SEARCHING

Introduction

- Process of finding an element within the list of elements in order or randomly.
- Retrieval: Successful Search.
- A table of records in which a key is used for retrieval is often called a **SEARCH TABLE** or **DICTIONARY.**
- Internal Searching – Whole data in main memory
- External Searching – Most data is kept in auxiliary memory.

Searching Methods

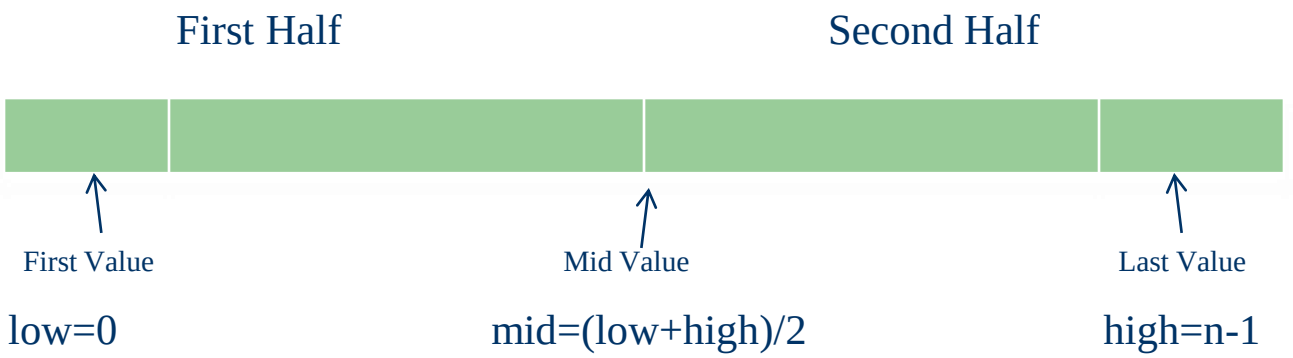
- Sequential or Linear Searching.
- Binary Search.
- Hashing.

Sequential Search

- Searches on unordered and ordered tables in sequential manner until the desired record is not found or table end is not reached.
- It is simple and good for small arrays.
- Mostly used when data is not sorted.
- Efficiency:
  - Best – O(1)
  - Average – O(n/2)
  - Worst – O(n)
- Less efficient if array size is large.
- Not efficient on sorted arrays.

Binary Search

- This technique works better on sorted arrays and can be applied only on sorted arrays.
- Not applied on Linked Lists.
- Requires less number of comparisons than linear search.
- Efficiency: O(log2n).
- Logic behind the technique:



Hashing

Introduction

- The search operation on a sorted array using the binary search method takes O(log2n)
- We can improve the search time by using an approach called Hashing.
- Usually implemented on Dictionaries.
- Well, there are lots of applications out there that need to support ONLY the operations INSERT, SEARCH, and DELETE. These are known as “dictionary” operations.
- Hashing can make this happen in O(1) and is quite fast in practice.

Dictionary

- A dictionary is a collection of elements
- Each element has a field called **key**
  - (key, value)
- Every key is usually distinct.
- Typical dictionary operations are:
  - **Insert** a pair into the dictionary
  - **Search** the pair with a specified key
  - **Delete** the pair with a specified key
- Collection of student records in a class
  - (key, value) =(student-number, a list of assignment and exam marks)
  - All keys are distinct

Dictionary as an Ordered Linear List

- $L = (e_1, e_2, e_3, \ldots, e_n)$
- Each e_i is a pair (key, value)
- Array or chain representation
  - unsorted array: O(*n*) search time
  - sorted array: O(log*n*) search time
  - unsorted chain: O(*n*) search time
  - sorted chain: O(*n*) search time

Hash Table

- A **hash table** is a data structure that stores elements and allows insertions, lookups, and deletions to be performed in O(1) time.
- A **hash table** is an alternative method for representing a dictionary
- In a hash table, a **hash function** is used to map keys into positions in a table. This act is called **hashing**
- Hash Table Operations
  - **Search**: compute f(k) and see if a pair exists
  - **Insert**: compute f(k) and place it in that position
  - **Delete**: compute f(k) and delete the pair in that position
- In ideal situation, hash table search, insert or delete takes $\Theta(1)$

Why we need Hash Tables

- Internet routers is a good example of why hash tables are required.
- A router table (especially in those routers in the backbone networks of internet operators) may contain hundreds of thousands or millions of entries.

  When a packet has to be routed to a specific IP ad dress, the router has to determine the best route by querying the router table in an efficient manner. Hash Tables are used as an efficient lookup struct ure having as key the IP address and as value the path that should be follow for that address.

Why we need Hash Tables

Any form of search on strings needs Hash.

Consider for example, you have a list of tokens (basically each token is a string) is used as an index of a table with some property.

```
URL_index["google"] = { "http://www.google.com", 100}
URL_index["yahoo"] = { "http://www.yahoo.com", 90}
URL_index["amazon"] = { "http://www.amazon.com", 85}
```

Now, you have a search string *s* is either of the above, you need to match the most relevant string. A brute force match will require StringCompare against each. However, you can make the Hash table of the URL_index and quickly make the search possible.

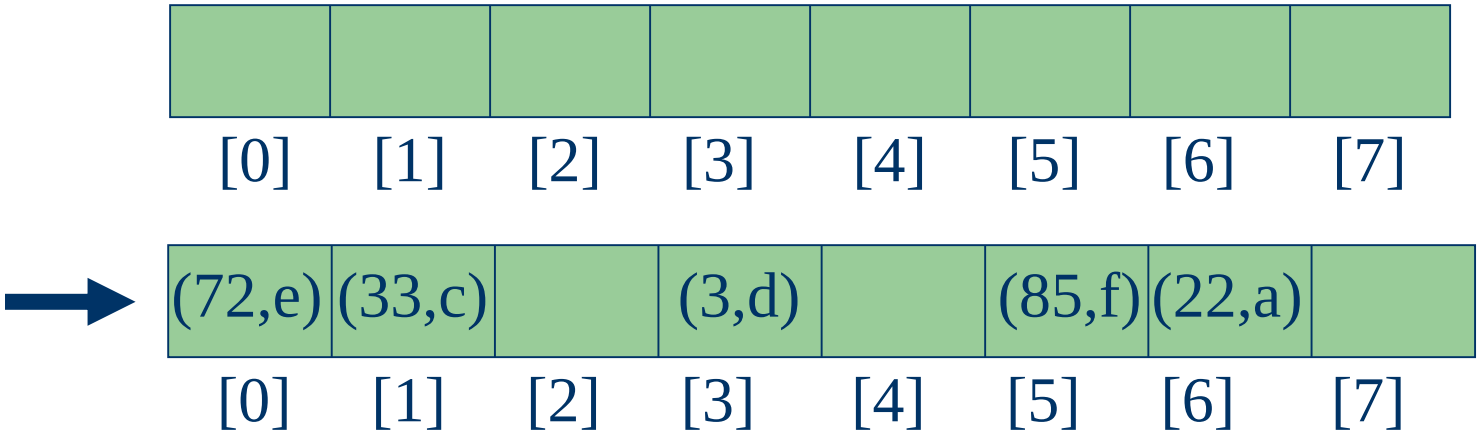
All search mechanisms in side databases, uses Hash table based indexes.

How Does it Work

- The **table** part is just an ordinary array, it is the **Hash** that we are interested in.
- The **Hash** is a function that transforms a key into address or index of array(table) where the record will be stored. If the size of the table is *N*, then the integer will be in the range 0 to *N*-1. The integer is used as an index into the arr ay. Thus, in essence, the key itself indexes the array.
- If *h* is a hash function and *k* is key then *h(k)* is called the *hash of the key* and is the index at which a record with the key k should be placed.
- The hash function generates this address by performing some simple arithmetic or logical operations on the key.

Ideal Hashing Example

- Pairs are: (22,a),(33,c),(3,d),(72,e),(85,f)-
  -(key, value) pairs
- Hash table is ht[0:7], m = 8 (where m is the number of positions in the hash table)
- Hash function ***h* is k % m** = k % 8
- Where are the pairs stored?



Hashing Function Methods (Hashing Methods)

- **Division Hash Method**
  - The key K is divided by some number m and the remainder is used as the hash address of K.
    - *h(k)=k mod m*
  - This gives the indexes in the range 0 to m-1 so the hash table should be of size m
  - This is an example of uniform hash function if value of m will be chosen carefully.
  - Generally a prime number is a best choice which will spread keys evenly.
  - A uniform hash function is designed to distribute the keys roughly evenly into the available positions within the array (or hash table).

Hashing Function Methods

- **The Folding Method**
  - The key K is partitioned into a number of parts ,each of which has the same length as the required address with the possible exception of the last part .
  - The parts are then added together , ignoring the final carry, to form an address.
  - Example: If key=356942781 is to be transformed into a three digit address.

    P1=356, P2=942, P3=781 are added to yield 079.

Hashing Function Methods

- **The Mid- Square Method**
  - The key K is multiplied by itself and the address is obtained by selecting an appropriate number of digits from the middle of the square.
  - The number of digits selected depends on the size of the table.
  - Example: If key=123456 is to be transformed.
  - (123456)2=15241383936
  - If a three-digit address is required, positions 5 to 7 could be chosen giving address 138.

Hashing a string key

- Table size [0..99]
- A..Z ---> 1,2, ...26
- 0..9 ----> 27,...36
- Key: CS1 --->3+19+28 (concat) = 31,928
- $(31,928)^2$ = 1,019,397,184 - 10 digits
- Extract middle 2 digits (5th and 6th) as table size is 0..99.
- Get 39, so: H(CS1) = 39.

Characteristics of a Good Hash Function

- The hash value is fully determined by the data being hashed.
- The hash function uses all the input data.
- The hash function "uniformly" distributes the data across the entire set of possible hash values.
- The hash function generates very different hash values for similar strings.