第11章 优先队列

- 优先队列的定义
- 用字典实现优先队列
- 优先级树和堆
- 用数组实现堆
- 可并优先队列
- 优先队列的应用——哈夫曼编码

学习要点:

- 理解以集合为基础的抽象数据类型优先队列
- 理解用字典实现优先队列的方法
- 理解优先级树和堆的概念
- 掌握用数组实现堆的方法
- 理解以集合为基础的抽象数据类型可并优先队列
- 理解左偏树的定义和概念
- 掌握用左偏树实现可并优先队列的方法
- 掌握堆排序算法

第11章 优先队列

- 优先队列的原型
  - 排队上车，老弱病残者优先上车
  - 排队候诊，危急病人优先就诊
  - 洗相馆为顾客洗照片，加钱加急者优先洗
  - 分时操作系统运行程序，小程序优先
  - 在一个集合中搜索，按元素的某种特征值，大(或小)的优先
  - ……处理或服务时只关心对象中谁的优先级最高
  - 通常的队列是一种优先队列最先到者优先级最高

11.1 优先队列的定义

- 优先队列也是一个以集合为基础的抽象数据类型。
- 优先队列中的每一个元素都有一个优先级值。优先队列中元素**x**的优先级值记为**p(x)**，它可以是一个实数，也可以是一个一般的全序集中的元素。优先级值用来表示该元素出列的优先级。
- 通常约定优先级值小的优先级高。也可以约定优先级值大的优先级高。

11.1 优先队列的定义

- 优先队列支持的基本运算有：
  - **(1)Min(H)**：返回优先队列**H**中具有最小优先级的元素。
  - **(2)Insert(x, H)**：将元素**x**插入优先队列**H**。
  - **(3)DeleteMin(H)**：删除并返回优先队列**H**中具有最小优先级的元素。

11.2 用字典实现优先队列

- 优先队列与字典的相似性与区别：
  - 优先队列中元素的优先级值可以看作是字典中元素的线性序值。
  - 在字典中，不同的元素具有不同的线性序值，其插入运算仅当要插入元素**x**的线性序值与当前字典中所有元素的线性序值都不同时才执行。
  - 对于优先队列来说，不同的元素可以有相同的优先级值。因此，优先队列的插入运算即使在当前优先队列中存在与要插入元素**x**有相同的优先级值的元素时，也要执行元素**x**的插入。

11.2 用字典实现优先队列

- 由于优先队列与字典的相似性,除了散列表之外，所有实现字典的方法都可用于实现优先队列。
  - 用有序链表实现优先队列;**(Insert低效)**
  - 用二叉搜索树实现优先队列;**(Insert,DeleteMin, Min均低效)**
  - 用**AVL**树实现优先队列;**(逻辑复杂)**
  - 用无序链表实现优先队列;**(DeleteMin, Min均低效)**

  ……

  都有缺点。原因在于没有考虑到优先队列的特性。

11.3 优先级树和堆

- 优先队列的特征：
  - **DeleteMin**和**Min**只关心优先级最高的元素
  - **Insert**的元素不要求全局的序关系

  因此实现优先队列的结构只要求方便**DeleteMin**和**Min**，而对**Insert**也只要求不给结构的维护带来太大的麻烦。

  根据这两个特征，人们发明了优先级树。

11.3优先级树和堆

优先级树的概念

优先级树是满足下面的优先级性质的二叉树：

- (1)树中每一结点存储一个元素。
- (2)任一结点中存储的元素的优先级值不大(小)于其儿子结点中存储的元素的优先级值，即父结点的优先级不低于其儿子结点的优先级。

换句话说，越接近根的结点中的元素的优先级越高，越方便被访问，因为根最方便被访问。

相应的优先级树称为极小(大)化优先级树。

11.3优先级树和堆

- 用优先级树实现优先队列仍有不足：
  - **Insert(x)**和**DeleteMin(x)**后对结构的维护，在最坏情况下，仍需**O(h)=O(n)**。
- 如果让优先级树近似满，从而**h=[log n]**达到最小，那么，结果将令人满意：在最坏情况下， **Min( )**将只需**O(1)**, **Insert(x)**和**DeleteMin(x)**后对结构的维护只需**O(logn)**。
- 因而引入堆的概念并用堆来实现优先队列。

11.3优先级树和堆

- 堆的概念：
  - 如果一棵优先级树是一棵近似满二叉树，那么，这棵具有优先级性质的近似满二叉树(外形像堆)就叫做堆。
- 用堆实现优先队列：
  - **Min( )、Insert(x)和DeleteMin(x)**运算的实现

11.4 用数组实现堆

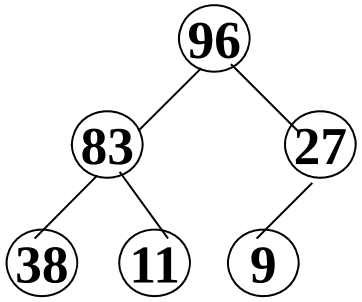
- 用数组表示堆：
  - 从1开始对堆的结点从根开始自上而下逐层、每层从左到右进行编号，然后让结点中的元素按编号在数组A中与下标对号入座。
- 用数组表示堆的优点：
  - 存储紧凑，空间利用率高
  - 父子关系简单清晰：存放在A[i]的是结点i的元素，A[2i]和A[2i+1]分别是结点i的左和右儿子结点2i和2i+1的元素，A[i/2]是结点i的父结点的元素。
- 数组实现优先队列极小化堆MinHeap

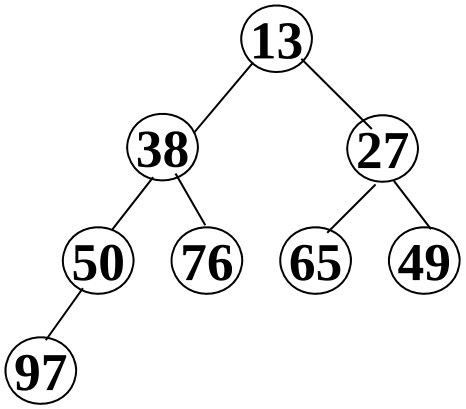
堆排序算法

- 堆的定义：n个元素的序列(k1,k2,……kn)，当且仅当满足下列关系时，称之为堆。

$$\begin{cases} k_i \le k_{2i} \\ k_i \le k_{2i+1} \end{cases} \quad 或 \quad \begin{cases} k_i \ge k_{2i} \\ k_i \ge k_{2i+1} \end{cases} \qquad (i=1,2,\ldots\ldots\lfloor n/2 \rfloor)$$

例1（96，83，27，38，11，9） 例2（13，38，27，50，76，65，49，97）

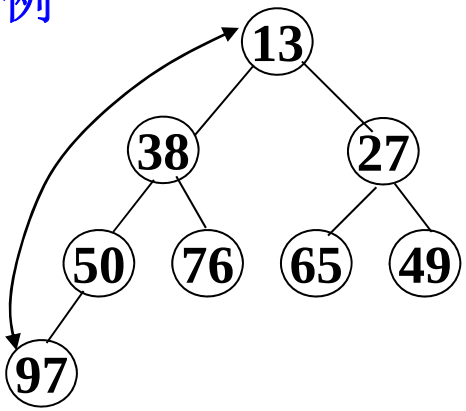


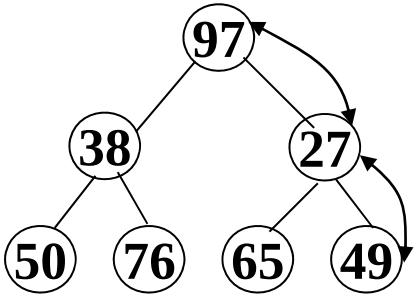


可将堆序列看成完全二叉树，则堆顶元素（完全二叉树的根）必为序列中n个元素的最小值或最大值

- 堆排序算法基本思想：将无序序列建成一个堆，得到关键字最小（或最大）的记录；输出堆顶的最小（大）值后，使剩余的**n-1**个元素重又建成一个堆，则可得到**n**个元素的次小值；重复执行，得到一个有序序列，这个过程叫堆排序。
- 堆排序需解决的两个问题：
  - 如何由一个无序序列建成一个堆？
  - 如何在输出堆顶元素之后，调整剩余元素，使之成为一个新的堆？
- 第二个问题解决方法——筛选
  - 方法：输出堆顶元素之后，以堆中最后一个元素替代之；然后将根结点值与左、右子树的根结点值进行比较，并与其中小者进行交换；重复上述操作，直至叶子结点，将得到新的堆，称这个从堆顶至叶子的调整过程为“筛选”。

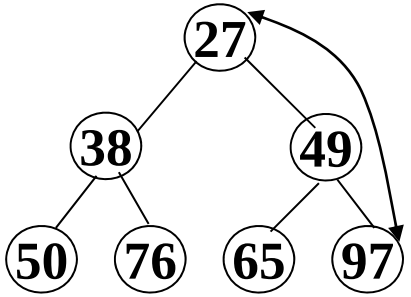
例





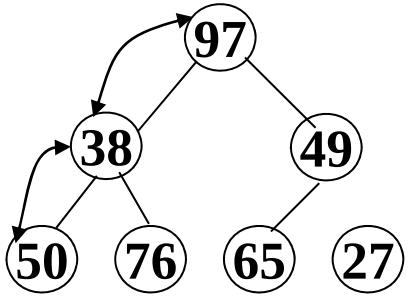
13

输出：13



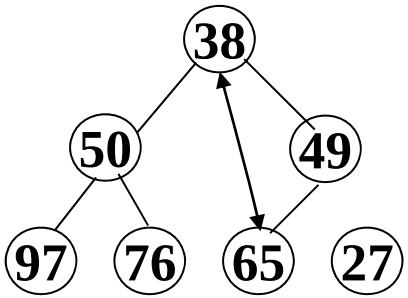
13

输出：13



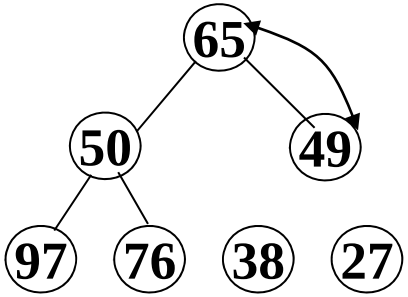
13

输出：13 27



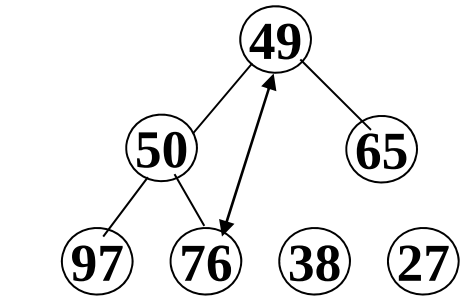
13

输出：13 27



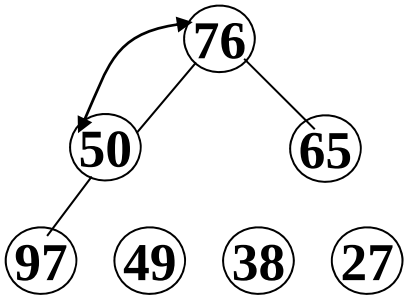
13

输出：13 27 38



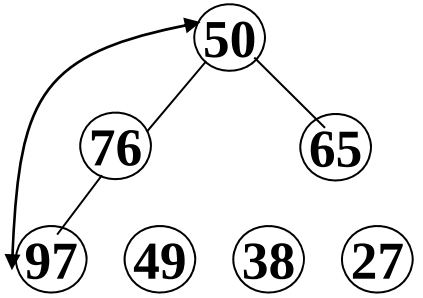
13

输出：13 27 38



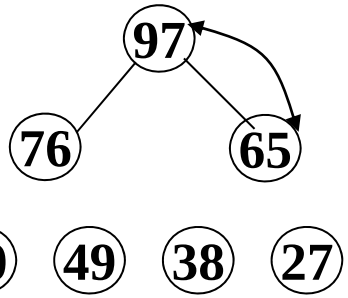
13

输出：13 27 38 49



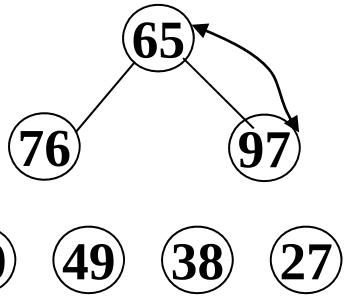
13

输出：13 27 38 49



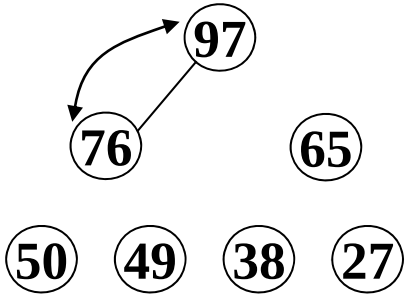
13

输出：13 27 38 49 50



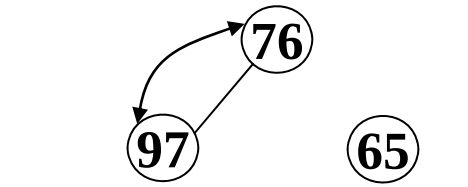
13

输出：13 27 38 49 50



13

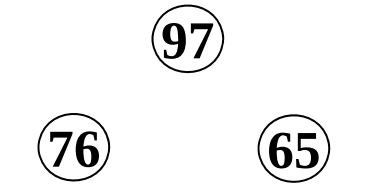
输出：13 27 38 49 50 65



50 49 38 27

13

输出：13 27 38 49 50 65



50 49 38 27

13

输出：13 27 38 49 50 65 76

97

76 65

50 49 38 27

13

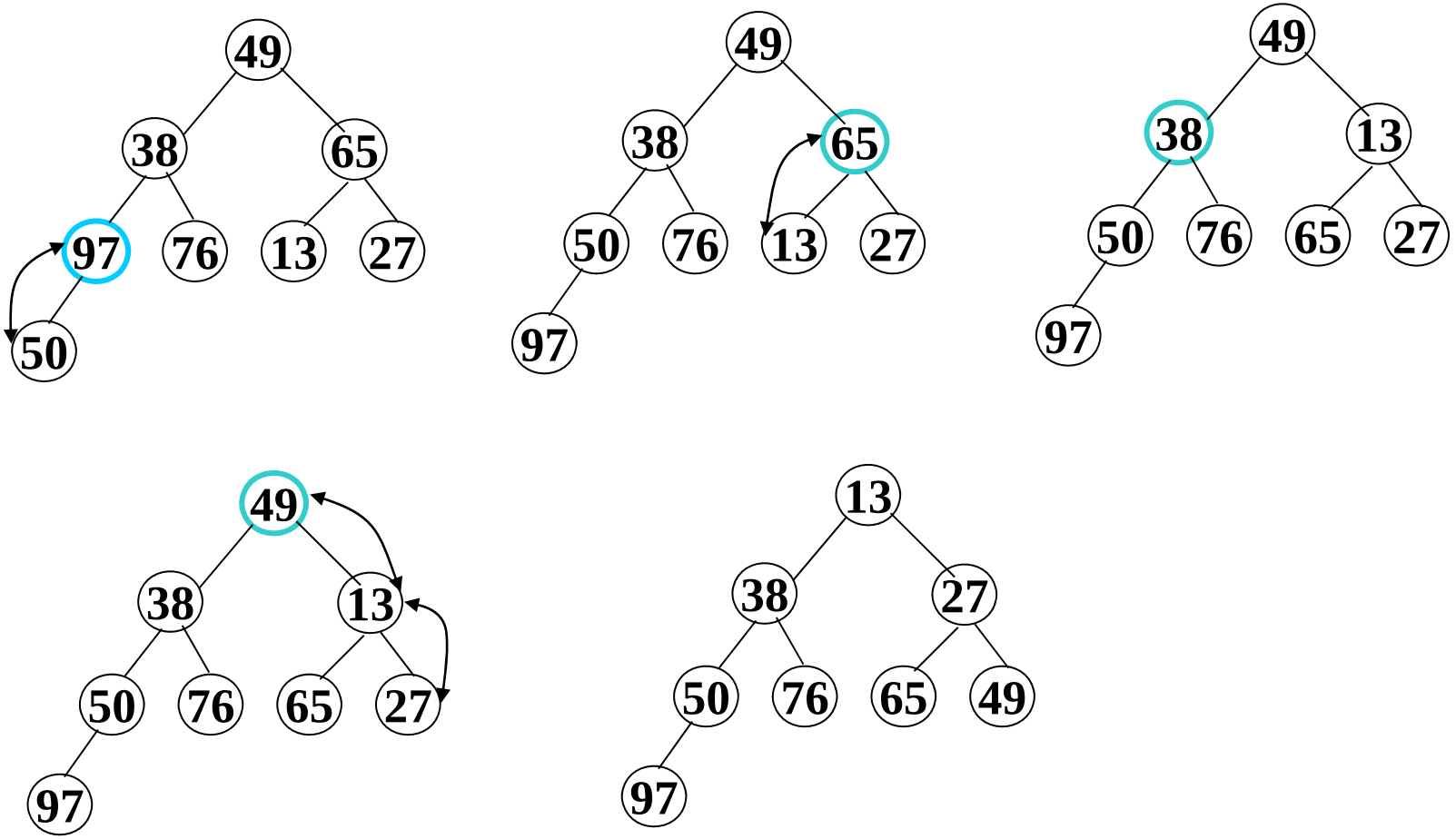
输出：13 27 38 49 50 65 76 97

- 算法描述

```
int sift(T r[],int k,int m)
{ int i,j;
  T x;
  i=k;  x=r[i]; j=2*i;
  while(j<=m)
  {  if((j<m)&&(r[j]>r[j+1]))
     j++; //这里判断r[i]的右儿子r[j+1]是否存在，以及比较左右儿子的大小
     if(x>r[j])
     {  r[i]=r[j];
        i=j;
        j*=2;
     }
     else  break;
  }
  r[i]=x;
}
```

- 第一个问题解决方法
  - 方法：从无序序列的第⌊n/2⌋个元素（即此无序序列对应的完全二叉树的最后一个非终端结点）起，至第一个元素止，进行反复筛选

例 含8个元素的无序序列（49，38，65，97，76，13，27，50）



- 算法描述

```
void heapsort(T r[],int n)
{  int i;
   T x;
   for(i=n/2;i>=1;i--)
      sift(r,i,n);
   for(i=n;i>=2;i--)
   {  x=r[1];
      r[1]=r[i];
      r[i]=x;
      sift(r,1,i-1);
   }
}
```

- 算法描述
  - 时间复杂度：最坏情况下**T(n)=O(nlogn)**
  - 空间复杂度：**S(n)=O(n)**

11.5 可并优先队列

可并优先队列也是一个以集合为基础的抽象数据类型。除了必须支持优先队列的**Insert**和**DeleteMin**运算外，可并优先队列还支持**2**个不同优先队列的合并运算**Concatenate**。

用堆来实现优先队列，可在**O(logn)**时间内支持同一优先队列中的基本运算。但合并**2**个不同优先队列的效率不高。下面讨论的左偏树结构不但能在**O(logn)**时间内支持同一优先队列中的基本运算，而且还能有效地支持**2**个不同优先队列的合并运算**Concatenate**。

11.5 可并优先队列

11.5.1 左偏树的定义

左偏树是一类特殊的优先级树。与优先级树类似，左偏树也有极小化左偏树与极大化左偏树之分。为了确定起见，下面所讨论的左偏树均为极小化左偏树。常用的左偏树有左偏高树和左偏重树2种不同类型。顾名思义，左偏高树的左子树偏高，而左偏重树的左子树偏重。下面给出其严格定义。

若将二叉树结点中的空指针看作是指向一个空结点，则称这类空结点为二叉树的前端结点。并规定所有前端结点的高度（重量）为0。

对于二叉树中任意一个结点x，递归地定义其高度s(x)为：

$s(x) = \min \{ s(L), s(R) \} + 1$

其中L和R分别是结点x的左儿子结点和右儿子结点。

11.5 可并优先队列

11.5.1 左偏树的定义

一棵优先级树是一棵左偏高树，当且仅当在该树的每个内结点处，其左儿子结点的高（s值）大于或等于其右儿子结点的高（s值）。

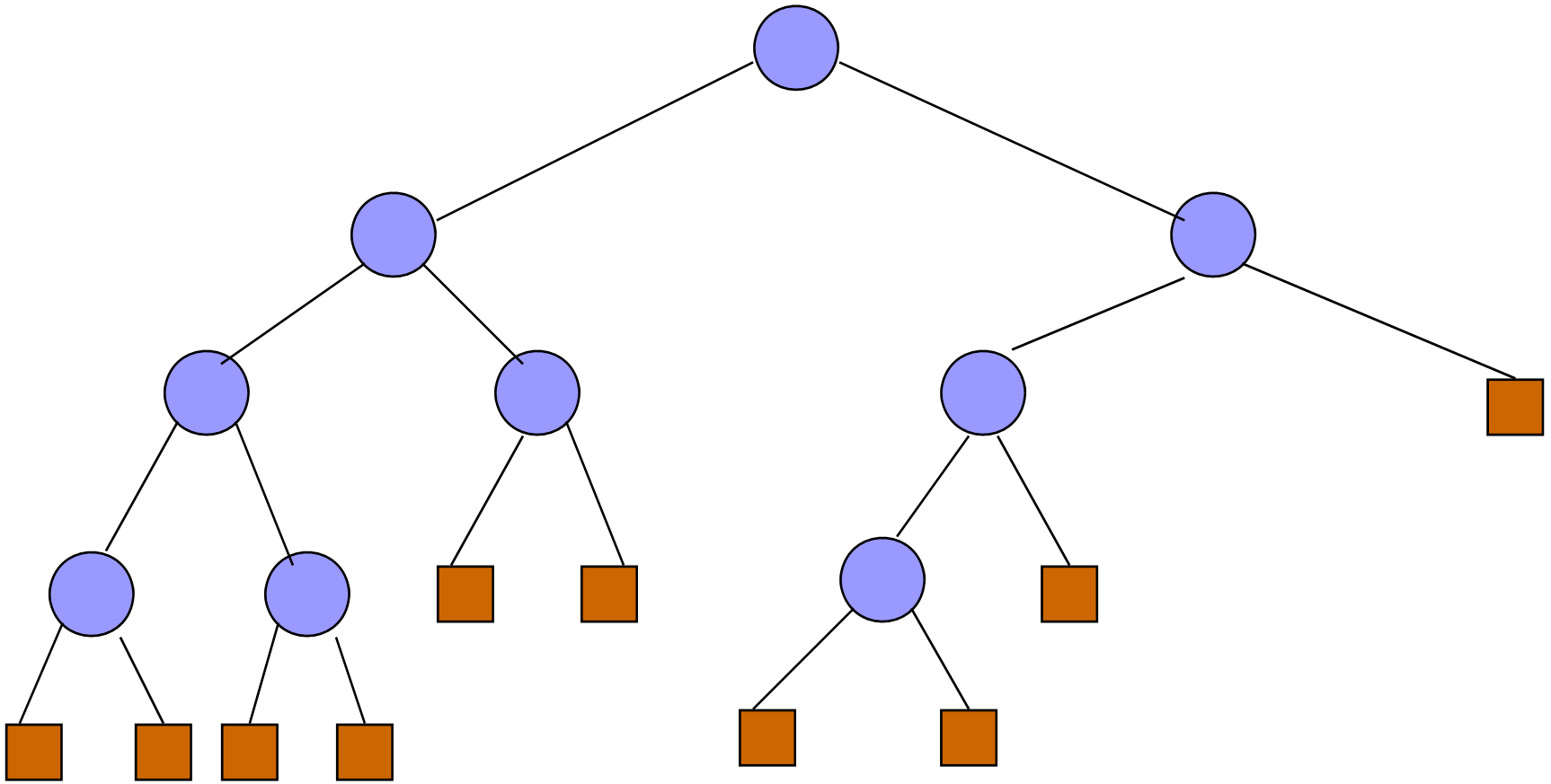
对于二叉树中任意一个结点x，其重量w(x)递归地定义为：

w(x) = w(L) + w(R) + 1

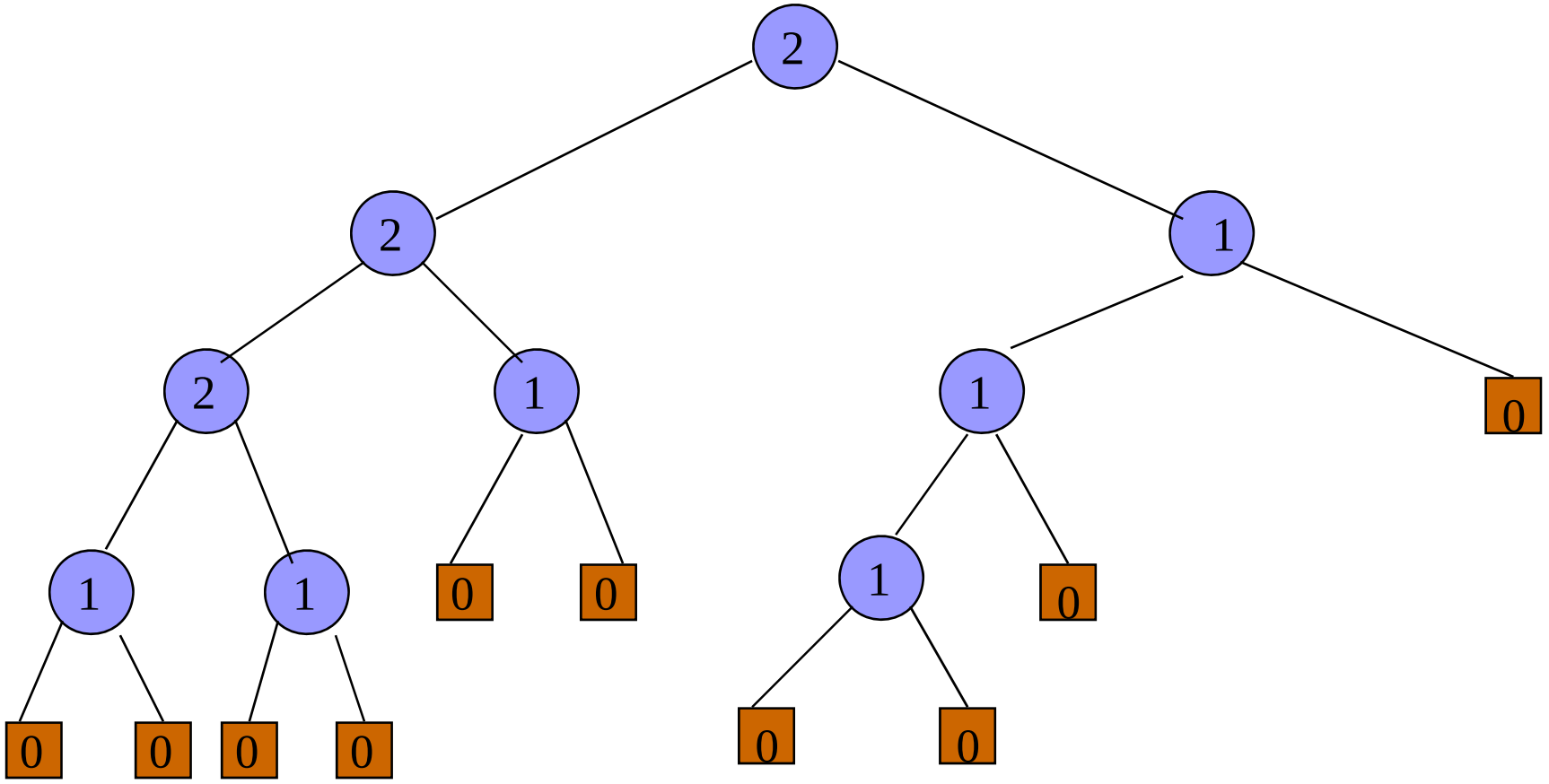
其中L和R分别是结点x的左儿子结点和右儿子结点。

一棵优先级树是一棵左偏重树，当且仅当在该树的每个内结点处，其左儿子结点的重（w值）大于或等于其右儿子结点的重（w值）。

A Leftist Tree



s() Values Example



11.5 可并优先队列

11.5.2 左偏高树的性质

左偏高树具有下面性质：
设x是一棵左偏高树的任意一个内结点，则
(1)以x为根的子树中至少有2^s(x) - 1个结点。
(2)如果以x为根的子树中有m个结点，则s(x)的值不超过log(m+1)。
(3)从x出发的最右路经的长度恰为s(x)。

11.6 优先队列的应用——Huffman编码

- 问题的提出：
  - 已知一个文本文件，要求把它转换成一个电子文档，以便存储在磁介质中或在网络上传输。从存储的角度看，我们自然希望该电子文档越短越好，即存储占用的空间越少越好；从传输的角度看，我们自然也希望该电子文档越短越好，即传输占用的时间越少越好。因此，我们要求转换成的电子文档尽量地短，尽量地压缩。
  - 有许多名家说过，把一个问题表述清楚了，就已经解决了问题的一半。这说明问题的表述很重要，表述得越清楚、越精确，对问题的解决越好。上述问题还需要更精确的表述。

11.6 优先队列的应用——Huffman编码

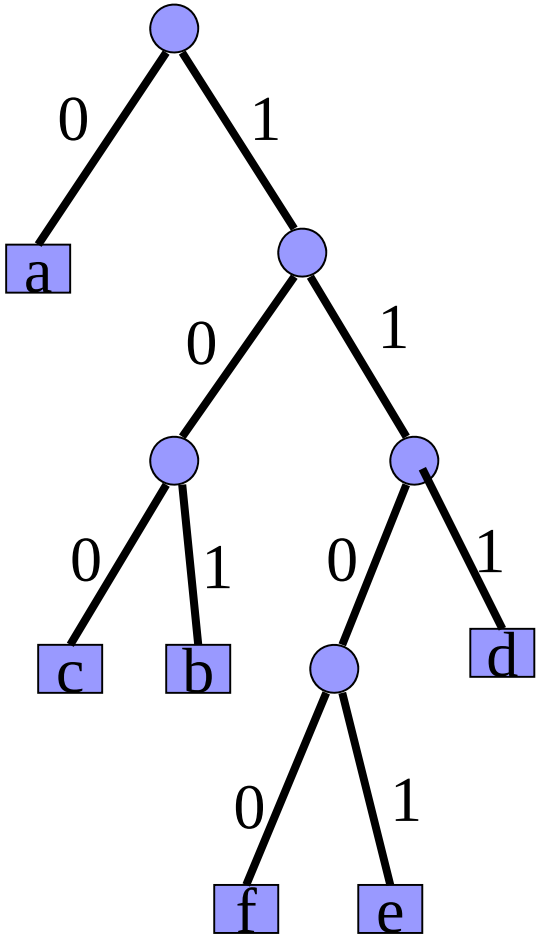
表述Huffman编码问题的准备

对涉及的有关对象、概念、术语、和记号的准备

- 一个文本文件就是一个字符串，记为**F**。
- 文本文件中所用到的不同的字符组成的集合，记为**C**。
- **C**中的字符**c**在文件中出现的频率/次数，记为**f(c)**或**fc**。
- 字符的编码——**0/1**串。如字符‘**a**’的**ASCII**码是‘**01100001**’。
- 字符编码的码长——**0/1**串的串长。记**c**∈**C**的码长为**l(c)**。
- 文件**F**编码的码长——原文本文件编码后的**0/1**串的累计长。记为$L(F)=\sum_{c\in C} f(c)* l(c)$ 。
- 字符的定长编码。 **ASCII**码是一种定长编码。不可能压缩。
- 字符的变长编码——字符的码长随字符而变。

11.6 优先队列的应用——Huffman编码

- 表述Huffman编码问题的准备（续）
  - 编码与解码——既要编码又要解码，以便复原文件。
  - 二义性：不能造成一串编码有两种理解。
  - 前缀码：为了解码时不会出现二义性，要求任意一个字符的编码不能是另一个字符的前缀。
  - 字符集C的前缀码的表示——以C中的字符为叶结点的二叉树，如{a＝0,b＝101,c＝100,d＝111,e＝1101,f＝1100} 可表示成右图的二叉树。这棵树叫字符集C的前缀编码树，记为T。



11.6 优先队列的应用——Huffman编码

- Huffman编码问题的表述——问题数学化
  - 经上述准备，我们看到，字符$c \in C$的码长$l(c)$正是c在字符集C的前缀编码树T中的深度，记为$d_T(c)$。这样，我们的问题可更具体、更精确地表述为：
  - 已知字符串F和出现在F中的字符集C及C中每一字符c出现的频率/次数$f(c)$，要求C的一棵最优的前缀编码树T，使得F的编码长$\sum_{c \in C} f(c) * d_T(c) \equiv B(T)$达到最小。这棵最优的前缀编码树叫Huffman编码树。相应的前缀编码叫Huffman编码。

11.6 优先队列的应用——Huffman编码

- 求解问题的设想与分析
  - (1) 问题的目标是求C的最优前缀编码树T，因此，我们应该先分析一下C的最优的前缀编码树T的性质：它是一棵以C中的字符为叶结点的健全二叉树(这容易用反证法加以证明)。因而一定有两个最深的叶结点是兄弟结点。
  - (2) 按照大数学家、大教育家波利亚的观点，“除数学和论证逻辑外,我们所有的知识都是由一些猜想所构成的。”为了得到有用的结论、方法，总是从猜想开始。而猜想的依据是合情推理。

    就摆在面前的问题而言，可直观地设想：最深的叶结点中有两个兄弟是C中频度最低的字符x和y，因为编码最长的字符应该是频度最低的字符(这需要证明)。

11.6 优先队列的应用——Huffman编码

- 求解问题的设想与分析（续）
  - (3) 用字符z代替字符x和y，让f(x)+f(y)作为字符z的频度f(z)，相应地，在T中删去兄弟叶结点x和y，而且用z取代已经成为叶结点的x和y的父结点，得到一棵新的健全二叉编码树T'，那么，T'应该是C-{x,y}∪{z}的最优前缀编码树(这需要证明)。
    - 若上述设想与分析正确，那么，问题就有了解法：把C中的字符以其频度为优先级值，且以小者优先组织成优先队列Q。然后反复地执行下面两个语句：

      ① 删除Q中优先级最高的两个字符，设为x和y；

      ② 虚拟字符z(分别以x和y为左和右儿子)，并赋予优先级值f(z)=f(x)+f(y)，插入Q；

      直到Q为空，其结果就是C的最优前缀编码树。

11.6 优先队列的应用——Huffman编码

Huffman编码问题的解决

- 从理论上看待Huffman编码问题，在求解问题的设想与分析中的(2)和(3)所猜想的分别是问题的解的贪心选择性质和最优子结构性质。这两个性质保证了问题可以用基于优先队列的前述算法正确地加以求解。所以问题的真正解决有待对贪心选择性质和最优子结构性质作理论上的证明。此外，还得给出Huffman编码和解码的简洁实现。

11.6 优先队列的应用——Huffman编码

Huffman编码问题的解决（续）

- (1) 问题解的贪心选择性质：设x和y是C中具有最小频度的两个字符，则存在C的一个最优前缀码使x和y具有最长的相同码长且仅最后一位编码不同，即x和y是编码树中最深的一对叶结点兄弟。

  证明：

11.6 优先队列的应用——Huffman编码

Huffman编码问题的解决（续）

- **(2)** 问题解的最优子结构性质：设**x**和**y**是**C**的最优编码树**T**中的两个叶子且为兄弟，**z**是它们的父亲。若将**z**看作是具有频率**f(z)=f(x)+f(y)**的字符，则树 **T′=T-**｛**x,y**｝表示字符集**C′=C-**｛**x,y**｝∪｛**z**｝的最优编码树。

证明：

字符：	a	e	i	s	t	sp	nl
权值：	10	15	12	3	4	13	1

压缩与哈夫曼树

11.6 优先队列的应用——Huffman编码

Huffman编码问题的解决（续）

(3) Huffman编码和解码的简洁实现

从算法的动画演示中已经看到了Huffman编码和解码的一种实现方式。它们的简洁表述作为练习。