Hashing

Hash function

(I.) Division method

$h(k) = k \bmod m$

(II.) Multiplication method

$$h(k) = \lfloor m(k \cdot A \bmod 1) \rfloor$$

$$A = 0.6180 = \frac{\sqrt{5}-1}{2}$$

(III) .midsquare method.

$h(k) = s$

where s is obtained from deleting digits from both sides of k^2

ex $k = 2345$

$k^2 = 5499025$

taking 4^{th} & 5^{th} digit from right

$\boxed{h(k) = 99}$.

(IV) Folding method

Step I :- key value is divided into no. of parts ($k_1, k_2 \ldots k_n$)

Step 2 → these parts are added together,

→ hash value is obtained by ignoring last carry, if any.

Ex $k = 9235$

parts = 92, 35

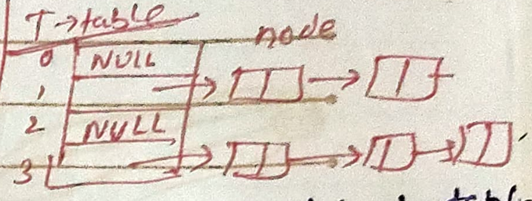
sum of parts = 127

$h(k) = 27$

Resolving collisions

(1.) separating chaining or synonyms chaining

↳ each hash table slot T[i] contains a linked list of all the keys whose hash value is i.



initialize - chained hash-table

```
for (i=0; i<=m; i++){
    t[i] = NULL;
}
```

searching (T(x)→ size of LL)

```
node * search(note *t[], int x){
    node* ptr = t[h(x)];
    while(ptr != NULL && ptr→data != x)
        ptr = ptr → next
    if (ptr→data == x)
        return ptr;
    else
        return NULL
}
```

Inserting

```
{node * ptr = (node*) malloc(sizeof())
 ptr→data = x
 ptr→next = t[h(x)];
 t[h(x)] = ptr;
}
```

deletion

delete node in LL.

(2) Open addressing

↳ each entry of hash table either contain element or some sentinal value (i.e -1) to indicate that slot is free.

↳ if slot is filled, then other slots are examined systematically, to find free slot, if no slot found then overflow condition occured

Probing → process of examining the slots in hash table

Open addressing
- (I) linear probing
- (II) quadratic probing
- (III) double hashing

(I) Linear probing → very easy to implement but → it suffers from a problem known as **primary clustering**

$$\boxed{h(k,i) = [h'(k) + i] \bmod m}$$

$i = 0, 1, 2, \ldots, m-1$ → probe no.

$$\boxed{h'(k) = k \bmod m}$$

primary clustering refers to many such blocks separated by free slots

1	2	3	–	5	4	–	–

first take $i=0$
if collision occurs $i = i+1$
again check and repeat process

(II) Quadratic Probing (better that linear probing)

$$\boxed{h(k,i) = [h'(k) + c_1 i + c_2 i^2] \bmod m}$$

$i = 0, 1, 2, \ldots, m-1$

$$\boxed{h'(k) = k \bmod m}$$

firstly $i=0$, if collision $i=1$ and so on.

(III) Double Hashing (best method for open addressing as it uses two hash function)

$$\boxed{h(k,i) = [h_1(k) + i h_2(k)] \bmod m}$$

$i = 0, 1, \ldots m-1$

$\boxed{h_1(k) = k \bmod m}$, $\boxed{h_2(k) = k \bmod m'}$

m' = less than m
i.e. $(m-1), (m-2)$

ReHashing

If at any stage the hash table ~~becomes nearly full~~, the running ~~time~~ for the operation will ~~start taking~~ too much time, and even the insert operation may fail for open addressing with quadratic probing. This can happen if there are too many deletions intermixed with too many insertions.

In such a situation :-

1. create a new hash-table of size double than the original hash table.
2. scan the original hash table, and for each key, compute new hash value and insert into new hash table.
3. free the memory occupied by the original hash table.

Direct addressing

↳ applicable only when we can allocate an ~~array that has one position for every possible key~~.