Unit 1 lo except Rehashing, Extendible Hashing

https://youtu.be/zeMa9sg-VJM
https://youtu.be/dxrLtf-Fybk
https://youtu.be/AYcsTOeFVas

REHASHING
https://www.youtube.com/watch?v=uaGWFN6djLw

LINEAR PROBING EXAMPLE
https://courses.cs.washington.edu/courses/cse326/00wi/handouts/lecture16/sld015.htm

UNIT - I

What is a dictionary

→ Dictionary is a general purpose DS for storing objects.
→ " has associated keys. Each key has a single associated value.

HASHING:

- It is a technique used to map (key, value) pairs into a table, using a hash function.
- It is used for storage and faster access of information.
- It performs search in constant time. i.e. O(1)
* It is used for implementing symbol tables.

COMPONENTS OF HASHING:

1. Hash Table
2. Hash functions
3. Collisions
4. Collision Resolution Techniques.

HASH TABLE: A DS. that stores data in associative manner. Given a (key, value) pair, hash table stores the value at corresponding key.

HASH FUNCTION: (Generally division-modulo func. used. $h(x) = k \bmod m$ (k → key, m → table size))

* To map a key to an entry, we map the key to an integer. It is in-turn used for indexing the position.
* Hash function is a mathematical function used for mapping the key to an integer.

Characteristics of Good hash function:

* Minimize Collisions
* Be easy & quick to compute
* Distribute key-value pairs evenly across the table
* Have a high load factor for given set of keys.

load factor:

* It is used for determining efficiency of hash function
* Specifies whether values are uniformly distributed.

$$\text{load factor} = \frac{\text{No. of elements in hash table}}{\text{Hash table size.}}$$

COLLISIONS: It is a condition when two keys are hashed to same slot.

COLLISION RESOLUTION & TECHNIQUES:

The process of finding an alternate location for a key in the case of a collision is called collision resolution.

TECHNIQUES: (HASHING)

Direct Chaining (or Closed Addressing)

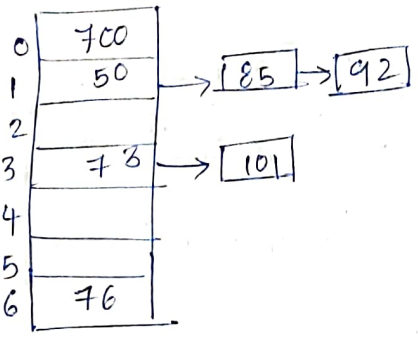
* Separate Chaining (open hashing)

Open Addressing

* Linear probing (linear search)
* Quadratic probing (Non-linear search)
* Double Hashing (Use two hash functions)

SEPARATE CHAINING:

* Implemented using linked list. Combines linked list & hashtable
* Element is stored in a node of a linked list.
* When two or more elements hash to the same location, these elements are stored as nodes of a singly linked list. (known as a chain)



Adv:

* Easy to implement
* hash table doesn't fill up
* less sensitive to hash-func or load-factor.
* Used when no. of keys to be inserted is unknown.

Disadv:

* Bad Cache performance
* Wastage of Space
* If Chain gets long, worst case time complexity for search is $O(n)$.

OPEN ADDRESSING:

* Unlike separate Chaining, in open addressing all values are stored in hash table
* This is known as closed hashing.

LINEAR PROBING:

* The interval b/w probes is fixed at 1.

$$h(x) = k \bmod m$$

$$\text{rehash}(key) = (h(x)+1) \bmod m$$

$k \rightarrow$ key, $m \rightarrow$ table size, $h(x) \rightarrow$ hash func.

Challenges in linear probing:

Primary Clustering: Consecutive elements form groups. Takes time to search for free slot.

Secondary Clustering: Two records have same collision chain.

Quadratic probing:

→ To work efficiently → * Table size should be prime number.
* Never be half full.

* The interval b/w probe is proportional to hash value.
* Rehash (key) = $(h(x) + k^2) \bmod m$.

Double hashing:

* uses two hash functions.

$h_1(x) = k \bmod m$

$h_2(x) = PRIME - (k \% PRIME)$

* PRIME smaller than table size.

$rehash(key) = [h_1(x) + i * h_2(x)] \bmod m$.

REHASHING:

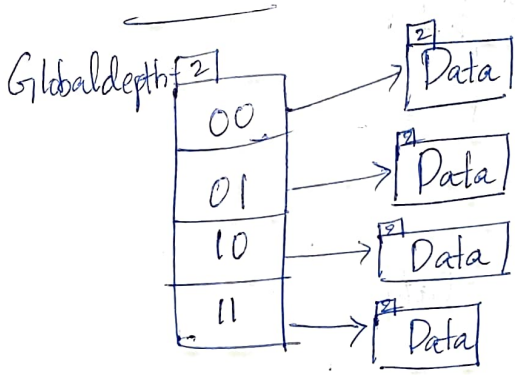
* Default value of load factor (α) is 0.75.
* When $\alpha > 0.75$, complexity of O(1) changes & increases.
* So to overcome this, table size is doubled.
* Table size is doubled & elements are mapped to the new table.

Extendible HASHING:

* Directories & buckets used to map data.

Directories: Stores addresses of buckets in pointers.

Buckets: Used to hash actual data



Directories: Containers store pointers to buckets.
No. of Directories = 2^Global depth.

Buckets:
↓
Stores hashed keys.

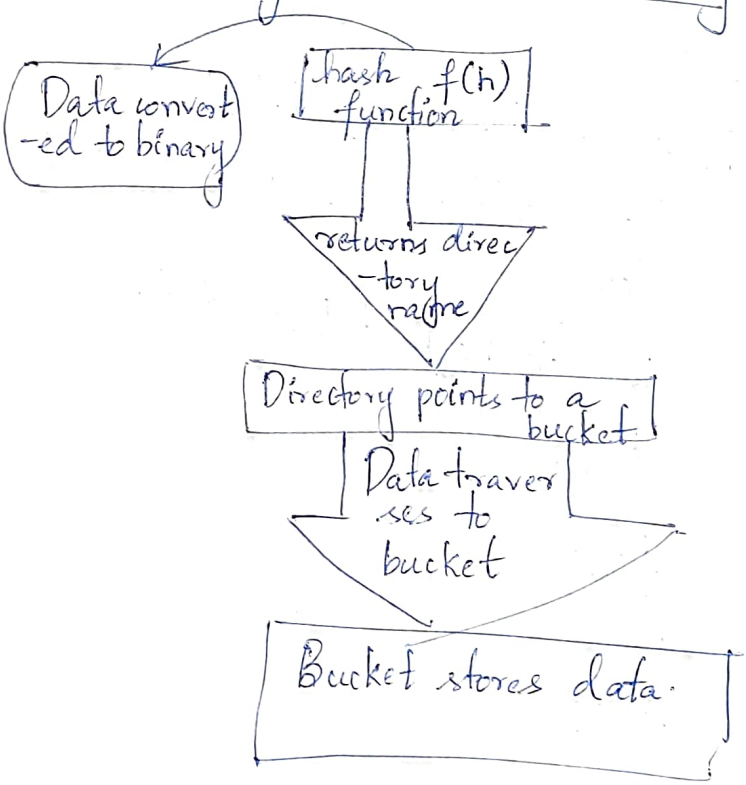
Global depth: * used by directories.
No. of bits used by hash function.
Global depth = No. of bits in directory id.

Local depth: Associated with buckets
Same as global depth.

Bucket splitting:
No. of elements in a bucket exceeds, bucket is split into two parts.

Directory Expansion:
↓
performed when local depth exceeds global depth.

Basic Working of Extendible hashing:



Step 1: Data converted to binary form. (b → 110001)

Step 2: Global depth is checked. LSB's taken ↓
(Suppose global depth is 3, then 001 is used to search directory).

Step 3: When directory is found, overflow is checked.
i) Bucket overflow → splitting buckets
ii) Directory overflow → Directory expansion

Step 4: After overflow is resolved, element is stored.

Linear probing	Quadratic Probing	Double Hashing
* Fastest among three	* Easier to implement	* Efficient memory usage
* Suffers from primary & secondary clustering.	* Suffers from secondary clustering	* Complicated to implement.
* Interval of probe is set to 1	* Interval of probe is proportional to hash value.	* Interval b/w probes is computed by hash functions

Dictionary ~~functions~~ operations

1. Retrieve a Value
2. Insert & update a value
3. Remove a key, value pair
4. Verify existence of key

Applications:

1. Credit card authorization
2. DNS mapping of host names
3. Word-definition pairs.

Search Algorithm:

```
Algorithm findAllElements(k)
Input: key k
Output: All elements associated to k
Create an empty list L
  B = D.entries
  While B.hasNext() do:
        e = B.next()
        if (key(e) == k):
            L.insertLast(e)

  end while
  return L.elements()
end
```

Algorithm Insert(k,v)
input: k,v pair
Output: store k,v in Dictionary D.
Create a new entry e.
e = (k,v)
S. insertlast(e)
return e.

Algorithm remove(e)
input: e
Output: return e if found & removed else return NULL.
B = S.positions{} (It is assumed e does not store location in S)
while B.hasnext() do
p = B.next()
if (p.element == e):
remove p
return e
end while
return NULL.