You can view this report online at : https://www.hackerrank.com/x/tests/1517174/candidates/50646065/report

Full Name:	Breeha Qasim
Email:	bq08283@st.habib.edu.pk
Test Name:	**CS 102 - Lab 8 - Spring 2023**
Taken On:	3 Mar 2023 10:34:39 PKT
Time Taken:	2168 min 45 sec/ 2880 min
Work Experience:	< 1 years
Invited by:	Muzammil
Skills Score:	Problem Solving (Intermediate) 30/40
Tags Score:	

87.5%

70/80

scored in **CS 102 - Lab 8 - Spring 2023** in 2168 min 45 sec on 3 Mar 2023 10:34:39 PKT

Recruiter/Team Comments:

No Comments.

	Question Description	Time Taken	Score	Status
Q1	Creating Hashes > Coding	46 min 29 sec	0/ 0	✓
Q2	Get and Put > Coding	7 hour 14 min 54 sec	40/ 40	✓
Q3	Counting Shopping Cart Items > Coding	46 min 8 sec	30/ 40	✓

QUESTION 1

✓ Correct Answer

Score 0

Creating Hashes > Coding

QUESTION DESCRIPTION

You will create different types of hashes in this question. Create the following functions:

- `hash_function(key,size)`
  - If the key is of type integer, use Divide Modulo function, i.e. ***key mod size*** to generate and return the slot index
  - If the key is of type string, then the slot index will be calculated by the sum of Unicode values of each letter in the string. Don't forget to get the modulus value so that it remains within the range of hash table size.

- `rehash(old_hash,size, iteration)`
  - Use Linear Probing to resolve collisions
  - Here, ***old_hash*** represents the original hash function value, ***size*** is the size of hash table, and ***iteration*** is the iteration number to get the next interval in case of collisions

Python comes with its built-in library for hashing, i.e. **hashlib**, that provides methods for many known encryption techniques, one of which is md5 hash. For details, refer to the given links. Define the following function using the built-in methods for md5 hash. This function is <u>not</u> going to be used in later questions.

- `md5hash(key)`
    - Use Python library to define this hash function.
    - You may use any string value to test your code, in Custom Input. Refer to the given links to learn more about how to use md5 hash methods to get encoded data in form of bytes or hexadecimal equivalent value.

This question has no test cases. After you create the functions get them verified by the course staff in the lab.

Refer to these web pages to read further on md5 hash:
What is md5?
How to use md5 in Python

INTERVIEWER GUIDELINES

```
def hash_function(key,size):
    if type(key) == int:
        return key % size
    sum = 0
    for i in range(len(key)):
        sum = sum + ord(key[i])
    return sum % size

def rehash(old_hash,size,iteration):
    return (old_hash + iteration) % size
import hashlib
def md5hash(key):
    hash_key = hashlib.md5(key.encode())
    return hash_key
```

CANDIDATE ANSWER

Language used: **Python 3**

```

def hash_function(key,size):
    #total_sum=0
    if type(key)==int:
        return key%size
    elif type(key)==str:
        total_sum=0
        for i in range(len(key)):
            ASCII=ord(str(i))
            total_sum+=ASCII
        return total_sum%size

def rehash(old_hash,size,iteration):
    return (old_hash+iteration)%size

#def md5hash(key):




```

No Comments

QUESTION 2

Get and Put › Coding

Correct Answer

Score 40

QUESTION DESCRIPTION

You will create a hashing implementation in this question using the hash functions created in the last question. Create the following two functions to implement the primary operations of hashing:

- `put(keys, values, key, data)`
  - The `put` function uses the `key` to create a hash and puts the `data` in the appropriate place.
  - It first calls the hash function (as defined in the last question) to calculate the slot index using `key`
  - If the given slot is empty, i.e. contains None keyword, then adds the `data` in that slot
  - If the slot index has the same `key`, then simply update the `data` in its corresponding slot
  - If the slot index is occupied by a different `key`, then there is collision, so handle it using the rehash function, as defined in the last question
  - Here, `keys` refer to the list containing keys of the hash table, and `values` refer to the list containing values/data of the corresponding keys
- `get(keys, values, key)`
  - The `get` function gets the data from the given `key` by looking up the hash table.
  - In case of collision, it should rehash until value is found, or the entire hash table has been checked
- `main`
  - This function takes all the inputs, as explained in the **Input Details** below.
  - Then define two separate lists to represent the hash table. One list to keep the keys of the hash table and another list to keep the values associated with those keys. Both of these lists will have the same indices for any key-value pair. For eg, if list_of_keys[0] gets you a key then list_of_values[0] will get you the value of that key. Initially create lists of `size` (taken as input) and populate them using the `None` keyword.
  - For each of the given inputs (Input details given below), call the put function to test your code. In the end, print the updated hash table, i.e. list of keys followed by list of values.
  - Finally test the get functions with the given inputs (Input details given below), and print the returned values

Input Details:
The first line of the input is `size` of the hash table.
The second line of the input is `n` which is the amount of entries to do in the hash table.
The next `n` inputs are in separate lines in the form `key data`
All remaining `n` inputs are the keys to validate if the functions work correctly

INTERVIEWER GUIDELINES

```
keys = [None] * size
values = [None] * size

def hash_function(key,size):
    if type(key) == int:
        return key % size
    sum = 0
    for i in range(len(key)):
        sum = sum + ord(key[i])
    return sum % size

def rehash(old_hash,size):
    return (old_hash + 1) % size

def put(keys,values,key, data,size):
    hash_value = hash_function(key, size)
    if keys[hash_value] == None:
        keys[hash_value] = key
        values[hash_value] = data
    else:
        if keys[hash_value] == key:
            values[hash_value] = data
        else:
            next_hash = rehash(hash_value,size)
            while keys[next_hash] != None and keys[next_hash] != key:
                next_hash = rehash(hash_value, size)
            if keys[next_hash] == None:
                keys[next_hash] = key
                values[next_hash] = data
            else:
                values[next_hash] = data

def get(keys,values,key,size):
    start = hash_function(key, size)
    data = None
    position = start
    while keys[position] != None:
        if keys[position] == key:
            data = values[position]
            return data
        else:
            position = rehash(start, size)
            if position == start:
                break
    return data
```

CANDIDATE ANSWER

Language used: **Python 3**

```
def hash_function(key,size):
    #total_sum=0
    if type(key)==int:
        return (key%size)
    if type(key)==str:
        total_sum=0
        for i in range(len(key)):
            ASCII=ord(str(i))
            total_sum+=ASCII
        #print(total_sum,key)
        return(total_sum%size)

def rehash(old_hash,size,iteration):
    return (old_hash+iteration)%size

def put(keys,values,key,data):
    call=hash_function(key,len(keys))
    if keys[call]==None:
        keys[call]=key
        values[call]=data
        return call
    elif keys[call]==key:
        values[call]=data
    else:
        i=1
        call2=rehash(call,len(keys),i)
        while keys[call2] != None:
            i+= 1
            call2=rehash(call,len(keys),i)
        keys[call2] = key
        values[call2]  = data

def get(keys,values,key):
    x=hash_function(key,len(keys))
    if keys[x]==key:
        return values[x]
    else:
        j=1
        y=rehash(x,len(keys),j)
        while keys[y]!=key:
            j+=1
            y=rehash(x,len(keys),j)
        keys[y] = key
        return(values[y])
def main():
    size= int(input())
    n=int(input())
    keys=[]
    values=[]
    for i in range(size):
        keys.append(None)
        values.append(None)
    for i in range(n):
        keydata=input().split()
        key=keydata[0]
        data=keydata[1]
        put(keys,values,key,data)
        print(get(keys,values,key))
        #print(keys, values)
```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
Testcase 0	Easy	Sample case	✓ Success	10	0.0381 sec	8.25 KB
Testcase 1	Easy	Sample case	✓ Success	10	0.0317 sec	8.15 KB
Testcase 2	Easy	Hidden case	✓ Success	10	0.0499 sec	8.05 KB
Testcase 3	Easy	Hidden case	✓ Success	10	0.0454 sec	7.99 KB

No Comments

QUESTION 3

✓ Correct Answer

Score 30

Counting Shopping Cart Items › Coding

QUESTION DESCRIPTION

We are building an auto counter for a shopping mart. Different types of objects go through a scanner and we need to build the code to count how much of each object has gone through the scanner. You will implement the given scenario using the hash functions defined in the last two questions. **You may try different hash functions of your own to get bonus marks.**

In addition to all the previous functions, define the following functions:

- `CountFrequency(keys, values, list_items, size)`
  - For each ***object*** in `list_items`, this function will access the hash table using `get` function to get the ***count*** value associated with it.
  - Each ***object*** and its ***count*** are added as a tuple to a list, in the form: `[(object1, count), (object2, count), (object3, count)]`
  - Finally, return the updated list of tuples.
- `SortingFunction(lst_of_tuples)`
  - This function sorts the items in `lst_of_tuples` in **descending order** of their ***count*** values and returns the sorted list. Remember each tuple is of the form `(object1, count)` as defined in the previous function.
  - For example:
    - If `lst_of_tuples` = [('x', 3), ('s', 1), ('t', 4), ('a', 4), ('y', 3)]
    - Then the `sortingFunction` function would return the sorted list as: [('t', 4), ('a', 4), ('x', 3), ('y', 3), ('s', 1)]
    - You can use any Sorting Algorithm that you implemented in the Last Lab.
- `main`

- Define two separate lists to represent the hash table. One list to keep the keys of the hash table and another list to keep the values associated with those keys. Both of these lists will have the same indices for any key-value pair. For eg, if list_of_keys[0] gets you a key then list_of_values[0] will get you the value of that key. Initially create lists of `size` (Assume **hash table of size 20** for this question) and populate them using the `None` keyword.
- This function then takes the first input that represents the number of items to be added to the hash table
- For the given number of items, take each ***object*** as string input, and use it as a key to insert it in the hash table using the `put` function, where key will represent the ***object*** of type string, and its ***count*** (of type int) will become its value in the hash table. If the same object appears in the list of objects, get the previous count of the object using the `get` function, and update its ***count*** in the hash table
- Then call the `CountFrequency` function to get the items as list of tuples, and print it as shown in Sample Output.
- Apply `SortingFunction` on the items list to get the list in descending order of their count.
- Finally, print each ***object*** and its ***count*** as shown in the Sample Output.

Sample Case

Input:

4 -> Number of items
cake -> First item
bread -> Second item
bread -> Third item
bread -> Fourth item

Output:

bread 3
cake 1

INTERVIEWER GUIDELINES

```
size = 20
keys = [None] * size
values = [None] * size
def hash_function(key,size):
    if type(key) == int:
        return key % size
    sum = 0
    for i in range(len(key)):
        sum = sum + ord(key[i])
    return sum % size

def rehash(old_hash,size):
    return (old_hash + 1) % size

def put(keys,values,key, data,size):
    hash_value = hash_function(key, size)
    if keys[hash_value] == None:
        keys[hash_value] = key
        values[hash_value] = data
    else:
        if keys[hash_value] == key:
            values[hash_value] = data
        else:
            next_hash = rehash(hash_value,size)
            while keys[next_hash] != None and keys[next_hash] != key:
                next_hash = rehash(hash_value, size)
            if keys[next_hash] == None:
                keys[next_hash] = key
                values[next_hash] = data
            else:
                values[next_hash] = data

def get(keys,values,key,size):
    start = hash_function(key, size)
    data = None
    position = start
    while keys[position] != None:
        if keys[position] == key:
            data = values[position]
            return data
        else:
            position = rehash(start, size)
            if position == start:
                break
    return data
def CountFrequency(items):
    for prod in items:
        if prod not in keys:
            put(keys,values,prod,0)
        put(keys,values,prod,get(keys,values,prod) + 1)
    ans = []
    for i in range(size):
        if keys[i] != None and values != None and (keys[i],values[i]) not
in ans:
            A = (keys[i],values[i])
            ans.append(A)
    return ans
```

CANDIDATE ANSWER

Language used: **Python 3**

```python
def hash_function(key):
    total_sum=0
    for i in range(len(key)):
        ASCII=ord(str(i))
        total_sum+=ASCII

        #print(total_sum,key)
    return(total_sum%len(key))

def rehash(keys,old_hash):
    return (old_hash+1)%len(keys)

def put(keys,values,key):
    call=hash_function(key)
    if key in keys:
        values[call]=values[call]+1
        #return call
    elif keys[call]==None:
        keys[call]=key
        values[call]=1
    else:
        while keys[call] != None:
            call=rehash(keys,call)
        keys[call] = key
        values[call]  = 1

def CountFrequency(keys,values,list_items,size):
    values=[]
    keys=[]
    lst=[]
    for i in range(20):
        values.append(None)
        keys.append(None)
    for i in list_items:
        put(keys,values,i)
    for i in range(20):
        if keys[i]!=None:
            lst.append((keys[i],values[i]))
    return lst
def SortingFunction(lst):
    for i in range(len(lst)):
        maximum=i
        for j in range(i+1,len(lst)):
            if lst[j][1]>lst[maximum][1]:
                maximum=j
        lst[i],lst[maximum]=lst[maximum],lst[i]
def main():
    list_items=[]
    keys=[]
    values=[]
    size=int(input())
    for i in range(size):
        list_items.append(input())
    ot=CountFrequency(keys,values,list_items,size)
    #print(ot)
    SortingFunction(ot)
    for i in ot:
        print(i[0], i[1])



```

TESTCASE	DIFFICULTY	TYPE	STATUS	SCORE	TIME TAKEN	MEMORY USED
TestCase 0	Easy	Sample case	✓ Success	10	0.0602 sec	9.36 KB
TestCase 1	Easy	Sample case	✗ Wrong Answer	0	0.0671 sec	9.27 KB
TestCase 2	Easy	Hidden case	✓ Success	10	0.0607 sec	9.39 KB
Testcase 3	Easy	Hidden case	✓ Success	10	0.101 sec	9.38 KB

No Comments