Tutorial 4: Greedy algorithms

1 Buying items from the SuperCheapStore

Suppose we would like to buy n items from the SuperCheapStore (SCS) where all items are currently priced at 1\$. Unfortunately, there is no delivery and we have to transport the items home. And to make matters worse:

- we can fit only one item in our truck; and
- it takes one day to drive home and back.

Worst of all, SCS charges us for the storage of undelivered items and the charge for storage of item i grows exponentially as the original price times a factor $c_i > 1$ each day. This means that if item i is picked up d days from now, the charge will be c_i^d dollars. In which order should we pick up our items from SCS so that total amount of charges is as small as possible?

Develop a greedy algorithm to solve this problem assuming that $c_i \neq c_j$ for $i \neq j$. Prove that your algorithm gives an optimal solution. What is the running time of your algorithm?

1.1 Solution

Sort the items in decreasing order of c_i. Pick them up in this order. We claim that the total cost will be minimized. Suppose (by re-indexing) that $c_1 > c_2 > \cdots > c_n$. Then we pick up item i on day i which costs c_i^{i-1}. Item c_1 is picked up on the first trip, so it costs $1 = c_1^0$. The total cost is $c_1^0 + c_2^1 + c_3^2 + \ + c_n^{n-1}$.

Consider any different solution S. We will show that its cost can be decreased. Since solution S is different there must be two items i and j where $c_i < c_j$ but we pick up item i before item j. Suppose we pick up item i after d days and item j after $d+k$ days. The cost of these two items in solution S is $c_i^d + c_j^{d+k}$.

Consider a new solution S' where we swap these two items. The cost of these two items in solution S' is $c_i^{d+k} + c_j^d$. The costs for all the other items remain the same. We want to prove that S' costs less, i.e. that

$$c_i^{d+k} + c_j^d < c_i^d + c_j^{d+k}.$$

Rearranging, we want to prove:

$$c_i^d(c_i^k - 1) < c_j^d(c_j^k - 1)$$

This is clear because $c_i < c_j$. Since we can decrease the cost of any solution different from the greedy one, therefore the greedy solution minimizes the cost.

The time complexity of this algorithm is $\Theta(n \log n)$ to sort.

2 Total completion time

Definition 1. An instance of the *minimal total completion time* problem is a sequence of n jobs that have processing times $p_1, p_2, \ldots, p_n$ which are positive integers. A valid solution to such an instance is an ordering of the jobs $1, \ldots, n$ such that when the jobs are processed one at a time in that order, the sum of their completion times is minimized.

Design a greedy algorithm for solving the minimal total completion time problem and prove that it is correct.

2.1 Solution

We can minimize the total completion time by sorting the jobs in increasing order of processing time and picking in that order. We will prove that this is optimal by showing that for any other ordering the cost could be further decreased.

Consider an order $l_1, l_2, ..., l_n$ that is not the increasing order. Since the order is not increasing there should exist an index i such that $l_i \geq l_{i+1}$. Now swap l_i and l_{i+1}. All the completion times for the jobs other than i and $i+1$ will be unchanged. Let C be the completion time of the job $i-1$. The cost corresponding to jobs i and $i+1$ before swapping is $(C+l_i)+(C+l_i+l_{i+1}) = 2C+2l_i+l_{i+1}$. The cost corresponding to jobs i and $i+1$ after swapping is $(C+l_{i+1})+(C+l_{i+1}+l_i) = 2C+2l_{i+1}+l_i$. Change in cost is $l_{i+1} - l_i < 0$, so the total cost decreased after the swap. Hence any order that is not increasing can not be an optimal solution.

The time complexity of the algorithm is $\Theta(n \log n)$ as we need to sort the processing times.