International Islamic University Chittagong

Department of Computer Science & Engineering

Spring - 2021

Course Code: CSE-2321

Course Title: Data Structures

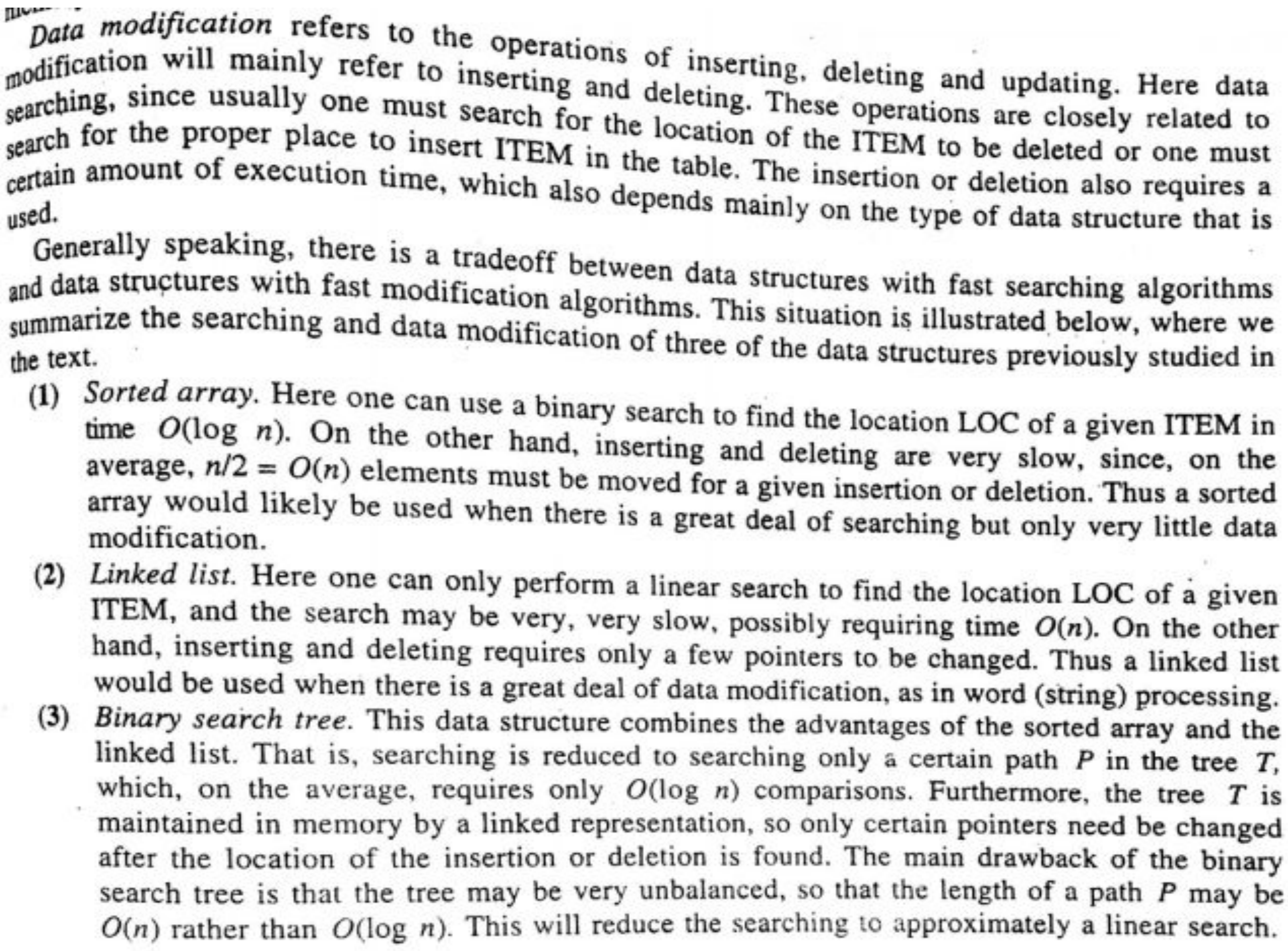
Mohammed Shamsul Alam

Professor, Dept. of CSE, IIUC

Lecture – 13

Hashing

Searching & Data Modification



- *Data modification* refers to the operations of inserting, deleting and updating. Here data modification will mainly refer to inserting and deleting. These operations are closely related to searching, since usually one must search for the location of the ITEM to be deleted or one must search for the proper place to insert ITEM in the table. The insertion or deletion also requires a certain amount of execution time, which also depends mainly on the type of data structure that is used.

Generally speaking, there is a tradeoff between data structures with fast searching algorithms and data structures with fast modification algorithms. This situation is illustrated below, where we summarize the searching and data modification of three of the data structures previously studied in the text.

(1) *Sorted array*. Here one can use a binary search to find the location LOC of a given ITEM in time $O(\log n)$. On the other hand, inserting and deleting are very slow, since, on the average, $n/2 = O(n)$ elements must be moved for a given insertion or deletion. Thus a sorted array would likely be used when there is a great deal of searching but only very little data modification.

(2) *Linked list*. Here one can only perform a linear search to find the location LOC of a given ITEM, and the search may be very, very slow, possibly requiring time $O(n)$. On the other hand, inserting and deleting requires only a few pointers to be changed. Thus a linked list would be used when there is a great deal of data modification, as in word (string) processing.

(3) *Binary search tree*. This data structure combines the advantages of the sorted array and the linked list. That is, searching is reduced to searching only a certain path P in the tree T, which, on the average, requires only $O(\log n)$ comparisons. Furthermore, the tree T is maintained in memory by a linked representation, so only certain pointers need be changed after the location of the insertion or deletion is found. The main drawback of the binary search tree is that the tree may be very unbalanced, so that the length of a path P may be $O(n)$ rather than $O(\log n)$. This will reduce the searching to approximately a linear search.

Hashing

- **Hashing** is a searching technique, which is essentially independent of the number n of input data.

- The idea of hashing can be introduced by the following example.

  Suppose a company with 68 employees assigns a 4-digit employee number to each employee which is used as the primary key in the company‘s employee file. We can, in fact, use the employee number as the address of the record in memory. The search will require no comparisons at all. Unfortunately, this technique will require space for 10,000 memory locations, whereas space for fewer than 30 such locations would actually be used. Clearly, this tradeoff of space for time is not worth the expense.

- The general idea of using the key to determine the address of a record is an excellent idea, but it must be modified so that a great deal of space is not wasted.
- This modification takes the form of a function H from the set K of keys into the set L of memory addresses. Such a function, H: K → L is called a **hash function** or **hashing function.**

Hash Table

- **Hash table** is a data structure used for storing and retrieving data quickly. Insertion of data in the hash table is based on the key value. Hence every entry in the hash table is associate with some key. For example, for an employee record in the hash table employee ID will works as a key.
- There are three components that are involved with performing storage and retrieval with Hash Tables:
  - ❑ **A hash table**: This is a fixed size table that stores data of a given type.
  - ❑ **A hash function**: This is a function that converts a piece of data into an integer. Sometimes we call this integer a **hash value**. The integer should be at least as big as the hash table. When we store a value in a hash table, we compute its hash value with the hash function, take that value modulo the hash table size, and that's where we store/retrieve the data.
  - ❑ **A collision resolution strategy**: There are times when two pieces of data have hash values that, when taken modulo the hash table size, yield the same value. That is called a collision. We need to handle collisions.

Hash Function

- Hash function is a function which is used to put data into hash table. Hence one can use the same as function to retrieve the data from hash table. Thus hash function is used to implement a hash table.
- 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 and quick to compute.
  - Secondly 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. However, certain general techniques do help.
- Some popular hash functions are :

a) Division hash function method

b) Mid square hash function method

c) Digit folding or folding hash function method

Hash Function

a) Division method

Choose a number **m** larger than the number **n** of keys in **K.** (The number m is usually chosen to be a **prime number** or a number without small divisors, since this frequently minimizes the number of collisions.) The hash functions H is defined by

H(k) = k(mod m) or H(k) = k(mod m) + 1

Here k (mod m) denotes the remainder when k is divided by m.

The second formula is used when we want the hash addresses to range from 1 to m rather than from 0 to m-1.

b) Midsquare method

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

H(k) = l

Where l is obtained by deleting digits from both ends of k^2.

c) Folding method:

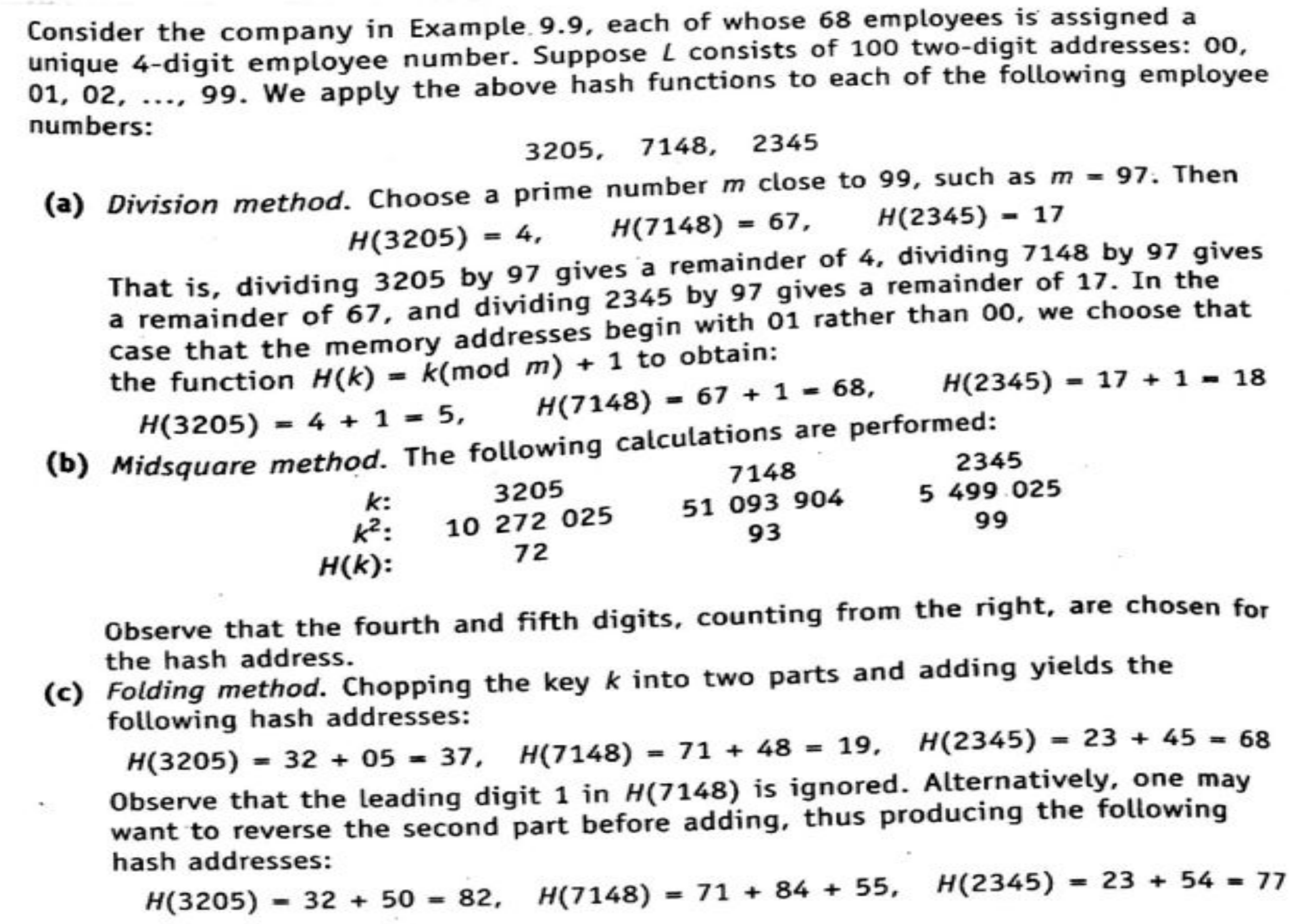
The key k is partitioned into a number of parts, k_1, k_2, ..., k_r, where each part, except possibly the last, has the same number of digits as the required address. Then the parts are added together, ignoring the last carry. That is,

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

Where the leading-digit carries, if any, are ignored.

Sometimes, for extra - ”milling”, the even-numbered parts, k_2,k_4,....., are each reversed before the addition.

Hash Function



Consider the company in Example 9.9, each of whose 68 employees is assigned a unique 4-digit employee number. Suppose L consists of 100 two-digit addresses: 00, 01, 02, ..., 99. We apply the above hash functions to each of the following employee numbers:

$$3205, \quad 7148, \quad 2345$$

(a) *Division method.* Choose a prime number m close to 99, such as $m = 97$. Then

$$H(3205) = 4, \qquad H(7148) = 67, \qquad H(2345) = 17$$

That is, dividing 3205 by 97 gives a remainder of 4, dividing 7148 by 97 gives a remainder of 67, and dividing 2345 by 97 gives a remainder of 17. In the case that the memory addresses begin with 01 rather than 00, we choose that the function $H(k) = k(\text{mod } m) + 1$ to obtain:

$$H(3205) = 4 + 1 = 5, \qquad H(7148) = 67 + 1 = 68, \qquad H(2345) = 17 + 1 = 18$$

(b) *Midsquare method.* The following calculations are performed:

k:	3205	7148	2345
k^2:	10 272 025	51 093 904	5 499 025
$H(k)$:	72	93	99

Observe that the fourth and fifth digits, counting from the right, are chosen for the hash address.

(c) *Folding method.* Chopping the key k into two parts and adding yields the following hash addresses:

$$H(3205) = 32 + 05 = 37, \quad H(7148) = 71 + 48 = 19, \quad H(2345) = 23 + 45 = 68$$

Observe that the leading digit 1 in $H(7148)$ is ignored. Alternatively, one may want to reverse the second part before adding, thus producing the following hash addresses:

$$H(3205) = 32 + 50 = 82, \quad H(7148) = 71 + 84 + 55, \quad H(2345) = 23 + 54 = 77$$

Collision Resolution

- Suppose we want to add a new record R with key k to our file F, but suppose the memory location address H(k) is already occupied. This situation is called **collision**.
- If collision occurs then it should be handled by applying some techniques. Such techniques are called **collision resolution technique**.
- The goal of collision resolution techniques is to minimize collisions. There are two methods of handling collisions.

  1. **Open hashing** or **Separate Chaining**
  2. **Closed hashing** or **Open addressing**
- The difference between open hashing and closed hashing is that in Open hashing the collision are stored outside table and in Closed hashing the collisions are stored in the same table at some another slot.

Closed hashing or **Open addressing**

- In Closed hashing the collisions are stored in the same table at some another slot. For Closed hashing one of the following technique is adopted.

  1. Linear probing
  2. Quadratic probing
  3. Double probing or Double hashing

Linear probing

- Suppose that a new record R with a key k is to be added to the memory table T, but that the memory location with hash address H(k)=h is already filled. One natural way to resolve the collision is to 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 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 an unsuccessful search. The above collision resolution is called **linear probing**.
- One main disadvantage of linear probing is that records tend to **cluster**, that is, appear next to one another.

Closed hashing or **Open addressing**

Example:

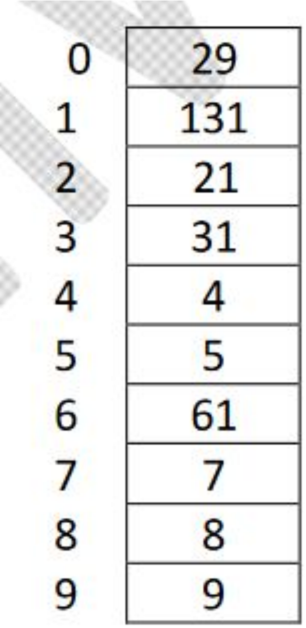
Consider that following keys are to be inserted in the hash table:

131, 4, 8, 7, 21, 5, 31, 61, 9, 29.

The hash table size is 10. We will use division hash function i.e.

H (Key) = key % table size

For instance the element 131 can be placed at H (Key) = 131 % 10 =1.



0	29
1	131
2	21
3	31
4	4
5	5
6	61
7	7
8	8
9	9

Class work: Draw the table when H (Key) = key % 11 + 1

Closed hashing or **Open addressing**

Quadratic probing

Suppose a record R with key k has the hash address H(k) = h. Then, instead of searching the locations with addresses h, h+1, h+2,....., we linearly search the locations with addresses h, h+1, h+4, h+9, h+16,........h+i^2,.....

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

This method uses following formula:

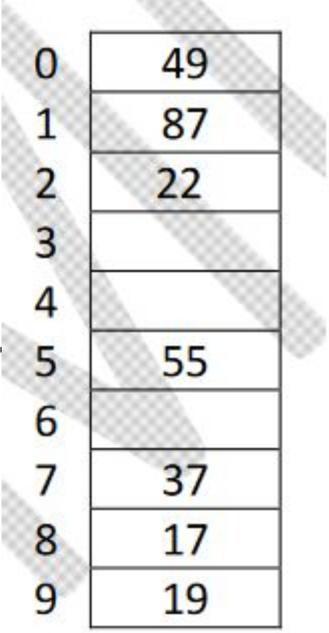
$$H(Key) = (H(Key) + i^2) \% m$$

Where ‘m’ can be table size or any prime number.

Example: -

Insert following elements in the hash table with table size 10.

37, 19, 55, 22, 17, 49, 87.



0	49
1	87
2	22
3	
4	
5	55
6	
7	37
8	17
9	19

Closed hashing or **Open addressing**

Double hashing

Here a second hash function H‘ is used for resolving a collision, as follows. Suppose a record R with key k has the hash addresses 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.

This method uses following formula:

H1 (key) = k % table size

A popular second hash function is:

H2 (key) = M - (K % M)

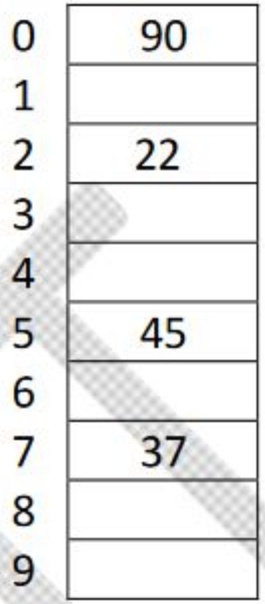
Where M is prime number smaller than the size of the table.

H(Key) = (H1 + i. H2) % table size

Example: consider the following elements to be placed in the Hash table of size 10.

37, 90, 45, 22, 17, 49, 55.

Now find where 17 will be inserted.



0	90
1	
2	22
3	
4	
5	45
6	
7	37
8	
9	

Closed hashing or **Open addressing**

Remark: 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]. Afterwards, suppose we meet T[r] while searching for another record R‘. This does not necessarily mean that the search is unsuccessful. Thus, when deleting 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.

Open hashing or Separate **Chaining**

Open Hashing, is a technique in which the data is not directly stored at the hash key index (k) of the Hash table. Rather the data at the key index (k) in the hash table is a pointer to the head of the data structure where the data is actually stored. In the most simple and common implementations the data structure adopted for storing the element is a linked-list.

In this technique when a data needs to be searched, it might become necessary (worst case) to traverse all the nodes in the linked list to retrieve the data.

Example:-

Consider the keys to be placed in the home buckets are

131, 3, 4, 21, 61, 24, 7, 97, 8, 9.

A chain is maintained for colliding elements. For example 131 has a home bucket index 1. Similarly keys 21 and 61 demand for home bucket index 1. Hence a chain is maintained at index 1. Similarly the chain at index 4 and 7 is maintained.

Separate Chaining Vs Open Addressing

Separate Chaining	Open Addressing
Keys are stored inside the hash table as well as outside the hash table.	All the keys are stored only inside the hash table. No key is present outside the hash table.
The number of keys to be stored in the hash table can even exceed the size of the hash table.	The number of keys to be stored in the hash table can never exceed the size of the hash table.
Deletion is easier.	Deletion is difficult.
Extra space is required for the pointers to store the keys outside the hash table.	No extra space is required.
Cache performance is poor. This is because of linked lists which store the keys outside the hash table.	Cache performance is better. This is because here no linked lists are used.
Some buckets of the hash table are never used which leads to wastage of space.	Buckets may be used even if no key maps to those particular buckets.