Birla Institute of Technology and Science, Pilani Hyd Campus
CS F211: Data Structures and Algorithms
2nd Semester 2024-25 Lab No: 11
Hash Maps

Program 1:

Given an unordered_map () class as shown below. The output of the code is also given for a better understanding. C++ uses hash tables to implement maps. Your task in this code is to complete the two tasks given in the driver code (line no: 21 and line no: 27) to get the desired output.

```
1  #include <iostream>
2  #include <unordered_map>
3  using namespace std;
4  void displayBucket(unordered_map<string, int> u_map, int i)
5  {
6      for (auto it = u_map.begin(i); it != u_map.end(i); ++it)
7          cout << "(" << it->first << ", " << it->second << ")";
8      cout << '\n';
9  }
10 int main()
11 {
12     unordered_map<string, int> mymap;
13     mymap["Rajesh"] = 10;
14     mymap["Akash"] = 20;
15     mymap["Shyamala"] = 30;
16     mymap["Radhika"] = 30;
17     mymap["Rohit"] = 30;
18     mymap["Sachin"] = 30;
19
20     // Count the number of buckets in the unordered_map
21     //Write your code here (Task 1)
22     cout << "Mymap has " << n << " buckets.\n\n";
23
24     // Prints elements present in each bucket
25     for (unsigned i = 0; i < n; i++)
26     {
27         //Write your code here (Task 2)
28     }
29
30     return 0;
31 }
```

```
Mymap has 13 buckets.

Bucket 0 contains:
Bucket 1 contains:
Bucket 2 contains: (Sachin, 30)
Bucket 3 contains:
Bucket 4 contains:
Bucket 5 contains: (Radhika, 30)(Akash, 20)
Bucket 6 contains: (Rajesh, 10)
Bucket 7 contains:
Bucket 8 contains:
Bucket 9 contains: (Shyamala, 30)
Bucket 10 contains:
Bucket 11 contains:
Bucket 12 contains: (Rohit, 30)


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

Program 2: Caching (filemanager.cpp attached)

The filemanager.cpp program given here allows users to create new files and read existing files. Each file is described by its name and content (or text). The program uses caching to prevent repetitive file reads. The cache is a fixed-length vector containing files that have already been read. A hashmap maps file name to the index of the file in the cache vector. When a requested file is present in the cache, the manager returns the content directly from the cache, but if it is not present, it frees a space in the cache and reads the requested file into it. The manager uses a least-recently used (LRU) strategy as discussed in the class for replacement—when in need of space, the LRU file is replaced by the new one. The content of the file is then output.

Task 1: Complete the get() method of the file manager.

Task 2: The hashmap used by the manager uses separate chaining to resolve the collisions. Modify the class to implement linear or quadratic probing.

Task 3: The freeSpace() method given in the program runs in O(n), which essentially defeats the purpose of having a hashmap. Optimise the function by using a priority queue or a heap to retrieve the LRU file in O(logn). You may use STLs to solve these tasks.

The output should look like:

```
Adding hello.txt ...
Free space found at 0
>> Hello World!

Adding bio.txt ...
Free space found at 1
>> My name is X

Adding lorem.txt ...
Free space found at 2
>> Lorem ipsum dolor sit amet, consectetur adipiscing elit

Fetching hello.txt ...
hello.txt found in cache at index 0
>> Hello World!

Adding sample.txt ...
No free space found, removing bio.txt to clear index 1
>> The quick brown fox jumps over the lazy dog

Fetching bio.txt ...
bio.txt not found in cache
No free space found, removing lorem.txt to clear index 2
>> My name is X
```

Program 3: DNS Resolution (resolver.cpp attached)

When you are using the Internet, the domain of the website that you wish to access (like www.google.com) is first converted into a corresponding IP address (like 8.8.8.8) through a system called DNS (Domain Name System). You might think that DNS makes use of a single server that contains all domain-IP mappings, but in reality, DNS consists of many servers that operate in a hierarchical fashion. Let us consider an imaginary world where DNS requests are resolved the following way (this is inspired by how DNS resolutions work in real life).

1. A request for the IP corresponding to “in.ts.hyderabad.gov” is sent to the root DNS server. This request is forwarded to the “.in” DNS server.
2. The “.in” DNS server forwards the request to the “.ts” DNS server.
3. The “.ts” DNS server forwards the request to the “.hyderabad” DNS server.
4. The “.hyderabad” DNS server forwards the request to the “.gov” authoritative server.
5. The “.gov” authoritative server contains the requested IP address, and this is returned.

Note that in this case, the DNS server which we refer to as “.hyderabad” is only used in resolving URLs that are prefixed with “in.ts.hyderabad”. So, a URL prefixed with “bits.hyderabad” would go through a different server. Therefore, the system of DNS servers should be understood as a tree and not a directed acyclic graph. Each DNS server keeps track of its immediate children servers using a hashmap. A partial implementation of the code is present in resolver.cpp.

Task 1: Complete the findRecursive() method of the Server class. You can refer to the implementation of the addRecursive() method for help.
Task 2: In the current implementation, a server can behave as both a DNS server and an authoritative server. Modify the code to prevent this behaviour.

Outputs are as shown below:

```
Finding bits.hyd.hpc ...

Looking for .bits child server
Found .bits child server, forwarding to it

Looking for .hyd child server
Not found, URL must be invalid

Adding bits.hyd.hpc ...

Looking for .bits child server
.bits child server found
Forwarding to .bits child server

Looking for .hyd child server
Not found, creating .hyd child server
Forwarding to .hyd child server

Looking for .hpc child server
Not found, creating .hpc child server
Forwarding to .hpc child server

Authoratative server reached, IP 111.154.47.72 set

Finding bits.hyd.hpc ...

Looking for .bits child server
Found .bits child server, forwarding to it

Looking for .hyd child server
Found .hyd child server, forwarding to it

Looking for .hpc child server
Found .hpc child server, forwarding to it

Authoratative server reached
>> 111.154.47.72
```

```
Adding bits.pilani.hpc ...

Looking for .bits child server
Not found, creating .bits child server
Forwarding to .bits child server

Looking for .pilani child server
Not found, creating .pilani child server
Forwarding to .pilani child server

Looking for .hpc child server
Not found, creating .hpc child server
Forwarding to .hpc child server

Authoratative server reached, IP 192.222.115.111 set

Adding in.ka.bangalore.gov ...

Looking for .in child server
Not found, creating .in child server
Forwarding to .in child server

Looking for .ka child server
Not found, creating .ka child server
Forwarding to .ka child server

Looking for .bangalore child server
Not found, creating .bangalore child server
Forwarding to .bangalore child server

Looking for .gov child server
Not found, creating .gov child server
Forwarding to .gov child server

Authoratative server reached, IP 94.3.93.46 set
```
