Algorithms Homework 3

Aladdin Persson (aladdin.persson@hotmail.com)

1 Homework 3

1.1 Question 1

I don't understand the question as I don't see how we can maximize the usage of the drive and at the same time maximize the amount of stored files. We are then maximizing two different things and which one should we prioritze the most? I'll assume that we instead of maximizing total amount of files we wish to maximize the usage of the drive, even if this would only include a single file.

a) Let us consider the case of $g_1 = 1$, $g_1 = 3$, $g_3 = 5$ with size of disk is 5. The greedy would pick g_1, g_2 and store 2 files with a stored space of 4. The optimal solution is to only store f_3 with $g_3 = 5$ and store exactly that amount of space on the disk.

b) I think this is the knapsack problem where the weights equal the values. We want to maximize the sum of the weights which are the file sizes subject to the constraint that the sum of the file sizes should be less or equal than capacity.

The subproblems of this problem is that if we store the file with size g_i how much can we store with capacity $C - g_i$ and also how much can we store if we simply disregard this item. We can then form a matrix where we consider having capacities 0, 1, ..., C on one axis and on the other the possible files to store. Assume arrays are 0-indexed in the algorithm below, C total disk space, n total files, sizes array with the disk size for each file.

Algorithm 1 Knapsack(n, C, sizes)

```
Initialize matrix M with C+1 columns and n+1 rows with each element equal to 0
for i from 1 to n do
    for c from 1 to C do
        if size(i − 1) > c then M(i)(c) = M(i-1)(c)
        else:
            M(i)(c) = maxM(i-1)(c), sizes(i-1) + M(i-1)(c-sizes(i-1))
return M(n)(C)
```

c) The running time is forming the matrix of size nC which leads to $\mathcal{O}(nC)$ running time which is pseudo-polynomial.

1.2 Question 2

a) The greedy algorithm fails if we for example wish to return 13 in any currency and we have the coins $1, 6, 7, 10$. The greedy would take $10, 1, 1, 1$ with total amount of coins equal to 4, but the optimal would be $7 + 6 = 13$ and only 2 coins.

b) My first idea was that it also seems quite similar to the Knapsack problem but here it would instead be to minimize the sum of v_i subject to the constraint that the sum of weights should equal capacity C. The value for each coin is 1, with weight being what the coin is worth, and C the amount of money we are supposed to give. I didn't find a nice solution using this, because I couldn't figure out how to adapt the knapsack problem to an equality constraint.. Feedback on this is appreciated if it's a good idea/possible to do this.

My idea was to first consider the subproblems, assuming we stand with M money to give back and a set of the coins we are able to give. Assuming we know the optimal (least amount of coins to give) if we have $M - coins(i)$ money to give back for each coin in set of coins, then we can quite easily check each of the coins possible to give. Then simply take the one that gives the least amount coins to return. Drawing this graph for a simple case of for example $M = 11$, $coins = \{1, 2, 3\}$ then we quickly see that the subproblems will repeat themselves which gives us the idea that a dynamical approach might work well.

For initialization we can choose that of we have 0 money to return then the optimal is 0. For the rest we can initialize them as infinity and whenever the we obtain something less we update it.

```
minCoins(0) = 0
for m from 1 to money do
    Set mincoins(m) = ∞
    for i from 1 to n do
        if m ≥ coins(i) then
            numCoins = mincoins(m − coins(i)) + 1
            if numCoins < mincoins(m) then
                mincoins(m) = numCoins
return minCoins(money)
```

c) The running time should be the amount of money times the amount of coins. Since each operation in the if and within the for loops are constant operations. $\mathcal{O}(mn)$ if we let m be the amount of money and n amount of coins.