COL106: Assignment 4

Trie, Red-Black tree and Priority queue

September 13, 2019

Logistics:
Release date: September 13, 2019
Submission deadline: TBA
Total marks: 5
PDF Version: Assignment 4 PDF

Brief description:
In this assignment you need to work with *Tries*, *Red-Black trees* and *Priority queues.* There will be **Four** components of the assignment. The first three will check *tries, red-black trees* and *priority queues* independently. The last part of the assignment will be a combination of all the previous components.

1 General instructions

The grading will be done automatically. To ensure a smooth process, an interface will be provided to you, which you are **NOT** suppose to change. Your solution classes will implement these interfaces.
For each of the component, you will be given and *input* file, which will contain the commands that your code must execute. As per the command, the program will produce the *output*, which will be compared against an expected output for grading. Please ensure that you follow the proper formatting criteria, failing to do so will results in *0* for that particular component.

1.1 Code skeleton

You are provided with the skeleton of the code. This contains the interfaces and other relevant information. Your task is to fill up the empty functions. The code also contains *driver code* for all the components of assignment. These will be used to check the

correctness of the code. Please **DO NOT** modify the interface and the driver code. You are free to change and implement other parts in any way you like.
Code can be downloaded from here: Download code

1.1.1 Building and Running

In the code, within the src folder, you can use the following commands to check your code.

```
make
```

This will check all the components. Components can also be checked independently:

```
make trie
make rbtree
make pq
make pm
```

for Trie, Red-Black tree, Priority-Queue and Project-Management (4th Component) respectively.

2 Trie [1 Mark]

Trie is an efficient information reTrieval data structure. Using Trie, search complexities can be brought to optimal limit (key length) [2].
In this part of the assignment, you need to implement a Trie data structure. To make things interesting, you will be implementing a telephone directory using Tries. Name of a person will be the key (assuming all names are unique). Associate with every name will be a `Person` object.

```java
package Trie;
public class Person {
    public Person(String name, String phone_number) {
    }
    public String getName() {
        return "";
    }
}
```

Listing 1: Person class.

2.1 Interface

You version of Trie must implement the TrieInterface as shown in Listing 2 and also provided as code.

```java
package Trie;

public interface TrieInterface<T> {

    boolean insert(String word, T value);

    TrieNode<T> search(String word);

    TrieNode<T> startsWith(String prefix);

    void printTrie(TrieNode trieNode);

    boolean delete(String word);

    void print();

    void printLevel(int level);
}
```

Listing 2: Interface specifications for Trie.

2.2 Input specifications

Commands:

1. INSERT: It takes a String as input and inserts that into the trie.
2. DELETE: It takes a String as an input and deletes that from the trie.
3. SEARCH: It takes a String as input and returns *true* or *false*, based on whether that word is present in trie or now.
4. MATCH: It takes a String as an input, and return all words where the suffix is the entered String.
5. PRINT: Print all the words inside the trie.
6. PRINTLEVEL: Print the specified level.

Sample input file:

```
INSERT
Diljeet Singh, +91987654321
INSERT
```

```
Bhavesh Kumar, +91987654321
INSERT
Chayan Malhotra, +91987654321
INSERT
Ekta Mittal, +91987654321
INSERT
Farhan Khan, +91987654321
INSERT
Dishant Goyal, +91987654321
INSERT
Dishant Kumar, +91987654321
INSERT
Dishant Gupta, +91987654321
SEARCH
Dishant Goyal
MATCH Di
MATCH di
DELETE
Dishant Goyal
SEARCH
Dishant Goyal
MATCH SK
PRINTLEVEL 2
PRINT
```

Listing 3: Input for Trie.

Expected Output file:

```
Inserting: Diljeet Singh
Inserting: Bhavesh Kumar
Inserting: Chayan Malhotra
Inserting: Ekta Mittal
Inserting: Farhan Khan
Inserting: Dishant Goyal
Inserting: Dishant Kumar
Inserting: Dishant Gupta
Searching: Dishant Goyal
FOUND
[Name: Dishant Goyal, Phone=+91987654321]
Matching: Di
MATCHED:
[Name: Diljeet Singh, Phone=+91987654321]
[Name: Dishant Kumar, Phone=+91987654321]
```

```
[Name: Dishant Goyal, Phone=+91987654321]
[Name: Dishant Gupta, Phone=+91987654321]
Matching: di
NOT FOUND
Deleting: Dishant Goyal
Searching: Dishant Goyal
NOT FOUND
Matching: SK
NOT FOUND
Level 2: h h i k a
-------------
Printing Trie
Level 1: B C D E F
Level 2: h h i k a
Level 3: a a s l t r
Level 4: v y h j a h
Level 5: e a a e   a
Level 6: s n n e M n
Level 7: h   t t i
Level 8:   M     t K
Level 9: K a G K S t h
Level 10: u l u u i a a
Level 11: m h p m n l n
Level 12: a o t a g
Level 13: r t a r h
Level 14: r
Level 15: a
Level 16:
-------------
```

Listing 4: Ouput for Trie.

3 Red-Black Tree [1 Mark]

In this part you need to implement a Red-Black tree. A tutorial on Red-Black tree can be found here [1]. In the part the basic operations on a Red-Black tree, insert and search will be tested. You will be given an input file, whose format is listed in Section 3.2. A sample output for the input command given in Section 3.2 is shown in 7
In this case also you will implement a telephone directory, with an added feature that a person can have multiple numbers.

3.1 Specifications

You Red-Black tree, must implement the interface as shown in listing 5.

```
package RedBlack;
public interface RBTreeInterface<T extends Comparable, E> {

    void insert(T key, E value);

    RedBlackNode<T, E> search(T key);
}
```

Listing 5: Input for Trie.

Things to keep in mind:

- All the items insert into the RB-Tree has a key and the corresponding value with it. In this version of Red-Black tree, a *key* can have multiple items. If we are trying to insert an element with a key which is already present in the tree, the value will get attached to that key. This can be seen in the Listing 6, Line 11 and 13.

3.2 Input specifications

Commands:

1. INSERT int: Insert an element to the tree.
2. SEARCH int: Searches for a particular element in the tree.

Sample input (ignore the line numbers):

```
1  INSERT
2  Diljeet Singh, +91987654321
3  INSERT
4  Bhavesh Kumar, +91987654321
5  INSERT
6  Chayan Malhotra, +91987654321
7  INSERT
8  Ekta Mittal, +91987654321
9  INSERT
10 Farhan Khan, +91987654321
11 INSERT
12 Dishant Goyal, +91987654321
13 INSERT
14 Dishant Goyal, +91999999999
15 INSERT
```

```
Dishant Kumar, +91987654321
INSERT
Dishant Gupta, +91987654321
SEARCH
Dishant Goyal
SEARCH
Sandeep
```

Listing 6: Input for RedBlack Tree.

Expected Output (ignore the line numbers):

```
Inserting: Diljeet Singh
Inserting: Bhavesh Kumar
Inserting: Chayan Malhotra
Inserting: Ekta Mittal
Inserting: Farhan Khan
Inserting: Dishant Goyal
Inserting: Dishant Goyal
Inserting: Dishant Kumar
Inserting: Dishant Gupta
Searching for: Dishant Goyal
[Name: Dishant Goyal, Phone=+91987654321]
[Name: Dishant Goyal, Phone=+91999999999]
Searching for: Sandeep
Not Found
```

Listing 7: Output for RedBlack Tree.

4 Priority queues [1 Mark]

In this part you need to implement a *priority queue*. Specifically, you will be implementing a *max-heap* which is an implementation of priority queue.
You will need to implement a scoring record using Max Heap. This will contains, students name and their corresponding marks. The max-heap will use the marks to arrange the students, i.e. the student with the highest marks will be on the top.

4.1 Specifications

```
package PriorityQueue;

public interface PriorityQueueInterface<T extends Comparable> {

    void insert(T element);

    T extractMax();
}
```

Listing 8: Interface for PriorityQueue.

Commands

1. INSERT
   name marks: Insert the current integer in the tree.
2. EXTRACTMAX: Extract the student with highest marks and print it. Extract operations also removes this from the max-heap.

Sample input (ignore the line numbers):

```
INSERT
Diljeet Singh, 10
INSERT
Bhavesh Kumar, 100
INSERT
Dishant Kumar, 67
EXTRACTMAX
EXTRACTMAX
EXTRACTMAX
EXTRACTMAX
```

Listing 9: Input for PriorityQueue.

Expected Output (ignore the line numbers):

```
Inserting: Diljeet Singh
Inserting: Bhavesh Kumar
Inserting: Dishant Kumar
Student{name='Bhavesh Kumar', marks=100}
Student{name='Dishant Kumar', marks=67}
Student{name='Diljeet Singh', marks=10}
Heap is empty.
```

Listing 10: Output for PriorityQueue.

5 Project Management (Scheduler) [2 Marks]

In this part of the assignment you need to combine all the previous components of the assignment, Trie, Red-Black Tree and Priority Queue to implement a Job scheduler. The main part of this part are:

1. **Project:**
   The project class will be have a *name*, *budget* and *priority* (as shown in Listing 11).

```
package ProjectManagement;
public class Project {
}
```

Listing 11: Project class

2. **User:**

```
package ProjectManagement;
public class User implements Comparable<User> {
    @Override
    public int compareTo(User user) {
        return 0;
    }
}
```

Listing 12: User class

3. **Job:**

```
package ProjectManagement;
public class Job implements Comparable<Job> {
    @Override
    public int compareTo(Job job) {
        return 0;
    }
}
```

Listing 13: Job class

   A job can have two status: `REQUESTED, COMPLETED`.

5.1 Specifications

The main component is *Job*. As shown in Listing 13, each Job will belong to a Project and created by an User. The name of the Jobs will be unique (this is guaranteed in the test cases). All the jobs have a running time, i.e. the time required to run this job. The priority

of a job is same as of that its project and a job can only run if its running time is less than the current budget of the Project. Successfully running a Job, will reduce the budget of that project by running time of the project.
All the projects will be stored in a Trie, using the project name as the *key*. Project names will be unique. All the Jobs will be stored in a *Priority Queue*, specifically a Max-Heap, using their priorities as the key.

5.2 Commands

A sample input file is shown in Listing 15.

1. USER: Create the user with given user name.
2. PROJECT: Create a project. NAME PRIORITY BUDGET
3. JOB: Create a job. NAME PROJECT USER RUNTIME
4. QUERY: Return the status of the Job queried.
5. ADD: Increase the budget of the project. PROJECT BUDGET
6. EMPTY_LINE: Let the scheduler execute a single JOB.

5.3 Scheduler specifications

The scheduler will execute a single whenever it will encounter an empty line in the input specifications. After the end of the INP file, scheduler will continue to execute jobs till there are jobs left that can be executed.
Each time the scheduler executes a job, it will do the following:

1. It selects the job with the highest priority from the MAX HEAP.
2. It first check the running time of the Job, say t.
3. It will then fetch the project from the RB-Tree and check its budget, say B.
4. If $B \geq t$ then the code is executed.
    - Executing a job means:
    - Set the status of the job to complete.
    - Increase the global time by job time.
    - Set the completed time of the job as the current global time.
    - Decrease the budget of the project by run-time of the job. i.e. $\hat{B} = B - t$, where $\hat{B}$ is the new budget of the project.

5. If: $B < t$, then select the next job and try to execute this.

6. A scheduler will return in following cases:

   - It successfully executed a job.
   - There are no jobs to be executed.
   - None of the jobs can be executed because of the budget issue.

7. After the execution returns, process the next *batch* of commands (all the commands till next EMPTY_LINE or EOF).

8. If there are no more commands in the INP file, then let the scheduler execute jobs till there are no jobs left, or no jobs can be executed because of budget issues. This marks the END of the execution.

9. Print the stats of the current system. See Listing 16.

```java
package ProjectManagement;
public interface SchedulerInterface {
    void run_to_completion();
    void handle_project(String[] cmd);
    void handle_job(String[] cmd);
    void handle_user(String name);
    void handle_query(String key);
    void handle_empty_line();
    void handle_add(String[] cmd);
    void print_stats();
    void schedule();
}
```

Listing 14: Interface specification

```
USER Rob
USER Harry
USER Carry
PROJECT IITD.CS.ML.ICML 10 15
PROJECT IITD.CS.OS.ASPLOS 9 100
PROJECT IITD.CS.TH.SODA 8 100
JOB DeepLearning IITD.CS.ML.ICML Rob 10
JOB ImageProcessing IITD.CS.ML.ICML Carry 10
JOB Pipeline IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans IITD.CS.TH.SODA Carry 10

```

```
QUERY Kmeans
QUERY Doesnotexists

JOB DeepLearning IITD.CS.ML.ICM Rob 10
JOB DeepLearning IITD.CS.ML.ICML Rob2 10
JOB DeepLearning IITD.CS.ML.ICML Rob 10
JOB ImageProcessing IITD.CS.ML.ICML Carry 10
JOB Pipeline IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans IITD.CS.TH.SODA Carry 10

JOB DeepLearning11 IITD.CS.ML.ICML Rob 10
JOB ImageProcessing1 IITD.CS.ML.ICML Carry 10
JOB Pipeline1 IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans1 IITD.CS.TH.SODA Carry 10

JOB DeepLearning2 IITD.CS.ML.ICML Rob 10
JOB ImageProcessing2 IITD.CS.ML.ICML Carry 10
JOB Pipeline2 IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans2 IITD.CS.TH.SODA Carry 10

ADD IITD.CS.ML.ICML 60
JOB DeepLearning3 IITD.CS.ML.ICML Rob 10
JOB ImageProcessing3 IITD.CS.ML.ICML Carry 10
JOB Pipeline3 IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans3 IITD.CS.TH.SODA Carry 10

QUERY Kmeans

JOB DeepLearning4 IITD.CS.ML.ICML Rob 10
JOB ImageProcessing4 IITD.CS.ML.ICML Carry 10
JOB Pipeline4 IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans4 IITD.CS.TH.SODA Carry 10

QUERY Kmeans
```

Listing 15: Input specification

```
Creating user
Creating user
Creating user
Creating project
Creating project
Creating project
Creating job
Creating job
Creating job
Creating job
Running code
  Remaining jobs: 4
  Executing: DeepLearning from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 5
Execution cycle completed
Querying
Kmeans: NOT FINISHED
Querying
Doesnotexists: NO SUCH JOB
Running code
  Remaining jobs: 3
  Executing: ImageProcessing from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: Pipeline from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 90
Execution cycle completed
Creating job
No such project exists. IITD.CS.ML.ICM
Creating job
No such user exists: Rob2
Creating job
Creating job
Creating job
Creating job
Running code
  Remaining jobs: 5
  Executing: DeepLearning from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: ImageProcessing from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: Pipeline from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 80
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
```

```
  Remaining jobs: 6
  Executing: DeepLearning11 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: ImageProcessing1 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: Pipeline1 from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 70
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
  Remaining jobs: 7
  Executing: DeepLearning2 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: ImageProcessing2 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: Pipeline2 from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 60
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
  Remaining jobs: 15
  Executing: ImageProcessing from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 55
Execution cycle completed
Querying
Kmeans: NOT FINISHED
Running code
  Remaining jobs: 14
  Executing: Kmeans3 from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 90
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
```

```
  Remaining jobs: 17
  Executing: Pipeline3 from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 50
Execution cycle completed
Querying
Kmeans: NOT FINISHED
Running code
  Remaining jobs: 16
  Executing: Kmeans4 from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 80
System execution completed
Running code
  Remaining jobs: 15
  Executing: Kmeans1 from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 70
System execution completed
Running code
  Remaining jobs: 14
  Executing: ImageProcessing4 from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 45
System execution completed
Running code
  Remaining jobs: 13
  Executing: DeepLearning4 from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 35
System execution completed
Running code
  Remaining jobs: 12
  Executing: ImageProcessing3 from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 25
System execution completed
Running code
  Remaining jobs: 11
  Executing: Kmeans from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 60
System execution completed
Running code
  Remaining jobs: 10
  Executing: ImageProcessing2 from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 15
System execution completed
Running code
```

```
  Remaining jobs: 9
  Executing: Kmeans2 from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 50
System execution completed
Running code
  Remaining jobs: 8
  Executing: Kmeans from: IITD.CS.TH.SODA
  Project: IITD.CS.TH.SODA budget remaining: 40
System execution completed
Running code
  Remaining jobs: 7
  Executing: Pipeline4 from: IITD.CS.OS.ASPLOS
  Project: IITD.CS.OS.ASPLOS budget remaining: 40
System execution completed
Running code
  Remaining jobs: 6
  Executing: ImageProcessing from: IITD.CS.ML.ICML
  Project: IITD.CS.ML.ICML budget remaining: 5
System execution completed
Running code
  Remaining jobs: 5
  Executing: DeepLearning3 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: DeepLearning2 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: ImageProcessing1 from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: DeepLearning from: IITD.CS.ML.ICML
  Un-sufficient budget.
  Executing: DeepLearning11 from: IITD.CS.ML.ICML
  Un-sufficient budget.
System execution completed
--------------STATS---------------
Total jobs done: 16
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=70,
   name='Kmeans3'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=60,
   name='ImageProcessing'}
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=10,
   name='DeepLearning'}
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=120,
   name='DeepLearning4'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=110,
   name='ImageProcessing4'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=130,
   name='ImageProcessing3'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=150,
   name='ImageProcessing2'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=100,
   name='Kmeans1'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=140,
   name='Kmeans'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=160,
   name='Kmeans2'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=20,
   name='Pipeline'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=90,
   name='Kmeans4'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=50,
   name='Pipeline2'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=40,
   name='Pipeline1'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=80,
   name='Pipeline3'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=180,
   name='Pipeline4'}
------------------------
Unfinished jobs:
```

```
Job{user=’Rob’, project=’IITD.CS.ML.ICML’,
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name=’DeepLearning3’}
Job{user=’Rob’, project=’IITD.CS.ML.ICML’,
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name=’DeepLearning2’}
Job{user=’Carry’, project=’IITD.CS.ML.ICML’,
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name=’ImageProcessing1’}
Job{user=’Rob’, project=’IITD.CS.ML.ICML’,
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name=’DeepLearning’}
Job{user=’Rob’, project=’IITD.CS.ML.ICML’,
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name=’DeepLearning11’}
Total unfinished jobs: 5
--------------STATS DONE----------------
```

Listing 16: Output for INP in Listing 15

6 Submission instructions

As always, you need to create all your .java files in a directory named *src*, compress this directory to zip format and rename the zip file in the format entrynumber_assignment3.zip. For example, if your entry number is `2012CSZ8019`, the zip file should be named `2012CSZ8019_assignment4.zip`. Then you need to convert this zip file to base64 format as follows and submit the .b64 file on Moodle.

```
base64 entrynumber_assignment4.zip > entrynumber_assignment4.zip.b64
```

Inside the src directory, at the minimum you need to have a `README` and a file named assignment4.java. In the `README`, you need to report the time complexities of various operations for both the implementations. You should also report any interesting findings based on your experiments with the two implementations.

Please note that we will run MOSS on the submitted code. Anyone found with a copied code, either from Internet or from another student, will be dealt as per the class policy.

7 FAQ

References

[1] Painting nodes black with red-black trees - basecs - medium. https://medium.com/basecs/painting-nodes-black-with-red-black-trees-60eacb2be9a5. (Accessed on 09/10/2019).

[2] Trie — (insert and search) - geeksforgeeks. https://www.geeksforgeeks.org/trie-insert-and-search/. (Accessed on 09/12/2019).