Hashing

What is hashing??

- In this course, we have learned to store data in different ways.
- What is our objective behind that: Make search procedure faster so that a data item from data set can be retrieved immediately
- All the storing and search techniques read by us require that ***we walk through some data*** and finally get the required record of key ***k***.
- The **organization of file** and the order in which the keys are **inserted effect the number of key**s that must be inspected before obtaining the desired one.
- Till now we have not studied any technique where **looking at the primary key of data**, we can say that the record of key K is stored at address M.
- This means ***there was no relationship between key and the address of record.***
- ***Hashing is a technique which allows us to evaluate the address of record given the primary key of the record***

What is hashing??

- In general, we have seen search techniques which take search time somewhere between O(log(n)) to O(n).
- Internet has grown to millions of users generating terabytes of contents every day and also continuously growing.
- It is impossible to find anything in the internet, unless we develop some new data structures and algorithms for storing and accessing data
- **Calculate the amount of time taken by an O(n) and an O(log(n)) search algorithm if data size is one tera byte and it takes one nano second to search one byte**. (What is tera byta? (2^{40}) home assignment, will discuss in next class)
- We will discuss a new technique called hashing that allows us to update and retrieve any entry in constant time O(1).
- If each key is to be retrieved in a single access, the location of the record within the table can depend only on the key not on the locations of other keys.

Approximating size of data

- 1,024 Bytes = 1 Kilobyte
- 1,024 Kilobytes = 1 Megabyte
- 1,024 Megabytes = 1 Gigabyte
- 1,024 Gigabytes = 1 Terabyte
- 1,024 Terabytes = 1 Petabyte
- 1,024 Petabytes = 1 Exabyte (In 2000, 3 exabytes of information was created.)
- 1,024 Exabytes = 1 Zettabyte
- 1,024 Zettabyte = 1 Zottabyte
- 1,024 Zottabyte = 1 Brontobyte (That is a 1 followed by 27 zeroes.)

Example1

- Let us take an example:
- Suppose there are 360 students in first year and we give them rollno as 1 to 360 and store the data of students in an array of size 360.
- This means array index is same as student rollno.
- If I have to search rollno 240, I will directly go to 240 location and get the record.
- Suppose we can not give rollnos to be 1 to 360, we want to allot the numbers as 50001 to 50360, then also we can store in an array of size 360 and get rollno 50240 mapped to address 240 by extracting only last three digits.

Example2

- Let us take another example:
- Suppose a manufacturing company has an inventory file consisting of 100 or more parts.
- Each part has a unique three digit number associated with it.
- We can declare an array of size 1000 to store the details of each part:

Parttype part[1000]; parttype is a structure to store details of each part

- For example: If we have car part shop then if the number for tyre is 50, For steering it is 70 then…
- In the array at location 50, all the details about tyres will be stored and location 70 all the information about steerings will be stored if the partno is is used as key to store the information of that part in that index of array
- Obviously, if there are 110 parts 890 entries will be empty out of 1000

Query??

- Is it possible to define a relationship between address of record and primary key of record??
- Depends upon the key values
- May not be always possible
- Keys may be sparse leading to waste of space.

Defining hashing..

- A function that transforms a key into a table index is called a ***hash function.***
- If **h** is a **hash function** and **k** is key then **h(k)** is called **hash of key k**
- Suppose there are 90 parts in a shop but partno. can be of 4 digits. (can have values from 0001 to 9999)
- In that case, we can take an array of size 100 and use hash function as:
- h(k) = k mod 100
- The value that hash function produces will cover the entire set of indices in the table

Continued…

- But this has a flaw: All keys 60, 160, 260……4060….. will be mapped to location 60
- Example: suppose array size is 10 and keys are 12 48 569 23 22 50 29 111
- So hash function will be **key mod 10**
-

a[0]	**a[1]**	**a[2]**	**a[3]**	**a[4]**	**a[5]**	**a[6]**	**a[7]**	**a[8]**	**a[9]**
50	111	12	23					48	569

- **And we do not know how to store keys 22, 29**

Hashing or map data structure

- Hashing is a Map Data Structure in mathematical sense
- A map is a relation between two sets.
- We can define **Map M** as a set of pairs, where each pair is of the form **(key, value),**
- For given a **key k**, we can find a value using some kind of a “function” that maps keys to values.
- Suppose we have a set of strings {“abc”, “def”, “ghi”} that we’d like to store in a table.
- Our objective here is to search a record or update the record as quickly as possible from a table, actually in O(1).

Cont..

- We are not concerned about ordering them or maintaining any order at all.
- Let us think of a simple method to do this. Suppose we assign “a” = 1, “b”=2, … etc to all alphabetical characters.
- We can then simply compute a number for each of the strings by using the sum of the characters as follows. “abc” = 1 + 2 + 3=6, “def” = 4 + 5 + 6=15 , “ghi” = 7 + 8 + 9=24
- If we assume that we have a table of size 10 to store these strings, we can compute the location of the string by taking the sum mod 10.

Problems with Hashing

- Hashing has one big flaw.
- Suppose two keys **k1** and **k2** are such that **h(k1) equals h(k2),** then after storing record of **k1**, where should we add record of **k2**
- Two record can not occupy same space
- ***This condition or situation is called hash collision or clash***
- ***Another flaw is that it is difficult to retrieve sorted data.***

Collision

- There are two basic methods to handle collision:

1. **Rehashing**: In this case, a rehash function is applied successively until an empty position is found

2. **Chaining**: Build a linked list of items whose keys hash to same values

Rehashing

- **Rehashing** is a general method for resolving hash clashes also called **open addressing**
- A **rehash rh** accepts **one array index** and produces **another array index**
- If the array location h(k) is already occupied by a record of different key, **rh** is applied to this value to find another location
- This process continues until an empty location is found

Linear probing

- Linear probing is a simplified version of rehashing
- ***In this method resolving hash clashes are resolved by placing the record in the next available position in the array.***
- This means in this case rehash function rh becomes (i+1) mod 100 if hash function is: i mod 100

Chaining

- ***It builds a linked list of items whose keys map to same hash value.***
- This small linked list is traversed sequentially
- Basically, we have an array of structures where each field has key value and address field.
- If more than one key is mapped to same table location, ***a node is created to store that key*** and the **address of that node is stored in the address field.**

Properties of a good hashing function

- A good hash function is one which minimizes the collisions and spreads the records uniformly throughout the table.
- Due to this, it is desirable to have array larger than the actual number of records
- The larger the range of the hash function, the less likely it is that two keys will yield the same hash function.
- In that case, space may be wasted
- We need a solution which involves space/time tradeoff

How to choose a hash function??

- Two main criteria in selecting hash function are:
- It should be easy and quick to compute
- It should achieve even distribution of the keys that actually occurs across the range of indices
- **Truncation**: If key contains large number of digits, truncate the key (take the mod)
- **Folding**: partition the key into several parts and combine the parts and then take mod

Examining the rehash function more closely

While inserting data **K** (a key) into an array **A** through **hashing**:

1. A **hash function** is applied to K
2. Then location h(K) of array A is searched for K
   a. If the location is empty we insert the key K
   b. else if (A(h(K)) != K) then r(h(K)) (rehash function is applied) and step 2 is repeated until either empty location is found or key K is found

Note: **The above loop can execute forever if the table is full or it is not possible to insert new record through the defined hash and rehash functions**

Cont..

- We need to detect this situation and
- We have to choose a rehash function where there is a very low or zero probability that there is space in the array and it does not find it.
- We can detect the situation of table being full by keeping a counter for the records entered
- ***But it is possible for the algorithm to loop indefinitely even if there are empty locations.***
- For example if rh(i) = (i+2)%100, then any key which hashes into an odd integer rehashes into successive odd integers and any key which hashes into even integer rehashes into an even integer
- Therefore**, rh(i) = (i+c)% tablesize** is taken where c and tablesize are **relatively prime** (both can not be divided by any single integer except 1)

Deletion from hash table

- Difficult to delete in a hash table if it uses rehashing
- Deletion requires searching and then delete the key
- There are different kinds of situations:

1. The record r1 having key K is searched after applying hash function and it is found at h(K) (say location p). ***Simply delete if there were no collisions for this location p otherwise it can not be deleted and..***

We simply mark the location deleted instead of actually deleting

2. But we can not have too many deleted entries also because an unsuccessful search would require a search through entire table.