**Linear probing :**

1)Make a structure of "image" with an image array of 784 places to store the pixels of the image.

2)Make a structure of "data\_item" with data of type image and key.

3)Make an array of data\_item(s) of 1000 places for 1000 images.

4)Initialize the hashtable by setting all keys to (-1)

5) In the main we read the file with all images and their ids and we use the function (put) to put it in the hashtable, we then use switch(case) to choose whether we will search for the image and use function (get) or will delete the image and use function (removeImg).

Function hashcode:

-It takes the image as a parameter and initializes sum to 0.  
- It loops to add the elements of the each row in an index of an array of 28 places as they are 28 rows.  
-It loops to multiply the sum of the row to its number.  
-It loops to calculate the sum of all elements of the array.  
-It returns sum%1000.

Function put :

-It takes the image array and the id as parameters.  
-It undergoes some mathematical operations in function (hashcode) to get the place where the image will be stored.  
-It checks if this place is empty in the hashtable then it adds the image to it.  
-Else it loops until it finds the next empty place in the hashtable and adds the image to it.  
-If it reaches the end of the array[1000] it starts from zero again to search for an available place.

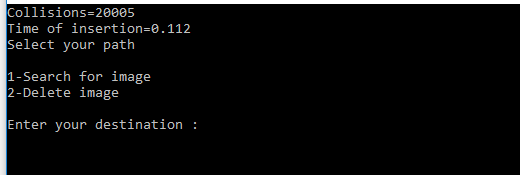
Function get:

-It takes in the image as a parameter  
-It calculates the hashcode of the image to find its expected place.  
-It loops for a 1000 times as long as (c is not equal to 784) and it searches the calculated place if it finds the image it returns its id, if not it searches all the other places until it finds it and returns its id, if it didn't find it it returns -1.

Function removeImg:

-It takes in the image as a parameter  
-It calculates the hashcode of the image to find its expected place.  
-It loops for a 1000 times as long as (c is not equal to 784) and it searches the calculated place if it finds the image it returns its id and sets the key of the hashcode to -1, if not it searches all the other places until it finds it and returns its id and sets the key of the hashcode to -1, if it didn't find it it returns -1.

**Time of insertion and collisions:**



**Separate Chaining :**

1)Make a structure of "image" with an image array of 784 places to store the pixels of the image.

2)Make a structure of "data\_item" with data of type image and key.

3)Make a struct of node with data\_itm and a next node,and make an array of 1000 pointers to nodes and set them to null.

4)Make a function to create a newnode of the given value and set its next to null.

5) In the main we read the file with all images and their ids and we use the function (put) to put it in the hashtable, we then use switch(case) to choose whether we will search for the image and use function (get) or will delete the image and use function (removeImg).

Function hashcode:

-It takes the image as a parameter and initializes sum to 0.  
- It loops to add the elements of the each row in an index of an array of 28 places as they are 28 rows.  
-It loops to multiply the sum of the row to its number.  
-It loops to calculate the sum of all elements of the array.  
-It returns sum%1000.

Function put :

-It takes the image array and the id as parameters.  
-It undergoes some mathematical operations in function (hashcode) to get the place where the image will be stored.  
-It puts the information of the image in a newnode.  
-Checks if the current hashtable points to null then it sets it to point to the new node.  
-Else it loops in the list till it reaches the last node and sets its next to the new added node.

Function get:

-It takes in the image as a parameter  
-It calculates the hashcode of the image to find its expected place.  
-It the hashtable points at null at this place it returns -1.  
-Else It loops until the last node in the list if it finds the image it returns its id else it  
 returns -1.

Function removeImg:

-It takes in the image as a parameter  
-It calculates the hashcode of the image to find its expected place.  
-It the hashtable points at null at this place it returns -1.  
-Else It loops until the last node in the list if it finds the image it returns its id and sets the hashtable to point to the next node else it returns -1.

**Time of insertion and collisions :**

