中位数和顺序统计量

在一个由 n 个元素组成的集合中，第 i 个**顺序统计量**(order statistic)是该集合中第 i 小的元素。例如，在一个元素集合中，**最小值**是第 1 个顺序统计量($i=1$)，**最大值**是第 n 个顺序统计量($i=n$)。用非形式化的描述来说，一个**中位数**(median)是它所属集合的“中点元素”。当 n 为奇数时，中位数是唯一的，位于 $i=(n+1)/2$ 处。当 n 为偶数时，存在两个中位数，分别位于 $i=n/2$ 和 $i=n/2+1$ 处。因此，如果不考虑 n 的奇偶性，中位数总是出现在 $i=\lfloor(n+1)/2\rfloor$处(**下中位数**)和 $i=\lceil(n+2)/2\rceil$处(**上中位数**)。为了简便起见，本书中所用的“中位数”都是指下中位数。

本章将讨论从一个由 n 个互异的元素构成的集合中选择第 i 个顺序统计量的问题。为了方便起见，假设集合中的元素都是互异的，但实际上我们所做的都可以推广到集合中包含重复元素的情形。我们将这一问题形式化定义为如下的**选择问题**：

输入：一个包含 n 个(互异的)数的集合 A 和一个整数 i，$1\leqslant i\leqslant n$。

输出：元素 $x\in A$，且 A 中恰好有 $i-1$ 个其他元素小于它。

我们可以在 $O(n\lg n)$时间内解决这个选择问题，因为我们可以用堆排序或归并排序对输入数据进行排序，然后在输出数组中根据下标找出第 i 个元素即可。本章将介绍一些更快的算法。

在 9.1 节中，我们将讨论从一个集合中选择最小元素和最大元素的问题。对于一般化选择问题的更有意思的讨论将在接下来的两节中进行。9.2 节将分析一个实用的随机算法，它在元素互异的假设条件下可以达到 $O(n)$的期望运行时间。9.3 节将给出一个更具有理论意义的算法，它在最坏情况下的运行时间为 $O(n)$。

9.1 最小值和最大值

在一个有 n 个元素的集合中，需要做多少次比较才能确定其最小元素呢？我们可以很容易地给出 $n-1$ 次比较这个上界：依次遍历集合中的每个元素，并记录下当前最小元素。在下面的程序中，我们假设该集合元素存放在数组 A 中，且 $A.length=n$：

```
MINIMUM(A)
1  min = A[1]
2  for i = 2 to A.length
3      if min > A[i]
4          min = A[i]
5  return min
```

当然，最大值也可以通过 $n-1$ 次比较找出来。

这是我们能得到的最好结果吗？是的，对于确定最小值问题，我们可以得到其下界就是 $n-1$ 次比较。对于任意一个确定最小值的算法，可以把它看成是在各元素之间进行的一场锦标赛。每次比较都是锦标赛中的一场比赛，两个元素中较小的获胜。需要注意的是，除了最终获胜者以外，每个元素都至少要输掉一场比赛。因此，我们得到结论：为了确定最小值，必须要做 $n-1$ 次比较。因此，从所执行的比较次数来看，算法 MINIMUM 是最优的。

同时找到最小值和最大值

在某些应用中，我们必须要找出一个包含 n 个元素的集合中的最小值和最大值。例如，一个图形程序可能需要转换一组(x，y)数据，使之能适合一个矩形显示器或其他图形输出装置。为了做到这一点，程序必须首先确定每个坐标中的最小值和最大值。

就这一点来说，用渐近最优的 $\Theta(n)$ 次比较，在 n 个元素中同时找到最小值和最大值的方法是显然的：只要分别独立地找出最小值和最大值，这各需要 $n-1$ 次比较，共需 $2n-2$ 次比较。

事实上，我们只需要最多 $3\lfloor n/2 \rfloor$ 次比较就可以同时找到最小值和最大值。具体的方法是记录已知的最小值和最大值。但我们并不是将每一个输入元素与当前的最小值和最大值进行比
214 较——这样做的代价是每个元素需要 2 次比较，而是对输入元素成对地进行处理。首先，我们将一对输入元素相互进行比较，然后把较小的与当前最小值比较，把较大的与当前最大值进行比较。这样，对每两个元素共需 3 次比较。

如何设定已知的最小值和最大值的初始值依赖于 n 是奇数还是偶数。如果 n 是奇数，我们就将最小值和最大值的初值都设为第一个元素的值，然后成对地处理余下的元素。如果 n 是偶数，就对前两个元素做一次比较，以决定最小值和最大值的初值，然后与 n 是奇数的情形一样，成对地处理余下的元素。

下面来分析一下总的比较次数。如果 n 是奇数，那么总共进行 $3\lfloor n/2 \rfloor$ 次比较。如果 n 是偶数，则是先进行一次初始比较，然后进行 $3(n-2)/2$ 次比较，共 $3n/2-2$ 次比较。因此，不管是哪一种情况，总的比较次数至多是 $3\lfloor n/2 \rfloor$。

练习

9.1-1 证明：在最坏情况下，找到 n 个元素中第二小的元素需要 $n+\lceil \lg n \rceil - 2$ 次比较。（提示：可以同时找最小元素。）

***9.1-2** 证明：在最坏情况下，同时找到 n 个元素中最大值和最小值的比较次数的下界是 $\lceil 3n/2 \rceil - 2$。（提示：考虑有多少个数有成为最大值或最小值的潜在可能，然后分析一下每一次比较会如何影响这些计数。）

9.2 期望为线性时间的选择算法

一般选择问题看起来要比找最小值这样的简单问题更难。但令人惊奇的是，这两个问题的渐近运行时间却是相同的：$\Theta(n)$。本节将介绍一种解决选择问题的分治算法。RANDOMIZED-SELECT 算法是以第 7 章的快速排序算法为模型的。与快速排序一样，我们仍然将输入数组进行递归划分。但与快速排序不同的是，快速排序会递归处理划分的两边，而 RANDOMIZED-SELECT 只处理划分的一边。这一差异会在性能分析中体现出来：快速排序的期望运行时间是
215 $\Theta(n \lg n)$，而 RANDOMIZED-SELECT 的期望运行时间为 $\Theta(n)$。这里，假设输入数据都是互异的。

RANDOMIZED-SELECT 利用了 7.3 节介绍的 RANDOMIZED-PARTITION 过程。与 RANDOMIZED-QUICKSORT 一样，因为它的部分行为是由随机数生成器的输出决定的，所以 RANDOMIZED-SELECT 也是一个随机算法。以下是 RANDOMIZED-SELECT 的伪代码，它返回数组 $A[p..r]$ 中第 i 小的元素。

```
RANDOMIZED-SELECT (A, p, r, i)
1  if p == r
2      return A[p]
3  q = RANDOMIZED-PARTITION(A, p, r)
4  k = q-p+1
5  if i == k          // the pivot value is the answer
6      return A[q]
7  else if i < k
8      return RANDOMIZED-SELECT(A, p, q-1, i)
9  else return RANDOMIZED-SELECT(A, q+1, r, i-k)
```

RANDOMIZED-SELECT 的运行过程如下：第 1 行检查递归的基本情况，即 $A[p..r]$中只包括一个元素。在这种情况下，i 必然等于 1，在第 2 行，我们只需将 $A[p]$返回作为第 i 小的元素即可。其他情况，就会调用第 3 行的 RANDOMIZED-PARTITION，将数组 $A[p..r]$划分为两个(可能为空的)子数组 $A[p..q-1]$和 $A[q+1..r]$，使得 $A[p..q-1]$中的每个元素都小于或等于 $A[q]$，而 $A[q]$小于 $A[q+1..r]$中的每个元素。与快速排序中一样，我们称 $A[q]$为**主元**(pivot)。RANDOMIZED-SELECT 的第 4 行计算子数组 $A[p..q]$内的元素个数 k，即处于划分的低区的元素的个数加 1，这个 1 指主元。然后，第 5 行检查 $A[q]$是否是第 i 小的元素。如果是，第 6 行就返回 $A[q]$。否则，算法要确定第 i 小的元素落在两个子数组 $A[p..q-1]$和 $A[q+1..r]$的哪一个之中。如果 $i<k$，则要找的元素落在划分的低区。第 8 行就在低区的子数组中进一步递归查找。如果 $i>k$，则要找的元素落在划分的高区中。因为我们已经知道了有 k 个值小于 $A[p..r]$中第 i 小的元素，即 $A[p..q]$内的元素，所以，我们所要找的元素必然是 $A[q+1..r]$中的第 $i-k$ 小的元素。它在第 9 行中被递归地查找。上述程序看起来允许递归调用含有 0 个元素的子数组，但练习 9.2-1 要求证明这种情况不可能发生。

RANDOMIZED-SELECT 的最坏情况运行时间为 $\Theta(n^2)$，即使是找最小元素也是如此，因为在每次划分时可能极不走运地总是按余下的元素中最大的来进行划分，而划分操作需要$\Theta(n)$时间。我们也将看到该算法有线性的期望运行时间，又因为它是随机化的，所以不存在一个特定的 会导致其最坏情况发生的输入数据。 216

为了分析 RANDOMIZED-SELECT 的期望运行时间，我们设该算法在一个含有 n 个元素的输入数组 $A[p..r]$上的运行时间是一个随机变量，记为 $T(n)$。下面我们可以得到 $\mathrm{E}[T(n)]$的一个上界：程序 RANDOMIZED-PARTITION 能等概率地返回任何元素作为主元。因此，对每一个 $k(1\leqslant k\leqslant n)$，子数组 $A[p..q]$有 k 个元素(全部小于或等于主元)的概率是 $1/n$。对所有 $k=1$，2，…，n，定义指示器随机变量 X_k 为：

$$X_k=\mathrm{I}\{\text{子数组 } A[p..q]\text{正好包含 } k \text{ 个元素}\}$$

然后，假设元素是互异的，我们有：

$$\mathrm{E}[X_k]=1/n \tag{9.1}$$

当调用 RANDOMIZED-SELECT 并选择 $A[q]$作为主元时，事先并不知道是否会立即得到正确答案而结束，或者在子数组 $A[p..q-1]$上递归，或者在子数组 $A[q+1..r]$上递归。这个决定依赖于第 i 小的元素相对于 $A[q]$落在哪个位置。假设 $T(n)$是单调递增的，通过评估最大可能的输入数据递归调用所需时间，我们可以给出递归调用所需时间的上界。也就是说，为了得到上界，我们假定第 i 个元素总是在划分中包含较大元素的一边。对一个给定的 RANDOMIZED-SELECT，指示器随机变量 X_k 恰好在给定的 k 值上取值 1，对其他值都为 0。当 $X_k=1$ 时，我们可能要递归处理的两个子数组的大小分别为 $k-1$ 和 $n-k$。因此可以得到递归式：

$$\begin{aligned}T(n)&\leqslant\sum_{k=1}^{n}X_k\cdot(T(\max(k-1,n-k))+O(n))\\&=\sum_{k=1}^{n}X_k\cdot T(\max(k-1,n-k))+O(n)\end{aligned}$$ 217

两边取期望值，得到

$$\begin{aligned}\mathrm{E}[T(n)]&\leqslant\mathrm{E}\left[\sum_{k=1}^{n}X_k\cdot T(\max(k-1,n-k))+O(n)\right]\\&=\sum_{k=1}^{n}\mathrm{E}[X_k\cdot T(\max(k-1,n-k))]+O(n)\quad\text{(期望的线性性质)}\\&=\sum_{k=1}^{n}\mathrm{E}[X_k]\cdot\mathrm{E}[T(\max(k-1,n-k))]+O(n)\quad\text{(利用公式(C.24))}\end{aligned}$$

$$= \sum_{k=1}^{n} \frac{1}{n} \cdot \mathrm{E}[T(\max(k-1,n-k))] + O(n) \quad \text{（利用公式(9.1)）}$$

公式(C.24)的应用依赖于 X_k 和 $T(\max(k-1, n-k))$是独立的随机变量。练习 9.2-2 要求证明这个命题。

下面来考虑一下表达式 $\max(k-1, n-k)$。我们有

$$\max(k-1,n-k) = \begin{cases} k-1 & \text{若 } k > \lceil n/2 \rceil \\ n-k & \text{若 } k \leqslant \lceil n/2 \rceil \end{cases}$$

如果 n 是偶数，则从 $T(\lceil n/2 \rceil)$到 $T(n-1)$的每一项在总和中恰好出现两次。如果 n 是奇数，除了 $T(\lfloor n/2 \rfloor)$出现一次以外，其他这些项也都会出现两次。因此，我们有

$$\mathrm{E}[T(n)] \leqslant \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} \mathrm{E}[T(k)] + O(n)$$

我们将用替代法来得到 $\mathrm{E}[T(n)]=O(n)$。假设对满足这个递归式初始条件的某个常数 c，有 $\mathrm{E}[T(n)] \leqslant cn$。假设对小于某个常数的 n，有 $T(n)=O(1)$(稍后将用到这个常数)。同时，还要选择一个常数 a，使得对所有的 $n>0$，上式中 $O(n)$项所描述的函数(用来表示算法运行时间中的非递归部分)有上界 an。利用这个归纳假设，可以得到：

$$\begin{aligned}
\mathrm{E}[T(n)] &\leqslant \frac{2}{n} \sum_{k=\lfloor n/2 \rfloor}^{n-1} ck + an \\
&= \frac{2c}{n}\left(\sum_{k=1}^{n-1} k - \sum_{k=1}^{\lfloor n/2 \rfloor - 1} k\right) + an \qquad 218 \\
&= \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(\lfloor n/2 \rfloor - 1)\lfloor n/2 \rfloor}{2}\right) + an \\
&\leqslant \frac{2c}{n}\left(\frac{(n-1)n}{2} - \frac{(n/2-2)(n/2-1)}{2}\right) + an \\
&= \frac{2c}{n}\left(\frac{n^2-n}{2} - \frac{n^2/4-3n/2+2)}{2}\right) + an \\
&= \frac{c}{n}\left(\frac{3n^2}{4} + \frac{n}{2} - 2\right) + an \\
&= c\left(\frac{3n}{4} + \frac{1}{2} - \frac{2}{n}\right) + an \\
&\leqslant \frac{3cn}{4} + \frac{c}{2} + an \\
&= cn - \left(\frac{cn}{4} - \frac{c}{2} - an\right)
\end{aligned}$$

为了完成证明，还需要证明：对足够大的 n，最后一个表达式至多是 cn，等价地，$cn/4-c/2-an \geqslant 0$。如果在上式两边加上 $c/2$，并且提取因子 n，就可以得到 $n(c/4-a) \geqslant c/2$。只要我们选择的常数 c 能够满足 $c/4-a>0$，即 $c>4a$，就可以将两边同除以 $c/4-a$，得到

$$n \geqslant \frac{c/2}{c/4-a} = \frac{2c}{c-4a}$$

因此，如果假设对所有 $n<2c/(c-4a)$，都有 $T(n)=O(1)$，那么就有 $\mathrm{E}[T(n)]=O(n)$。我们可以得出这样的结论：假设所有元素是互异的，在期望线性时间内，我们可以找到任一顺序统计量，特别是中位数。

练习

9.2-1 证明：在 RANDOMIZED-SELECT 中，对长度为 0 的数组，不会进行递归调用。

9.2-2　请讨论：指示器随机变量 X_k 和 $T(\max(k-1, n-k))$是独立的。

9.2-3　给出 RANDOMIZED-SELECT 的一个基于循环的版本。

9.2-4　假设用 RANDOMIZED-SELECT 去选择数组 $A=\langle 3, 2, 9, 0, 7, 5, 4, 8, 6, 1\rangle$的最小元素，给出能够导致 RANDOMIZED-SELECT 最坏情况发生的一个划分序列。

9.3　最坏情况为线性时间的选择算法

我们现在来看一个最坏情况运行时间为 $O(n)$的选择算法。像 RANDOMIZED-SELECT 一样，SELECT 算法通过对输入数组的递归划分来找出所需元素，但是，在该算法中能够保证得到对数组的一个好的划分。SELECT 使用的也是来自快速排序的确定性划分算法 PARTITION（见 7.1 节），但做了修改，把划分的主元也作为输入参数。

通过执行下列步骤，算法 SELECT 可以确定一个有 $n>1$ 个不同元素的输入数组中第 i 小的元素。（如果 $n=1$，则 SELECT 只返回它的唯一输入数值作为第 i 小的元素。）

1. 将输入数组的 n 个元素划分为$\lfloor n/5\rfloor$组，每组 5 个元素，且至多只有一组由剩下的 $n \bmod 5$ 个元素组成。

2. 寻找这$\lceil n/5\rceil$组中每一组的中位数：首先对每组元素进行插入排序，然后确定每组有序元素的中位数。

3. 对第 2 步中找出的$\lceil n/5\rceil$个中位数，递归调用 SELECT 以找出其中位数 x（如果有偶数个中位数，为了方便，约定 x 是较小的中位数）。

4. 利用修改过的 PARTITION 版本，按中位数的中位数 x 对输入数组进行划分。让 k 比划分的低区中的元素数目多 1，因此 x 是第 k 小的元素，并且有 $n-k$ 个元素在划分的高区。

5. 如果 $i=k$，则返回 x。如果 $i<k$，则在低区递归调用 SELECT 来找出第 i 小的元素。如果 $i>k$，则在高区递归查找第 $i-k$ 小的元素。

为分析 SELECT 的运行时间，我们先要确定大于划分主元 x 的元素个数的下界。图 9-1 给出了一些形象的说明。在第 2 步找出的中位数中，至少有一半大于或等于中位数的中位数 x[⊖]。因此，在这$\lceil n/5\rceil$个组中，除了当 n 不能被 5 整除时产生的所含元素少于 5 的那个组和包含 x 的那个组之外，至少有一半的组中有 3 个元素大于 x。不算这两个组，大于 x 的元素个数至少为：

$$3\left(\left\lceil \frac{1}{2}\left\lceil \frac{n}{5}\right\rceil\right\rceil-2\right)\geqslant\frac{3n}{10}-6$$

类似地，至少有 $3n/10-6$ 个元素小于 x。因此，在最坏情况下，在第 5 步中，SELECT 的递归调用最多作用于 $7n/10+6$ 个元素。

现在，我们可以设计一个递归式来推导 SELECT 算法的最坏情况运行时间 $T(n)$了。步骤 1、2 和 4 需要 $O(n)$时间。（步骤 2 是对大小为 $O(1)$的集合调用 $O(n)$次插入排序。）步骤 3 所需时间为 $T(\lceil n/5\rceil)$，步骤 5 所需时间至多为 $T(7n/10+6)$。

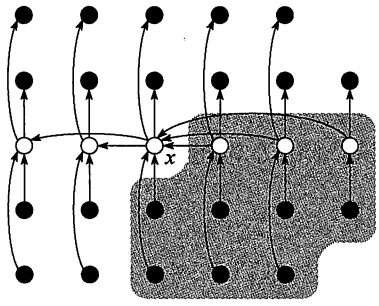

图 9-1　对算法 SELECT 的分析。所有 n 个元素都由小圈来表示，并且每一组的 5 个元素在同一列上。其中，每组的中位数用白色圈表示，而中位数的中位数 x 也被标识出来（当查找偶数个元素的中位数时，使用较小的中位数）。箭头从较大的元素指向较小的元素，从图中可以看出，在 x 的右边，每一个包含 5 个元素的组中有 3 个元素大于 x。在 x 的左边，每一个包含 5 个元素的组中有 3 个元素小于 x。大于 x 的元素的背景以阴影来显示

⊖　因为我们假设这些数是互异的，所以除了 x 以外的所有元素都大于或小于 x。

这里，我们假设 T 是单调递增的，此外，我们还要作如下假设(这一假设初看起来似乎没有什么动机)，即任何少于 140 个元素的输入需要 $O(1)$时间。后面，我们很快就会说明这个魔数 140 的
221 起源。根据上述假设，可以得到如下递归式：

$$T(n) \leqslant \begin{cases} O(1) & \text{若 } n < 140 \\ T(\lceil n/5 \rceil) + T(7n/10+6) + O(n) & \text{若 } n \geqslant 140 \end{cases}$$

我们用替换法来证明这个运行时间是线性的。更明确地说，我们将证明对某个适当大的常数 c 和所有的 $n>0$，有 $T(n)\leqslant cn$。首先，假设对某个适当大的常数 c 和所有的 $n<140$，有 $T(n)\leqslant cn$；如果 c 足够大，这个假设显然成立。同时，还要挑选一个常数 a，使得对所有的 $n>0$，上述公式中的 $O(n)$项所对应的函数(用来描述算法运行时间中的非递归部分)有上界 an。将这个归纳假设代入上述递归式的右边，得到：

$$\begin{aligned} T(n) &\leqslant c\lceil n/5 \rceil + c(7n/10+6) + an \\ &\leqslant cn/5 + c + 7cn/10 + 6c + an \\ &= 9cn/10 + 7c + an \\ &= cn + (-cn/10 + 7c + an) \end{aligned}$$

如果下式成立，上式最多是 cn：

$$-cn/10 + 7c + an \leqslant 0 \tag{9.2}$$

当 $n>70$ 时，不等式(9.2)等价于不等式 $c\geqslant 10a(n/(n-70))$。因为假设 $n>140$，所以有 $n/(n-70)\leqslant 2$。因此，选择 $c\geqslant 20a$ 就能够满足不等式(9.2)。(注意，这里常数 140 并没有什么特别之处，我们可以用任何严格大于 70 的整数来替换它，然后再相应地选择 c 即可。)因此，最坏情况下 SELECT 的运行时间是线性的。

与比较排序一样(见 8.1 节)，SELECT 和 RANDOMIZED-SELECT 也是通过元素间的比较来确定它们之间的相对次序的。在第 8 章中，我们知道在比较模型中，即使是在平均情况下，排序仍然需要 $\Omega(n\lg n)$时间(见思考题 8-1)。第 8 章的线性时间排序算法在输入上作了一些假设。相反，本章中的线性时间选择算法不需要任何关于输入的假设。它们不受限于 $\Omega(n\lg n)$的下界约束，因为它们没有使用排序就解决了选择问题。因此，在本章引言部分介绍的排序和索引方法不
222 是解决选择问题的渐近高效率方法。

练习

9.3-1 在算法 SELECT 中，输入元素被分为每组 5 个元素。如果它们被分为每组 7 个元素，该算法仍然会是线性时间吗？证明：如果分成每组 3 个元素，SELECT 的运行时间不是线性的。

9.3-2 分析 SELECT，并证明：如果 $n\geqslant 140$，则至少$\lceil n/4 \rceil$个元素大于中位数的中位数 x，至少$\lceil n/4 \rceil$个元素小于 x？

9.3-3 假设所有元素都是互异的，说明在最坏情况下，如何才能使快速排序的运行时间为$O(n\lg n)$。

***9.3-4** 对一个包含 n 个元素的集合，假设一个算法只使用比较来确定第 i 小的元素，证明：无需额外的比较操作，它也能找到第 $i-1$ 小的元素和第 $n-i$ 大的元素。

9.3-5 假设你已经有了一个最坏情况下是线性时间的用于求解中位数的“黑箱”子程序。设计一个能在线性时间内解决任意顺序统计量的选择问题算法。

9.3-6 对一个包含 n 个元素的集合来说，k **分位数**是指能把有序集合分成 k 个等大小集合的第 $k-1$个顺序统计量。给出一个能找出某一集合的 k 分位数的$O(n\lg k)$时间的算法。

9.3-7 设计一个 $O(n)$时间的算法，对于一个给定的包含 n 个互异元素的集合 S 和一个正整数 $k\leqslant n$，该算法能够确定 S 中最接近中位数的 k 个元素。

9.3-8 设 $X[1..n]$和 $Y[1..n]$为两个数组，每个都包含 n 个有序的元素。请设计一个 $O(\lg n)$时间的算法来找出数组 X 和 Y 中所有 $2n$ 个元素的中位数。

9.3-9 Olay 教授是一家石油公司的顾问。这家公司正在计划建造一条从东向西的大型输油管道，这一管道将穿越一个有 n 口油井的油田。公司希望有一条管道支线沿着最短路径从每口油井连接到主管道(方向或南或北)，如图 9-2 所示。给定每口油井的 x 和 y 坐标，教授应该如何选择主管道的最优位置，使得各直线的总长度最小？证明：该最优位置可以在线性时间内确定。 223

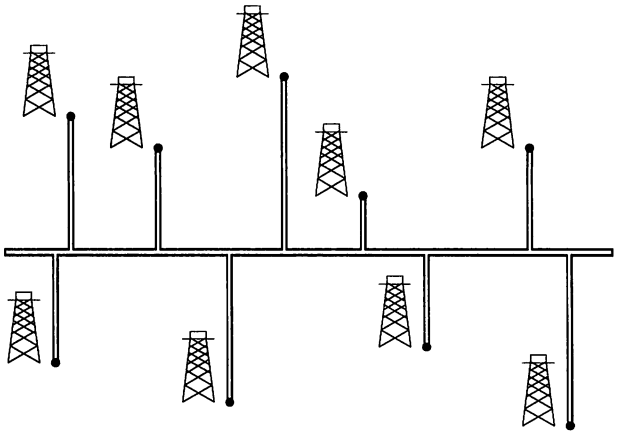

图 9-2　Olay 教授需要确定东西向石油管道的位置，使得南北向的支线管道的总长度最小

思考题

9-1 (*有序序列中的 i 个最大数*)　给定一个包含 n 个元素的集合，我们希望利用基于比较的算法找出按顺序排列的前 i 个最大元素。请设计能实现下列每一项要求，并且具有最佳渐近最坏情况运行时间的算法，以 n 和 i 来表示算法的运行时间：

a. 对输入数据排序，并找出前 i 个最大数。

b. 对输入数据建立一个最大优先队列，并调用 EXTRACT-MAX 过程 i 次。

c. 利用一个顺序统计量算法来找到第 i 大的元素，然后用它作为主元划分输入数组，再对前 i 大的数排序。 224

9-2 (*带权中位数*)　对分别具有正权重 w_1，w_2，…，w_n，且满足 $\sum_{i=1}^{n} w_i = 1$ 的 n 个互异元素 x_1，x_2，…，x_n 来说，带权中位数 x_k(较小中位数)是满足如下条件的元素：

$$\sum_{x_i<x_k} w_i < \frac{1}{2}$$

和

$$\sum_{x_i>x_k} w_i \leqslant \frac{1}{2}$$

例如，如果元素是 0.1，0.35，0.05，0.1，0.15，0.05，0.2，并且每个元素的权重等于本身(即对所有 $i=1$，2，…，7，都有 $w_i=x_i$)，那么中位数是 0.1，而带权中位数是 0.2。

a. 证明：如果对所有 $i=1$，2，…，n 都有 $w_i=1/n$，那么 x_1，x_2，…，x_n 的中位数就是 x_i 的带权中位数。

b. 利用排序，设计一个最坏情况下 $O(n\lg n)$时间的算法，可以得到 n 个元素的带权中位数。

c. 说明如何利用像 9.3 节的 SELECT 这样的线性时间中位数算法，在 $\Theta(n)$最坏情况时间内求出带权中位数。

邮局位置问题的定义如下：给定权重分别为 w_1，w_2，…，w_n 的 n 个点 p_1，p_2，…，p_n，我们希望找到一个点 p(不一定是输入点中的一个)，使得 $\sum_{i=1}^{n} w_i d(p, p_i)$ 最小，这里 $d(a, b)$ 表示点 a 与 b 之间的距离。

d. 证明：对一维邮局位置问题，带权中位数是最好的解决方法，其中，每个点都是一个实数，点 a 与 b 之间的距离是 $d(a, b)=|a-b|$。

e. 请给出二维邮局位置问题的最好解决方法：其中的点是(x, y)的二维坐标形式，点$a=(x_1, y_1)$与$b=(x_2, y_2)$之间的距离是 **Manhattan 距离**，即 $d(a, b)=|x_1-x_2|+|y_1-y_2|$。

9-3 (小顺序统计量) 要在 n 个数中选出第 i 个顺序统计量，SELECT 在最坏情况下需要的比较次数 $T(n)$满足 $T(n)=\Theta(n)$。但是，隐含在 Θ 记号中的常数项是非常大的。当 i 相对 n 来说很小时，我们可以实现一个不同的算法，它以 SELECT 作为子程序，但在最坏情况下
225 所做的比较次数更少。

a. 设计一个能用 $U_i(n)$次比较在 n 个元素中找出第 i 小元素的算法，其中，

$$U_i(n)=\begin{cases} T(n) & \text{若 } i \geqslant n/2 \\ \lfloor n/2 \rfloor + U_i(\lceil n/2 \rceil) + T(2i) & \text{其他} \end{cases}$$

(提示：从$\lfloor n/2 \rfloor$个不相交对的两两比较开始，然后对由每对中的较小元素构成的集合进行递归。)

b. 证明：如果 $i<n/2$，则有 $U_i(n)=n+O(T(2i)\lg(n/i))$。

c. 证明：如果 i 是小于 $n/2$ 的常数，则有 $U_i(n)=n+O(\lg n)$。

d. 证明：如果对所有 $k \geqslant 2$ 有 $i=n/k$，则 $U_i(n)=n+O(T(2n/k)\lg k)$。

9-4 (随机选择的另一种分析方法) 在这个问题中，我们用指示器随机变量来分析 RANDOMIZED-SELECT，这一方法类似于 7.4.2 节中所用的对 RANDOMIZED-QUICKSORT 的分析方法。

与快速排序中的分析一样，我们假设所有的元素都是互异的，输入数组 A 的元素被重命名为 z_1，z_2，…，z_n，其中 z_i 是第 i 小的元素。因此，调用 RANDOMIZED-SELECT(A，1，n，k)返回 z_k。

对所有 $1 \leqslant i < j \leqslant n$，设

$$X_{ijk} = \mathrm{I}\{\text{在执行算法查找 } z_k \text{ 期间，} z_i \text{ 与 } z_j \text{ 进行过比较}\}$$

a. 给出 $\mathrm{E}[X_{ijk}]$的准确表达式。(提示：你的表达式可能有不同的值，依赖于 i、j、k 的值。)

b. 设 X_k 表示在找到 z_k 时 A 中元素的总比较次数，证明：

$$\mathrm{E}[X_k] \leqslant 2\left(\sum_{i=1}^{k}\sum_{j=k}^{n}\frac{1}{j-i+1}+\sum_{j=k+1}^{n}\frac{j-k-1}{j-k+1}+\sum_{i=1}^{k-2}\frac{k-i-1}{k-i+1}\right)$$

c. 证明：$\mathrm{E}[X_k] \leqslant 4n$。

226 **d.** 假设 A 中的元素都是互异的，证明：RANDOMIZED-SELECT 的期望运行时间是 $O(n)$。

本章注记

最坏情况下线性时间查找中位数的算法是由 Blum、Floyd、Pratt、Rivest 和 Tarjan[50]设计的。快速的随机化版本则是由 Hoare[169]提出的。Floyd 与 Rivest[108]设计了一个改进的随机化版本，它递归地从一个小的样本集中选取元素作为划分的主元。

目前，确定中位数所需的精确比较次数仍然是未知的。Bent 与 John[41]给出了一个寻找中位数的比较次数的下界，即 $2n$。Schönhage、Paterson 和 Pippenger[302]给出了一个 $3n$ 的上界。Dor 和 Zwick 证明了上述这两个界，并给出了一个略小的上界 $2.95n$[93]，他们给出的下界是$(2+\varepsilon)n$[94]，以一个
227 很小的正数 ε，略微改进了 Dor 等人[92]的结果。Paterson[272]描述了这些结果以及其他相关的工作。