National University of Computer and Emerging Sciences

School of Computing Fall 2020 Islamabad Campus

CS218: Data Structures (Fall 2020)

Semester Project

(Deadline: 21st December 2020 09:00 AM)

Project groups: This project can be done within a group of three (3) students. There is no restriction on the selection of group members. Students can make groups according to their preferences. The group members must belong to the same section.

Submission: All submissions MUST be uploaded on Google classroom. Solutions sent to the emails will not be graded. To avoid last minute problems (unavailability of Internet, load shedding etc.), you are strongly advised to start working on the project from day one.

You are required to use Visual Studio 2019 for the project. Combine all your work (solution folder) in one .zip file after performing "Clean Solution". Submit zip file on slate within given deadline. If only .cpp file is submitted it will not be considered for evaluation.

Plagiarism: **Zero marks** in the project for all members if any significant part of project is found plagiarized. A code is considered plagiarized if **more than 20%** code is not your own work.

Distributed Hash Tables

We are considering a scenario where the data is not located on a single machine but rather stored on multiple machines geo-distributed across the Internet. In such a scenario, searching or retrieval of data is very challenging, as it is hard to determine on which machine the required data is stored. The data structures used for efficiently search data stored on a single machine such as hash tables cannot be directly employed in a distributed environment. The Figure 1 shows the concept of data storage and lookup in a distributed environment.

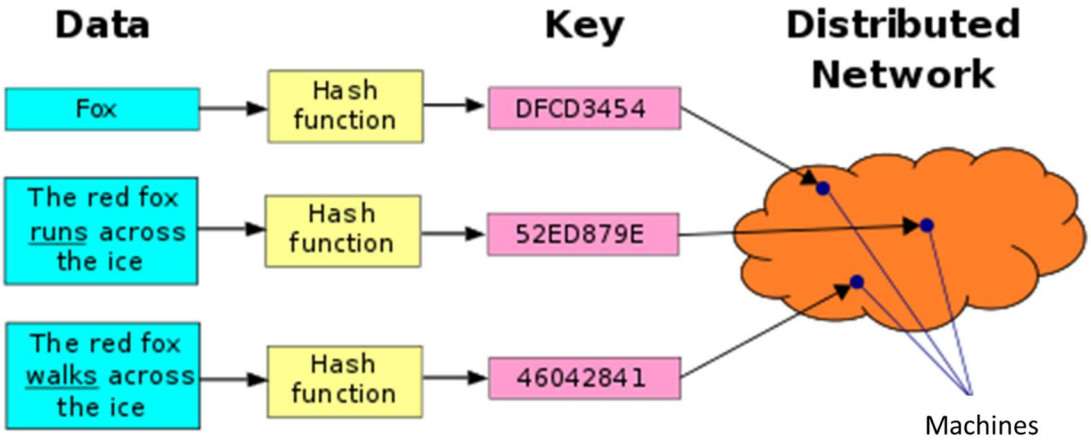

Figure 1

Typically, data is stored as a *key*, *value* pair. For example, *key* might be the name of the student or the student ID and *value* is the complete information about the student. Likewise, the *key* can be the name of the patient and *value* can be the medical record. Similarly, the *key* can be the name of an image file and *value* represents the file itself.

A distributed hash table (DHT) provides a lookup (or search) service similar to a hash table: (key, value) pairs are stored in a DHT and any participating machine can efficiently retrieve the value associated with a given key. Responsibility for maintaining the mapping from keys to values is distributed among the machines. In particular,

- Hash function such as SHA-1 is used to map key to identifier space. For example, in Figure 2 a key is mapped to an id e_1 in the identifier space. Another key is mapped to the id e_2 and so on.
- Machines are also mapped into identifier space using the hash function on their addresses such as IP address.
- Each machine is responsible for a range of ids in the identifier space. For example, Figure 2 depicts that machine 1 is responsible for storing the data (i.e., key, value pairs) whose keys are mapped to e_1 and e_2. Likewise, machine 3 is responsible for the ids e_4 and e_5.
- When a machine leaves, its id range is merged with a neighbor's range. For example, according to Figure 2 machine 4 will be responsible for the ids in the range e4 to e6, in case machine 3 leaves the network.
- Likewise, when a new machine joins, the id range of neighbor is subdivided.

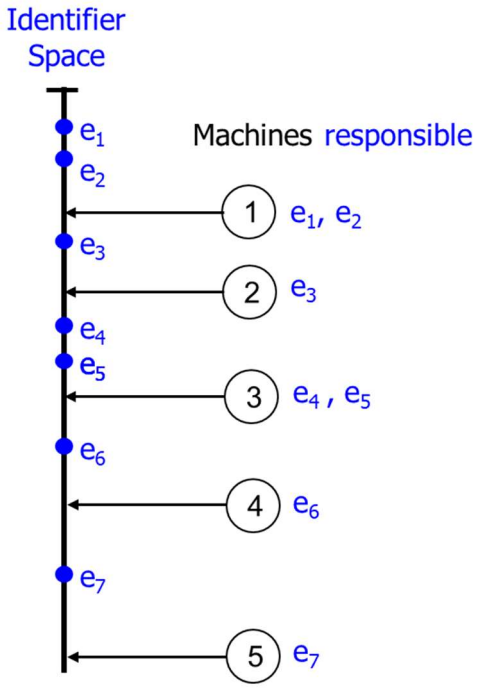

Figure 2

In this project, we will implement a special type of DHT that has a circular identifier space, named Ring DHTs. As most of the students are not familiar with the concepts of computer networks and distributed systems, the following description simplifies many aspects of the DHT and primarily focuses on emulating the DHT functionality on a single machine.

Properties of Ring DHT

- Ring DHT uses 160-bit circular identifier space using SHA-1. Figure 3 shows only a 4-bit identifier space to simplify the explanation.
- Randomly chosen identifiers (ids in short) are assigned to machines (also called nodes in the following): `H(Node Address) → Node Id or Machine ID`
  Figure 3 shows 5 machines with ids 1, 4, 7, 12 and 15.
- Randomly chosen identifiers are assigned to data, i.e. (key, value) pairs: `H(key) → Data Id`
- Data with identifier e is stored on the node/machine with the smallest identifier p ≥ e. This node/machine is called successor of e, denoted as `succ(e)`. For example, Figure 3 depicts that machine with id 12 is responsible for storing the data with ids 8,9,10,11 and 12, because all these ids are less than or equal to the id of machine 12. The data with all these ids might not be available for storage but if such a data arrives this machine, i.e., machine with id 12, will be used for the storage. Likewise, machine with id 4 is responsible for storing data with ids 2, 3 and 4 (if available).

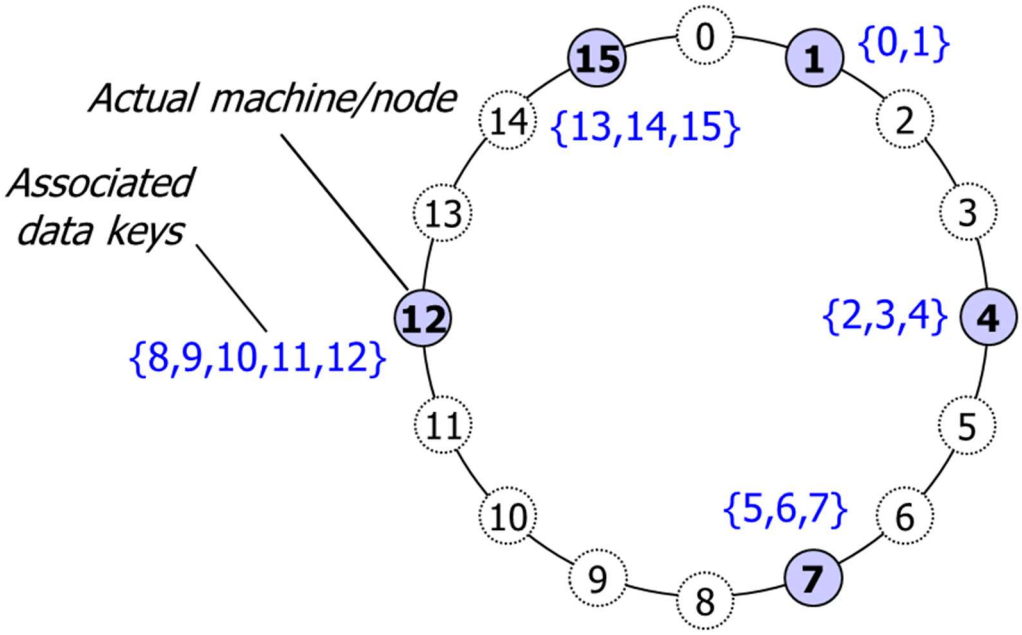

Figure 3

- Each machine stores the data in the form of an AVL tree. **Kindly note** that the example in Figure 3 is very simplistic, however, in reality a machine might be required to store millions of keys, value pairs. Once the machine (on which a data is stored) is identified, AVL tree searching can be used to retrieve the actual data.

Implementation notes: From the implementation perspective, you are supposed to make a circular singly linked list of machines.

Implementation notes: The AVL tree is used to index keys. The corresponding values are stored in the files on the file system. A single file contains up to 100 values. In this case, each AVL tree entry contains the file path and line number where the associated value is stored. For example,

- Key = i190020, Value = Musa bin Tariq
- Key = Intel, Value = price 40$, volume 20000

However, if a value is a file it is stored separately. For example,

- Key = wizard of OZ, Value = actual movie in mp4 file format

Search Algorithm

The request to search a data, i.e., given a key search for the value, can arrive on any machine. The machine will first determine whether the data is locally stored, i.e., id e of the data is less than or equal to the id p of the machine ($p \geq e$). If data is not locally found, the search request is forwarded to the next node in the circular linked list and so on. This very simple search algorithm is not very efficient with a complexity of `O(N)` where N is the number of machines in the distributed system. Why?

To efficiently resolve data identifier e to machine `succ(e)`, each machine p contains routing table FT_p of at most `O(log N)` entries, where N is the number of machines in the distributed system. Each routing table entry i contains the shortcut/pointer to the distant node, i.e., $FTp[i]=succ(p + 2^{i-1})$, where `i=1,...,l≤log(N)`. Figure 4 shows a 5-bit identifier space. The FT of machine 1 includes the links (also called shortcuts or pointers) to the machines who are responsible, i.e., `succ`, for the ids (1+1), (1+2), (1+4), (1+8), (1+16).

Note: Figure 4 depicts the routing tables of machines with ids 1, 4, 9, 11, 14 and 28. Try to construct these routing tables by hand to properly understand the concept of shortcuts.

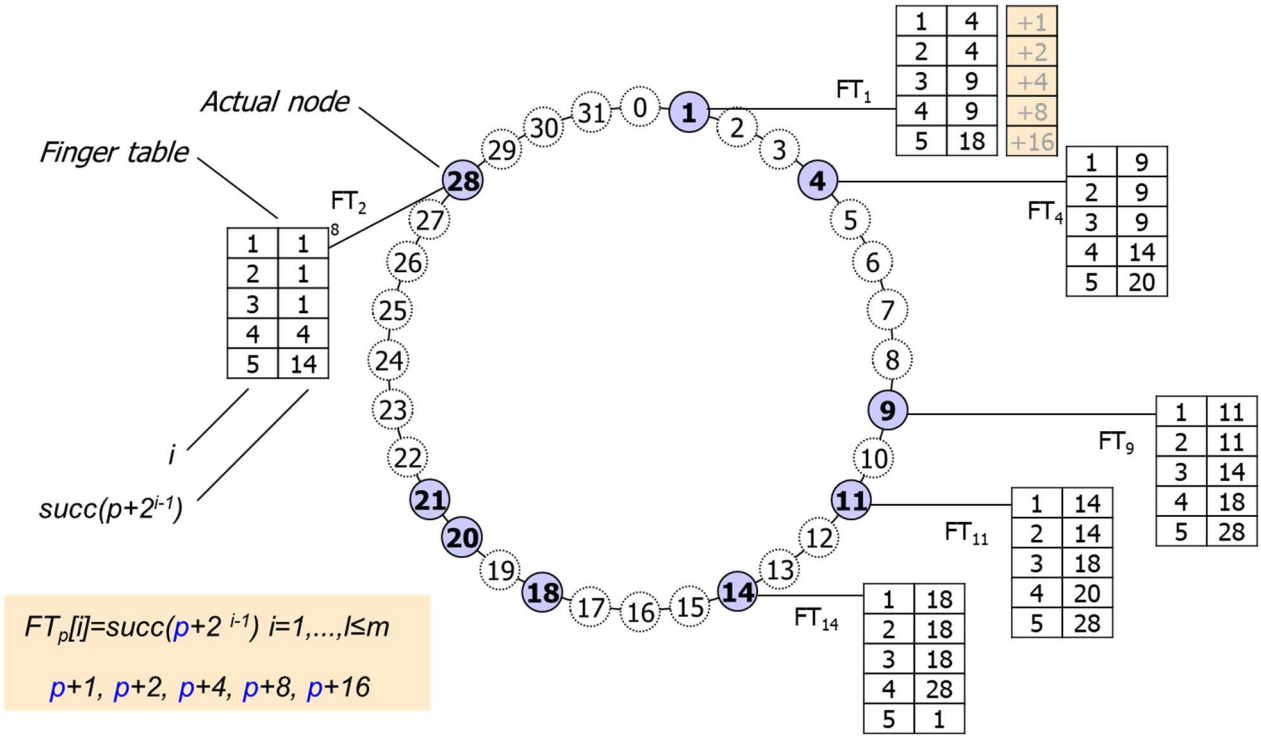

Figure 4

Implementation notes: The routing table is implemented as a doubly linked list. The routing table entries will maintain the address/pointer of the linked list node that corresponds to the machine.

Given the properly maintained routing tables, the search query will be routed as follows:

- A machine p receives a key, i.e., `H(key) = e` and starts the process to search for the corresponding value.
- Machine p receiving search(e) considers following cases:
  - `p=e`, i.e. value is stored on the same machine. Use AVL search to find the data and return the results.
  - `p < e and e ≤ FTp [1].` In this case, the search request is forwarded to the machine `FTp[1]`, i.e., first entry of routing table entry.
  - `FTp[j] < e ≤ FTp[ j+1].` In this case the search request is forwarded to the machine `FTp[j].`

Note: The identity space is circular and modulus operation is required. It is omitted in the above description for the sake simplicity.

The above search algorithm using routing table will result in `O(log N)` lookups, where N is the number of machines in the system. Why? Figure 5 shows the working of above algorithm, where the search query for a data with `H(key) = 12` is originated at node 28.

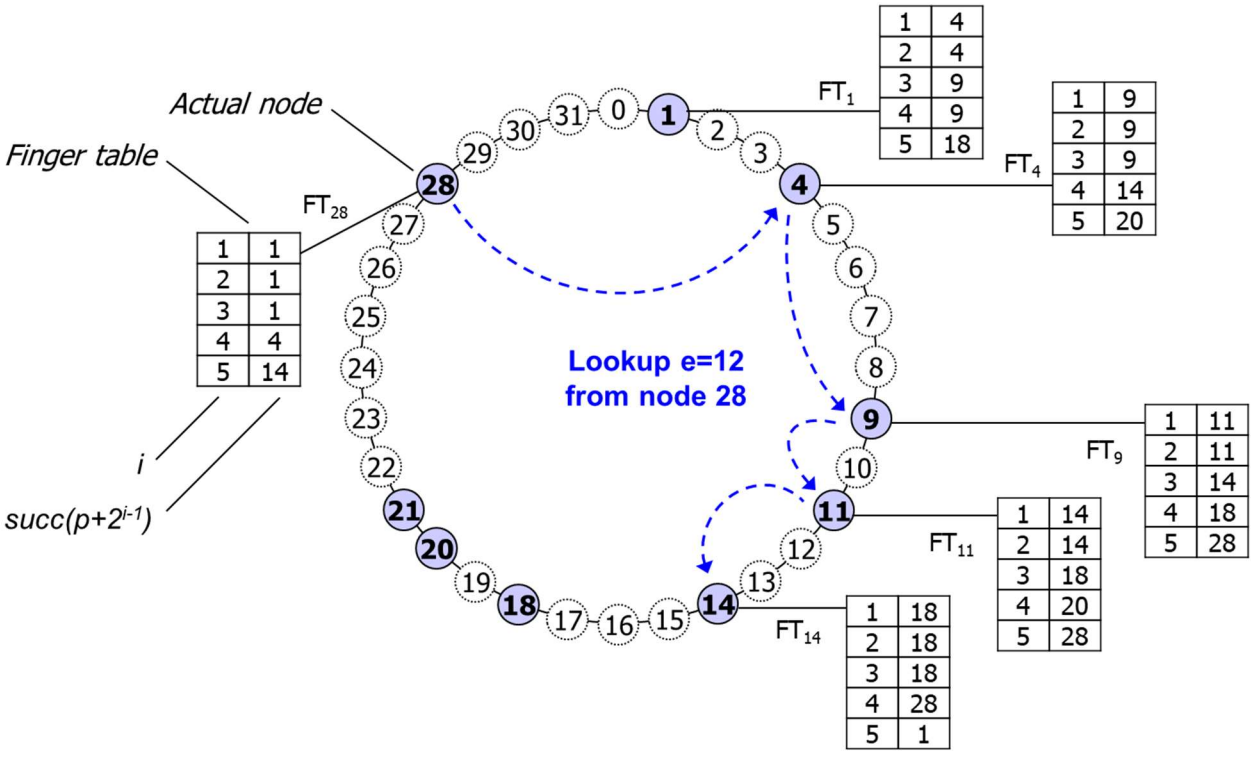

Figure 5

Figure 6 depicts another example of the searching for data with `H(key) = 26` starting from node 1.

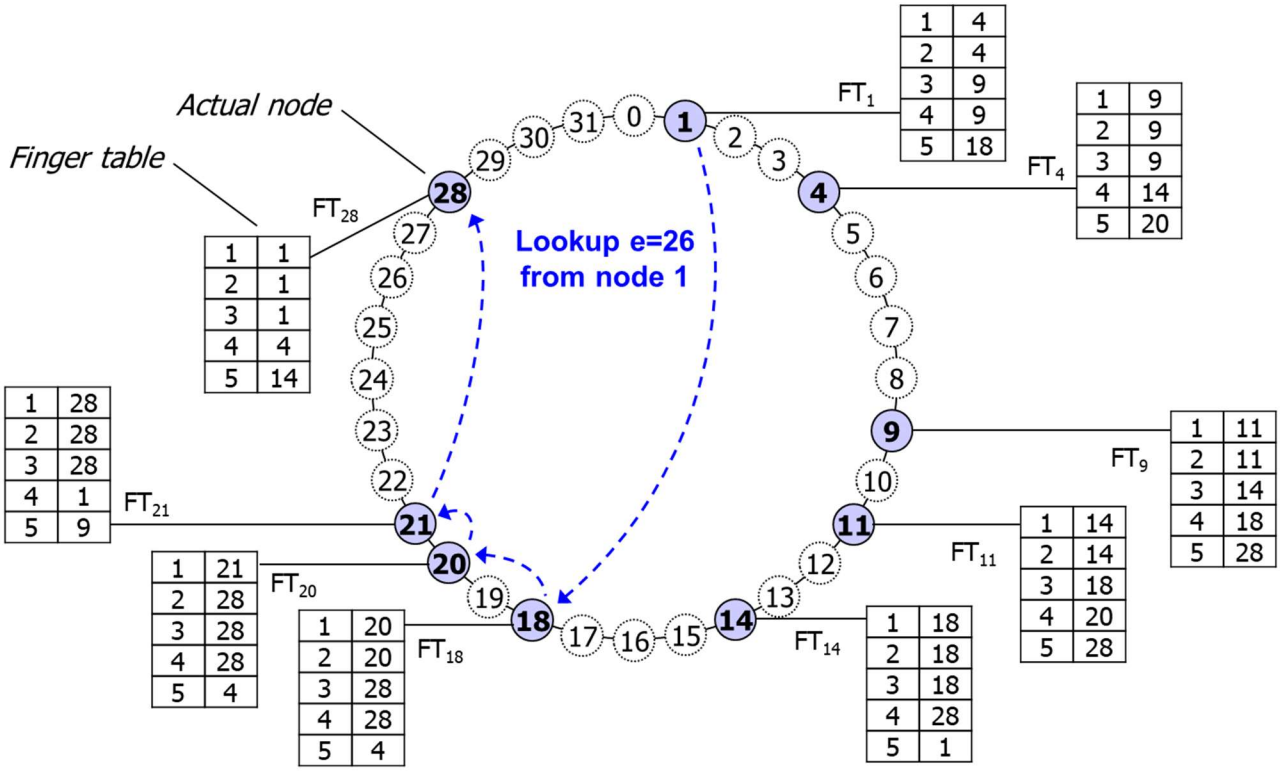

Figure 6

New machine joining

When a new machine joins, the following steps are performed.

- `H(Machine Address) = p` is calculated and machine is placed in the ring accordingly.
- The routing table of p is initialized and routing table of other machines will be updated.
- Id range will be subdivided between p and `succ(p+1)`, i.e., each data e with `e ≤ p` is moved from `succ(p+1)` to p. The AVL trees will be adjusted accordingly.

Machine leaving

In this project, we assume that machines leave gracefully, i.e., all the stored data is distributed to other machine(s) before departure from the network. This may result in the redistribution of Id ranges and adjustment to the AVL trees.

Data Storage query

The data storage request can arrive on any machine. On receiving the key, value pair, a machine will first find the data Id, i.e., `H(key) = e`. An algorithm similar to the search will be used to arrive at machine that is `succ(e)` and responsible for storing the data.

Commands: Your project must support the following commands and options.

1. Option to specify the number of machines.
2. Option to specify the size of identifier space in bits, i.e., 160 bits, 4 bits etc.
3. Option to manually assign Id to each machine. If manual Ids are not assigned by the user, the system must automatically assign Ids.
4. Option to insert data in the form of key, value pairs from any machine. The insertion must show the complete path taken by the request to arrive at the correct machine. Additionally, the option to print AVL tree.
5. Option to remove data by specifying the key from any machine. The output must show the complete path taken by the request to arrive at the correct machine and corresponding value that is removed from the AVL tree. The updated AVL tree must be printed.
6. Option to print the routing table of any machine.
7. Option to print the AVL tree maintain on any machine along with the location of files (and line numbers) on which the corresponding values are stored.
8. Option to add new machines on the fly without disrupting the functionality of Ring DHT.
9. Option to remove any machine on the fly without disrupting the functionality of Ring DHT.

What to submit

Submit your code for this project, programmed in C++ using Visual Studio 2019. A document highlighting the design in terms of relationships/associations between different classes of your program must be submitted. The code needs to be well documented so that grader can get a good idea of what each of your procedures do.
Program modularity, clarity, documentation etc., along with correct implementation will be considered for grading.

Good Luck!