Lab 3
Perfect Hashing

Name : Ahmed Khaled

ID : 9

-Universal Hashing :

-The Matrix method is used to generate a universal hash family.

```
public void generateUniversalHF()
{

    h = new int[b];
    Random r = new Random();

    for (int i = 0; i < b; i++) {
        h[i] = r.nextInt();
    }

}
```

Generating a random matrix h of size (b x 32) , b = log2(M) , M : Table Size .

```
public int h(int k )
{
    if(b==0)
        return 0 ;

    StringBuilder sb = new StringBuilder();
    int numOfOnes ;

    for (int i = 0; i < b; i++) {

        numOfOnes = Integer.bitCount(k&h[i]);
        sb.append(numOfOnes%2);
    }

    return Integer.parseInt(sb.toString(),2);
}
```

The Hashing Function : multiplying the matrix h by the key k using the bitwise operation (AND) between each row in the matrix h and the key k and counting the number of ones mod 2 to generate its index in the hash table.

-O(N^2)-Space Solution :

```
public void buildTable() throws Exception {

   if(elements == null)
       throw new Exception("EmptyElementsListException") ;

   int n = elements.size();
   int b = (int) Math.floor(Math.log(n*n)/Math.log(2));

   uh = new UniversalHashing(b) ;

   boolean collision;
   int idx , e ;

    do
    {

        table = new Object[n*n];
        collision  = false ;
        uh.generateUniversalHF();

        for (int i = 0; i < elements.size(); i++) {
            e=elements.get(i);
            idx = uh.h(e);

            if(table[idx]!=null && ((int)table[idx])!=e)
            {
                numOfTries++;
                collision =true ;
                break;
            }

            table[idx] = e;
        }

    }while(collision);

    elements = null ;
}
```

1-Calculate b for a table of size N^2.

2-Constructu Table of Size N^2 .

3-Generate a random matrix h of size (bx32) for universal hashing .

4- Try to hash each element into the hash table using the matrix h
If there is any collision re-build the table and generate a new random matrix for universal hashing (go to Step #2) .

-O(N)-Space Solution :

```
public void buildTable() throws Exception {

    if(elements == null)
           throw new Exception("EmptyElementsListException") ;

     int idx ;
     int n = elements.size();
     int b = (int) Math.floor(Math.log(n)/Math.log(2));

     table = new PerfectHashTableNN[n];
     uh = new UniversalHashing(b) ;
     uh.generateUniversalHF();

     for (int i = 0; i < elements.size(); i++) {

         idx = uh.h(elements.get(i));

         if(table[idx]==null)
         {
             table[idx] = new PerfectHashTableNN();
         }

         ((PerfectHashTableNN)table[idx]).insert(elements.get(i));
    }
     for (int i = 0; i < elements.size(); i++) {
        if(table[i]!=null)
        {
            try{
             ((PerfectHashTableNN)table[i]).buildTable();
            }catch(Exception e)
            {
                e.printStackTrace();
            }
        }
    }

     elements = null ;
 }
```

1-Calculate b for a table of size N for the first-level hash function .

2-Construct a table of size N .

3-Hash each element in the bins of the table .

4-For each bin build a PerfectHashTable that was implemented in O(N^2)-Space Solution using the elements stored in each slot in the first-level of hashing .

- Calculating the required space to construct the two-level hashing table and verifying that it is consuming O(N)-Space .

```
public void printSize()
{
    int sum = 0 ;
    for (int i = 0; i < table.length; i++) {
        if(table[i]!=null)
        {
             sum +=((PerfectHashTableNN)table[i]).table.length;
        }
    }

    System.out.printf("The Size of the table ="+sum+ "=%.1f n\n",(sum*1.0/table.length));
}
```

Sample runs:

N=20

-O(N)

```
* The Size of the table = 50= 3.5 n

* Number of times of re-building :
---------------------------------
  -Bin #0 :0
  -Bin #1 :0
  -Bin #2 :0
  -Bin #3 :0
  -Bin #4 :0
  -Bin #5 :0
  -Bin #6 :0
  -Bin #7 :0
  -Bin #8 :0
  -Bin #9 :0
  -Bin #10 :0
  -Bin #11 :0
  -Bin #12 :0
  -Bin #13 :1
  -Bin #14 :1
  -Bin #15 :0
  -Bin #16 :0
  -Bin #17 :0
  -Bin #18 :0
  -Bin #19 :0

* Number of times to re-build all bins =2
```

-O(N^2)

```
The Size Of The Table = 400
Number of times to re-build the hash table =0
```

N=30

-O(N)

```
* The Size of the table = 64= 3.1 n

* Number of times of re-building :
---------------------------------
  -Bin #0 :0
  -Bin #1 :0
  -Bin #2 :0
  -Bin #3 :1
  -Bin #4 :0
  -Bin #5 :0
  -Bin #6 :0
  -Bin #7 :0
  -Bin #8 :1
  -Bin #9 :0
  -Bin #10 :0
  -Bin #11 :0
  -Bin #12 :1
  -Bin #13 :0
  -Bin #14 :1
  -Bin #15 :0
  -Bin #16 :0
  -Bin #17 :0
  -Bin #18 :0
  -Bin #19 :0
  -Bin #20 :0
  -Bin #21 :0
  -Bin #22 :0
  -Bin #23 :0
  -Bin #24 :0
  -Bin #25 :0
  -Bin #26 :0
  -Bin #27 :0
  -Bin #28 :0
  -Bin #29 :0

* Number of times to re-build all bins =4
```

-O(N^2)

```
The Size Of The Table = 900
Number of times to re-build the hash table =1
```