Middle East Technical University | Department of Computer Engineering

CENG 351

Data Management and File Structures

Fall '2017-2018

Programming Assignment 2 - Bitcoin/Altcoin Hashing

Due Date: January 8, 2018, 23:59

1 Story

With the unbelievable hype of Bitcoin, everyone is trying to get on the train of this gold rush. While you're trying to understand what's going on, you realised that there are way too many things to check *(concept of blockchain, ICOs, AirDrops, mining opportunities, faucets, ponzi schemes, arbitrage, transactions, fees, fluctuations, unconfirmed transactions ...)*. So, you've decided not to deal with this anymore.

Hmm, that doesn't quite make sense. I can't continue the story like that. Going back...

... You realized that there are way too many coins to keep track. So, you've decided to partially buy each of them with all of your savings!

Meh, better but not enough. You should not invest any amount that could affect your life if you lose it. Going back again...

... So you've decided to keep track of every coin before buying. As a promising computer engineer on track, you realised that if you used a hash table, you could use this project in Ceng351, and your assistant would be proud of you *(and even donate you some bitcoins to get aboard!)*.

Well, long story short, your task is to create an extendible hashing application for crypto currencies. Note that this is just a **simulation** for hashing, which means that the hashing will be done **in-memory**. There will **NOT** be any disk access, except file reading for GUI.

(Actually it's partly implemented, but you will get all the credits for it! Yay!)

2 TL; DR

- Implement missing methods in stated classes
- Use given attributes / methods
- Create your own attributes / methods that do not clash with provided ones.
- Parse input file / command line inputs
- Write output to command line
- Have fun with GUI. If you have time, dig the code and try to improve.
  (I'll send you satoshis(?) if you improve the GUI.)

3 Requirements

3.1 GUI Classes and Main Class

GUI classes are already implemented for your convenience. You are not expected to modify anything in:

1. CengCoinExchange.java
2. GUI.java
3. GUIBucket.java
4. GUIBucketList.java
5. GUIHashRow.java
6. GUIHashTable.java
7. GUIConnectorContainer.java

You can modify these files as you wish, but note that you should NOT upload these files, which means any changes in these files will NOT affect grading.

However, there are few methods/variables that is required by GUI to work. They're already marked in your source code.

Please note that you are allowed to add methods, attributes as you like.

Also note that you're free to create additional classes with additional files, the compilation will be done with `*.java` command.

You will calculate the hash of the coins with the formula:
`int hashValue = coin.key() % (hashMod (will be given as parameter));`

The implementation is partly done, you are expected to use given methods/attributes, and implement missing methods in those classes:

3.2 CengCoinParser.java

```
public class CengCoinParser {

  public static ArrayList<CengCoin> parseCoinsFromFile(String filename)
  {
    ArrayList<CengCoin> coinList = new ArrayList<CengCoin>();

    // You need to parse the input file in order to use GUI tables.
    // TODO: Parse the input file, and convert them into CengCoins

    return coinList;
  }
  public static void startParsingCommandLine() throws IOException
  {
    // TODO: Start listening and parsing command line -System.in-.
    // There are 4 commands:
    // 1) quit : End the app, gracefully. Print nothing, call nothing.
    // 2) add : Parse and create the coin, and call ↩
        CengCoinExchange.addCoin(newlyCreatedCoin).
    // 3) search : Parse the key, and call CengCoinExchange.searchCoin(parsedKey).
    // 4) print : Print the whole hash table with the corresponding buckets, ↩
        call CengCoinExchange.printEverything().

    // Commands (quit, add, search, print) are case-insensitive.
  }
}
```

3.3 CengBucket.java

```
public class CengBucket {

  // GUI-Based Methods
  // These methods are required by GUI to work properly.

  public int coinCount()
  {
    // TODO: Return the coin count in the bucket.
    return 0;
  }
  public CengCoin coinAtIndex(int index)
  {
    // TODO: Return the corresponding coin at the index.
    return null;
  }
  public int getHashPrefix()
  {
    // TODO: Return hash prefix length.
    return 0;
  }
  public Boolean isVisited()
  {
    // TODO: Return whether the bucket is found while searching.
    return false;
  }
}
```

3.4 CengBucketList.java

```
public class CengBucketList {

  public CengBucketList()
  {
    // TODO: Create a bucket list with only 1 bucket.
  }

  public void addCoin(CengCoin coin)
  {
    // TODO: Empty Implementation
  }
  public void searchCoin(Integer key)
  {
    // TODO: Empty Implementation
  }
  public void print()
  {
    // TODO: Empty Implementation
  }

  // GUI-Based Methods
  // These methods are required by GUI to work properly.
  public int bucketCount()
  {
    // TODO: Return all bucket count.
    return 0;
  }
  public CengBucket bucketAtIndex(int index)
  {
    // TODO: Return corresponding bucket at index.
    return null;
  }
}
```

3.5 CengHashRow.java

```
public class CengHashRow {
  // GUI-Based Methods
  // These methods are required by GUI to work properly.
  public String hashPrefix()
  {
    // TODO: Return row's hash prefix (such as 0, 01, 010, ...)
    return "-1";
  }
  public CengBucket getBucket()
  {
    // TODO: Return the bucket that the row points at.
    return null;
  }
  public boolean isVisited()
  {
    // TODO: Return whether the row is used while searching.
    return false;
  }
}
```

3.6 CengHashTable.java

```
public class CengHashTable {

  public CengHashTable()
  {
    // TODO: Create a hash table with only 1 row.
  }

  public void addCoin(CengCoin coin)
  {
    // TODO: Empty Implementation
  }
  public void searchCoin(Integer key)
  {
    // TODO: Empty Implementation
  }
  public void print()
  {
    // TODO: Empty Implementation
  }

  // GUI-Based Methods
  // These methods are required by GUI to work properly.

  public int prefixBitCount()
  {
    // TODO: Return table's hash prefix length.
    return 0;
  }

  public int rowCount()
  {
    // TODO: Return the count of HashRows in table.
    return 0;
  }

  public CengHashRow rowAtIndex(int index)
  {
    // TODO: Return corresponding hashRow at index.
    return null;
  }
}
```

4 Command Line Specifications

(Assumption: Every file needed is available in current folder.)

Usage:

```
javac *.java
```

```
java CengCoinExchange <hashMod> <bucketSize> <inputFileName> <enableGUI>
```

Ex: `java CengCoinExchange 8 2 coins_20.txt True`

5 Commands

5.1 Adding Coin:

```
add|-coinKey-|-coinName-|-coinValue-|-coinCurrency-
```

Ex: `add|0|Ethereum|46|€`

Output: None.

5.2 Searching Coin:

```
search|-coinKey-
```

Ex: `search|5`

Output: Print every hash row that can lead to the bucket for the corresponding key.

Details are in Output Format section.

5.3 Printing Hash Table:

```
print
```

Output: Print every hash row, and every bucket, and every coin within buckets.

Details are in Output Format section.

5.4 Quit:

```
quit
```

Output: None. End the application, gracefully.

6 Input Format

Input file only holds coins with command `list`, line by line. A line is shown as: *(Delimiter is pipe, |)*

`list|-coinKey-|-coinName-|-coinValue-|-coinCurrency-`

Ex: `list|14|Monero|51|€`

- There will not be any erroneous input, neither in inputFile, nor in command line.
- Coin keys are unique, and always numeric. Other attributes may not (and probably will not) be unique.
- Coin attributes may have spaces, tabs, punctuation marks, except newline and the delimiter (pipe). Parse accordingly.

About main parameters:

- `hashMod` : Integer, > 1, and will always be 2^n.
- `bucketSize` : Integer, > 0.
- `inputFileName` : String, will only be used for GUI. A dummy file will be provided while grading.
- `guiEnabled` : Boolean, will be always `False` while grading.

7 Output Format

7.1 Coins

Coins should be printed with its hash value, and every information.

```
<coin>
   <hash>-hashValueOfCoin-</hash>
   <value>-key-|-name-|-value-|-currency-</value>
</coin>
```

Ex:

```
<coin>
   <hash>001</hash>
   <value>1|Dash|68|€</value>
</coin>
```

7.2 Buckets

Bucket should print the information of self, and the coins holding.

```
<bucket>
   <hashLength>-bucketHashPrefixLength-</hashLength>
   -coin1-
   -coin2-
</bucket>
```

Ex:

```
<bucket>
   <hashLength>2</hashLength>
   <coin>
      <hash>000</hash>
      <value>0|Ethereum|46|€</value>
   </coin>
   <coin>
      <hash>001</hash>
      <value>1|Dash|68|€</value>
   </coin>
</bucket>
```

7.3 Hash Rows:

Rows should be printed like:

```
<row>
   <hashPrefix>-rowHashPrefix-</hashPrefix>
   -bucket-
</row>
```

Ex:

```
<row>
   <hashPrefix>00</hashPrefix>
   <bucket>
      <hashLength>2</hashLength>
      <coin>
         <hash>000</hash>
         <value>0|Ethereum|46|€</value>
      </coin>
      <coin>
         <hash>001</hash>
         <value>1|Dash|68|€</value>
      </coin>
   </bucket>
</row>
```

7.4 Hash Table Printing

Whole table should be printed like:

```
BEGIN TABLE
-row1-
-row2-
...
END TABLE
```

Ex:

```
print
BEGIN TABLE
<row>
        <hashPrefix>00</hashPrefix>
        <bucket>
                <hashLength>2</hashLength>
                <coin>
                        <hash>000</hash>
                        <value>0|Ethereum|46|€</value>
                </coin>
                <coin>
                        <hash>001</hash>
                        <value>1|Dash|68|₺</value>
                </coin>
        </bucket>
</row>
<row>
        <hashPrefix>01</hashPrefix>
        <bucket>
                <hashLength>2</hashLength>
                <coin>
                        <hash>010</hash>
                        <value>2|IOTA|2|€</value>
                </coin>
        </bucket>
</row>
<row>
        <hashPrefix>10</hashPrefix>
        <bucket>
                <hashLength>1</hashLength>
        </bucket>
</row>
<row>
        <hashPrefix>11</hashPrefix>
        <bucket>
                <hashLength>1</hashLength>
        </bucket>
</row>
END TABLE
```

7.5 Search Printing

Basically the same with Hash Table Printing, but with only the rows that leads to the bucket that has the corresponding coin with the key.

Ex:

```
BEGIN SEARCH
-foundRow1-
-foundRow2-
...
END SEARCH
```

If not found, you should print "None" between begin-end search.

```
search|2
BEGIN SEARCH
<row>
        <hashPrefix>01</hashPrefix>
        <bucket>
                <hashLength>2</hashLength>
                <coin>
                        <hash>010</hash>
                        <value>2|IOTA|2|€</value>
                </coin>
        </bucket>
</row>
END SEARCH
```

8 Submission

What to submit?

1. CengBucket.java
2. CengBucketList.java
3. CengCoin.java
4. CengCoinParser.java
5. CengHashRow.java
6. CengHashTable.java

Note: You should **NOT** upload CengCoinExchange.java or any file that starts with GUI. They will be overridden while testing.

Archive them with:

```
tar -cvf <yourID>.tar.gz CengBucket.java CengBucketList.java CengCoin.java
CengCoinParser.java CengHashRow.java CengHashTable.java
```

It will be extracted with:

```
tar -xvf <yourID>.tar.gz
```

Note: If extracting results within a folder, your submission cannot be graded.

Note: If you are using an IDE (like Eclipse), do not change the package of the classes.

As mentioned in Command Line Specifications, your submission should be able to run with:

```
javac *.java
```

```
java CengCoinExchange <hashMod> <bucketSize> <inputFileName> <enableGUI>
```

Also as mentioned before, GUI will **NOT** be used while grading.

Good luck, have fun.

- engin