Analysis of Algorithms

Homework 1

Abraham Murciano

November 8, 2020

1 The 0-1 Knapsack Problem

We are given a set of n items and a knapsack which can carry up to a weight of W. For each item i, its weight is w_i and its value v_i. Each item can be put into the knapsack only as a whole. The problem is to find a subset of items of maximal value which can be carried in the knapsack without exceeding the weight limit.

Part A

Suppose there is a greedy algorithm which attempts to solve this problem, defined as follows.

```
function GreedyKnapsack(items, w, v, W)
    let knapsack = {}
    let current weight = 0
    let current value = 0
    sort items by value to weight ratio (highest first)
    for i in items do
        if current weight + w_i ≤ W then
            insert i into knapsack
            increment current weight by w_i
            increment current value by v_i
    return knapsack
```

Now suppose we attempt to use this algorithm to give us the optimal knapsack for a list of items with the following parameters.

$$\text{items} = \{1, 2, 3\}$$
$$W = 9$$
$$w_1 = 6,\ w_2 = 5,\ w_3 = 4$$
$$v_1 = 7,\ v_2 = 5,\ v_3 = 4$$

Our algorithm would first sort the items by value to weight ratio, and since item 1 has a slightly higher ratio it will be the first item. Then the algorithm will check if item 1 fits into the knapsack, which it does. Therefore it will be inserted. Now, when it moves onto the next two items, neither will fit, since the knapsack already has item one. So the knapsack returned will only contain item 1, thus yielding a total value of 7.

However, this is not the optimal solution, since we can fit both items 2 and 3, for a total weight of 9, and a total value of 9.

Part B

The example described in Part A satisfies the condition that when the weights are sorted in increasing order, namely $(4, 5, 6)$, the values also turn out to be sorted in increasing order, namely $(4, 5, 7)$. As shown in Part A, this is in fact a counter example to the correctness of a greedy algorithm, so we may conclude that the greedy algorithm defined above is not necessarily correct even when the weights and values are in the same order.

2 Coin Exchange Problem

We are given a banknote of value A and (an unlimited amount of) coins of values $\{c^i : c, i \in \mathbb{N} \wedge 0 \leq i \leq k \wedge c > 1 \wedge k > 1\}$. The goal is to find a way to change the banknote for the minimal number of coins.

Part A

Suppose we have a problem of this format, whose optimal solution to reach the value A is some set of coins x (possibly including many different coins of the same value), we can be certain that for any subset of these coins $s \subseteq x$, s will be an optimal solution to reach the value which the coins in s sums up to.

This can be easily proven since if this was not the case, and there was a set t such that $|t| < |s|$, whose coins summed up to the same values as those in s, then we would have a solution to the initial problem $y = (x - s) \cup t$ such that $|y| < |x|$. And this is a contradiction to our assumption that x was an optimal solution.

Part B

A greedy algorithm that solves this problem is as follows.

---

function GreedyCoinExchange(c, k, A)
  if $A = 0$ **then return** $\{\}$
  let $a = \text{Min}(\lfloor log_c A \rfloor, k)$
  return $\{$a coin of value $c^a\} \cup$ GreedyCoinExchange(c, k, $A - c^a$)

To prove the correctness of this algorithm, we must show that the solution obtained by it is always optimal. We will do this by use of a couple of lemmas.

The first lemma we must use is as follows. Suppose that we have a set of coins S (possibly with multiple coins of the same denomination) whose values are all of the form c^j, where $j < k$, c is a constant, and j may change from coin to coin. Additionally, suppose that $\sum_{s \in S} \text{value}(s) \geq c^k$. Then there exists some set $T \subseteq S$ such that $\sum_{t \in T} \text{value}(t) = c^k$.

Our second lemma states that the optimal solution to reach the value A must contain at least one coin of the highest available value less than or equal to A.

We will prove the lemma by contradiction. Suppose the optimal solution to reach a goal of A did not contain the largest available coin value less than or equal to A. We'll call this value c^a, as labelled in the algorithm. That means that the optimal solution is a set of coins S, all smaller than or equal to c^{a-1}, which reach A, and reach or exceed c^a. However, by our first lemma, there must be some subset of S which exactly equals c^a, since all coins in S are powers of c. Therefore, we would be able to replace said subset with a single coin of value c^a, thus giving us a more optimal solution than S, showing that S was not optimal, which is a contradiction. Thus our lemma must be correct.

We can now prove the correctness of the algorithm using induction. With a base case of $A = 1$, the algorithm will return a set containing a single coin of value 1, which is trivially the optimal solution.

Now we assume that for all $A < u$, GreedyCoinExchange(c, k, A) returns an optimal solution, and show from there that GreedyCoinExchange(c, k, u) also does. Our algorithm returns a set containing the largest coin available to us which does not exceed u, as well as the optimal solution to the remaining value. And since according to our lemma the optimal solution must contain said coin, our solution must be optimal.