DATA STRUCTURES

(this is an additional material and not exhaustive study material)

Indexing: Hashing - Hash Functions – Separate Chaining – Open Addressing: Linear Probing- Quadratic Probing- Double Hashing- Rehashing – Extendible Hashing.
Disjoint Sets: Basic data structure - Smart Union Algorithms - Path Compression.

HASHING

Hash Table is a data structure which stores data in an associative manner. In a hash table, data is stored in an array format, where each data value has its own unique index value. Access of data becomes very fast if we know the index of the desired data.

Thus, it becomes a data structure in which insertion and search operations are very fast irrespective of the size of the data. Hash Table uses an array as a storage medium and uses hash technique to generate an index where an element is to be inserted or is to be located from.

In hashing, large keys are converted into small keys by using **hash functions**. The values are then stored in a data structure called **hash table**. The idea of hashing is to distribute entries (key/value pairs) uniformly across an array. Each element is assigned a key (converted key). By using that key you can access the element in **O(1)** time. Using the key, the algorithm (hash function) computes an index that suggests where an entry can be found or inserted.

Hashing is implemented in two steps:

1. An element is converted into an integer by using a hash function. This element can be used as an index to store the original element, which falls into the hash table.
2. The element is stored in the hash table where it can be quickly retrieved using hashed key.

   hash = hashfunc(key)

   index = hash % array_size

In this method, the hash is independent of the array size and it is then reduced to an index (a number between 0 and array_size − 1) by using the modulo operator (%).

Hash function

A hash function is any function that can be used to map a data set of an arbitrary size to a data set of a fixed size, which falls into the hash table. The values returned by a hash function are called hash values, hash codes, hash sums, or simply hashes.

To achieve a good hashing mechanism, It is important to have a good hash function with the following basic requirements:

1. Easy to compute: It should be easy to compute and must not become an algorithm in itself.
2. Uniform distribution: It should provide a uniform distribution across the hash table and should not result in clustering.

3. Less collisions: Collisions occur when pairs of elements are mapped to the same hash value. These should be avoided.

   Note: Irrespective of how good a hash function is, collisions are bound to occur. Therefore, to maintain the performance of a hash table, it is important to manage collisions through various collision resolution techniques.

Five popular hashing functions are as follows:-

Division Method: An integer key x is divided by the table size m and the remainder is taken as the hash value. It can be defined as
H(x)=x%m+1
For example, x=42 and m=13, H(42)=45%13+1=3+1=4

Midsquare Method: A key is multiplied by itself and the hash value is obtained by selecting an appropriate number of digits from the middle of the square. The same positions in the square must be used for all keys.
For example if the key is 12345,
square of this key is value 152399025. If 2 digit addresses is required then position 4th and 5th can be chosen, giving address 39.

Folding Method: A key is broken into several parts. Each part has the same length as that of the required address except the last part. The parts are added together, ignoring the last carry, we obtain the hash address for key K.

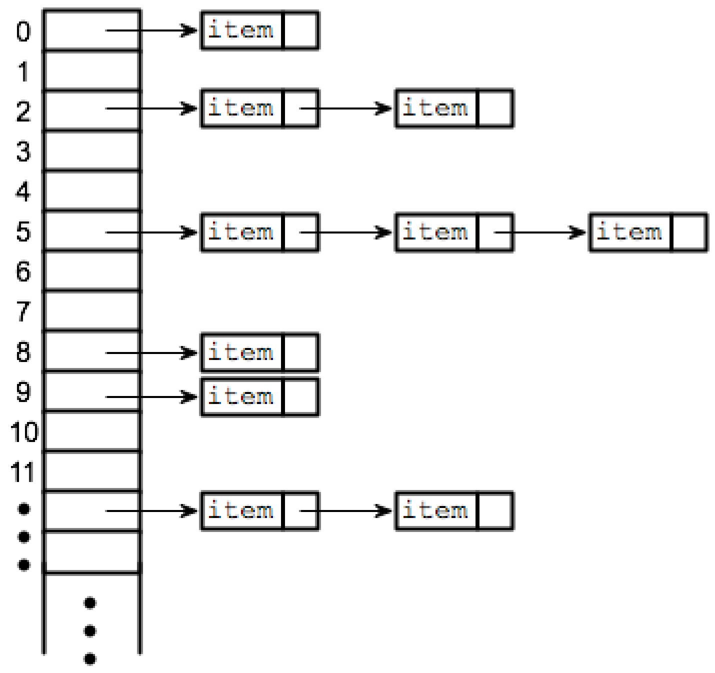
Multiplicative method: In this method a real number c such that 0<c<1 is
selected. For a nonnegative integral key x, the hash function is defined as
H(x)=[m(cx%1)]+1
Here,cx%1 is the fractional part of cx and [] denotes the greatest integer less than or equal to its contents.

Digit Analysis: This method forms addresses by selecting and shifting digits of the original key. For a given key set, the same positions in the key and same rearrangement pattern must be used. For example, a key 7654321 is transformed to the address 1247 by selecting digits in position 1,2,4 and 7 then by reversing their order.

Collision resolution techniques

Separate chaining (open hashing)

Separate chaining is one of the most commonly used collision resolution techniques. It is usually implemented using linked lists. In separate chaining, each element of the hash table is a linked list. To store an element in the hash table you must insert it into a specific linked list. If there is any collision (i.e. two different elements have same hash value) then store both the elements in the same linked list.



The cost of a lookup is that of scanning the entries of the selected linked list for the required key. If the distribution of the keys is sufficiently uniform, then the average cost of a lookup depends only on the average number of keys per linked list. For this reason, chained hash tables remain effective even when the number of table entries (N) is much higher than the number of slots.

For separate chaining, the worst-case scenario is when all the entries are inserted into the same linked list. The lookup procedure may have to scan all its entries, so the worst-case cost is proportional to the number (N) of entries in the table.

Linear probing (open addressing or closed hashing)

In open addressing, instead of in linked lists, all entry records are stored in the array itself. When a new entry has to be inserted, the hash index of the hashed value is computed and then the array is examined (starting with the hashed index). If the slot at the hashed index is unoccupied, then the entry record is inserted in slot at the hashed index else it proceeds in some probe sequence until it finds an unoccupied slot.

The probe sequence is the sequence that is followed while traversing through entries. In different probe sequences, you can have different intervals between successive entry slots or probes.

When searching for an entry, the array is scanned in the same sequence until either the target element is found or an unused slot is found. This indicates that there is no such key in the table. The name "open addressing" refers to the fact that the location or address of the item is not determined by its hash value.

Linear probing is when the interval between successive probes is fixed (usually to 1). Let's assume that the hashed index for a particular entry is **index**. The probing sequence for linear probing will be:

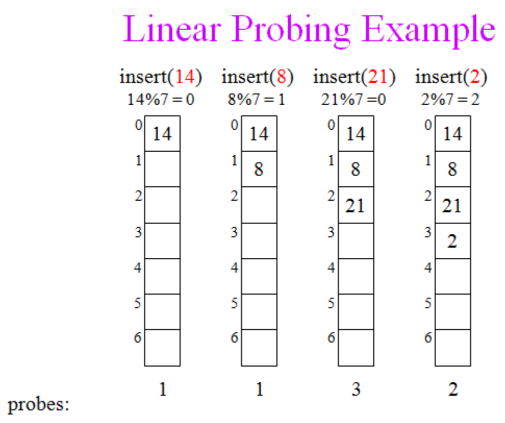
index = index % hashTableSize
index = (index + 1) % hashTableSize
index = (index + 2) % hashTableSize
index = (index + 3) % hashTableSize

and so on…

- Array-based implementation.
- All elements stored in hash table itself.
- When collisions occur, use a systematic (consistent) procedure to store elements in free slots of the table.
- Three Types of Open Addressing

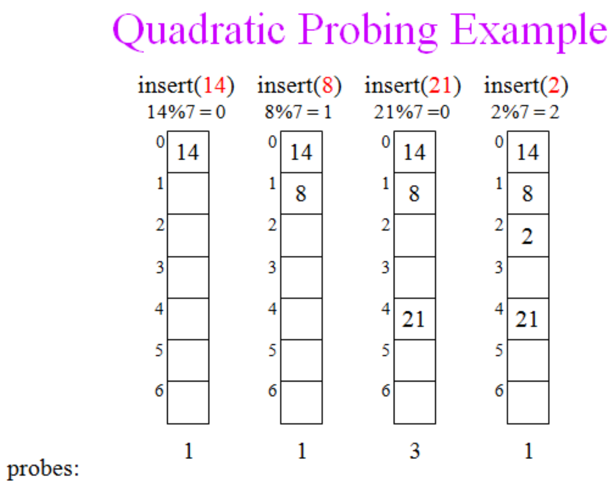
(i) Linear probing (linear search)

- When collision occurs, scan down the array one cell at a time looking for an empty cell
- $h_i(X) = (Hash(X) + i) \bmod \mathit{TableSize} \quad (i = 0, 1, 2, \ldots)$
- Compute hash value and increment it until a free cell is found



(ii) Quadratic probing (nonlinear search)

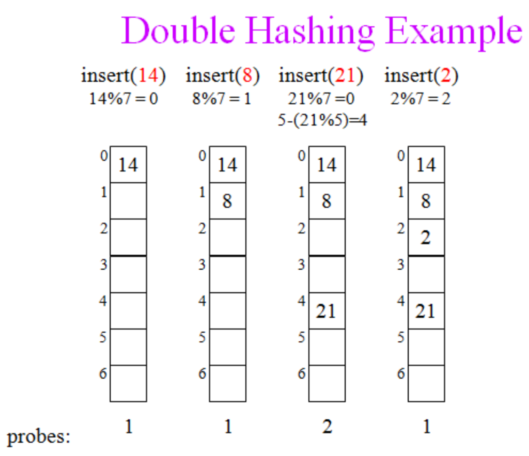
- Spread out the search for an empty slot – Increment by i^2 instead of i
- $h_i(X) = (Hash(X) + i^2) \% \mathit{TableSize}$
- $h0(X) = Hash(X) \% TableSize$
- $h1(X) = Hash(X) + 1 \% TableSize$
- $h2(X) = Hash(X) + 4 \% TableSize$
- $h3(X) = Hash(X) + 9 \% TableSize$



(iii) Double hashing (uses two hash functions)

- **Apply primary hash function**
- **If collision occurs then s**pread out the search for an empty slot by using a second hash function
- Example :

Primary Hash Function	Hash1(X)) = (X mod R)
Secondary Hash Function	$Hash_2(X) = R – (X \bmod R)$
	where R is a prime smaller than *TableSize*



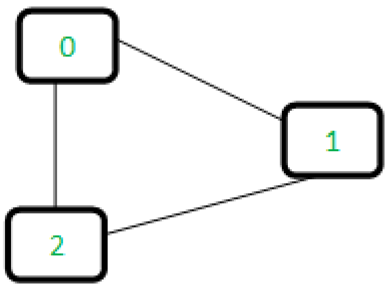
Disjoint Set :

A disjoint-set data structure is a data structure that keeps track of a set of elements partitioned into a number of disjoint (non-overlapping) subsets. A union-find algorithm is an algorithm that performs two useful operations on such a data structure:

Find: Determine which subset a particular element is in. This can be used for determining if two elements are in the same subset.
Union: Join two subsets into a single subset.

Application of disjoint-set data structure is to check whether a given graph contains a cycle or not.

Union-Find Algorithm can be used to check whether an undirected graph contains cycle or not. This is another method based on Union-Find. This method assumes that the graph doesn't contain any self-loops.
To keep track of the subsets an array is used, call it parent[].
Let us consider the following graph:



For each edge, make subsets using both the vertices of the edge. If both the vertices are in the same subset, a cycle is found.

Initially, all slots of parent array are initialized to -1 (means there is only one item in every subset).

0	1	2
-1	-1	-1

Now process all edges one by one.

Edge 0-1: Find the subsets in which vertices 0 and 1 are. Since they are in different subsets, we take the union of them. For taking the union, either make node 0 as parent of node 1 or vice-versa.

0	1	2	← 1 is made parent of 0 (1 is now representative of subset {0,1})
1	-1	-1	

Edge 1-2: 1 is in subset 1 and 2 is in subset 2. So, take union.

0	1	2	← 2 is made parent of 1 (2 is now representative of subset {0,1,2})
1	2	-1	

Edge 0-2: 0 is in subset 2 and 2 is also in subset 2. Hence, including this edge forms a cycle.

0->1->2 // 1 is parent of 0 and 2 is parent of 1