Assignment No. – 2

Title:- Dictionary Data Structure.

Problem Statement:- Implement all functions of dictionary ADT using hashing and handle collision using chaining with / without replacement. Data:- Set of pairs(key, value), keys are mapped to values, keys must be comparable, keys must be unique. Standard operations Insert(key,value), Find(key) and Delete(key).

Learning Objectives:-

1. To understand dictionary Data structure.
2. To understand hashing with chaining with replacement.
3. To implement dictionary by using hashing as well as object oriented concepts.

Learning Outcomes: - After successfully completing this assignment students are able to :-

1. Understand and implement various operations on dictionary data structure.
2. Analyze the working of different hashing implementations.
3. Implement dictionary adt by hashing technique.

Software Tools Used:- Linux GCC compiler, Eclipse Editor.

Hardware Used:- PC configuration – Intel core2 duo 2.93 GHz CPU, 4 GB RAM, 320 GB HDD,17'' Monitor.

Theory:-

Open Addressing

Like separate chaining, open addressing is a method for handling collisions. In Open Addressing, all elements are stored in the hash table itself. So at any point, the size of the table must be greater than or equal to the total number of keys (Note that we can increase table size by copying old data if needed).
Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.

Search(k): Keep probing until slot's key doesn't become equal to k or an empty slot is reached.

Delete(k): ***Delete operation is interesting***. If we simply delete a key, then the search may fail. So slots of deleted keys are marked specially as "deleted".
The insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.
Open Addressing is done in the following ways:

a) Linear Probing: In linear probing, we linearly probe for next slot. For example, the typical gap between two probes is 1 as seen in the example below.
Let **hash(x)** be the slot index computed using a hash function and **S** be the table size
If slot hash(x) % S is full, then we try (hash(x) + 1) % S

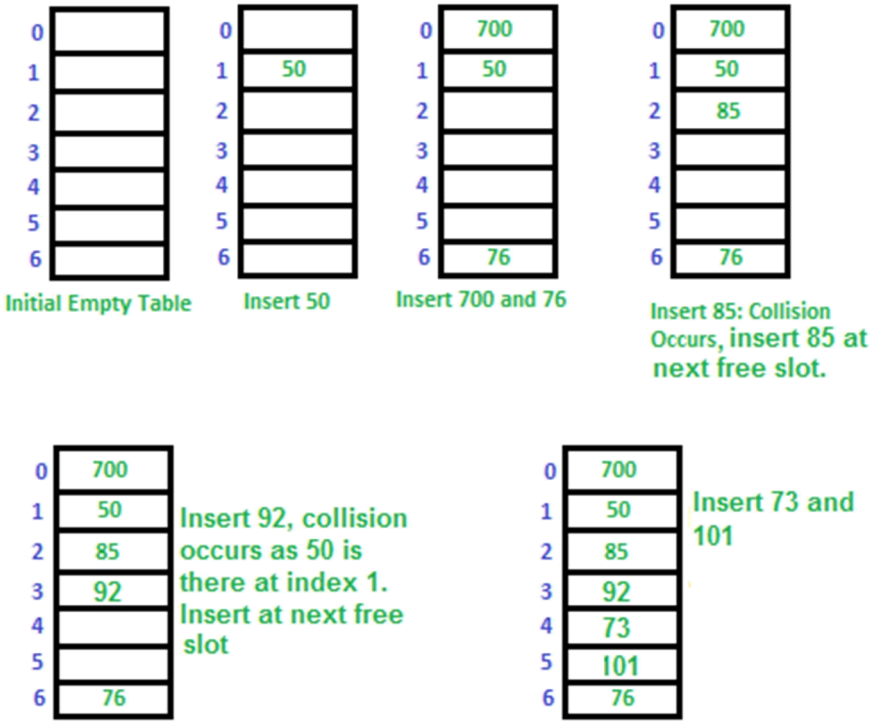
If (hash(x) + 1) % S is also full, then we try (hash(x) + 2) % S

If (hash(x) + 2) % S is also full, then we try (hash(x) + 3) % S

……

……

Let us consider a simple hash function as "key mod 7" and a sequence of keys as 50, 700, 76, 85, 92, 73, 101.



Challenges in Linear Probing :

1. **Primary Clustering:** One of the problems with linear probing is Primary clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search for an element.
2. **Secondary Clustering*:*** Secondary clustering is less severe, two records only have the same collision chain (Probe Sequence) if their initial position is the same.

Linear Probing with chaining (with replacement) :-

-Misplaced start of chain can be restored using chaining with replacement.

- so while inserting any new element there could be three situations:

a) Hashed location is empty.

- In this case new element is directly stored into hashed location.

b) Hashed location is occupied by synonym of the current element.

- It is stored according linear probing strategy & added at the end of chain.

c) Hashed location is occupied by element that is not the synonym of current element.

- Such a element is removed from the location and stored in hash table using linear probing strategy, also it is removed from its chain and added at the end of chain.

Algorithm:-

Insert(pair)

1. Start
2. HL = key % 10
3. If HL is empty, then store new record at HL, go to 6.
4. If HL is occupied by synonym , then store new record according linear probing and add new record in the chain of synonym at the end, go to 6.
5. If HL is occupied by other element, then remove old element from HL and insert it in hash table in linear probing strategy, also remove element from chain and add at the end of chain, go to 6.
6. Stop.

Search(key)

1. Start
2. HL = key %10
3. If HL contains desired record
   then return HL, go to
   Otherwise
   Sequentially search the chain for desired record till end
   If record with given found
   return its location
   Otherwise
   return -1
4. Stop

Analysis :-

Time complexity for insert and search

O(1) – best case , O(n) – worst case

Conclusion :-

Write it in your words