Data Compression and Error Detection Integration

Paulo E. D. Pinto
UERJ, Universidade Estadual do Rio de Janeiro, RJ, Brasil.
Fábio Protti
Universidade Federal do Rio de Janeiro, Instituto de Matemática and NCE, RJ, Brasil.
*
Jayme L. Szwarcfiter
Universidade Federal do Rio de Janeiro, Instituto de Matemática, NCE and COPPE,RJ, Brasil.
†

Abstract

Hamming [4] proposed, in 1980, the Hamming Huffman Trees, structures for data compression, which can combine the Huffman encoding with the noise protection of Hamming encoding. Those structures can detect 1 bit-errors introduced during transmission of compressed data. This work extends that proposal and discusses the Odd Detection Tress (ODT), whose basic property is to detect any odd number of wrong bits introduced in a message. It is presented the special case where the frequencies are constant and the Optimal ODT are caracterized for this situation. It is highlighted the low cost for the addition of that feature, related to Huffman. It is also discussed one algorithm to the general case.

1 Introduction

Data compression is an area of permanent interest in Computer Science and, although it has produced relevant results since the 50's [5], it is still on a continuous evolution. New approaches to old problems have emerged, and new problems have come to light due to the Internet. The latter ones must deal with big data volumes, where compression is an economic imperative [1].

The most traditional method of compression is the Huffman method, which is very efficient and is based on statistics over ocurrences of single symbols (or aggregates of symbols) in a data set. This method constructs a binary tree for coding/decoding symbols, and each one of them is represented by a leaf of the tree. The encoding of a symbol σ is given by the path from the root of the tree to the leaf corresponding to σ, adding a bit 1 (resp. 0) when the path goes down to the right (resp. left) child of a node. It is not surprising that we can find in the literature a great number of new approaches and adaptations based on this method [3]. One of these proposals is described in the book by Richard Hamming, *Coding and Information Theory* [4], which is known by its deep treatment of error detection/correction in data transmition. Hamming proposed a new data structure, the Hamming-Huffman Tree, which combines the advantages of the Huffman compression with the noise protection of Hamming codes – tasks which are nowadays performed separatedly when processing network transmition/reception. However, Hamming-Huffman Trees have not received further developments, and remain up to now as an open field of research.

The present work is based on this idea. We discuss the structure proposed by Hamming and present a new alternative, the *Odd Detection Trees* (for simplicity, called here ODTs.)

1.1 Hamming-Huffman Trees

As said above, Hamming-Huffman Trees (HHTs) are intended to combine the benefits of Huffman compression with the noise protection of Hamming codes. The idea is that the data coding itself should contain redundancies that allow the detection of certain kinds of errors. In Hamming's proposal, all the 1-bit errors introduced during the transmission would be detected when received. This is equivalent to forbid certain encodings which, when present in the reception, would signal an error. Similarly to the Huffman Trees, HHTs are strictly binary trees whose leaves corespond to encodings. However, error leaves are also introduced in such a way that, for each encoding, every possible change in one bit due to a transmission error leads to one of these error leaves when decoding.

We present below an example from [4], p.76. The fre-

quencies are assumed to be equal. Table 1 shows the symbols and their corresponding encodings. Figure 1 shows the corresponding HHT.

Character	Encoding
a	000
b	0110
c	1010
d	1100
e	1111

Table 1. Example of a Hamming-Huffman Code.

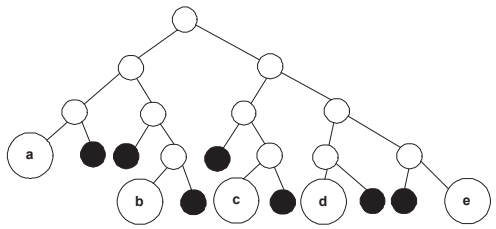

Figure 1. HHT for 5 symbols

In the above example, the prefixes $001, 010, 0111, 100$, $1011, 1101$ and 1110 are forbidden prefixes and, when occurring in the decoding process, they signal a change of one bit in a given encoding.

ODTs enlarge the initial Hamming's proposal, since they aim to detect not only one but all the errors ocurring an odd number of times, introduced in a encoded message. Let us first characterize the *Odd Detection Codes* (ODCs.)

2 Odd Detection Codes

Let Σ be a finite alphabet. An *encoding* for a givem symbol $\sigma \in \Sigma$ is a finite sequence of 0's and 1's associated to σ. Each 0 or 1 is a *bit* of the encoding. The bits of an encoding are labeled $1, 2, 3, \ldots$, from left to right. The *size* of an encoding e is the label of the rightmost bit in e. The segment $s(e, i, j)$ is the subsequence of e starting at the bit labelled i and ending at the bit labelled j.

An *Odd Detection Code* (ODC) for Σ is a set P of distinct encodings associated to the symbols of Σ, having the following properties:

1. It is a *prefix code*, that is, no encoding is a prefix of any other one. This is necessary to avoid ambiguity when decoding messages
2. When decoding a message, it allows the detection of an odd number of spurious bits in the encoded message

The following properties of an ODC are easily verifiable:

1. In an ODC P, for each pair of distinct encodings $e_i, e_j \in P$ with $|e_i| \leq |e_j|$, the following property holds: e_i and the initial segment $s(e_j, 1, |e_i|)$ of e_j are different. This is a direct consequence of the definition of an ODC.
2. Encodings with the same size have the same parity (the parity of an encoding is the parity of the number of 1's it contains.)

These properties are necessary but not sufficient in order to guarantee that a code is actually an ODC. However, if we restrict the second property by adopting only encodings with even parity, then we do have a sufficient condition to assure the existence of an ODC, according to the following theorem:

Theorem 1 *Let P be a code satisfying the following properties:*

1. *for each pair of distinct encodings $e_i, e_j \in P$ with $|e_i| \leq |e_j|$, e_i and $s(e_j, 1, |e_i|)$ are distinct;*
2. *all the encodings in P have even parity.*

Then P is an ODC.

Proof: Let P be a code with the above properties. The first property means that P is a prefix code. The second one means that, initially, the message has even parity and that, during the decoding process, if a character is recognized then the corresponding segment has even parity. If an odd number of spurious bits are introduced in the message, its parity becomes odd. Consequently, even if some segments are wrongly recognized during the decoding process, in the worst case the last segment has odd parity and the error is signaled. Therefore any introduction of an odd number of spurious bits will be detected and consequently P is an ODC. □

Given an ODC P, it is easy to see that we can find another equivalent ODC P', where all the encodings have even parity. The equivalence to P means that there is a bijection between the elements of P and P', such that the corresponding elements have the same size. In order to find such a P', it is enough to order the encodings according to their size and perform the following procedure: Let $d_1, d_2 \ldots, d_k$ be the distinct encoding sizes appearing in P. For $i = 1, \ldots, k$, take the encodings with size d_i and, in case that those encodings have odd parity, change the bit with label d_i in all encodings with size equal to or greater than d_i. Therefore, P' is ODC. Consequently, all the ODC codes we will refer to are supposed to have even parity.

Given an alphabet $\Sigma = \{s_1, s_2, \ldots, s_n\}$ with corresponding non-negative access frequencies $(f_1, f_2, \ldots, f_n)$, an

Optimal ODC (OODC) P for Σ is one minimizing the size of the encoded message. The problem of finding an OODC can be stated as follows:

Input: An alphabet $\Sigma = \{s_1, s_2, \ldots, s_n\}$, $n > 1$, and a vector of corresponding non-negative integer access frequencies $(f_1, f_2, \ldots, f_n)$.

Output: An ODC P for Σ that minimizes the objective function $f = \sum_{i=1}^{n} f_i.d_i$, where d_i is the size of the encoding corresponding to s_i in P.

2.1 Odd Detection Trees

Prefix codes can be represented by binary trees, and this fact allows to deduce some properties of the code from the properties of the associated binary tree. In what follows, we show how this association is done and study the resulting trees, especially the characterization of OODCs. All concepts related to binary trees are used from [2, 6].

Every ODC can be represented by a strictly rooted binary tree, where three kinds of nodes are used: *internal nodes, encoding leaves* and *error leaves*. The *encoding leaves* contain the alphabet symbols. The path from the root to each leaf is equivalent to the encoding assigned to the symbol represented by that leaf, where bit 0 corresponds to a left edge in the path and bit 1 to a right edge. In the deepest level of the tree, every encoding leaf has a sibling which is an *error leaf*. Sometimes, error leaves occur in intermediate levels. Error leaves are intended to signal errors. Figure 2 shows ODTs for one, two and three encodings.

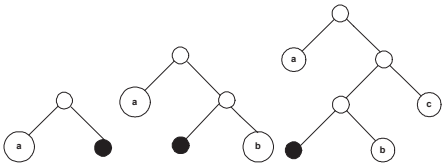

Figure 2. Examples of ODTs

An *Odd Detection Tree (ODT)* for n encodings is a rooted strictly binary tree with n leaves having even parity. Leaves with odd parity are error leaves. An *Odd Detection Tree type II (ODT-II)* for n encodings is the resulting tree by inverting the left and right subtrees of the root in an ODT. Consequently, all the encodings in a ODT-II have odd parity.

3 Optimal Odd Detection Trees

From now on, we will characterize the *Optimal Odd Detection Trees (OODTs)*.

An *Optimal Odd Detection Tree (OODT)* is an ODT associated to an OODC. An *Optimal Odd Detection Tree type II (OODT-II)* is the resulting tree by inverting the left and right subtrees of the root in an OODT.

The ODTs presented in Figure 2 are also OODTs. It is easy to see that, for $n = 1, 2, 3$, there exists only one possible OODT for each value (exactly the ones presented), independently of the frequencies.

The problem of finding an OODC can be translated in terms of OODTs. Given an alphabet with n symbols and its frequencies, the problem is equivalent to find an ODT whose *Weighted Encoding External Path Length (WEEPL)* is minimum. The WEEPL is the subsum of the Weighted External Path Length of an ODT by considering only the encoding leaves.

Let us define a special kind of ODT. An *Equilibrated ODT* with n encodings is an ODT satisfying the following property:

- for $n \leq 3$, it is one of the trees in Figure 2
- for $n > 3$, its left subtree is an Equilibrated ODT having $\lfloor n/2 \rfloor$ encoding leaves, and its right subtree is an Equilibrated ODT-II having $\lceil n/2 \rceil$ encodings. (An *Equilibrated ODT-II* is an ODT-II constructed from an Equilibrated ODT.)

The ODT presented in Figure 3 is an Equilibrated ODT for 11 encodings.

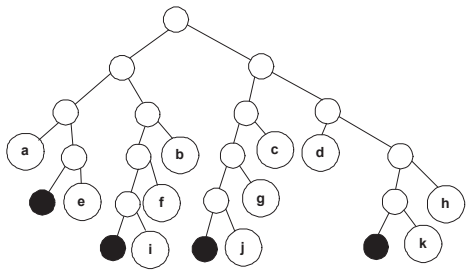

Figure 3. Equilibrated ODT for 11 encodings

First, let us consider the special case of an OODT where the frequencies are equal. We will assume thal all frequencies are one. Consequently, the WEEPL will be equal to the Encoding External Path Length of an OODT.

3.1 OODT for encodings with equal frequencies

The basic principle used to find an OODT when the frequencies are equal is that this problem has an optimal substructure, which means that the subtrees envolved are also optimal. There are a few special issues to consider, for instance the fact that some subtrees, as we shall see, cannot have only one encoding. This fact leads to an efficient dynamic programming algorithm to solve the problem, based

on the recurrence given by Theorem 2. Let $S(n)$ be the Encoding External Path Length of an OODT with n encodings, where all the frequencies are equal to one.

Theorem 2

$$S(n) = \begin{cases} 1, \ if\, n = 1 \\ 3, \ if\, n = 2 \\ n + \min\{S(n-1), \\ \min_{1<i<n-1}\{S(i) + S(n-i)\}\}, \ if\, n > 2 \end{cases}$$

Proof: The results for $n = 1$ and 2 can be verified in Figure 2. If $n > 2$, the external minimization in the expression is taken over two cases:

a) the left subtree has only one encoding. Then in this case we have $S(n) = 1+(n-1)+S(n-1) = n+S(n-1)$, since at the left side there is only one encoding, and at the right side there is an OODT-II for $n-1$ encodings. Observe now that all the encodings are one level deeper relatively to the root.

b) the left subtree has more than one encoding. Let i be the number of encodings of the left subtree (which is an OODT), $1 < i < n-1$. The right subtree is an OODT-II with $n-i$ encodings. All the encodings are one level deeper relatively to the root. The least possible value for $S(n)$ is then given by the internal minimization: $S(n) = Min\{i + S(i) + (n-i) + S(n-i)\} = n + Min\{S(i) + S(n-i)\}$, $1 < i < n-1$.

Thus, the external minimization of the expression results from the minimization of itens a) and b). □

We can make a dynamic programming algorithm based on the above recurrence, since in order to obtain the optimal value for n encodings one has to use only the optimal values for lower number of encodings. The algorithm fills three vectors: S, K_{min} and K_{max}. In S it is placed the value of the Encoding External Path Length of the OODT; in K_{min} and K_{max}, the minimum and maximum number of encodings in the left subtree of an OODT.

Algorithm OODTCF
Input: n

$S[1] \leftarrow 1$; $K_{max}[1] \leftarrow 1$; $K_{min}[1] \leftarrow 1$;
For $i = 2$ **to** n **do:**
  $m \leftarrow S[i-1]$; $K_{max}[i] \leftarrow 1$; $K_{min}[i] \leftarrow 1$;
  For $j = 2$ **to** $\lfloor i/2 \rfloor$ **do:**
    If $S[j] + S[n-j] < m$
      Then $m \leftarrow S[j] + S[n-j]$;
        $K_{max}[i] \leftarrow j$; $K_{min}[i] \leftarrow j$;
      Else If $S[j] + S[n-j] = m$
        Then $K_{max}[i] \leftarrow j$
  End-for
  $S[i] \leftarrow i + m$
Enf-for

It can be observed that, instead of a unique solution, we have a range of possible numbers of encodings in the left subtree of an OODT. That is the reason why K_{min} and K_{max} are used. K_{max} indicates the largest solution value which is less than or equal to $\lfloor n/2 \rfloor$. The algorithm analyzes only half of the possible OODTs, not treating symmetrical solutions. Its complexity is clearly $O(n^2)$. In Table 2, built from an actual implementation of this algorithm, we present the values of $S(n)$, $K_{min}(n)$ and $K_{max}(n)$, for n between 1 and 40.

n	S	K_{min}	K_{max}	n	S	K_{min}	K_{max}
1	1	1	1	21	102	9	10
2	3	1	1	22	108	10	11
3	6	1	1	23	114	11	11
4	10	1	2	**24**	120	12	12
5	14	2	2	25	127	12	12
6	18	3	3	26	134	12	13
7	23	3	3	27	141	12	13
8	28	3	4	28	148	12	14
9	33	3	4	29	155	12	14
10	38	4	5	30	162	12	15
11	43	5	5	31	169	12	15
12	48	6	6	32	176	12	16
13	54	6	6	33	183	12	16
14	60	6	7	34	190	12	17
15	66	6	7	35	197	12	17
16	72	6	8	36	204	12	18
17	78	6	8	37	211	13	18
18	84	6	9	38	218	14	19
19	90	7	9	39	225	15	19
20	96	8	10	40	232	16	20

Table 2. OODTs parameters for equal frequencies

One can highlight some facts related to Table 2:

1. The Equilibrated ODT seems to be always an OODT. This fact is given by the columns for K_{max}.
2. By using K_{min} and K_{max}, it is possible to build, recursively, the OODT.
3. For a given n, in most cases, there is more than one possible OODT. This happens when the columns K_{min} and K_{max} are different. However, when n is of the form $n = 3.2^k$, k integer, there is only one OODT.

The first fact we will prove is that the Equilibrated ODT is also an OODT, when the frequencies are equal. Let us initially present an expression $S(n)$ for the Encoding External Path Length of Equilibrated ODTs and then give the evidence that this expression is valid for any OODT with the same number of encodings. From now on, $\log n$ will allways refer to base 2.

Theorem 3 *In an Equilibrated ODT with n encodings, equal frequencies:*

$$S(n) = \begin{cases} 1, & \text{if } n = 1 \\ 3, & \text{if } n = 2 \\ n.(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}, & \text{if } n > 2 \end{cases}$$

Proof: For $n = 1, 2$, the results are the same of Table 2. For $n > 2$, the proof is by induction on n. For $n = 3, 4$ and 5, the results are also the same of Table 2. We should analyze the case where $n > 5$.

Induction Hypothesis: The expression is correct for $2 < j < n$.

Induction Step: Applying the definition and the induction hypothesis we have:

$S(n) = n + S(\lfloor n/2 \rfloor) + S(\lceil n/2 \rceil)$.

Thus:

$$S(n) = n + \lfloor n/2 \rfloor \left(\left\lceil \log \frac{\lfloor n/2 \rfloor}{3} \right\rceil + 3 \right) - 3.2^{\lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil} + \\ + \lceil n/2 \rceil \left(\left\lceil \log \frac{\lceil n/2 \rceil}{3} \right\rceil + 3 \right) - 3.2^{\lceil \log \frac{\lceil n/2 \rceil}{3} \rceil}$$

We then have two cases:

a) n is of the form $n = 3.2^{q-1} + 1$.

Then:

$\lceil \log \frac{n}{3} \rceil = q$, $\lceil \log \frac{\lceil n/2 \rceil}{3} \rceil = q - 1$, $\lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil = q - 2$, $\lfloor n/2 \rfloor = 3.2^{q-2}$.

Hence:

$S(n) = n + \lfloor n/2 \rfloor (q-2+3) - 3.2^{q-2} + \lceil n/2 \rceil (q-1+3) - 3.2^{q-1} =$
$n + (\lfloor n/2 \rfloor + \lceil n/2 \rceil).(q-1+3) - 3.2^{q-2} - 3.2^{q-2} - 3.2^{q-1} =$
$n(q+3) - 3.2^q = n(\lceil log \frac{n}{3} \rceil + 3) - 3.2^{\lceil log \frac{n}{3} \rceil}$. That is, the result is still valid for n.

b) n is of the form $n = 3.2^q$, or $n = 3.2^{q-1} + p$, $1 < p < 3.2^{q-1}$.

Then:

$\lceil \log \frac{n}{3} \rceil = q$, $\lceil \log \frac{\lceil n/2 \rceil}{3} \rceil = \lceil \log \frac{\lfloor n/2 \rfloor}{3} \rceil = q - 1$.

Hence:

$S(n) = n + \lfloor n/2 \rfloor (q-1+3) - 3.2^{q-1} + \lceil n/2 \rceil (q-1+3) - 3.2^{q-1} =$
$n + (\lfloor n/2 \rfloor + \lceil n/2 \rceil)(q - 1 + 3) - 3.2^{q-1} - 3.2^{q-1} =$
$n + n(q - 1 + 3) - 3.2^{q-1} - 3.2^{q-1} = n(q+3) - 3.2^q =$
$n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}$. That is, the result is still valid for n.

In both cases, the result is still valid for n, so the proof is complete. □

The next two theorems characterize completely an OODT when the frequencies are equal. It is supposed that the number of encoding leaves in the left subtree is not greater than the number of leaves in the right subtree. However, if that number is greater than one, the symmetrical ODT is also an OODT.

Theorem 4 *Equilibrated ODTs are OODTs when the frequencies are equal.*

The proof of this theorem is done by induction on the number n of encodings, using the basic recurrence for an OODT and the expression from Theorem 3. If we have equal frequencies and suppose that both subtrees of an OODT are Equilibrated ODTs with i encodings in the left subtree, then the function $S(n) = n + S(i) + S(n - i)$ is a non-increasing function on i in the interval $[3, \lfloor n/2 \rfloor]$. Consequently, the function has a minimum at $\lfloor n/2 \rfloor$. This implies that the Equilibrated tree for n encodings is an OODT, and the induction is complete. So, the solution of the recurrence of Theorem 2 is given by Theorem 3.

In order to characterize completely the OODTs with n encodings where the frequencies are equal, let $EL(n)$ be the number of encodings in the left subtree of an OODT.

Theorem 5 $i_{min} \leq EL(n) \leq \lfloor n/2 \rfloor$, *where*

$$i_{min} = \begin{cases} 3.2^{\lfloor \log \frac{n}{3} \rfloor - 1} & \text{if } n \leq 9.2^{\lfloor \log \frac{n}{3} \rfloor - 1} \\ n - 3.2^{\lfloor \log \frac{n}{3} \rfloor} & \text{otherwise} \end{cases}$$

The proof of this theorem follows from the previous one. The only point is to verify which is the smallest value of i for which the function $S(n) = n + S(i) + S(n - i)$ is constant in the interval $[i, \lfloor n/2 \rfloor]$. This verification enables us to determine the above values for i_{min}. In Table 3 we present the optimal configurations for OODTS having 1 to 30 encodings, as a consequence of Theorem 5. For each value of number of encodings n, it is shown the possible number of encodings in the left and right subtrees, respectively. The number of encodings in the left subtree is always considered as being not greater than the corresponding number of encodings in the right subtree, although it is possible to have the symmetrical configurations, in general.

n	Opt Confi gs	n	Opt Confi gs
1	1/0	16	6/10 7/9 8/8
2	1/1	17	6/11 7/10 8/9
3	1/2	18	6/12 7/11 8/10 9/9
4	1/3 2/2	19	7/12 8/11 9/10
5	2/3	20	8/12 9/11 10/10
6	3/3	21	9/12 10/11
7	3/4	22	10/12 11/11
8	3/5 4/4	23	11/12
9	3/6 4/5	24	12/12
10	4/6 5/5	25	12/13
11	5/6	26	12/14 13/13
12	6/6	27	12/15 13/14
13	6/7	28	12/16 13/15 14/14
14	6/8 7/7	29	12/17 13/16 14/15
15	6/9 7/8	30	12/18 13/17 14/16 15/15

Table 3. Optimal Configurations for OODT with equal frequencies

In the above table, observe that when n is of the form $n = 3.2^q$, $n = 3.2^q + 1$, or $n = 3.2^q - 1$, there is only

one optimal configuration. For n = 1, 2, and 3, we also have only one configuration.

Let us now characterize the minimum and maximum depths of OODTs with n encodings and equal frequencies. It is rather intuitive that the OODTs with minimum and maximum depths can be obtained from the following ideas:

1. OODTs with minimum depth can be recursively constructed by using left subtrees with maximum number of encodings, provided that this number is not greater than the number of encodings in the right subtree. As it was defined before, these are Equilibrated ODTs.

2. OODTs with maximum depth can also be recursively constructed by using left subtrees with minimum number of encodings.

Let $D_{min}(n)$ be the minimum depth of an OODT with n encodings and equal frequencies, $D_{max}(n)$ the maximum depth, and n_{min} the minimum number of encodings in the left subtree. We have the following recurrences:

$$D_{min} = \begin{cases} 1, & \text{if } n = 1 \\ 1 + D_{min}(\lfloor n/2 \rfloor), & \text{if } n > 1 \end{cases}$$

$$D_{max} = \begin{cases} 1, & \text{if } n = 1 \\ 1 + D_{max}(n - n_{min}), & \text{if } n > 1 \end{cases}$$

Table 4 was built from the above recurrences and data from Table 2, for number of encodings n varying from 1 to 60. Now we present the solution of the recurrences for D_{min} and D_{max}.

Theorem 6

$$D_{min} = \begin{cases} 1, & \textit{if } n = 1, \\ \lceil \log n \rceil + 1, & \textit{if } n > 1 \end{cases}$$

Proof: The proof is by induction on n. Using the recurrence, we have $D_{min}(1) = 1$, $D_{min}(2) = 2$, $D_{min}(3) = 3$. These results agree with the previous ones.

Induction Hypothesis. For $1 < i < n$, we have $D_{min}(i) = \lceil \log i \rceil + 1$.

Induction Step. Using the definition and the induction hypothesis, $D_{min}(n) = 1 + D_{min}(\lceil n/2 \rceil) = 1 + \lceil \log \frac{n}{2} \rceil + 1$, because the tree with minimum depth should also have as right subtree a tree with minimum depth and the value $\lceil n/2 \rceil$ is the smallest possible number of encodings in that subtree. We have two cases:

a) n is even, $n = 2k$, $k > 1$. Hence:

$D_{min}(n) = \lceil \log \frac{2k}{2} \rceil + 2 = \lceil \log k \rceil + 2 = \lceil \log 2k \rceil + 1 = \lceil \log n \rceil + 1$.

b) n is odd, $n = 2k + 1$, $k > 1$. Hence:

n	D_{min}	D_{max}	n	D_{min}	D_{max}
1	1	1	31	6	7
2	2	2	32	6	7
3	3	3	33	6	7
4	3	4	34	6	7
5	4	4	35	7	7
6	4	4	36	7	7
7	4	5	37	7	7
8	4	5	38	7	7
9	5	5	39	7	7
10	5	5	40	7	7
11	5	5	41	7	7
12	5	5	42	7	7
13	5	6	43	7	7
14	5	6	44	7	7
15	5	6	45	7	7
16	5	6	46	7	7
17	6	6	47	7	7
18	6	6	48	7	7
19	6	6	49	7	8
20	6	6	50	7	8
21	6	6	51	7	8
22	6	6	52	7	8
23	6	6	53	7	8
24	6	6	54	7	8
25	6	7	55	7	8
26	6	7	56	7	8
27	6	7	57	7	8
28	6	7	58	7	8
29	6	7	59	7	8
30	6	7	60	7	8

Table 4. Minimum and maximum depths of OODTs with equal frequencies

$D_{min}(n) = \lceil \log \frac{2k+1}{2} \rceil + 2 = \lceil \log \frac{2k+2}{2} \rceil + 2 = \lceil \log(k+1) \rceil + 2 =$
$\lceil \log 2(k+1) \rceil + 1 = \lceil \log(2k+1) \rceil + 1 = \lceil \log n \rceil + 1$.

In both cases, the result is also valid for n, so the proof is complete. □

One initial comparison between the last result and that one for the Huffman tree, whose depth DH_{min} is given by $DH_{min} = \lceil \log n \rceil$, shows that the difference of one unit seems to indicate the need of one extra bit, in the case of OODTs, due to the parity restriction.

Theorem 7

$$D_{max} = \begin{cases} 1, & \textit{if } n = 1 \\ \lceil \log \frac{n}{3} \rceil + 3, & \textit{if } n > 1 \end{cases}$$

Proof: The proof is by induction on n. Using the recurrence, we have $D_{max}(1) = 1$, $D_{max}(2) = 2$, $D_{max}(3) = 3$, $D_{max}(4) = 4$, $D_{max}(5) = 4$. These results agree with the previous ones.

Induction Hypothesis. For $5 < i < n$ we have $D_{max}(i) = \lceil \log \frac{i}{3} \rceil + 3$.

Induction Step. We have that $D_{max}(n) = 1 + D_{max}(n - n_{min})$, where $n - n_{min}$, given by Theorem 5, is the maximum possible number of encodings in the right subtree. Consequently, the resulting depth is the largest possible. We should analyze three possibilities, according to Theorem 5:

a) n is of the form $n = 3.2^q \leq 9.2^{\lfloor \log \frac{n}{3} \rfloor - 1}$, $q = \lfloor \log \frac{n}{3} \rfloor$.

Then n_{min} is given by $n_{min} = 3.2^{\lfloor \log \frac{n}{3} \rfloor - 1}$. Therefore, $n - n_{min} = n_{min} = 3.2^{q-1}$, and $D_{max}(n) = 1 + \lceil \log \frac{3.2^{q-1}}{3} \rceil + 3 = \lceil \log 2^{q-1} \rceil + 4 = q - 1 + 4 = q + 3 = \lceil \log \frac{n}{3} \rceil + 3$, due to the form of n.

b) n is of the form $n = 3.2^{q-1} + p \leq 9.2^{\lfloor \log \frac{n}{3} \rfloor - 1}$, $1 < p \leq 3.2^{q-2}$

Then n_{min} is given by $n_{min} = 3.2^{\lfloor \log \frac{n}{3} \rfloor - 1} = 3.2^{q-2}$. Therefore, $n - n_{min} = 3.2^{q-1} + p - 3.2^{q-2} = 3.2^{q-2} + p$, and $D_{max}(n) = 1 + \lceil \log \frac{3.2^{q-2}+p}{3} \rceil + 3 = \lceil \log(2^{q-2} + \frac{p}{3}) \rceil + 4 = q - 2 + 1 + 4 = q + 3$, due to the forms of n and p. Since $\lceil \log \frac{n}{3} \rceil = q$, then $D_{max}(n) = \lceil \log \frac{n}{3} \rceil + 3$.

c) n is of the form $n = 3.2^{q-1} + p > 9.2^{\lfloor \log \frac{n}{3} \rfloor - 1}$, $3.2^{q-2} < p < 3.2^{q-1}$.

Then n_{min} is given by $n_{min} = n - 3.2^{\lfloor \log \frac{n}{3} \rfloor}$. Therefore, $n - n_{min} = 3.2^{\lfloor \log \frac{n}{3} \rfloor}$, and $D_{max}(n) = 1 + \lceil \log \frac{3.2^{\lfloor \log \frac{n}{3} \rfloor}}{3} \rceil + 3 = \lceil \log 2^{\lfloor \log \frac{n}{3} \rfloor} \rceil + 4 = \lfloor \log \frac{n}{3} \rfloor + 4$. In this case, according to Theorem 5, n is not of the form $n = 3.2^q$. Hence, $\lfloor \log \frac{n}{3} \rfloor = \lceil \log \frac{n}{3} \rceil - 1$ and $D_{max}(n) = \lceil \log \frac{n}{3} \rceil + 3$.

In all cases, the result is still valid for n, so the induction is complete. □

We can compare, in a deeper way, OODTs with Huffman trees, when we have equal frequencies. We know that, in this case, Huffman trees are complete strictly binary trees. In this tree, the Encoding External Path Length is also the External Path Length (EPL = EEPL). If we have n encodings, then $EPL = n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}$ [7].

In Table 6 we present data comparing OODTs with Huffman trees. We show the EEPL for OODTs (a), the average length of encodings in this tree (a/n), the EEPL for the Huffman tree (b), the average length of encodings in this tree (b/n), and the difference between these two average lenghts. This parameter can be interpreted as the additional cost to endow Huffman trees with error detection.

We observe from Table 5 that the additional cost to endow Huffman trees with error detection is always less than one, which is an intuitive upper limit, as commented before. But this limit seems to be tighter, once the difference, shown in the last column, is always in the interval [0.33 , 0.5]. Next, we will prove that this is indeed the theorethic interval where this difference lies in.

n	a	a/n	b	b/n	Difference
10	38	3,8	34	3,4	0,4
24	120	5	112	4,67	0,33
32	176	5,5	160	5	0,5
48	288	6	272	5,67	0,33
64	416	6,5	384	6	0,5
100	708	7,08	672	6,72	0,36
192	1536	8	1472	7,67	0,33
500	4732	9,46	4488	8,98	0,48
1000	10464	10,46	9976	9,98	0,48
5000	63856	12,77	61808	12,36	0,41
10000	137712	13,77	133616	13,36	0,41

Table 5. Comparison between OODT and Huffman trees for equal frequencies

Theorem 8 *The difference between the average encoding lengths of an OODT and a Huffman tree, with $n > 1$ encodings and equal frequencies, lies in the interval $[1/3, 1/2]$. It is minimum when n is of the form $n = 3.2^k$, and maximum when n is of the form $n = 2^k$.*

Proof: We have four cases:

a) n is of the form $n = 2^k$, $k > 1$. Then $\lceil \log n \rceil = k$ and $\lceil \log \frac{n}{3} \rceil = \lceil \log ((2^{k-2})(\frac{4}{3})) \rceil = \lceil (k-2) + \log \frac{4}{3} \rceil = k - 2 + 1 = k - 1$. The difference between the average encoding lenghts is:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n} - \frac{(n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil})}{n}$$

By simplifying this expression, we obtain:

$$d = \frac{2^k(k - 1 + 3) - 3.2^{k-1} - (2^k(k+1) - 2^k)}{2^k} = 1/2$$

Notice that this result is also valid for $n = 2$.

b) n is of the form $n = 3.2^k$, $k \geq 0$. Then $\lceil \log n \rceil = \lceil k + \log 3 \rceil = (k + 2)$ and $\lceil \log \frac{n}{3} \rceil = \lceil \log 2^k \rceil = k$. The difference between the average encoding lengths is:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n} - \frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

Again, by simplifying this expression, we obtain:

$$d = \frac{3.2^k k + 9.2^k - 3.2^k - 3.2^k k - 9.2^k + 2^{k+2}}{3.2^k} = 1/3$$

c) n is of the form $n = 2^k + p$, $1 < p < 2^k$. Then $\lceil \log n \rceil = k + 1$ and $\lceil \log \frac{n}{3} \rceil$ may be equal to $k - 1$ or k. We have two subcases:

c.1) If $\lceil \log \frac{n}{3} \rceil = k - 1$, we have:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n}$$

$$- \frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

By simplifying and using the fact that $p > 1$, we have:

$$d = \frac{n(k - 1 + 3) - 3.2^{k-1} - n(k + 1 + 1) + 2^{k+1}}{n} < 1/2$$

c.2) If $\lceil \log \frac{n}{3} \rceil = k$, we have:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n}$$

$$- \frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

By simplifying and using the fact that, in this case, $p < 2^k$, we have:

$$d = \frac{n(k + 3) - 3.2^{k} - n(k + 1 + 1) + 2^{k+1}}{n} < 1/2$$

Thus, in the two subcases above, the maximum difference is $1/2$.

d) n is of the form $n = 3.2^k + p$, $1 < p < 3.2^k$. Then $\lceil \log n \rceil$ may be $k + 2$ or $k + 3$, and $\lceil \log \frac{n}{3} \rceil = k + 1$. We have two additional subcases:

d.1) If $\lceil \log n \rceil = k + 2$, we have:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n}$$

$$- \frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

Thus, since $p > 1$:

$$d = \frac{n(k + 1 + 3) - 3.2^{k+1} - n(k + 2 + 1) + 2^{k+2}}{n} > 1/3$$

d.2) If $\lceil \log n \rceil = k + 3$, we have:

$$d = \frac{n(\lceil \log \frac{n}{3} \rceil + 3) - 3.2^{\lceil \log \frac{n}{3} \rceil}}{n}$$

$$- \frac{n(\lceil \log n \rceil + 1) - 2^{\lceil \log n \rceil}}{n}$$

By using the fact that $p < 3.2^k$:

$$d = \frac{n(k + 1 + 3) - 3.2^{k+1} - n(k + 3 + 1) + 2^{k+3}}{n} > 1/3$$

Thus, in the two subcases above, the maximum difference is $1/3$. The proof is complete. □

Theorem 8 shows that, when the frequencies are equal, we have a better result than that suggested by intuition, because the difference between the average encoding lengths of the OODT and the Huffman tree has $1/2$ as an upper limit, which is a negligible difference for large values for n. Therefore, the cost paid to add the functionality of error detection to Huffman Trees is low.

4 OODT for general frequencies

For the general case for OODTs, where the frequencies may be different, we will sketch an algorithm which is built from the next lemma, easily demonstrable.

Lemma 9 *Let c_i,c_j be two encoding leaves in an OODT with access frequencies f_i,f_j and depths d_i, d_j, respectively. If $f_i \geq f_j$ then $d_i \leq d_j$.*

This principle is also valid for Huffman trees. It derives a Dynamic Programming algorithm, which is based in the search for the best place for the leaf with greater frequency. A secondary principle that helps is the fact that the placement of the leaf with greater frequency defines a induced forest, starting in that level, where the remaining $n - 1$ leaves belong to, and this forest is also an optimal one. This kind of algorithm is generally $O(n^3)$, but the challenge is to find a greedy solution, as we have for Huffman trees.

References

[1] R. Baeza And Y. Netto. *Modern Information Retrieval*. Addison Wesley, 1999.

[2] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, 2nd edition, 2001.

[3] M. J. Golin, C.Kenyon And N. E. Young. *Huffman Coding with Unequal Letter Costs* . Proceedings of the 34th Annual ACM Symposium on Theory of Computing. (2002) 785-791.

[4] R. W. Hamming.*Coding And Information Theory*. Prentice Hall, 1980.

[5] D. A. Huffman. *A Method for the Construction of Minimum Redundancy Codes*. Proceedings of the IRE, 40:1098-1101, 1951.

[6] J. L. Szwarcfiter And L. Markenzon. *Estruturas de Dados e Seus Algoritmos*. LTC Editora, 1994.

[7] D. E. Knuth. *The Art of Computer Programming*. Addison Wesley, 1973.