COL106: Assignment 4

Trie, Red-Black tree and Priority queue

Updated: October 14, 2019

1 Fixes

- Please note some changes in the output format for Trie.
- **Allowed imports:** List, Stack and Queue.

Logistics:
Release date: September 13, 2019
Submission deadline: ~~30^{th} September, 23:55~~ 2^{nd} October, 23:55
Total marks: 5
PDF Version: Assignment 4 PDF
FAQ: See Section 8
Code can be downloaded from here : Download code
Changes:

- Download the new Makefile. Replace the current Makefile in the src folder with this one. This takes care of the file encoding issues while comparing.
- Constant files, the files that you are NOT suppose to change can be found here: Download constant files. Just replace these files in their respective directories. You can make changes to other files as per your requirements.

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.

2 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 a penalty or no marks for that particular component.

2.1 Code skeleton

You are provided with the skeleton of the code. This contains the interfaces and other relevant information. Your task is to implement these 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

2.1.1 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.

3 Trie [1 Mark]

Trie is an efficient information reTrieval data structure. Using Trie, search complexities can be brought to optimal limit (key length) [3].
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.

```
1 package Trie;
2 public class Person {
3     public Person(String name, String phone_number) {
4     }
```

```
    public String getName() {
        return "";
    }
}
```

Listing 1: Person class.

3.1 Interface

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

```
package Trie;
/**
 * DO NOT EDIT THIS FILE.
 */
public interface TrieInterface<T> {
    /**
     * @param word  Word to be input in the Trie
     * @param value Associated value of the word
     * @return Success or failure
     */
    boolean insert(String word, T value);
    /**
     * @param word Search for this word, Case-Sensitive
     * @return Returns the Trienode associated if the word is
       found else NULL
     */
    TrieNode<T> search(String word);
    /**
     *
     * @param prefix Search a particular prefix
     * @return Returns the last Trienode associated with the
       prefix. Eg: If PARIS and PARROT is in the Tries,
       searching for PAR, returns the trienode of first R
     */
    TrieNode<T> startsWith(String prefix);
    /**
     *
     * @param trieNode Prints all the possible word possible
       from this Trienode
     *                 Eg: PAR and PARIS,
       printTrie(startWith("PAR")) should print PARIS and
```

```
         PARROT i.e all the words with suffix PAR
     */
    void printTrie(TrieNode trieNode);
    /**
     *
     * @param word Delete a word from the Trie
     * @return Success or Failure
     */
    boolean delete(String word);
    /**
     * Print the complete Trie
     */
    void print();
    /**
     * Print a specific level of the Trie.
     *
     * @param level
     */
    void printLevel(int level);
}
```

Listing 2: Interface specifications for Trie.

3.2 Input specifications

Commands:

1. INSERT: It takes a Person name and phone number (in next line) 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 prefix is the entered String. *Printing is done in a lexicographical order.*
5. PRINTLEVEL: Print the specified level *in lexicographical order separated by comma and DO NOT print spaces.*
6. PRINT: Print all the *LEVELS* of the trie. The print format same as that of PRINTLEVEL.

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
DELETE
Dishant Goyal
```

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

Listing 4: Ouput for Trie.

4 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 [2]. In this part, the basic operations on a Red-Black tree, insert and search will be tested. Note: you are not required to implement the *delete* feature. You will be given an input file, whose format is listed in Section 4.2. A sample output for the input command given in Section 4.2 is shown in 7
In this case also you will implement a telephone directory, with an extra feature that a person can have multiple numbers.

4.1 Specifications

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

```
1  package RedBlack;
2  public interface RBTreeInterface<T extends Comparable, E> {
3      /**
4       * Insert and element using the "key" as the key and the
           corresponding value.
5       * Please note that value is a generic type and it can be
           anything.
6       *
7       * @param key
8       * @param value
9       */
10     void insert(T key, E value);
11     /**
12      * Search using the key.
13      *
14      * @param key
15      * @return
16      */
17     RedBlackNode<T, E> search(T key);
18 }
```

Listing 5: Interface for Red-Black tree.

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 /appended to that key. This can be seen in the Listing 6.

4.2 Input specifications

Commands:

1. INSERT: Insert a Person into the tree.
2. SEARCH: Searches for a person 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
16 Dishant Kumar, +91987654321
17 INSERT
18 Dishant Gupta, +91987654321
19 SEARCH
20 Dishant Goyal
21 SEARCH
22 Sandeep
```

Listing 6: Input for RedBlack Tree.

Expected Output (ignore the line numbers):

```
1  Inserting: Diljeet Singh
2  Inserting: Bhavesh Kumar
3  Inserting: Chayan Malhotra
4  Inserting: Ekta Mittal
5  Inserting: Farhan Khan
6  Inserting: Dishant Goyal
7  Inserting: Dishant Goyal
8  Inserting: Dishant Kumar
9  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.

5 Priority queues [1 Mark]

In this part you will be working with a *priority queue*. Specifically, you will be implementing a *max-heap* which is an implementation of priority queue.
You will need to implement a *marks scoring system* 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.

5.1 Specifications

```java
package PriorityQueue;
/**
 * DO NOT EDIT
 *
 * @param <T>
 */
public interface PriorityQueueInterface<T extends Comparable> {
    /**
     * @param element Insert and element to the Priority Queue
     */
    void insert(T element);
    /**
     * Extract the current maximum element from the Queue
        (assuming a max heap).
     * @return
     */
    T extractMax();
}
```

Listing 8: Interface for PriorityQueue.

Commands

1. INSERT
   name marks: Insert the student in the tree. Student name and marks are give in the next line. Students name will be unique.

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):

```
1  INSERT
2  Diljeet Singh, 10
3  INSERT
4  Bhavesh Kumar, 100
5  INSERT
6  Dishant Kumar, 67
7  EXTRACTMAX
8  EXTRACTMAX
9  EXTRACTMAX
10 EXTRACTMAX
```

Listing 9: Input for PriorityQueue.

Expected Output (ignore the line numbers):

```
1 Inserting: Diljeet Singh
2 Inserting: Bhavesh Kumar
3 Inserting: Dishant Kumar
4 Student{name='Bhavesh Kumar', marks=100}
5 Student{name='Dishant Kumar', marks=67}
6 Student{name='Diljeet Singh', marks=10}
7 Heap is empty.
```

Listing 10: Output for PriorityQueue.

6 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 (Project management). 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).

```
1 package ProjectManagement;
2 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`.

6.1 Specifications

The main component in this part of the assignment is a *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 be executed 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.

6.2 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.

6.3 Scheduler specifications

The scheduler will execute a single job whenever it will encounter an empty line in the input specifications. After the end of the INP (input file) file, scheduler will continue to execute jobs till there are jobs left that can be executed.
Each time the scheduler wants to execute 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 it executes the job. 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 from the max-heap (where jobs are stored) and try to execute this.
6. A scheduler will return in following cases:
   - It successfully executed a single 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 (input file) 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;
/**
 * DO NOT MODIFY
 */
public interface SchedulerInterface {
    /**
     * @param cmd Handles Project creation. Input is the
       command from INP1 file in array format (use space to
       split it)
     */
    void handle_project(String[] cmd);
    /**
     * @param cmd Handles Job creation. Input is the command
       from INP1 file in array format (use space to split it)
     */
    void handle_job(String[] cmd);
    /**
     * @param name Handles user creation
     */
    void handle_user(String name);
    /**
     * Returns status of a job
     *
     * @param key
     */
    void handle_query(String key);
    /**
     * Next cycle, is executed whenever an empty line is found.
     */
    void handle_empty_line();
    /**
     * Executed as a thread to server a job.
     */
    void schedule();
    /**
     * Add budget to a project Input is the command from INP1
       file in array format (use space to split it)
```

```
34          *
35          * @param cmd
36          */
37         void handle_add(String[] cmd);
38         /**
39          * If there are no lines in the input commands, but there
               are jobs which can be executed, let the system run till
               there are no jobs left (which can be run).
40          */
41         void run_to_completion();
42         /**
43          * After execution is done, print the stats of teh system
44          */
45         void print_stats();
46     }
```

Listing 14: Interface specification

```
1  USER Rob
2  USER Harry
3  USER Carry
4  PROJECT IITD.CS.ML.ICML 10 15
5  PROJECT IITD.CS.OS.ASPLOS 9 100
6  PROJECT IITD.CS.TH.SODA 8 100
7  JOB DeepLearning IITD.CS.ML.ICML Rob 10
8  JOB ImageProcessing IITD.CS.ML.ICML Carry 10
9  JOB Pipeline IITD.CS.OS.ASPLOS Harry 10
10 JOB Kmeans IITD.CS.TH.SODA Carry 10
11
12 QUERY Kmeans
13 QUERY Doesnotexists
14
15 JOB DeepLearningNoProject IITD.CS.ML.ICM Rob 10
16 JOB DeepLearningNoUser IITD.CS.ML.ICML Rob2 10
17
18 JOB DeepLearning1 IITD.CS.ML.ICML Rob 10
19 JOB ImageProcessing1 IITD.CS.ML.ICML Carry 10
20 JOB Pipeline1 IITD.CS.OS.ASPLOS Harry 10
21 JOB Kmeans1 IITD.CS.TH.SODA Carry 10
22
23 JOB DeepLearning2 IITD.CS.ML.ICML Rob 10
24 JOB ImageProcessing2 IITD.CS.ML.ICML Carry 10
25 JOB Pipeline2 IITD.CS.OS.ASPLOS Harry 10
```

```
JOB Kmeans3 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

JOB DeepLearning5 IITD.CS.ML.ICML Rob 10
JOB ImageProcessing5 IITD.CS.ML.ICML Carry 10
JOB Pipeline5 IITD.CS.OS.ASPLOS Harry 10
JOB Kmeans5 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
Running code
Remaining jobs: 1
Executing: Kmeans 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: 4
Executing: DeepLearning1 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: 80
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
Remaining jobs: 5
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: 70
Execution cycle completed
```

```
ADDING Budget
Creating job
Creating job
Creating job
Creating job
Running code
Remaining jobs: 11
Executing: ImageProcessing from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 55
Execution cycle completed
Querying
Kmeans: COMPLETED
Running code
Remaining jobs: 10
Executing: DeepLearning1 from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 45
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
Remaining jobs: 13
Executing: ImageProcessing1 from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 35
Execution cycle completed
Creating job
Creating job
Creating job
Creating job
Running code
Remaining jobs: 16
Executing: DeepLearning2 from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 25
Execution cycle completed
Querying
Kmeans: COMPLETED
Running code
Remaining jobs: 15
Executing: ImageProcessing2 from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 15
System execution completed
```

```
Running code
Remaining jobs: 14
Executing: DeepLearning3 from: IITD.CS.ML.ICML
Project: IITD.CS.ML.ICML budget remaining: 5
System execution completed
Running code
Remaining jobs: 13
Executing: ImageProcessing3 from: IITD.CS.ML.ICML
Un-sufficient budget.
Executing: DeepLearning4 from: IITD.CS.ML.ICML
Un-sufficient budget.
Executing: ImageProcessing4 from: IITD.CS.ML.ICML
Un-sufficient budget.
Executing: DeepLearning5 from: IITD.CS.ML.ICML
Un-sufficient budget.
Executing: ImageProcessing5 from: IITD.CS.ML.ICML
Un-sufficient budget.
Executing: Pipeline3 from: IITD.CS.OS.ASPLOS
Project: IITD.CS.OS.ASPLOS budget remaining: 60
System execution completed
Running code
Remaining jobs: 7
Executing: Pipeline4 from: IITD.CS.OS.ASPLOS
Project: IITD.CS.OS.ASPLOS budget remaining: 50
System execution completed
Running code
Remaining jobs: 6
Executing: Pipeline5 from: IITD.CS.OS.ASPLOS
Project: IITD.CS.OS.ASPLOS budget remaining: 40
System execution completed
Running code
Remaining jobs: 5
Executing: Kmeans1 from: IITD.CS.TH.SODA
Project: IITD.CS.TH.SODA budget remaining: 80
System execution completed
Running code
Remaining jobs: 4
Executing: Kmeans3 from: IITD.CS.TH.SODA
Project: IITD.CS.TH.SODA budget remaining: 70
System execution completed
Running code
Remaining jobs: 3
```

```
Executing: Kmeans3 from: IITD.CS.TH.SODA
Project: IITD.CS.TH.SODA budget remaining: 60
System execution completed
Running code
Remaining jobs: 2
Executing: Kmeans4 from: IITD.CS.TH.SODA
Project: IITD.CS.TH.SODA budget remaining: 50
System execution completed
Running code
Remaining jobs: 1
Executing: Kmeans5 from: IITD.CS.TH.SODA
Project: IITD.CS.TH.SODA budget remaining: 40
System execution completed
--------------STATS---------------
Total jobs done: 19
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=10,
   name='DeepLearning'}
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=30,
   name='Kmeans'}
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=50,
   name='Pipeline2'}
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=70,
   name='DeepLearning1'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=80,
   name='ImageProcessing1'}
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=90,
   name='DeepLearning2'}
```

```
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=100,
   name='ImageProcessing2'}
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=COMPLETED, execution_time=10, end_time=110,
   name='DeepLearning3'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=120,
   name='Pipeline3'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=130,
   name='Pipeline4'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
   jobstatus=COMPLETED, execution_time=10, end_time=140,
   name='Pipeline5'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=150,
   name='Kmeans1'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=160,
   name='Kmeans3'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=170,
   name='Kmeans3'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=180,
   name='Kmeans4'}
Job{user='Carry', project='IITD.CS.TH.SODA',
   jobstatus=COMPLETED, execution_time=10, end_time=190,
   name='Kmeans5'}
------------------------
Unfinished jobs:
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name='ImageProcessing3'}
Job{user='Rob', project='IITD.CS.ML.ICML',
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name='DeepLearning4'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name='ImageProcessing4'}
Job{user='Rob', project='IITD.CS.ML.ICML',
```

```
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name='DeepLearning5'}
Job{user='Carry', project='IITD.CS.ML.ICML',
   jobstatus=REQUESTED, execution_time=10, end_time=null,
   name='ImageProcessing5'}
Total unfinished jobs: 5
--------------STATS DONE---------------
```

Listing 16: Output for INP in Listing 15

7 Submission instructions

As always compress *src* directory to zip format and rename the zip file in the format entrynumber_assignment4.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
```

Folder structure: Inside the src directory, you need to have a `README.pdf` (case sensitive) and your solution (exactly following the folder structure that of the code provided.). Please do not rename the existing directories. 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.

Grading: While grading we will replace the *driver code* and the *interface* code with the original files before the compilation. So please ensure that your code compiles and run correctly with the original driver and interface files.

MOSS: 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.

8 FAQ

- See some fixes here: 1
- Project Management: In what data-structure projects are stored?
  Ans: Trie
- Trie: Match: match the search term with *pre-fix* of entries.

- Lists and its subclasses (ArrayList, LinkedList) etc. are allowed.
- Maps (e.g. HashMap) are also allowed.
- You need to override the *toString()* method in classes to print in a particular format.
- Printing order in Trie: Lexicographical order. DO NOT print spaces (which is present in the name). You need to store spaces in the Trie, otherwise MATCH command will print first-name and second-name together(without a space) and it will not match the output.
- Trie: Names can contain any character whose ASCII code is between **32-126**, see an ASCII table here: [1].
- Trie: If a particular entry is not found, print `NOT FOUND`.
- Build issues: Please use the `Makefile` provided.
- Only List, Stack, Queue and *Maps* arze allowed to be imported.
- There are some encoding issue in file compare. We are working on it.
- Extract-Max in Priority Queue should follow FIFO (if the priority of two objects is same).
- Casting from and to Object is allowed.
- Please adhere to return types of the function. It will be used in the driver code.
- You can use Iterator.
- Print: Trie. Last level is empty, which represents the end of the Trie.
- Question: Adding budget to a project, what to do with its unfinished (but already tried) jobs?
  Answer: https://piazza.com/class/jyic9aa2xyb34g?cid=649
- Project insertion and retrieval doubt: We are inserting project in a trie and retrieving from a RB tree, how?
  Answer:When a new Project is created, it is stored in a Trie. RB-Tree is used as a NOT_READY queue. To store the jobs which cannot be executed because there is not enough budget left. As mentioned in the RB-Tree implementation, using a single key we can store multiple objects.
- Files NOT to be edited: Please make no changes to the Driver code for the Trie, RedBlack and PriorityQueue. And please do not change any of the interface files.
- Size of Max-heap: Upper bound on the size of the heap: 10000. You can use lists.

- Project management, empty_line, run_to_completion:

```
"1. How is schedule() different from handle_empty_line()?"

handle_empty_line is there so do dome pre-processing before calling schedule.
If you feel like this is not required, use this:

public void handle_empty_line() {
schedule();
}


"2. What is the use of run()? What is schedule() doing inside that?
 If it is for completing the pending jobs at the end, then why
 run_to_completion()?"

The driver uses a thread to execute the jobs. schedule() contains
the logic to check the budget, then either execute the code or move
 it to a NON_READY queue.
As can be seen from the driver code, run_to_completion() is used
 when we are done processing the INPUT FILE (i.e. no more
commands are left to process).
```

- What to return in RBTree when key is not found?
  A RedBlackNode with NULL key and NULL values.

- If a key has multiple values associates with it, then the output of search query for that key should have values printed in lexicographic order or FIFO.

- Priority order:
  FIFO order comes after priority.
  If there are two jobs with a different priority, then you take the job which has the higher priority, irrespective of who came first.
  FIFO has to be maintained only if their priorities are same.

- *print_stats* specification: After the system is done with executing all the commands, it tries to executed as many job it can. Once it is done with that, it prints the stats of the system and exists. The order of printing:

  1. Print all the executed job first in the order which they were executed. The job executed first should be printed first.

2. After this print all the un-finished job. At this point, the *waiting-queue* or the MAX-HEAP used by the scheduler to find the job should be empty. Jobs must have been completed or will be in the *NOT_READY* queue.
   Printing order:
   - Process projects based on their priority.
   - Processing a project means, printing all the jobs belonging to that project.
   - Project priority is determined in the same way as that of the jobs. The project having a higher value of the "priority" has the higher priority. If the value of the priorities is same, then the project created first has the higher priority.
   - With-in a project: Print jobs based on their priority. See `Priority order` in this FAQ to decide priority of a job.

References

[1] Ascii table - ascii character codes and html, octal, hex and decimal chart conversion. `http://www.asciitable.com/`. (Accessed on 09/15/2019).

[2] 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).

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