The Greedy Algorithm: USACO Bronze

Videh Seksaria, TJHSST SCT

1 Introduction

A Greedy Algorithm is similar to a dynamic programming algorithm, they both are optimization algorithms. Greedy algorithms work by repeatedly selecting a local maxima, to find the global maxima. They are efficient, powerful, and easy to code. The problem with them is that it is often difficult to decide when to use one. The hardest part of applying a greedy algorithm is simply realizing that a problem is greedy. The most common error is applying greedy where it does not apply. Many problems appear to be greedy, but actually require a dynamic programming solution. If you can prove a greedy solution, then use it, but be careful that the solution is actually greedy.

2 Examples

The Fractional Knapsack Problem:
Problem: Shreenath was at the mall when he saw different boxes of chocolate bars. He wanted to buy as much as he could, so he borrowed a knapsack of size n. In the store, there were k chocolate packs. For each pack i, he could take each bar of chocolate individually without taking the entire pack. Each pack held bars of different weight, wi, and different value, vi. Find the greatest amount of chocolate he can buy and hold in his knapsack, if he is given unlimited funding. Note: Being the Holiday Season, the store does not mind if Shreenath splits a bar of chocolate into pieces, and only selects one.
Solution: Simply sort the donuts buy value to pound ration, and iterate backwards, selecting each chocolate bar until we reach the limit on space. If we have space at the end, we can split a bar to take that remaining space.

The Task Scheduling Problem:
Problem:
We have n tasks that all require the same set of resources, and only one task can run at one time. Each activity i starts at time si, and ends at time ei, where $si \leq ei$. What is the maximum number of tasks that can be performed.
Solution: Sort the tasks by increasing order of the time they finish(if there is a tie, pick either choice). Iterate through the list, choosing the first task, and every other task based upon compatibility by those previously selected.

3 Types of Greedy Algorithms

By now, you have probably realized the two types of greedy algorithms. The sort/scan, and the iterative type. In the sort/scan type, you sort the given elements by a certain attribute, then pick them off, one by one. The above problems are examples of this type. Iterative greedy problems can be divided into n stages, where at each stage, the optimal may be chosen greedily. These kinds of greedy problems are often actually require dynamic programming, so be careful.

4 Greedy vs. Dynamic Programming

Both techniques are optimization techqieus, and build solutions form a collection of choices. The greedy method, however, makes its choices going forward, never looking back, or ahead. Dynamic programming on the other hand computes the solution from the bottom up, and by joining numerous sub solutions. Any greedy problem can be solved with DP, few DP problems can be solved by greedy. Consider the famous Coin Problem. Given coin dominations (including a 1 cent coin), and a value of money to create, find the smallest number of coins coins required to create that amount. The greedy solution of taking the largest denomination when possible, does not always work. The correct solution if to use dynamic programming.

5 Problems

1. [Dean, Kolstad, 2006]
Farmer John has leashed his N ($1 \leq N \leq 50; 000$) cows to stakes (at integer locations) next to a fence. Every cow is straining her leash as much as she can to the east (though never beyond the end of the fence). FJ's wife can locate herself halfway between two integer points along the fence and cut all the leashes that cross in front of her at that place. Given the length of each leash and the location of its stake in the ground, what is the minimal number of cuts she must perform to free the cows?

2. Fractional Knapsack [Traditional]
You are a thief, who has entered a jewelry store. In the store, there are N boxes, each with a type of jewel worth some value V and weighing some weight W. Your knapsack can only carry X units of weight. If there is an infinite amount of jewelry in each box, what is the maximum value you can fit in your knapsack if you can choose only to take part of a jewel?

3. Barn Repair [1999 USACO Spring Open]
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.

4. Mixing Milk [USACO Training Pages]
Since milk packaging is such a low margin business, it is important to keep the price of the raw product (milk) as low as possible. Help Merry Milk Makers get the milk they need in the cheapest possible manner. The Merry Milk Makers company has several farmers from which they may buy milk, and each one has a (potentially) different price at which they sell to the milk packing plant. Moreover, as a cow can only produce so much milk a day, the farmers only have so much milk to sell per day. Each day, Merry Milk Makers can purchase an integral amount of milk from each farmer, less than or equal to the farmer's limit. Given the Merry Milk Makers' daily requirement of milk, along with the cost per gallon and amount of available milk for each farmer, calculate the minimum amount of money that it takes to fulfill the Merry Milk Makers' requirements. Note: The total milk produced per day by the farmers will be sucient to meet the demands of the Merry Milk Makers.

5. PlayGame [TopCoder 2004]
You are playing a computer game and a big fight is planned between two armies. You and your computer opponent will line up your respective units in two rows, with each of your units facing exactly one of your opponents units and vice versa. Then, each pair of units, who face each other will fight and the stronger one will be victorious, while the weaker one will be captured. If two opposing units are equally strong, your unit will lose and be captured. You know how the computer will arrange its units, and must decide how to line up yours. You want to maximize the sum of the strengths of your

units that are not captured during the battle.

6. Indivisible [TopCoder 2005]
Given a positive integer n, you are to find the largest subset S of 1,...,n such that no member of S is divisible by another member of S. If there is more than one subset of the maximum size, choose the lexicographically earliest such subset. A subset S is lexicographically earlier than a subset T if the smallest element that is a member of one of the subsets, but not both, is a member of S. Return the subset S as a int[] in increasing order.

7. MatrixTransforming [TopCoder 2006]
Given two matrices a and b, both composed of zeros and ones, return the minimal number of operations necessary to transform matrix a into matrix b. An operation consists of flipping (one becomes zero and zero becomes one) all elements of some contiguous 3 3 submatrix. If a cannot be transformed into b, return -1.

8. Cow Tanning [Cow, 2001]
To avoid burns while tanning, each of the C ($1 \leq C \leq 50{,}000$) cows must cover her hide with sunscreen when they're at the beach. Cow i has a minimum and maximum SPF rating that will work. If the SPF rating is too low, the cow will sunburn; if the SPF rating is too high, the cow doesn't tan. The cows have L ($1 \leq \mathrm{L} \leq 50{,}000$) bottles of sunscreen lotion, each bottle i with an SPF rating. Bottle i can cover *coveri* ($1 \leq$ *coveri* $\leq$ C) cows with lotion. A cow may lotion from only one bottle. What is the maximum number of cows that can protect themselves while tanning given the available lotions?

Credits: Problems taken from old SCT lectures, and other sources. Basic formatting and ideas also taken from previous SCT greedy algorithm lectures.