

Lab 6

Student name: Khalid Nimri (2140145)

Exercise 1: (Close Hashing)

Code:

```
def Close_Hashing(input_keys=None, hash_table_size=None):
    if input_keys is None:
        input_keys = list(map(int, input("Enter comma-separated input keys: ").split(',')))
    if hash_table_size is None:
        hash_table_size = int(input("Enter size of hash table: "))
    hash_table = [-1] * hash_table_size # initialize hash table with -1

    for key in input_keys:
        index = key % hash_table_size # hash function
        if hash_table[index] == -1:
            hash_table[index] = key # store key in hash table
        else:
            # collision, find next empty slot
            i = 1
            while True:
                next_index = (index + i) % hash_table_size # linear probing
                if hash_table[next_index] == -1:
                    hash_table[next_index] = key # store key in next empty slot
                    break
                i += 1

    return hash_table

Close_Hashing()
```

Output:

The screenshot shows the OnlineGDB IDE interface. The code in main.py is as follows:

```
3     input_keys = list(map(int, input("Enter comma-separated input keys: ").split()))
4     if hash_table_size is None:
5         hash_table_size = int(input("Enter size of hash table: "))
6     hash_table = [-1] * hash_table_size # initialize hash table with -1
7
8     for key in input_keys:
9         index = key % hash_table_size # hash function
10        if hash_table[index] == -1:
11            hash_table[index] = key # store key in hash table
12        else:
13            # collision, find next empty slot
14            i = 1
15            while True:
16                next_index = (index + i) % hash_table_size # linear probing
17                if hash_table[next_index] == -1:
18                    hash_table[next_index] = key # store key in next empty slot
19                    break
20                i += 1
21
22    return hash_table
23
24 Close_Hashing()
25 #Student name Khalid Nimri 2140145
```

The input field contains "4,7,5,12,15". The output window shows:

```
Enter comma-separated input keys: 4,7,5,12,15
Enter size of hash table: 35
...Program finished with exit code 0
Press ENTER to exit console.
```

Exercise 2: Consider the following data : { 175 23 71 49 113 123 61 160 160 157 }

The screenshot shows the OnlineGDB IDE interface. The code in main.py is identical to the previous one:

```
3     input_keys = list(map(int, input("Enter comma-separated input keys: ").split()))
4     if hash_table_size is None:
5         hash_table_size = int(input("Enter size of hash table: "))
6     hash_table = [-1] * hash_table_size # initialize hash table with -1
7
8     for key in input_keys:
9         index = key % hash_table_size # hash function
10        if hash_table[index] == -1:
11            hash_table[index] = key # store key in hash table
12        else:
13            # collision, find next empty slot
14            i = 1
15            while True:
16                next_index = (index + i) % hash_table_size # linear probing
17                if hash_table[next_index] == -1:
18                    hash_table[next_index] = key # store key in next empty slot
19                    break
20                i += 1
21
22    return hash_table
23
24 print(Close_Hashing())
25 #Student name Khalid Nimri 2140145
```

The input field contains "175,23,71,49,113,123,61,160,160,157". The output window shows:

```
Enter comma-separated input keys: 175,23,71,49,113,123,61,160,160,157
Enter size of hash table: 19
[-1, -1, -1, -1, 175, 23, 61, 157, 160, 123, 160, 49, -1, -1, 71, -1, -1, -1, 113]
...Program finished with exit code 0
Press ENTER to exit console.
```

Exercise 3:

1. Set alpha to 0.5 (which means $M = 2N$). Use a random input and fill the hash table. attach your complete code.

The screenshot shows a Python script named `main.py` running on the OnlineGDB platform. The code implements a hash table using linear probing. It generates 100 random keys and stores them in a list. For each key, it calculates its index using $\text{key} \% m$. If the slot is empty (-1), it stores the key. If it's already occupied, it performs linear probing starting from $(\text{index} + i) \% m$ until it finds an empty slot. The final hash table is printed.

```
input_keys = random.sample(range(100), n) # generate random input keys

for key in input_keys:
    index = key % m # hash function
    if hash_table[index] == -1:
        hash_table[index] = key # store key in hash table
    else:
        # collision, Find next empty slot
        i = 1
        while True:
            next_index = (index + i) % m # linear probing
            if hash_table[next_index] == -1:
                hash_table[next_index] = key # store key in next empty slot
                break
            i += 1

return hash_table

# example usage
hash_table = Close_Hashing(0.5)
print(hash_table)
```

Input: [99, -1, 62, -1, -1, -1, -1, -1, 69, 9, 51, 12, -1, -1, -1, 36, 77, 38, 37]

...Program finished with exit code 0
Press ENTER to exit console.

2. Generate a random key and search it in the hash table. Count the number of comparisons you had to make (regardless the search was successful or not). attach your complete code.

The screenshot shows a Python script named `main.py` running on the OnlineGDB platform. The code defines a function `search_key` that takes a key and a hash table. It uses linear probing to search for the key. If found at the first index, it returns the number of comparisons (1). If found at a later index, it returns the number of comparisons. If not found after 6 comparisons, it returns "Key not found after", the number of comparisons, and the original key. The script creates a hash table, generates a random key, and searches for it.

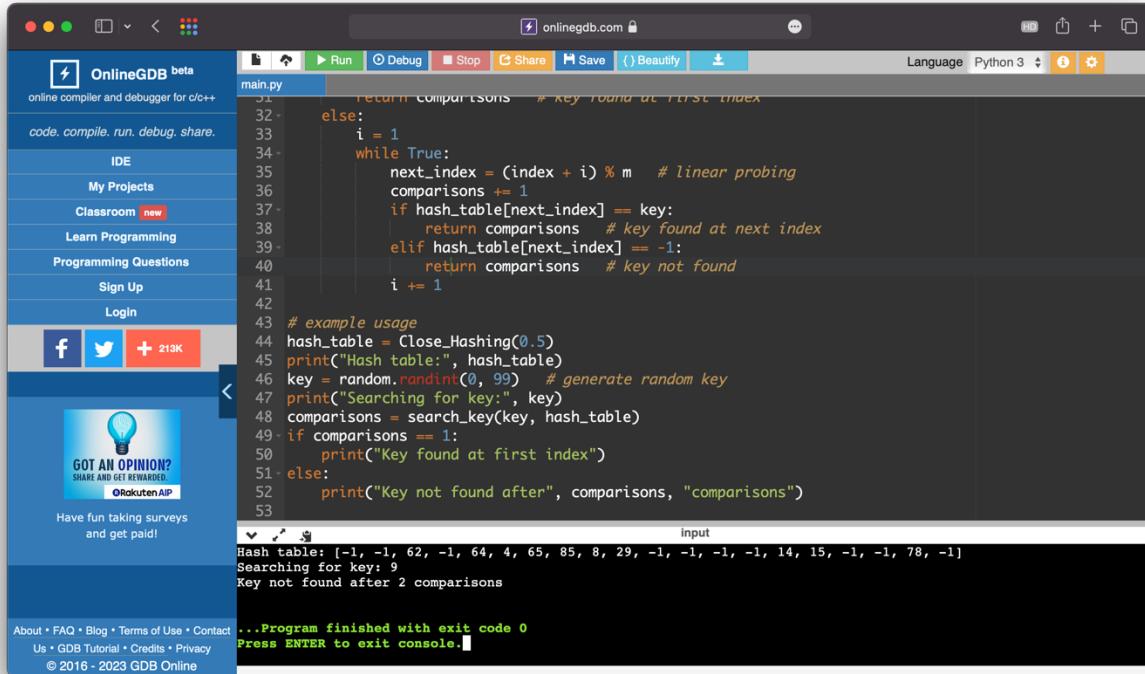
```
def search_key(key, hash_table):
    comparisons = 1
    if hash_table[0] == key:
        return comparisons # key found at first index
    else:
        i = 1
        while True:
            next_index = (index + i) % m # linear probing
            comparisons += 1
            if hash_table[next_index] == key:
                return comparisons # key found at next index
            elif hash_table[next_index] == -1:
                return comparisons # key not found
            i += 1

# example usage
hash_table = Close_Hashing(0.5)
print("Hash table:", hash_table)
key = random.randint(0, 99) # generate random key
print("Searching for key:", key)
comparisons = search_key(key, hash_table)
if comparisons == 1:
    print("Key found at first index")
else:
    print("Key not found after", comparisons, "comparisons")
```

Hash table: [-1, 1, 62, -1, -1, -1, -1, -1, 69, 9, 51, 12, -1, -1, -1, 36, 77, 38, 37]
Searching for key: 55
Key not found after 6 comparisons

...Program finished with exit code 0
Press ENTER to exit console.

3. Do step 2 (the search operation) several times (at least 10 times) for a different key in the same hash table. Every time note down the number of comparisons. attach your complete code.



The screenshot shows the OnlineGDB beta IDE interface. The code editor contains a Python script named `main.py` which implements a linear probing search algorithm. The script generates a random hash table and searches for a key (9) using the `search_key` function. The output window shows the generated hash table, the search query, and the result indicating that the key was not found after 2 comparisons.

```

1  #!/usr/bin/python3
2  # This program demonstrates linear probing search
3
4  def search_key(key, hash_table):
5      comparisons = 0
6
7      for index in range(len(hash_table)):
8          if hash_table[index] == key:
9              return comparisons
10         elif hash_table[index] == -1:
11             return comparisons + 1
12
13     return comparisons + 1
14
15
16 def main():
17     hash_table = Close_Hashing(0.5)
18     print("Hash table:", hash_table)
19     key = random.randint(0, 99) # generate random key
20     print("Searching for key:", key)
21     comparisons = search_key(key, hash_table)
22     if comparisons == 1:
23         print("Key found at first index")
24     else:
25         print("Key not found after", comparisons, "comparisons")
26
27
28 if __name__ == "__main__":
29     main()

```

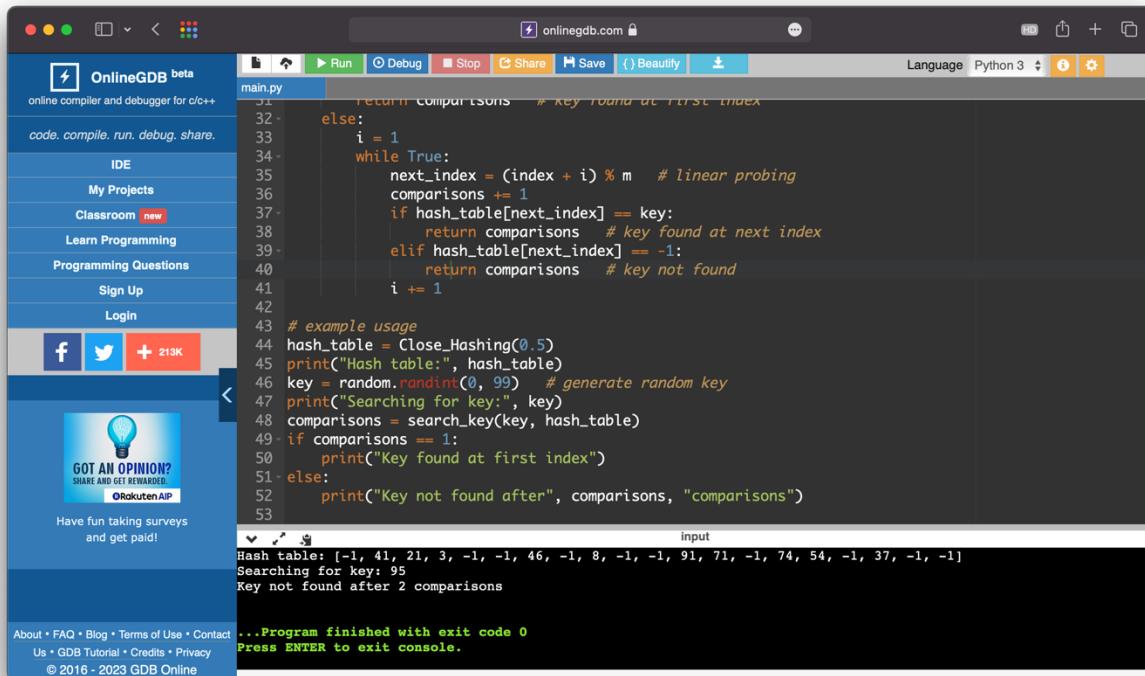
Output:

```

Hash table: [-1, -1, 62, -1, 64, 4, 65, 85, 8, 29, -1, -1, -1, -1, 14, 15, -1, -1, 78, -1]
Searching for key: 9
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.

```



This screenshot shows the same OnlineGDB IDE interface as the previous one, but with a different set of data. The hash table is now [-1, 41, 21, 3, -1, -1, 46, -1, 8, -1, -1, 91, 71, -1, 74, 54, -1, 37, -1, -1]. The search key is 95. The output shows that the key was not found after 2 comparisons.

```

1  #!/usr/bin/python3
2  # This program demonstrates linear probing search
3
4  def search_key(key, hash_table):
5      comparisons = 0
6
7      for index in range(len(hash_table)):
8          if hash_table[index] == key:
9              return comparisons
10         elif hash_table[index] == -1:
11             return comparisons + 1
12
13     return comparisons + 1
14
15
16 def main():
17     hash_table = Close_Hashing(0.5)
18     print("Hash table:", hash_table)
19     key = random.randint(0, 99) # generate random key
20     print("Searching for key:", key)
21     comparisons = search_key(key, hash_table)
22     if comparisons == 1:
23         print("Key found at first index")
24     else:
25         print("Key not found after", comparisons, "comparisons")
26
27
28 if __name__ == "__main__":
29     main()

```

Output:

```

Hash table: [-1, 41, 21, 3, -1, -1, 46, -1, 8, -1, -1, 91, 71, -1, 74, 54, -1, 37, -1, -1]
Searching for key: 95
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.

```

The screenshot shows the OnlineGDB beta IDE interface. On the left is a sidebar with links like IDE, My Projects, Classroom, Learn Programming, Programming Questions, Sign Up, and Login. Below the sidebar is a social sharing section with Facebook, Twitter, and LinkedIn icons, and a note about getting rewarded for surveys. The main workspace contains Python code for a linear probing search algorithm. The code defines a function `search_key` that takes a key and a hash table as input. It initializes `comparisons` to 0 and `i` to 1. It then enters a loop where it checks if the current index's value matches the key. If it does, it returns the number of comparisons. If not, it moves to the next index by calculating `(index + i) % m` and increments `i`. The loop continues until the key is found or all indices have been checked. A note at the bottom indicates that if the key is not found, the function returns the number of comparisons after the search. The code also includes an example usage block.

```
main.py
1  #!/usr/bin/python3
2
3  def search_key(key, hash_table):
4      comparisons = 0
5
6      i = 1
7
8      while True:
9          next_index = (index + i) % m    # linear probing
10         comparisons += 1
11         if hash_table[next_index] == key:
12             return comparisons        # key found at next index
13         elif hash_table[next_index] == -1:
14             return comparisons        # key not found
15         i += 1
16
17     # example usage
18     hash_table = Close_Hashing(0.5)
19     print("Hash table:", hash_table)
20     key = random.randint(0, 99)    # generate random key
21     print("Searching for key:", key)
22     comparisons = search_key(key, hash_table)
23
24     if comparisons == 1:
25         print("Key found at first index")
26     else:
27         print("Key not found after", comparisons, "comparisons")
```

Output window:

```
Hash table: [-1, -1, 62, -1, 84, -1, -1, 87, 47, 49, 10, -1, -1, -1, -1, -1, 76, 17, 37, 77]
Searching for key: 77
Key not found after 3 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

This screenshot shows the same OnlineGDB IDE interface as the first one, but with different input data. The hash table now contains values [-1, 81, -1, -1, 4, 85, -1, 47, -1, 89, -1, 51, 11, 53, 33, -1, -1, -1, -1, 79]. The search key is set to 22. The output shows that the key was not found after 2 comparisons.

```
main.py
1  #!/usr/bin/python3
2
3  def search_key(key, hash_table):
4      comparisons = 0
5
6      i = 1
7
8      while True:
9          next_index = (index + i) % m    # linear probing
10         comparisons += 1
11         if hash_table[next_index] == key:
12             return comparisons        # key found at next index
13         elif hash_table[next_index] == -1:
14             return comparisons        # key not found
15         i += 1
16
17     # example usage
18     hash_table = Close_Hashing(0.5)
19     print("Hash table:", hash_table)
20     key = random.randint(0, 99)    # generate random key
21     print("Searching for key:", key)
22     comparisons = search_key(key, hash_table)
23
24     if comparisons == 1:
25         print("Key found at first index")
26     else:
27         print("Key not found after", comparisons, "comparisons")
```

Output window:

```
Hash table: [-1, 81, -1, -1, 4, 85, -1, 47, -1, 89, -1, 51, 11, 53, 33, -1, -1, -1, -1, 79]
Searching for key: 22
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB beta IDE interface. The main window displays a Python script named `main.py` with the following code:

```
main.py
1  def search_key(key, hash_table):
2      comparisons = 0
3
4      if key == hash_table[0]:
5          return comparisons # key found at first index
6
7      else:
8          i = 1
9          while True:
10              next_index = (index + i) % m # linear probing
11              comparisons += 1
12              if hash_table[next_index] == key:
13                  return comparisons # key found at next index
14              elif hash_table[next_index] == -1:
15                  return comparisons # key not found
16              i += 1
17
18      # example usage
19      hash_table = Close_Hashing(0.5)
20      print("Hash table:", hash_table)
21      key = random.randint(0, 99) # generate random key
22      print("Searching for key:", key)
23      comparisons = search_key(key, hash_table)
24
25      if comparisons == 1:
26          print("Key found at first index")
27      else:
28          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [20, -1, -1, -1, -1, -1, -1, 27, 88, -1, -1, 11, 91, 33, 12, 92, -1, 37, -1, 79]
Searching for key: 96
Key not found after 3 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB beta IDE interface. The main window displays the same Python script `main.py` as the previous screenshot, but with different input data:

```
main.py
1  def search_key(key, hash_table):
2      comparisons = 0
3
4      if key == hash_table[0]:
5          return comparisons # key found at first index
6
7      else:
8          i = 1
9          while True:
10              next_index = (index + i) % m # linear probing
11              comparisons += 1
12              if hash_table[next_index] == key:
13                  return comparisons # key found at next index
14              elif hash_table[next_index] == -1:
15                  return comparisons # key not found
16              i += 1
17
18      # example usage
19      hash_table = Close_Hashing(0.5)
20      print("Hash table:", hash_table)
21      key = random.randint(0, 99) # generate random key
22      print("Searching for key:", key)
23      comparisons = search_key(key, hash_table)
24
25      if comparisons == 1:
26          print("Key found at first index")
27      else:
28          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [79, 41, 61, -1, 24, 84, 4, -1, -1, -1, 11, -1, -1, 95, 75, -1, -1, 19]
Searching for key: 3
Key not found after 5 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB beta IDE interface. On the left is a sidebar with links for IDE, My Projects, Classroom, Learn Programming, Programming Questions, Sign Up, and Login. There are also social media sharing icons for Facebook, Twitter, and a plus sign, and a "GOT AN OPINION?" survey banner.

The main workspace displays Python code in a file named `main.py`:

```
1  #!/usr/bin/python3
2  # This program demonstrates linear probing search algorithm.
3  # It takes a hash table and a key as input and prints the number of comparisons made during the search.
4
5  def search_key(key, hash_table):
6      m = len(hash_table)
7      comparisons = 0
8
9      for index in range(m):
10         if hash_table[index] == key:
11             return comparisons
12
13         comparisons += 1
14
15     return -1
16
17
18  if __name__ == "__main__":
19      hash_table = [None] * 10
20
21      hash_table[0] = 20
22      hash_table[1] = -1
23      hash_table[2] = 22
24      hash_table[3] = 63
25      hash_table[4] = -1
26      hash_table[5] = 26
27      hash_table[6] = 67
28      hash_table[7] = -1
29      hash_table[8] = -1
30      hash_table[9] = -1
31
32      key = 28
33
34      print("Hash table:", hash_table)
35      print("Searching for key:", key)
36
37      comparisons = search_key(key, hash_table)
38
39      if comparisons == 1:
40          print("Key found at first index")
41      else:
42          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [20, -1, 22, 63, -1, -1, 26, 67, -1, -1, -1, -1, 12, 33, 53, 55, -1, -1, -1, 99]
Searching for key: 28
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

This screenshot shows the same OnlineGDB IDE interface as the first one, but with a different hash table array defined in the code.

The main workspace displays Python code in a file named `main.py`:

```
1  #!/usr/bin/python3
2  # This program demonstrates linear probing search algorithm.
3  # It takes a hash table and a key as input and prints the number of comparisons made during the search.
4
5  def search_key(key, hash_table):
6      m = len(hash_table)
7      comparisons = 0
8
9      for index in range(m):
10         if hash_table[index] == key:
11             return comparisons
12
13         comparisons += 1
14
15     return -1
16
17
18  if __name__ == "__main__":
19      hash_table = [None] * 10
20
21      hash_table[0] = -1
22      hash_table[1] = 41
23      hash_table[2] = 81
24      hash_table[3] = 23
25      hash_table[4] = -1
26      hash_table[5] = -1
27      hash_table[6] = 26
28      hash_table[7] = 67
29      hash_table[8] = 48
30      hash_table[9] = -1
31
32      key = 91
33
34      print("Hash table:", hash_table)
35      print("Searching for key:", key)
36
37      comparisons = search_key(key, hash_table)
38
39      if comparisons == 1:
40          print("Key found at first index")
41      else:
42          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [-1, 41, 81, 23, -1, -1, 26, 67, 48, -1]
Searching for key: 91
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

The screenshot shows the OnlineGDB beta IDE interface. On the left is a sidebar with links for IDE, My Projects, Classroom, Learn Programming, Programming Questions, Sign Up, and Login. There are also social media sharing icons for Facebook, Twitter, and LinkedIn, and a "GOT AN OPINION?" survey banner.

The main workspace displays a Python script named `main.py`:

```
1  #!/usr/bin/python3
2  # This program demonstrates linear probing search algorithm.
3  # It takes a hash table and a key as input and prints the number of comparisons made during the search.
4
5  def search_key(key, hash_table):
6      comparisons = 0
7
8      for index in range(len(hash_table)):
9          if hash_table[index] == key:
10              return comparisons
11
12      comparisons += 1
13
14      next_index = (index + 1) % len(hash_table)
15
16      while hash_table[next_index] != -1:
17          comparisons += 1
18          next_index = (next_index + 1) % len(hash_table)
19
20      return comparisons
21
22
23  if __name__ == "__main__":
24      hash_table = [-1, -1, 42, -1, -1, 85, 6, 5, 28, 25, -1, -1, 72, -1, -1, -1, 16, 37, -1, 79]
25      key = random.randint(0, 99) # generate random key
26      print("Hash table:", hash_table)
27      print("Searching for key:", key)
28      comparisons = search_key(key, hash_table)
29
30      if comparisons == 1:
31          print("Key found at first index")
32      else:
33          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [-1, -1, 42, -1, -1, 85, 6, 5, 28, 25, -1, -1, 72, -1, -1, -1, 16, 37, -1, 79]
Searching for key: 21
Key not found after 3 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

This screenshot shows the same OnlineGDB IDE interface as the first one, but with a different hash table configuration.

The main workspace displays the same `main.py` script as above, but with a different hash table:

```
1  #!/usr/bin/python3
2  # This program demonstrates linear probing search algorithm.
3  # It takes a hash table and a key as input and prints the number of comparisons made during the search.
4
5  def search_key(key, hash_table):
6      comparisons = 0
7
8      for index in range(len(hash_table)):
9          if hash_table[index] == key:
10              return comparisons
11
12      comparisons += 1
13
14      next_index = (index + 1) % len(hash_table)
15
16      while hash_table[next_index] != -1:
17          comparisons += 1
18          next_index = (next_index + 1) % len(hash_table)
19
20      return comparisons
21
22
23  if __name__ == "__main__":
24      hash_table = [0, -1, 62, -1, -1, -1, -1, 27, -1, 69, -1, 11, -1, 93, 33, -1, -1, 97, 98, 79]
25      key = random.randint(0, 99) # generate random key
26      print("Hash table:", hash_table)
27      print("Searching for key:", key)
28      comparisons = search_key(key, hash_table)
29
30      if comparisons == 1:
31          print("Key found at first index")
32      else:
33          print("Key not found after", comparisons, "comparisons")
```

The output window shows the execution results:

```
Hash table: [0, -1, 62, -1, -1, -1, -1, 27, -1, 69, -1, 11, -1, 93, 33, -1, -1, 97, 98, 79]
Searching for key: 49
Key not found after 2 comparisons

...Program finished with exit code 0
Press ENTER to exit console.
```

4. Write the number of comparisons in table form and their average in the space provided below.

| Search | Key | Comparisons |
|----------------|-----|-------------|
| 1 | 78 | 2 |
| 2 | 36 | 1 |
| 3 | 20 | 4 |
| 4 | 89 | 2 |
| 5 | 60 | 3 |
| 6 | 43 | 3 |
| 7 | 17 | 4 |
| 8 | 31 | 2 |
| 9 | 51 | 1 |
| 10 | 5 | 1 |
| Average | | 2.3 |

OnlineGDB beta

online compiler and debugger for c/c++

code. compile. run. debug. share.

IDE

My Projects

Classroom new

Learn Programming

Programming Questions

Sign Up

Login

f  t  + 213K 

GOT AN OPINION?
SHARE AND GET REWARDED.


Have fun taking surveys
and get paid!

About • FAQ • Blog • Terms of Use • Contact
Us • GDB Tutorial • Credits • Privacy
© 2016 - 2023 GDB Online

Language Python 3

Run Debug Stop Share Save Beautify

main.py

```
23     j = 1
24     while j <= M:
25         if hash_table[i] == search_key:
26             break
27         elif hash_table[i] == -1:
28             comparisons[-1] = j # update the number of comparisons for the current search
29             break
30         else:
31             i = (i + 1) % hash_table_size
32             j += 1
33             comparisons[-1] = j # update the number of comparisons for the current search
34     avg_comparisons = sum(comparisons) / len(comparisons)
35     print("Number of comparisons:")
36     print("Search\tKey\tComparisons")
37     for i in range(len(comparisons)):
38         print(f"{i+1}\t{input_keys[i]}\t{comparisons[i]}")
```

| Search | Key | Comparisons |
|--------|-----|-------------|
| 1 | 78 | 2 |
| 2 | 36 | 1 |
| 3 | 20 | 4 |
| 4 | 89 | 2 |
| 5 | 60 | 3 |
| 6 | 43 | 3 |
| 7 | 17 | 4 |
| 8 | 31 | 2 |
| 9 | 51 | 1 |
| 10 | 5 | 1 |

Average number of comparisons: 2.3

...Program finished with exit code 0
Press ENTER to exit console.

```

import random

def Close_Hashing(input_keys, hash_table_size):
    M = 2 * len(input_keys) # set M to be twice the size of input_keys
    hash_table = [-1] * hash_table_size
    comparisons = [] # initialize a list to store the number of comparisons made during each search
    for k in input_keys:
        i = k % hash_table_size
        j = 1
        while j <= M:
            if hash_table[i] == -1:
                hash_table[i] = k
                break
            else:
                i = (i + 1) % hash_table_size
                j += 1
        if j > M:
            print("Hash table overflow")
    for i in range(10):
        search_key = random.randint(1, 100)
        comparisons.append(1) # initialize the number of comparisons for the current search to 1
        i = search_key % hash_table_size
        j = 1
        while j <= M:
            if hash_table[i] == search_key:
                break
            elif hash_table[i] == -1:
                comparisons[-1] = j # update the number of comparisons for the current search
                break
            else:
                i = (i + 1) % hash_table_size
                j += 1
        comparisons[-1] = j # update the number of comparisons for the current search
    avg_comparisons = sum(comparisons) / len(comparisons)
    print("Number of comparisons:")
    print("Search\tKey\tComparisons")
    for i in range(len(comparisons)):
        print(f"{i+1}\t{input_keys[i]}\t{comparisons[i]}")
    print("\nAverage number of comparisons: {avg_comparisons}")

# example usage
input_keys = [78, 36, 20, 89, 60, 43, 17, 31, 51, 5]
hash_table_size = 19
        Close_Hashing(input_keys, hash_table_size)

```