Cheque sorter case study

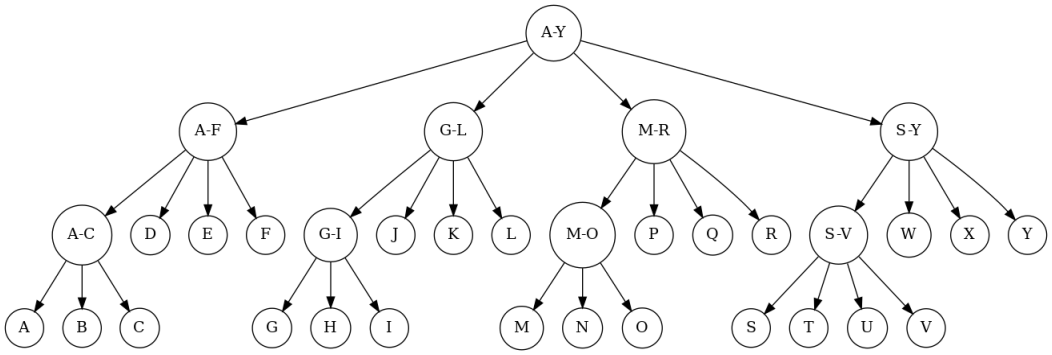
Valentin Bartier, Lucas Aresi, Othman Touijer, Maxime Challier

January 2018

Contents

1 Questions

Question 1: The graph that corresponds to the example is the following:



This graph is not unique. For instance, in step 2 we could decide to settle the bank A,B and C and put the bank D,E and F in the same stack and still respect the sorting method given as an example. As the number of cheques are different this would change the value of the solution. In our graph we program the sorter 9 times ($9p$ minutes). Then for each stack we have to count the number of cheques to sort and divide it by v the speed of the sorter). All the cheques are sorted in 49713.3 seconds (about 828 minutes or 14 hours.)

Question 2: We show that given a feasible solution of the cheques sorter problem we can build a graph that respects the given conditions and vice-versa.
Given a complete solution we build the following graph: let r be the root corresponding to the stack containing all the cheques. Then we add to each node that correspond to a stack S containing the cheques of more than 1 bank a number of successors corresponding to the number of stack that S is planned to be divided in. Hence we have a graph we exactly one node without predecessor, where each node has at most s successors and the exact set of node having no successors correspond to the stack of cheques containing only the cheques of one bank.
Reversely let consider a graph that respects the given conditions. The nodes having only leaves as successors gives last programming of the sorter. We can delete these leaves and obtain a graph that also respect the conditions and repeat the process until we have only one node left. The programming sequence we obtain results in having exactly N final stack with each stack being divided in at most s new stack as each node of the graph has at most s successors and hence is a feasible solution.

Question 3: The value of a solution depends on the programming time and the sorting time. Each internal nodes correspond to a stack containing more than one bank and hence correspond to a programming of the sorter. Hence the total programming time is equal to $p|I|$. The sorting time is the total number of cheques we sort during the whole process divided by the sorting speed. Let k_i be the number of times cheques from bank i are sorted. Then the total number of cheques sorted is:

$$\sum_{i=1}^{N} n_i k_i$$

Let $i \in F$ be a final vertices. It corresponds to a stack containing only cheques from bank i. As its height is h_i the cheques from bank i have been in $h_i - 1$ stacks (without counting the initial stack) before being totally sorted. Then it has been sorted one last time to obtain the leaf i. Hence for $i \in F$ we have $k_i = h_i$. Hence the value of the solution is:

$$p|I| + \frac{1}{v}\sum n_i h_i$$

Question 4: Let M be a feasible solution of the problem where we allowed to merge stacks. Suppose that at some point we decided to merge stacks s_i and s_j into a stack s_{ij}. If one or the two stacks correspond to settled banks then not doing the merge clearly gives a feasible solution that is as good as M and we can suppose that the two stacks each contains more than 1 bank and are internal nodes of M.
Let suppose without loss of generality that $h_i \leq h_j$ and let r_{ij} be the last common ancestors of s_i and s_j. We build a solution S as follows: S is the same graph as M except for the sub-graph rooted in r_{ij}: we put all the cheques of s_j in the stack containing s_i and sort the other cheques of r_{ij} as in M. We obtain a successor of r_{ij} containing s_i, s_j and eventually other cheques (that were stacked with only s_i in M). We sort this stack as we did in M keeping always the cheques of s_i and s_j in the same stack. Ultimately we obtain a stack containing only the cheques of s_i and s_j at an height $h_M(s_i)$. We can sort this stack as we did in M and obtain a solution with final height of leaves lesser or equals than the ones in M as $h_M(s_i) \leq h_M(s_{ij})$.
Hence we can find a solution as least as good as M were we do not merge the two stacks. It follows that the set of solutions such that no stacks are merged is dominant.

Question 5: Let consider a graphical solution of the cheques sorting problem. We add nodes to the graph in the following way: to each nodes having at least 1 successor and $k < s$ successors we add $s - k$ successors that correspond to settled dummy banks with 0 cheques. The graph obtained is a $s - tree$ and correspond to a solution of the same value than the previous graph:

- we only add leaves, hence the programming time of the sorter does not change.
- let I' be the set of dummy banks we add. All the dummy banks are leaves and we do not modify the height of leaves corresponding to real banks hence the sorting time of the cheques is:

  $$\sum_{F \cup I'} n_i h_i = \sum_F n_i h_i + \sum_{I'} n_i h_i = \sum_F n_i h_i$$

  as for all $i \in I'$, $n_i = 0$.

The constructed solution has the same value as the one we considered first. Hence for every graphical solution of the problem, there exists a solution that is a s-tree that and has the same value.

2 Huffman's algorithm

In order to solve the cheque sorter problem using Huffman's algorithm we use the notion of nodes: a node is an object defined by its weight (number of cheques), its id (the number of the place its appears in in the given instance file and which is only use to output the final heights in the correct order), and s or less sons. We the use the following algorithm:

Initialization: Create N nodes with weight n_i, no sons, and set their id according to the order of appearance in the instance file.
Stopping condition: If there are s nodes or less create the root r having sons the remaining nodes and for weight the sum of their weight. (If their are strictly less that s nodes remaining add dummy bank with weight 0 to complete). Set its id to -1. (-1 for id indicate that a node is not a leaf and thus does not correspond to a sorted bank). Stop and output r.
Iterations: Else their remains strictly more than s nodes. Replace the s nodes of minimum weight with a new node having these s nodes as sons and for weight the sum of their weight. Set its id to -1. Go to the stopping condition.

2.1 Proof of optimality

Let $B = \{n_1, n_2, ..., n_N\}$ be the set of bank weights sorted in increasing order. We will prove the optimality by induction on the number N of banks.
Induction hypothesis: For any instance with $N' < N$ banks (leaves), the Huffman algorithm output an optimal s-tree.
Initialization: If $N < s$ the tree is optimal: all banks are grouped into one unique nodes (the stopping condition is true) meaning that all the banks are sorted in one step.
Induction: Before starting the induction part of the proof, let us remark the following property:

For any set of banks $B = \{n_1, n_2, ..., n_N\}$ there exist an optimal s-tree in which the s leaves with the smallest weights have the highest height and are siblings (have the same parent node).

Proof. Due to question 9, in any optimal tree we know that the s leaves with the smallest weights have an height above or equal to the height of the other leaves. Now let consider the node v parent of n_1. Then either the siblings of n_1 are exactly $n_2...n_s$ and we are done, either n_1 has a sibling n_i with $i > s$ and there exists $j \in 1..s$ such that the parent of node n_j is not v. Furthermore, we have $h_1 \leq h_j \leq h_i = h_1$ due to question 9. Hence we can swap the nodes n_i and n_j and obtain a tree with same value in which n_j is a sibling of n_1. By repeating this process we can obtain an optimal s-tree in which $n_1, ..., n_s$ are siblings.

Let now be T_H the tree output by the Huffman algorithm for the set of banks $B = \{n_1, n_2, ..., n_N\}$ with $N > s$. In the first step, the algorithm will create a new node v of weight $\sum_{i=1}^{s} n_i$ having for sons the nodes $n_1, n_2, ..., n_s$ and continue with the set of weight $B' = \{\sum_{i=1}^{s} n_i, n_{s+1}, ..., n_N\}$ (note that the node v may not be the node of smallest weight here). Let $T_{H'}$ be the tree output by the algorithm on the set B'. By induction hypothesis, $T_{H'}$ is optimal. Furthermore, T_H can is obtained from $T_{H'}$ by adding s sons to the node v with weights $n_1, n_2, ...n_s$. Let $val(T) = \sum n_i h_i$ the value of a solution tree.

We have:

$$val(T_H) = val(T_{H'}) + (\sum_{i=1}^{s} n_i)(h_{T_{H'}}(v) + 1) - (\sum_{i=1}^{s} n_i)h_{T_{H'}}(v)$$

Indeed to obtain T_H we had s sons to v which becomes leaves and v is not a leaf anymore. Hence:

$$val(T_H) = val(T_{H'}) + \sum_{i=1}^{s} n_i$$

Let now T_O be an optimal s-tree for the set of banks B in which $n_1, ..., n_s$ are siblings. If we replace the parents of these nodes by a leaf of weight $\sum_{i=1}^{s} n_i$ we obtain (by reasoning in the same way we did for T_H and $T_{H'}$) a tree $T_{O'}$ on the set of weight B' such that:

$$val(T_O) = val(T_{O'}) + \sum_{i=1}^{s} n_i$$

Furthermore, as $T_{H'}$ is optimal by induction hypothesis, we know that $val(T_{H'}) \leq val(T_{O'})$. Hence:

$$\begin{aligned} val(T_H) &= val(T_{H'}) + \sum_{i=1}^{s} n_i \\ &\leq val(T_{O'}) + \sum_{i=1}^{s} n_i \\ &\leq val(T_O) \end{aligned}$$

As T_O is optimal, we have $val(T_H) = val(T_O)$ and T_H is optimal. #