HASH TABLE

What is Hash Table?

→ It is a data structure that stores data in an associative manner such that we can access data easily if we know its index number.

It stores elements in pair of Index value (key) and data.

Hash Function (Hashing):

→Hashing is the process of converting a given key into another value. A hash function is used to generate the new value according to a mathematical algorithm.

Let K be a key and h(x) be a hash function then h(K) gives us the new hash value of the stored element with key K.

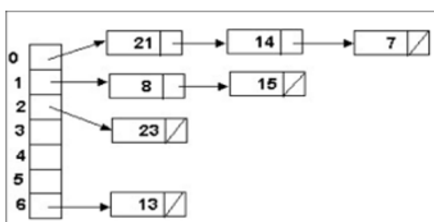
Hash Collision:

→ It is possible that the hash function generates same hash value for different key, this will lead to conflict. This is called Hash collision.

How to avoid Hash Collision?

→ There are different techniques that are implemented to avoid or reduce Hash Collision.

- Collision resolution by chaining- If same hash values are generated for different key value then elements are stored at the same index value by making use of linked list. Each index has its own linked list.



❖ Open Addressing- It does not make use of any linked list but in this case the size of hash table must be larger than the total number of key values.

- Linear Probing: It a slot of particular hash value is occupied us check some other slot by using some linear equation.
  Ex: let h(x) be some hash function for some key K then,
  h”(x,i)={h(x)+i}%(size of table) ;i =0,1,2……number of attempts to solve collision; is the hash function that checks for the next slot if there is collision.
  But it leads to formation of clusters of data blocks (Primary Clustering) and if any key hashes into that cluster then it will need several attempts of traversal to solve collision.

- Quadratic Probing: Unlike Linear probing we move i^2 spots from the hash value position where we had the collision.
  h”(x, i) = {h(x) +ai+bi^2} %(size of table);
  i =0, 1, 2……number of attempts to solve collision.
  Drawback: When the table is more than half full it becomes very difficult to find an empty slot.

- Double hashing: If there is a collision we use a secondary hash function h2(K)= m-(K%m);m is any prime number less that size of table
  h”(K)= {h1(K)+ih2(K)}%m.
  h2 (K) must never evaluate to 0.

Implementation:

Rabin-Karp Algorithm:-

This Algorithm is a commonly used pattern-search Algorithm. So what exactly do we mean by pattern searching?
A formal definition would say that given a text [0…n-1] and a pattern[0…m -1] print all occurrence of that pattern[] in the text[].
Ex: Text: The Ice is cold.
Pattern: Ice.
Introduction:-
So, the algorithm slides the pattern one by one and match the hash value of the pattern, with the hash value of current substring of text, and if the hash values matches then only it start matching individual characters.
The use of hashing converts the string to a numeric which speeds up testing the equality of pattern and the substring. The algorithm exploits the fact that if two strings are equal, their hash values are also equal. Thus string matching is reduced to computing the hash value of the search pattern and then looking for substring of the text with that hash value.
Here we assume the length of text is 'N' is greater than length of the pattern 'M'.
So we calculate hash value of our pattern.
Complexity:-

- Best Case and average Case- O(m+n)
- Worst Case- O (m*n); When all the character of text and pattern are same.

Hash Function By Rabin-Karp
Requirement: Hash at the next shift must be efficiently computable (O(1)) from the current hash value and next character in text.

Hash Value of pattern:
$h(x) = \sum (\text{Value/weight of character}) * d^{(m-1)}) \ \% \ q.$

- d= total number of characters usually taken 256 (You can assume any suitable d)
- m= total length of pattern
- q= any suitable prime number.

If hash value of text and pattern match then character matching is performed and if not then hash value of next window is calculated by subtracting first term and adding the next term.

Rehashing (Hash Value of next text window):
hash(txt[s…s+m]) = d (hash(txt[s…s+m-1] - (txt[s] * h)+ txt[s+m]) %q

Code:

```
#include <stdio.h>
#include <string.h>
#define d 256
void numOfPattern (char pattern[], char text[], int q){
    int m = strlen(pattern);
    int n = strlen(text);
    int i, j;
    int p = 0;
    int t = 0;
    int h = 1;
    //The value of h would be pow(d,m -1)%q
    for(i = 0; i < m-1; i++)
    {
        h = (h*d)%q;
    }
    for(i = 0; i < m; i++)
    {
        p = (d*p + pattern[i])%q;
        t = (d*t + text[i])%q;
    }
    for(i = 0; i <= n - m; i++){
        /*
           Check the hash values of current window of the text
           and pattern. If the hash value matches then only check
           for characters one by one.

        */
       if(p == t){
           //Check for the characters one by one.
           for(j = 0; j < m; j++){
               if(text[i+j]!=pattern[j]){
                   break;
               }
```

```
            }
            //if p == t & pat[0...m-1]=txt[i,i+1,..i+m-1]
            if(j == m){
                printf("Pattern found at position :%d \n",i+1);
            }
        }
        /*
         Calculate hash value for next window of text:
         remove the leading digit, add the trailing digit.
        */
       if(i < n-m){
           t = (d*(t-text[i]*h)+text[i+m])%q;
           //we might get negative value of t, convert it into positive.
           if(t<0){
               t = t+q;
           }
       }
     }
}
int main()
{
  char text[] = "DDPPDPDD";
  char pattern[] = "DD";
  int q = 11;
  numOfPattern(pattern, text, q);
}
```