Project – 1 : Hash Tables

The following project contains a zip file, which consists of 3 source code files(java) and corresponding output text files which contains number of flows utilized and the values that each of the hash table inserted into. Below are the steps which guides to run the project on command line to the functionality of 3 hashing techniques – Multi Hash Table, Cuckoo Hash Table, DLeft Hash Table with the respective input values of entries, segments, flows, steps and hash functions.

This project is tested out on both IntelliJ and command line for execution, and below are the screenshots for the outputs.

1. Multi Hash Table:

This algorithm has two functions namely – "constructMultiHashtable" and "insert_MH" which are called for each of the entries to check if there is any available slot to be pushed into the hash table.

a. Using command line, firstly, we need to navigate into the desired directory path of the source code. Secondly, we need to compile the source code using the command "javac P1_MultiHashTable.java"
b. To run the code, we need to use the command "java P1_MultiHashTable". This outputs the values – Number of flows used, hash table entries of flow id's on to the console.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_MultiHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_MultiHashTable
Total Flows Used : 812
--------------------
Hash value at 0 : 0,
Hash value at 1 : 4978,
Hash value at 2 : 1849,
Hash value at 3 : 1232,
```

c. To get the console output to a file we need to execute the command "java P1_MultiHashTable > multiHashOuptut.txt", this allows to capture all the console output to the respective file.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_MultiHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_MultiHashTable > MultiHashOutput.txt
(base) jaswanth@jaswanths-MacBook-Air project1 % ls
DLeft.java                      P1_DLeftHashTable.class         P1_MultiHashTable.java          multiHashingTable.java
DLeftOutput.txt                 P1_DLeftHashTable.java          P1_MultiHash_Output.txt         multiHashingTableOutput.txt
MultiHashOutput.txt             P1_DLeftHash_Output.txt         cuckooHashTable.java
P1_CuckooHashTable.java         P1_MultiHashTable.class         cuckooHashTableOutput.txt
(base) jaswanth@jaswanths-MacBook-Air project1 % cat MultiHashOutput.txt
Total Flows Used : 819
--------------------
Hash value at 0 : 2196,
Hash value at 1 : 5245,
Hash value at 2 : 3246,
```

2. Cuckoo Hash Table:

This algorithm has two functions namely – "constructCuckooHashTable", "insert_CH" and "changeFlowIDToAnotherSlot" which are called for each of the entries to check if there is any available slot to be pushed into the hash table.

a. Using command line, firstly, we need to navigate into the desired directory path of the source code. Secondly, we need to compile the source code using the command "javac P1_CuckooHashTable.java"
b. To run the code, we need to use the command "java P1_CuckooHashTable". This outputs the values – Number of flows used, hash table entries of flow id's on to the console.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_CuckooHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_DLeftHashTable
Total flows Used: 863
Hash value at 0 : 6782,
Hash value at 1 : 3778,
Hash value at 2 : 7709,
Hash value at 3 : 2771,
Hash value at 4 : 6765,
Hash value at 5 : 5664,
```

c. To get the console output to a file we need to execute the command "java P1_ CuckooHashTable > CuckooHashOutput.txt", this allows to capture all the console output to the respective file.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_CuckooHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_DLeftHashTable > CuckooHashOuptut.txt
(base) jaswanth@jaswanths-MacBook-Air project1 % cat CuckooHashOuptut.txt
Total flows Used: 875
Hash value at 0 : 9319,
Hash value at 1 : 9315,
Hash value at 2 : 8214,
Hash value at 3 : 6106,
Hash value at 4 : 1973,
Hash value at 5 : 2937,
```

3. DLeft Hash Table:

This algorithm has two functions namely – "constructHashTable" and "checkinsert_DF" which are called for each of the entries to check if there is any available slot to be pushed into the hash table.

a. Using command line, firstly, we need to navigate into the desired directory path of the source code. Secondly, we need to compile the source code using the command "javac P1_DLeftHashTable.java"
b. To run the code, we need to use the command "java P1_DLeftHashTable". This outputs the values – Number of flows used, hash table entries of flow id's on to the console.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_DLeftHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_DLeftHashTable
Total flows Used: 864
Hash value at 0 : 9180,
Hash value at 1 : 2250,
Hash value at 2 : 135,
Hash value at 3 : 2224,
Hash value at 4 : 4911,
Hash value at 5 : 8034,
```

c. To get the console output to a file we need to execute the command "java P1_DLeftHashTable > DLeftHashOutput.txt", this allows to capture all the console output to the respective file.

```
(base) jaswanth@jaswanths-MacBook-Air project1 % javac P1_DLeftHashTable.java
(base) jaswanth@jaswanths-MacBook-Air project1 % java P1_DLeftHashTable > DLeftHashOuptut.txt
(base) jaswanth@jaswanths-MacBook-Air project1 % cat DLeftHashOuptut.txt
Total flows Used: 870
Hash value at 0 : 5265,
Hash value at 1 : 8653,
Hash value at 2 : 5273,
Hash value at 3 : 7341,
Hash value at 4 : 8666,
Hash value at 5 : 7348,
```

Observations:

We can clearly see that cuckoo hashing technique helps to fill more flow ids into the hash table when compared with the multi hashing table. Also, with the DLeft hashing algorithm we see that more flows are filled in the left part of the Hash table when compared with the right part. It employs a process to fill up the left part first and later works in filling the right part of the hash table.