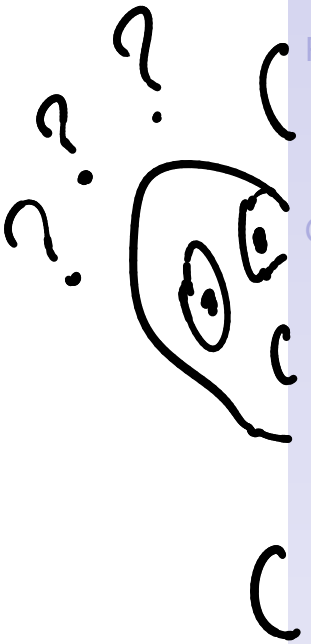
(Faster Computation of)

Optimal Prefix Free Codes with Partial Sorting

Jérémy Barbay

jeremy@barbay.cl

[2016-06-23 Thu]@Ljubljana

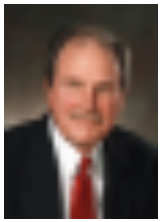


Full version available at http://arxiv.org/abs/1602.03934.

Outline

Optimal Prefix Free Codes (OPFC)

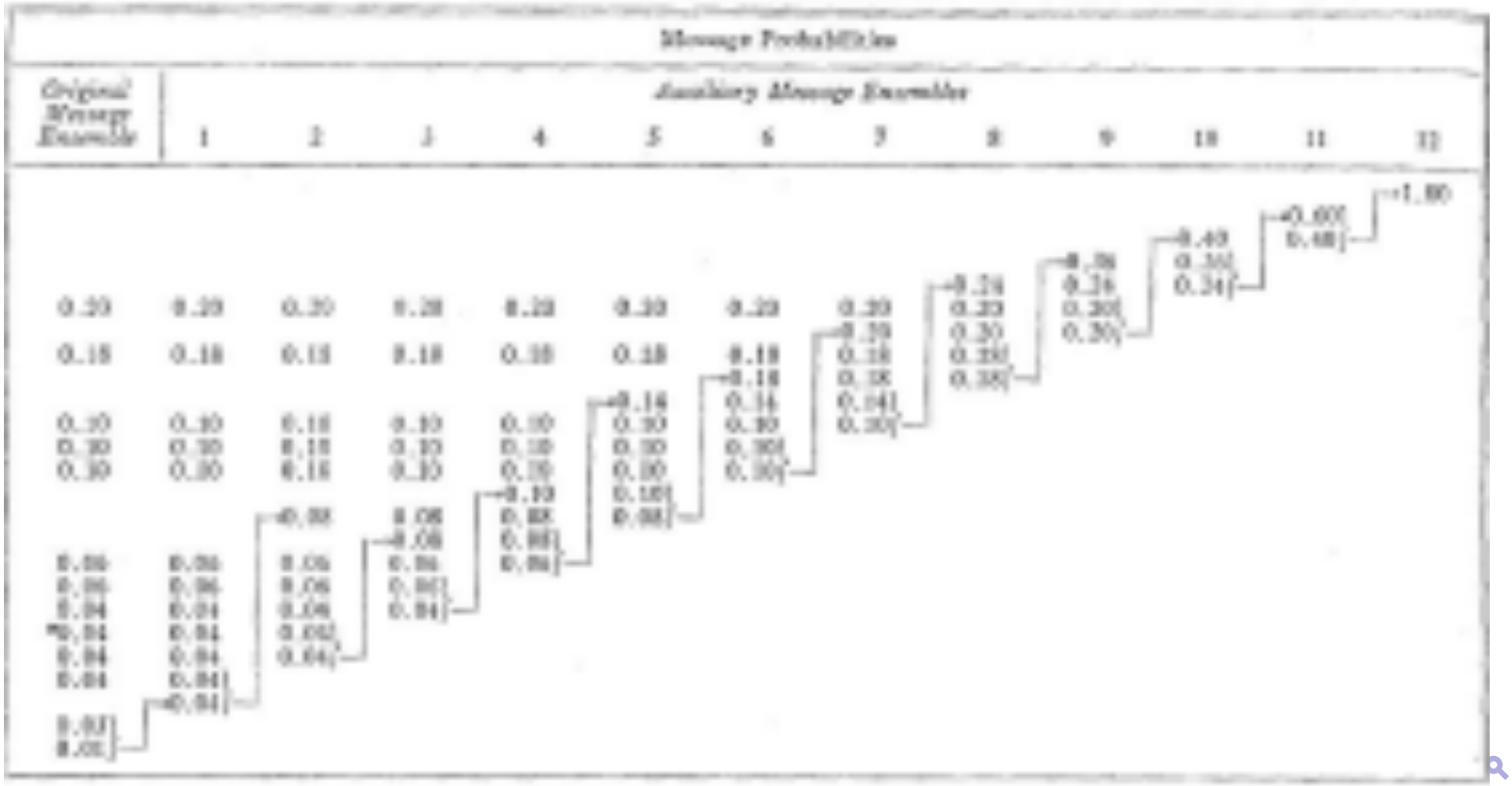
(aka Huffman Codes) (Fano, Shannon)



[Huffman, 1952]

TABLE I

GENERAL BINARY CODING PROCEDURE



Optimal Prefix Free Codes (OPFC)

Past Results

- $O(n \lg n)$ time [Huffman, 1952]
- $\texttt{SORT} + O(n)$ time [Leeuwen, 1976]
- $\texttt{SORT} + O(r(1 + \log \frac{n}{r}))$ time [Moffat and Turpin, 1998]

- $O(L)$ space [Milidiú et al., 2001]
- $O(nk)$ time [Belal and Elmasry, 2006]
- $O(n16^k)$ time [Belal and Elmasry, 2010]

n: number of frequencies
r: distinct symbol weight
L: maximal code length [Milidiú et al., 2001]
k: number of distinct codelengths [Belal and Elmasry, 2006]

Past Results

- $O(n \lg n)$ time [Huffman, 1952]
- $\texttt{SORT} + O(n)$ time [Leeuwen, 1976]
- $\texttt{SORT} + O(r(1 + \log \frac{n}{r}))$ time [Moffat and Turpin, 1998]

- $O(L)$ space [Milidiú et al., 2001]
- $O(nk)$ time [Belal and Elmasry, 2006]
- $O(n16^k)$ time [Belal and Elmasry, 2010]

n: number of frequencies
r: distinct symbol weight

L: maximal code length [Milidiú et al., 2001]
k: number of distinct codelengths [Belal and Elmasry, 2006]

Past Results

- $O(n \lg n)$ time [Huffman, 1952]
- $\texttt{SORT} + O(n)$ time [Leeuwen, 1976]
- $\texttt{SORT} + O(r(1 + \log \frac{n}{r}))$ time [Moffat and Turpin, 1998]

- $O(L)$ space [Milidiú et al., 2001]
- $O(nk)$ time [Belal and Elmasry, 2006]
- $O(n16^k)$ time [Belal and Elmasry, 2010]

n: number of frequencies
r: distinct symbol weight
L: maximal code length [Milidiú et al., 2001]
k: number of distinct codelengths [Belal and Elmasry, 2006]

Past Results

- $O(n \lg n)$ time [Huffman, 1952]
- SORT$+O(n)$ time [Leeuwen, 1976]
- SORT$+O(r(1+\log \frac{n}{r}))$ time [Moffat and Turpin, 1998]

- $O(L)$ space [Milidiú et al., 2001]
- ~~$O(nk)$~~ time [Belal and Elmasry, 2006]
- $O(n16^k)$ time ? [Belal and Elmasry, 2010]

n: number of frequencies
r: distinct symbol weight
L: maximal code length [Milidiú et al., 2001]
k: number of distinct codelengths [Belal and Elmasry, 2006]

Questions

1. What's the complexity in the *algebraical model*?
   - $O(n(1+\lg k)) \subset O(\min\{n \lg n, nk\})$?
   - $O(nk)$?
   - $(128, 134, 130, \ldots, 129, 255)$ is easy!
2. What's the complexity in the RAM model?
   - $1, 2, 4, 8, 16, 32, \ldots, 2^c$ becomes easy (to sort)!

Questions

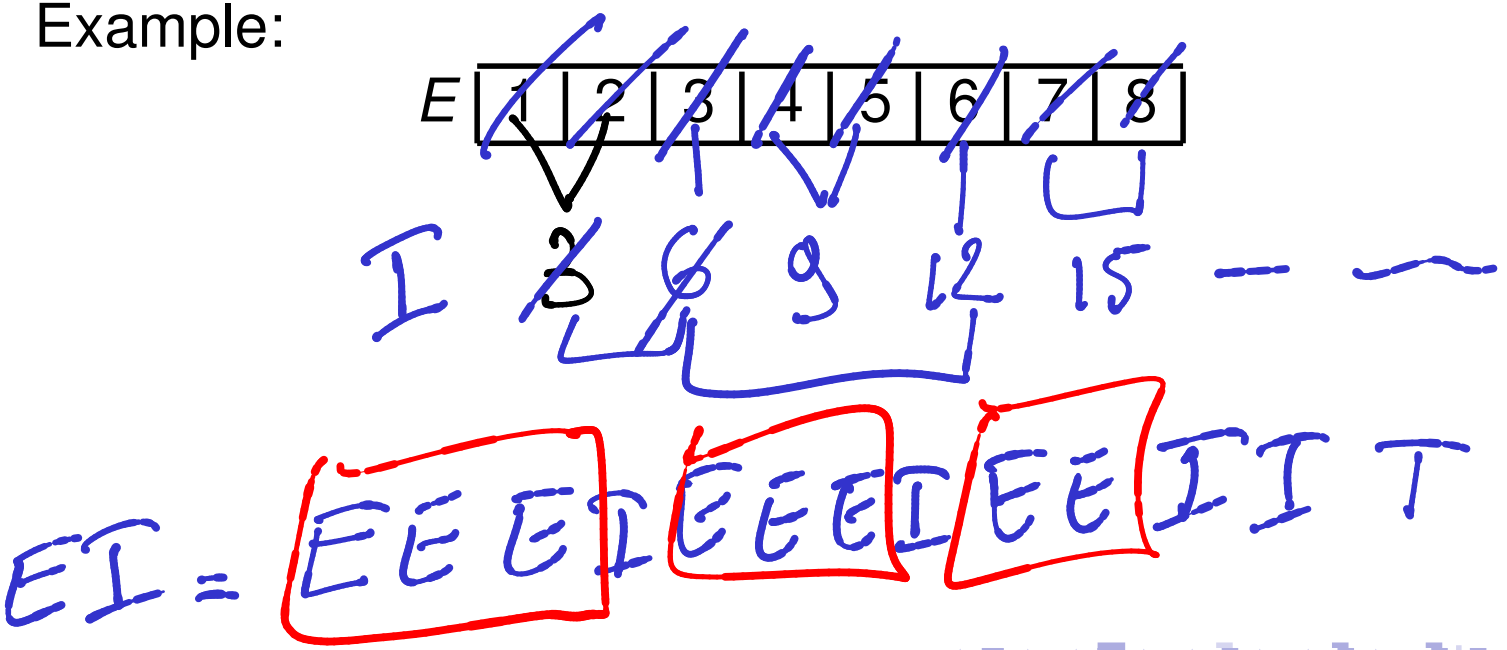
1. What's the complexity in the *algebraical model*?
   - $O(n(1+\lg k)) \subset O(\min\{n \lg n, nk\})$?
   - $O(nk)$?
   - $(128, 134, 130, \ldots, 129, 255)$ is easy!
2. What's the complexity in the RAM model?
   - $1, 2, 4, 8, 16, 32, \ldots, 2^c$ becomes easy (to sort)!

Outline

EI Signature

EI signature: string $\mathcal{S}(W) \in \{E, I\}^{2n-1}$ marking, at each step of Huffman's algorithm, whether an external or internal node is chosen as the minimum .

Example:



Deferred Data Structures

"Lazy" data structures introduced by
[Motwani and Raghavan, 1986]: partially sort, iteratively.

```
class PartiallySortedArray():
    def __init__(self,A):
    def __len__(self):
    def select(self,r):
    def rank(self,x):
    def rankRight(self,x):
    def partialSum(self,r):
    def rangeSum(self,left,right):
```

Complexity: $O(n \lg \min\{r+s, n\} + r \lg n)$

(Implementations available at `https://github.com/jyby/DDSRankSelectInMultisets`.)

Group Dock Merge Algorithm

The Python code is (relatively) simple:

```
INITIALIZE(w)
while len(ext)>0 :
    GroupExternals(w,ext,ints)
    DockInternals(w,ext,ints)
    MixOneIWithOneE(w,ext,ints)
WRAPUP(w,ints)
return ints[0]
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`

Analysis

alternation: number α of times HUFFMAN's algorithm switches between selecting "internal" and "external" nodes. Note: $\alpha \in [1..n-1]$.

Analysis of the complexity of the algorithm:

- $O(\alpha)$ loop iterations
- $O(\alpha(1 + \lg \frac{n-1}{\alpha}))$ queries
- $O(n(1 + \lg \alpha))$ comparisons (using [Motwani and Raghavan, 1986])

Outline

Measures in Practice (Part 1)

On a selection of texts from Project GUTEMBERG:

Filename	$\lvert T \rvert$	n	α	L	k
shakespeare.txt	904061	67780	440	20	16
pg7925.txt	247215	24208	228	18	15
32575-0.txt	68849	13575	115	16	13
pg31471.txt	64959	11398	121	16	13
pg24742.txt	48039	10323	103	16	13
pg25373.txt	24075	4944	72	15	12
pg12944.txt	13930	5639	51	14	11
pg4545.txt	11887	4011	49	14	11
14529-0.txt	7944	3099	40	13	10

$\lvert T \rvert$: total number of words
n: number of frequencies
α: alternation

L: maximal code length [Milidiú et al., 2001]
k: number of distinct codelengths [Belal and Elmasry, 2006]

Measures in Practice (Part 2)

$\|T\|$	n	α	L	k
7944	3099	40	13	10
11887	4011	49	14	11
12160	3662	55	14	11
13575	3596	53	14	11
13930	5639	51	14	11
21325	4586	74	14	10
22745	6900	72	15	11
24075	4944	72	15	12
24998	5851	73	15	12
28983	7213	81	15	11
31559	7658	87	15	12
32315	6693	88	15	12
34078	6130	94	15	11
35319	7636	88	15	12
38858	10389	99	15	11
38887	8846	99	15	12
46464	7592	106	16	13
48039	10323	103	16	13
52639	8495	110	16	13
54887	10209	110	16	13
59603	7880	119	16	13
62115	9528	112	16	13
62145	9836	127	16	13
62996	11427	118	16	13
63131	17447	112	16	13
64959	11398	121	16	13
66313	12601	119	16	13
68849	13575	115	16	13
71611	11975	128	16	13
73837	14178	135	16	13
78785	13603	138	16	13
81644	10259	149	16	13

$\|T\|$	n	α	L	k
81980	12592	140	16	12
83071	17951	131	16	12
88554	12856	137	16	13
90623	13341	153	17	14
93454	16744	131	17	14
104139	10943	170	17	14
107280	14482	147	17	14
113941	20077	162	17	14
115188	15313	163	17	14
118667	19174	159	17	14
118840	18783	160	17	14
120853	19182	161	17	14
123930	11789	177	17	14
128758	25804	173	17	13
132412	13923	188	17	14
138879	19665	171	17	14
138885	14852	179	17	14
160154	15040	192	17	14
169689	62113	159	18	14
187421	22189	201	18	14
215135	33780	210	18	15
219229	20700	208	18	15
232323	23788	233	18	15
247215	24208	228	18	15
268034	49443	227	18	15
383657	19968	318	19	16
428917	32830	300	19	16
566188	42045	360	19	16
634937	40213	371	19	16
824146	34057	426	20	17
904061	67780	441	20	16

Conjectures

- In RAM:

$$\texttt{OPFC} \in O(n)?$$
$$\texttt{IntegerSort} \in O(n \lg \lg n)$$

- In the Algebraic Model:

$$\begin{aligned} \texttt{OPFC} &\in O(n(1 + \mathcal{H}(n_1, \ldots, n_k)))? \\ &\subset O(n(1 + \lg k))? \\ &\subset O(n(1 + \lg \alpha)) \end{aligned}$$

Questions for you

1. Is it worth continuing?
   - Computing OPFC in less time: which applications?
   - Will faster OPFC algorithms inspire other results?

2. Which data sets to consider?
   - texts from the GUTEMBERG Project, word-based.
   - BioInformatics?
   - components of the Fourier's transform in JPEG?

Questions for you

1. Is it worth continuing?
   - Computing OPFC in less time: which applications?
   - Will faster OPFC algorithms inspire other results?

2. Which data sets to consider?
   - texts from the GUTEMBERG Project, word-based.
   - BioInformatics?
   - components of the Fourier's transform in JPEG?

Questions for you

1. Is it worth continuing?
   - Computing OPFC in less time: which applications?
   - Will faster OPFC algorithms inspire other results?

2. Which data sets to consider?
   - texts from the GUTEMBERG Project, word-based.
   - BioInformatics?
   - components of the Fourier's transform in JPEG?

Bibliography

Belal, A. A. and Elmasry, A. (2006).

Distribution-sensitive construction of minimum-redundancy prefix codes.
In Durand, B. and Thomas, W., editors, *Proceedings of the International Symposium on Theoretical Aspects of Computer Science (STACS)*, volume 3884 of *Lecture Notes in Computer Science*, pages 92–103. Springer.

Belal, A. A. and Elmasry, A. (2010).

Distribution-sensitive construction of minimum-redundancy prefix codes.
CoRR, abs/cs/0509015.
Version of Tue, 21 Dec 2010 14:22:41 GMT, with downgraded results from the ones in the conference version [Belal and Elmasry, 2006].

Huffman, D. A. (1952).

A method for the construction of minimum-redundancy codes.
Proceedings of the Institute of Radio Engineers (IRE), 40(9):1098–1101.

Leeuwen, J. V. (1976).

On the construction of Huffman trees.
In *Proceedings of the International Conference on Automata, Languages, and Programming (ICALP)*, pages 382–410, Edinburgh University.

Milidiú, R. L., Pessoa, A. A., and Laber, E. S. (2001).

Three space-economical algorithms for calculating minimum-redundancy prefix codes.
IEEE Transactions on Information Theory (TIT), 47(6):2185–2198.

Moffat, A. and Turpin, A. (1998).

Efficient construction of minimum-redundancy codes for large alphabets.
IEEE Transactions on Information Theory (TIT), pages 1650–1657.

Motwani, R. and Raghavan, P. (1986).

Deferred data structuring: Query-driven preprocessing for geometric search problems.
In *Proceedings of the Annual Symposium on Computational Geometry (SoCG)*, pages 303–312.

Outline

def gdmCodeTree(w):

```
if len(w) == 0 :
    return None
elif len(w)==1:
    return ENode(w,0)
w,ext,ints = INITIALIZE(w)
while len(ext)>0 :
    w,ext,ints = GroupExternals(w,ext,ints)
    w,ext,ints = DockInternals(w,ext,ints)
    w,ext,ints = MixOneIWithOneE(w,ext,ints)
w,ints = WRAPUP(w,ints)
return ints[0]
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`

def INITIALIZE(w):

```
ext = [ENode(w,i) for i in range(2,len(w))]
ints = [INode(w,ENode(w,0),ENode(w,1))]
return w,ext,ints
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`

def GroupExternals(w,ext,ints):

```
r = w.rankRight(ints[0].weight())
nbNodes = r-len(w)+len(ext)
nbPairs = nbNodes/2
for i in range(0,nbPairs):
    ints.append(INode(w,ext[0],ext[1]))
    ext = ext[2:]
if 2*nbPairs < nbNodes:
    ints.append(INode(w, ext[0],ints[0]))
    ext = ext[1:]
    ints = ints[1:]
return w,ext,ints
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`

def DockInternals(w,ext,ints):

```
while len(ext)>0 and len(ints)>1
      and ints[-1].weight() <= ext[0].weight():
    nbPairsToForm = len(ints) // 2
    for i in range(nbPairsToForm):
        ints.append(INode(w,ints[2*i],ints[2*i+1]))
    ints = ints[2*nbPairsToForm:]
return w,ext,ints
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`

MixOneIWithOneE(w,ext,ints)

```
if len(ext)>1 and len(ints)>1:
 children = []
 for i in range(2):
  if len(ext)==0 or
     (len(ints)>0 and ints[0].weight()<ext[0].weight()):
   children.append(ints[0])
   ints = ints[1:]
  else:
   children.append(ext[0])
   ext = ext[1:]
 ints.append(INode(w,children[0],children[1]))
elif len(ext)==1 and len(ints)>1:
 if ints[1].weight() < ext[0].weight():
  ints = ints[2:]+[INode(w,ints[0],ints[1])]
 else:
  ints = ints[1:]+[INode(w,ints[0],ext[0])]
  ext = []
elif len(ext)>1 and len(ints)==1:
(...)
return w,ext,ints
```

def WRAPUP(w,nodes):

```
while len(nodes) > 1:
    if len(nodes) % 2 == 1:
        nodes[-1].weight()
    for i in range( len(nodes) // 2):
        nodes.append(INode(w,nodes[0],nodes[1]))
        nodes = nodes[2:]
nodes[0].weight()
return w,nodes
```

(Implementation available at `https://github.com/jyby/PrefixFreeCodes`