Middle East Technical University

Department of Computer Engineering

CENG 351

Data Management & File Structures

Fall 2021–2022

Programming Assignment 2 — Pokemon Hashing

1 About The Homework

You are a young, ambitious, enthusiast Pokemon Trainer who started training at the age of 11. You had an unstoppable winning streak on duels against every single soul. Unfortunately, one day you've faced the tragedy. You've lost the final duel on the city tournaments. After that lost, you even thought to quit training. However, your ambitious personality surpassed.

You decided to collect every single Pokemon, thus creating the most powerful training set the world had ever seen. However, you need to keep track of every single Pokemon. As a Pokemon Trainer who is pursuing Bachelor's degree on Computer Engineering, you decided to use Hash table for storing Pokemons.

```
For the sake of the story, assume that we foresaw your situation
and decided to help you by creating a GUI and providing
some helpful structures.
```

In short, you need to create an **extendible hashing** application for storing the Pokemons. Your application will be a **simulation** for hashing, which means that the hashing will be done **in-memory**. There will **NOT BE ANY DISK ACCESS** (except reading the file for GUI). You will also be able to use the GUI that we've provided. GUI is just created for better understanding of the homework. Neither you have to use it nor it will be graded.

2 TODO's

2.1 Must

- Parse the user input
- Implement the methods inside CengHashTable.java (if you feel like, you may use other helper classes)
- Write output to the command line with proper format

2.2 Optional

All the stuff related to the GUI is optional.

- You can fill other Ceng methods for GUI (or for better implementation practice)
- Parse the input file for GUI

3 Files

There are 13 files provided with studentPack.tar.gz. 6 of them are related with GUI which are:

- GUI
- GUIBucket
- GUIBucketList
- GUIConnectorContainer
- GUIHashRow
- GUIHashTable

CengPokeKeeper is the main class of the application.
There are 6 more classes designed for the application which are:

- CengBucket
- CengBucketList
- CengPoke
- CengPokeParser
- CengHashRow
- CengHashTable

You should not change the Main class or GUI related classes. You should complete the parts inside the given 6 classes. You don't need to use all of them or you can add any other classes to the project. However, **please check how the CengPokeKeeper runs your classes or how you should run it**. The main file will not be changed so complete your design according to that.

3.1 CengBucket

Keeps track of your Pokemons inside the buckets. There are several methods that should be implemented in order to use the GUI properly. These methods are separated by comments. You can define your own variables or methods inside the class.

3.2 CengBucketList

Keeps track of all buckets. There are several methods that should be implemented in order to use the GUI properly. These methods are separated by comments. You can define your own variables or methods inside the class.

3.3 CengPoke

Your Pokemon class. Keeps the information of a Pokemon. The informations of them are comming from the input. A Pokemon should have:

- PokemonKey
- PokemonName
- PokemonPower
- PokemonType

3.4 CengPokeParser

Parser class. There are two methods inside the class. One of them, parseTheFile, is parsing the file which will be used for GUI. Its implementation is optional.
The other method is parseCommandLine. This is a must method for you to implement. The method should continuously read the commandline until it gets "quit" command. The input details will be explained later.

3.5 CengHashRow

Keeps track of each hash row. There are several GUI based methods you need to fill inside the class. Each hash row points exactly one bucket. However, a bucket can be pointed by more than one hash row.

3.6 CengHashTable

The upper most class of your extendible hashing application. Keeps track of your whole table. It should know all your hashrows and bucketlist. The main tasks should be handled here (directly or by directing to the related methods).

4 Tasks

There are 5 tasks that you should follow in order to create a proper extendible hashing application.

4.1 Add Pokemon

You need to add the given pokemon to the hash table by using hashMod variable. The hashMod variable will be provided as a parameter.

You can find the correct bucket using the following formula:

```
int hashValue = (int) pokeKey % hashMod
```

You should perform the insertion by most significant bits (**prefix**).

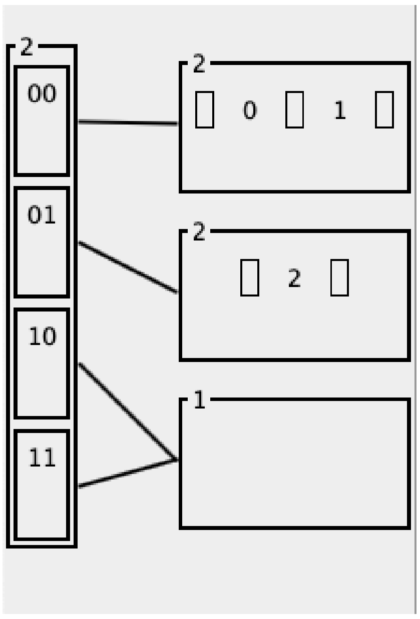
Buckets can hold certain amount of Pokemons (this value will also be given as a parameter). So, you should extend the table if needed, otherwise it will not be extendible hashing :)

*There are **no output** for this task.*

You can follow the steps explained in the lecture slides, except we are using the **prefix** of the hashValue.

Ex:

```
Assume the following:
bucketSize = 2
hashMod = 8
And the table is in the given state.
```



If we add 19 to the table. The hashValue of 19 is 3 which is:
011 in 3 bits (since hashMod is 8 there should be 3 bits (2^3)). Since we are using **prefix** of the hashValue and the global depth is 2, we should take the first 2 bits of 011 which is 01. So, we should insert 19 to the bucket of the row 01 (next to 2).

Also, note that there can be multiple extensions for 1 insertion. (Ex: Bucket is full, extend and the bucket is again full, extend again.).

4.2 Delete Pokemon

You need to delete the given pokemon from the hash table. You don't need to merge any buckets. Don't shrink the table also. Just find the bucket that stores the given pokemon and delete the pokemon from the Bucket. There is no additional operation needed.

In this simulation, there can be a lot of empty buckets since, we are not merging neither the buckets nor the rows.

Moreover, **after** deletion, you should print the number of empty buckets on the table.

Exact output format will be explained on the Output section.

4.3 Search Pokemon

Search the given pokemon on your table. You should print every hashrow (along with buckets that they point) that contains the pokemon mentioned.

Exact output format will be explained on the Output section.

4.4 Print Table

Print the whole hash table. (Check table output from outputs section).

Exact output format will be explained on the Output section.

4.5 Quit

Finish the application do nothing else.

5 Run the Application

```
javac *.java
java CengPokeKeeper <hashMod> <bucketSize> <inputFileName> <enableGUI>

Ex: java CengPokeKeeper 8 2 pokemons.txt True
```

- hashMod > 1 and will be 2^n format.
- bucketSize > 0
- inputFileName is a file that is inside the same directory. Since it is GUI related it will not be tested. However, you should provide some file in order to CengPokeKeeper to work.
- enableGUI will be False for testing.

6 Input Formats

6.1 Command inputs

The command line inputs to use the application properly. There is 5 different input types for each of your tasks.
For all tasks inputs are separated with delimiter tab if needed. (the - sign will not be in the actual input)

6.1.1 Add Pokemon

```
add    -pokeKey-    -pokeName-  -pokePower-    -pokeType-
```

All given values corresponding to the Pokemon attributes (check CengPoke class).

6.1.2 Delete Pokemon

```
delete    -pokeKey-
```

6.1.3 Search Pokemon

```
search    -pokeKey-
```

6.1.4 Print Table

```
print
```

6.1.5 Quit

```
quit
```

6.2 Input File

Input file will only be used for GUI. Each line in the input file corresponds to a Pokemon. Each line starts with a special tag "poke". As in the command line inputs the separation is done by the delimiter of tab if needed. Ex:

```
poke    -pokeKey-   -pokeName-    -pokePower-   -pokeType-
```

6.3 General Input Specifications

- Either Input file, or command line inputs will not contain any errors.
- Input file will not be used for testing as well as any GUI specific features.
- Pokemons' key will be unique.
- Pokemon names or types might contain special characters (except delimiter).

7 Output Format

All the printing will be done JSONLike format. You don't need to numerate multiple same type keys (ex: pokemons inside buckets. "poke" tag is sufficient for all). Be careful with commas. Each inner segment should be aligned with tab (\t) character.

7.1 Pokemon

Printing a pokemon should be done as follows:

```
"poke": {
     "hash": hashValueofthePokemon,
     "pokeKey": -pokeKey-,
     "pokeName": -pokeName-,
     "pokePower": -pokePower-,
     "pokeType": -pokeType-
}
```

7.2 Bucket

Printing a bucket should include its own information and also pokemons inside of it as shown in the Pokemon output:

```
"bucket": {
    "hashLength": hashLengthofBucket,
    "pokes": [
        "poke": {
            POKEMON PRINT
        },
        "poke": {
            POKEMON PRINT
        },
        .
        .
        "poke": {
            POKEMON PRINT
        }
    ]
}
```

7.3 Row

Printing a row should print the row's hash Prefix and the bucket that it points to. For bucket check bucket output style.

```
"row": {
    "hashPref": hashPrefixofTheRow,
    "bucket": {
        BUCKET PRINT
    }
}
```

7.4 Table

Printing the entire table should print all the rows inside the table.

```
"table": {
    "row": {
        ROW PRINT
    },
    "row": {
        ROW PRINT
    },
    .
    .
    .
    "row": {
        ROW PRINT
    }
}
```

7.5 Search

Output of the search command should give information of all rows that points to a bucket that contains the given pokemon. The format is given below (similar to the table printing but only rows that contains the buckets of the searched pokemon).

```
"search": {
    "row": {
        ROW PRINT
    },
    "row": {
        ROW PRINT
    },
    .
    .
    .
    "row": {
        ROW PRINT
    }
}
```

If the Pokemon is not found (there is no such pokemon), leave the search empty.

```
"search": {
}
```

Please check sample output file for consistency among input spacings or commas.

7.6 Delete

After deletion, you should print the number of empty buckets on the table.

```
"delete": {
   "emptyBucketNum": numberOfEmptyBuckets (an integer value)
}
```

8 Submission

You should send the following files:

- CengBucket.java
- CengBucketList.java
- CengPoke.java
- CengPokeParser.java
- CengHashRow.java
- CengHashTable.java

You need to submit the homework as tar.gz file. You can use the following command:

```
tar -cvf <yourID>.tar.gz CengBucket.java CengBucketList.java
CengPoke.java CengPokeParser.java CengHashRow.java CengHashTable.java
```

They will be extracted using:

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

Be sure that your tar extraction **does not** result within a folder. In such case, your homework can not be graded.

Do not send the Main class or GUI related classes. Even if you send they will be changed, modifications inside them will not be used.

9 Important Notes

- You should use Java (version 13).
- The assignment is designed for individual work, **YOU ARE NOT ALLOWED TO SHARE CODE SEGMENTS WITH YOUR FRIENDS**. That will be considered as **cheating**.
- Using the sources on the internet are also **not allowed**. Again, this also will be considered as **cheating**.