University of Freiburg
Dept. of Computer Science
Prof. Dr. F. Kuhn
P. Bamberger, P. Schneider

Algorithm Theory
Sample Solution Exercise Sheet 3

Due: Tuesday, 9th of November, 2021, 4 pm

Exercise 1: Scheduling *(8 Points)*

Given n jobs of lengths $t_1 \dots, t_n$ with *one deadline* $d \geq 0$, we want to schedule these jobs such that the *average lateness* is minimized. That is, for each job i we want to find a start time and finishing time $0 \leq s(i) \leq f(i)$ with $f(i) - s(i) = t_i$ such that the intervals $[s(i), f(i)]$ are pairwise non-overlapping (overlapping start- and endpoints are allowed) and the average over all $L(i) = \max\{0, f(i) - d\}$ is minimal.

(a) Describe a greedy algorithm for this problem. *(3 Points)*

(b) Prove that it computes an optimal solution. *(5 Points)*

Sample Solution

(a) We schedule the jobs by length t_i, starting with the shortest and ending with the longest. That is, we first sort the jobs by length and then set $s(1) = 0, f(1) = t_1$ and for all $i \geq 2 : s(i) = f(i-1)$, $f(i) = s(i) + t_i$. This minimizes the sum of all latenesses (and hence the average lateness).

(b) We prove this with an exchange argument. Let O be an optimal solution. We transfer O to a greedy solution without increasing the total lateness (if the job lengths are not pairwise distinct, there are different greedy solutions). To simplify presentation, assume that each job is represented by an integer such that $O = (1, \dots, n)$. If O is not a greedy solution, there must be jobs i and $i+1$ with $t_i > t_{i+1}$. We exchange jobs i and $i+1$ and compare the old and new finishing times of all jobs:

$$f_{new}(i) = \sum_{j<i} t_j + t_{i+1} + t_i = \sum_{j \leq i+1} t_j = f_{old}(i+1)$$

and

$$f_{new}(i+1) = \sum_{j<i} t_j + t_{i+1} < \sum_{j<i} t_j + t_i = f_{old}(i)$$

For the latenesses it follows $L_{new}(i) = L_{old}(i+1)$ and $L_{new}(i+1) \leq L_{old}(i)$. The finishing time and thus the lateness of all other jobs do not change. We obtain

$$\begin{aligned}\sum_{j=1}^{n} L_{new}(j) &= \sum_{j=1}^{i-1} L_{new}(j) + L_{new}(i) + L_{new}(i+1) + \sum_{j=i+2}^{n} L_{new}(j) \\ &\leq \sum_{j=1}^{i-1} L_{old}(j) + L_{old}(i+1) + L_{old}(i) + \sum_{j=i+2}^{n} L_{old}(j) = \sum_{j=1}^{n} L_{old}(j)\end{aligned}$$

So we have seen that exchanging jobs i and $i+1$ did not increase the sum of all latenesses and thus the average lateness did not increase. We proceed this way until the jobs are sorted by length, i.e., we obtain a greedy solution. It follows inductively that the average lateness of this solution is not larger than the one of O and therefore the greedy solution is optimal.

Exercise 2: Prefix Codes *(12 Points)*

Imagine you have n characters $c_1, \dots, c_n$ and each has a frequency $f_1, \dots, f_n$ (w.l.o.g. sorted ascending) with which it occurs in a text. The goal is to compute a code over $\{0, 1\}$ for each character (i.e., assign a unique bit sequence to each character) which is prefix-free, i.e., no codeword is a prefix of another.

Such a *prefix code* can be obtained by constructing a full binary tree[1]: Use the characters $c_1, \dots, c_n$ as leaves, assign 0 and 1 to all edges, such that internal nodes have a child with a 0-edge *and* a child with a 1-edge. The code of c_i is then given by the bits on the path from the root to the leaf c_i.

The goal is to minimize the total length of a message with the given frequency of symbols, i.e. $\sum_{i=1}^{n} f_i \cdot \ell_i$, where ℓ_i is the length of the codeword of c_i. Analogously, we want to find a full binary tree that minimizes $\sum_{i=1}^{n} f_i \cdot d_i$, where d_i is the (unweighted) length of the path from root to c_i (depth).

Such a tree can be constructed with a greedy method: Start with $c_1, \dots, c_n$ as leaves (w.l.o.g. sorted by frequency). Add an internal node and make the two least frequent characters c_1, c_2 its children (break ties arbitrarily). The internal node becomes a new character c_{n+1} with frequency $f_{n+1} = f_1 + f_2$. Then "remove" the leaves c_1, c_2 and recurse on the characters $c_3, \dots, c_{n+1}$ (i.e., treat c_{n+1} as new leaf). We call the resulting tree the *greedy tree* and the resulting prefix-code for $c_1, \dots, c_n$ the *greedy code*.

(a) Construct the greedy tree and greedy code for $n = 6$ characters with frequency $f_i = i$. *(3 Points)*

Remark: for more consistent solutions, assign 0 the left-child edges and 1 to right-child edges.

(b) Show that there is an *optimal* full binary tree T with leaves $c_1, \dots, c_n$ (i.e., that minimizes $\sum_{i=1}^{n} f_i \cdot d_i$), in which the two least frequent elements c_1, c_2 are siblings. *(5 Points)*

Hint: Show that for two siblings c_j, c_k which are at largest depth in some full binary tree it does not make $\sum_{i=1}^{n} f_i \cdot d_i$ larger if we swap c_j with c_1 and c_k with c_2.

(c) Give an inductive argument that the greedy code is an optimal prefix code. *(4 Points)*

Sample Solution

(a) *Remark: You may have recognized that the greedy algorithm corresponds to the procedure to compute* Huffman-codes *(proposed by David A. Huffman).*

By recursively merging leaves we obtain the following "greedy-tree" (Huffman tree).

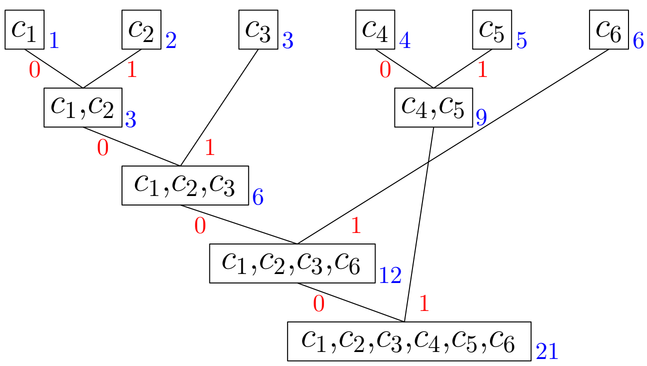

Figure 1: "Greedy"-tree (Huffman-tree) with internal nodes containing merged characters. Frequency of nodes to the bottom left (blue).

The resulting *Huffman code* is

[1] In a full binary tree each node has 0 or 2 children.

char	code
c_1	0000
c_2	0001
c_3	001
c_4	10
c_5	11
c_6	01

(b) For a full binary tree T' with leaves $c_1, \dots, c_n$, define the *weight* of T' as $w(T') = \sum_{i=1}^{n} f_i \cdot d_i$.

Let T' be a full binary tree with leaves $c_1, \dots, c_n$ where c_1, c_2 are *not* siblings and *optimal* weight $w(T') = \sum_{i=1}^{n} f_i \cdot d_i$. Let c_j, c_k be two siblings at largest depth $d' := d_j, d_k$ in T'. We essentially show how to swap c_j or c_k with a node of smaller frequency thereby making the weight of the resulting tree not larger. This means that we can swap c_j with c_1 and c_k with c_2 since $f_1 \leq f_j$ and $f_2 \leq f_k$ (by definition).

W.l.o.g., we focus on swapping c_j with c_1. Let $\delta := d_j - d_1 \geq 0$ be the difference in depth of the two leaves. Consider the tree T, where we swap c_1 with c_j. In particular, swapping the leaves means that our new weight is $w(T) = f_1 d_j + f_j d_1 + \sum_{i \neq 1,j} f_i \cdot d_i$. We have

$$
\begin{aligned}
w(T') = \sum_{i=1}^{n} f_i \cdot d_i &= f_1 d_1 + f_j d_j + \sum_{i \neq 1,j} f_i \cdot d_i \\
&= f_1(d_j - \delta) + f_j(d_1 + \delta) + \sum_{i \neq 1,j} f_i \cdot d_i \\
&= f_1 d_j + f_j d_1 + \delta(f_i - f_1) + \sum_{i \neq 1,j} f_i \cdot d_i \\
&= w(T) + \underbrace{\delta(f_i - f_1)}_{\geq 0}
\end{aligned}
$$

This implies $w(T) \leq w(T')$. As c_2 and c_k fulfill the same requirements which we used here, they can be swapped analogously, without increasing the weight of the resulting tree T. Since we assumed T' was already optimal we have $w(T) = w(T')$, i.e., T is also optimal.

(c) Since an optimal prefix code (that minimizes $\sum_{i=1}^{n} f_i \cdot \ell_i$, where ℓ_i is the length of the code of c_i) must be constructable from a full binary tree with leaves $c_1, \dots, c_n$ and optimal weight (we give that argument further below) we only have to show that our greedy tree has optimal weight $w(T) = \sum_{i=1}^{n} f_i \cdot d_i$.

Let T_n be the greedy tree fore $c_1, \dots, c_n$. In base case T_2 we have just 2 characters and our greedy tree has just two leaves and a root, which is clearly optimal. Let now $n \in \mathbb{N}$. Our hypothesis is that the claim is true for $n-1$ (or less) characters, more specifically we hypothesize that $w(T_{n-1})$ is optimal for any greedy tree T_{n-1} with $n-1$ characters. Now we want to show that $w(T_n)$, i.e. the weight of the greedy tree for $c_1, \dots, c_n$ is optimal as well.

For comparison, let T be an *optimal* full binary tree for $c_1, \dots, c_n$. Further, we assume that c_1, c_2 are siblings in T, which is *not* a restriction in terms of the minimal weight that can be achieved as we showed in part (b). Let d_i be the depth of c_i in T. Let T' be the subtree of $c_3, \dots, c_{n+1}$, where c_{n+1} is the parent of c_1, c_2. Let $f_{n+1} = f_1 + f_2$ and let $d_{n+1} = d_1 - 1$ be the depth of c_{n+1} in T'. Then we have

$$
\begin{aligned}
w(T) = \textstyle\sum_{i=1}^{n} f_i \cdot d_i &= \left(\textstyle\sum_{i=3}^{n+1} f_i \cdot d_i\right) + f_1 \cdot d_1 + f_2 \cdot d_2 - f_{n+1} \cdot d_{n+1} \\
&= w(T') + f_1 \cdot d_1 + f_2 \cdot d_2 - f_{n+1} \cdot d_{n+1}. \\
&= w(T') + (f_1 + f_2) \cdot d_1 - (f_1 + f_2) \cdot (d_1 - 1) = w(T') + f_1 + f_2.
\end{aligned}
$$

Since $w(T)$ is optimal, it is *necessary* that the weight $w(T')$ of the subtree T' is optimal as well (otherwise $w(T)$ could be larger by making $w(T')$ larger). It is also sufficient that $w(T')$ is optimal so that $w(T)$ is optimal, since $w(T) = w(T') + f_1 + f_2$ depends only on $w(T')$ whereas f_1, f_2 do not depend on T. So $w(T)$ is optimal if and only if $w(T')$ is optimal.

We already know that $w(T_{n-1})$ is an optimal solution for $c_3, \ldots, c_{n+1}$ from the hypothesis, thus $w(T) = w(T_{n-1}) + (f_1 + f_2)$. Note that in the same way as above, by construction of T_n, we also have $w(T_n) = w(T_{n-1}) + (f_1 + f_2)$, thus $w(T_n) = w(T)$.

Proof that an optimal prefix code can be constructed from a full binary tree with n leaves (not required): Every prefix code can be considered as a binary tree with leaves $c_1, \ldots, c_n$. Let us prove this first. Let c_i be the character with the longest codeword of length d. Consider the full and complete binary tree of depth d.

The codeword of some character c_j represents a unique node in that tree: start from the root, go left for a 0, right for 1, the node you end up corresponds to c_j. Then remove all internal nodes which do not have a descendant that corresponds to some c_j. After removal, all leaves must correspond to some character. Moreover, all nodes corresponding to some c_j must be leaves, since the code is prefix-free.

The optimal prefix code must correspond to a full binary tree. This is because if we have an internal node with only a single child, we can merge it with its child and thus make the codewords of all descendant leaves of this node by one bit shorter.