Algorithm 9.4 Greedy search for constructing an elimination ordering

```
Procedure Greedy-Ordering (
    H    // An undirected graph over X
    s    // An evaluation metric
)
1   Initialize all nodes in X as unmarked
2   for k = 1 ... |X|
3     Select an unmarked variable X ∈ X that minimizes s(H, X)
4     π(X) ← k
5     Introduce edges in H between all neighbors of X
6     Mark X
7   return π
```

find an ordering that achieves the global minimum, we can eliminate variables one at a time in a greedy way, so that each step tends to lead to a small blowup in size.

The general algorithm is shown in algorithm 9.4. At each point, the algorithm evaluates each of the remaining variables in the network based on its heuristic cost function. Some common cost criteria that have been used for evaluating variables are:

- **Min-neighbors:** The cost of a vertex is the number of neighbors it has in the current graph.
- **Min-weight:**The cost of a vertex is the product of *weights* — domain cardinality — of its neighbors.
- **Min-fill:** - The cost of a vertex is the number of edges that need to be added to the graph due to its elimination.
- **Weighted-min-fill:** The cost of a vertex is the sum of weights of the edges that need to be added to the graph due to its elimination, where a weight of an edge is the product of weights of its constituent vertices.

Intuitively, min-neighbors and min-weight count the size or weight of the largest clique in $\mathcal{H}$ after eliminating X. Min-fill and weighted-min-fill count the number or weight of edges that would be introduced into $\mathcal{H}$ by eliminating X. It can be shown (exercise 9.10) that none of these criteria is universally better than the others.

This type of greedy search can be done either deterministically (as shown in algorithm 9.4), or stochastically. In the stochastic variant, at each step we select some number of low-scoring vertices, and then choose among them using their score (where lower-scoring vertices are selected with higher probability). In the stochastic variants, we run multiple iterations of the algorithm, and then select the ordering that leads to the most efficient elimination — the one where the sum of the sizes of the factors produced is smallest.

Empirical results show that these heuristic algorithms perform surprisingly well in practice. Generally, Min-Fill and Weighted-Min-Fill tend to work better on more problems. Not surprisingly, Weighted-Min-Fill usually has the most significant gains when there is some significant variability in the sizes of the domains of the variables in the network. Box 9.C presents a case study comparing these algorithms on a suite of standard benchmark networks.