DATA STRUCTURE AND ALGORITHM

Lecturer: Er. Saroj Ghimire

Qualification: Msc.CSIT, BE(COMPUTER)

Lincoln University College

Overview

- Definition of Searching
- Searching Algorithm: Sequential and Binary search
- Concept of Hash: Hash Tables, Hash Function
- Collision Resolution Techniques

Objectives

- To understand searching algorithm in large and small dataset
- Learning different techniques of hashing and its importance

Definition of Searching

- Searching is a process of checking and finding an element from a list of elements.
- Let A be a collection of data elements, i.e., A is a linear array of say n elements. If we want to find the presence of an element "data" in A, then we have to search for it.
- The search is successful if data does appear in A and unsuccessful if otherwise.
- Searching Algorithms are designed to check for an element or retrieve an element from any data structure where it is stored.
- Based on the type of search operation, these algorithms are generally classified into two categories:
  - Linear search
    - Small arrays
    - Unsorted arrays
  - Binary search
    - Large arrays
    - Sorted arrays

Linear search

- In linear search, each element of an array is read one by one sequentially and it is compared with the desired element.
- A search will be unsuccessful if all the elements are read and the desired element is not found.
- Linear search is mostly used to search an unordered list in which the items are not sorted.

- Let A be an array of n elements, A[1],A[2],A[3], ... A[n]. "data" is the element to be searched. Then this algorithm will find the location "loc" of data in A. Set loc = – 1,if the search is unsuccessful.

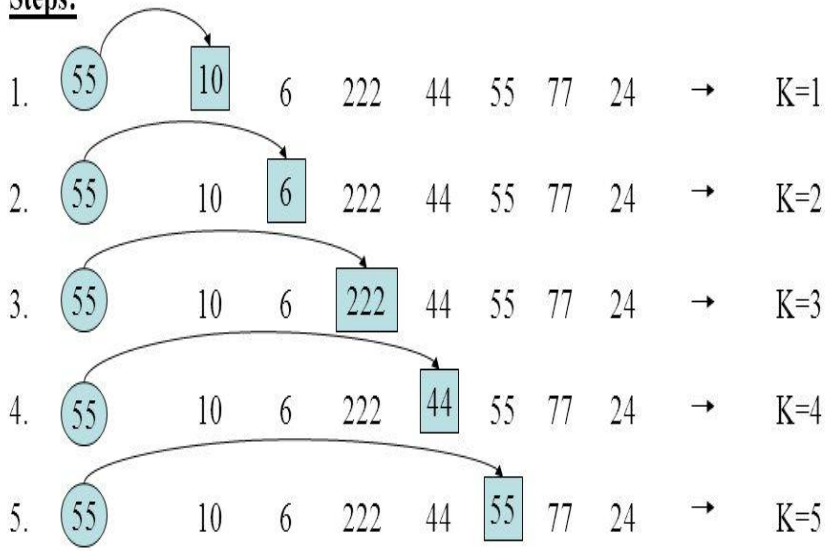
1. Input an array A of n elements and "data" to be searched and initialize loc = – 1.
2. Initialized i = 0; and repeat through step 3 if (i < n) by incrementing i by one .
3. If (data = A[i])
   1. loc = I
   2. GOTO step 4
4. If (loc > 0)
   1. Display "data is found and searching is successful"
5. Else
   1. Display "data is not found and searching is unsuccessful"
6. Exit

Example of Linear Search

List : 10, 6, 222, 44, 55, 77, 24

Key: 55

Steps:



So, Item 55 is in position 5.

4

Binary Search

- Binary search is an extremely efficient algorithm when it is compared to linear search.
- Binary search technique searches "data" in minimum possible comparisons.
- Suppose the given array is a sorted one, otherwise first we have to sort the array elements.

Binary Search

Algorithm

1. Find the middle element of the array (i.e., n/2 is the middle element if the array or the sub-array contains n elements).
2. Compare the middle element with the data to be searched, then there are following three cases.
    1. If it is a desired element, then search is successful.
    2. If it is less than desired data, then search only the first half of the array, i.e., the elements which come to the left side of the middle element.
    3. If it is greater than the desired data, then search only the second half of the array, i.e., the elements which come to the right side of the middle element.

Repeat the same steps until an element is found or exhaust the search area.

Binary Search (example)

Suppose we have an array of 7 elements

9	10	25	30	40	45	70

1. Following steps are generated if we binary search a data = 45 from the above array.

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 0; UB = 6 (lower bound and upper bound)

mid = (0 + 6)/2 = 3

A[mid] = A[3] = 30

2. Since (A[3] < data) - i.e., 30 < 45 - reinitialize the variable LB, UB and mid

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 3 UB = 6

mid = (3 + 6)/2 = 4

A[mid] = A[4] = 40

Binary Search (example)

3. Since (A[4] < data) - i.e., 40 < 45 - reinitialize the variable LB, UB and mid

9	10	25	30	40	45	70
0	1	2	3	4	5	6

LB = 4 UB = 6

mid = (4 + 6)/2 = 5

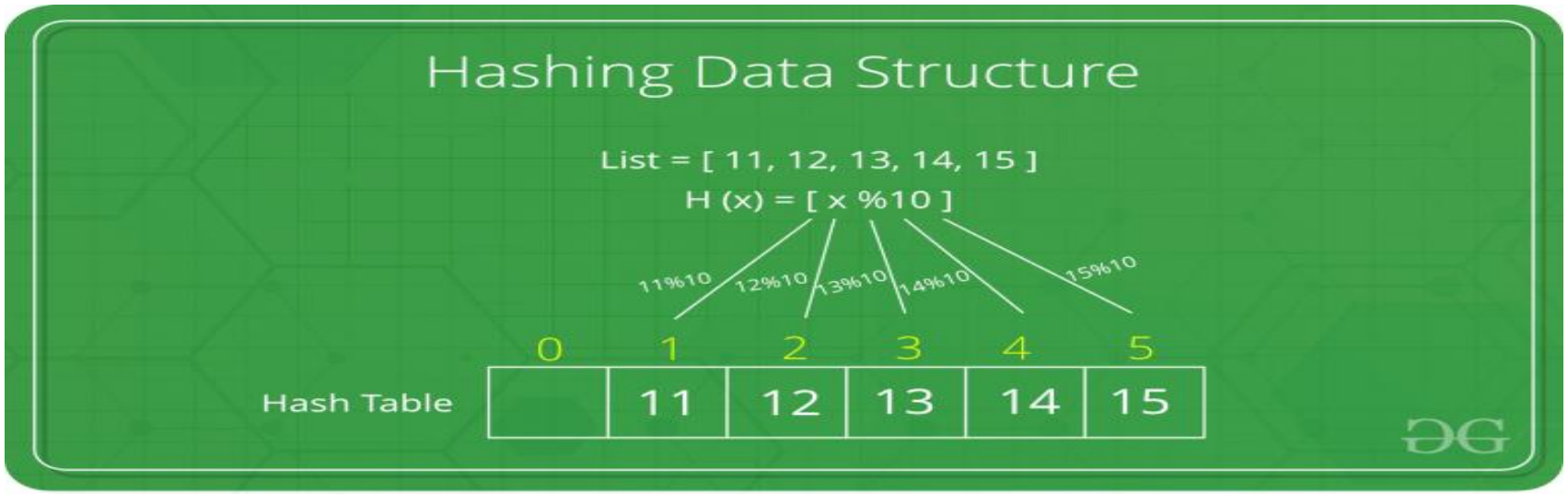
A[mid] = A[5] = 45

Step 4:

Since (A[5] == data) - i.e., 45 == 45 - searching is successful.

Concept of Hash: Hash Tables, Hash Function

- Hashing is the process of mapping large amount of data item to smaller table with the help of hashing function. Hashing is also known as **Hashing Algorithm** or **Message Digest Function**.
- Hashing is a technique or process of mapping keys, values into the hash table by using a hash function.
- It is done for faster access to elements. The efficiency of mapping depends on the efficiency of the hash function used.
- Let a hash function H(x) maps the value at the index **x%10** in an Array. For example if the list of values is [11,12,13,14,15] it will be stored at positions {1,2,3,4,5} in the array or Hash table respectively.



Concept of Hash: Hash Tables, Hash Function

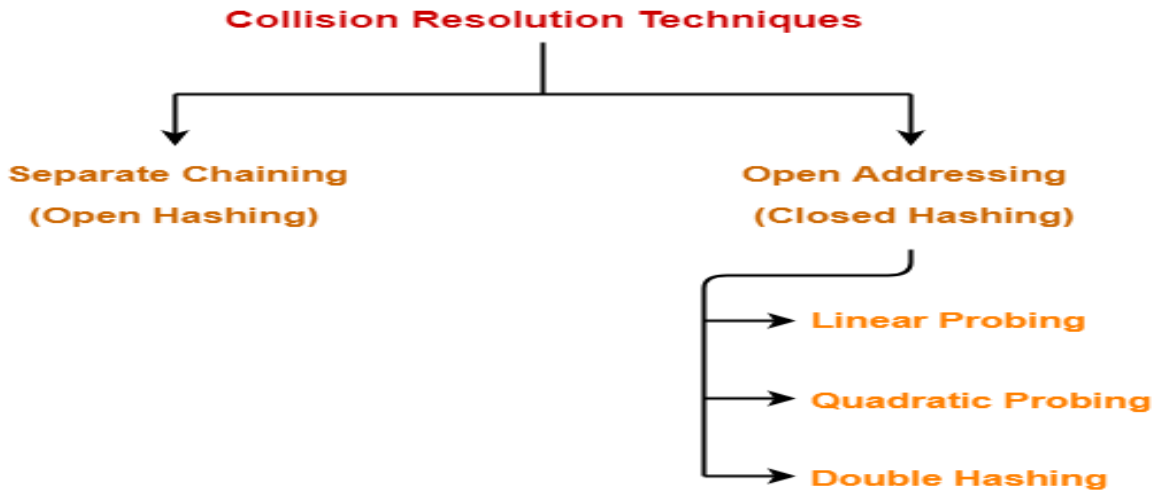
- A fixed process converts a key to a hash key is known as a **Hash Function.**
- This function takes a key and maps it to a value of a certain length which is called a **Hash value** or **Hash.**
- Hash value represents the original string of characters, but it is normally smaller than the original.
- It transfers the digital signature and then both hash value and signature are sent to the receiver. Receiver uses the same hash function to generate the hash value and then compares it to that received with the message.
- If the hash values are same, the message is transmitted without errors.

Concept of Hash: Hash Tables, Hash Function

- Hash table or hash map is a data structure used to store key-value pairs.
- It is a collection of items stored to make it easy to find them later.
- It uses a hash function to compute an index into an array of buckets or slots from which the desired value can be found.
- It is an array of list where each list is known as bucket.It contains value based on the key.
- Hash table is used to implement the map interface and extends Dictionary class. Hash table is synchronized and contains only unique elements

Collision Resolution Techniques

- When the hash value of a key maps to an already occupied bucket of the hash table, it is called as a **Collision**
- A collision occurs when two different keys hash to the same value
  - E.g. For Table Size = 17, the keys 18 and 35 hash to the same value
  - 18 mod 17 = 1 and 35 mod 17 = 1
- Collision Resolution Techniques are the techniques used for resolving or handling the collision.



Separate Chaining

- ✓ To handle the collision, this technique creates a linked list to the slot for which collision occurs.
- ✓ The new key is then inserted in the linked list.
- ✓ These linked lists to the slots appear like chains.

Example :Use separate chaining technique for collision resolution.

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table- 50, 700, 76, 85, 92, 73 101

1. Draw empty hash table and possible range of hash values is (0,6)
2. The first key to be inserted in the hash table = 50.The hash table to which key 50 maps = 50 mod 7 = 1
3. similarly for each item to inserted we calculate hash value

   700mod 7=0

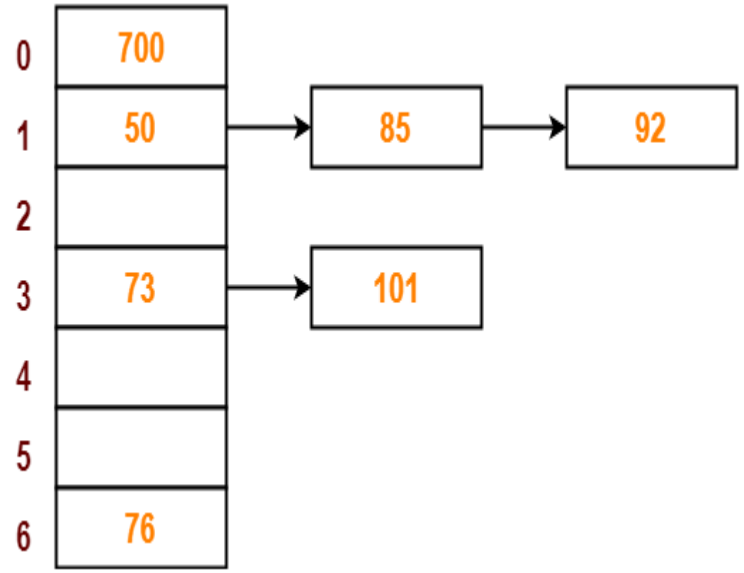
   76mod 7=6

   85mod 7=1

   92mod 7=1

   73mod 7=3

   101mod 7=3



Open Addressing

- The open addressing is another technique for collision resolution. Unlike chaining, it does not insert elements to some other data-structures.
- It inserts the data into the hash table itself. The size of the hash table should be larger than the number of keys.
- There are three different popular methods for open addressing techniques. These methods are −
  - Linear Probing
  - Quadratic Probing
  - Double Hashing

linear probing

- ✓ In linear probing, When collision occurs, we linearly probe for the next bucket.
- ✓ We keep probing until an empty bucket is found.

Using the hash function 'key mod 7', insert the following sequence of keys in the hash table- 50, 700, 76, 85, 92, 73,101

1. Draw empty hash table and possible range of hash values is (0,6)
2. The first key to be inserted in the hash table = 50.The hash table to which key 50 maps = 50 mod 7 = 1
3. Bucket of the hash table to which key 700 maps = 700 mod 7 = 0
4. Bucket of the hash table to which key 76 maps = 76 mod 7 = 6.
5. Bucket of the hash table to which key 85 maps = 85 mod 7 = 1. Since bucket-1 is already occupied, so collision occurs. To handle the collision, linear probing technique keeps probing linearly until an empty bucket is found. The empty bucket is 2 so we insert 85 in bucket 2.
6. Bucket of the hash table to which key 92 maps = 92 mod 7 = 1.Since bucket-1 is already occupied, so collision occurs. The first empty bucket is 3 so 92 inserted in it.
7. Bucket of the hash table to which key 73 maps = 73 mod 7 = 3. Since bucket-3 is already occupied, so collision occurs. The first empty bucket is 4 so 73 inserted in it.
8. 101 maps = 101 mod 7 = 3. The first empty bucket is 5 so 101 inserted in it.
9. 76 maps = 76 mod 7 = 6.

0	700
1	50
2	85
3	92
4	73
5	101
6	76

Quadratic Probing

- ✓ In quadratic probing, When collision occurs, we probe for i^2'th bucket in i^{th} iteration.
- ✓ We keep probing until an empty bucket is found.

Quadratic Probing -- Example

- Example:
  - Table Size is 11 (0..10)
  - Hash Function: **h(x) = x mod 11**
  - Insert keys: 20, 30, 2, 13, 25, 24, 10, 9
    - 20 mod 11 = 9
    - 30 mod 11 = 8
    - 2 mod 11 = 2
    - 13 mod 11 = 2 ➔ $2+1^2=3$
    - 25 mod 11 = 3 ➔ $3+1^2=4$
    - 24 mod 11 = 2 ➔ $2+1^2$, $2+2^2=6$
    - 10 mod 11 = 10
    - 9 mod 11 = 9 ➔ $9+1^2$, $9+2^2$ mod 11, $9+3^2$ mod 11 =7

0	
1	
2	2
3	13
4	25
5	
6	24
7	9
8	30
9	20
10	10

Double Hashing

- In double hashing, We use another hash function. First hash function is typically hash1(key) = key % TABLE_SIZE
- A popular second hash function is : hash2(key) = PRIME – (key % PRIME) where PRIME is a prime smaller than the TABLE_SIZE.
- It requires more computation time as two hash functions need to be computed.

Table Size = 10 elements
$Hash_1(key) = key \% 10$
$Hash_2(key) = 7 - (k \% 7)$

Insert keys : 89, 18, 49, 58, 69

Hash(89) = 89 % 10 = 9

Hash(18) = 18 % 10 = 8

Hash(49) = 49 % 10 = 9 a collision !
= 7 – (49 % 7)
= 7 positions from [9]

Hash(58) = 58 % 10 = 8
= 7 – (58 % 7)
= 5 positions from [8]

Hash(69) = 69 % 10 = 9
= 7 – (69 % 7)
= 1 position from [9]

[0]	49
[1]	
[2]	
[3]	69
[4]	
[5]	
[6]	
[7]	58
[8]	18
[9]	89

Further Readings

- Brian W. Kernighan and Dennis M. Ritchie, The C Programming Language, Prentice Hall, 1988.
- Data Structures and Algorithms; Shi-Kuo Chang; World Scientifi c.
- Data Structures and Effi cient Algorithms, Burkhard Monien, Thomas Ottmann, Springer.
- Kruse Data Structure & Program Design, Prentice Hall of India, New Delhi
- Mark Allen Weles: Data Structure & Algorithm Analysis in C Second Adition. Addison-Wesley publishing
- RG Dromey, How to Solve it by Computer, Cambridge University Press.
- Shi-kuo Chang, Data Structures and Algorithms, World Scientifi c
- Sorenson and Tremblay: An Introduction to Data Structure with Algorithms.
- Thomas H. Cormen, Charles E, Leiserson & Ronald L. Rivest: Introduction to Algorithms.
- Prentice-Hall of India Pvt. Limited, New Delhi Timothy A. Budd, Classic Data Structures in C++, Addison Wesley