Decode-efficient prefix codes for hierarchical memory models

Shashwat Banchhor [*] Rishikesh Reddy [*] Yogish Sabharwal [*‡] Sandeep Sen [*†]
[*] Indian Institute of Technology, New Delhi [†] Shiv Nadar University, U.P. [‡] IBM Research, New Delhi

Model

Introduction. Our scratchpad model is similar to the two-level hierarchical model proposed in [1] comprising of a limited size fast memory (scratchpad memory) and an unlimited size main memory. The cost of accessing a location in the scratchpad and main memory is 1 unit and q units respectively. Decoding the input is typically done by traversing the stored prefix tree. We consider the class of algorithms that store nodes of the prefix tree in the scratchpad – one prefix tree node in one scratchpad addressable memory location.

Problem

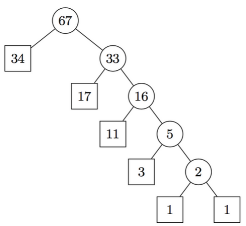
Consider an alphabet C. For each character c in C, let f(c) denote the frequency of c. Given a prefix tree T corresponding to a prefix code P for C, let d(c) denote the depth of the leaf corresponding to the encoding of c in the tree T . The average code length of the encoding is given by $\ell(T) = \Sigma_{c \in C} f(c) \cdot d(c)$. Given a constant P m (scratchpad size), we define the decode time of the encoding to be

$$decodeTime(T, m) = \Sigma_{c \in C} f(c) + q \cdot \Sigma_{c \in C: d(c) > \log(m)} f(c) \cdot (d(c) - \log(m)).$$

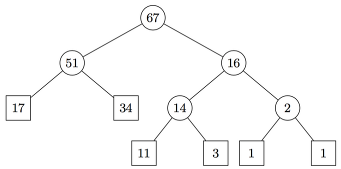
Given constants m and L, our goal is to find a prefix tree, T , that minimizes decodeTime(T, m) subject to $\ell(T) \leq L$.

Illustration

<u>Code Length</u>:
5·1+5·1+4·3+3·11+2·17+1·34 = **123**
<u>Decode Time</u>:
(1+2q)·1+(1+2q)·1+(1+q)·3+1·11+1·17+1·34 = **67+7q**



<u>Code Length</u>:
3·1+3·1+3·3+3·11+2·17+2·34 = **150**
<u>Decode Time</u>:
1·1+1·1+1·3+1·11+1·17+1·34 = **67**



Approach

We present an efficient algorithm that solves the above mentioned problem optimally for a given alphabet C, a threshold codelength parameter *L* and a scratchpad size parameter *m*.

- This is based on a property of the forest outside the scratch pad. We call these as a Huffman Forest which is an intermediate step in the construction of optimal tree using Huffman algorithm.
- We solve for the position of nodes which remain in the scratchpad using a Dynamic programming algorithm: *FindTop*.

We present an efficient algorithm that solves the above mentioned problem optimally for a given alphabet C, a threshold parameter L and a scratchpad size parameter m. The running time of the algorithm is polynomial in the size of the fast memory (poly(m)) and near linear in the size of the alphabet (|C|log|C|).

Input: Alphabet $C = \{c_1, c_2, \ldots, c_n\}$; m (addressable size of fast memory); L (code length threshold)
Output: Optimal Tree in DecodeTime satisfying codelength constraint: ***Best***

$Best \leftarrow$ `invalid`
for $k \leftarrow m$ ***downto*** 1 **do**
  Huffman forest F_k (k trees) $\leftarrow n-k$ iterations of Huffman algorithm on C
  T_{top} = `FindTop`(Trees with only one character in F_k and their frequencies, k)
  Merge T_{top} and F_k to obtain tree $Curr$
  if $Length_{Curr} \leq \min(L, Length_{Best})$ **then**
    $Decode_{Curr} \leftarrow$ DecodeTime of $Curr$
    if $Decode_{Curr} < Decode_{Best}$ **then**
      $Best = Curr$
return ***Best***

References

[1] Sandeep Sen, Siddhartha Chatterjee, and Neeraj Dumir, "Towards a theory of cache-efficient algorithms," J. ACM, vol. 49, no. 6, pp. 828–858, 2002.