**Project 1**

**Venkata Sai Karthik Metlapalli**

**UFID: 65764476**

**Description**

* This project involves java-based implementation of multi-hashing table, Cuckoo hash table, and d-left hash table.
* The zipped folder includes the source codes and the output text files generated for each of the hashing algorithms

1. **Multi Hash Table:**

* This Multi Hash table is implemented in JAVA and the class file name is Multihash.java.
* To run this file, go to the IDS-1 directory, then type the following commands in command prompt/ terminal
  + - javac Multihash.java
    - java Multihash <no of TableEntries> <no of Flows> <no of Hashes>

This program accepts input parameters through command line arguments. For the demo input used for output file generation, the values of number of table entries, number of flows, and number of hashes are 1000, 1000, and 3, respectively.

In case of wrong input parameters, the program throws valid exception messages to the user suggesting to correct the input.

* After passing the inputs, the parametrized constructor initializes the values passed as class level variables.
* Then, flow IDs are randomly generated using random() function in java and it is assumed that there is one packet per flow for the sake of simplicity.
* The array s[number of hashes for each flow] is also initialized with unique random numbers using the generate function. We also initialize an empty hashTable whose length is equal to the number of entries passed as input.
* Next, in the fillHashTable function, we calculate the XOR of each flowID with all the random numbers in array s[no Of hashes] and store in resultHash array.
* For each of these hashes in resultHash, we try to check if the particular entry is available in hashTable. If it is available, we insert this into the hashtable and increase the totalCount of flows recorded in the hashTable and continue with other flowIds. Else, we discard the flow as no entry was available to insert the value.
* The result is then stored in the “NewOutputMultiHashTable.txt” According to the output the first line is the total count of flow IDs in the hash table and the remaining lines denote the unique flow IDs residing in the table.
* Finally using this technique for the demo input more than 825+ entries of the hash table have been occupied.

1. **Cuckoo Hash Table:**

* This Cuckoo hash table is implemented in JAVA and the class file name is Cuckoohash.java.
* To run this file, go to the IDS-1 directory, then type the following commands in command prompt/ terminal
  + - javac Cuckoohash.java
    - java Cuckoohash <no of TableEntries> <no of Flows> <no of Hashes> <no of Cuckoo Steps>

This program accepts input parameters through command line arguments. For the demo input used for output file generation, the values of number of table entries, number of flows, number of hashes, and number of cuckoo steps are 1000, 1000, 3 and 2 respectively.

* After passing the inputs, the parametrized constructor initializes the values passed as class level variables.
* Then, flow IDs are randomly generated using random() function in java and it is assumed that there is one packet per flow for the sake of simplicity.
* The array s[number of hashes for each flow] is also initialized with unique random numbers using the generate function. We also initialize an empty hashTable whose length is equal to the number of entries passed as input.
* Next, in the fillHashTable function, we calculate the XOR of each flowID with all the random numbers in array s[no Of hashes] and store in resultHash array.
* For each of these hashes in resultHash, we try to check if the particular entry is available in hashTable. If it is available, we insert this into the hashtable and increase the totalCount of flows recorded in the hashTable and continue with other flowIds.
* If all the entries are occupied, we call the Move(Hash, numberOfCuckooSteps) function.This method is used to check if the flowID that currently occupies the entry where the current flow’s hash would be inserted could be moved to some other entry in the hash table. This check is performed for the number of cuckoo steps and if it is possible then move value to different entry and thus current flowID could be inserted in the emptied entry of hash table.
* It generates the output file “NewOutputCuckooHashTable.txt” for storing the data of the has table. As per the project requirement, the first line is the count of flow ID’s in the hash table and the rest of the lines consist of the unique flow ID’s residing in the table.
* Finally using this technique for the demo input more than 915+ entries of the hash table have been occupied.

1. **D-Left Hash Table:**

* This Cuckoo hash table is implemented in JAVA and the class file name is Cuckoohash.java.
* To run this file, go to the IDS-1 directory, then type the following commands in command prompt/ terminal
  + - javac DLefthash.java
    - java DLefthash <no of TableEntries> <no of Flows> <no of Segments>

This program accepts input parameters through command line arguments. For the demo input used for output file generation, the values of number of table entries, number of flows, number of hashes, and number of segments are 1000, 1000, 4 respectively. So, each segment has 250 table entries.

* After passing the inputs, the parametrized constructor initializes the values passed as class level variables.
* Then, flow IDs are randomly generated using random() function in java and it is assumed that there is one packet per flow for the sake of simplicity.
* The array s[number of hashes for each flow] is also initialized with unique random numbers using the generate function. We also initialize an empty hashTable whose length is equal to the number of entries passed as input.
* Next, in the fillHashTable function, we calculate the XOR of each flowID with all the random numbers in array s[no Of hashes] and store in resultHash array.
* For each of these hashes in resultHash, we try to check if the particular entry is available in hashTable. Here the priority of filling the entries is from leftmost segment to right.
* If an entry is available in the left most segment, we fill it and continue with next flowID. Else, we try to sequentially check the segments to the right and insert if an empty entry is available. Else the flow is discarded.
* It generates the output file “NewOutputDLeftHashTable.txt” for storing the data of the has table. As per the project requirement, the first line is the count of flow IDs in the hash table and the rest of the lines consist of the unique flow ID’s residing in the table.
* Finally using this technique for the demo input more than 870+ entries of the hash table have been occupied.