Hashing

Definition

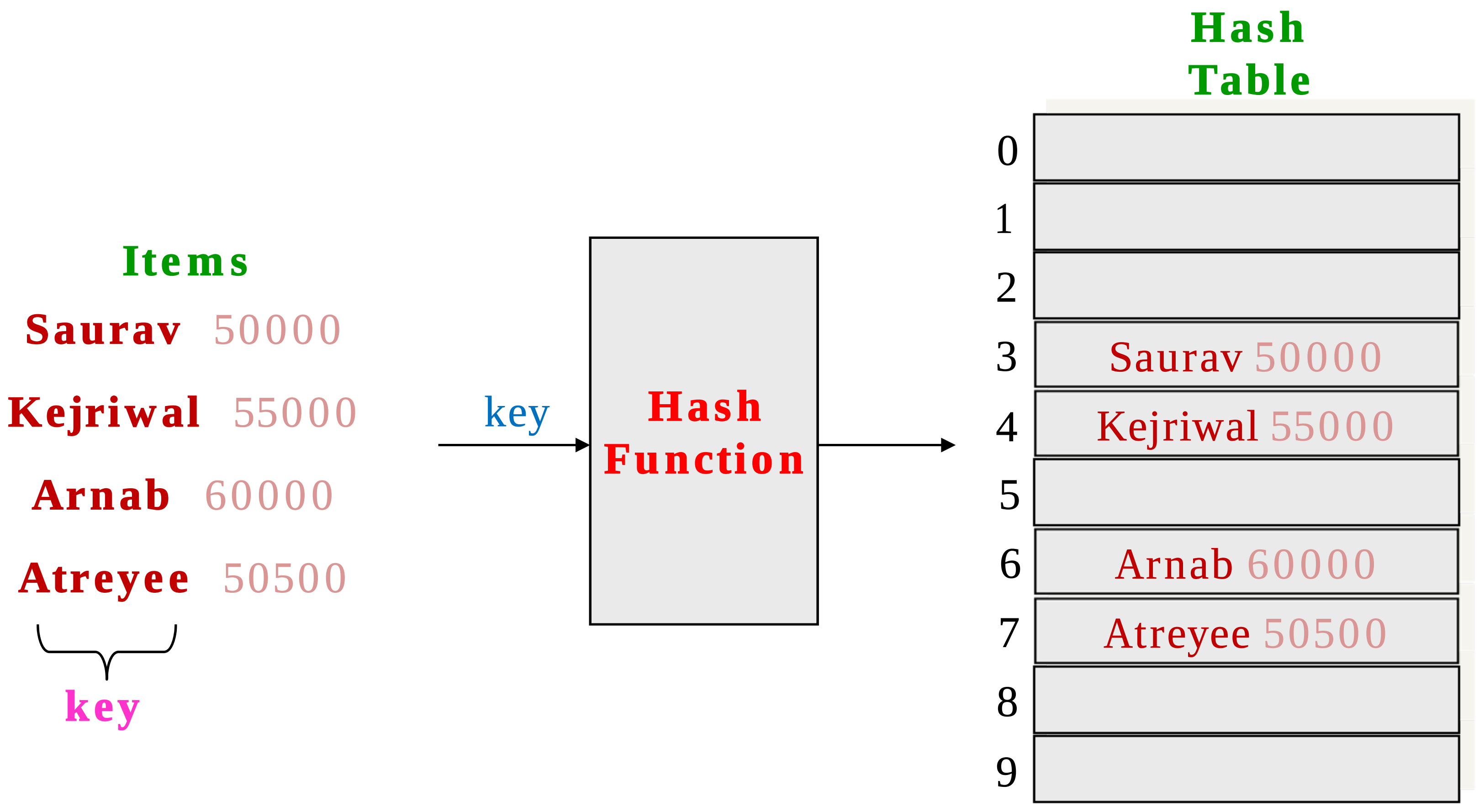
- **Hash table** is an array that holds records.
- **Hashing** is the process of *mapping* a key value to a position in a table.
- **Hash function** maps key values to positions.
- **Insertion**, **deletion**, **Searching** in a hash table can be done in **O(1)** regardless of the **hash table size**.

Definition

- This data structure, however, is ***not efficient*** in operations that require any ordering among the elements, such as findMin, findMax and printing the entire table in sorted order.

Bhaskar Sardar, Information Technology Department, *Jadavpur University, India*

Example



Items
Saurav 50000
Kejriwal 55000
Arnab 60000
Atreyee 50500
key
key
Hash Function
Hash Table
0
1
2
3 Saurav 50000
4 Kejriwal 55000
5
6 Arnab 60000
7 Atreyee 50500
8
9

Hash Functions

- **The hash function:**
  - must be **simple** to compute.
  - must distribute the keys **evenly** among all positions.
- If we know which keys will occur in advance we can write *perfect* hash functions, but we don't.

Bhaskar Sardar, Information Technology Department, Jadavpur University, India

Hash Functions

- **Problems:**
  - Keys may not be numeric.
  - Number of possible keys is much **larger** than the space available in table.
  - Different **keys** may **map** into **same location**
    - Hash function is not one-to-one **=> collision**.
    - If there are too many **collisions**, the performance of the hash table will ***suffer dramatically***.

Bhaskar Sardar, Information Technology Department, *Jadavpur University, India*

Hash Functions

- **Modular Arithmetic**
  - Hash function **h (k) = k mod N**
    - Pairs are: (22,a), (33,c), (3,d), (8,e), (85,f).
    - Hash table is T[0:7], N = 8.

(8,e)	(33,c)		(3,d)		(85,f)	(22,a)	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

Hash Functions

(8,e)	(33,C)		(3,d)		(85,f)	(22,a)	
[0]	[1]	[2]	[3]	[4]	[5]	[6]	[7]

- Where does (27,g) go?
- Keys that have the same home location are ***synonyms***.
  - 3 and 27 are synonyms with respect to the hash function that is in use.
- The location for (27,g) is already occupied.
  - **Collision occurred**

Hash Functions

- **Middle of square:**
  - h(k):= return middle digits of $\mathbf{k}^2$
- **Folding:**
  - Partition the key **k** into several parts, and add the parts together to obtain the hash address
  - e.g., k=12320324111220; partition **k** into 123, 203, 241, 112, 20; then return the address 123+203+241+112+20=699

Bhaskar Sardar, Information Technology Department, *Jadavpur University, India*

Collision Resolution

❖ Separate Chaining

- Eliminate overflows by permitting each location to maintain a **linked list** for synonyms.

❖ Open Addressing

- Ensures that all elements are stored directly into the hash table
  - **Linear Probing**
  - **Quadratic Probing**
  - **Double Hashing**

Bhaskar Sardar, Information Technology Department, *Jadavpur University, India*

Collision Resolution

- **Linear Probing**
  - resolves collisions by placing the data into the next open slot in the table
    - h(k) = k % 17
    - Insert pairs whose keys are 6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45

0				4				8				12				16
34	0	45				6	23	7			28	12	29	11	30	33

Bhaskar Sardar, Information Technology Department, Jadavpur University, India

Collision Resolution

- Linear Probing

```
int insert ( int k )
{
        int i = 0, j;
        j = hash ( k );
        do
        {
                if ( T [ j ] == false )
                {
                        A [ j ] == k;
                        T [ j ] == true;
                        return j;
                }
                i ++;
                j = ( j +i ) % N;
        } while ( i < N );
return ( -1 );
}
```

Collision Resolution

- Linear Probing

```
int search ( int k )
{
        int i = 0, j;
        j = hash ( k );
        do
        {
                if ( A [ j ] == k)
                        return i;
                i ++;
                j = ( j + i ) % N;
        } while ( ( T [ j ] == True ) && ( i < N) );
return -1;
}
```

Collision Resolution

- **Problems**
  - Keys tend to cluster together
    - Synonyms are not distributed in the hash table
  - Increase the search time
  - Problem with delete
    - a special flag is needed to distinguish deleted from empty positions.

Collision Resolution

- **Quadratic Hashing**
  - Use a **quadratic function** to compute the *next index* in the table to be probed.
    - **The idea** here is to skip regions in the table with possible clusters.
    - If the i^{th} position is occupied we check the $i+1^{st}$, next we check $i+4^{th}$, next $i+9^{th}$, etc.
  - We may not be sure that all locations in the table are probed
    - No guarantee to find a location even if the table is not full !!!

Bhaskar Sardar, Information Technology Department, *Jadavpur University, India*

Collision Resolution

- **Double Hashing**
  - One of the best methods for dealing with collisions.
  - If the location is full, then a second hash function is calculated and combined with the first hash function.
    - **Second hash function** is used to get a fixed increment for the "probe" sequence.
    - $h(k) = (h_1(k) + i\, h_2(k)) \% N$

Bhaskar Sardar, Information Technology Department, Jadavpur University, India

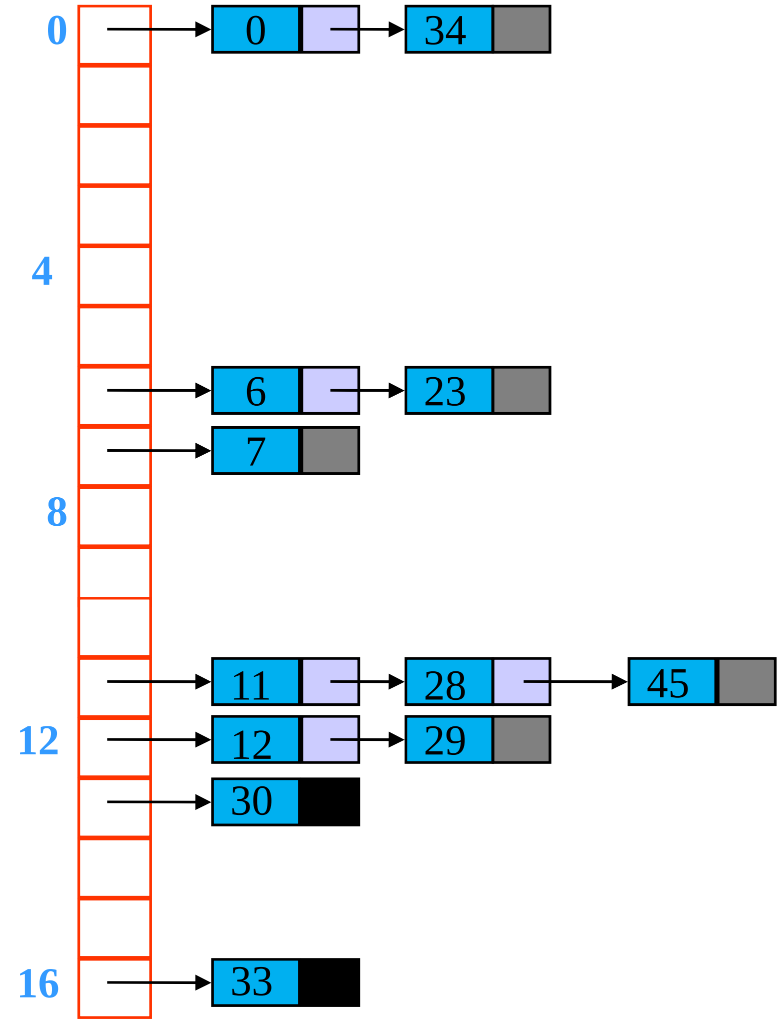
Collision Resolution

- **Separate Chaining**
  - **The idea** is to keep a list of all keys that hash to the same value.
    - The array elements are pointers to the first nodes of the lists.
    - A new item is inserted to the front of the list.
  - **Advantages:**
    - Better **space utilization** for large items.
    - Simple collision handling: searching linked list.
    - Overflow: **Stores more items** than hash table size.
    - Deletion is quick and easy: deletion from the linked list.

Collision Resolution

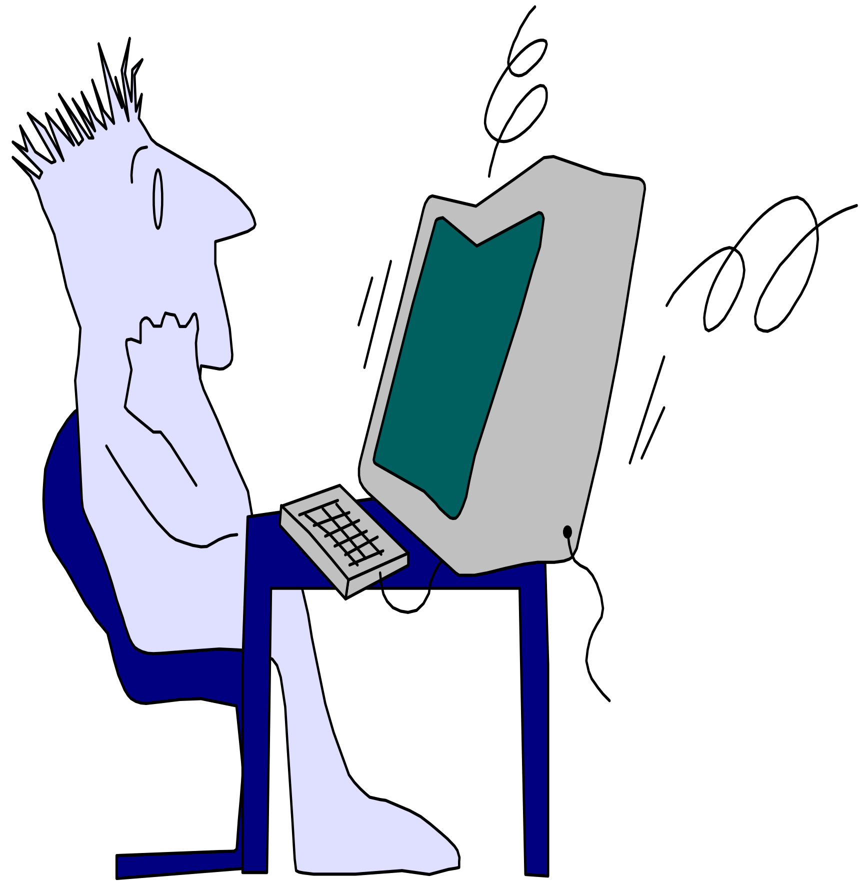
- **Separate Chaining**
  - Store following keys:

    6, 12, 34, 29, 28, 11, 23, 7, 0, 33, 30, 45
  - **h (k) = k % 17**



Any Doubt ?

- Please feel free to write to me:



bhaskargit@yahoo.co.in