Greedy Algorithm

Greedy algorithms are algorithms which follow the problem solving meta-heuristic of making the locally optimum choice at each stage with the hope of finding the global optimum. For instance, applying the greedy strategy to the traveling salesman problem yields the following algorithm: "At each stage visit the nearest unvisited city to the current city.

Greedy algorithms do not consistently find the globally optimal solution, because they usually do not operate exhaustively on all the data. They can make commitments to certain choices too early which prevent them from finding the best overall solution later. For example, all known greedy algorithms for the graph coloring problem and all other NP-complete problems do not consistently find optimum solutions. Nevertheless, they are useful because they are quick to think up and often give good approximations to the optimum. If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice. Examples of such greedy algorithms are Kruskal's algorithm and Prim's algorithm for finding minimum spanning trees and the algorithm for finding optimum Huffman trees. The theory of matroids, as well as the even more general theory of greedoids, provide whole classes of such algorithms.

In general, greedy algorithms have five pillars:

- A candidate set, from which a solution is created
- A selection function, which chooses the best candidate to be added to the solution
- A feasibility function, that is used to determine if a candidate can be used to contribute to a solution
- An objective function, which assigns a value to a solution, or a partial solution, and
- A solution function, which will indicate when we have discovered a complete solution

Example : There is a long list of stalls, some of which need to be covered with boards. You can use up to N ($1 \leq N \leq 50$) boards, each of which may cover any number of consecutive stalls. Cover all the necessary stalls, while covering as few total stalls as possible. How will you solve it?

The basic idea behind greedy algorithms is to build large solutions up from smaller ones. Unlike other approaches, however, greedy algorithms keep only the best solution they find as they go along. Thus, for the sample problem, to build the answer for N = 5, they find the best solution for N = 4, and then alter it to get a solution for N = 5. No other solution for N = 4 is ever considered. Greedy algorithms are fast, generally linear to quadratic and require little extra memory. Unfortunately, they usually aren't correct. But when they do work, they are often easy to implement and fast enough to execute.

Problems with Greedy algorithms

There are two basic problems to greedy algorithms.

1. **How to build ?**

How does one create larger solutions from smaller ones? In general, this is a function of the problem. For the sample problem, the most obvious way to go from four boards to five boards is to pick a board and remove a section, thus creating two boards from one. You should choose to remove the largest section from any board which covers only stalls which don't need covering (so as to minimize the total number of stalls covered).

To remove a section of covered stalls, take the board which spans those stalls, and make into two boards: one of which covers the stalls before the section, one of which covers the stalls after the second.

2. **Does it work ?**

The real challenge for the programmer lies in the fact that greedy solutions don't always work. Even if they seem to work for the sample input, random input, and all the cases you can think of, if there's a case where it won't work, at least one (if not more!) of the judges' test cases will be of that form.

For the sample problem, to see that the greedy algorithm described above works, consider the following:

Assume that the answer doesn't contain the large gap which the algorithm removed, but does contain a gap which is smaller. By combining the two boards at the end of the smaller gap and splitting the board across the larger gap, an answer is obtained which uses as many boards as the original solution but which covers fewer stalls. This new answer is better, so therefore the assumption is wrong and we should always choose to remove the largest gap.

If the answer doesn't contain this particular gap but does contain another gap which is just as large, doing the same transformation yields an answer which uses as many boards and covers as many stalls as the other answer. This new answer is just as good as the original solution but no better, so we may choose either.

Thus, there exists an optimal answer which contains the large gap, so at each step, there is always an optimal answer which is a superset of the current state. Thus, the final answer is optimal.

If a greedy solution exists, use it. They are easy to code, easy to debug, run quickly, and use little memory, basically defining a good algorithm in contest terms. The only missing element from that list is correctness. If the greedy algorithm finds the correct answer, go for it, but don't get suckered into thinking the greedy solution will work for all problems.

Source:

http://www.learnalgorithms.in/#