University Practicals Question Bank:

8.i
Use digit extraction method(1st ,3rd,and 5th)for hashing the following values 224562,140145,14467,137456,214576,199645,214562,162145,234534 in an array of 19 elements. Use linear probe method to resolve any collisions.

8.ii
Use Fold shift method for hashing the following values 224562, 140145, 14467, 137456, 214576, 199645, 214562, 162145, 234534 in an array of 19 elements. Use linear probe method to resolve any collisions.

8.iii
Use Fold boundary method for hashing the following values 224562,140145,14467,137456,214576,199645,214562,162145,234534 in an array of 19 elements. Use linear probe method to resolve any collisions.

8.iv
Use direct hashing method to insert the keys 99, 33, 23, 44, 56, 43, 19 in an array of 10 elements. Use linear probe method to resolve any collisions.

Note: In this question the array size is mentioned as 10. But according to the direct hashing (hashkey is the element itself). So, these given numbers cannot be accommodated in the array size of 10, so the array size is changed to 100 and is solved using the same.

8.v
Use subtraction hashing method to insert the keys 99, 33, 23, 44, 56, 43, 19 in an array of 10 elements. Use linear probe method to resolve any collisions.

Note: in this question the array size is mentioned as 10. But according to the subtraction technique (list size-key) cannot be accommodated in the array size of 10, so it is to be solved using modulo division instead of subtraction.

8.vi
Use modulo division hashing method to insert the keys 55, 65, 20, 12, 66, 26, 90 in an array of 13 elements. Use linear probe method to resolve any collisions.

Solution

Refer to the C++ source code below.

C++ Code for Different Hashing (collision Resolution using linear probe):

```
#include<iostream.h>
#include<stdlib.h>
#include<conio.h>
#include<stdio.h>
#include<string.h>

const int SIZE = 19;
const int PHYSIZE = 20;

class hash
{
        private:

        long int *hashtable;
        int filled;
        int collisions;
        int size,psize;

        public:

        hash();
        hash(int);

        void store(long int,int);
        void printtable();
        void printdensity();

        void modulodiv(long int);
        void foldshift(long int);
        void foldboundary(long int);
        void digitextraction(long int);
        void direct(long int);

};
hash :: hash()
{
        int i;
        size=SIZE;
        psize=PHYSIZE;
        hashtable=new long int[PHYSIZE];

        for(i=0;i<PHYSIZE;i++)
        {
                hashtable[i]=0;
        }
        collisions=0;
        filled=0;
```

```
}
hash :: hash(int s)
{
        int i;

        size=s;
        psize=s+1;

        hashtable=new long int[psize];

        collisions=0;
        filled=0;

        for(i=0;i<psize;i++)
        {
                hashtable[i]=0;
        }

}
void hash :: printdensity()
{
        cout<<"\nNo.of Collisions:"<<collisions<<endl;
        cout<<"\nFilled:"<<filled<<endl;
        cout<<"\nDensity:"<<((filled*100)/size)<<" %";

}
void hash :: printtable()
{
        for(int i=0;i<psize;i++)
        {

                cout<<"\n"<<i<<"."<<hashtable[i];
        }
}
void hash :: store(long int num,int hashkey)
{
        while(hashtable[hashkey]!=0 && hashkey<=size)
        {
                hashkey++;
                collisions=collisions+1;
        }
        filled++;
        if(hashkey>size)
        {
                hashkey=0; //wrap around
        }
        hashtable[hashkey]=num;
}

//hashing functions
void hash :: modulodiv(long int n)
{
        int key;
        key =(n%size)+1;
        store(n,key);
}

void hash :: foldshift(long int n)
{
        int div=0,rem,key;
        long int temp=n;

        while(temp!=0)
        {
                rem=temp%100;
                div=div+rem;
                temp=temp/100;

        }

        key=(div%size)+1;
        store(n,key);

}

void hash :: foldboundary(long int num)
{
        int addn,key;
        char r1[3],r2[3],r3[3];
        char no[7];
        for(int i=0;i<3;i++)
        {
                r1[i]='\0';
                r2[i]='\0';
                r3[i]='\0';

        }
        ltoa(num,no,10);
        strncat(r1,no+1,1);
        strncat(r1,no,1);

        strncat(r2,no+2,2);
        strncat(r3,no+5,1);
        strncat(r3,no+4,1);
        addn=atoi(r1)+atoi(r2)+atoi(r3);

        key=(addn%size)+1;
        store(num,key);

}
void hash::digitextraction(long int n)
{
        long int t=n;
        int p1=t%100;
        t=t/100;
        int p2=t%100;
        t=t/100;
        int p3=t;

        p1=p1/10;
        p2=p2/10;
        p3=p3/10;

        int tot=(p3*100)+(p2*10)+p1;

        int nkey=(tot % size)+1;

        store(n,nkey);
}

void hash::direct(long int key)
{
        hashtable[key]=key;
}

int main()
{
        clrscr();
        int choice=0;
        int n,i,s;
        while(choice<6)
        {
        cout<<"\n Choose a Hashing Technique:";
        cout<<"\n 1.Modulo Division"
            <<"\n 2.Fold shift"
            <<"\n 3.Fold boundary"
            <<"\n 4.Direct hashing"
            <<"\n 5.Digit extraction"
            <<"\n 6.Exit"
            <<"\n Enter your option:";
        cin>>choice;
```

```
switch(choice)
{
        case 1:
        {
                hash h1(13);
                cout<<"\nfor practical 8.vi";
                long int userkeys[]={55, 65, 20, 12, 66, 26, 90};
                for(int x=0;x<7;x++)
                {
                        h1.modulodiv(userkeys[x]);
                }

                h1.printdensity();
                h1.printtable();

                hash h2(10);
                cout<<"\nfor practical 8.v";
                long int userkeys[]={99, 33, 23, 44, 56, 43, 19 };

                for(int x=0;x<7;x++)
                {
                        h2.modulodiv(userkeys[x]);
                }
                h2.printdensity();
                h2.printtable();
                break;
        }
        case 2:
        {

                hash h1(19);
                long int userkeys[]={224562, 140145, 14467, 137456,
                                14576, 199645, 214562, 162145, 234534 };
                for(int x=0;x<9;x++)
                {
                        h1.foldshift(userkeys[x]);
                }
                h1.printdensity();
                h1.printtable();
                break;
        }
        case 3:
        {
                hash h1(19);
                long int userkeys[]={224562, 140145, 14467, 137456,
                214576, 199645, 214562, 162145, 234534 };
                for(int x=0;x<9;x++)
                {
                        h1.foldboundary(userkeys[x]);
                }
                h1.printdensity();
                h1.printtable();
                break;
        }
        case 4:
        {

                hash h1(100);
                long int userkeys[]={99, 33, 23, 44, 56, 43, 19};
                for(int x=0;x<7;x++)
                {
                        h1.direct(userkeys[x]);
                }
                h1.printdensity();
                h1.printtable();
                break;
        }
        case 5:
        {
                hash h1(19);
                long int userkeys[]={224562, 140145, 14467, 137456,
                        214576, 199645, 214562, 162145, 234534 };
                for(int x=0;x<n;x++)
                {
                        h1.digitextraction(userkeys[x]);
                }
                h1.printdensity();
                h1.printtable();
                break;
        }
        case 6:
        {
                break;
        }
```

```
                        default:
                        {
                                cout<<"\nInvalid choice";
                                break;
                        }
                }
        }


getch();
return 0;
}
```

University Theory Questions:

8.vii: Explain the following:

a. Hashing along with an algorithm.

Ans.: Hashing is the process of mapping large amount of data item to a smaller table with the help of hashing function. The implementation of hash table is called hashing.
In a hashed search, the key, through an algorithmic function, determines the location of the data. Because we are searching an array, we use a hashing algorithm to transform the key into the index that contains the data we need to locate. Another way to describe hashing is a key-to-address transformation in which the keys map to addresses in a list.

Key ---------→ Hash Function ---------→ Address

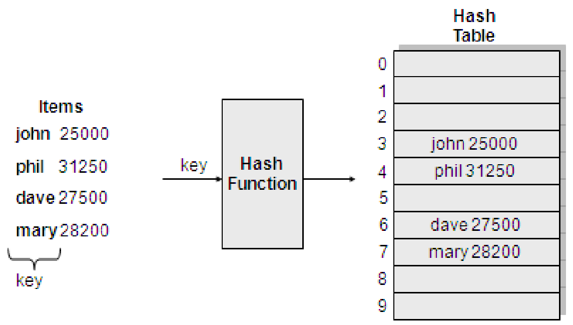
<u>**Hashing Algorithim**</u>

Algorithm hash (Key, size, maxAddr, addr)
This algorithm converts an alphanumeric key of size characters into an integral address.

```
    Pre     key is a key to be hashed
            Size is the number of characters in the key
            maxAddris maximum possible address for the list
    Post    addr contains the hashed address

    1   set looper to 0
    2   set addr to 0
        Hash key
    3   for each character in key
        1   if ( character not space )
            1   add character to address
            2   rotate addr 12 bits right
        2   end if
    4       end loop

    Test for negative address

    5  if  ( addr < 0 )
           1   addr =  absolute ( addr )
    6 end if
    7 addr = addr modulo maxAddr
end hash
```

b. Synonym, collision, prime area and home address.

Ans.: Generally, the population of keys for a hashed list is greater than the storage area for the data. Because there are many keys for each index location in the array, more than one item may hash to the same location in the array. We call the set of keys that hash to the same location in our list synonyms.



If the actual data that we insert into our list contain two or more synonyms, we can have collisions.
A Collision occurs when a hashing algorithm produces an address for an insertion key and that address is already occupied. The address produced by the hashing algorithm is known as the home address The memory that contains all of the home address is known as the prime area. When two keys collide at a home address, we must resolve the collision by placing one of the keys and its data in another location.

c. Probe, load factor.

Ans.: When we need to locate an element in a hashed list, we must use the same algorithm that we used to insert it into the list. Consequently, we first hash the key and check the home address to determine whether it contains the desired element. If it does, the search is complete. If not, we must use the collision resolution algorithm to determine the next location and continue until we find the element or determine that it is not in the list. Each calculation of an address and test for success is known as a probe.

As a rule of thumb, a hashed list should not be allowed to become more than 75% full.
Load factor
The Load factor of a hashed list is the number of elements in the list divided by the number of physical elements allocated for the list, expressed as a percentage. Traditionally, Load factor is assigned the symbol alpha (α). The formula in which k represents the number of filled elements in the list and n represents the total number of elements allocated to the list is:

$$\alpha = \frac{k}{n} \times 100$$

d. Pseudo random method and key offset.

Ans.: In pseudorandom hashing the key is used as the seed in a pseudorandom-number generator and the resulting random number is the scaled into the possible address range using modulo division method. Given a fixed seed, pseudorandom-number generators always generate the same series of numbers. That is what allows us to use them in hashing. A common random-number generator is shown below:

Y = ax + c

To use the pseudorandom-number generator as a hashing method, We set x to the key, Multiply it by the coefficient a, and then add the constant c. The result is then divided by the list size, with the remainder being the hashed address. For maximum efficiency, the factors **a** and **c** should be prime numbers.

Example: We use 17 and 7 for factors a and c, respectively. Also, the list size in the example is the prime number 307.

Y = ((17 * 121267) + 7) modulo 307
Y = (2061546 + 7) modulo 307
Y = 2061546 modulo 307
Y = 41

Key Offset:

Key offset is a double hashing method that produces different collisions paths for different keys. Key offset calculates the new address as a function of the old address and the key. One of the simplest versions simply adds the quotient of the key divided by the list size to the address to determine the next collision resolution address, as shown in the formula below:

Offset = [key / listsize]
address = ((offset + old address) modulo listsize)

Example:

When the key is 166702 and the list size is 307, using the modulo-division hashing method generates an address of 1.

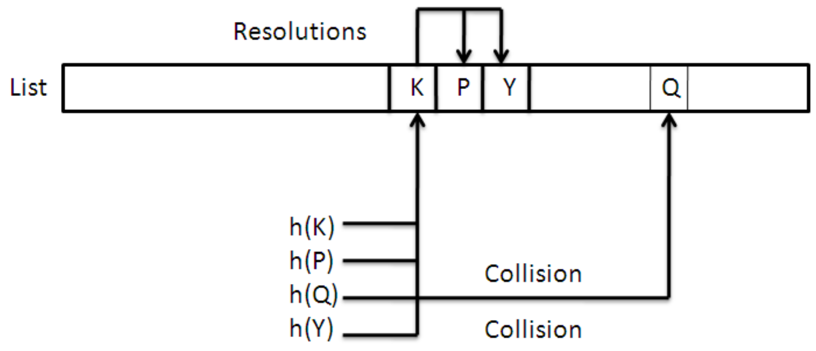
offset = [166702 / 307] = 543
address = ((543 + 001) modulo 307) = 237

If 237 were also a collision, we would repeat the process to locate the next address, as shown below:

Offset = [166702 / 307] = 543
address = ((543 = 237) modulo 307) = 166.

e. Clustering.

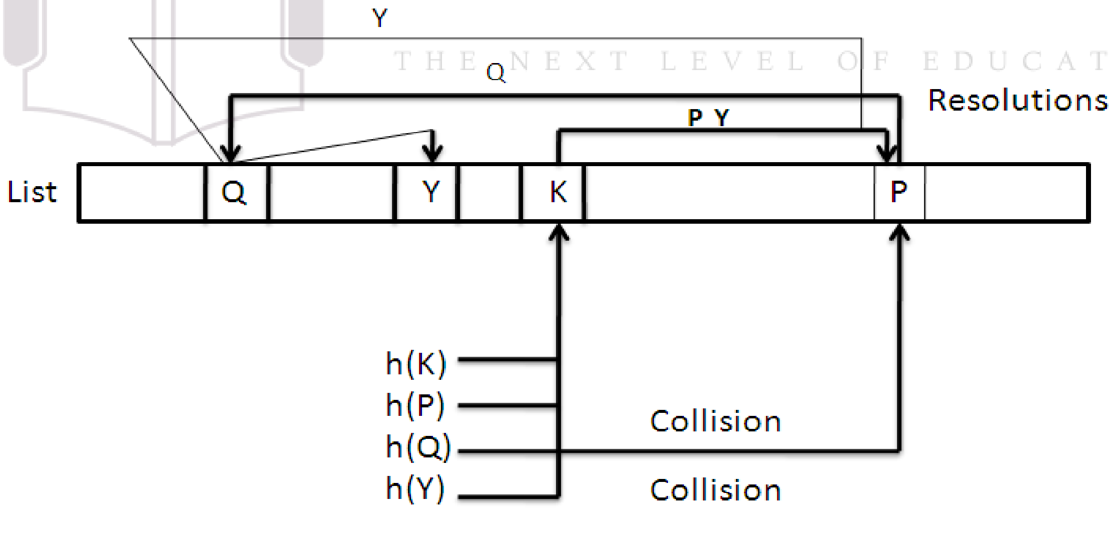
Ans.: As data are added to a list and collisions are resolved, some hashing algorithims tend to cause data to group within the list. This tendency of data to build up unevenly across a hashed list is known as clustering. Computer scientists have identified two distinct types of cluster. The first, primary clustering, occurs when data cluster around a home address. Primary clustering is easy to identify.



(a)Primary clustering

In Figure (a), we see that K, P, and Y cluster around K's home address. In this example the collision resolution is based on the home address. Q, on the other hand, hashes to its own address remotely located from K's home address.

Secondary clustering occurs when data become grouped along a collision path throughout a list. This type of clustering is not easy to identify. In secondary clustering the data are widely distributed across the whole list, so the list appears to be well distributed. If the data all lie along a well- traveled collision path, However, the time to locate a requested element of data can increase.

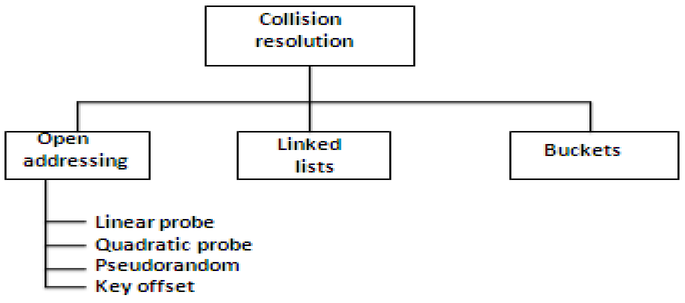


(b) Secondary clustering

In the Figure (b), the same four keys are inserted into the list. The collision resolution in this example is not based on the home address; it spreads the collisions across the entire list. Thus, we see that while K, P, and Y are still synonyms that hash to the same home address. They are not clustered around K's home address; they are spread across the file. Also note that because P was placed in Q’s home address, a secondary collision was created.

f. The different types of Collision resolution.

Ans.: When we hash a new key to an address, we may create a collision. There are several methods for handling collisions, each of them independent of the hashing algorithm. That is each hashing method can be used with each of the collision resolution methods:



Open addressing:

The first collision resolution method, open addressing, resolves collisions in the prime area- That is, the area that contains all of the home address. This technique is opposed to linked list resolution, in which the collisions are resolved placing the data in a separate overflow area. We discuss four different methods: Linear probe, quadratic probe, double-hashing and key offset.

In Linear probe, which is the simplest, when data cannot be stored in the home address we resolve the collision by adding 1 to the current address. As an alternative to a simple linear probe, we can add 1, subtract 2, add 3, Subtract 4, and so forth until we locate an empty element.
For example,given a collision at location 341, we would try 342, 340, 343, 339, and so forth until we located an empty.
Linear probes have two advantages.
First, they are quite simple to implement. Second, data tend to remain near there home address.

In the Quadratic probe, the increment is the collision probe number squared. Thus for the first probe we add 1^2 , for the second collision probe we add 2^2 , for the third collision probe we add 3^2 , and so forth until we either find an empty element or we exhaust the possible elements.
A potential disadvantage of the quadratic probe is the time required to square the Probe number. The quadratic probe has one limitaion: it is not possible to generate a new address For every element in the list.

In pseudorandom hashing the key is used as the seed in a pseudorandom-number generator, and the resulting random number is the scaled into the possible address

range using modulo-division method. Given a fixed seed, pseudorandom-number generators always generate the same series of numbers. That is what allows us to use them in hashing. A common random-number generator is shown below:
Y = ax + c

Key Offset:

Key offset is a double hashing method that produces different collisions paths for different keys. Key offset calculates the new address as a function of the old address and the key. One of the simplest versions simply adds the quotient of the key divided by the list size to the address to determine the next collision resolution address, as shown in the formula below:

Offset = [key / listsize]
address = ((offset + old address) modulo listsize)

Example:
When the key is 166702 and the list size is 307, using the modulo-division hashing method generates an address of 1.

offset = [166702 / 307] = 543
address = ((543 + 001) modulo 307) = 237

If 237 were also a collision, we would repeat the process to locate the next address, as shown below:

Offset = [166702 / 307] = 543
address = ((543 = 237) modulo 307) = 166.

Bucket hashing

Another approach to handling the collision problems is bucket hashing, In which keys are hashed to buckets, nodes that accommodate multiple data occurrences. Because a bucket can hold multiple data, collisions are postponed until the bucket is full. There are two problems with this concept. First, it uses significantly more space because many of the buckets are empty or partially empty at any given time. Second, it does not completely resolve the collision problem.

.

8.viii

a) **May 2012 Q2 a)**

What is hashing? Explain the terms synonym, collision and home address.
(Refer to 8.vii above)

Using <u>digit extraction</u> (1st, 3rd and 5th) method for hashing and <u>linear probing</u> method for collision resolution; store the keys given below in an array of 19 elements. How many collisions occurred? Determine the density of the list.
224562, 137456, 214562, 140145, 214576, 162145.
Sol :

Hashing: ***<u>Digit Extraction (1,3,5) +</u>*** ***<u>Modulo Division</u>***

Collision Resolution: ***<u>Linear Probe</u>***

<u>Hashing calculation</u>

- 224562 $\rightarrow$ 246 % 19 + 1 = 19
- 137456 $\rightarrow$ 175 % 19 + 1 = 5
- 214562 $\rightarrow$ 246 % 19 + 1 = 19
    19 + 1 -> 20 % 19 + 1 = 2 (1)
- 140145 $\rightarrow$ 104 % 19 + 1 = 10
- 214576 $\rightarrow$ 247 % 19 + 1 = 1
- 162145 $\rightarrow$ 124 % 19 + 1 = 11

<u>Hash list (Collision)</u>

1 - - 214576
2 - - 214562 (1)
3 - -
4 - -
5 - - 137456
6 - -
7 - -
8 - -
9 - -
10 - -140145
11 - -162145
12 - -
13 - -
14 - -
15 - -
16 - -
17 - -
18 - -
19 - - 224562

Collisions : 1
Density : 6 / 19 ~ 32 %

b) May 2011 Q2 b)

What is hashing?
(Refer to 8.vii above)

Using the digit-extraction method (1, 3, 5) & linear probing, store the keys shown below in an array with 19 elements. How many collisions occurred? What is the density of the list after the keys have been inserted?
25668, 14512, 24578, 22501, 27867, 13651, 15934, 21890, 71211.

Sol :

Hashing: ***Digit Extraction (1,3,5) + Modulo Division***

Collision Resolution: ***Linear Probe***

Hashing calculation

- 25668 → 268 % 19 + 1 = 3
- 14512 → 152 % 19 + 1 = 1
- 24578 → 258 % 19 + 1 = 12
- 22501 → 251 % 19 + 1 = 5
- 27867 → 287 % 19 + 1 = 3
  3 + 1-> 4 **(1)**
- 13651 → 161 % 19 + 1 = 10
- 15934 → 194 % 19 + 1 = 5
  5 + 1-> 6 **(2)**
- 21890 → 280 % 19 + 1 = 15
- 71211 → 721 % 19 + 1 = 19

Hash list (Collision)

1 - - 14512
2 - -
3 - - 25668
4 - - 27867 (1)
5 - - 22501
6 - - 15934 (2)
7 - -
8 - -
9 - -
10 - - 13651
11 - -
12 - - 24578
13 - -
14 - -
15 - - 21890
16 - -
17 - -
18 - -
19 - - 71211

Collisions : 2
Density : 9 / 19 ~ 47 %

c) Dec 2010 Q4 b)

Define clustering in hash list. (Refer to 8.vii above)

Using mid-square method and key offset, store the keys shown below in array of size 13: 55, 65, 20, 12, 66, 26, 90.

Sol :

Hashing: ***Mid-Square***

Collision Resolution: ***Key Offset***

Hashing calculation

- 55 $\rightarrow 55^2$ = 3025 -> 02 % 13 + 1 = 3
- 65 $\rightarrow 65^2$ = 4225 -> 22 % 13 + 1 = 10
- 20 $\rightarrow 20^2$ = 0400 -> 40 % 13 + 1 = 2
- 12 $\rightarrow 12^2$ = 0144 -> 14 % 13 + 1 = 2 **(1)**
  [12/13]=0 + 2 = 2 % 13 + 1 =3
  [12/13]=0 + 3 = 3 % 13 + 1 =4
- 66 $\rightarrow 66^2$ = 4356 -> 35 % 13 + 1 = 10 **(2)**
  [66/13]=5 +10 = 15 % 13 +1 =3
  [66/13]=5 + 3 = 8 % 13 + 1 = 9
- 26 $\rightarrow 26^2$ = 0676 = 67 % 13 + 1 = 3 **(3)**
  [26/13] = 2 + 3 = 5 % 13 +1 = 6
- 90 $\rightarrow 90^2$ = 8100 -> 10 % 13 + 1 = 11

Hash list (Collision)

1 - -
2 - - 20
3 - - 55
4 - - 12 (1)
5 - -
6 - - 26 (3)
7 - -
8 - -
9 - - 66 (2)
10 - - 65
11 - -90
12 - -
13 - -

Collisions : 5
Density : 7 / 13 ~ 53 %

(d) May 2010 Q2 a)

What is hashing? Define the terms collision, probe and the load factor.
(Refer to 8.vii above)

Insert the keys 99 33 23 44 56 43 19. Using the **division method** and **quadratic probing** as the collision resolution method into a list of size 10. Also find the number of collisions, probes for each element and the density of the list.

Sol :

Hashing: ***Division(Modulo-division)***

Collision Resolution: ***Quadratic Probe***

Hashing calculation

- 99 → 99 % 10 + 1 = 10
- 33 → 33 % 10 + 1 = 4
- 23 → 23 % 10 + 1 = 4 **(1)**
  $4 + 1^2 = 5$ % 10+1= 6
- 44 → 44 % 10 + 1 = 5
- 56 → 56 % 10 + 1 = 7
- 43 → 43 % 10 + 1 = 4 **(2)**
  $4 + 1^2 = 5$ % 10 +1= 6
  $6 + 2^2 = 10$ % 10 +1=1
- 19 → 19 % 10 + 1 = 10 **(3)**
  $10 + 1^2 = 11$ %10 +1=2

Hash list (Collision)

1 - - 43 (2)
2 - - 19 (3)
3 - -
4 - - 33
5 - - 44
6 - - 23 (1)
7 - - 56
8 - -
9 - -
10 - - 99

Collisions : 4
Density : 7 / 10 ~ 70 %

No. of Probes for each element:

Collision at 23: Probes =1
Collision at 43: Probes =2
Collision at 19: Probes =1

e) Dec 2009 Q5 a)

Define hashing(Refer to 8.vii above)

Using Modulo-Division method and linear probing, store the keys shown below in the array with 19 elements. How many collisions occurred? What is the density of the list after all keys have been inserted?
224562, 137456, 214562, 140145, 214576, 162145, 144467, 199645, 234534.

Sol :

Hashing: <u>Modulo Division</u>
Collision Resolution: <u>Linear Probe</u>

<u>Hashing calculation</u>

-224562 % 19 + 1 = 2
-137456 % 19 + 1 = 11
-214562 % 19 + 1 = 15
-140145 % 19 + 1 = 2 -> 3 **(1)**
-214576 % 19 + 1 = 10
-162145 % 19 + 1 = 19
-144467 % 19 + 1 = 11 -> 12 **(2)**
-199645 % 19 + 1 = 13
-234534 % 19 + 1 = 18

<u>Hash list (Collision)</u>

1-
2 - 224562
3- 140145 (1)
4 -
5 -
6 -
7 -
8 -
9 -
10 - 214576
11 - 137456
12 - 144467 (2)
13 - 199645
14 -
15 - 214562
16 -
17 -
18 - 234534
19 - 162145

Collisions : 2
Density : 9 / 19 ~ 47%

f) May 2009 Q2 a)

Using midsquare hashing method store the keys given below in an array of size 1000.

224562, 137456, 214562, 140145, 214576, 162145, 144467, 199645, 234534.

Because of the key length square only the first three digits of the key. Use pseudorandom number generator for rehashing is collisions occur (Take a = 3 and c = -1 as factors).

(**Note:** Since the array size mentioned is 1000 which cannot be represented. So the size assumed is 19)

Sol :

Hashing : **Mid-square (first 3 digits)^2 % 19 + 1**
Collision Resolution : **Random Number Generator**
(3 * address – 1) % 19 + 1

Hashing calculation

- 224562 = 224^2 050176 -> 017 % 19 +1 = 18
- 137456 = 137^2 018769 -> 876 % 19 +1 = 3
- 214562 = 214^2 045796 -> 579 % 19 +1 = 10
- 140145 = 140^2 019600 -> 960 % 19 +1 = 11
- 214576 = 214^2 045796 -> 579 % 19 +1 = 10
    (3 * 10 – 1) % 19 + 1 = 11
    (3 * 11 – 1) % 19 + 1 = 14 (1)
- 162145 = 162^2 026244 -> 624 % 19 +1 = 17
- 144467 = 144^2 020736 -> 073 % 19 +1 = 17
    (3 * 17 – 1) % 19 + 1 = 13 (2)
- 199645 = 199^2 039601 -> 960 % 19 +1 = 11
    (3 * 11 – 1) % 19 + 1 = 14
    (3 * 14 – 1) % 19 + 1 = 4 (3)
- 234534 = 234^2 054756 -> 475 % 19 + 1 = 1

Hash list (Collision)

1 - 234534
2 -
3 - 137456
4 - 199645 (3)
5 -
6 -
7 -
8 -
9 -
10 - 214562
11 - 140145
12 -
13-144467 (2)
14-214576 (1)
15 -
16 -
17 - 162145
18 - 224562
19 -

Collisions : = 5
Density : = 9 / 19 ~ 47 %

g) May 2008 Q3 b)

Define hash search. (Refer to 8.vii above)

Using the **rotation method** for hashing and **linear probe** method for resolving collision store the keys shown below in an array with 19 elements. First rotate the rightmost 2 digits to the left and then use the digit extraction (1st , 3rd and 5th digits)254761, 167953, 244664, 160849, 254872, 182741, 174767, 129449, 234534. How many collisions occurred? What is the density of the list after all keys have been inserted?

Sol :

Hashing: **<u>Rotation Method (rotating the rightmost two digits to the left),then digit extraction (first, third and fifth digits).</u>**
Collision Resolution: **<u>Linear Probe</u>**

<u>Hashing calculation</u>

254761 = 612547 = 624 % 19 + 1 = 17
167953 = 531679 = 517 % 19 + 1 = 5
244664 = 642446 = 624 % 19 + 1 = 17 -> 18 (1)
160849 = 491608 = 410 % 19 + 1 = 12
254872 = 722548 = 724 % 19 + 1 = 3
182741 = 411827 = 412 % 19 + 1 = 14
174767 = 671747 = 614 % 19 + 1 = 7
129449 = 491294 = 419 % 19 + 1 = 2
234534 = 342345 = 324 % 19 + 1 =
2 -> 3 -> 4 (2)

<u>Hash list (Collision)</u>

1 -
2 - 129449
3 - 254872
4 – 234534 (2)
5 - 167953
6 -
7 - 174767
8 -
9 -
10 -
11 -
12 - 160849
13 -
14 - 182741
15 -
16 -
17 - 254761
18 - 244664 (1)
19 -

Collisions : = 3
Density : = 9 / 19 ~ 47 %

h) May 2001 Q3 a)

Using the digit-extraction method (first, third and fifth digits) & quadratic probing, store the keys shown below in an array with 19 elements. How many collisions occurred? What is the density of the list after all keys have been inserted? 224562, 137456, 214562, 140145, 214576, 162145, 144467, 199645, 234534.

Sol :

Hashing : **<u>Digit Extraction (1,3,5) +Modulo Division</u>**
Collision Resolution :**<u>Quadratic Probe</u>**

<u>Hashing calculation</u>

- 224562 → 246 % 19 + 1 = 19
- 137456 → 175 % 19 + 1 = 5
- 214562 → 246 % 19 + 1 = 19
  19 + 1^2 -> 20 % 19 + 1 = 2 (1)
- 140145 → 104 % 19 + 1 = 10
- 214576 → 247 % 19 + 1 = 1
- 162145 → 124 % 19 + 1 = 11
- 144467 → 146 % 19 + 1 = 14
- 199645 → 194 % 19 + 1 = 5
  5 + 1^2 = 6 % 19 + 1 = 7 (2)
- 234534 → 243 % 19 + 1 = 16

<u>Hash list (Collision)</u>

1 - - 214576
2 - - 214562 (1)
3 - -
4 - -
5 - - 137456
6 - -
7 - - 199645 (2)
8 - -
9 - -
10 - -140145
11 - - 162145
12 - -
13 - -
14 - - 144467
15 - -
16 - - 234534
17 - -
18 - -
19 - - 224562

Collisions :2
Density :9 / 19 ~ 47%