

Analysis of Algorithms

Homework 1

Abraham Murciano

November 4, 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 +  $w_i \leq W$  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.

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

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  $\{\text{a coin of value } c^a\} \cup \text{GREEDYCOINEXCHANGE}(c, k, A - c^a)$ 

```

To prove the correctness of this algorithm, we must show that the solution obtained by it is always optimal. We will do this in three stages.

1. Show that the optimal solution must contain less than c coins of the same value for all values less than c^k .
2. Prove that no other solutions satisfy the criterion of stage one.
3. Show that the solution provided by the algorithm satisfies the criterion, and thus, by stages one and two, must be the optimal solution.

To start off, we must understand that the optimal solution must contain less than c coins of the same value for all values less than c^k . This is easily demonstrated since if there were c coins of the same value $c^a < c^k$ in a solution, then we could replace the c coins of value c^a with one coin of value $c^{a+1} \leq c^k$, and we would end up with the same total value, thus showing that our solution was not optimal.

Now we must prove stage two. We must show that for all other solutions which are not optimal, there must be at least one coin denomination $c^a < c^k$ for which there are at least c coins.