迭代将执行情况 3 或情况 4。c)情况 3：$degree[x]=degree[next\text{-}x]\neq degree[sibling[next\text{-}x]]$，且 $key[x]\leqslant key[next\text{-}x]$。将 $next\text{-}x$ 从根表中去掉，并将其连接到 x 上，构造一棵 B_{k+1} 树。d)情况 4：$degree[x]=degree[next\text{-}x]\neq degree[sibling[next\text{-}x]]$，且 $key[next\text{-}x]\leqslant key[x]$。我们将 x 从根表中去掉，并将其连接到 $next\text{-}x$ 上，也构造一棵 B_{k+1} 树

在情况 3 或情况 4 之后，为 while 循环的下一次执行所做的设置是相同的。我们刚刚连接了两个 B_k 树以形成一棵 B_{k+1} 树，即现在 x 所指向的树。在 BINOMIAL-HEAP-MERGE 输出的根表中已有 0，1 或 2 个其他的 B_{k+1} 树，故现在 x 就为根表上 1，2 或 3 棵 B_{k+1} 树中的第一个。如果仅有 x 一个，则在下一轮循环中进入情况 1：$degree[x]\neq degree[next\text{-}x]$。如果 x 是两个中的第一个，则在下一轮循环中进入情况 3 或情况 4。当 x 为三个中的第一个时，才在下一轮循环中进入情况 2。

BINOMIA-HEAP-UNION 的运行时间为 $O(\lg n)$，其中 n 为二项堆 H_1 和 H_2 中总的结点数。设 H_1 包含 n_1 个结点，H_2 包含了 n_2 个结点，故 $n=n_1+n_2$。又因为 H_1 至多包含 $\lfloor \lg n_1 \rfloor +1$ 个根，而 H_2 至多包含 $\lfloor \lg n_2 \rfloor +1$ 个根，故在调用了 BINOMIAL-HEAP-MERGE 后，H 包含至多 $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor +2 \leqslant 2\lfloor \lg n \rfloor +2=O(\lg n)$ 个根。因此，执行 BINOMIAL-HEAP-MERGE 的时间为 $O(\lg n)$。while 循环的每次迭代需要 $O(1)$ 时间，又因为在每次循环中，或者将指针指向 H 根表中的下一位，或从根表中去掉一个根，于是就总共要执行至多 $\lfloor \lg n_1 \rfloor + \lfloor \lg n_2 \rfloor +2$ 次迭代。所以，总的时间为 $O(\lg n)$。

466
~
467

插入一个结点

下面的过程将结点 x 插入二项堆 H 中，假定结点 x 已被分配，且 $key[x]$也已填有内容。

```
BINOMIAL-HEAP-INSERT(H, x)
1  H' ← MAKE-BINOMIAL-HEAP()
2  p[x] ← NIL
3  child[x] ← NIL
4  sibling[x] ← NIL
5  degree[x] ← 0
6  head[H'] ← x
7  H ← BINOMIAL-HEAP-UNION(H, H')
```

这个过程先在 $O(1)$时间内，构造一个只包含一个结点的二项堆 H'，再在 $O(\lg n)$时间内，将其与包含 n 个结点的二项堆 H 合并。对 BINOMIAL-HEAP-UNION 的调用还负责释放临时二项堆 H'。(另一种不调用 BINOMIAL-HEAP-UNION 的直接实现在练习 19.2-8 中给出。)

抽取具有最小关键字的结点

下面的过程从二项堆 H 中抽取具有最小关键字的结点，并返回一个指向该结点的指针。

```
BINOMIAL-HEAP-EXTRACT-MIN(H)
1  find the root x with the minimum key in the root list of H,
       and remove x from the root list of H
2  H' ← MAKE-BINOMIAL-HEAP()
3  reverse the order of the linked list of x's children,
       and set head[H'] to point to the head of the resulting list
4  H ← BINOMIAL-HEAP-UNION(H, H')
5  return x
```

这个过程的工作如图 19-7 所示。输入二项堆 H 如图 19-7a 所示；图 19-7b 示出了第 1 行后的情形：具有最小关键字的根 x 被从 H 的根表中去掉。如果 x 为一棵 B_k 树的根，则根据引理 19.1的性质 4，x 的各子女从左到右分别为 B_{k-1}，B_{k-2}，…，B_0 树的根。图 19-7c 说明了通
468 过在第 3 行中逆转 x 的子女表，得到一个包含 x 树中除 x 而外的每个结点的二项堆 H'。因为在第 1 行中 x 树已从 H 中去掉，故第 4 行中合并 H 和 H'所得的结果二项堆(如图 19-7d 所示)包含原先 H 中除 x 以外的所有结点。最后，第 5 行返回 x。

因为如果 H 有 n 个结点，则第 1～4 行每次需时间 $O(\lg n)$。所以，BINOMIAL-HEAP-EXTRACT-MIN 的运行时间为 $O(\lg n)$。

减小关键字的值

下面的过程将二项堆 H 中的某一结点 x 的关键字减小为一个新值 k。如果 k 大于 x 的当前关键字值，这个过程就引发一个错误。

```
BINOMIAL-HEAP-DECREASE-KEY(H, x, k)
 1  if k > key[x]
 2    then error "new key is greater than current key"
 3  key[x] ← k
 4  y ← x
 5  z ← p[y]
 6  while z ≠ NIL and key[y]<key[z]
 7      do exchange key[y] ↔ key[z]
 8         ▷ If y and z have satellite fields, exchange them, too.
 9         y ← z
10         z ← p[y]
```

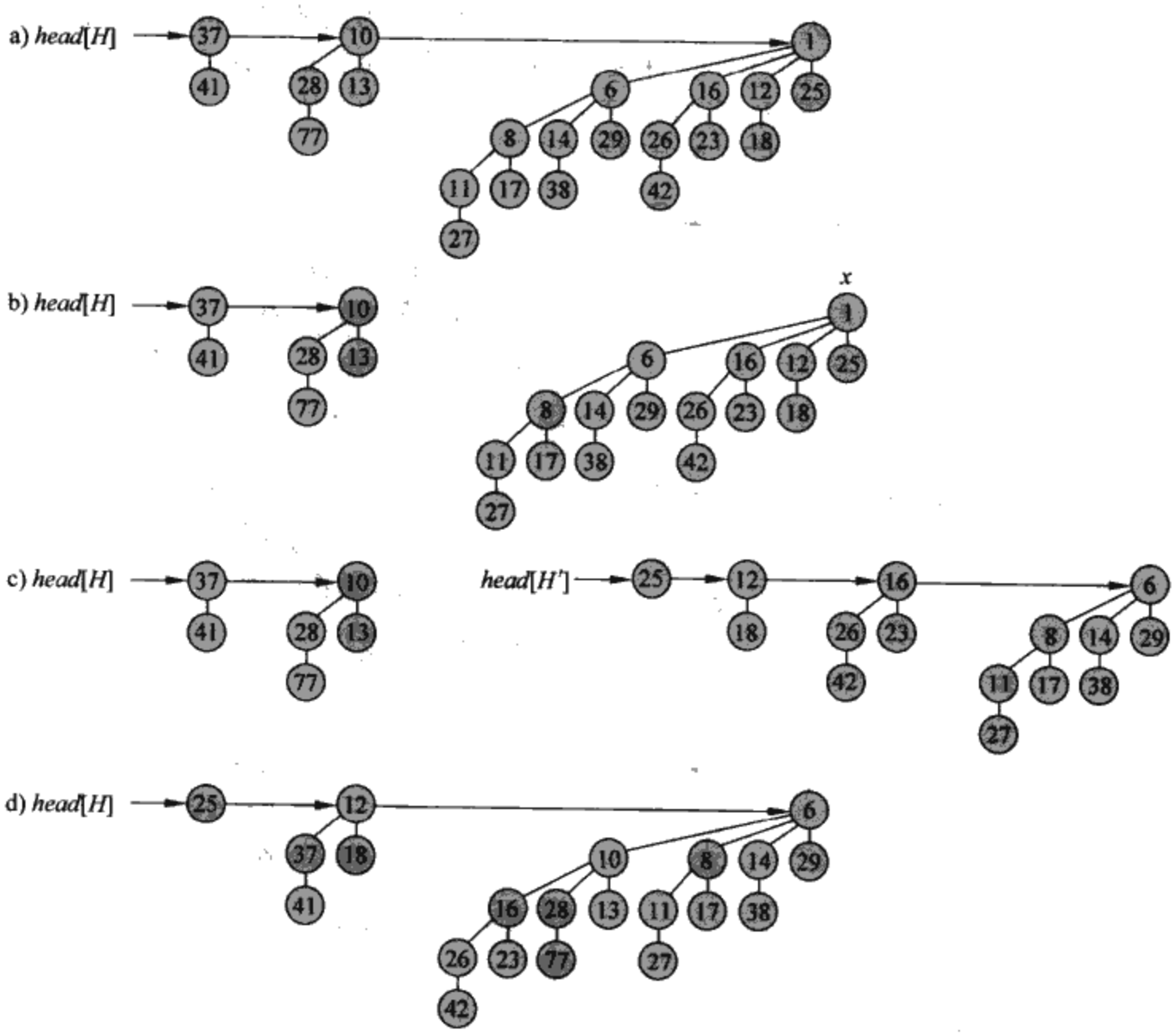


图 19-7　BINOMIAL-HEAP-EXTRACT-MIN 的执行过程。a)一个二项堆 H。b)具有最小关键字的根 x 被从 H 的根表中去掉。c)x 的子女结点链表被逆转，形成另一个二项堆 H'。d)H 和 H' 合并的结果

如图 19-8 所示，这个过程以与二叉最小堆中相同的方式来减小一个关键字，使该关键字在堆中“冒泡上升”。在确保新关键字不大于当前关键字并将其赋给 x 后，该过程就沿树上升。开始时 y 指向结点 x。在第 6～10 行 while 循环的每次迭代中，将 $key[y]$与 y 的父结点 z 的关键字作比较。如果 y 为根或$key[y] \geqslant key[z]$，则该二项树已是最小堆有序。否则，结点 y 就违反了最小堆有序，故将其关键字与其父结点 z 的关键字相交换，同时还要交换其他的卫星数据。然后，这个过程将 y 置为 z，在树中上升一层，并继续下一次循环。

BINOMIAL-HEAP-DECREASE-KEY 过程的时间为 $O(\lg n)$。根据引理 19.1 的性质 2，x 的最大深度为 $\lfloor \lg n \rfloor$，故第 6～10 行的while 循环迭代至多 $\lfloor \lg n \rfloor$ 次。

删除一个关键字

很容易在 $O(\lg n)$时间内从二项堆 H 中删除一个结点 x 的关键字及卫星数据。在下面的实现中，假定当前在二项堆中的所有结点的关键字都不为$-\infty$。

```
BINOMIAL-HEAP-DELETE(H, x)
1  BINOMIAL-HEAP-DECREASE-KEY(H, x, -∞)
2  BINOMIAL-HEAP-EXTRACT-MIN(H)
```

这个过程使结点 x 在整个二项堆中具有唯一最小的关键字，即给其一个关键字$-\infty$。(练

469
~
470

习 19.2-6处理了$-\infty$不能作为关键字出现的情形，包括暂时作为关键字的情形。)然后，通过调用 BINOMIAL-HEAP-DECREASE-KEY，来使该关键字及其卫星数据信息冒泡上升至树根。接着，再通过调用 BINOMIAL-HEAP-EXTRACT-MIN 将根从 H 中去掉。

过程 BINOMIAL-HEAP-DELETE 的时间为 $O(\lg n)$。

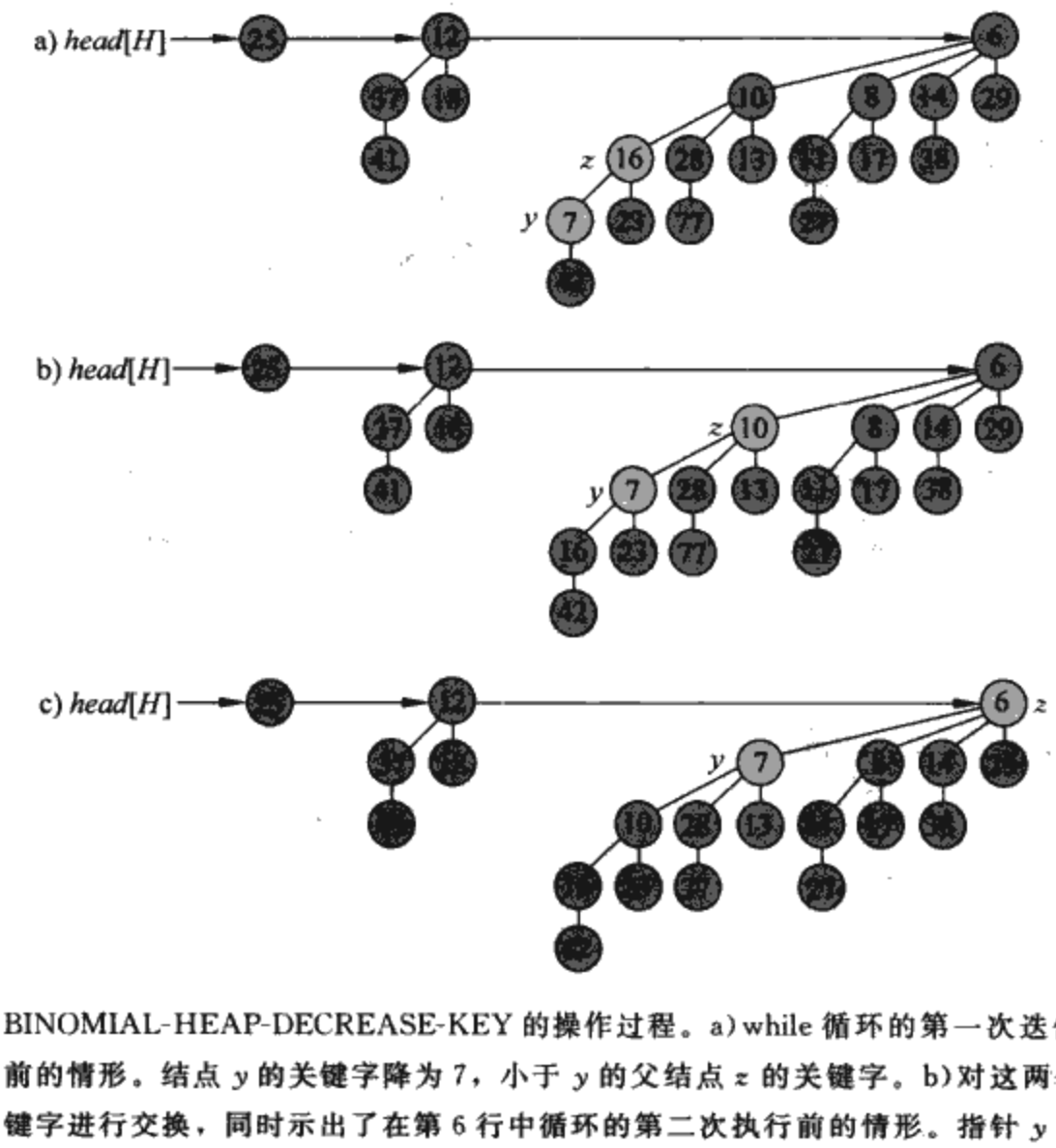


图 19-8　BINOMIAL-HEAP-DECREASE-KEY 的操作过程。a)while 循环的第一次迭代(第 6 行)前的情形。结点 y 的关键字降为 7，小于 y 的父结点 z 的关键字。b)对这两个结点的关键字进行交换，同时示出了在第 6 行中循环的第二次执行前的情形。指针 y 和 z 移向树中的上一层，但仍然违反最小堆序要求。c)再做一次交换，并将指针 y 和 z 再向上移一层，就可以满足最小堆序的要求，故 while 循环结束

练习

471

19.2-1　请写出 BINOMIAL-HEAP-MERGE 的伪代码。

19.2-2　请给出将关键字为 24 的结点插入如图 19-7d 中所示的二项堆后，所得的结果二项堆。

19.2-3　请给出将关键字为 28 的结点从图 19-8c 中的二项堆中删除后的结果。

19.2-4　讨论使用如下循环不变式时 BINOMIAL-HEAP-UNION 的正确性。

在第 9～21 行每一次 while 循环迭代开始时，x 指向下列之一的根：

- 该度数下唯一的根；
- 该度数下仅有两根中的第一个；
- 该度数下仅有三个根中的第一或第二个。

而且，根表中在 x 的前驱之前的所有根结点都有唯一度数，如果 x 的前驱的度数不同于 x，则它在根表中的度数也是唯一的。最后，根表结点的顺序是按结点度数单调递增的。

19.2-5 请解释，如果关键字的值可以是∞，为什么过程 BINOMIAL-HEAP-MINIMUM 可能无法正确工作？重写这个过程的伪代码，使之在这种情况下也能正常工作。

19.2-6 假设无法表示出关键字$-\infty$。重写 BINOMIAL-HEAP-DELETE 过程，使之在这种情况下能正确地工作。运行时间仍应为 $O(\lg n)$。

19.2-7 讨论二项堆上的插入与一个二进制数增值的关系，以及合并两个二项堆与将两个二进制数相加之间的关系。

19.2-8 根据练习 19.2-7，在不调用 BINOMIAL-HEAP-UNION 的前提下，重写 BINOMIAL-HEAP-INSERT，将一个结点直接插入一个二项堆中。

472

19.2-9 证明：如果将根表按度数排成严格递减序（而不是严格递增序）保存，仍可以在不改变渐近运行时间的前提下实现每一种二项堆操作。

19.2-10 请找出使 BINOMIAL-HEAP-EXTRACT-MIN、BINOMIAL-HEAP-DECREASE-KEY 以及 BINOMIAL-HEAP-DELETE 的运行时间为 $\Omega(\lg n)$的输入。解释为什么 BINOMIAL-HEAP-INSERT，BINOMIAL-HEAP-MINIMUM 以及 BINOMIAL-HEAP-UNION 的最坏运行时间是 $\overset{\infty}{\Omega}(\lg n)$，而不是 $\Omega(\lg n)$（见思考题 3-5）。

思考题

19-1　2-3-4 堆

在第 18 章中介绍了 2-3-4 树，其中每个内结点（非根可能）有两个、三个或四个子女，且所有的叶结点的深度相同。在这个问题里，我们来实现 2-3-4 堆，它支持可合并堆的操作。

2-3-4 堆与 2-3-4 树有些不同之处。在 2-3-4 堆中，关键字仅存在于叶结点中，且每个叶结点 x 仅包含一个关键字于其 $key[x]$域中。另外，叶结点中的关键字之间没有什么特别的次序；亦即，从左至右来看，各关键字可以排成任何次序。每个内结点 x 包含一个值 $small[x]$，它等于以 x 为根的子树的各叶结点中所存储的最小关键字。根 r 包含了一个 $height[r]$域，即树的高度。最后，2-3-4 堆主要是在主存中的，故无需任何磁盘读写。

请实现下面各 2-3-4 堆操作。对包含 n 个元素的 2-3-4 堆，a)～e)中的每个操作的运行时间都应是 $O(\lg n)$。f)中的 UNION 操作的运行时间应是 $O(\lg n)$，其中 n 为两个输入堆中的元素个数。

a) MINIMUM，返回一个指向最小关键字的叶结点的指针。

b) DECREASE-KEY，它将某一给定叶结点 x 的关键字减小为一个给定的值 $k \leqslant key[x]$。

c) INSERT，插入具有关键字 k 的叶结点 x。

d) DELETE，删除一给定叶结点 x。

e) EXTRACT-MIN，抽取具有最小关键字的叶结点。

473

f) UINION，合并两个 2-3-4 堆，返回一个 2-3-4 堆并破坏输入堆。

19-2　采用二项堆的最小生成树算法

第 23 章要介绍两个在无向图中寻找最小生成树的算法。这里我们可以看到如何利用二项堆来设计一个不同的最小生成树算法。

给定一个连通的无向图 $G=(V, E)$，以及权值函数 $w: E \to \mathbf{R}$。称 $w(u, v)$为边(u, v)的权。希望找出 G 的最小生成树：一个无环子集 $T \subseteq E$ 连接 V 中所有结点，且总的权值

$$w(T) = \sum_{(u,v)\in T} w(u,v)$$

为最小。

下面的伪代码构造一个最小生成树 T，可用 23.1 节的技术来证明这个过程是正确的。对 V 的结点保持一个划分$\{V_i\}$，且对每组 V_i，都有一个与 V_i 中结点关联的边的集合

$$E_i \subseteq \{(u,v) : u \in V_i \text{ 或 } v \in V_i\}$$

```
MST(G)
 1  T ← ∅
 2  for each vertex v_i ∈ V[G]
 3       do V_i ← {v_i}
 4          E_i ← {(v_i, v) ∈ E[G]}
 5  while there is more than one set V_i
 6       do choose any set V_i
 7          extract the minimum-weight edge(u, v) from E_i
 8          assume without loss of generality that u ∈ V_i and v ∈ V_j
 9          if i ≠ j
10            then T ← T ∪ {(u, v)}
11                 V_i ← V_i ∪ V_j, destroying V_j
12                 E_i ← E_i ∪ E_j
```

请说明如何用二项堆来实现此算法，以便管理点集和边集。需要对二项堆的表示做改变吗？需要增加图 19-1 中所没有的可合并堆操作吗？给出你的实现的运行时间。

474

本章注记

二项堆是在 1978 年由 Vuillemin[307]提出的。其后，Brown[49，50]对它们的性质做了深入的研究。

475