RAIRO. INFORMATIQUE THÉORIQUE ET APPLICATIONS

MARKUS E. NEBEL

Digital search trees with keys of variable length

RAIRO. Informatique théorique et applications, tome 30, n° 6 (1996), p. 507-520

<http://www.numdam.org/item?id=ITA_1996__30_6_507_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 30, n° 6, 1996, pp. 507-520)

DIGITAL SEARCH TREES WITH KEYS OF VARIABLE LENGTH (*)

by Markus E. NEBEL ([1])

Communicated by J. BERSTEL

Abstract. – *In this paper we consider Digital Search Trees (DST's) with keys of variable length, where a key might be a prefix of any other. Since the traditional insertion algorithm for DST's does not work in this case, a modification which can deal with prefixes is presented. Afterwards an average case analysis of the new algorithm is performed.*

Résumé. – *Dans cet article, nous considérons des arbres de recherche digitaux où les clés sont de longueur variable, une clé pouvant être un préfixe d'une autre. Comme l'algorithme d'insertion traditionnel ne s'applique plus dans ce cas, nous présentons une modification qui prend en compte les préfixes. Nous effectuons l'analyse en moyenne de ce nouvel algorithme.*

1. INTRODUCTION

A fundamental problem of computer science is to store data in such a way that it can be retrieven easily. We often use key values to identify a whole record of data. Digital Search Trees (DST's for short) represent one possible data structure which allows us to store, search and delete data using the key values. DST's are similar to the well known Binary Search Trees but instead of comparing key values they make use of the digital representation of the keys. If the keys can be represented as binary numbers it makes sense to refer the bth bit of a key, where the bits are numbered from left to right. Then, to insert a record (k, data) with key k into a DST, we set x to point to the root and b to 1, and perform the following operations until termination:

(i) If x is *nil* then store (k, data) in a new node $x \uparrow$ and terminate.

(ii) If $\mathit{key}\ (x) = k$ then terminate (k is already stored in the tree).

(*) Received October 1995.

([1]) Johann Wolfgang Goethe-Universität, Frankfurt, Fachbereich Informatik, D-60054 Frankfurt am Main, Germany. E-mail: nebel@sads.informatik.uni-frankfurt.de

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/96/06/$ 7.00/

(iii) Otherwise, if the bth bit of k is 0 then set x to *left* (x);
if the bth bit of k is 1 then set x to *right* (x).

(iv) Set b to $b + 1$.

(v) Goto (i).

There is no problem in searching or deleting a key. For a detailed description of DST's and Binary Search Trees *see* [6].

The insertion algorithm presented can be used only if the binary representations of all possible keys have the same length or if no key is a prefix of any other.

In former papers, e.g. [3], [5], the keys were assumed to be of infinite length to guarantee prefix-freeness. However, if the set of possible keys is known *a priori*, prefix-freeness can always be achieved by attaching a binary string which is not a binary representation of a key. But there are situations, in which we do not know all keys *a priori*, e.g. if we measure some quantity of a chemical or physical experiment electronically and get bitstrings as results. In this case we are not able to guarantee prefix-freeness. If we have prefixes it might happen that all positions that can be visited during the insertion of a key are already in use. Figure 1 illustrates this situation.

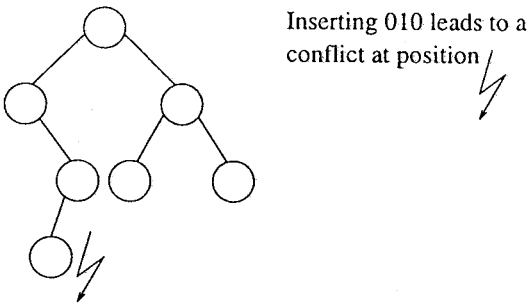

Figure 1. – The problem of using DST's with non-prefixfree keys.

We now present a modification of the insertion algorithm which can deal with keys of finite length that might be non-prefixfree. We use $\emptyset$ to denote the case that the bth bit of k does not exist. As before, we set x to the root of our DST and b to 1.

(i) If x is nil then store $(k, \textit{data})$ in a new node $x \uparrow$ and terminate.

(ii) If *key* $(x) = k$ then terminate (k already stored in the tree).

(iii) Otherwise, if the bth bit of k is 0 then set x to *left* (x);
if the bth bit of k is 1 then set x to *right* (x);
if the bth bit of k is $\emptyset$ then
set *temp* to $x \uparrow$. (*key, data*);
store (k, *data*) in the node $x \uparrow$;
continue insertion with *temp*.

(iv) Set b to $b + 1$.

(v) Goto (i).

This modification is based on the idea that a key does not need to stay at the same position for the lifetime of the tree. If the conflict of Figure 1 occurs the position of the new key, say ν, must be occupied by a key, say κ, whose binary representation κ_2 owns ν_2 as a prefix. Thus, there are some additional bits to be used by the new insertion algorithm. After the new key was stored in its last possible position the insertion process is continued with κ using these bits. Figure 2 shows an example.

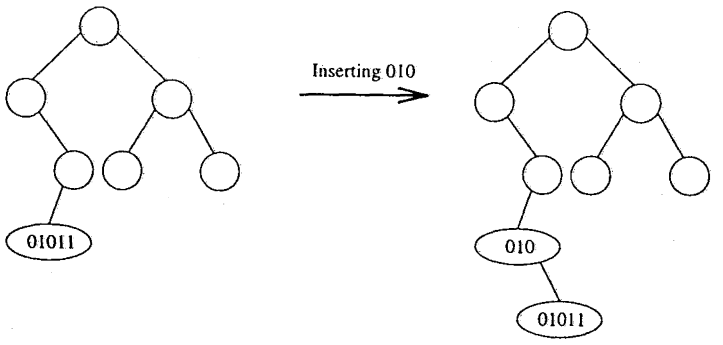

Figure 2. – The new insertion-strategy.

Compared with the traditional insertion algorithm the additional amount of work arises from conflict situations. The question is how often does such a conflict occur on the average. In the following section we introduce a formal model which is used to analyze this quantity in section 3.

2. A FORMAL MODEL

In this section we introduce the so-called Π-Tree which is a formal model that is used to analyze the average number of conflicts during the insertion process in the next section.

DEFINITION 1: *Let S be a set of keys. A key $x \in S$ is called* maximal *if it is not a proper prefix of any other key $y \in S$.*

We consider the case in which for every maximal key x all prefixes of x (including the empty word ε) are also in the set of keys. Since the number of conflicts is proportional to the number of prefixes in the set of keys this corresponds to a worst case model.

DEFINITION 2: *A* Π-Tree *is a tuple (S, f) with:*

- *S is a finite set of keys,*
- *f is an injective function from S into $\{0,1\}^*$ with the property that if ϕ is an image of f then all prefixes of ϕ are also images of f.*

The meaning of S is straightforward. The function f represents the binary representation of the keys of S. This way it frees us from any real existing coding and guarantees (because of the prefix condition) that the worst case mentioned before is considered. The existence of a binary representation ε is only for technical reasons and does not include any restriction.

We now analyze the average number of conflicts that arise during the insertion process of DST's considering Π-Trees.

3. THE AVERAGE CASE ANALYSIS

In this section we analyze the average behavior of the new insertion algorithm considering Π-Trees. We regard those Π-Trees in which every single key is exactly once inserted. That means, we use the insertion algorithm for DST's assuming the set of keys S together with the binary representation given by f. All permutations are assumed to be equally likely. We count the average number of conflicts that arise during the insertion of a single key.

Recall, that we still have a variable parameter, which is the number of maximal keys in the set S. This parameter is also determined by the function f.

It is well known that DST's tend to grow balanced which is a good reason to regard Π-Trees with $\lceil \frac{n}{2} \rceil$ maximal keys. This case corresponds to a well-balanced tree structure. But we were not able to derive a recurrence relation (or anything else) for the number of conflicts for this case directly. So we proceed as follows: First of all, we consider the case of exactly one maximal key which corresponds to a linear list. Again, this is the worst case since during the insertion of the key p there is a conflict if the position of p given by f is in use by any other key q. This is only possible if the

image of p is a (proper) prefix of the image of q. The number of possible situations of this kind is maximal in the case of a linear list-type tree. After computing the average number of conflicts we use a tree-transformation which transforms the linear list into a well-balanced tree. By counting the number of conflicts that vanish during this transformation we get a result for the well-balanced case.

If there are only left (or only right) sons in the structure given by f then there is exactly one sequence of keys which does not come into any conflict during insertion. We call this sequence *start permutation.* It is possible to generate any other sequence from the start permutation by applying one or more transpositions. We only allow those transpositions that are mandatory to generate the sequence, which are only unambiguous in number. As we begin with the sequence for which every key is inserted in its own position every transposition means an extra conflict during insertion. The following example clarifies this fact.

EXAMPLE 1: *Let the set of keys S be {1,2,3,4}, and let the binary representations be given by $f(1) = \varepsilon$, $f(2) = 0$, $f(3) = 00$ and $f(4) = 000$. This leads to the start permutation 1,2,3,4. Consider the sequence 3,1,4,2. This sequence is generated from the start permutation by the transpositions (1,3), (2,3) and (3,4). With (a, b) we denote the exchange of the positions a and b in the sequence.*

Inserting 3 into the empty tree causes no conflict; 3 becomes the root (the position of 1). Inserting 1 causes a conflict and the insertion is continued with 3. But since the position of 3 cannot be reached it is inserted in the position of key 2. Now the insertion of 4 is free of conflicts but the key is inserted in the place of 3. Inserting 2 causes two conflicts. First, it arrives at its own position which is already in use by 3. Continuing the insertion with key 3 leads to a second conflict since the position of 3 is in use by 4.

It is easy to see that the number of conflicts during the insertion of a sequence is equal to the number of transpositions needed to generate the sequence from the start permutation in general.

We now give a recurrence for the minimal number of transpositions T_n that are needed to generate all $n!$ sequences of n keys from the start permutation:

$$\begin{aligned} T_0 &= 0, \\ T_n &= nT_{n-1} + n! - (n-1)!. \end{aligned} \tag{1}$$

It is clear that we do not need any transposition to generate the permutations of 0 elements. Equation (1) for $n \geq 1$ follows from the observation described now: The first $(n-1)!$ permutations are generated by attaching the new element to the end of all the permutations of $(n-1)$ elements. Since the new element is in its own position no extra transposition is needed and we get an amount of T_{n-1} transpositions. For all missing permutations the new element is in a position that is not its own. If x denotes the new element then we have the following situation: $a_1, \ldots, a_{i-1}, x, a_i, \ldots, a_{n-1}$ where $a_j, 1 \leq j \leq n-1$, is the key in position j of a permutation of $(n-1)$ elements. This permutation is generated from the permutation $a_1, \ldots, a_{i-1}, a_{n-1}, a_i, \ldots, a_{n-2}, x$ by applying an additional transposition. Therefore, for every missing permutation p of n elements there exists exactly one permutation of $(n-1)$ elements to which an additional transposition is applied to generate p. Since there are $(n-1)$ possible positions for x we have to generate all permutations of $(n-1)$ elements $(n-1)$ times. This contributes an amount of $(n-1)T_{n-1}$ and we have to add $n! - (n-1)!$ for all the additional transpositions. Summarizing we get (1).

Using the exponential generating function $F(z) := \sum_{n \geq 0} \frac{T_n}{n!} z^n$ we get

$$F(z) = \frac{z}{(1-z)^2} + \frac{\ln(1-z)}{(1-z)}.$$

Using standard techniques, this leads to

$$\frac{T_n}{n!} = n - H_n \Rightarrow T_n = n!(n - H_n),$$

where H_n is the n-th harmonic number. We now know the number of transpositions that are needed to generate all $n!$ permutations of n elements. This number equals the number of conflicts that appear while building up all Π-Trees of size n in the case of only one maximal key. Dividing T_n by $n \cdot n!$ we get our average number for the worst case tree structure.

Our aim now is to compute the average number of conflicts in the case of $\lceil \frac{n}{2} \rceil$ maximal keys which corresponds to a well balanced tree and therefore to a good approximation of a real DST. To do this we examine the effect of transforming the structure of a linear list into the structure of a balanced tree step by step. We will carry out one step of this transformation by shifting one node of the list to the other side of the tree (with regard to the root). This is done until the nodes are distributed over the two sides as well-balanced as possible. Afterwards, we apply the same transformation to the remaining linear lists of the two sides recursively until no list being longer than 2 is

left. Figure 3 shows how this transformation builds a well-balanced tree from a linear list. Therein the transformation is recursively applied only to the shaded nodes. During the transformation the number of conflicts decreases from tree structure to tree structure. Therefore we are interested in the number of conflicts that vanish during the transformation. If we are able to determine this number we know the number of conflicts for the well-balanced tree by subtracting it from our result for the linear list. The total number of conflicts is directly related to the number of positions in the tree in which a conflict may happen during insertion. If we look at the first tree in Figure 3 we see that the node which is marked by x could have a conflict with at most 5 other nodes. By shifting this node to the right side, 4 of the 5 possible conflicts will be eliminated. Only the root is still a position in which a conflict may happen. Thus, $\frac{4}{5}$ of conflicts vanish on the average. It is more difficult to determine this number for the second step of the transformation, because the right subtree is not empty and therefore, the possible combinations of conflicts in the left and right subtree must be considered. In the second tree of Figure 3 the node y may have 4 conflicts on the left side. Shifting it to the right side eliminates 4 but contributes 2 conflicts. Totally, we get rid of $4 - 2 = 2$ conflicts but must consider the $2 \cdot 4 = 8$ possible combinations. This results into an average of $\frac{1}{4}$ vanishing conflicts. For the third step of our transformation we could use these arguments for the shaded linear list recursively, which leads to an average number of $\frac{1}{2}$.

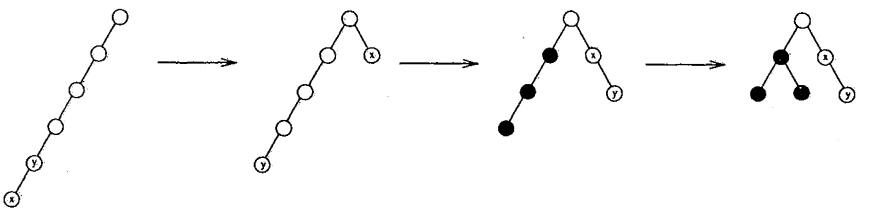

Figure 3. – The transformation of a linear list into a well-balanced tree.

If a_n denotes the average number of conflicts that vanish during the transformation of the linear list with n nodes into a well-balanced tree, we have:

$$a_0 := 0$$

$$a_n = a_{\lfloor \frac{n}{2} \rfloor} + a_{\lfloor \frac{n-1}{2} \rfloor} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{i} + \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{1}{i-n}. \tag{2}$$

The two sums in (2) result from the sum $f(n) := \sum_{i=1}^{\lfloor \frac{n}{2} \rfloor} \frac{n-2i}{(n-i)i}$ which equals the summation of the just derived average numbers. Instead of the two sums we can write $H_{\lfloor \frac{n}{2} \rfloor} - H_{n-1} + H_{\lceil \frac{n}{2} \rceil - 1}$, where H_n is the n-th harmonic number. Let Z_n denote the number of conflicts that arise during the insertion of all $n!$ permutations for a well-balanced tree structure. We have

$$Z_n = T_n - a_n n! = (n - H_n - a_n)n!.$$

From this, together with (2), it follows that

$$\begin{aligned} a_n &= n - H_n - \frac{Z_n}{n!} \\ &= \lfloor \frac{n}{2} \rfloor - H_{\lfloor \frac{n}{2} \rfloor} - \frac{Z_{\lfloor \frac{n}{2} \rfloor}}{(\lfloor \frac{n}{2} \rfloor)!} + \lfloor \frac{n-1}{2} \rfloor - H_{\lfloor \frac{n-1}{2} \rfloor} \\ &\quad - \frac{Z_{\lfloor \frac{n-1}{2} \rfloor}}{(\lfloor \frac{n-1}{2} \rfloor)!} + H_{\lfloor \frac{n}{2} \rfloor} - H_{n-1} + H_{\lceil \frac{n}{2} \rceil - 1}. \end{aligned}$$

This equation can be simplified and the substitution $u_n := \frac{Z_n}{n!}$ provides

$$u_n = u_{\lfloor \frac{n}{2} \rfloor} + u_{\lfloor \frac{n-1}{2} \rfloor} + \frac{n-1}{n}, \tag{3}$$

with the initial condition $u_0 := 0$. We solve this recurrence by regarding the difference $\Delta u_n := u_{n+1} - u_n$. Using the recurrence (3) we get

$$\Delta u_n = \Delta u_{\lfloor \frac{n-1}{2} \rfloor} + [h(n+1) - h(n)], \tag{4}$$

where $\Delta u_0 := 0$ and $h(n) := \frac{n-1}{n}$. In the sequel, we denote the difference $h(n+1) - h(n)$ by $g(n)$ and consequently we have $g(n) = \frac{1}{n} - \frac{1}{n+1}$. If we iterate the recurrence (4) $(s-1)$ times we get

$$\Delta u_n = \Delta u_{\lfloor \frac{n-2^s+1}{2^s} \rfloor} + \sum_{j=0}^{s-1} g\left(\lfloor \frac{n-2^j+1}{2^j} \rfloor \right),$$

with $u_n = \sum_{i=0}^{n-1} \Delta u_i$. After $\lfloor \log_2(\frac{1}{2}(n+1)) \rfloor$ iterations we reach the initial condition. Thus we have

$$\Delta u_n = \sum_{j=0}^{\lfloor \log_2(\frac{1}{2}(n+1)) \rfloor} g\left(\lfloor \frac{n-2^j+1}{2^j} \rfloor \right),$$

which gives the following closed-form expression for u_n :

$$u_n = \sum_{i=0}^{n-1} \sum_{j=0}^{\lfloor \log_2(\frac{1}{2}(i+1)) \rfloor} g\left(\lfloor \frac{i - 2^j + 1}{2^j} \rfloor \right).$$

This expression does not give much information about the behavior of u_n but we can regard the plots of Figure 4 and 5 to get more information. The sequence u_n seems to be of linear growth. The behavior of $\frac{u_n}{n}$ seems to be a constant of about 0.4 together with an oscillating function. To prove this conjecture we try to derive an asymptotic expression using the modified recurrence

$$y_0 = y_1 := 0,$$
$$y_n = y_{\lfloor \frac{n}{2} \rfloor} + y_{\lceil \frac{n}{2} \rceil} + \frac{n-2}{n-1}.$$

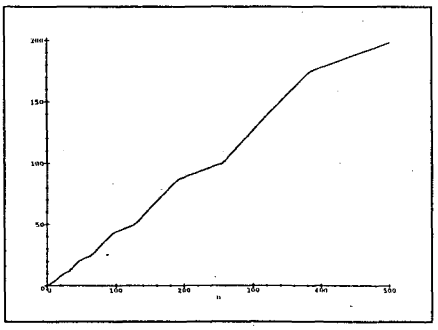

Figure 4. – A plot of u_n.

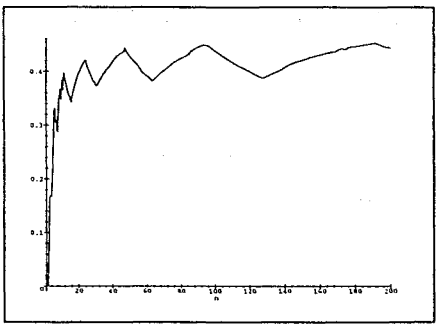

Figure 5. – A plot of $\frac{u_n}{n}$.

Note that $y_n = u_{n-1}$. We will apply the method introduced by P. Flajolet and M. Golin for analyzing recurrences of the *divide and conquer* form described in [2]. The before mentioned method is based on Dirichlet-generating functions which, for a sequence $\{w_n\}$, are defined as follows:

$$W(s) = \sum_{n=1}^{\infty} \frac{w_n}{n^s}.$$

The coefficients of a Dirichlet-series can be recovered by an inversion formula known as the Mellin-Perron formula.

LEMMA 1 (Mellin-Perron): *Assume that the Dirichlet series $W(s)$ converges absolutely for $\Re(s) > 2$. Then*

$$\frac{n}{2i\pi}\int_{3-i\infty}^{3+i\infty} W(s)n^s \frac{ds}{s(s+1)} = \sum_{k=1}^{n-1}(n-k)w_k.$$

For a detailed proof see [1]. Now consider the recurrence

$$f_n = f_{\lfloor \frac{n}{2} \rfloor} + f_{\lceil \frac{n}{2} \rceil} + e_n, \qquad n \geq 2,$$

where e_n is a known sequence and f_1 is fixed by an initial condition. In order to avoid ambiguity we set $e_0 = f_0 = e_1 = 0$. Distinguishing between odd and even cases, we find for $m \geq 1$

$$\begin{aligned} f_{2m} &= 2f_m + e_{2m}, \\ f_{2m+1} &= f_m + f_{m+1} + e_{2m+1}. \end{aligned}$$

Taking backward differences with $\nabla f_n = f_n - f_{n-1}$ and $\nabla e_n = e_n - e_{n-1}$ yields

$$\begin{aligned} \nabla f_{2m} &= \nabla f_m + \nabla e_{2m}, \\ \nabla f_{2m+1} &= \nabla f_{m+1} + \nabla e_{2m+1}, \end{aligned}$$

for $m \geq 1$. Now taking forward differences of the preceding quantities, $\Delta\nabla f_n = \nabla f_{n+1} - \nabla f_n$ and $\Delta\nabla e_n = \nabla e_{n+1} - \nabla e_n$ we get for $m \geq 1$

$$\begin{aligned} \Delta\nabla f_{2m} &= \Delta\nabla f_m + \Delta\nabla e_{2m}, \\ \Delta\nabla f_{2m+1} &= \Delta\nabla e_{2m+1}, \end{aligned} \tag{5}$$

with $\Delta\nabla f_1 = f_2 - 2f_1 = e_2 = \Delta\nabla e_1$. We now regard the Dirichlet-generating function for $w_n = \Delta\nabla f_n$ which is obtained by multiplying (5) by n^{-s} and summing over all n. The following relations hold

$$\begin{aligned} W(s) &= \sum_{m=1}^{\infty} \frac{\Delta\nabla f_m}{(2m)^s} + \Delta\nabla f_1 + \sum_{n=2}^{\infty} \frac{\Delta\nabla e_n}{n^s} \\ &= \frac{W(s)}{2^s} + \sum_{n=1}^{\infty} \frac{\Delta\nabla e_n}{n^s} = \frac{1}{(1-2^{-s})} \sum_{n=1}^{\infty} \frac{\Delta\nabla e_n}{n^s}. \end{aligned}$$

Since $\sum_{k=1}^{n-1}(n-k)\Delta\nabla f_k = f_n - nf_1$ the Mellin-Perron formula yields a direct integral representation for f_n. Summarizing this discussion leads to the following lemma [2]:

LEMMA 2: *If* $e_n = O(n)$ *holds, then*

$$f_n = nf_1 + \frac{n}{2i\pi}\int_{3-i\infty}^{3+i\infty} \frac{\Xi(s)n^s}{1-2^{-s}}\frac{ds}{s(s+1)},$$

where

$$\Xi(s) = \sum_{n=1}^{\infty}\frac{\Delta\nabla e_n}{n^s}.$$

The condition $e_n = O(n)$ guarantees the convergence of the associated Dirichlet series for $\Re(s) > 2$. Since this growing condition is satisfied for our $e_n = \frac{n-2}{n-1}$ we can use this lemma. We first have to compute $\Xi(s)$ associated with our sequence e_n. We have

$$\Delta\nabla e_n = \frac{n-1}{n} - \frac{n-2}{n-1} - \frac{n-2}{n-1} + \frac{n-3}{n-2} = \frac{2}{3n^2-2n-n^3} =: \Psi(n).$$

Clearly, $\Psi(z)$ has poles at $z = 1$ and $z = 2$. If we regard the equation $\Delta\nabla e_1 = e_2 - 2e_1 + e_0$ we get $\Delta\nabla e_1 = 0$ since e_0 and e_1 do not contribute anything as we can see from the recurrence. In an analogous way we obtain $\Delta\nabla e_2 = \frac{1}{2}$, and from this we get the following Dirichlet-generating function for $\Delta\nabla e_n$:

$$\Xi(s) = \sum_{n=1}^{\infty}\frac{\Delta\nabla e_n}{n^s} = \frac{1}{2^{s+1}} + \sum_{n=3}^{\infty}\frac{\Psi(n)}{n^s}.$$

The sum $\sum_{n=3}^{\infty}\frac{\Psi(n)}{n^s}$ converges and is $O(1)$ for every s with $\Re(s) \geq -2+\varepsilon$, $\varepsilon > 0$. Therefore, Lemma 2 tells us that

$$\frac{y_n}{n} = y_1 + \frac{1}{2i\pi}\int_{3-i\infty}^{3+i\infty}\frac{\Xi(s)n^s}{1-2^{-s}}\frac{ds}{s(s+1)}.$$

We choose the contour Γ of Figure 6 to evaluate this integral using the residue theorem. Let $\varepsilon > 0$, fix $\alpha = -2+\varepsilon$ and let $R > 0$. Then Γ is the

counterclockwise contour around

$$\Gamma_1\Gamma_2\Gamma_3\Gamma_4 \quad \text{with} \quad \begin{aligned} \Gamma_1 &= \{3+iy : |y| \leq R\}, \\ \Gamma_2 &= \{x+iR : \alpha \leq x \leq 3\}, \\ \Gamma_3 &= \{\alpha+iy : |y| \leq R\}, \\ \Gamma_4 &= \{x-iR : \alpha \leq x \leq 3\}. \end{aligned}$$

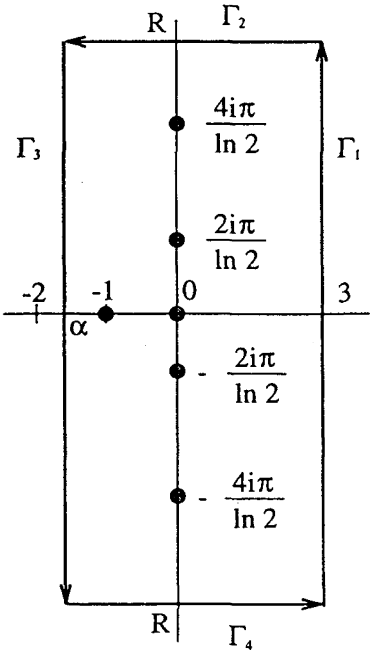

Figure 6. – The contour Γ.

If $I(s)$ denotes the kernel of the integral, *i.e.* $I(s) = \frac{\Xi(s)n^s}{1-2^{-s}} \frac{1}{s(s+1)}$, then for $R \to \infty$ our integral equals $\frac{1}{2i\pi} \int_{\Gamma_1} I(s)ds$. The integrals $|\int_{\Gamma_2} I(s)ds|$ and $|\int_{\Gamma_4} I(s)ds|$ are $O(\frac{1}{R^2})$ both. For Γ_3 we have $|\int_{\Gamma_3} I(s)ds| = O(n^\alpha)$. Thus $\frac{y_n}{n}$ equals $O(n^\alpha)$ plus the sum of the residues of $I(s)$ inside Γ. The singularities of $I(s)$ inside Γ are

- a double pole at $s = 0$,
- a simple pole at $s = -1$, and
- simple poles at $s = \frac{2ki\pi}{\ln 2} =: \chi_k, k\varepsilon\mathbb{Z}\backslash\{0\}$.

For the double pole at $s = 0$ we evaluate the coefficient of $\frac{1}{s}$ of the Laurent series for $I(s)$. We get

$$\mathrm{Res}|_{s=0}(I(s)) = \frac{\theta'(0)}{\ln(2)} + \frac{1}{2}\theta(0),$$

where $\theta(s) := \frac{\Xi(s)}{s+1}\, e^{s\ln(n)}$, *i.e.* $I(s) = \frac{1}{1-2^{-s}}\,\theta(s)\frac{1}{s}$. For the pole at $s = -1$ we have

$$\mathrm{Res}|_{s=-1}(I(s)) = \frac{1}{n}\,\Xi(-1).$$

In order to compute the residues of the poles at χ_k, $k \in \mathbb{Z}\backslash\{0\}$, we set $\omega = s - \chi_k$ and regard the Laurent series for $\omega = 0$. We get

$$\mathrm{Res}|_{s=\chi_k} = \frac{e^{2\pi i k \log_2 n}\Xi(\chi_k)}{\chi_k(1+\chi_k)\ln 2}, \qquad k \in \mathbb{Z}\backslash\{0\}.$$

Summing these residues yields

$$\frac{y_n}{n} = \frac{\theta'(0)}{\ln(2)} + \frac{1}{2}\theta(0) + \frac{1}{n}\,\Xi(-1) + \sum_{k\in\mathbb{Z}\backslash\{0\}} \frac{e^{2\pi i k \log_2 n}\Xi(\chi_k)}{2\pi i k(1+\chi_k)} + O(n^{-2+\varepsilon}).$$

Since $\theta(0) = 0$ the second summand vanishes. The same happens to the third summand for large n. The first summand is approximately 0.4270667..., *i.e.* our average number of conflicts during the insertion of a single key is the sum of a constant and an oscillating function. So what we need now is the amplitude of the oscillating function. To bound this we use $|\sum| \le \sum|\cdot|$ and get

$$\sum_{k\in\mathbb{Z}\backslash\{0\}} \left| \frac{\Xi(\chi_k)}{2\pi i k(1+\chi_k)} \right| \approx 0.0363462300939807763... \qquad (6)$$

If we just evaluate the sum numerically then we have approximately for $n \in [128..512]$:

$$-0.0334797879... \lessapprox \sum_{k\in\mathbb{Z}\backslash\{0\}} \frac{e^{2\pi i k \log_2 n}\Xi(\chi_k)}{2\pi i k(1+\chi_k)} \lessapprox 0.0294285861...$$

For $\frac{y_n}{n}$, equation (6) yields an upper bound ≤ 0.463413. Consequently, the average number of conflicts in the case of $\lceil \frac{n}{2} \rceil$ maximal keys is smaller than a small constant and our conjecture has been proved. Figure 7 shows the function $\frac{u_n}{n}$ together with our asymptotic result for $n \to \infty$.

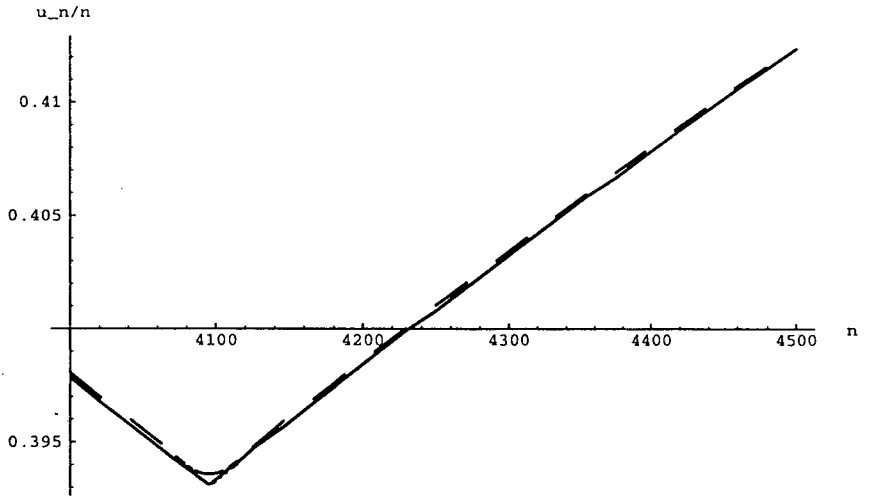

Figure 7. – Asymptotic – – – – and exact function ——— .

4. CONCLUDING REMARKS

This paper introduced a new insertion algorithm for DST's which can deal with non-prefixfree keys. We have shown that the average amount of additional work (compared with the traditional algorithm) is only small.

It is well known (see [4]) that $lim_{n\to\infty}\mathbb{E}[h(n)] = \log(n)$ in probability *i.e.* the expected height of a DST with n nodes is $\log(n)$ if n is large which means that DST's tend to be well-balanced. Since our model assumes the existence of all prefixes of a maximal key our result is to be understood as an upper bound.

REFERENCES

1. T. M. Apostol, *Introduction to Analytic Number Theory*, Springer-Verlag, 1976.
2. P. Flajolet and M. Golin, Mellin transforms and asymptotics, *Acta Informatica*, 1994, *31*, pp. 673-696.
3. P. Flajolet and R. Sedgewick, Digital Search Trees Revisited, *SIAM Journ. of Comput.*, 1986, *15*, n° 3, pp. 748-767.
4. G. H. Gonnet and R. Baeza-Yates, *Handbook of Algorithms and Data Structures*, Addison-Wesley, 1991.
5. P. Kirschenhofer and H. Prodinger, Further Results on Digital Search Trees, *Theoretical Computer Science*, 1988, *58*, pp. 143-154.
6. D. E. Knuth, *The Art of Computer Programming, Vol. 3: Sorting and Searching*, Addison-Wesley, 1973.