The reason that this is a greedy algorithm is that at each stage we perform a merge without regard to global considerations. We merely select the two smallest trees.

If we maintain the trees in a priority queue, ordered by weight, then the running time is $O(C \log C)$, since there will be one *build_heap*, $2C - 2$ *delete_min*s, and $C - 2$ *insert*s, on a priority queue that never has more than C elements. A simple implementation of the priority queue, using a linked list, would give an $O(C^2)$ algorithm. The choice of priority queue implementation depends on how large C is. In the typical case of an ASCII character set, C is small enough that the quadratic running time is acceptable. In such an application, virtually all the running time will be spent on the disk I/O required to read the input file and write out the compressed version.

There are two details that must be considered. First, the encoding information must be transmitted at the start of the compressed file, since otherwise it will be impossible to decode. There are several ways of doing this; see Exercise 10.4. For small files, the cost of transmitting this table will override any possible savings in compression, and the result will probably be file expansion. Of course, this can be detected and the original left intact. For large files, the size of the table is not significant.

The second problem is that as described, this is a two-pass algorithm. The first pass collects the frequency data and the second pass does the encoding. This is obviously not a desirable property for a program dealing with large files. Some alternatives are described in the references.

10.1.3. Approximate Bin Packing

In this section, we will consider some algorithms to solve the *bin packing* problem. These algorithms will run quickly but will not necessarily produce optimal solutions. We will prove, however, that the solutions that are produced are not too far from optimal.

We are given n items of sizes $s_1, s_2, \ldots, s_n$. All sizes satisfy $0 < s_i \leq 1$. The problem is to pack these items in the fewest number of bins, given that each bin has unit capacity. As an example, Figure 10.20 shows an optimal packing for an item list with sizes 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8.

Figure 10.20 Optimal packing for 0.2, 0.5, 0.4, 0.7, 0.1, 0.3, 0.8