Albert-Ludwigs-Universität, Inst. für Informatik
Prof. Dr. Fabian Kuhn
H. Ghodselahi, Y. Maus

November 28, 2016

Algorithm Theory, Winter Term 2016/17
Problem Set 3 - Sample Solution

Exercise 1: Fractional Knapsack problem (16 points)

In the fractional Knapsack problem a thief robbing a store finds n items. The i-th item is worth v_i euro and weighs w_i kilograms, where v_i and w_i are integers. The thief wants to take a load which is as valuable as possible, but he can carry at most W kilograms in his knapsack, for some integer W. However, the thief can take fractions of items, rather than having to make a binary $(0\text{-}1)$ choice for each item.

Consider the following greedy algorithm to solve the fractional problem: we first compute the value per weight v_i/w_i for each item. The thief begins by taking as much as possible of the item with the greatest value per weight. If the supply of that item is exhausted and he can still carry more, he takes as much as possible of the item with the next greatest value per weight, and so forth, until he reaches his weight limit W. Hence, by sorting the items by value per weight, the greedy algorithm runs in $O(n \log n)$.

a) **(6 points)** Show that the aforementioned greedy algorithm computes an optimal solution for the fractional Knapsack problem.

b) **(10 points)** Now, the goal is to improve the runtime to solve the fractional Knapsack problem from $\mathcal{O}(n \log n)$ to $\mathcal{O}(n)$.

The *median of medians* algorithm[1] returns the median of a given array in time $\mathcal{O}(n)$. Use the *median of medians* algorithm as a black box to devise an algorithm that returns an optimal solution for the fractional Knapsack problem with linear running time (in n).

Solution

a) Items sorted by value per weight amounts: $v_1/w_1 \geq v_2/w_2 \geq \ldots \geq v_n/w_n$. Let the greedy solution be $S =< x_1, x_2, \ldots, x_k, \ldots, x_n >$ where x_i indicates the fraction of item i that has been taken. Note that all $x_i = 1$ for $i < k$, except possibly for the last item, i.e., $i = k$. For all $i > k$ we have $x_i = 0$. Consider any optimal solution $S^* =< y_1, y_2, \ldots, y_n >$ where y_i denotes the fraction of item i taken in S^* where for all i we have $0 \leq y_i \leq 1$. The thief's knapsack must be full in both greedy and optimal solutions, i.e.,

$$\sum_{i=1}^{k} x_i w_i = \sum_{i=1}^{n} y_i w_i = W.$$

Consider the first item i where the two solutions differ. With respect to the greedy algorithm description, it takes a greater amount of item i than the optimal algorithm (because the greedy solution always takes as much as it can). Let $\delta = x_i - y_i$. Consider the following new solution $S' =< z_1, z_2, \ldots, z_n >$ constructed

[1] If you are interested in the *median of the medians* algorithm you can find more information about it online. However, as mentioned, you can use it as a blackbox in this exercise.

from S^* (note that at the beginning $z_i := y_i$ for all $i \in \{1, \ldots, n\}$): for $j < i$, keep $z_j = y_j$ and let $z_i = x_i$. Further, in S', remove items of total weight $\delta \cdot w_i$ from items $i+1$ to n. This is always doable because

$$\sum_{j=i+1}^{n} y_j \cdot w_j \geq \delta \cdot w_i$$

since

$$\sum_{j=i}^{k} x_j \cdot w_j = \sum_{j=i}^{n} y_j \cdot w_j.$$

The total value of solution S', therefore, is not now smaller than the total value of solution S^* because

$$v_i/w_i \geq v_j/w_j$$

for all $j \in \{i+1, \ldots, n\}$. Since S^* is largest possible solution and value of S' cannot be smaller than that, the total values of S^* and S' must be equal. Thus the solution S' is also optimal. By repeating this process, we will eventually convert S^* into S, without changing the total value of the optimal solution. Therefore S is also optimal.

b) As it is mentioned in the question, we use the linear-time *median of medians* algorithm as a black box to calculate the median m of the v_i/w_i ratios. Next, partition the items into three sets: $G = \{i : v_i/w_i > m\}$, $E = \{i : v_i/w_i = m\}$, and $L = \{i : v_i/w_i < m\}$ that takes linear time. Note that the sizes of G and L are both less than $n/2$. Let

$$W_G = \sum_{i \in G} w_i$$

and

$$W_E = \sum_{i \in E} w_i$$

denote the total weights of the items in sets G and E, respectively.

- If $W_G > W$, then the original problem with n items and knapsack capacity W reduced to an instance of the problem with only items in G and knapsack capacity W such that any solution for the new instance is indeed a solution for the original problem.
- Otherwise we have $W_G \leq W$. In this case take all items in set G, and take as much of the items in set E as will fit in the remaining capacity $W - W_G$.
  - If $W_G + W_E \geq W$ (i.e., there is no capacity left after taking all the items in set G and all the items in set E that fit in the remaining capacity $W - W_G$), then we are done.
  - Otherwise we have $W_G + W_E < W$. Then after taking all the items in sets G and E, recurse on the set of items L and knapsack capacity $W - W_G - W_E$.

To analyze this algorithm, note that each recursive call takes linear time, exclusive of the time for a recursive call that it may make. When there is a recursive call, there is just one, and it is for a problem of at most half the size. Thus, the running time is given by the recurrence $T(n) \leq T(n/2) + \mathcal{O}(n)$ whose solution is $T(n) = \mathcal{O}(n)$.

Exercise 2: Matroids (14 points)

We have defined matroids in the lecture. For a matroid (E, I), a maximal independent set $S \in I$ is an independent set that cannot be extended. Thus, for every element $e \in E \setminus S$, the set $S \cup \{e\} \notin I$.

a) **(4 points)** Show that all maximal independent sets of a matroid (E, I) have the same size. (This size is called the rank of a matroid.)

b) **(5 points)** Consider the following greedy algorithm: the algorithm starts with an empty independent set $S = \emptyset$. Then, in each step the algorithm extends S by the minimum weight element $e \in E \setminus S$ such that $S \cup e \in I$, until S is a maximal independent set. Show that the algorithm computes a maximal independent set of minimum weight.

c) **(5 points)** Let E be any finite subset of the natural numbers $\mathbb{N}$ and $k \in \mathbb{N}$ be any natural number. Define a collection of sets $I := \{X \subseteq E : \forall x \neq y \in X,\ x \not\equiv y \pmod{k}\}$. Show that (E, I) is a matroid.

Solution

a) Assume that there are two maximal independent sets S and T with $|S| \neq |T|$. Without loss of generality (w.l.o.g.) we assume that $|S| < |T|$. The exchange property tells us that there exists an element $x \in T \setminus S$ which can be added to S while $S \cup \{x\}$ is still independent, which is a contradiction to the maximality of S.

b) Let $S = (e_1, \ldots, e_r)$ be the greedy solution and $T = (f_1, \ldots, f_r)$ be any other solution, where r is the rank of the matroid (note that both S and T need to have cardinality r to be maximal).

For the sake of contradiction assume that $w(T) < w(S)$, where $w(T) = \sum_{i=1}^{r} w(f_i)$ and the same for S. We also assume that e_i and f_i are already ordered by increasing weight. We now let k to be the smallest index such that $w(f_k) < w(e_k)$ (note that for smaller indices equality may not hold).

Consider the sets $S_{k-1} := \{e_1, \ldots, e_{k-1}\}$ and $T_k := \{f_1, \ldots, f_k\}$. Using the exchange property we get that there is a $j \in [k]$ such that $S_{k-1} \cup \{f_j\}$ is independent. Since T is ordered, we know that $w(f_j) \leq w(f_k) < w(e_k)$. It means that in step k the greedy algorithm skipped element f_j even though f_j has less weight than e_k and it would not have violated the independence of S_{k-1}. It contradicts the algorithm description.

c) To prove that (E, I) is a matroid we need to investigate the three matroid properties:

1) **Empty set is independent:** This property is trivially satisfied since there does not exist any pair of elements in the empty set to be equal modulo k.
2) **Hereditary property:** Let us assume that $A \in I$. Then there does not exist any pair of elements a and b in A such that $a \equiv b \pmod{k}$. It is clear that by removing any element from A, still, there does not exist any pair of elements with the mentioned property.
3) **Augmentation property:** First we need to notice that for any independent set $X \in I$ we have $|X| \leq k$, because for any integer k there exist k different residue classes of integers mod k.

   Now, let us assume that there exist two independent sets $A, B \in I$ such that $|B| < |A| \leq k$. We define A' to be the set of residue classes (mod k) of all integers in A, and B' is the set of residue classes (mod k) of all integers in B. Let $C' = A' \setminus B'$, which is not empty. Then any integer in A which belongs to one of the residue classes in C' can be added to B for its extension.

Exercise 3: Road Trip Planning (10 points)

You want to plan your next road trip with your girlfriend/boyfriend. Because your girlfriend/boyfriend is very picky you can only stay at very nice hotels. Along the way, there are n very nice hotels, at kilometer posts $a_1 < a_2 < \ldots < a_n$, where each a_i is measured from the starting point. However, you do not need to stop at all hotels but you need to stop at a_n, which is your final destination.

To actually be able to do some sightseeing on the way and to reach your destination on time you would like to travel approximately 200 kilometers a day. Due to the distances between the hotels this is not always possible and if you drive x kilometers in one day the potential for being unsatisfied is $(200 - x)^2$.

You want to plan your trip such that the total potential for being unsatisfied is minimized, i.e., the sum over all travel days, of the daily potentials.

Devise an efficient algorithm to solve the problem. Besides a description of your algorithm provide pseudocode for your algorithm. Do not forget to state and explain its runtime.

Solution

First let us look at a recursive formula for finding a solution to this. For any stop a_i there is an optimal route to get there which is the minimal potential for being unsatisfied of all routes from a_0 to a_i. This is going to result from either taking the optimal route to a_{i-1} then going to a_i, or taking the optimal route to a_{i-2} then going to a_i, and so on, until looking at going straight from 0 to a_i. Therefore, let $P(i)$ denote the minimal potential of being unsatisfied if the final destination will be a_i. Thus, S_i can be recursively defined as follows:

$$S_i = \min_{j<i}\{S_j + \big(200 - (a_i - a_j)\big)^2\}$$

where $S_0 = 0$. The pseudocode of the algorithm is provided as follows:

Algorithm 1: determine a set of stops that minimize the total potential of being unsatisfied

Input: an array $a[]$ of all possible stops
$S[]$ /* stores the S_i values. */

$R[]$ /* stores the previous stop that yields the optimal route. */

$S[0] \leftarrow 0$
$R[0] \leftarrow 0$
for $i \leftarrow 1$ ***to*** n **do**
  $previousStop = 0$
  $minPenalty = \infty$
  for $j \leftarrow 0$ ***to*** i **do**
    if $S[j] + \big(200 - (a[i] - a[j])\big)^2 <$ *minPenalty* **then**
      $minPenalty = S[j] + \big(200 - (a[i] - a[j])\big)^2$
      $previousStop = j$
    end
  end
  $S[i] = minPenalty$
  $R[i] = previousStop$
end
return $S[]$ *and* $R[]$

As for the time complexity of this algorithm we can merely count the number of comparisons. The outer for loop iterates n times, from 1 to n. The inner loop iterates i times, so we will have a total number of comparisons equal to $1 + 2 + 3 + ... + n$, which equals $n(n + 1)/2$. This is $\Theta(n^2)$.