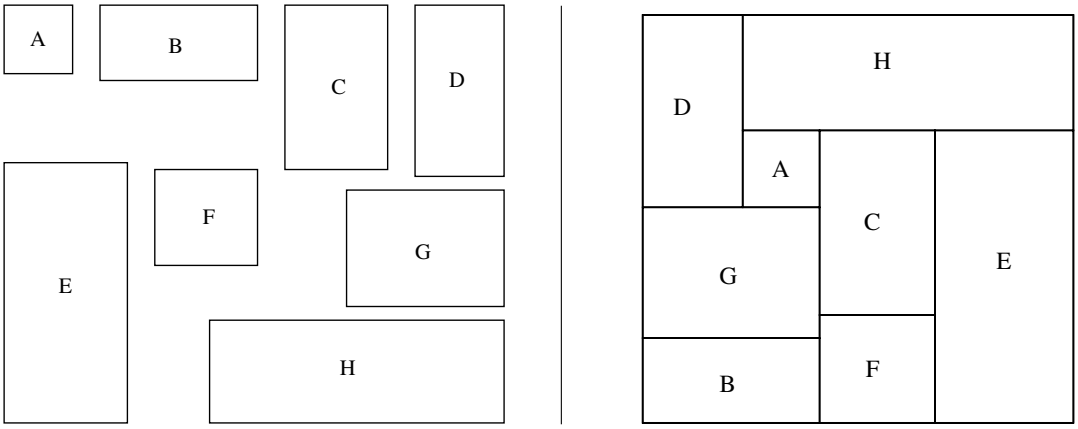

INPUT OUTPUT

17.9 Bin Packing

Input description: A set of n items with sizes $d_1, \ldots, d_n$. A set of m bins with capacity $c_1, \ldots, c_m$.

Problem description: Store all the items using the smallest number of bins.

Discussion: Bin packing arises in a variety of packaging and manufacturing problems. Suppose that you are manufacturing widgets cut from sheet metal or pants cut from cloth. To minimize cost and waste, we seek to lay out the parts so as to use as few fixed-size metal sheets or bolts of cloth as possible. Identifying which part goes on which sheet in which location is a bin-packing variant called the *cutting stock* problem. Once our widgets have been successfully manufactured, we are faced with another bin-packing problem—namely how best to fit the boxes into trucks to minimize the number of trucks needed to ship everything.

Even the most elementary-sounding bin-packing problems are NP-complete; see the discussion of integer partition in Section 13.10 (page 427). Thus, we are doomed to think in terms of heuristics instead of worst-case optimal algorithms. Fortunately, relatively simple heuristics tend to work well on most bin-packing problems. Further, many applications have peculiar, problem-specific constraints that tend to frustrate sophisticated algorithms for bin packing. The following factors will affect the choice of heuristic:

- *What are the shapes and sizes of the objects?* – The character of a bin-packing problem depends greatly on the shapes of the objects to be packed. Solving a standard jigsaw puzzle is a much different problem than packing squares

into a rectangular box. In *one-dimensional bin packing*, each object's size is given simply as an integer. This is equivalent to packing boxes of equal width into a chimney of that width, and makes it a special case of the knapsack problem of Section 13.10 (page 427).

When all the boxes are of identical size and shape, repeatedly filling each row gives a reasonable, but not necessarily optimal, packing. Consider trying to fill a 3×3 square with 2×1 bricks. You can only pack three bricks using one orientation, while four bricks suffice with two.

- *Are there constraints on the orientation and placement of objects?* – Many boxes are labeled "this side up" (imposing an orientation on the box) or "do not stack" (requiring them sit on top of any box pile). Respecting these constraints restricts our flexibility in packing and hence will increase in the number of trucks needed to send out certain shipments. Most shippers solve the problem by ignoring the labels. Indeed, your task will be simpler if you don't have to worry about the consequences of them.

- *Is the problem on-line or off-line?* – Do we know the complete set of objects to pack at the beginning of the job (an *off-line* problem)? Or will we get them one at a time and deal with them as they arrive (an *on-line* problem)? The difference is important, because we can do a better job packing when we can take a global view and plan ahead. For example, we can arrange the objects in an order that will facilitate efficient packing, perhaps by sorting them from biggest to smallest.

The standard off-line heuristics for bin packing order the objects by size or shape and then insert them into bins. Typical insertion rules are (1) select the first or leftmost bin the object fits in, (2) select the bin with the most room, (3) select the bin that provides the tightest fit, or (4) select a random bin.

Analytical and empirical results suggest that *first-fit decreasing* is the best heuristic. Sort the objects in decreasing order of size, so that the biggest object is first and the smallest last. Insert each object one by one into the first bin that has room for it. If no bin has room, we must start another bin. In the case of one-dimensional bin packing, this can never require more than 22% more bins than necessary and usually does much better. First-fit decreasing has an intuitive appeal to it, for we pack the bulky objects first and hope that little objects can fill up the cracks.

First-fit decreasing is easily implemented in $O(n \lg n + bn)$ time, where $b \leq \min(n, m)$ is the number of bins actually used. Simply do a linear sweep through the bins on each insertion. A faster $O(n \lg n)$ implementation is possible by using a binary tree to keep track of the space remaining in each bin.

We can fiddle with the insertion order in such a scheme to deal with problem-specific constraints. For example, it is reasonable to take "do not stack" boxes last (perhaps after artificially lowering the height of the bins to leave some room up

top to work with) and to place fixed-orientation boxes at the beginning (so we can use the extra flexibility later to stick boxes on top).

Packing boxes is much easier than packing arbitrary geometric shapes, enough so that a general approach packs each part into its own box and then packs the boxes. Finding an enclosing rectangle for a polygonal part is easy; just find the upper, lower, left, and right tangents in a given orientation. Finding the orientation and minimizing the area (or volume) of such a box is more difficult, but can be done in both two and three dimensions [O'R85]. See the Implementations section for a fast approximation to minimum enclosing box.

In the case of nonconvex parts, considerable useful space can be wasted in the holes created by placing the part in a box. One solution is to find the *maximum empty rectangle* within each boxed part and use this to contain other parts if it is sufficiently large. More advanced solutions are discussed in the references.

Implementations: Martello and Toth's collection of Fortran implementations of algorithms for a variety of knapsack problem variants are available at *http://www.or.deis.unibo.it/kp.html*. An electronic copy of [MT90a] has also been generously made available.

David Pisinger maintains a well-organized collection of C-language codes for knapsack problems and related variants like bin packing and container loading. These are available at *http://www.diku.dk/∼pisinger/codes.html*.

A first step towards packing arbitrary shapes packs each in its own minimum volume box. For a code to find an approximation to the optimal packing, see *http://valis.cs.uiuc.edu/∼sariel/research/papers/00/diameter/diam_prog.html*. This algorithm runs in near-linear time [BH01].

Notes: See [CFC94, CGJ96, LMM02] for surveys of the extensive literature on bin packing and the cutting stock problem. Keller, Pferschy, and Psinger [KPP04] is the most current reference on the knapsack problem and variants. Experimental results on bin-packing heuristics include [BJLM83, MT87].

Efficient algorithms are known for finding the largest empty rectangle in a polygon [DMR97] and point set [CDL86].

Sphere packing is an important and well-studied special case of bin packing, with applications to error-correcting codes. Particularly notorious was the "Kepler conjecture"—the problem of establishing the densest packing of unit spheres in three dimensions. This conjecture was finally settled by Hales and Ferguson in 1998; see [Szp03] for an exposition. Conway and Sloane [CS93] is the best reference on sphere packing and related problems.

Milenkovic has worked extensively on two-dimensional bin-packing problems for the apparel industry, minimizing the amount of material needed to manufacture pants and other clothing. Reports of this work include [DM97, Mil97].

Related Problems: Knapsack problem (see page 427), set packing (see page 625).