对于某些概率分布以及 k 的一些值，答案都可以精确地计算出来。然而随着 k 的增大，分析明显地变得困难，因此使用计算机模拟银行的运作很有吸引力。用这种方法，银行管理人员可以确定为保证合理、通畅的服务需要多少出纳员。

模拟由处理中的事件组成。这里的两个事件是(a)一位顾客到达，和(b)一位顾客离去，从而腾出一名出纳员。

我们可以使用概率函数来生成一个输入流，它由每位顾客的到达时间和服务时间的序偶组成，并按到达时间排序。我们不必使用一天中的准确时间，而是使用一份单位时间量，称之为一个滴答(tick)。

进行这种模拟的一种方法是启动处在 0 滴答处的一台模拟钟表。我们让钟表一次走一个滴答，同时查看是否有事件发生。如果有，就处理这个(些)事件，搜集统计资料。当没有顾客留在输入流且所有的出纳员都空闲的时候，模拟结束。

这种模拟策略的问题是，它的运行时间不依赖顾客数或事件数(每位顾客有两个事件)，但是却依赖滴答数，而后者实际又不是输入的一部分。为了看清为什么问题在于此，假设将钟表的单位改成毫(千分之一)滴答(millitick)并将输入中的所有时间乘以 1000，则结果将是：模拟用时长 1000 倍!

避免这种问题的关键是在每一个阶段让钟表直接走到下一个事件时间。从概念上看这是容易做到的。在任一时刻，可能出现的下一事件要么是(a)在输入文件中下一顾客的到达，要么是(b)在一名出纳员处一位顾客离开。由于事件将要发生的所有的时间都是可以达到的，因此我们只需找出在最近的将来发生的事件并处理这个事件。

如果事件是离开，那么处理过程包括搜集离开的顾客的统计资料以及检验队伍(队列)看是否还有另外的顾客在等待。如果有，那么我们加上这位顾客，处理需要的统计资料，计算顾客将要离开的时间，并将离开加到等待发生的事件集中去。

如果事件是到达，则检查处于空闲的出纳员。如果没有，就把该到达放到队伍(队列)中去；否则，我们分配顾客一个出纳员，计算顾客的离开时间，并将离开加到等待发生的事件集中去。

顾客在等待的队伍可以实现为一个队列。由于我们需要找到最近的将来发生的事件，因此合适的办法是将等待发生的离开的集合编入一个优先队列中。下一事件是下一个到达或下一个离开(哪个先发生就是哪个)；它们都容易达到。

现在就可以为模拟编写例程了，虽然很可能耗费时间。如果有 C 个顾客(因此有 $2C$ 个事件)和 k 个出纳员，那么模拟的运行时间将会是 $O(C \log(k+1))$，因为计算和处理每个事件花费 $O(\log H)$，其中 $H = k+1$ 为堆的大小[○]。

6.5 d-堆

二叉堆是如此简单，以至于它们几乎总是用在需要优先队列的时候。d-堆是二叉堆的简单推广，它就像一个二叉堆，只是所有的节点都有 d 个儿子(因此，二叉堆是 2-堆)。

图 6-19 表示的是一个 3-堆。注意，d-堆要比二叉堆浅得多，它将 insert 操作的运行时间改进为 $O(\log_d N)$。然而，对于大的 d，deleteMin 操作费时得多，因为虽然树是浅了，但是 d 个儿子中的最小者是必须要找出的，如使用标准的算法，这会花费 $d-1$ 次比较，于是将操作的用时提高到 $O(d \log_d N)$。如果 d 是常数，那么当然两个的运行时间都是 $O(\log N)$。虽然仍然可以使用一个数组，但是，现在找出儿子和父亲的乘法和除法都有个因子 d，除非 d 是 2 的幂，否则

○ 我们用 $O(C \log(k+1))$而不用 $O(C \log k)$以避免 $k=1$ 情形的混乱。