

COL106: Assignment 4

Trie, Red-Black tree and Priority queue

Updated: September 18, 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: 30th September, 23:55

Total marks: 5

PDF Version: Assignment 4 PDF

FAQ: See Section 8

Code can be downloaded from here: Download code

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 an *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 result 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     }
5     public String getName() {
6         return "";
7     }
8 }
```

Listing 1: Person class.

3.1 Interface

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

```
1 package Trie;
2
3 public interface TrieInterface<T> {
4
5     boolean insert(String word, T value);
6
7     TrieNode<T> search(String word);
8
9     TrieNode<T> startsWith(String prefix);
10
11     void printTrie(TrieNode trieNode);
12
13     boolean delete(String word);
14
15     void print();
16
17     void printLevel(int level);
18 }
```

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:

```
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 Kumar, +91987654321
15 INSERT
16 Dishant Gupta, +91987654321
17 SEARCH
18 Dishant Goyal
19 MATCH Di
20 MATCH di
21 DELETE
22 Dishant Goyal
23 SEARCH
24 Dishant Goyal
25 MATCH SK
26 PRINTLEVEL 2
27 PRINT
28 DELETE
29 Dishant Goyal
```

Listing 3: Input for Trie.

Expected Output file:

```
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 Kumar
8 Inserting: Dishant Gupta
```

```

9 Searching: Dishant Goyal
10 FOUND
11 [Name: Dishant Goyal, Phone=+91987654321]
12 Matching: Di
13 MATCHED:
14 [Name: Diljeet Singh, Phone=+91987654321]
15 [Name: Dishant Kumar, Phone=+91987654321]
16 [Name: Dishant Goyal, Phone=+91987654321]
17 [Name: Dishant Gupta, Phone=+91987654321]
18 Matching: di
19 NOT FOUND
20 Deleting: Dishant Goyal
21 DELETED
22 Searching: Dishant Goyal
23 NOT FOUND
24 Matching: SK
25 NOT FOUND
26 Level 2: a,h,h,i,k
27 -----
28 Printing Trie
29 Level 1: B,C,D,E,F
30 Level 2: a,h,h,i,k
31 Level 3: a,a,l,r,s,t
32 Level 4: a,h,h,j,v,y
33 Level 5: a,a,a,e,e
34 Level 6: M,e,n,n,n,s
35 Level 7: h,i,t,t
36 Level 8: K,M,t
37 Level 9: G,K,K,S,a,h,t
38 Level 10: a,a,i,l,u,u,u
39 Level 11: h,l,m,m,n,n,p
40 Level 12: a,a,g,o,t
41 Level 13: a,h,r,r,t
42 Level 14: r
43 Level 15: a
44 Level 16:
45 -----
46 Deleting: Dishant Goyal
47 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 RBTInterface<T extends Comparable, E> {
3
4     void insert(T key, E value);
5
6     RedBlackNode<T, E> search(T key);
7 }
```

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 /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
10 Searching for: Dishant Goyal
11 [Name: Dishant Goyal , Phone=+91987654321]
12 [Name: Dishant Goyal , Phone=+91999999999]
13 Searching for: Sandeep
14 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

```
1 package PriorityQueue;  
2  
3 public interface PriorityQueueInterface<T extends Comparable> {  
4  
5     void insert(T element);  
6  
7     T extractMax();  
8 }
```

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 Sandeep Kumar , 100  
5 INSERT  
6 Bhavesh Kumar , 100  
7 INSERT  
8 Dishant Kumar , 67  
9 EXTRACTMAX  
10 EXTRACTMAX  
11 EXTRACTMAX  
12 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 {
3 }

```

Listing 11: Project class

2. User:

```

1 package ProjectManagement;
2 public class User implements Comparable<User> {
3     @Override
4     public int compareTo(User user) {
5         return 0;
6     }
7 }

```

Listing 12: User class

3. Job:

```

1 package ProjectManagement;
2 public class Job implements Comparable<Job> {
3     @Override
4     public int compareTo(Job job) {
5         return 0;
6     }
7 }

```

```

6     }
7 }

```

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.

```

1 package ProjectManagement;
2 public interface SchedulerInterface {
3     void run_to_completion();
4     void handle_project(String[] cmd);
5     void handle_job(String[] cmd);
6     void handle_user(String name);
7     void handle_query(String key);
8     void handle_empty_line();
9     void handle_add(String[] cmd);
10    void print_stats();
11    void schedule();
12 }

```

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 DeepLearning IITD.CS.ML.ICM Rob 10
16 JOB DeepLearning IITD.CS.ML.ICML Rob2 10
17 JOB DeepLearning IITD.CS.ML.ICML Rob 10
18 JOB ImageProcessing IITD.CS.ML.ICML Carry 10
19 JOB Pipeline IITD.CS.OS.ASPLOS Harry 10
20 JOB Kmeans IITD.CS.TH.SODA Carry 10
21
22 JOB DeepLearning11 IITD.CS.ML.ICML Rob 10
23 JOB ImageProcessing1 IITD.CS.ML.ICML Carry 10
24 JOB Pipeline1 IITD.CS.OS.ASPLOS Harry 10
25 JOB Kmeans1 IITD.CS.TH.SODA Carry 10
26
27 JOB DeepLearning2 IITD.CS.ML.ICML Rob 10
28 JOB ImageProcessing2 IITD.CS.ML.ICML Carry 10
29 JOB Pipeline2 IITD.CS.OS.ASPLOS Harry 10
30 JOB Kmeans2 IITD.CS.TH.SODA Carry 10
31
32 ADD IITD.CS.ML.ICML 60
33 JOB DeepLearning3 IITD.CS.ML.ICML Rob 10
34 JOB ImageProcessing3 IITD.CS.ML.ICML Carry 10
35 JOB Pipeline3 IITD.CS.OS.ASPLOS Harry 10
36 JOB Kmeans3 IITD.CS.TH.SODA Carry 10
37
38 QUERY Kmeans
39
40 JOB DeepLearning4 IITD.CS.ML.ICML Rob 10
41 JOB ImageProcessing4 IITD.CS.ML.ICML Carry 10

```

```

42 JOB Pipeline4 IITD.CS.OS.ASPLOS Harry 10
43 JOB Kmeans4 IITD.CS.TH.SODA Carry 10
44
45 QUERY Kmeans

```

Listing 15: Input specification

```

1  Creating user
2  Creating user
3  Creating user
4  Creating project
5  Creating project
6  Creating project
7  Creating job
8  Creating job
9  Creating job
10 Creating job
11 Running code
12     Remaining jobs: 4
13     Executing: DeepLearning from: IITD.CS.ML.ICML
14     Project: IITD.CS.ML.ICML budget remaining: 5
15 Execution cycle completed
16 Querying
17 Kmeans: NOT FINISHED
18 Querying
19 Doesnotexists: NO SUCH JOB
20 Running code
21     Remaining jobs: 3
22     Executing: ImageProcessing from: IITD.CS.ML.ICML
23     Un-sufficient budget.
24     Executing: Pipeline from: IITD.CS.OS.ASPLOS
25     Project: IITD.CS.OS.ASPLOS budget remaining: 90
26 Execution cycle completed
27 Creating job
28 No such project exists. IITD.CS.ML.ICM
29 Creating job
30 No such user exists: Rob2
31 Creating job
32 Creating job
33 Creating job
34 Creating job
35 Running code
36     Remaining jobs: 5

```

```

37   Executing: DeepLearning from: IITD.CS.ML.ICML
38   Un-sufficient budget.
39   Executing: ImageProcessing from: IITD.CS.ML.ICML
40   Un-sufficient budget.
41   Executing: Pipeline from: IITD.CS.OS.ASPLOS
42   Project: IITD.CS.OS.ASPLOS budget remaining: 80
43 Execution cycle completed
44 Creating job
45 Creating job
46 Creating job
47 Creating job
48 Running code
49   Remaining jobs: 6
50   Executing: DeepLearning11 from: IITD.CS.ML.ICML
51   Un-sufficient budget.
52   Executing: ImageProcessing1 from: IITD.CS.ML.ICML
53   Un-sufficient budget.
54   Executing: Pipeline1 from: IITD.CS.OS.ASPLOS
55   Project: IITD.CS.OS.ASPLOS budget remaining: 70
56 Execution cycle completed
57 Creating job
58 Creating job
59 Creating job
60 Creating job
61 Running code
62   Remaining jobs: 7
63   Executing: DeepLearning2 from: IITD.CS.ML.ICML
64   Un-sufficient budget.
65   Executing: ImageProcessing2 from: IITD.CS.ML.ICML
66   Un-sufficient budget.
67   Executing: Pipeline2 from: IITD.CS.OS.ASPLOS
68   Project: IITD.CS.OS.ASPLOS budget remaining: 60
69 Execution cycle completed
70 Creating job
71 Creating job
72 Creating job
73 Creating job
74 Running code
75   Remaining jobs: 15
76   Executing: ImageProcessing from: IITD.CS.ML.ICML
77   Project: IITD.CS.ML.ICML budget remaining: 55
78 Execution cycle completed

```

```
79 Querying
80 Kmeans: NOT FINISHED
81 Running code
82   Remaining jobs: 14
83   Executing: Kmeans3 from: IITD.CS.TH.SODA
84   Project: IITD.CS.TH.SODA budget remaining: 90
85 Execution cycle completed
86 Creating job
87 Creating job
88 Creating job
89 Creating job
90 Running code
91   Remaining jobs: 17
92   Executing: Pipeline3 from: IITD.CS.OS.ASPLOS
93   Project: IITD.CS.OS.ASPLOS budget remaining: 50
94 Execution cycle completed
95 Querying
96 Kmeans: NOT FINISHED
97 Running code
98   Remaining jobs: 16
99   Executing: Kmeans4 from: IITD.CS.TH.SODA
100  Project: IITD.CS.TH.SODA budget remaining: 80
101 System execution completed
102 Running code
103   Remaining jobs: 15
104   Executing: Kmeans1 from: IITD.CS.TH.SODA
105   Project: IITD.CS.TH.SODA budget remaining: 70
106 System execution completed
107 Running code
108   Remaining jobs: 14
109   Executing: ImageProcessing4 from: IITD.CS.ML.ICML
110   Project: IITD.CS.ML.ICML budget remaining: 45
111 System execution completed
112 Running code
113   Remaining jobs: 13
114   Executing: DeepLearning4 from: IITD.CS.ML.ICML
115   Project: IITD.CS.ML.ICML budget remaining: 35
116 System execution completed
117 Running code
118   Remaining jobs: 12
119   Executing: ImageProcessing3 from: IITD.CS.ML.ICML
120   Project: IITD.CS.ML.ICML budget remaining: 25
```

```
121 System execution completed
122 Running code
123     Remaining jobs: 11
124     Executing: Kmeans from: IITD.CS.TH.SODA
125     Project: IITD.CS.TH.SODA budget remaining: 60
126 System execution completed
127 Running code
128     Remaining jobs: 10
129     Executing: ImageProcessing2 from: IITD.CS.ML.ICML
130     Project: IITD.CS.ML.ICML budget remaining: 15
131 System execution completed
132 Running code
133     Remaining jobs: 9
134     Executing: Kmeans2 from: IITD.CS.TH.SODA
135     Project: IITD.CS.TH.SODA budget remaining: 50
136 System execution completed
137 Running code
138     Remaining jobs: 8
139     Executing: Kmeans from: IITD.CS.TH.SODA
140     Project: IITD.CS.TH.SODA budget remaining: 40
141 System execution completed
142 Running code
143     Remaining jobs: 7
144     Executing: Pipeline4 from: IITD.CS.OS.ASPLOS
145     Project: IITD.CS.OS.ASPLOS budget remaining: 40
146 System execution completed
147 Running code
148     Remaining jobs: 6
149     Executing: ImageProcessing from: IITD.CS.ML.ICML
150     Project: IITD.CS.ML.ICML budget remaining: 5
151 System execution completed
152 Running code
153     Remaining jobs: 5
154     Executing: DeepLearning3 from: IITD.CS.ML.ICML
155     Un-sufficient budget.
156     Executing: DeepLearning2 from: IITD.CS.ML.ICML
157     Un-sufficient budget.
158     Executing: ImageProcessing1 from: IITD.CS.ML.ICML
159     Un-sufficient budget.
160     Executing: DeepLearning from: IITD.CS.ML.ICML
161     Un-sufficient budget.
162     Executing: DeepLearning11 from: IITD.CS.ML.ICML
```



```

163     Un-sufficient budget.
164 System execution completed
165 -----STATS-----
166 Total jobs done: 16
167 Job{user='Carry', project='IITD.CS.TH.SODA',
    jobstatus=COMPLETED, execution_time=10, end_time=70,
    name='Kmeans3'}
168 Job{user='Carry', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=60,
    name='ImageProcessing'}
169 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=10,
    name='DeepLearning'}
170 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=120,
    name='DeepLearning4'}
171 Job{user='Carry', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=110,
    name='ImageProcessing4'}
172 Job{user='Carry', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=130,
    name='ImageProcessing3'}
173 Job{user='Carry', project='IITD.CS.ML.ICML',
    jobstatus=COMPLETED, execution_time=10, end_time=150,
    name='ImageProcessing2'}
174 Job{user='Carry', project='IITD.CS.TH.SODA',
    jobstatus=COMPLETED, execution_time=10, end_time=100,
    name='Kmeans1'}
175 Job{user='Carry', project='IITD.CS.TH.SODA',
    jobstatus=COMPLETED, execution_time=10, end_time=140,
    name='Kmeans'}
176 Job{user='Carry', project='IITD.CS.TH.SODA',
    jobstatus=COMPLETED, execution_time=10, end_time=160,
    name='Kmeans2'}
177 Job{user='Harry', project='IITD.CS.OS.ASPLOS',
    jobstatus=COMPLETED, execution_time=10, end_time=20,
    name='Pipeline'}
178 Job{user='Carry', project='IITD.CS.TH.SODA',
    jobstatus=COMPLETED, execution_time=10, end_time=90,
    name='Kmeans4'}
179 Job{user='Harry', project='IITD.CS.OS.ASPLOS',
    jobstatus=COMPLETED, execution_time=10, end_time=50,

```

```

180     name='Pipeline2'}
Job{user='Harry', project='IITD.CS.OS.ASPLOS',
    jobstatus=COMPLETED, execution_time=10, end_time=40,
    name='Pipeline1'}
181 Job{user='Harry', project='IITD.CS.OS.ASPLOS',
    jobstatus=COMPLETED, execution_time=10, end_time=80,
    name='Pipeline3'}
182 Job{user='Harry', project='IITD.CS.OS.ASPLOS',
    jobstatus=COMPLETED, execution_time=10, end_time=180,
    name='Pipeline4'}
183 -----
184 Unfinished jobs:
185 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=REQUESTED, execution_time=10, end_time=null,
    name='DeepLearning3'}
186 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=REQUESTED, execution_time=10, end_time=null,
    name='DeepLearning2'}
187 Job{user='Carry', project='IITD.CS.ML.ICML',
    jobstatus=REQUESTED, execution_time=10, end_time=null,
    name='ImageProcessing1'}
188 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=REQUESTED, execution_time=10, end_time=null,
    name='DeepLearning'}
189 Job{user='Rob', project='IITD.CS.ML.ICML',
    jobstatus=REQUESTED, execution_time=10, end_time=null,
    name='DeepLearning11'}
190 Total unfinished jobs: 5
191 -----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.
- Trie: Names can contain any character whose ASCII code is between 0-127, 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 and Queue are 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.
- 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.

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).