H A S H I N G

The search time of each sorting algorithm depends on the number of elements (n) in the collection of data S. there is a searching technique, called *HASHING* or *HASH ADDRESSING* which is independent of the number of elements (n).

Definition: Let us assume that there is a file **F** of **n** records with a set **K** of keys, which uniquely determine the records in **F**. Again we assume that **F** is maintained in memory by a table **T** of **m** memory locations and that **L** is the set of memory addresses of the location in **T**. **H** is a function from the set **K** of keys into the set **L** of memory addresses. Such a function,

$$\mathbf{H : K \rightarrow L}$$

is called a hash function or hashing function.

DIFFERENT HAS FUNCTIONS

The two principal criteria used in selecting a hash function (H : K → L) are as follows. First of all, the function H should be very easy to compute. Second the function H should, as far as possible, uniformly distribute the hash addresses throughout the set L so that there are a minimum number of collisions. Naturally, there is no guarantee that the second condition can be completely fulfilled without actually knowing beforehand the keys and addresses.

Now we discuss three popular hash functions:

1. DIVISION METHOD:

Choose a primary number or a number without small divisor - **x** larger than the number **n** of keys in **K**. then the hash function can be defined as:

$$\mathbf{H (k) = k (} \textit{mod} \mathbf{ x) + starting\ address}$$

where **k** is any key from **K**.

Example: Suppose that there are n = 60 keys available for m = 100 memory locations ranging from 0 to 99. Let k = 3271, now we choose x = 97. Hence the required address will be 3271 (*mod* 97) + 0= 33.

2. MIDSQUARE METHOD:

The key **k** is squared. Then the hash function is defined by,

$$\mathbf{H (k) = l}$$

where **l** is obtained by deleting digits from both ends of $\mathbf{k^2}$**.** We emphasize that the same position of $\mathbf{k^2}$ must be used for all of the keys.

Exm: Suppose that a **q-digit address** is to be generated from a **p-digit key**. To do so, the p-digit key is first squared to yield a **2p**-digit value. The q-digit address, then, is

selected from the middle of this 2p-digit result. For example, let p = 4, q = 3 and key be “3271”. Square of “3271” is “10699441”. Since 3-digit address are to be extracted from the middle, the first 3 and the last 2 digits can be removed and “994” may be selected as the address. The method can be modified to select “699” as the address.

3. FOLDING METHOD:
The key **k** is partitioned into a number of parts, $\mathbf{k_1}$, $\mathbf{k_2}$, **…** , $\mathbf{k_r}$, where each part, except possible the last, has the same number of digits as the required address. Then the even numbered parts, $\mathbf{k_2}$, $\mathbf{k_4}$, **…** are each reversed and all parts added together, ignoring the last carry. That is,

$$\mathbf{H (k) = k_1 + k_2 + \ldots + k_r}$$

Exm: Suppose we have an 8-digit key as “39427827” and we have to generate a 3-digit address from it. When we partitioned from right to left, we have three groups as shown below:

039 / 427 / 828

reversing even numbered parts:

039 / 724 / 828

summation of the above parts: 1591. Discarding the carry, that is selecting the last three digits we get the desired address as “591”.

In case of the keys with characters instead of digits, this can be converted into a number consisting their corresponding ASCII values. For example, let the key value be “CS1958”. Then the ASCII values for characters we get k = “67831958”. Then the partitioned values with reversed even numbered parts we get:

67 / 138 / 958

Now, adding the groups (1163) and discarding the carry we get the following 3-digits address as “163”.

Example: *Consider a company where there are 68 employees and each employee is assigned a unique 4-digit employee-number. Suppose the memory addresses are 00, 01, 02, **…** , 99. Apply Division, Mid-square and Folding hash functions to each of the following employee-number: 3205, 7148, 2345.*

Solution:
[N. B. In case of the storage locations starting from an address “x” then the addresses computed by the hash functions are to be added to “x” to get the effective address.]

1. Using Division method:
Let us choose a prime number m = 97 (close to 99)
Then H (3205) + 0 = 4, h (7148) + 0 = 67, H (2345) + 0 = 17

2. Using Mid-square method:
The following calculations are performed:

k	3205	7148	2345
k^2	*102* 72 025	*510* 93 904	*54* 99 025
H(k)	72	93	99

Here the forth and the fifth digits counting from the right are chosen as the hash addresses.

3. Using Folding method:
Chopping the key "k" into two parts, each containing two digits, reversing even numbered parts from left to right, and by adding parts we get the required underlined addresses as follows:

H (3205) = 32 + 50 = 82
H (7148) = 71 + 84 = 1 55
H (2345) = 23 + 54 = 77

COLLITION AND LOAD FACTOR

Suppose we want to add a new record **R** with key **k** to a file **F**, but suppose the memory location addressing **H(k)** is already occupied. This situation is called collision. The procedure of resolving collisions that are chosen depends on many factors. One important factor is the ratio of the number **n** of keys in **K** (which is the number of records in the actual file **F**) to the number **m** of hash addresses in **L**. this ratio, **$\lambda = n / m$** is called the load-factor. Collision occurs when **λ** tends to **1**.

It can be proved that collision is almost impossible to avoid although **λ** is very small. For example, suppose a class of students has 24 heads and say the table has space for 390 records. One random hash function is to choose the students' date of birth as its input to generate hash addresses. Although the load factor **$\lambda = 24 / 390 = 6.15\%$** - is very small, it can easily be observed that there is a better than fifty – fifty chances that two of the students may have same birth date. Hence, quality of the key plays an important role in probability of collision.

EFFICIENCY OF HASH FUNCTION

The efficiency of a hash function with collision resolution technique is measured by the average number of **probes** (*key comparisons*) required to find the locations for a record with a given key **k**. the efficiency depends mainly on the load factor **λ**. Specifically we are interested in the following two quantities:

S (λ) = *Average number of probes for a successful search.*

U (λ) = *Average number of probes for a unsuccessful search.*

CONFLICT RESOLUTION TECHNIQUE

Every hash function is expected to generate the same address for distinct key values. Key values, which are transformed to the same addresses, are referred to as

synonyms. Therefore, one must know how to cope with such situations. Methods to deal with synonyms are known as Collision Resolution Techniques. These techniques fall into two broad classes called **Open Addressing** and **Chaining**. There are several methods in each class.

If a conflict occurs while inserting a key and if the conflict is resolved using *open addressing* method then an empty position is found out within the hash table itself and the new key is inserted in the empty position. So, in *open addressing method*, no space outside the hash table is used for storing the key value.

In chaining method, synonyms are placed in linked lists. Nodes of the linked lists may be allocated from some space outside the hash table.

OPEN ADDRESSING

In this method, if the hash address is found to be already occupied by an element then next addresses following the computed one are sequentially examined in a specific order to locate the first empty position. The concerned element is stored in the empty address location. For the purpose of sequential examination, the hash table is considered to be *circular*. That is, the first address in the hash table is taken as the position next to the last address.

There are three types of popular open addressing methods as follows:

A > LINEAR PROBING: Suppose that a new record **R** with key **k** is to be added to the memory table **T**, but that memory location with hash address **H (k) = h** is already filled. One natural way to resolve the collision is achieved by *linear probing*, in which, we assign **R** to the first available location following **T [h]**. We assume that the table **T** with **m** locations is circular, so that **T [1]** comes after **T [m]**. Accordingly, with such a collision resolution procedure, we will search for the record **R** in the table **T** by linearly searching the locations **T [h], T [h + 1], T [h + 2], … …** until finding **R** or meeting an empty location, which indicates the unsuccessful search.

The above collision resolution is called **linear probing**. The average number of probes for a successful search and for an unsuccessful search can be computed by the following respective quantities:

$$S(\lambda) = \tfrac{1}{2}\left(1 + 1/(1-\lambda)\right)$$

$$U(\lambda) = \tfrac{1}{2}\left(1 + 1/(1-\lambda)^2\right)$$

Where **($\lambda = n / m$)** is the load factor.

Example of Linear Probing with the calculation of successful and unsuccessful prods:

Suppose the table **T** has 11 memory locations: T[1], T[2], **…** ,T[11], and suppose the file **F** consists of 8 records, A, B, C, D, E, X, Y, Z with the following hash addresses:

record	A	B	C	D	E	X	Y	Z
H(k)	4	8	2	11	4	11	5	1

Suppose the 8 records are entered into the table **T** in the above order. Then the file **F** will appear in memory as follows:

Address →	01	02	03	04	05	06	07	08	09	10	11
Table T	X	C	Z	A	E	Y	-	B	-	-	D

[N. B. Although Y is the only record with address H(k) = 5, the record is not assigned to T[5], since T[5] is already been filled by E because of the previous collision at T[4] does not appear in T[1]. **]**

The number of probes for successful search for:

Record A is 1
Record B is 1
Record C is 1
Record D is 1
Record E is 2
Record X is 2
Record Y is 2
Record Z is 3

13

Hence, the average number of probes for the successful searches:

$$S = (13 / 8) = 1.6 \text{ (approx.)}$$

Now the number of probes for unsuccessful searches for a record, which is not available in the list:

With h = 01 is 7
With h = 02 is 6
With h = 03 is 5
With h = 04 is 4
With h = 05 is 3
With h = 06 is 2
With h = 07 is 1
With h = 08 is 2
With h = 09 is 1
With h = 10 is 1
With h = 11 is 8

40

Hence, the average number of probes for the unsuccessful searches:

$$U = (40 / 11) = 3.6 \text{ (approx.)}$$

[N. B. The first sum adds the number of key comparisons (probes) to find each of the 8 records, and the second sum adds the number of key comparisons (probes) to find an empty location for each of the 11 locations.

But theoretically, we have m = 11 and n = 8. Hence, $\lambda = 11 / 8$

S (λ) = 2.3 (calculated approximately according to the formula)

U (λ) = 7.2 (calculated approximately according to the formula) **]**

Disadvantage of linear probing: One major disadvantage of linear probing is that records tends to cluster, that they appear next to one another, when the load factor is greater than 50%. Such a clustering substantially increases the average searching time for a record. To overcome this problem, the following open addressing mechanisms are effective.

B> QUARDRATIC PROBING: Suppose a record **R** with key **k** has the hash address **H (k) = h**. Then, instead of searching the location with addresses **h, h + 1, h + 2, … …** we linearly search the location with addresses as follows:

h, (h + 1), (h + 4), (h + 9), … … … , (h + i^2), … … …

If there are **m** locations in the table **T**, which is a prime number, then the above sequence will access half of the locations in **T**.

C> DOUBLE ADDRESSING: Here a second hash function **H'** is used for resolving a collision as follows, suppose a record **R** with key **k** has the hash address **H (k) = h** and **H'(k) = h' ≠ m**. Then we linearly search the locations with addresses:

h, (h + h'), (h + 2h'), (h + 3h'), … … …

If **m** is a prime number, then the above sequence will access all the locations in the table **T**.

Disadvantage of open addressing: One major disadvantage in any type of open addressing procedure is in the implementation of deletion. Specifically suppose a record **R** is deleted from the location **T [r]**. Afterward, suppose we met **T [r]** while searching for another record **R'**. This does not necessarily mean that the search is unsuccessful. There is a solution to overcome this problem, when deletion the record **R**, we must label the location **T [r]** to indicate that it previously did contain a record. Accordingly, open addressing may seldom be used when a file **F** is constantly changing.

CHAINNING METHOD

This process involves in maintaining two tables in memory. First of all, as before, there is a table **T** in memory which contains the records in **F**, except that **T** now has an additional field **LINK** which is used so that all records in **T** with same hash

address **h** may be linked together to form a *linked list*. Second there is a hash address table **LIST** that contains pointers to the linked lists in **T**.

Suppose a new record **R** with key **k** is added to the file **F**. We place **R** in the first available location in the table **T** and then add **R** to the linked list with pointer

LIST [H (k)]

If the linked list of records is not sorted, then **R** is simply inserted at the deigning of its linked list. Searching for a record or deleting a record is similar to the searching a node or deleting a node from the linked list.

The average number of probes, using chaining, for a successful search and for an unsuccessful search are as following approximate values:

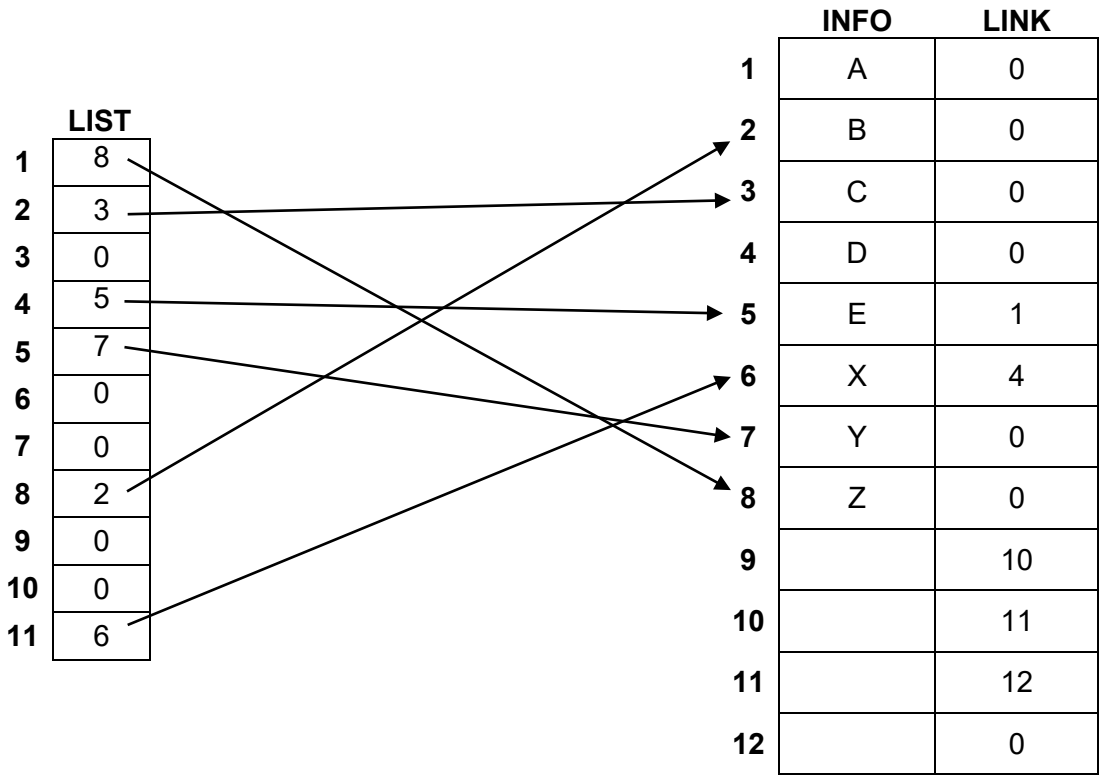
$$S(\lambda) \approx 1 + \tfrac{1}{2}\lambda \quad \text{and} \quad U(\lambda) \approx e^{-\lambda} + \lambda$$

Example: Place the following records according to the chaining method:

Suppose that the table **T** has the memory locations **T[1], T[2], … , T[11]** and suppose that the file **F** consists of 8 records **A, ,B , C, D, E, X, Y,** and **Z** with the following hash address.

record	A	B	C	D	E	X	Y	Z
H(k)	4	8	2	11	4	11	5	1

Solution:



N. B. Observe that the location of a record **R** in the table **T** is not related to its hash address. A record is simply put in the first node in the avail list of table **T**. In fact, the table **T** needs not to have the same number of elements as the hash address table.

Disadvantage of chaining method: The main disadvantage of the chaining method is that it needs **3m** memory cells for **m** number of data. But in case of linear probing in open addressing method, only **m** memory cells are required for **m** records.

********** **END** **********