

COMP3711: Design and Analysis of Algorithms

Tutorial 8

HKUST

Question 1

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 yields 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, \dots, 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 .

- (a) Compute $c_q = \lfloor n/25 \rfloor$ which denotes the largest number of quarters to make change for n cents. Let $n_q = n - 25c_q$ to denote the remaining cents after make change by c_q quarters. Compute $c_d = \lfloor n_q/10 \rfloor$ which denotes the largest number of dimes to make change for n_q cents. Let $n_d = n_q - 10c_d$ to denote the remaining cents after make change by c_d dimes. Compute $c_n = \lfloor n_d/5 \rfloor$ which denotes the largest number of nickels to make change for n_d cents. Let $n_n = n_d - 5c_n$ to denote the remaining cents after make change by c_n nickels. Then we use $c_p = n_n$ pennies to make change the remaining cents.

Solution 1

Proof of optimality: Assume the greedy solution G is not optimal. There exist an optimal solution O uses o_q quarters, o_d dimes, o_n nickels and o_p pennies which is different to G . Obviously, we have $o_q \leq c_q$. If $o_q \neq c_q$, there are $25(c_q - o_q)$ cents are made change by dimes, nickels and pennies. Convert O to use $c_q - o_q$ quarters to make change of these $25(c_q - o_q)$ cents instead of dimes, nickels and pennies, the number of coins used in O will not increase. Then, we have $o_q = c_q$. If $o_d < c_d$, convert O to use $c_d - o_d$ dimes to make change of the $10(c_d - o_d)$ cents instead of nickels and pennies, the number of coins used in O will not increase. Now, $o_q = c_q$ and $o_d = c_d$. If $o_n < c_n$, convert O to use $c_n - o_n$ nickels to make change of the $5(c_n - o_n)$ cents instead of pennies, the number of coins used in O will not increase. Eventually, O is transformed to G and the number of coins used in O haven't been increased during the transformation. Therefore, G is optimal solution.

Solution 1

- (b) Let a_i denotes the number of c^i coin used in a solution for make change of n cents. Observe that, in the greedy solution, $a_i < c$ for $0 \leq i < k$, i.e. for any non-greedy solution, there exist at least one i such that $0 \leq i < k$ and $a_i \geq c$.

Assume there is a non-greedy solution O which is optimal.

The number of coins used is $\sum_{j=0}^k a_j$. Let i be the index such that $0 \leq i < k$ and $a_i \geq c$. Obviously, $1 \cdot c^{i+1} = c \cdot c^i$, this implies we can modify O to use one c^{i+1} coin instead of $c \cdot c^i$ coins to make change of c^{i+1} cents from the n cents, where $c > 1$. Then, the total number of coins used in O becomes $(1 - c) + \sum_{j=0}^k a_j < \sum_{j=0}^k a_j$ which contradicts with our assumption that O is a non-greedy optimal solution.

Therefore, greedy solution is optimal solution.

- (c) Let 1, 4, 6 be the set of coin denominations. Suppose we make change for $n = 8$ cents. The greedy solution uses one 6 cents coin and two 1 cent coins, i.e. it uses 3 coins. However, the optimal solution should use two 4 cents coins only.

Question 2

In the old days, files were stored on tapes rather than disks. Reading a file from tape isn't like reading a file from disk; first we have to fast-forward past all the other files, and that takes a significant amount of time. Suppose we have a set of n files that we want to store on a tape, where file i has length $L[i]$. Given the array $L[1..n]$, your job is to design an algorithm to find the optimal order to store these files on a tape to minimize the cost. Note that the cost of reading file i is total length of all files stored before it, including file i itself. Your algorithm should run in $O(n \log n)$ time.

Question 2

- (a) Suppose each file is accessed with equal probability, and you want to minimize the expected cost. For example, if $L[1] = 3, L[2] = 6, L[3] = 2$, you would want to use the order $(3, 1, 2)$. This way, the expected cost is $2/3 + (2 + 3)/3 + (2 + 3 + 6)/3 = 6$, which is optimal. You need to prove the optimality of your algorithm.
- (b) Suppose the files are not accessed uniformly; file i will have probability $p[i]$ to be accessed. Given the array $L[1..n]$ and $p[1..n]$, how would you find an ordering that minimizes the expected cost? For example, if $L[1] = 3, L[2] = 6, L[3] = 2$ and $p[1] = 1/6, p[2] = 1/2, p[3] = 1/3$, then the optimal ordering would be $(3, 2, 1)$ with an expected cost of $2/3 + (2 + 6)/2 + (2 + 6 + 3)/6 = 6.5$. Remember to prove the optimality of your algorithm.

- (a) Sort all files in the decreasing order of their length.

Proof of optimality: Consider any order and any two consecutive files i, j . If $L[i] > L[j]$, then we can swap their order. This swap will increase the cost of i by $L[j]$, but will decrease the cost of j by $L[i]$, so will decrease the expected cost.

- (b) Sort all files according to the ratio $L[i]/p[i]$.

Proof of optimality: Consider any order and any two consecutive files i, j . If $L[i]/p[i] > L[j]/p[j]$, then we swap their order. This swap will increase the cost of i by $L[j]$, but will decrease the cost of j by $L[i]$. The net increase of the expected cost is thus $L[j]p[i] - L[i]p[j] < 0$.