Data structures I

HASH TABLES
(ASSIGNMENT 3)

Marwan ElSafty | 4690 | 2/5/2018

(1) Linear probing:

Main Functions Used:

- **getHT(image img):**

```
1. int getHT(image img) {
2.     int code, index, loops = 0; //2 VARIABLES USED TO ITERATE AND CHECK AT THE SAME TIME

3.     code = index = hashCode(img);
4.     while ((HashTable[index].available == 1 || hashCode(HashTable[index].data) != code) &
   & loops < SIZE) {
5.         index = (index + 1) % 1000; //CIRCULAR ARRAY
6.         loops++;
7.     }
8.     if (hashCode(HashTable[index].data) == code) return HashTable[index].key;
9.     else //RETURN -1 IF THE IMAGE DOESN'T EXIST
10.             return -1;
11.      }
```

- **removeHT(image img):**

```
1. int removeHT(image img) {
2.     int code, index, loops;
3.     code = index = hashCode(img);
4.     while ((HashTable[index].available == 1 || hashCode(HashTable[index].data) != code) &
   & loops < SIZE) {
5.         index = (index + 1) % 1000; //CIRCULAR ARRAY
6.         loops++;
7.     }
8.     if (hashCode(HashTable[index].data) == code) {
9.         int ID = HashTable[index].key;
10.             HashTable[index].available = 1;
11.             HashTable[index].key = -1;
12.             free( & HashTable[index].data);
13.             return ID;
14.         } else //IF THE IMAGE DOESN'T EXIST
15.             return -1;
16.      }
```

- **putHT(int key , image current_image):**

```
1. void putHT(int key, image current_image) {
2.     int index, code, loops = 0;
3.     index = code = hashCode(current_image);
4.     while ((HashTable[index].available == 1 || hashCode(HashTable[index].data) != code) &
   & loops < SIZE) {
5.         index = (index + 1) % 1000; //CIRCULAR ARRAY
6.         loops++;
7.     }
8.     HashTable[index].key = key;
9.     HashTable[index].data = current_image;
10.          HashTable[index].available = 0; // NOT AVAILABLE ANYMORE
11.      }
```

- **hashCode(image img):**

```
1. int hashCode(image img) {
2.     long int i, j, rowSum = 0, totalSum = 0;
3.     for (i = 0; i < ROW; i++) {
4.         for (j = 0; j < COLUMN; j++) {
5.             rowSum += img.img_arr[i * ROW + j];
6.         }
7.         totalSum += rowSum * (i + 1);
8.         rowSum = 0;
9.     } //MOD SIZE TO MAKE SURE THE INDEX CALCUATED DOESN'T OVERFLOW THE ARRAY SIZE
10.          totalSum = totalSum % 1000;
11.          return (int) totalSum;
12.      }
```

Extra Functions Used:

- **generateDatabase():**

```
void generateDatabse() {
    FILE * f = fopen("MNIST-data(datastructures_assignment3)_whitespace.txt", "r");
    intializeHT();
    image tempImg;
    int tempKey;
    int i, j;
    for (i = 0; i < SIZE; i++) {
        for (j = 0; j < IMAGE_SIZE; j++) { //SCAN THE PIXELS
            fscanf(f, "%d", & tempImg.img_arr[j]);
          } //SCAN THE ID
          fscanf(f, "%d", & tempKey); //PUT THE IMAGE AND ITS ID IN THE DATABASE
          putHT(tempKey, tempImg);
      }
      fclose(f);
  }
```

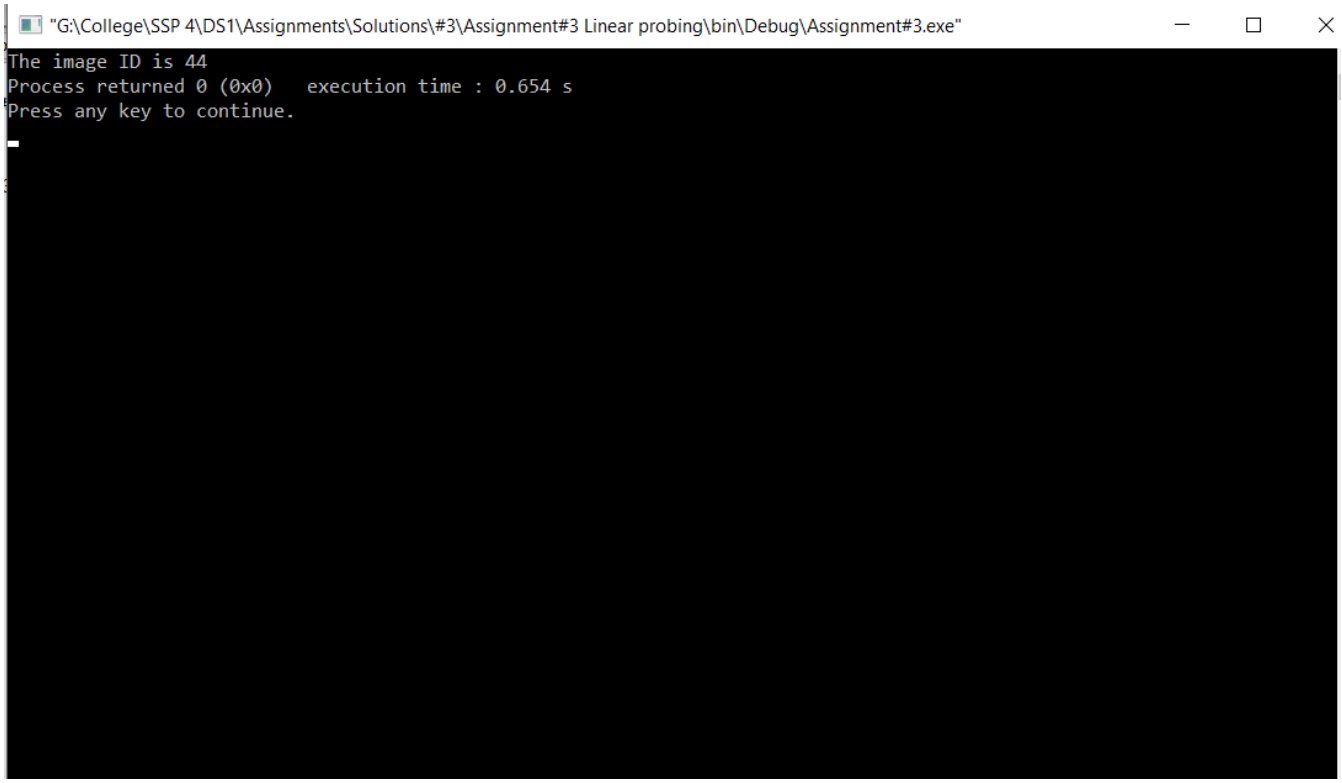
- **readTestFile(image * testImg):**

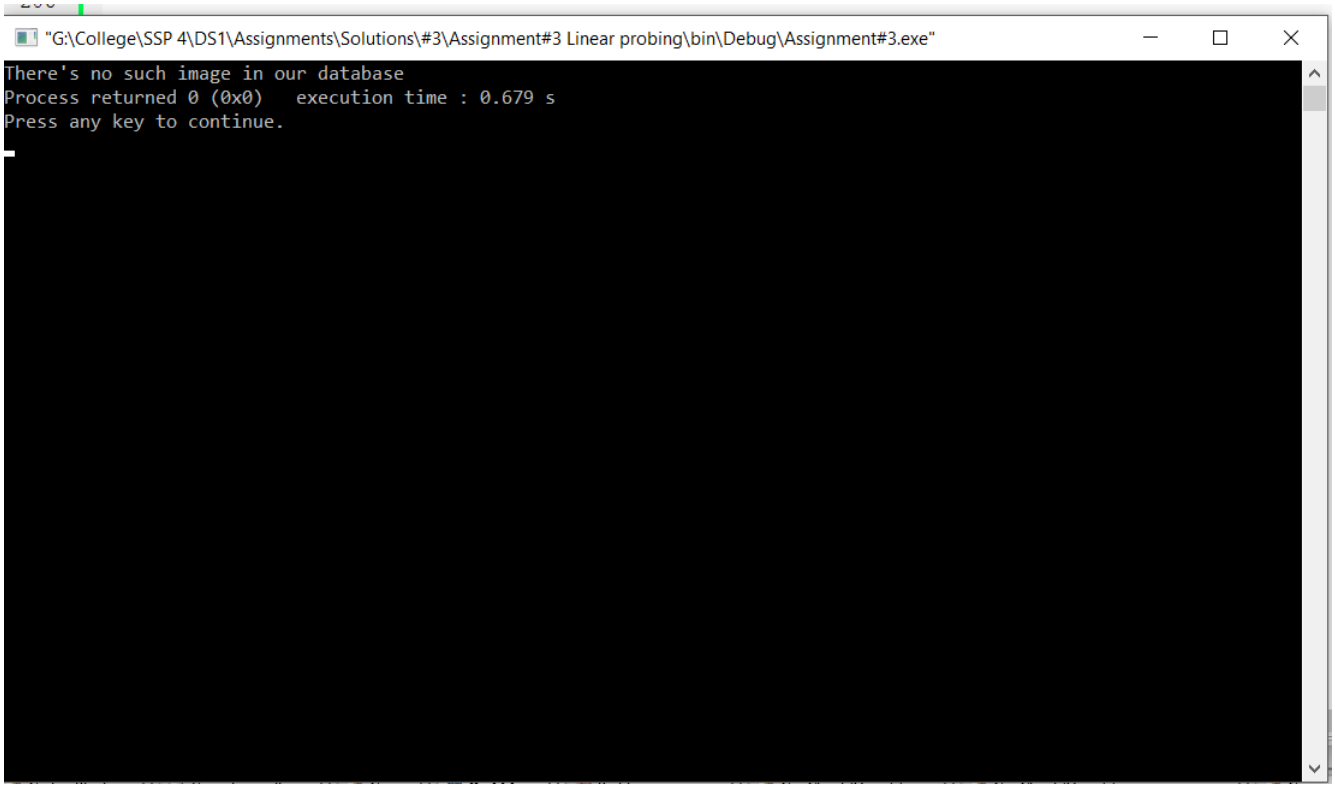
```
void readTestFile(image * testImg) {
    FILE * imgFile = fopen("test.txt", "r");
    int i = 0;
    for (i = 0; i < 784; i++) {
        fscanf(imgFile, "%d", & testImg - > img_arr[i]);
    }
    fclose(imgFile);
}
```

- **initializeHT():**

```
void intializeHT() {
    for (int i = 0; i < SIZE; i++) {
        HashTable[i].available = 1;
    }
}
```

Screenshots:





(2) Separate Chaining:

Main Functions Used:

- **getHT(image img):**

```
int getHT(image img) {
    int index = hashCode(img);
    data_item * currentStruct = HashTable[index];
    if (!HashTable[index]) return -1;
    while (currentStruct) {
        int result = compareIMG(img, currentStruct - > data);
        if (result == 1) return currentStruct - > key;
        else currentStruct = currentStruct - > next;
    }
          return -1;
     }
```

- **removeHT(image img):**

```
int removeHT(image img) {
    int index = hashCode(img);
    data_item * currentStruct = HashTable[index];
    data_item * previousStruct = HashTable[index];
    if (!HashTable[index]) return -1;
    else
        while (1) {
            if (compareIMG(currentStruct - > data, img)) {
                int tempKey = currentStruct - > key;
                      previousStruct - > next = currentStruct - > next;
                      currentStruct - > key = -1;
                      free(currentStruct);
                      return tempKey;
                  } else {
                      previousStruct = currentStruct;
                      currentStruct = currentStruct - > next;
                  }
              }
     }
```

- **putHT(int key , image current_image):**

```
1. void putHT(int key, image current_image) {
2.     int index = hashCode(current_image);
3.     if (!HashTable[index]) {
4.         data_item * newStruct = malloc(sizeof(data_item));
5.         newStruct - > data = current_image;
6.         newStruct - > key = key;
7.         newStruct - > next = NULL;
8.         HashTable[index] = newStruct;
9.     } else {
10.             data_item * current = HashTable[index];
11.             while (current - > next) current = current - > next;
12.             data_item * newStruct = malloc(sizeof(data_item));
13.             newStruct - > data = current_image;
14.             newStruct - > key = key;
15.             newStruct - > next = NULL;
16.             current - > next = newStruct; //PROBLEM MAY BE HERE
17.         }
18.     }
```

- **hashCode(image img):**

```
13.     int hashCode(image img) {
14.         long int i, j, rowSum = 0, totalSum = 0;
15.         for (i = 0; i < ROW; i++) {
16.             for (j = 0; j < COLUMN; j++) {
17.                 rowSum += img.img_arr[i * ROW + j];
18.             }
19.             totalSum += rowSum * (i + 1);
20.             rowSum = 0;
21.         } //MOD SIZE TO MAKE SURE THE INDEX CALCUATED DOESN'T OVERFLOW THE ARRAY SIZE
22.         totalSum = totalSum % 1000;
23.         return (int) totalSum;
24.     }
```

Extra Functions Used:

- **generateDatabase():**

```
16.      void generateDatabse() {
17.          FILE * f = fopen("MNIST-
   data(datastructures_assignment3)_whitespace.txt", "r");
18.          intializeHT();
19.          image tempImg;
20.          int tempKey;
21.          int i, j;
22.          for (i = 0; i < SIZE; i++) {
23.              for (j = 0; j < IMAGE_SIZE; j++) { //SCAN THE PIXELS
24.                  fscanf(f, "%d", & tempImg.img_arr[j]);
25.              } //SCAN THE ID
26.              fscanf(f, "%d", & tempKey); //PUT THE IMAGE AND ITS ID IN THE DATABASE
27.              putHT(tempKey, tempImg);
28.          }
29.          fclose(f);
30.      }
```

- **readTestFile(image * testImg):**

```
9. void readTestFile(image * testImg) {
10.          FILE * imgFile = fopen("test.txt", "r");
11.          int i = 0;
12.          for (i = 0; i < 784; i++) {
13.              fscanf(imgFile, "%d", & testImg - > img_arr[i]);
14.          }
15.          fclose(imgFile);
16.      }
```

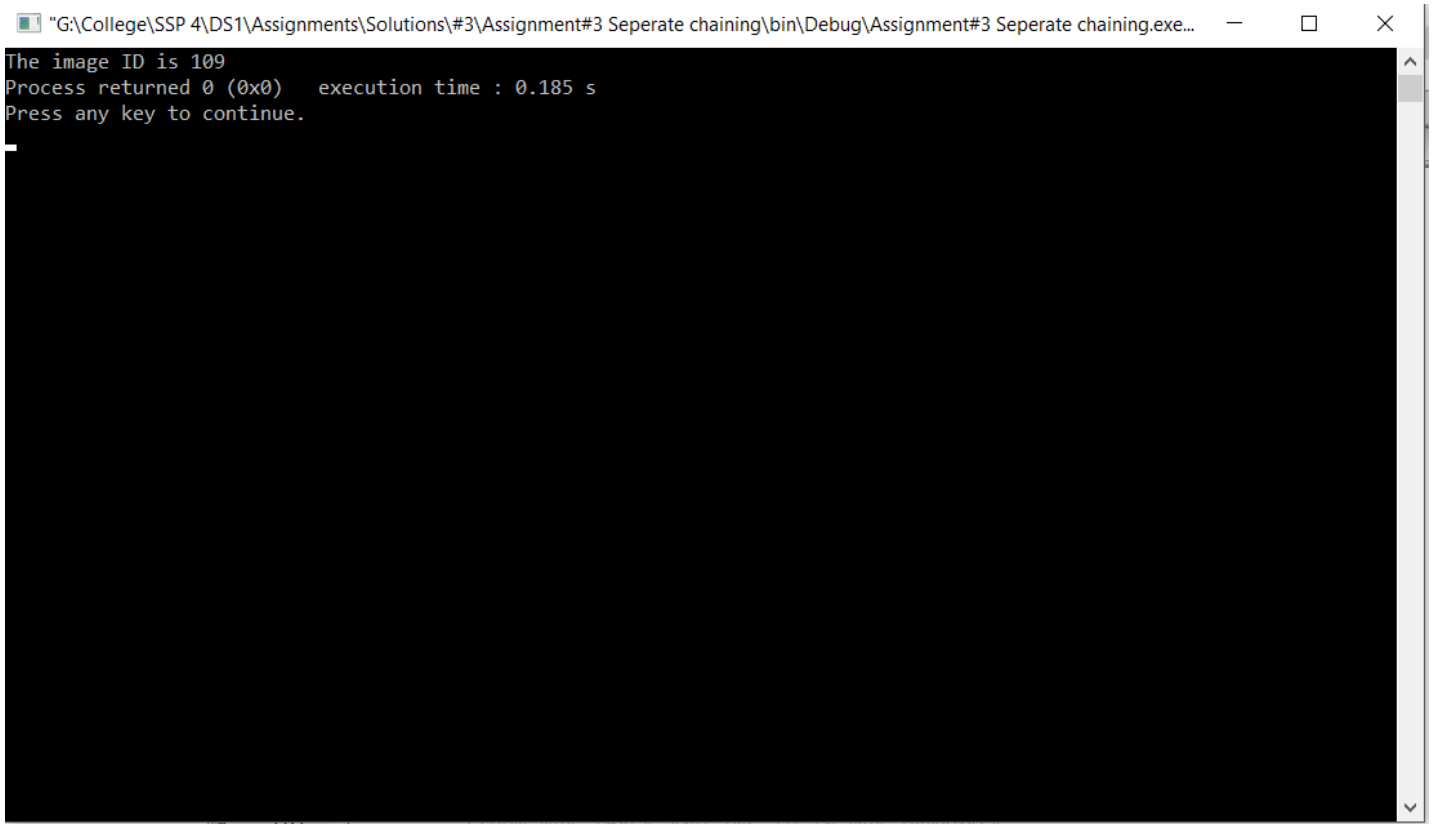
- **initializeHT():**

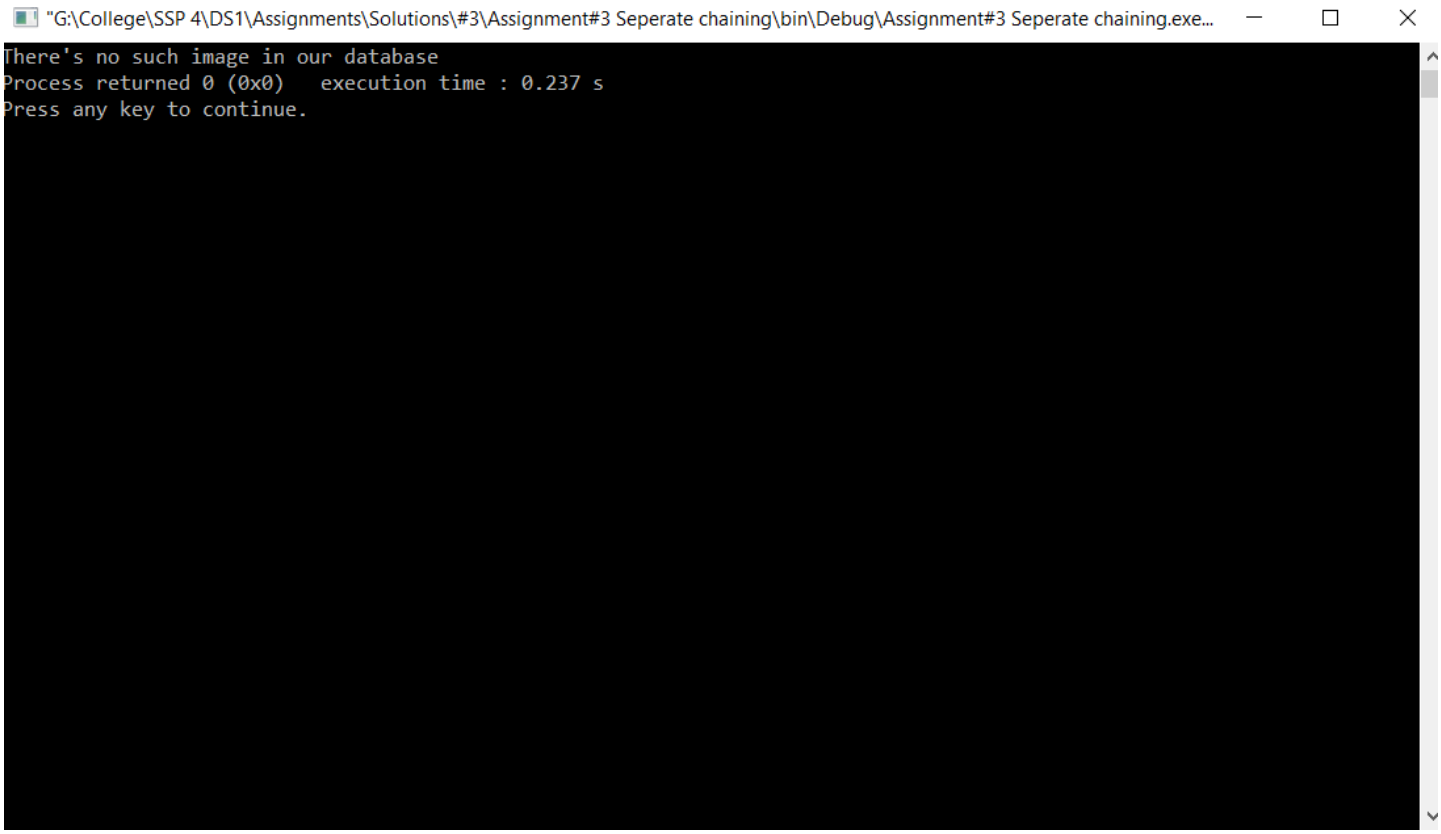
```
1. void initializeHT() {
2.     int i;
3.     for (i = 0; i < SIZE; i++) {
4.         HashTable[i] = NULL;
5.     }
6. }
```

- **compareIMG(image img1 , image img2):**

```
1. int compareIMG(image img1, image img2) {
2.     int i;
3.     for (i = 0; i < IMAGE_SIZE; i++) {
4.         if (img1.img_arr[i] != img2.img_arr[i]) return 0;
5.     }
6.     return 1;
7. }
```

Screenshots:




"G:\College\SSP 4\DS1\Assignments\Solutions\#3\Assignment#3 Seperate chaining\bin\Debug\Assignment#3 Seperate chaining.exe...
There's no such image in our database
Process returned 0 (0x0)   execution time : 0.237 s
Press any key to continue.

(3) Comparison:

- **Time needed for insertion of 1000 images using linear probing is: 0.800 ± 0.10 s**
- **Time needed for insertion of 1000 images using separate chaining is: 0.200 ± 0.50 s**