SOFTWARE IMPLEMENTATION OF A NEW METHOD OF COMBINATORIAL HASHING

by

P. Dubost
J. -M. Trousse

STAN-CS-75-511
SEPTEMBER 1975

COMPUTER SCIENCE DEPARTMENT
School of Humanities and Sciences
STANFORD UNIVERSITY

Software Implementation of a New Method of Combinatorial Hashing

Pierre Dubost and Jean-Michel Trousse

Stanford University

Abstract

This is a study of the software implementation of a new method of searching with retrieval on secondary keys.

A new family of partial match file designs is presented, the 'worst case' is determined, a detailed algorithm and program are given and the average execution time is studied.

This research was supported in part by National Science Foundation grant DCR72-03752 A02 and by the Office of Naval Research contract NR 044-402. Reproduction in whole or in part is permitted for any purpose of the United States Government.

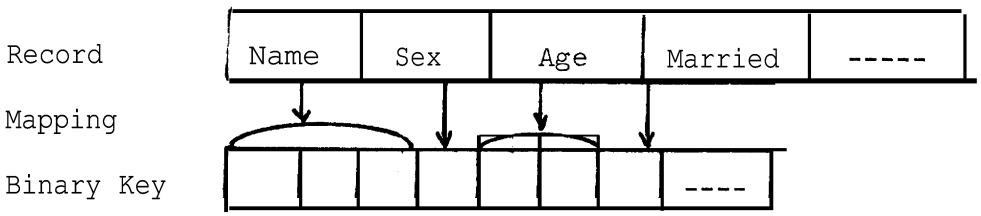
1. Retrieval on Secondary Key [2]

Common searching techniques use 'primary keys' which uniquely define a record. But, it is sometimes necessary to make a search on other fields of a record, called 'secondary keys'. We might want to retrieve some records from a file, given the values of some of these secondary keys. These values may define zero, one, or several records. A set of values is called a 'query'.

For example, if we considered an hypothetical CIA file containing information about all the American people, we might be interested in knowing which men are married, own two cars, and have been to France last year. We do not specify their age, residence, etc.

We now may assume that each secondary key is mapped by a common hashing technique into a short bit string. We then need a fast method of retrieval on a reasonably short binary key with some unspecified bits (noted *). This technique must map any specified value of this binary key into a shorter address. This address will correspond to a group of records called a 'bucket'. This technique must differ from a common hashing technique in that it allows some of the bits of the key to remain unspecified. In this case, it is clear that a query may lead to different buckets and the better the method is, the fewer buckets there will be for a given query.

Example: American People File



The previous query might be represented, for example, by: **0**111--- .

2. A New Family of Partial Match File Designs and its Binary Tree Representation.

W. A. Burkhard has recently presented a new family of partial match file designs [1]. The interesting aspect of these designs is that they can be represented by a binary tree. The binary tree leads directly to a simple software implementation. To obtain an even simpler implementation, we have slightly modified the Burkhard partial match file (PMF) and introduced a new family of PMF which has, as we show in the next section, the same worst case.

The mapping of binary keys into the bucket addresses is described by a table. Each bucket corresponds to a given value of some of the bits, the others remaining unspecified. Each entry in the tables gives a description of-the keys which might be in the corresponding bucket. For example, the bucket corresponding to the entry *10*1 might contain the following keys:

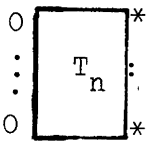
01001 , 01011 , 11001 , 11011 .

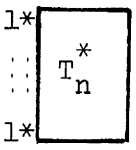
Any specified key must be mapped into one, and only one, bucket. This technique works for any query with an odd number of bits. The tables T_n are constructed by induction.

T_0 corresponds to a one bit key and two buckets:

0	0
1	1

T_{n+1} is deduced from T_n by the following method:





T_n^* being T_n circularly shifted by one position (instead of the symmetric image of T_n with columns in the reversing order, as in the Burkhard technique). It is clear that T_n will have $2n+1$ columns and 2^{n+1} lines. It then maps $(2n+1)$-bit keys into 2^{n+1} buckets.

The representation of each table T_n by a tree S_n is then very simple. A node of the tree S_n contains the position of a bit in the query to be checked and a leaf contains the address of a bucket.

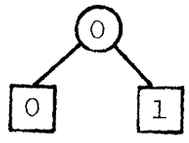
Let us consider the trees for n = 0 and 1 .

n = 0

The table T_0 is

0
0
1

which means, if the bit is 0 , the corresponding bucket address is 0 and if the bit is 1 , the corresponding bucket address is 1 . We can represent this procedure by a very simple tree S_0



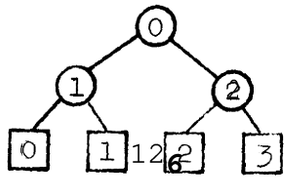
with the convention that -- if the bit checked at a node is 0 , we go to the left subtree (here the leaf [0]) and to the right sub-tree for 1 (here the leaf [1]). If the bit is a * , we go down in both subtrees.

n = 1

T_1 is

	0	1	2
0	0	0	*
1	0	1	*
2	1	*	0
3	1	*	1

the corresponding tree S_1 is



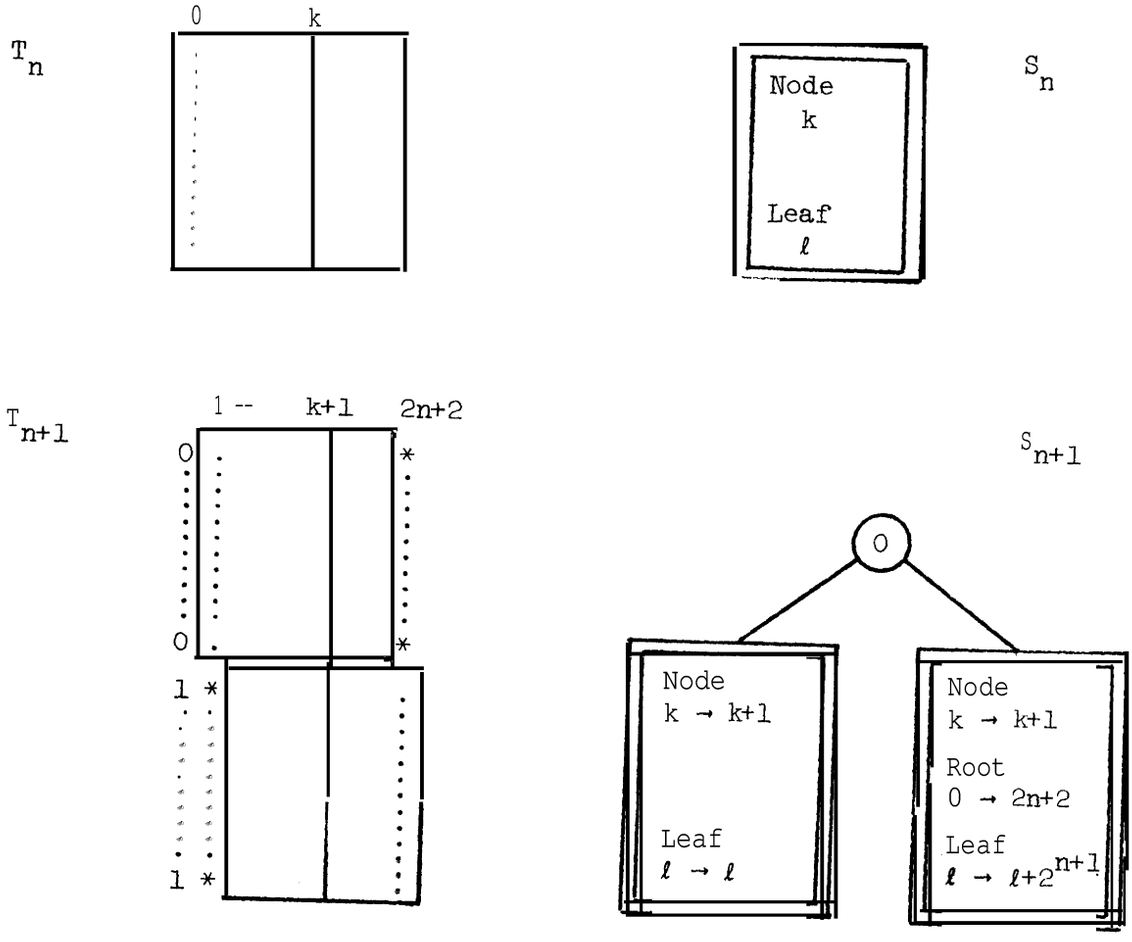
We have seen the passage from T_n to T_{n-i-1}. Similarly, we can define a transformation on the corresponding trees between S_n and S_{n+1}. Let us suppose that we have our tree S_n corresponding to the table 'I'$_n$:

The first bit separates T_{n+1} into two halves. If the first bit is 0, the corresponding bucket address is going to be contained in the first-half; or, if it is 1, in the second-half. Correspondingly, the root of S_{n+1} (node (0)) separates S_{n+1} into a left and right subtree.

In the first-half of T_{n+1}, T_n is directly inserted, which means if a query leads to the bucket ℓ in T_n, the query constructed with the preceding query (shifted by 1 position) with a 0 leading bit, is going to lead to the same bucket ℓ in T_{n+1}. So, the left subtree of S_{n+1} is S_n but with the content of each node increased by one.

In the second-half of T_{n+1}, T_n is inserted after a circular shift of one position. In the same way, the right subtree of S_{n+1} is S_n but with the content of each node increased by one, except the root (0) changed into (2n+2) and the content of each leaf increased by 2^{n+1}.

Let us represent on the same figure, the induction on T_n and S_n :



It is easy to check the transformation between S_0 and S_1 .

Let us now determine the various properties of the tree S_n (see T_4 and the corresponding tree S_4 on the next page)

A -- S_n is perfectly balanced.

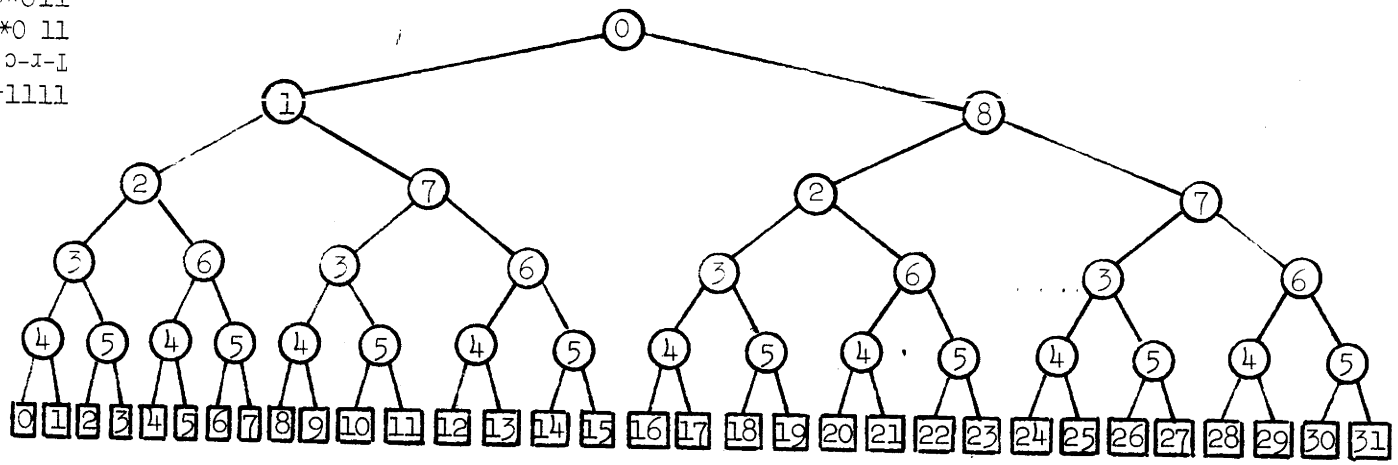
B -- At level i ($0 \leq i < n$) , the content of' the nodes is either i (denoted by min_i) or 2n+1-i (denoted by max_i). min_i is always a left son and max_i a right son.

C -- The content of a leaf can be computed directly by the path used from the root adding at each level i , 0 if we go to the left and 2^{n-i} , if we go to the right.

T_4

	01234567 8
0	00000****
1	00001****
2	0001*0***
3	0001*1***
4	001*0*0**
5	001*1*0**
6	001**01**
7	001**11**
8	01*00**0*
9	01*01**0*
10	01*1*0*0*
11	01*1*1*0*
12	01**0*01*
13	01**1*01*
14	01***011*
15	01***111*
16	1*000***0
17	1*001***0
18	1*01*0**0
19	1*01*1**0
20	1*1*0*0*0
21	1*1*1*0*0
22	1*1**01*0
23	1*1**11*0
24	1**00**01
25	1**0 1**01
26	1**1*0*01
27	1**1*1*01
28	1***0*011
29	1***1*0 11
30	illegible
31	1****1111

S_4



0 1 2 3 4

Properties A , B , C hold for S_0 , S_1 . Suppose that A , B , C hold for S_n :

Obviously, A holds for S_{n+1} .

The level i of S_n corresponds to the level i+1 of S_{n+1} . For $i > 0$, the node i of S_n gives i+1 and the node 2n+1-i gives $2n+2-i = 2(n+1)+1-(i+1)$. For $i = 0$, the node 0 of S_n gives 1 in the left subtree and $2n+2 = 2(n+1)+1-1$ in the right subtree. So, the property B holds for S_{n+1} .

The increase at level i+1 in S_{n+1} is either 0 or $2^{n+1-(i+1)}$ exactly the same as in level i in S_n . If we follow the same path in the left subtree of S_{n+1} as in S_n , we find ℓ as in S_n . If we follow the same path in the right subtree of S_{n+1} , we find $\ell+2^{n+1}$ since we had to go right in S_{n+1} at level 0 and add 2^{n+1} . . ., C holds for S_{n+1} .

The representation of this family of partial match file designs, by so simple a binary tree, leads to an easier proof of the worst case and a very simple search algorithm.

3. Worst Case.

The worst case is expressed in terms of the number of buckets $w_n(k)$ found when k bits are unspecified. W. A. Burkhard has shown that the worst case for his PMF family can be expressed in terms of the Fibonacci number in the following way:

$$w_n(k) = 2^k \qquad \text{for} \quad 0 \le k \le n+1-\left\lceil \frac{n}{2} \right\rceil$$

$$w_n(k) = 2^{n+1-k} \; F_{2k-n+1} \qquad \text{for} \quad n+1-\left\lceil \frac{n}{2} \right\rceil \le k \le n+1$$

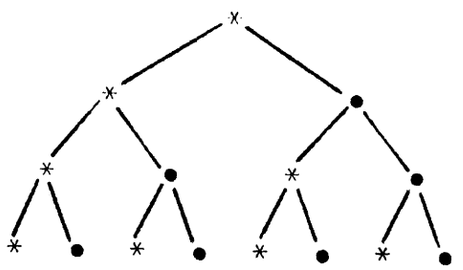
$$w_n(k) = 2^{k-(n+1)} \; F_{2n+4-k} \qquad \text{for} \quad n+1 < k \le 2n+1$$

Using the tree, we can prove those same results for our new PMF family. Let us set the notation: If the position $i \le n$ (or $i > n$) of the query contains a * , all the nodes which are left sons (or right sons)

result in a * in the level i (or $2n+3-i$) of S_n. In such a case, we are going to say that the level i has one star. Consequently, if all the nodes at a level i are *-s, we are going to say that the level i has two stars.

Finally, we represent S_n in the following way: a node is denoted by a * if it corresponds to an unspecified bit and by a • if it corresponds to a specified bit.

As an example, we can have the following tree:



Since in our study of the worst case we considered first the left son to be unspecified, a dot corresponds, in fact, to a bit 0.

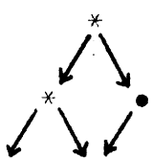
We can now study three different facts:

Fact 1. Let us see how the Fibonacci number arises in the worst case in studying the special case where all the levels contained one star. Such a tree with n+1 levels is denoted S_n^*.

For n = 0, we have S_0^*

2 buckets = F_3

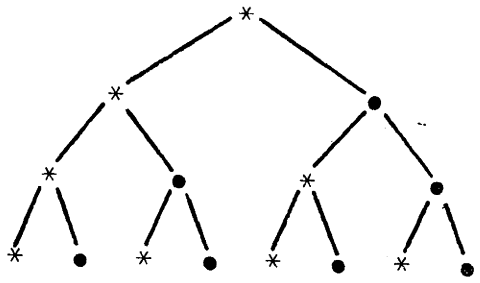
For n = 1, we have S_1^*



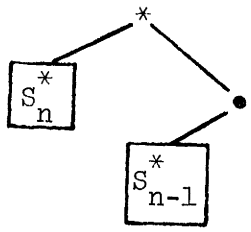
3 buckets = F_4

Let us assume that the formula for the number of buckets found, F_{n+3} holds for n and n-1, and let us study the case for n+1.

S^*_{n+1}



which can be expressed by:



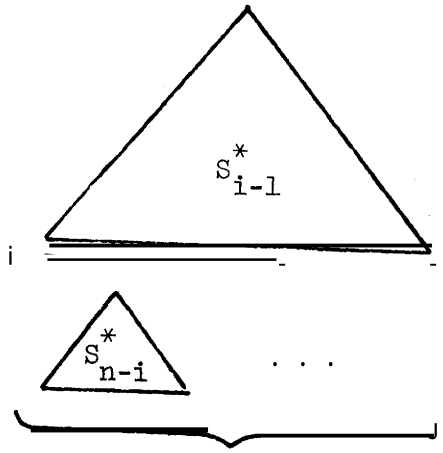
So, # of buckets for S^*_{n+1} = # of buckets for S^*_n + # of buckets for S^*_{n-1}

$$= F_{n+3} + F_{(n-1)+3} = F_{n+3} + F_{n+2}$$

$$= F_{n+4} = F_{(n+1)+3}$$

and the number of buckets for $S^*_n = F_{n+3}$.

Fact 2. Suppose a level i in S^*_n contains two stars, let us see at which level i the number of buckets is maximum. Such a tree can be represented by the following figure:



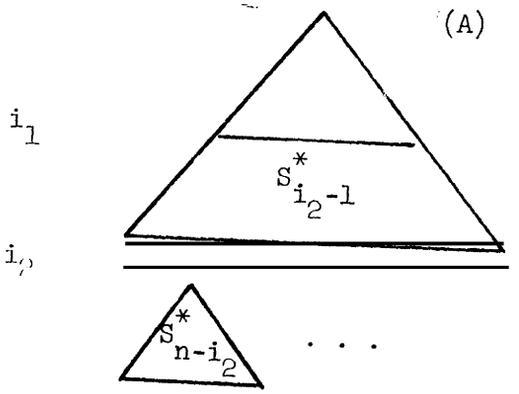
F_{i+2} trees S^*_{n-i} are explored and we have $F_{i+2}\ F_{n-i+3}$ buckets.

Now we have: $F_{i+2}F_{n-i+3} = F_{n+3} \cdot F_i F_{n+1-i}$.

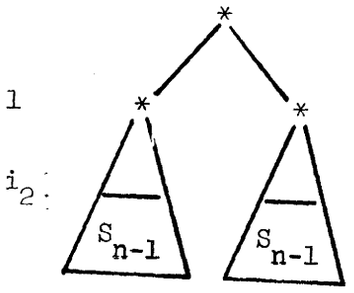
Then it can be easily shown that this product is maximum for i = 1 (or i = n+1) since $F_i F_{n+1-i}$ varies like $(-1)^{i+1}(F_{n-2i} + 2F_{n-2i+1})$. So, in a tree S_n^* a level i with two * s gives the worst case for i = 1 or n+1 .

We can now claim that if we have j levels with two stars , the worst case is obtained when those j levels are assembled either at the top of the tree or at the bottom.

Let us consider the case where j = 2 . We can represent the tree S_n^* in the following way:

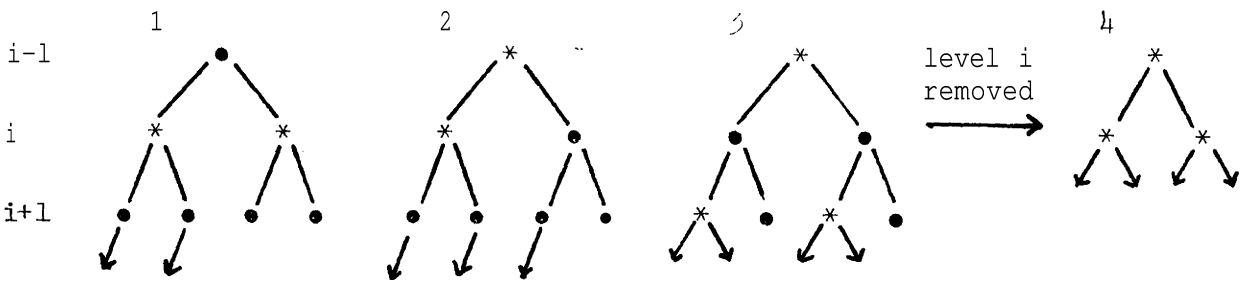


For any i_2 , the tree $S^*_{i_2-1}$ gives the maximum of entries at level i_2 if the level i_1 is equal to 1 , then we obtained the following representation: see (B).



Now S^*_{n-1} gives the maximum of buckets if i_2 in S^*_{n-1} equals 1 , so in S_n^* , $i_2 = 2$. This procedure can be clearly extended to any number j .

Fact 3. The # of paths generated by the respective positions of two stars, is the following:



1. On the same level, does not affect each other.
2. On two consecutive levels, the second * affects one of the paths generated by the other.
3. At least one level apart, each of the paths generated by the first * is affected by the second one.
4. We can also remark that any level without *s can be removed from the tree without changing the final number of buckets.

The analysis of the worst case for any k follows easily.

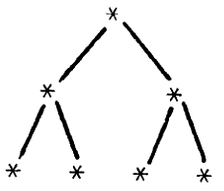
For $0 \leq k \leq n+1-\lceil \frac{n}{2} \rceil$, the number of stars in this range allows us to place all the *s one level apart so, going down the tree, each new * affects every path generated by the others. So

$$w_n(k) = 2^k$$ (equals the upper bound).

We can remark, also, that if we cancel all the levels without *s, we get a tree with only *s and k levels.

For $n+1-\lceil \frac{n}{2} \rceil \leq k \leq n+1$, the number of *s is such that n+1-k levels have no *s since no levels have two *s for the worst case (see Fact 3). Thus, we can cancel those n+1-k levels and we are left with a tree with n-(n+1-k) levels in which n+1-k levels have two stars. By Fact 2, these n+1-k levels with two *s have to be at the top of the tree for the worst case.' We can then represent the tree in the following way:

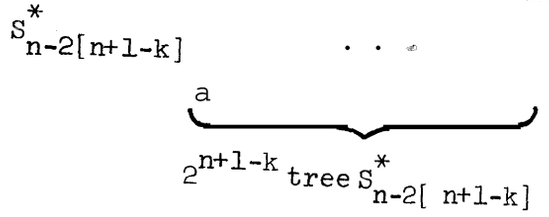
n+1-k levels



so we get:

$$w_n(k) = 2^{n+1-k} F_{n-2[n+1-k]+3}$$

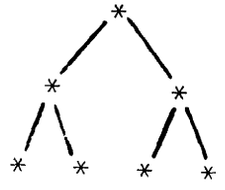
and finally,



$$w_n(k) = 2^{n+1-k} F_{2k-n+1}$$

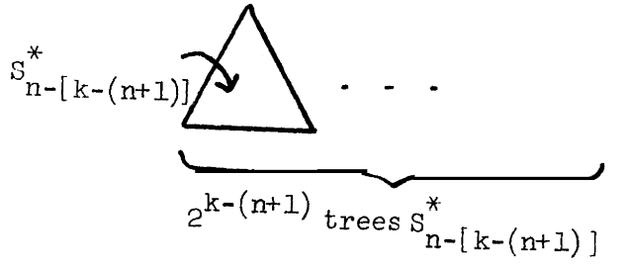
For $n+1 \leq k \leq 2n+1$, the number of * s is such that k-(n+1) levels have two stars and the others have one * . By Fact 2, the levels with two * s have to be at the top for the worst case. So, the tree can be represented in the following way:

k-(n+1) levels



We get:

$$w_n(k) = 2^{k-(n+1)} F_{n-[k-(n+1)]+3}$$



$$w_n(k) = 2^{k-(n+1)} F_{2n+4-k}$$

4. Searching Algorithm.

As we have previously seen, it is not necessary to keep the tree in memory because the value of the nodes may be computed when going down this tree.

We will use the following variables:

L = the level in the tree, beginning at zero.
B = the bucket address which is computed during the search.
QUERY(i) = the i-th bit of the query (starting at zero).

The algorithm is similar to a binary search. When getting to a node, the bit of the query specified in this node is tested. If it is zero, the left subtree is explored and if it is one, the right subtree is explored. In the case of an unspecified bit, the current level and bucket address at this node are saved on a stack, then the left subtree is explored. When getting to a leaf, the bucket address is stored in a table. If the stack is empty, the search is completed; otherwise, the level and bucket address of a node are popped from the stack and the right subtree of this node is explored.

Algorithm S [Searching all the bucket addresses corresponding to a given query.]

1. [INITIALIZE.] Set $i \leftarrow 0$, $B \leftarrow 0$, $L \leftarrow 0$.

2. [TEST BIT OF QUERY.] set $L \leftarrow L+1$, set $B \leftarrow 2B$. If query(i) = '1', go to 7.

3. [UNSPECIFIED BIT.] If query(i) = '*', push (L,B) on the stack.

4. [MOVE LEFT.] Set $i \leftarrow L$.

5. [TEST FOR LEAF.] If $L < N$, go to 2; otherwise, store B in the bucket table.

6. [TEST FOR DONE.] If stack empty, the algorithm terminates; otherwise, pop (L,B) from the stack.

7. [MOVE RIGHT.] Set $i \leftarrow 2N+1-L$, set $B \leftarrow B+1$, go to 5.

Algorithm S has been implemented in MIX. We give the MIX program and, in particular, the inner loop corresponding to Algorithm S in Appendix 1.

5. Detailed Study of the Average Search Time.

We will now study the average execution time $U_n(k)$ for computing all the bucket addresses corresponding to a query with k unspecified bits for a given n.

We will consider, in the following argument, two time costs:

C_D is the time cost for testing a node and getting to the next node or leaf (left or right son).

C_S is the additive cost when encountering a "*" for saving the parameters in the stack and then restoring them.

From the MIX program, we can see that C_D has a value of 8 when the son is not a leaf, and 9 when the son is a leaf. We can also see that C_S is equal to 10.

By considering that any time we get to the leaf, except the last time, we have to pop new parameters from the stack, we may take C_D = 8 and add the extra cost when getting to a leaf, to C_S. Then, we will take in the following study:

$$\left.\begin{array}{l} C_S = 11 \\ C_D = 8 \end{array}\right\} \quad \text{Units of MIX time}$$

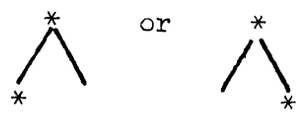
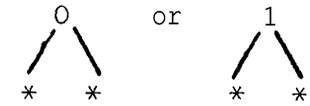
Let us now try to express $U_n(k)$ as a function of k and n. For a given n, k can take all the values between 0 and $2n+1$. We have seen the direct representation of Algorithm S by the tree S_n and the relation of the trees for n and $n+1$. Those preceding observations lead us to express the average time for $n+1$ in terms of the average time for n. We can remark that if the key 0 is not specified, the average time spent in the left and right sub-tree cannot be considered equal since the root of each of those subtrees are different.

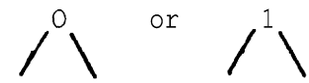
In respect of this last remark, we have isolated two types of average time:

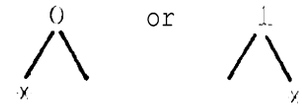
$A_n(k)$ = Average time for n if k keys are unspecified, the first key being specified.

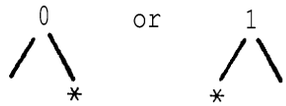
$B_n(k)$ = Average time for n if k keys are unspecified, the first key being unspecified.

Seven different cases, regarding the diverse combinations of the two first keys and the last one, are going to be considered:

Case 0 1  Case 5 

Case 0 2  Case 6 

Case 0 3  Case 7 

Case 0 4 

Case 01

$$B_{n+1}(k) = A_n(k-2) + B_n(k-1) + 2C_D + C_S$$

since we have to explore both subtrees, the average time in one is $A_n(k-2)$ and $B_n(k-1)$ in the other one. We have also to follow the two edges corresponding to the time $2C_D$ and to pop in and out the stack corresponding to C_S. In the same way we can compute the

$A^{(i)}_{n+1}(k)$ (or $B^{(i)}_{n+1}(k)$) in each case (i) :

(2) $B_{n+1}(k) = 2B_n(k-2) + 2C_D + C_S$

(3) $B_{n+1}(k) = 2A_n(k-1) + 2C_D + C_S$

(4) $A_{n+1}(k) = A_n(k-1) + C_D$

(5) $A_{n+1}(k) = B_n(k-1) + C_D$

(6) $A_{n+1}(k) = A_n(k) + C_D$

(7) $A_{n+1}(k) = B_n(k) + C_D$

We have now to compute the probability for each case to occur.

[Remark: for n+1 we have $2(n+1)+1 = 2n+3$ bits]

Case 1

{ Number of possibilities to have k stars with one star in 0 and one star in 1 } =

{ Number of symmetrical cases } × { Number of ways to select (k-2) items out of $(2n+3)-3 = 2n$ (since the items 0 , 1 and $2n+2$ are already chosen } × { Number of ways to fill the $2n+3-k$ left bits with 1 or 0 }

↑ 2 ↑ $\binom{2n}{k-2}$ ↑ 2^{2n+3-k}

so

(1) $N_{n+1} = 2\binom{2n}{k-2}2^{2n+3-k}$

In the same way we obtain:

$$N^{(2)}_{n+1} = \binom{2n}{k-3} 2^{2n+3-k} \qquad N^{(3)}_{n+1} = \binom{2n}{k-1} 2^{2n+3-k} \qquad N^{(4)}_{n+1} = \binom{2n}{k-1} 2^{2n+3-k}$$

$$N^{(5)}_{n+1} = \binom{2n}{k-2} 2^{2n+3-k} \qquad N^{(6)}_{n+1} = \binom{2n}{k} 2^{2n+3-k} \qquad N^{(7)}_{n+1} = \binom{2n}{k-1} 2^{2n+3-k} .$$

The total number of ways N^{B}_{n+1} when the first bit is unspecified is given by:

$$N^{(B)}_{n+1} = \binom{2n+2}{k-1} 2^{2n+3-k}$$

The total number. of ways $N^{(A)}_{n+1}$ when the first bit is specified is given by

$$N^{(A)}_{n+1} = \binom{2n+2}{k} 2^{2n+3-k} .$$

We can easily verify that

$$N^{(B)}_{n+1} = \sum_{i=1}^{3} N^{(i)}_{n+1} \qquad N^{(A)}_{n+1} = \sum_{i=4}^{7} N^{(i)}_{n+1}$$

and finally that

$$N_{n+1} = N^{(B)}_{n+1} + N^{(A)}_{n+1} = \binom{2n+3}{k} 2^{2n+3-k} = \left\{ \begin{array}{l} \text{total number of} \\ \text{queries having k} \\ \text{bits unspecified} \\ \text{for n+1} \end{array} \right\} .$$

We can derive $A_{n+1}(k)$ and $B_{n+1}(k)$

$$A_{n+1}(k) = \frac{1}{N^{(A)}_{n+1}} \sum_{i=4}^{7} N^{(i)}_{n+1} A^{(i)}_{n+1}(k) \quad , \quad B_{n+1}(k) = \frac{1}{N^{(B)}_{n+1}} \sum_{i=1}^{3} N^{(j)}_{n+1} B^{(j)}_{n+1}(k)$$

and finally $U_{n+1}(k)$

$$U_{n+1}(k) = \frac{1}{N_{n+1}} \left[\underset{n+1}{A} \textcircled{A} A_{n+1}(k) + \textcircled{B} N_{n+1} B_{n+1}(k) \right] .$$

If we write $W_n(k) = \binom{2n+1}{k} U_n(k)$, we get after computation the following recurrence formula:

$$W_{n+1}(k) = \alpha_{n+1k} + 2W_n(k-2) + 3W_n(k-1) + W_n(k)$$

with:

$$\alpha_{nk} = \binom{2n}{k-1}(2C_D + C_S) + \binom{2n}{k} C_D$$

and the initial condition:

$$W_0(0) = C_D \ , \quad W_0(1) = 2C_D + C \ .$$

Assuming

$$W_n(i) = 0 \qquad \forall i < 0 \quad \text{or} \quad \forall i > 2n+1 \ .$$

(1)

Let us now define the generating function $G_n(z) = \sum_{k \geq 0} W_n(k) z^k$.

From (1) we directly deduce that:

$$G_n(z) = \sum_{k \geq 0} \alpha_{nk} z^k + (2z^2 + 3z + 1) G_{n-1}(z) \ .$$

Using the formula $(1+z)^r = \sum_{k \geq 0} \binom{r}{k} z^k$ we can simplify the sum in a_{nk} .We get:

$$\sum_{k \geq 0} \alpha_{nk} z^k = (\alpha z + \beta)(1+z)^{2n} \quad \text{with} \quad \alpha = 2C_D + C_S \quad \text{and} \quad \beta = C_D \ .$$

When we express $G_{n-1}(z)$ in terms of $G_{n-2}(z)$ and $G_{n-2}(z)$ in terms of $G_{n-3}(z)$, etc., we finally obtain:

$$G_n(z) = (\alpha z + \beta) \sum_{i=0}^{n-1} (2z^2 + 3z + 1)^i (1+z)^{2(n-i)} + (2z^2 + 3z + 1)^n G_0(z$$

but

$$G_0(z) = \alpha z + \beta$$

so

$$G_n(z) = (\alpha z + \beta) \sum_{i=0}^{n} (2z^2 + 3z + 1)^i (1+z)^{2(n-i}$$

$$2z^2 + 3z + 1 = (1+z)(1+2z)$$

so we have

$$G_n(z) = (\alpha z + \beta)(1+z)^{2n} \sum_{i=0}^{n} \left(\frac{1+2z}{1+z} \right)^i$$

Now using the formula $\sum_{0 \le k \le n} z^k = \frac{1-z^{n+1}}{1-z}$ we obtain after computation:

$$G_n(z) = \left(\alpha + \frac{\beta}{z} \right) [(1+z)^n (1+2z)^{n+1} - (1+z)^{2n+1}] .$$

We can remark that $(1+2z)^{n+1} = [(1+z)+z]^{n+1} = \sum_{j \ge 0} \binom{n+1}{j} (1+z)^{n+1-j} z^j$.

Then the product $(1+z)^n (1+2z)^{n+1}$ is:

$$(1+z)^n (1+2z)^{n+1} = \sum_{k \ge 0} \left[\sum_{j \ge 0} \binom{n+1}{j} \binom{2n+1-j}{k-j} \right] z^k$$

and finally

$$G_n(z) = \left(\alpha + \frac{\beta}{z} \right) \sum_{k \ge 0} \left[\sum_{j \ge 0} \binom{n+1}{j} \binom{2n+1-j}{k-j} - \binom{2n+1}{k} \right] z^k .$$

We get after Computation a direct formula for

$$U_n(k) = \binom{2n+1}{k} W_n(k) :$$

$$\boxed{\begin{aligned} U_n(k) &= \alpha(C_{nk} - 1) + \frac{2n+1-k}{k+1}\beta\,(C_{n,\,k+1} - 1) \\ \text{with} \\ C_{nk} &= \frac{1}{\binom{2n+1}{k}} \sum_{j \geq 0} \binom{n+1}{j}\binom{2n+1-j}{2n+1-k} . \end{aligned}} \qquad (2)$$

We can remark that this formula gives for $k = 0$ and $2n+1$ results that we could have expected:

$$U_n(0) = \beta\ (n+1)$$

$$U_n(2n+1) = \alpha(2^{n+1} - 1) .$$

Asymptot ic Behavior

Using the big- O notation we have:

$$\binom{n+1}{j} = \frac{n^j}{j!}\left(1 + \frac{1+0-1\ \ldots\ -(j-2)}{n} + O(n^{-2})\right)$$

$$= \frac{n^j}{j!}\left(1 + \frac{1 - \frac{1}{2}(j-1)(j-2)}{n} + O(n^{-2})\right) .$$

Similarly

$$\binom{2n+1}{k} = \frac{(2n)^k}{k!}\left(1 + \frac{1+0-1\ \ldots\ -(k-2)}{2n} + O(n^{-2})\right)$$

and

$$\binom{2n+1-j}{k-j} = \frac{(2n)^{k-j}}{k-j!}\left(1 + \frac{\frac{1}{2}(k-1)(k-2) - \frac{1}{2}(j-1)(j-2)}{2n} + O(n^{-2})\right) .$$

So we get for C_{nk} after computation

$$C_{nk} = \left(\frac{3}{2}\right)^k + \frac{1}{4n} \sum_{j>0} \binom{k}{j} \left(\frac{1}{2}\right)^j \ldots\ldots\ldots + O(n^{-2}) .$$

Using the first and second derivatives of the generating function

$\sum_{j>0} \binom{k}{j} \left(\frac{1}{2}\right)^j = \left(\frac{3}{2}\right)^k$ we can give a direct formula for the sum

so

$$C_{nk} = \left(\frac{3}{2}\right)^k + \frac{k}{16n} (7-k) \left(\frac{3}{2}\right)^{k-2}_0 + O(n^{-2})$$

and we finally get for $U_n(k)$

$$U_n(k) = A(k)n + B(k) + O(n^{-1})$$

with $A(k) = \frac{2\beta}{k+1} \left(\left(\frac{3}{2}\right)^{k+1} - 1\right)$

and $B(k) = \alpha\left(\left(\frac{3}{2}\right)^k - 1\right) + \frac{1-k}{1+k} \beta\left(\left(\frac{3}{2}\right)^{k+1} - 1\right) + \frac{\beta}{8} (6-k)\left(\frac{3}{2}\right)^{k-1} .$

We can verify that for $k = 0$, we get $A(0) = B(0) = \beta$

A program has been written to compute the values of $U_n(k)$. The recurrence (1) has been used instead of the final formula (2) for an easier and more efficient implementation. The results for $n \leq 20$ and graphs showing the variation of $U_n(k)$ for a given k or a given n are shown in Appendix 2.

The method is very efficient when the number of *s stays within a reasonable limit. Assuming a 1 μs cycle time, the algorithm for $n = 15$ (31 bits), will take only 65 ms with 20 *s but will take 1.77 s for $k = 30$.

In any case, the searching time will be negligible compared to the time spent for retrieving the records themselves from secondary storage. As we have already seen in the asymptotic behavior of $U_t(k)$, the graphs show the almost linearity in n for $U_t(k)$, k being fixed.

Conclusions

The study of this new technique of retrieval on secondary keys shows two different aspects:

- A very 'good' worst case.
- A very simple and efficient software implementation.

Acknowledgment

The authors would like to thank D. E. Knuth for his advice and encouragement in this work.

References

[1] W. A. Burkhard, "Partial Match Retrieval File Designs," Computer Science Division, University of California, San Diego. January, 1975.

[2] D. E. Knuth, The Art of Computer Programming, Vol. 3, Sorting and Searching (Addison-Wesley, 1973), Section 6.5.

Appendix 1

MIX Program and in particular the inner loop for Algorithm S.

```
.... MIX ASSEMBLER PROGRAM .....|LOCATION |OP  |ADDRESS ..............  . STANFORD 360 VERSION 1.4 ..... 0055 03500L
     0000000004                  |N          EQU  4
     0000000099                  |OUTSIZ     EQU  99
     0000000009                  |SIZ        EQU  N*N+1
     0000000016                  |READER     EQU  16
     0000000018                  |PRINTER    EQU  18
     1824051929        0000      |USER       ALF  QUERY :
                       0001      |INBUF      ORIG *+16
     0224031205        0017      |           ALF  BUCKE
     2300010404        0018      |           ALF  T ADD
     1905222205        0019      |           ALF  RESSE
     2200540000        0020      |           ALF  S :
                       0021      |BLANKLIN   ORIG *+24
                       0045      |OUTBUF     ORIG *+OUTSIZ
                       0144      |QUERY      ORIG *+SIZ
                       0153      |STACK      ORIG *+N+1
   0001 00 16 36       0158      |NEWONE     IN   INBUF(READER)
  -0005 00 02 51       0159      |           ENT3 -1*SIZ
   0000 00 02 49       0160      |           ENT1 0
   0161 00 16 34       0161      |           JBUS *(READER)
   0001 01 05 15       0162      |           LDX  INBUF,1
  /-0001 00 04 47      0163      |           JXNZ 1F
   0000 00 02 05       0164      |           HLT
   0000 00 18 37       0165 0163 |1H         OUT  QUER(PRINTER)
   0000 00 02 48       0166      |2H         ENTA 0
   0001 00 02 06       0167      |           SLAX 1
   0046 00 01 48       0168      |           DECA 46
   0171 00 01 40       0169      |           JAZ  *+2
   0017 00 00 48       0170      |           INCA 17
  ;0001 00 01 48       0171      |           DECA 1
   0153 03 05 24       0172      |           STA  QUERY+SIZ,3
   0001 00 00 51       0173      |           INC3 1
  /-0001 00 04 47      0174      |           JXNZ 3F
   0001 00 00 49       0175      |           INC1 1
   0001 01 05 15       0176      |           LDX  INBUF,1
   0166 00 00 43       0177 0174 |3H         J3N  2B
  -0004 00 02 51       0178      |           ENT3 -1*N
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
                       0179      |*
```

```
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*
                                 0179          |*          REGISTERS
                                 0179          |*
                                 0179          |*               R1 : BUCKET ADDRESS B
                                 0179          |*               R3 : LEVEL IN THE TREE L MINUS N
                                 0179          |*               R4 : INDEX IN QUERY
                                 0179          |*               R5 : INDEX IN OUTPUT BUFFER
                                 0179          |*               R6 : STACK POINTER
                                 0179          |*
  CCCC 00 02 49                  0179          |          ENT1 0
 -0004 00 02 51                  0180          |          ENT3 -N
  0000 00 02 52                  0181          |          ENT4 0
  0000 00 02 53                  0182          |          ENT5 0
  0000 00 02 54                  0183          |          ENT6 0
/-CCC1 CC C0 39                  0184          |          JMP  2F
  0001 00 01 54                  0185          |6H        DEC6 1
  0153 06 01 11                  0186          |          LD3  STACK,6(0:1)       POP PREVIOUS CONTEXT FROM STACK
  0153 06 37 09                  0187          |          LD1  STACK,6(4:5)
  0001 00 C0 49                  0188          |7H        INC1 1                  MOVE RIGHT : B:=B+1
 -0005 03 03 52                  0189          |          ENN4 N-SIZ,3           I:=2*N+1-L
/-0001 00 02 43                  0190          |          J3P  9F
  0001 00 00 51                  0191 0184     |2H        INC3 1                  L:=L+1
  0000 01 00 49                  0192          |          INC1 0,1               B:=2*B
  0144 C4 C5 15                  0193          |8H        LDX  QUERY,4           TEST NEW BIT
  0188 00 02 47                  0194          |          JXP  7B                '1' MOVE RIGHT
/-0001 00 01 47                  0195          |          JXZ  4F                '0' MOVE LEFT
  0153 06 01 27                  0196          |          ST3  STACK,6(0:1)      '*' SAVE CONTEXT THEN MOVE LEFT
  0153 06 37 25                  0197          |W         ST1  STACK,6(4:5)
  0001 CC 00 54                  0198          |          INC6 1
  0004 03 02 52                  0199 0195     |4H        ENT4 N,3                MOVE LEFT : I:=L
  0191 00 05 43                  0200          |          J3NP 2B                TEST FOR LEAF
 0000 01 02 48                   0201 0190     |9H        ENTA 0,1
  0000 00 01 05                  0202          |          CHAR
  0046 05 29 31                  0203          |          STX  OUTBUF+1,5(3:5)
  0001 00 00 53                  0204          |          INC5 1
  0185 00 04 46                  0205          |          J6NZ 6B                IF STACK EMPTY TERMINATE
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
                                 0206          |*
```

```
                 0206          |*
                 0206          |*
    0045 00 02 49  0206        |PRINT        ENT1  OUTBUF
    0001 00 00 53  0207        |             INC5  1
   /-0001 00 36 29 0208        |             ST5   FAZ(4:4)
    0000 01 18 37  0209        |3H           OUT   0,1(PRINTER)
    0024 00 01 53  0210        |             DEC5  24
    0024 00 00 49  0211        |             INC1  24
    0209 00 02 45  0212        |             JSP   8B
    0021 00 18 37  0213        |             OUT   BLANKLIN(PRINTER)
    0045 00 02 49  0214        |             ENT1  OUTBUF
   -0001 01 00 07  0215 0208   |RMZ          MOVE  -1,1(0)
    0158 00 00 39  0216        |             JMP   NEWONE
       0158        0217                      END   NEWONE
0000)...........................................................................................
                   0000                      GO    0
QUERY*11110*10                                                               BUCKET ADDRESSES :
    013  014  021  022

QUERY*0*0*0*0*                                                               BUCKET ADDRESSES :
    000  001  004  005  006  016  017  020  021  022  024  025

QUERY*********                                                               BUCKET ADDRESSES :
    000  001  002  003  004  005  006  007  008  009  010  011  012  013  014  015  016  017  018  019  020  021  022
 023  024  025  026  027  028  029  030  031

QUERY****10111                                                               BUCKET ADDRESSES :
    001  002  006  014  030

QUERY*1*1*1*1*                                                               BUCKET ADDRESSES :
    012  013  015  017  020  021  023  028  029  031
```

Appendix 2

Results and Graphs for $U_1(k)$

the average execution time.

BB = 8.000000 CS = 11.00000

K	J	1	2	3	4	5	6	7	8	9	10
0	6	12	24	38	49	48	56	64	72	83	88
1	17	31	40	50	64	69	79	89	99	101	119
2	0	53	63	74	86	99	111	123	136	143	151
3	0	81	94	108	124	139	155	171	187	203	219
4	0	0	135	155	175	195	215	236	257	277	294
5	0	0	189	217	244	271	298	325	354	380	408
6	0	0	0	299	337	374	411	448	485	521	558
7	0	0	0	405	461	514	565	616	666	716	765
8	0	0	0	0	625	701	774	844	914	983	1051
9	0	0	0	0	837	949	1054	1155	1252	1349	1444
10	0	0	0	0	0	1276	1428	1578	1712	1843	1972
11	0	0	0	0	0	1701	1924	2134	2334	2529	2718
12	0	0	0	0	0	0	2577	2882	3172	3451	3723
13	0	0	0	0	0	0	3429	3873	4294	4697	5086
14	0	0	0	0	0	0	0	5178	5788	6373	6937
15	0	0	0	0	0	0	0	6885	7769	8615	9431
16	0	0	0	0	0	0	0	0	10378	11501	12581
17	0	0	0	0	0	0	0	0	13797	15553	17261
18	0	0	0	0	0	0	0	0	0	20176	22226
19	0	0	0	0	0	0	0	0	0	27521	31133
20	0	0	0	0	0	0	0	0	0	0	41567
21	0	0	0	0	0	0	0	0	0	0	55269

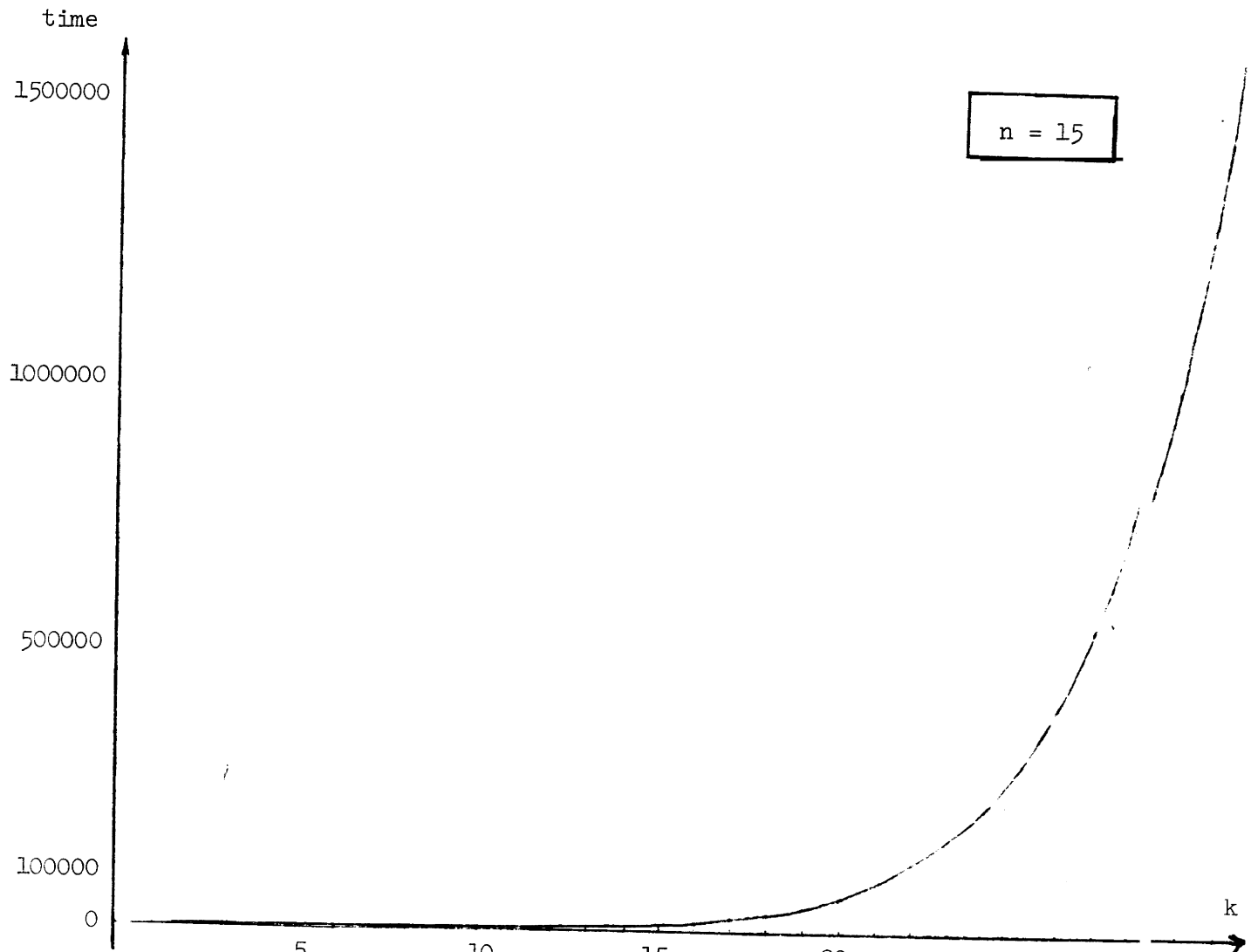
CD = 8.000000 S = 11.00000

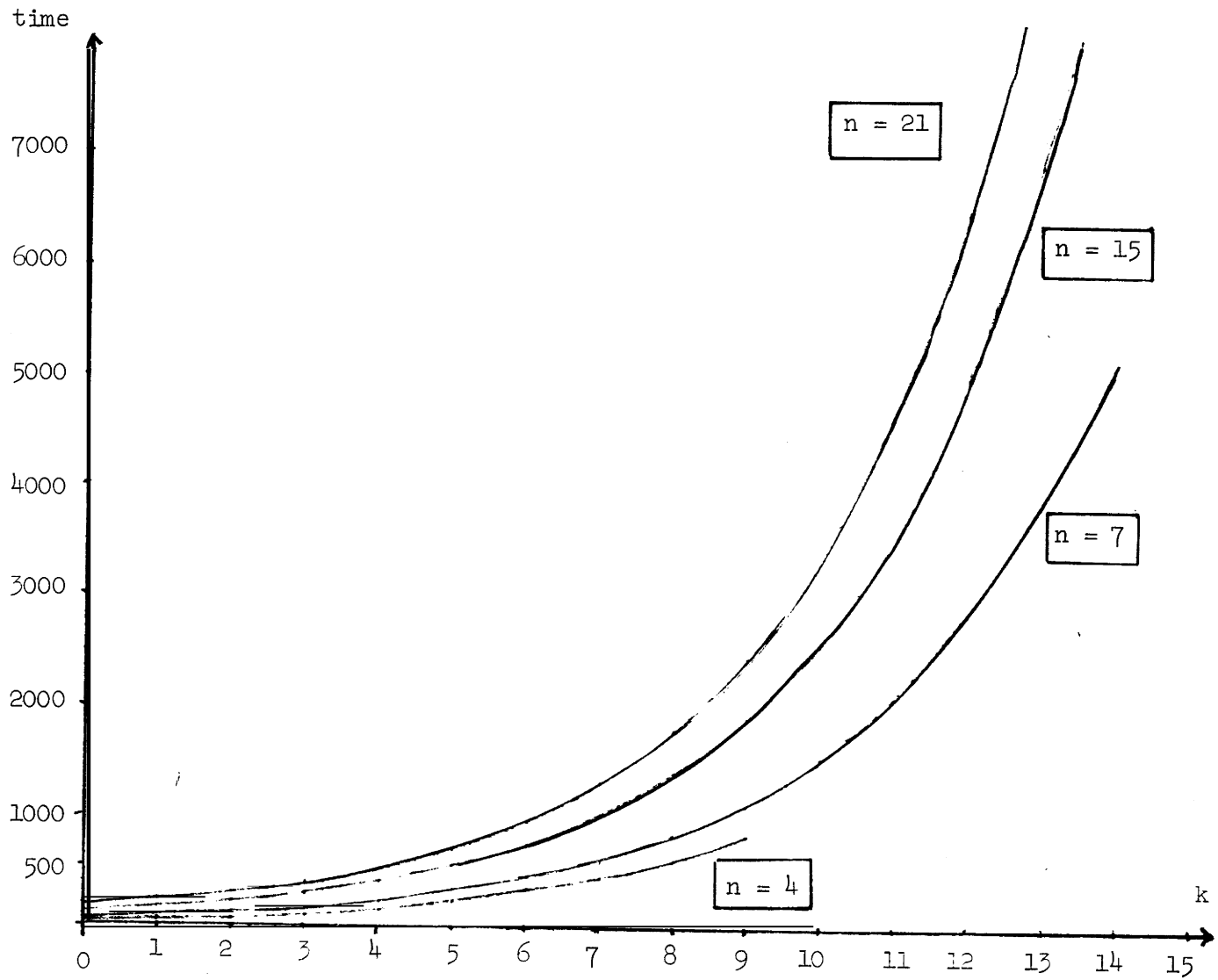
K \ N	11	12	13	14	15	16	17	18	19	20	21
0	96	104	112	120	128	136	144	152	160	168	176
1	129	139	149	159	169	179	189	199	209	219	229
2	174	186	199	211	224	237	249	262	275	287	300
3	235	251	267	283	300	316	332	348	364	381	397
4	319	340	361	382	403	424	445	466	487	508	529
5	435	463	490	518	545	573	600	628	656	683	711
6	595	631	668	705	742	778	815	852	888	925	962
7	815	865	914	963	1013	1062	1112	1161	1210	1260	1309
8	1119	1187	1254	1321	1389	1456	1523	1590	1657	1724	1790
9	1538	1631	1724	1817	1909	2001	2093	2185	2277	2368	2460
10	2114	2244	2374	2503	2631	2759	2886	3013	3140	3265	3392
11	2905	3089	3271	3452	3631	3809	3987	4164	4340	4515	4692
12	3989	4253	4508	4763	5016	5267	5517	5765	6013	6259	6505
13	5469	5843	6210	6573	6932	7288	7641	7992	8341	8688	9034
14	7486	8022	8545	9067	9579	10086	10588	11086	11581	12074	12564
15	10223	10996	11753	12497	13231	13955	14673	15384	16089	16793	17487
16	13925	15040	16132	17203	18257	19298	20326	21345	22354	23356	24351
17	18914	20524	22099	23644	25163	26660	28139	29602	31051	32488	33914
18	25609	27934	30207	32436	34628	36787	38917	41023	43108	45174	47224
19	34561	37709	41188	44406	47568	50683	53757	56794	59799	62775	65726
20	46478	51291	56012	60651	65214	69710	74146	78529	82865	87159	91415
21	62278	69175	75959	82636	89212	95697	102100	108427	114686	120885	127028
22	83139	92564	102708	112292	121755	131099	140333	149464	158500	167452	176325
23	110565	124559	139434	152169	165757	179199	192500	205666	218707	231631	244447
24	0	166270	186003	205615	225076	244372	263501	282463	301264	319913	338417
25	0	221157	249105	277009	304797	332430	359885	387149	414221	441104	467803

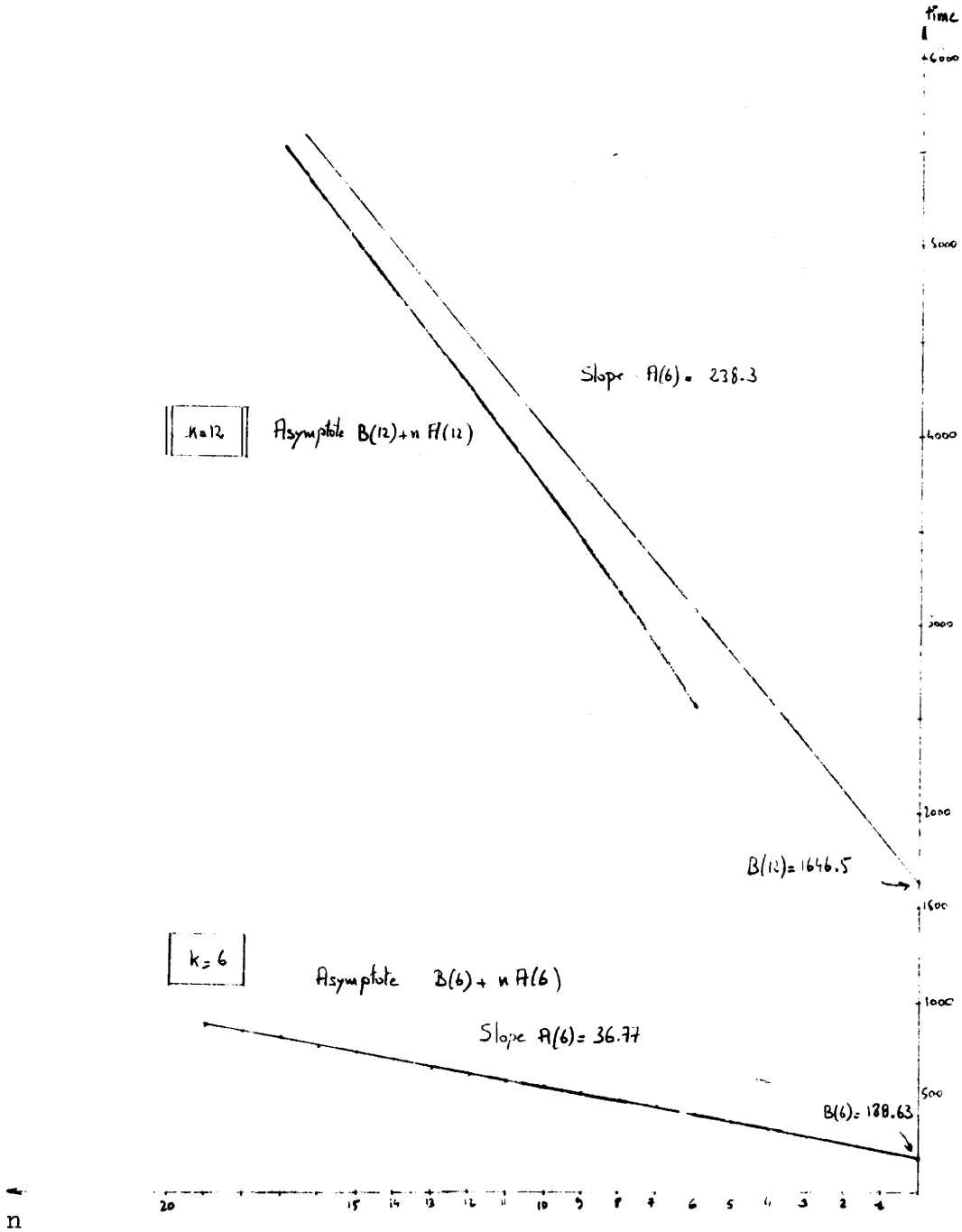
CC = 8.000000 CS = 11.000000

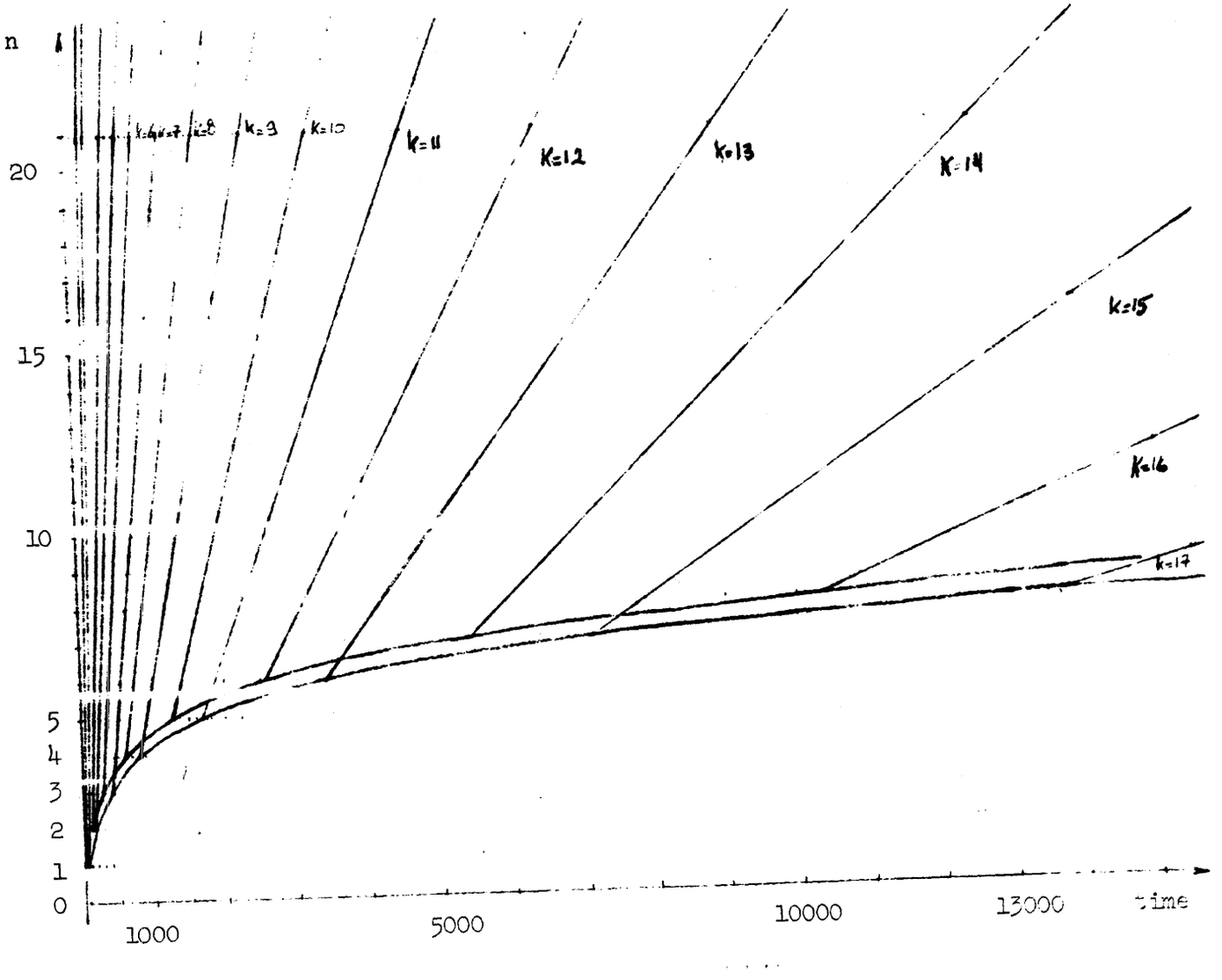
K \ N	11	12	13	14	15	16	17	18	19	20	21
26	0	0	332507	372057	411608	451069	490380	529906	568424	607124	508539
27	0	0	442341	498171	554265	610444	666596	722604	778442	834067	889455
28	0	0	0	664937	744195	823912	903857	983860	1063801	1143599	1223193
29	0	0	0	884709	996260	1108983	1222471	1336417	1450591	1564824	1678993
30	0	0	0	0	1329718	1488535	1649117	1810930	1973570	2136724	2300149
31	0	0	0	0	1769445	1992363	2218807	2447897	2678936	2911378	3144794
32	0	0	0	0	0	2659144	2977343	3300638	3627890	3958178	4290763
33	0	0	0	0	0	3538917	3984445	4439186	4901315	5369335	5842029
34	0	0	0	0	0	0	5317748	5955210	6607773	7267077	7937137
35	0	0	0	0	0	0	7077861	7968380	8881314	9812955	10760203
36	0	0	0	0	0	0	0	10634509	11911410	13219955	14555308
37	0	0	0	0	0	0	0	14155749	15935872	17768128	19545216
38	0	0	0	0	0	0	0	0	21267216	23824736	26455600
39	0	0	0	0	0	0	0	0	28311520	31870176	35546624
40	0	0	0	0	0	0	0	0	0	42531188	47653200
41	0	0	0	0	0	0	0	0	0	56623056	63737632
42	0	0	0	0	0	0	0	0	0	0	85056464
43	0	0	0	0	0	0	0	0	0	0	113246128

000.33 SECONDS IN EXECUTION


time
1500000
1000000
500000
100000
0
n = 15
k
5
10
15
20
25
30


time
7000
6000
5000
4000
3000
2000
1000
500
n = 21
n = 15
n = 7
n = 4
k
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


time
6000
5000
4000
3000
2000
1600
1000
500
k=12
Asymptote B(12)+n A(12)
Slope A(6) = 238.3
B(12)= 1646.5
k=6
Asymptote B(6) + n A(6)
Slope A(6)= 36.77
B(6)= 188.63
20
15
14
13
12
11
10
9
8
7
6
5
4
3
2
1
n


n
20
15
10
5
4
3
2
1
0
k=9
k=10
k=11
K=12
k=13
K=14
k=15
k=16
k=17
1000
5000
10000
13000
time