SIAM J. COMPUT.
Vol. 12, No. 4, November 1983


0097-5397/83/1204-0006 $01.25/0

# ANALYSIS OF EARLY-INSERTION STANDARD COALESCED HASHING*

WEN-CHIN CHEN† AND JEFFREY SCOTT VITTER†‡

**Abstract.** This paper analyzes the *early-insertion* standard coalesced hashing method (EISCH), which is a variant of the standard coalesced hashing algorithm (SCH) described in [Knu73], [Vit80] and [Vit82b]. The analysis answers the open problem posed in [Vit80]. The number of probes per successful search in full tables is 5% better with EISCH than with SCH.

**Key words.** analysis of algorithms, hashing, coalesced hashing, early-insertion, data structures, average-case

**1. Introduction.** One of the well-known data structures for information storage and retrieval is *coalesced hashing*, which was introduced in [Wil59] and analyzed in [Vit80], [Vit82b], [Knu73] and [GK81]. We will assume that each package of information is stored in computer memory as a *record*. There is a special field in each record, called the *key*, that uniquely identifies it. The job of a searching algorithm is to take an input $K$ and return the record (if any) that has $K$ as its key.

For purposes of notation, we let $M'$ denote the number of slots in the hash table. The first $M$ slots, which serve as the range of the hash function, are called the *address region*; the remaining $M'-M$ slots make up the *cellar*. We assume that the pre-defined hash function

$$\text{(1)} \qquad hash: \{\text{all possible keys}\} \to \{1, 2, \cdots, M\}$$

assigns each record to its *hash address* in a random (uniform) manner. We say that a *collision* occurs when the hash address of a record is already occupied, and the record must be inserted elsewhere. The special case in which $M = M'$ and there is no cellar is called *standard coalesced hashing*.

The coalesced hashing method has the property that a record is not moved once it is inserted. The algorithm can be described as follows: Given a record with key $K$, the algorithm searches for it in the hash table, starting at its hash address $hash(K)$ and following the links in the chain. If the record is found, the search is successful; otherwise, the end of the chain is reached and the search is unsuccessful, in which case the record is inserted as follows: If position $hash(K)$ is empty, then the record is stored at that location; otherwise, the record is stored in the largest-numbered empty slot in the table and is linked into the chain that contains slot $hash(K)$ (at some point after slot $hash(K)$). There are two different ways to link that record into the chain. The conventional method is to link the record to the *end* of chain that contains slot $hash(K)$. The second method, which was named *early-insertion* by Gary Knott, inserts the record into the chain *immediately after* slot $hash(K)$ by rerouting pointers. Insertion can be done faster with the early-insertion method when it is known a priori that the record is not already present in the table, since it isn't necessary to search to the end of the chain.

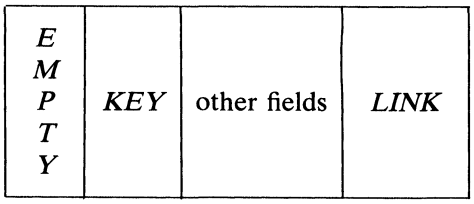
A formal description of the conventional insertion method appears below. Let us assume that each of the $M'$ contiguous slots in the coalesced hash table has the

* Received by the editors March 10, 1982, and in revised form October 5, 1982. This research was supported in part by an IBM Research contract.

† Department of Computer Science, Brown University, Providence, Rhode Island 02912.

‡ The research of this author was supported in part by National Science Foundation grant MCS-81-05324.

following organization:




For each value of $i$ between 1 and $M'$, $EMPTY[i]$ is a one-bit field that denotes whether the $i$th slot is unused, $KEY[i]$ stores the key (if any), and $LINK[i]$ is either the index to the next spot in the chain or else the null value 0.

**Algorithm C** (*conventional coalesced hashing search and insertion*). This algorithm searches an $M'$-slot hash table, looking for a given key $K$. If the search is unsuccessful and the table is not full, then $K$ is inserted.

The size of the address region is $M$; the hash function *hash* returns a value between 1 and $M$ (inclusive). For convenience, we make use of slot 0, which is always empty. The global variable $R$ is used to find an empty space whenever a collision must be stored in the table. Initially, the table is empty, and we have $R = M' + 1$; when an empty space is requested, $R$ is decremented until one is found. We assume that the following initializations have been made before any searches or insertions are performed: $M \leftarrow \lceil \beta M' \rceil$, for some constant $0 < \beta \leqq 1$; $EMPTY[i] \leftarrow$ **true**, for all $0 \leqq i \leqq M'$; and $R \leftarrow M' + 1$.

**C1.** (Hash) Set $i \leftarrow hash(K)$. (Now $1 \leqq i \leqq M$.)

**C2.** (Is there a chain?) If $EMPTY[i]$, then go to step C6. (Otherwise, the $i$th slot is occupied, so we will look at the chain of records that starts there.)

**C3.** (Compare.) If $K = KEY[i]$, the algorithm terminates successfully.

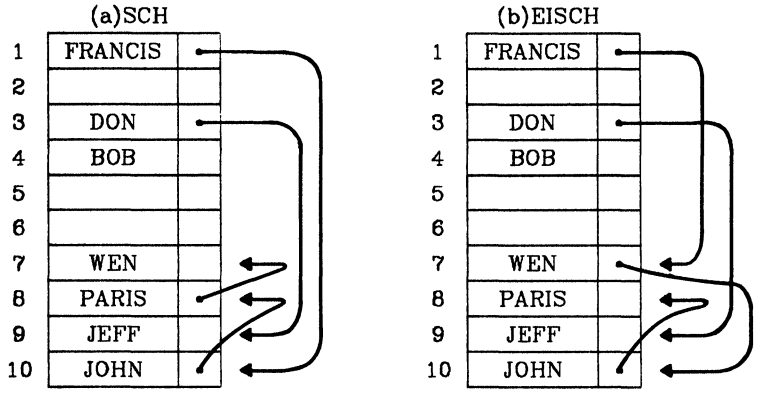
**C4.** (Advance to next record.) If $LINK[i] \neq 0$, then set $i \leftarrow LINK[i]$ and go back to step C3.

**C5.** (Find an empty slot. The search for $K$ in the chain was unsuccessful, so we will try to find an empty table slot to store $K$.) Decrease $R$ one or more times until $EMPTY[R]$ becomes **true**. If $R = 0$, then there are no more empty slots, and the algorithm terminates *with overflow*. Otherwise, append the $R$th cell to the chain by setting $LINK[i] \leftarrow R$; then set $i \leftarrow R$.

**C6.** (Insert new record.) Set $EMPTY[i] \leftarrow$ **false**, $KEY[i] \leftarrow K$, $LINK[i] \leftarrow 0$, and initialize the other fields in the record.

The early-insertion method can be implemented by the following two modifications: First, we add the assignment "Set $j \leftarrow i$" at the end of step C2, so that $j$ stores the hash address $hash(K)$. The second modification replaces the last sentence of step C5 by "Otherwise, link the $R$th cell into the chain immediately after the hash address $j$ by setting $LINK[R] \leftarrow LINK[j]$, $LINK[j] \leftarrow R$; then set $i \leftarrow R$."

An example of the two methods is given in Fig. 1. The record WEN collides with FRANCIS at slot 1. With the conventional insertion method pictured in Fig. 1(a), WEN is linked to the end of the chain containing FRANCIS, whereas in the early-insertion method in Fig. 1(b), WEN is inserted into the chain at the point between FRANCIS and JOHN. The average successful search time in Fig. 1(b) is slightly better than in Fig. 1(a), because inserting WEN immediately after FRANCIS (rather than at the end of the chain) reduces the search time for WEN from four probes to two




| Keys: | FRANCIS | DON | JOHN | BOB | JEFF | PARIS | WEN |
|---|---|---|---|---|---|---|---|
| Hash Addresses: | 1 | 3 | 1 | 4 | 3 | 10 | 1 |

ave. # probes per successful search: (a) 13/7 ≈ 1.86, (b) 12/7 ≈ 1.71.
ave. # probes per unsuccessful search: (a) 17/10 = 1.7, (b) 17/10 = 1.7.

FIG. 1. *Standard coalesced hashing,* $M' = M = 10$, $N = 7$. (a) SCH, (b) EISCH.

and increases the search time for JOHN from two probes to three. That results in a net decrease of one probe.

The analyses of coalesced hashing that have appeared in the literature have concentrated on the conventional method, in which a record is always linked to the end of the chain. Knuth [Knu73] analyzes the special case of standard coalesced hashing (SCH), in which $M = M'$ and there is no cellar. Vitter [Vit82b] analyzes the more general coalesced hashing method (CH), for which the cellar may be nonempty.

In this paper, we derive exact formulas for the average number of probes per search for the early-insertion standard coalesced hashing method (EISCH). This solves the open problem posed in [Vit80]. The average unsuccessful search times for the EISCH and SCH methods are the same, but the average successful search time is up to 5% better with EISCH than with SCH. The performance of early-insertion method when there is a cellar (EICH) is still unknown.

**2. The main result.** In this section we develop the probability model used in our analysis of early-insertion standard coalesced hashing EISCH and we state our main result, Theorem 1, which expresses the average number of probes per successful search for EISCH. We use the following parameters in our analysis:

$N$ = the number of inserted records,

$M'$ = the number of slots in the hash table,

$\alpha = N/M'$ = the load factor.

These quantities satisfy $0 \leqq N \leqq M'$ and $0 \leqq \alpha \leqq 1$. Since there is no cellar in the EISCH method, the address region size $M$ is equal to the table size $M'$.

In the average-case analysis, we assume that an unsuccessful search can begin at any of the $M$ slots in the address region with equal probability. This includes the special case of insertion. Similarly, each record in the hash table has the same chance of being the object of any given successful search. In other words, all searches and

insertions involve random keys. This model can be formalized by the following definitions.

DEFINITION 1. The sequence $a_1a_2 \cdots a_N$, where $1 \leqq a_i \leqq M$, is called a *hash sequence*. Every such sequence represents the insertion of $N$ records into a hash table of address size $M$; element $a_j$ denotes the hash address of the $j$th inserted record (i.e., *hash*(key of $j$th record) $= a_j$).

In our analysis, we will use the number of probes per search, i.e., the number of slots traversed, as a measure of search performance.

DEFINITION 2. We let $P'_N$ and $P_N$ denote the random variables describing the number of probes in unsuccessful and successful searches in a SCH table containing $N$ records. Similarly, we let $\bar{P}'_N$ and $\bar{P}_N$ be the random variables describing the number of probes in unsuccessful and successful searches in a EISCH table containing $N$ records.

The sample space for $P'_N$ and $\bar{P}'_N$ is

$$S' = \{[a_1, a_2, \cdots, a_N; a] \mid 1 \leqq a_j \leqq M, 1 \leqq a \leqq M\}, \tag{2}$$

where $a_1a_2 \cdots a_N$ represents the hash sequence of the $N$ inserted records, and $a$ is the starting address of the unsuccessful search. The $M^{N+1}$ elements in $S'$ each have probability $1/M^{N+1}$. The values of $P'_N$ and $\bar{P}'_N$ at sample point $[a_1, a_2, \cdots, a_N; a] \in S'$ are denoted by $P'_N[a_1, a_2, \cdots, a_N; a]$ and $\bar{P}'_N[a_1, a_2, \cdots, a_N; a]$.

Similarly, the sample space for $P_N$ and $\bar{P}_N$ is

$$S = \{[a_1, a_2, \cdots, a_N; n] \mid 1 \leqq a_j \leqq M, 1 \leqq n \leqq N\}, \tag{3}$$

where $a_1a_2 \cdots a_N$ represents the hash sequence of the $N$ inserted records, and $n$ specifies that the $n$th inserted record is the object of the successful search. The $NM^N$ elements in $S$ each have probability $1/NM^N$. The values of $P_N$ and $\bar{P}_N$ at sample point $[a_1, a_2, \cdots, a_N; n] \in S$ are denoted by $P_N[a_1, a_2, \cdots, a_N; n]$ and $\bar{P}_N[a_1, a_2, \cdots a_N; n]$.

DEFINITION 3. For SCH, the average unsuccessful and successful search times are defined by

$$C'_N = \frac{1}{M^{N+1}} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq a \leqq M}} P'_N[a_1, a_2, \cdots, a_N; a], \tag{4}$$

$$C_N = \frac{1}{NM^N} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq n \leqq N}} P_N[a_1, a_2, \cdots, a_N; n]. \tag{5}$$

For EISCH, the average unsuccessful and successful search times are defined by

$$\bar{C}'_N = \frac{1}{M^{N+1}} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq a \leqq M}} \bar{P}'_N[a_1, a_2, \cdots, a_N; a], \tag{6}$$

$$\bar{C}_N = \frac{1}{NM^N} \sum_{\substack{M^N \text{ hash sequences} \\ 1 \leqq n \leqq N}} \bar{P}_N[a_1, a_2, \cdots, a_N; n]. \tag{7}$$

Knuth [Knu73] analyzes the standard coalesced hashing method SCH, and derives the following unsuccessful and successful search times, as functions of loading factor

$\alpha = N/M$:

$$C'_N = 1+\frac{1}{4}\left(\left(1+\frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right) \approx 1+\frac{1}{4}(e^{2\alpha}-1-2\alpha), \tag{8}$$

$$\begin{aligned} C_N &= 1+\frac{1}{8}\frac{M}{N}\left(\left(1+\frac{2}{M}\right)^N - 1 - \frac{2N}{M}\right)+\frac{1}{4}\frac{N-1}{M} \\ &\approx 1+\frac{1}{8\alpha}(e^{2\alpha}-1-2\alpha)+\frac{1}{4}\alpha. \end{aligned} \tag{9}$$

In particular, we have $C'_M \approx 2.10$ and $C_M \approx 1.80$, when the table is full (i.e., when load factor $\alpha = 1$).

As Fig. 1 shows, the chains in SCH and EISCH contain the same elements, possibly in different orders. Since the average time for an unsuccessful search depends only on the length of the chains and not on the order of keys within the chains, the average unsuccessful search times $C'_N$ and $\bar{C}'_N$ for SCH and EISCH are equal.

The following theorem is the main result of this paper. It gives the average successful search time for EISCH, using the probability model described in Definitions 1–3.

THEOREM 1. *In an M-slot early-insertion standard coalesced hash table containing N inserted records, the average number of probes in a random successful search is*

$$\bar{C}_N = \frac{M}{N}\left(1+\frac{1}{M}\right)^N - \frac{M}{N}. \tag{10}$$

*In asymptotic form, this can be expressed as*

$$\bar{C}_N \approx \frac{1}{\alpha}(e^{\alpha}-1), \tag{11}$$

*where $M, N \to \infty$ and $\alpha = N/M$ remains constant. The approximation error is roughly $e^{\alpha}/(2M)$.*

The graphs in Fig. 2 reflect the fact that EISCH is slightly better than SCH. For example, when the table is full (i.e., load factor $\alpha = 1$), we have $C_N \approx 1.80$ and $\bar{C}_N \approx 1.72$, so EISCH is about 5% faster. The difference between the expected number of probes per successful search for SCH and EISCH is not significant when $\alpha$ is small. When the table is half full (i.e., load factor $\alpha = .5$), we have $C_N \approx 1.31$ and $\bar{C}_N \approx 1.30$, which is only a 0.5% improvement. One reason for this is that search times improve only when searching for records contained in chains of length greater than 3; when the load factor is small, the chains are usually short.

**3. Proof of the theorem.** The derivation of the successful search time for EISCH is more difficult than the analysis of SCH for the following reason: Any coalesced hash table has the property that the hash address of the $k$th record in a chain ($k > 1$) must be the location of one of the $k-1$ predecessors in the chain. In a random SCH table, each of the $k-1$ locations can be the hash address of the $k$th record with equal probability $1/(k-1)$. However, that is not true in a random EISCH table, as Fig. 3 illustrates. The hash address of the fourth ($k = 4$) record in a random chain of length 4 is the location of the first record in the chain with probability 2/6, it is the location of the second record with probability 1/6, and it is the location of the third record

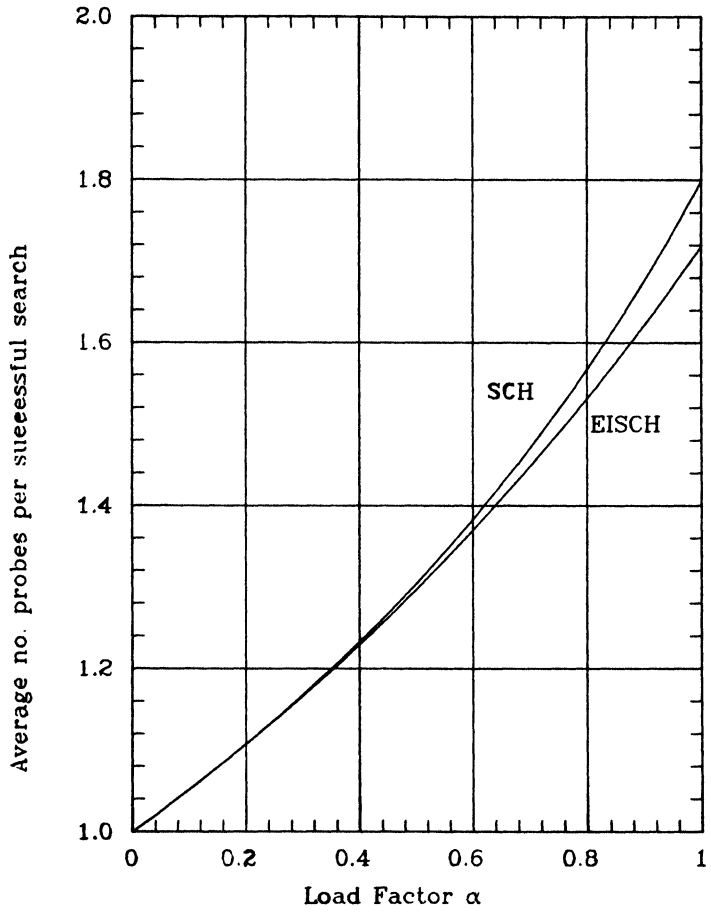



FIG. 2. *The average number of probes per successful search for* EISCH *and* SCH.

with probability 3/6. Our analysis is more difficult because it must take this into account.

For notational simplicity, the parameter $M$, which denotes the number of slots in the hash table, will not be written, but rather shall be implicit in all the terms defined in this section. In order to prove Theorem 1, let us start with some definitions.

DEFINITION 4. We let $A_N$ denote the quantity $NM^N\bar{C}_N$, where $\bar{C}_N$ is the average successful search time defined in Definition 3.

DEFINITION 5. For $1 \leqq i \leqq j \leqq l$, we let $c_N(l, i, j)$ denote the number of chains among all the $M^N$ hash tables that satisfy the following two properties:

(1) the length of the chain is $l$;

(2) the hash address of the $j$th record in the chain is the location of one of the first $i$ records in the chain.

We should note that $c_N(l, i, j) = 0$ for $l > N$. For $j = i$ or $j = i + 1$, the term $c_N(l, i, j)$ is equal to the total number of chains of length $l$ among all the $M^N$ hash tables, since every chain has the property that the hash address of the $j$th record in the chain ($j > 1$) must be the location of a preceding record.

From the above remarks we have the following lemma.

| Sequence of hash addresses | Order of the keys in the chain | Key that occupies the hash address of the last key in the chain | Relative position in the chain of that key |
|---|---|---|---|
| 1111 | adcb | a | 1 |
| 1141 | adbc | b | 3 |
| 1144 | abdc | b | 2 |
| 1114 | acbd | b | 3 |
| 1113 | acdb | a | 1 |
| 1143 | abcd | c | 3 |

FIG. 3. *The keys a, b, c and d are inserted (in that order) into an* EISCH *table containing* $M' = 4$ *slots. This table describes all six possible chains of length* 4 *in an* EISCH *table in which the keys a, b, c and d are stored in locations* 1, 4, 3 *and* 2, *respectively. The last column shows that the hash address of the last record in the chain is more likely to be the position of the third record in the chain rather than that of the second or the first.*

LEMMA 1. *The summation* $\sum_{1\leq l\leq N, 1\leq i\leq l} c_N(l, i, i)$ *is equal to* $NM^N$.

The following lemma expresses the quantity $A_N$ we want to evaluate in terms of $c_N(l, i, j)$.

LEMMA 2. *The term* $A_N$ *is equal to* $NM^N + B_N$, *where* $B_N = \sum_{1\leq l\leq N, 1\leq i<j\leq l} c_N(l, i, j)$.

*Proof.* First we will show that $A_N = \sum_{1\leq l\leq N, 1\leq i\leq j\leq l} c_N(l, i, j)$. A search for the $j$th record in a chain of length $l$ requires $j-i+1$ probes if the hash address of the $j$th record is the location of the $i$th record of the chain. Thus, searching for the $j$th record contributes $j-i+1$ to $A_N$, which, by definition, is the sum of the contributions of the $NM^N$ searches among all the $M^N$ hash tables. The search for the $j$th record contributes 1 to each of the following $j-i+1$ terms: $c_N(l, i, j), c_N(l, i+1, j), \cdots, c_N(l, j, j)$. Hence, we have

$$A_N = \sum_{\substack{1\leq l\leq N \\ 1\leq i\leq j\leq l}} c_N(l, i, j) = \sum_{\substack{1\leq l\leq N \\ 1\leq i=j\leq l}} c_N(l, i, j) + \sum_{\substack{1\leq l\leq N \\ 1\leq i<j\leq l}} c_N(l, i, j).$$

The first summation is equal to $NM^N$, by Lemma 1. The second summation is the definition of $B_N$. □

To evaluate $B_N$, we need the following recurrence.

LEMMA 3. *The term* $c_N(l, i, j)$ *defined in Definition* 5 *satisfies the recurrence*

$$\begin{aligned} c_{N+1}(l, i, j) &= (M-l)c_N(l, i, j) + (i-1)c_N(l-1, i-1, j-1) \\ &\quad + (j-i-1)c_N(l-1, i, j-1) \\ &\quad + (l-j)c_N(l-1, i, j) + \delta_{j=i+1}c_N(l-1, i, i) \end{aligned} \tag{12}$$

*for* $1\leq N\leq M-1$, $1\leq i<j\leq l\leq N+1$. *The notation* $\delta_R$ *denotes* 1 *if the relation* $R$ *is true and* 0 *otherwise.*

*Proof.* Let's consider the $M^N$ distinct hash sequences for $N$ inserted records. The only chains that can contribute to $c_{N+1}(l, i, j)$ after the $(N+1)$st insertion are chains that contribute to $c_N(l, i, j)$, $c_N(l-1, i-1, j-1)$, $c_N(l-1, i, j-1)$, $c_N(l-1, i, j)$ and $c_N(l-1, i, i)$.

When $l<N+1$, a chain that contributes to $c_N(l, i, j)$ after $N$ insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of the $M-l$ records outside the chain. This accounts for the $(M-l)c_N(l, i, j)$ term.

When $i>1$, a chain that contributes to $c_N(l-1, i-1, j-1)$ after $N$ insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted

record is the location of one of the first $i-1$ records in the chain. This accounts for the $(i-1)c_N(l-1, i-1, j-1)$ term.

When $j > i+1$, a chain that contributes to $c_N(l-1, i, j-1)$ after $N$ insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of the records between the $i$th and $(j-2)$nd records, inclusively, in the chain. This accounts for the $(j-i-1)c_N(l-1, i, j-1)$ term.

When $j < l$, a chain that contributes to $c_N(l-1, i, j)$ after $N$ insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of one of that last $l-j$ records in the chain. This accounts for the $(l-j)c_N(l-1, i, j)$ term.

When $j = i+1$, any chain of length $l-1$ after $N$ insertions will contribute to $c_{N+1}(l, i, j)$ after the next insertion if the hash address of the inserted record is the location of the $i$th record in the chain. This accounts for the $\delta_{j=i+1}(l-1, i, i)$ term. □

LEMMA 4. *The term $B_N$ defined in Lemma 3 is equal to $M(M+1)^N - (M+N)M^N$, for $1 \leqq N \leqq M$.*

*Proof.* Substituting Lemma 3 into $B_N$, we have, for $1 \leqq N \leqq M-1$,

$$
\begin{aligned}
B_{N+1} &= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} c_{N+1}(l, i, j) \\
&= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (i-1)c_N(l-1, i-1, j-1) \\
&\quad + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (j-i-1)c_N(l-1, i, j-1) + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (l-j)c_N(l-1, i, j) \\
&\quad + \sum_{\substack{2\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} \delta_{j=i+1}c_N(l-1, i, i) \\
&= \sum_{\substack{1\leqq l\leqq N+1\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 0\leqq i<j\leqq l}} ic_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq j\leqq l}} (j-i)c_N(l, i, j) \\
&\quad + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l+1}} (l+1-j)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq j\leqq l}} \delta_{j=i}c_N(l, i, i) \\
&= \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (M-l)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} ic_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (j-i)c_N(l, i, j) \\
&\quad + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} (l+1-j)c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i=j\leqq l}} c_N(l, i, i) \\
&= (M+1) \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i<j\leqq l}} c_N(l, i, j) + \sum_{\substack{1\leqq l\leqq N\\ 1\leqq i\leqq l}} c_N(l, i, i) \\
&= (M+1)B_N + NM^N.
\end{aligned}
$$

The last step follows from the definition of $B_N$ and from Lemma 1. By telescoping and by the fact that $B_1 = 0$, we get

$$B_N = M(M+1)^N - (M+N)M^N. \qquad \square$$

Now, we are finally ready to prove Theorem 1. Combining Lemma 2 and Lemma 4, we have

$$A_N = NM^N + B_N = M(M+1)^N - M^{N+1}.$$

Thus, we get

$$\bar{C}_N = \frac{A_N}{NM^N} = \frac{M}{N}\left(1+\frac{1}{M}\right)^N - \frac{M}{N}. \tag{13}$$

We can express this in asymptotic form as

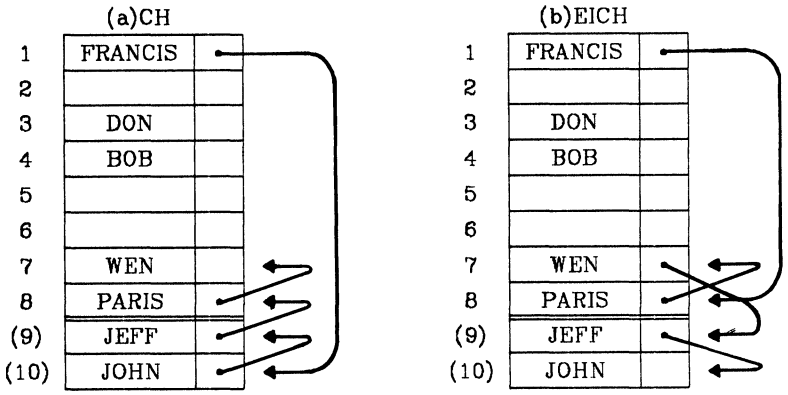
$$\bar{C}_N \approx \frac{1}{\alpha}(e^\alpha - 1), \tag{14}$$

where $N, M \to \infty$ and $\alpha = N/M$ remains constant in the range $0 \leqq \alpha \leqq 1$. The error term is approximately $e^\alpha/(2M)$.

**4. Conclusions and open problems.** We have analyzed the early-insertion standard coalesced hashing method (EISCH) and have shown that this method is slightly better than the standard coalesced hashing method (SCH). The average unsuccessful search times for these two methods are the same, but the average successful search time for EISCH is 5% better when the table is full.

Some interesting open problems concerning early-insertion remain unsolved. The early-insertion method can be used when there is a cellar and $M < M'$. We call this generalization EICH. The search performance of EICH is still unanalyzed. However, we conjecture that EICH is inferior to the coalesced hashing method (CH), since in EICH a chain's records that are stored in the cellar come at the *end* of the chain, whereas in CH they come *immediately* after the first record in the chain. In Fig. 4(b), the insertion of WEN causes both the cellar records JOHN and JEFF to move one link further from their hash addresses. That doesn't happen in Fig. 4(a).

Many hashing applications require that certain records be inserted and then later deleted. The best deletion algorithms are the ones that *preserve randomness*, because deleting a record is in some sense like never having inserted it. In particular, the formulas for the average search times after $N$ random insertions intermixed with $d$ deletions are the same as the formulas for the average search times after $N-d$ random insertions. The formal notion of what it means to preserve randomness is defined in [Vit82a].




| Keys: | FRANCIS | DON | JOHN | BOB | JEFF | PARIS | WEN |
|---|---|---|---|---|---|---|---|
| Hash Addresses: | 1 | 3 | 1 | 4 | 1 | 1 | 8 |

ave. # probes per successful search: (a) 14/7 ≈ 2.0, (b) 16/7 ≈ 2.286.
ave. # probes per unsuccessful search: (a) 13/8 = 1.625, (b) 17/8 = 2.125.

FIG. 4. *Coalesced hashing, $M' = 10$, $M = 8$, $N = 7$.* (a) CH, (b) EICH.

A deletion algorithm for coalesced hashing is given [Vit82a] and shown to preserve randomness for standard coalesced hashing (SCH). To delete a record, the algorithm removes it from the table, and then it repeatedly reinserts the record in the remainder of the chain, if any. There is a modification of this deletion algorithm that performs the reinsertions using the early-insertion idea. It does not preserve randomness for SCH, but it may possibly be "better-than-random." An interesting question is how this modified deletion algorithm (which uses early-insertion) affects the search times for EISCH. It does not preserve randomness, but it may possibly be better-than-random. There is currently no known deletion algorithm that preserves randomness for the EISCH, EICH and CH methods.

**Addendum.** Since the time this article was submitted, the authors have solved the first open problem listed in § 4. The analysis of the search time of early-insertion coalesced hashing for the general case (when there is a cellar) appears in *Analysis of some new variants of coalesced hashing*, Department of Computer Science, Brown University, Providence, RI, Technical Report No. CS-82-18, June 1982. That report also describes and analyzes a new method called varied-insertion coalesced hashing that performs better than both the unmodified method and the early-insertion method.

## REFERENCES

[GK81] D. H. GREENE AND D. E. KNUTH, *Mathematics for the Analysis of Algorithms.* Birkhauser, Boston, 1981.

[Knu73] D. E. KNUTH, *The Art of Computer Programming. Volume* 3: *Sorting and Searching.* Addison-Wesley, Reading, MA, 1973.

[Vit80] J. S. VITTER, *Analysis of coalesced hashing*, PhD dissertation, Department of Computer Science, STAN-CS-80-817 Tech. Rep. Stanford University, Stanford, CA, August 1980.

[Vit82a] ———, *Deletion algorithms for hashing that preserve randomness.* J. Algorithms, 3 (1982), pp. 261–275.

[Vit82b] ———, *Analysis of the search performance of coalesced hashing*, J. Assoc. Comput. Mach., 30 (April 1983).

[Wil59] F. A. WILLIAMS, *Handling identifiers as internal symbols in language processors*, Comm. ACM, 2 (1959), pp. 21–24.