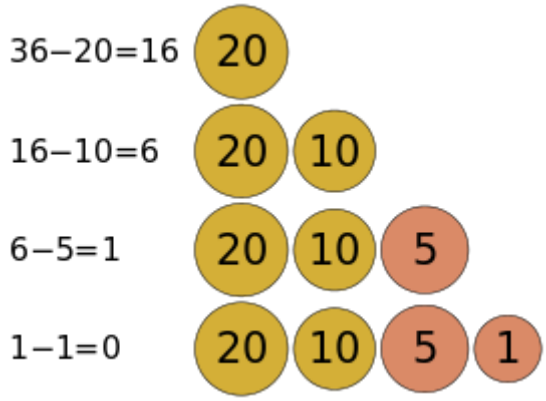
Greedy algorithm

A **greedy algorithm** is an algorithmic paradigm that follows the problem solving heuristic of making the locally optimal choice at each stage[1] with the hope of finding a global optimum. In many problems, a greedy strategy does not in general produce an optimal solution, but nonetheless a greedy heuristic may yield locally optimal solutions that approximate a global optimal solution in a reasonable time.

For example, a greedy strategy for the traveling salesman problem (which is of a high computational complexity) is the following heuristic: "At each step of the journey, visit the nearest unvisited city." This heuristic need not find a best solution, but terminates in a reasonable number of steps; finding an optimal solution typically requires unreasonably many steps. In mathematical optimization, greedy algorithms solve combinatorial problems having the properties of matroids.



Greedy algorithms determine minimum number of coins to give while making change. These are the steps a human would take to emulate a greedy algorithm to represent 36 cents using only coins with values {1, 5, 10, 20}. The coin of the highest value, less than the remaining change owed, is the local optimum. (In general the change-making problem requires dynamic programming to find an optimal solution; however, most currency systems, including the Euro and US Dollar, are special cases where the greedy strategy does find an optimal solution.)

Contents

Specifics

In general, greedy algorithms have five components:

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

Greedy algorithms produce good solutions on some mathematical problems, but not on others. Most problems for which they work will have two properties:

Greedy choice property

We can make whatever choice seems best at the moment and then solve the subproblems that arise later. The choice made by a greedy algorithm may depend on choices made so far, but not on future choices or all the solutions to the subproblem. It iteratively makes one

greedy choice after another, reducing each given problem into a smaller one. In other words, a greedy algorithm never reconsiders its choices. This is the main difference from dynamic programming, which is exhaustive and is guaranteed to find the solution. After every stage, dynamic programming makes decisions based on all the decisions made in the previous stage, and may reconsider the previous stage's algorithmic path to solution.

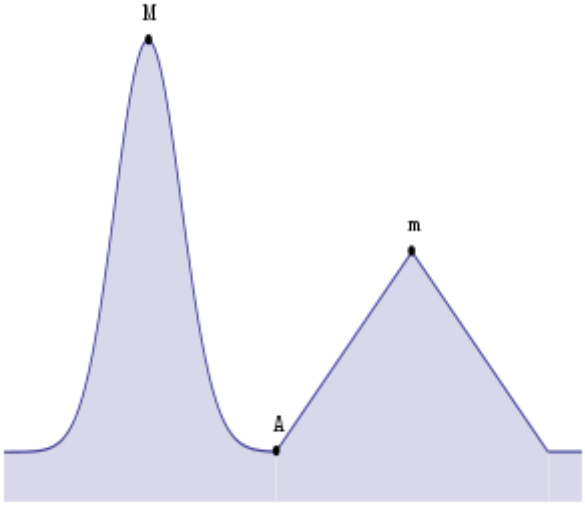
Optimal substructure

"A problem exhibits optimal substructure if an optimal solution to the problem contains optimal solutions to the sub-problems."[2]

Cases of failure

For many other problems, greedy algorithms fail to produce the optimal solution, and may even produce the *unique worst possible* solution. One example is the traveling salesman problem mentioned above: for each number of cities, there is an assignment of distances between the cities for which the nearest neighbor heuristic produces the unique worst possible tour[3]

Examples on how a greedy algorithm may fail to achieve the optimal solution.



Starting at A, a greedy algorithm will find the local maximum at "m", oblivious to the global maximum at "M".

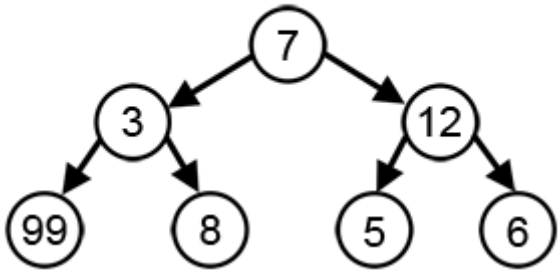
Types

Greedy algorithms can be characterized as being 'short sighted', and also as 'non-recoverable'. They are ideal only for problems which have 'optimal substructure'. Despite this, for many simple problem (e.g. giving change), the best suited algorithms are greedy algorithms. It is important, however, to note that the greedy algorithm can be used as a selection algorithm to prioritize options within a search, or branch-and-bound algorithm. There are a few variations to the greedy algorithm:

- Pure greedy algorithms
- Orthogonal greedy algorithms
- Relaxed greedy algorithms

Applications

Greedy algorithms mostly (but not always) fail to 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 coloring 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.



With a goal of reaching the largest-sum, at each step, the greedy algorithm will choose what appears to be the optimal immediate choice, so it will choose 12 instead of 3 at the second step, and will not reach the best solution, which contains 99.

If a greedy algorithm can be proven to yield the global optimum for a given problem class, it typically becomes the method of choice because it is faster than other optimization methods like dynamic programming. 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, and the more general theory of greedoids, provide whole classes of such algorithms.

Greedy algorithms appear in network routing as well. Using greedy routing, a message is forwarded to the neighboring node which is "closest" to the destination. The notion of a node's location (and hence "closeness") may be determined by its physical location, as in geographic routing used by ad hoc networks. Location may also be an entirely artificial construct as in small world routing and distributed hash table

Examples

- The activity selection problem is characteristic to this class of problems, where the goal is to pick the maximum number of activities that do not clash with each other
- In the Macintosh computer game *Crystal Quest* the objective is to collect crystals, in a fashion similar to the travelling salesman problem. The game has a demo mode, where the game uses a greedy algorithm to go to every crystal. The artificial intelligence does not account for obstacles, so the demo mode often ends quickly
- The matching pursuit is an example of greedy algorithm applied on signal approximation.
- A greedy algorithm finds the optimal solution to Malfatti's problem of finding three disjoint circles within a given triangle that maximize the total area of the circles; it is conjectured that the same greedy algorithm is optimal for any number of circles.
- A greedy algorithm is used to construct a Huffman tree during Huffman coding where it finds an optimal solution.
- In decision tree learning greedy algorithms are commonly used, however they are not guaranteed to find the optimal solution.

See also

- Epsilon-greedy strategy
- Greedy algorithm for Egyptian fractions
- Greedy source
- Matroid

Notes

1. Black, Paul E. (2 February 2005). "greedy algorithm" (http://xlinux.nist.gov/dads//HTML/greedyalgo.html) *Dictionary of Algorithms and Data Structures* U.S. National Institute of Standards and Technology (NIST). Retrieved 17 August 2012.
2. Introduction to Algorithms (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 "Greedy Algorithms".
3. Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP (http://www.sciencedirect.com/science/article/pii/S0166218X01001950) (G. Gutin, A. Yeo and A. Zverovich, 2002)

References

- *Introduction to Algorithms* (Cormen, Leiserson, and Rivest) 1990, Chapter 17 "Greedy Algorithms" p. 329.
- *Introduction to Algorithms* (Cormen, Leiserson, Rivest, and Stein) 2001, Chapter 16 "Greedy Algorithms".
- G. Gutin, A. Yeo and A. Zverovich, Traveling salesman should not be greedy: domination analysis of greedy-type heuristics for the TSP. Discrete Applied Mathematics 117 (2002), 81–86.
- J. Bang-Jensen, G. Gutin and A. Yeo, When the greedy algorithm fails. Discrete Optimization 1 (2004), 121–127.
- G. Bendall and F. Margot, Greedy Type Resistance of Combinatorial Problems, Discrete Optimization 3 (2006), 288–298.

External links

- Hazewinkel, Michiel, ed. (2001) [1994], "Greedy algorithm", *Encyclopedia of Mathematics*, Springer Science+Business Media B.V. / Kluwer Academic Publishers, ISBN 978-1-55608-010-4

- Python greedy coin example by Noah Gift.