Open Addressing

Open Addressing: 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).

Important Operations:

- Insert(k): Keep probing until an empty slot is found. Once an empty slot is found, insert k.
- Search(k): Keep probing until the 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 the deleted keys are marked specially as "deleted".

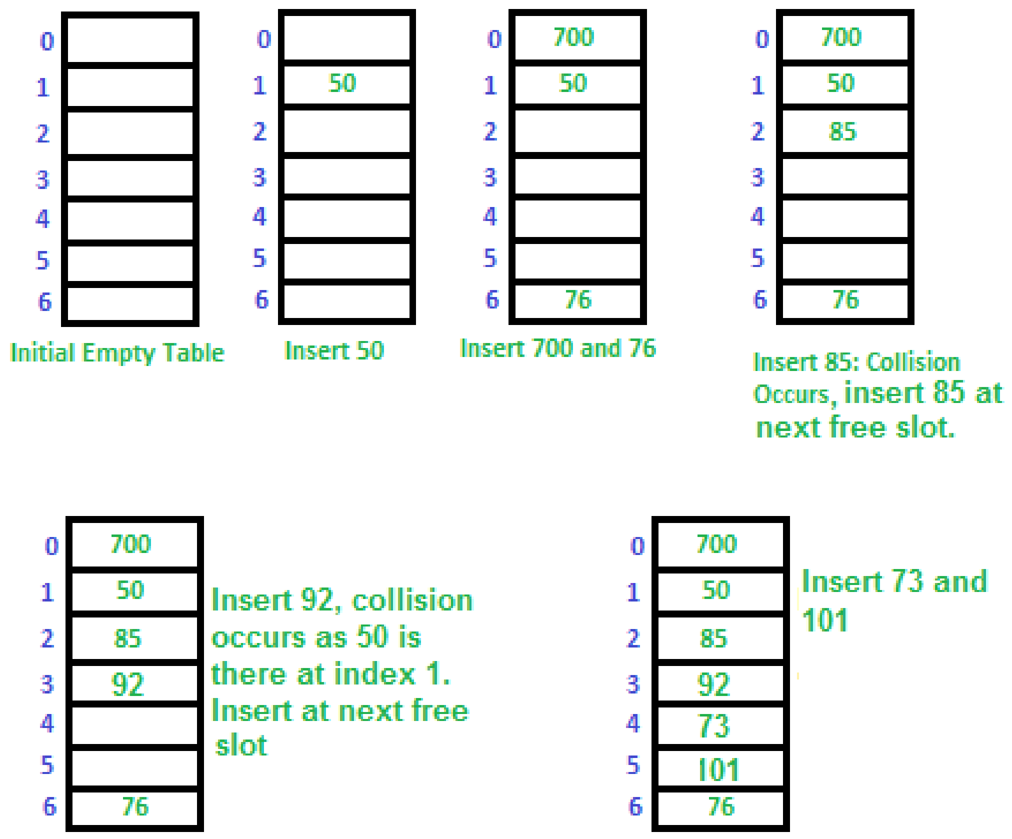
Insert can insert an item in a deleted slot, but the search doesn't stop at a deleted slot.

Open Addressing is done in the following ways:

1. ***Linear Probing:*** In linear probing, we linearly probe for the next slot. For example, the typical gap between the two probes is 1 as taken in the below example also.
   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
..................................................
..................................................
```

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


Initial Empty Table
Insert 50
Insert 700 and 76
Insert 85: Collision Occurs, insert 85 at next free slot.
Insert 92, collision occurs as 50 is there at index 1. Insert at next free slot
Insert 73 and 101
0 1 2 3 4 5 6
700
50
85
92
73
101
76

Clustering: The main problem with linear probing is clustering, many consecutive elements form groups and it starts taking time to find a free slot or to search an element.

2. ***Quadratic Probing*** We look for i'th slot in i'th iteration.

> let hash(x) be the slot index computed using hash function. If slot hash(x) % S is full, then we try (hash(x) + 1*1) % S If (hash(x) + 1*1) % S is also full, then we try (hash(x) + 2*2) % S If (hash(x) + 2*2) % S is also full, then we try (hash(x) + 3*3) % S
>
> ....

1. **Double Hashing** We use another hash function hash2(x) and look for i*hash2(x) slot in i'th rotation.

> let hash(x) be the slot index computed using hash function. If slot hash(x) % S is full, then we try (hash(x) + 1*hash2(x)) % S If (hash(x) + 1*hash2(x)) % S is also full, then we try (hash(x) + 2*hash2(x)) % S If (hash(x) + 2*hash2(x)) % S is also full, then we try (hash(x) + 3*hash2(x)) % S ....
> ....

1. See this for step by step diagrams.

Comparison of above three:

- Linear probing has the best cache performance but it suffers from clustering. One more advantage of Linear probing that it is easy to compute.
- Quadratic probing lies between the two in terms of cache performance and clustering.
- Double hashing has poor cache performance but no clustering. Double hashing requires more computation time as two hash functions need to be computed.

S.No.	**Seperate Chaining**	**Open Addressing**
1.	Chaining is Simpler to implement.	Open Addressing requires more computation.
2.	In chaining, Hash table never fills up, we can always add more elements to chain.	In open addressing, table may become full.
3.	Chaining is Less sensitive to the hash function or load factors.	Open addressing requires extra care for to avoid clustering and load factor.
4.	Chaining is mostly used when it is unknown how many and how frequently keys may be inserted or deleted.	Open addressing is used when the frequency and number of keys is known.
5.	Cache performance of chaining is not good as keys are stored using linked list.	Open addressing provides better cache performance as everything is stored in the same table.
6.	Wastage of Space (Some Parts of hash table in chaining are never used).	In Open addressing, a slot can be used even if an input doesn't map to it.
7.	Chaining uses extra space for links.	No links in Open addressing

Performance of Open Addressing:

Like Chaining, the performance of hashing can be evaluated under the assumption that each key is equally likely to be hashed to any slot of the table (simple uniform hashing).

```
m = Number of slots in the hash table
n = Number of keys to be inserted in the hash table

Load factor α = n/m  ( < 1 )

Expected time to search/insert/delete < 1/(1 - α)

So Search, Insert and Delete take (1/(1 - α)) time
```