Problem Set 7

November 5, 2015

16.2-1

Prove that the fractional knapsack problem has the greedy-choice property .

Solution

Lets assume that the knapsack can carry some weight,$W > 0$. Let I be the following instance of the knapsack problem: Let n be the number of items, let v_i be the value of the i^{th} item, let w_i be the weight of the i^{th} item. Assume the items have been ordered in increasing order by $\frac{v_i}{w_i}$ and that $W \geq w_n$. Let $s = (s_1, s_2, ..., s_n)$ be a solution. The greedy algorithm works by assigning $s_n = min(w_n, W)$, and then continuing by solving the subproblem $I' = (n1, v_1, v_2, ..., v_{n1}, w_1, w_2, ..., w_{n1}, W w_n)$ until it either reaches the state $W = 0$ or $n = 0$.

Suppose the optimal solution to I is $s_1, s_2, ..., s_n$, where $s_n < min(w_n, W)$. Let i be the smallest number such that $s_i > 0$. By decreasing s_i to $max(0, W w_n)$ and increasing s_n by the same amount, we get a better solution. Since this a contradiction the assumption must be false. Hence the problem has the greedy-choice property.

16.2-2

Give a dynamic-programming solution to the 0-1 knapsack problem that runs in $O(nW)$ time where n is the number of items and W is the maximum weight of items that the thief can put in his knapsack.

Solution

The algorithm takes as inputs the maximum weight W, the number of items n, and two sequences $v = (v_1, v_2, ..., v_n)$ and $w = (w_1, w_2, ..., w_n)$

```
DYNAMIC-0-1-KNAPSACK(v,w,n,W)
    for w = 0 to W
                do c[0,w] = 0
    for i = 1 to n
        do c[i,0] = 0
        for w = 1 to W do
            do if w_i <= w then
                        if v_i + c[i-1,w-w_i] > c[i-1,w] then
                        c[i,w] = v_i + c[i-1,w-w_i]
                        else
                                c[i,w] = c[i-1,w]
                        end if
                   else
                  c[i,w] = c[i-1,w]
                   end if
    return c[n,W]
```

The above algorithm takes $\theta(nW)$ total time. It takes $\theta(nW)$ to fill in the c table $(n+1)(W+1)$ entries each requiring $\theta(1)$ time to compute.

16.2-5

Describe an efficient algorithm that, given a set $\{x_1, x_2, ..., x_n\}$ of points on the real line, determines the smallest set of unit-length closed intervals that contains all of the given points. Argue that your algorithm is correct.

Solution

Algorithm : Set a pointer to negative infinity. Execute a while loop that conditions on all points not being covered by an interval yet. In each iteration find the lowest value point x_i such that $x_i > pointer$. Create a new interval $[x_i, x_i+1]$ and add it to the set of intervals. Then set pointer $= x_i+1$ and return to the beginning of the while loop. When the loop exits we have an optimal set of intervals that covers all points.

The **correcteness** of algorithm can be checked by the following scenario:
Some interval must cover the lowest valued point x_1. Of all possible intervals that contain x_1, $[x_1, x_1+1]$ is the optimal interval because no point in the given set has a value less than x_1. So setting the lower end of the interval as x_1 maximizes the number of points this interval can cover. Now we look for the next x_i, such that$x_i < x_j$ for $i < j < n$ and that $x_i > x_1 + 1$. There is some interval that must contain the point x_i. Given the first interval that we decided optimal for x_1, this interval is optimal for x_i because there is o value smaller than x_i that is not covered and so setting the lower end of this interval to x_i maximizes

the number of points that this interval can contain. The same argument can be repeatedly applied for the optimality of each interval choosen.

16-1

Coin Changing

Consider the problem of making change for n cents using the fewest number of coins. Assume that each coin's value is an integer.

a. Describe a greedy algorithm to make change consisting of quarters, dimes, nickels and pennies. Prove that your algorithm yeilds an optimal solution.

b. Suppose that the available coins are in the denominations that are powers of c, i.e., the denominations are $c^0, c^1, ..., c^k$ for some integers $c > 1$ and $k \geq 1$. Show that the greedy algorithm always yields an optimal solution.

c. Give a set of coin denominations for which the greedy algorithm does not yield an optimal solution. Your set should include a penny so that there is a solution for every value of n.

d. Give an $O(nk)$-time algorithm that makes change for any set of k different coin denominations, assuming that one of the coins is a penny.

Solution

a. Determine the largest coin whose value is less than or equal to n. Let this coin have value c. Give one such coin, and then recursively solve the subproblem of making change for $n - c$ cents.

To prove that this algorithm yields an optimal solution, we first need to show that the greedy-choice property holds, that is, that some optimal solution to making change for n cents includes one coin of value c, where c is the largest coin value such that $c \leq n$. Consider some optimal solution. If this optimal solution includes a coin of value c, then we are done. Otherwise, this optimal solution does not include a coin of value c. We have four cases to consider:

If $1 \leq n < 5$, then $c = 1$. A solution may consist only of pennies, and so it must contain the greedy choice.

If $5 \leq n < 10$, then $c = 5$. By supposition, this optimal solution does not contain a nickel, and so it consists of only pennies. Replace The pennies by one nickel to give a solution with four fewer coins.

If $10 \leq n < 25$, then $c = 10$. By supposition, this optimal solution does not contain a dime, and so it contains only nickels and pennies. Some subset of the nickels and pennies in this solution adds up to 10 cents, and so we can replace these nickels and pennies by a dime to give a solution with (between 1 and 9) fewer coins.

If $25 \leq n$, then c = 25. By supposition, this optimal solution does not contain a quarter, and so it contains only dimes, nickels, and pennies. If it contains three dimes, we can replace these three dimes by a quarter and a nickel, giving a solution with one fewer coin. If it contains at most two dimes, then some subset of the dimes, nickels, and pennies adds up to 25 cents, and so we can replace these coins by one quarter to give a solution with fewer coins.

Thus, we have shown that there is always an optimal solution that includes the greedy choice, and that we can combine the greedy choice with an optimal solution to the remaining subproblem to produce an optimal solution to our original problem. Therefore, the greedy algorithm produces an optimal solution.

b. Determine the largest j that $c^j \leq n$, give one coin of denomination c^j, and then recursively solve the subproblem of making change for $n - c^j$ cents.

To prove that the greedy algorithm produces an optimal solution, first we claim that, for i = 0, 1, . . . , k-1, the number of coins of denomination c^i used in an optimal solution for n cents is less than c. If not, we can improve the solution by using one more coin of denomination $c^i + 1$ and c fewer coins of denomination c^i.

Let$j = max\{0 \leq i \leq k : c^i \leq n\}$, so that the greedy solution uses at least one coin of denomination c^j; a nongreedy solution must use no coins of denomination c^j or higher. Thus for the non-greedy solution, we have,

$$\sum_{i=0}^{j-1} ai * c^i = n \geq c^j$$

However we have,$ai \leq c - 1$, so

$$\sum_{i=0}^{j-1} ai * c^i \leq \sum_{i=0}^{j-1} (c-1) * c^i = (c-1) \sum_{i=0}^{j-1} c^i = (c-1)(c^j 1)/(c-1) < c^j.$$

This contradiction shows the non-greedy solution is not optimal. And greedy algorithm has the running time O(k).

c. A simple example is the set of US denominations without the nickel. If you simply have the values 1, 10, 25, the greedy solution will fail.
For example, if you are trying to make change for 30 cents, the greedy solution will first take a quarter, followed by 5 pennies, a total of six coins. However, the optimal solution actually consists of three dimes.

d. Using dynamic programming. Let $c[j]$ denote the minimum number of coins we need to make change for j cents. Let the coin denominations be $d_1, d_2, ..., d_k$.

Since one of the coin is penny, there is a way to make change for any amount $j \geq 1$. Because of the optimal substructure, we have the following recursion:

$c[j] = 0$ if $j \leq 0$, and
$c[j] = 1 + min_{1 \leq i \leq k, d_i < j}\{c[j - d_i]\}$ if $j > 1$.

The size of memoization table is O(n). To fill up each cell will take O(k) comparisons. Therefore, It runs in O(nk) time.