COL106 (SPRING, 2020)

ASSIGNMENT 3

# Handling Queries of an Election

# 1 Overall Objective

In this assignment you will have to work with *binary search tree*, *binary heap*. There are three principal components of this assignments: **BST**, **Heap**, and **Election**. Election is a sample of a real-life problem where you have to handle certain queries related to election, say *Vidhan Sabha Election* occuring at multiple states of India at the same time. You will be using *binary search tree* and *binary max heap* to handle those queries.

# 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** supposed 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. Remember you just have to output normally using *(System.out.print())* and your output will automatically be printed in the file. **Please ensure that you follow the proper formatting criteria, failing which your submission will not be evaluated and receive no marks for that component.**

# 3 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 is available in a .zip file named Assignment3.zip.

# 4 Part-I: BST

In this part, you need to implement a *Binary Search Tree (BST)* (**You do not need to balance it**). The basic operations on a BST, *insert*, *update*, and *delete* will be tested. The BST is supposed to store $< key, value >$ pairs where both the $key$ and the $value$ are of *generic* type. Later in the *Election* problem, you can use suitable instance of this BST to handle certain queries.

Note: For simplicity, you can assume that all $keys$ are *unique*, that is, a particular $key$ cannot be associated with multiple $valeus$. You can assume all *values* which are to be stored in the *BST* are *unique*.

## 4.1 Specifications:

You need to implement the *interface* specified below:

```
package ElectionBSTHeap ;
public interface BSTInterface <T extends Comparable , E>{
/**
* Insert an element using the "key" as the key and the
corresponding value .
* Please note that value is a generic type and it can be
anything .
*
* @param key
* @param value
*/
void insert ( T key , E value ) ;
 /**
 * Update to new value using the key.
 *
 * @param key
 * @param value
 */
 void update(T key, E value);
 /**
 *
 * Delete element using key
 * @param key
 */
 void delete ( T key ) ;
 /**
 *
 * Print the keys according to level order traversal of the BST
 *
 */
```

```
 void printBST () ;
}
```

### 4.2 Input and Output:

**Commands:**

- INSERT : Insert a $< key, value >$ pair into the BST.
- UPDATE : Update the $value$ corresponding to a particular $key$.
- DELETE : Delete element from BST with a given $key$.
- PRINT_BST : Print the $key$, $value$ pairs stored in the BST according to level-order traversal.

**Sample Input:** Note: Ignore the line numbers.

```
INSERT
1, 15
INSERT
2, 5
INSERT
3, 1
INSERT
4, 6
INSERT
5, 20
INSERT
6, 16
INSERT
7, 25
INSERT
8, 21
INSERT
9, 26
DELETE
5
PRINT_BST
UPDATE
9, 300
PRINT_BST
```

**Expected Output:** Note: Ignore the line numbers.

```
Inserting: 1, 15
Inserting: 2, 5
Inserting: 3, 1
Inserting: 4, 6
Inserting: 5, 20
Inserting: 6, 16
Inserting: 7, 25
Inserting: 8, 21
Inserting: 9, 26
Deleting element with key 5:
Printing BST in level order:
1, 15
2, 5
8, 21
3, 1
4, 6
6, 16
7, 25
9, 26
Updating key 9 to value 300:
Printing BST in level order:
1, 15
2, 5
8, 21
3, 1
4, 6
6, 16
7, 25
9, 300
```

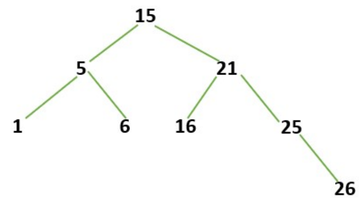


Figure 1: BST on first print command

For better understanding, look at the pictorial representation of the binary search tree (Fig:1) after the first print command (PRINT_BST) executes. Note that the picture contains only $values$.

## 5 Heap

In this part, you need to implement a *Binary Max Heap*. The basic operations on a Heap, *extractMax*, *insert*, *delete*, *increaseKey* will be tested. The Heap is supposed to store $< key, value >$ pairs where both the $key$ and the $value$ are of generic type. Later in the *Election* problem, you can use suitable instance of this Heap to handle certain queries.

**Note:** For simplicity, you can assume that $keys$ are *unique*, that is, a particular $key$ cannot be associated with multiple $valeus$. The binary max heap will be organized according to the $values$. You can also assume that all the *values* to be stored in the heap are *unique*.

*Binary heap* is an almost complete binary tree. Just make sure you follow the standard heap-filling constraints: Start filling level $i + 1$ only after level $i$ is completely filled. Fill a particular level $i$ of a *binary heap* from *left* to *right*.

### 5.1 Specifications:

You need to implement the *interface* shown below:

```
package Heap;
public interface HeapInterface <T extends Comparable, E>{
/**
* Insert an element using the "key" as the key and the
corresponding value into the binary max heap.
* The heap needs to be constructed w.r.t. value.
*
* @param key
* @param value
*/
void insert(T key , E value);
 /**
 * Remove the element having maximum value from the heap and return it.
 *
 */
E extractMax();
 /**
 *
 * Delete an element from heap using the given key
 * @param key
 */
 void delete(T key);
 /**
 *
 *  Update (increase) the value of a given "key" to the new "value"
 * @param key
 * @param value
 *
 */
 void increaseKey(T key, E value);
 /**
 *  Print all the keys stored in the heap according to their level-order
 */
 void printHeap();
}
```

### 5.2 Input and Output:

**Commands:**

- INSERT : insert a $< key, value >$ pair into the binary max heap.
- EXTRACT_MAX : extracts the element with maximum $value$ from the heap and return it. Note that EXTRACT_MAX also removes the maximum element from the heap.
- DELETE : delete an element from the heap with a given $key$.
- UPDATE : update the $value$ of a given $key$ in the heap.
- PRINT_HEAP : Print all the $key$, $value$ pairs stored in the heap according to their level order.

**Sample Input:**

```
INSERT
1, 100
INSERT
2, 10
INSERT
3, 30
INSERT
4, 50
INSERT
5, 150
INSERT
6, 1
INSERT
7, 3
PRINT_HEAP
DELETE
1
PRINT_HEAP
INSERT
8, 500
PRINT_HEAP
EXTRACT_MAX
PRINT_HEAP
UPDATE
7, 70
PRINT_HEAP
```

**Expected Output:**

```
Inserting: 1, 100
Inserting: 2, 10
Inserting: 3, 30
Inserting: 4, 50
Inserting: 5, 150
Inserting: 6, 1
Inserting: 7, 3
Printing heap in level order:
5, 150
1, 100
3, 30
2, 10
4, 50
6, 1
7, 3
Deleting element with key 1:
Printing heap in level order:
5, 150
4, 50
3, 30
2, 10
7, 3
6, 1
Inserting: 8, 500
Printing heap in level order:
8, 500
4, 50
5, 150
2, 10
7, 3
```

```
6, 1
3, 30
Extracting Max:
500
Printing heap in level order:
5, 150
4, 50
3, 30
2, 10
7, 3
6, 1
Updating key 7 to value 70:
Printing heap in level order:
5, 150
7, 70
3, 30
2, 10
4, 50
6, 1
```

## 6 Election

This is the last part of this assignment. In this part you need to use the previous data structures, *Heap* and *BST* to handle certain queries related to a particular instance of an Election process. This section will give you an impression of using standard data structures along with some additional logic to address a real-world scenario.

The Election problem is basically the problem of handling certain queries related to say, *Vidhan Sabha Election* occurring in multiple states of India at the same time. A *Candidate* is an *Object* to serve as a unit of information in this case. A *Candidate* object has six data members:

- *name* : name of the candidate
- *candID* : two candidates might have the same name. This *candID* is a unique identifier by which a candidate can be uniquely identified.
- *state* : The name of the state from which a candidate is fighting the election
- *district* : The name of the district of a particular state from which a candidate is fighting the election
- *constituency* : The name of the constituency of a particular state from which a candidate is fighting the election
- *party* : The name of the political party on behalf of which a candidate is fighting the election
- *votes* : The number of votes a particular candidate from a constituency in a state has scored at a particular instant

### 6.1 Specification:

You need to implement the following *interface*:

```java
package ElectionBSTHeap;

/**
 * DO NOT EDIT THIS FILE.
 */
public interface ElectionInterface<T> {
    /**
     * @param name,candID,state,district,constituency,party,votes to be input in the Election
     * @return Success or failure
     */
    void insert(String name, String candID, String state, String district, String constituency, String party, String votes);

    /**
     * @param inputs the votes for a given candidate
     */
    void updateVote(String name, String candID, String votes);
    /**
     * @param inputs the name of the constituency
     * prints the {name candID party} of top 3 (k=3) candidates in a constituency seperated by newlines.
     * example of output:
     * A 100 party1
```

```
     * B 200 party2
     * C 300 party3
     */
    void topkInConstituency(String constituency, String k);
    /**
     * @param inputs the name of the state
     * @return the name of the leading party of that state
     */
    void leadingPartyInState(String state);
    /**
     * @param inputs the name of the constituency for which suppose the EVM is suspected to be corrupted
     * delete all the candidates information for that constituency
     */
    void cancelVoteConstituency(String constituency);
    /**
     * @param inputs void
     * @return the name of the overall leading party across all the constituencies of all states
     */
    void leadingPartyOverall();
     /**
     * @param inputs the name of a party and a state
     * @return the %vote a particular party obtains in a given state
     */
    void voteShareInState(String party,String state);

    /**
     * @param inputs a state
     * Prints the <name, candID, state, district, constituency, party, votes> of all the candidates from your data structure (BST) according to level-order.
     * All the <name, candID, state, district, constituency, party, votes> should be separated by newlines.
     */
    void printElectionLevelOrder();
}
```

For the election problem, you need to employ some book-keeping for future references. You can use an instance of your BST for this purpose. Among all the above queries, there are three queries: *insert*, *updateVote*, *cancelVoteConstituency* for which you have to manage/modify the book-keeping data structure. The *printElectionState* query is there to check the intermediate states of this book-keeping data structure.

For other queries like *topkInConstituency*, you can use suitable instances of your *Heap*.

### 6.2 Input and Output:

**Commands:**

- INSERT : insert a candidate into your data structure with *name*, *candID*, *state*, *district*, *constituency*, *party*, and *votes* as associated information.
- UPDATE : Update votes obtained by a given candidate to a new value.
- TOP k IN CONSTITUENCY : Given a constituency, report the top k candidates ($< name, candID, party >$) of that constituency in decreasing order of $votes$. If you find that no. of candidates is less than k then return all the candidates in decreasing order of their obtained $votes$.
- LEADING PARTY IN STATE : Given a state, report the name of the political party scoring maximum votes in that state.
- DISCARD ALL VOTES IN CONSTITUENCY : Given a constituency, delete all the candidates belonging to that constituency from the data structure.
- GLOBAL LEADING PARTY : Report the name of the political party which received the maximum vote across all constituencies of all states.
- VOTE SHARES : Given the name of a political party and a state, report the % vote share of that party from that state (precise upto 1 decimal point).
- PRINT : Print all the $< name, candID, state, district, constituency, party, votes >$ pairs from your book-keeping data structure BST according to level-order.

The BST you have implemented in part-I of this assignment accepts *generic* types for both $key$ and $val$. You can use suitable instance of your BST here in *Election* with $candID$ as $key$, and other fields as $value$. Note that your book-keeping BST for *Election* is to be ordered based only on the obtained $votes$ of the candidates.

**Sample Input:**

```
INSERT
Cand1, 1, S1, D1, C1, P1, 1200
INSERT
Cand2, 100, S1, D1, C1, P2, 1000
INSERT
Cand3, 101, S1, D1, C1, P3, 1500
INSERT
Cand4, 102, S1, D1, C1, P4, 2000
INSERT
Cand5, 105, S1, D1, C1, P5, 500
INSERT
Cand6, 2, S1, D2, C2, P2, 4000
INSERT
Cand7, 3, S2, D3, C3, P3, 2100
INSERT
Cand8, 4, S3, D4, C4, P4, 800
INSERT
Cand9, 5, S3, D5, C5, P6, 1600
UPDATE
Cand1, 1, 2500
UPDATE
Cand8, 4, 1400
INSERT
Cand9, 6, S1, D1, C6, P6, 20
INSERT
Cand10, 7, S1, D2, C3, P2, 500
PRINT
TOP k IN CONSTITUENCY
C1, 3
LEADING PARTY IN STATE
S1
GLOBAL LEADING PARTY
VOTE SHARES
P1, S1
DISCARD ALL VOTES IN CONSTITUENCY
C1
```

**Expected Output:**

```
Inserting: Cand1, 1, S1, D1, C1, P1, 1200
Inserting: Cand2, 100, S1, D1, C1, P2, 1000
Inserting: Cand3, 101, S1, D1, C1, P3, 1500
Inserting: Cand4, 102, S1, D1, C1, P4, 2000
Inserting: Cand5, 105, S1, D1, C1, P5, 500
Inserting: Cand6, 2, S1, D2, C2, P2, 4000
Inserting: Cand7, 3, S2, D3, C3, P3, 2000
Inserting: Cand8, 4, S3, D4, C4, P4, 800
Inserting: Cand9, 5, S3, D5, C5, P6, 1500
Printing Election data:
Cand1, 1, S1, D1, P1, 1200
Cand2, 100, S1, D1, C1, P2, 1000
Cand3, 101, S1, D1, C1, P3, 1500
Cand5, 105, S1, D1, C1, P5, 500
Cand4, 102, S1, D1, C1, P4, 2000
Cand8, 4, S3, D4, C4, P4, 800
Cand9, 5, S3, D5, C5, P6, 1600
Cand6, 2, S1, D2, C2, P2, 4000
Cand7, 3, S2, D3, C3, P3, 2100
Updating Cand1, 1, 2500:
Updating Cand8, 4, 1400:
Inserting: Cand9, 6, S1, D1, C6, P6, 20
Inserting: Cand10, 7, S1, D2, C3, P2, 500
Reporting Top 3 in constituency C1:
Cand1, 1, P1
Cand4, 102, P4
```

```
Cand3, 101, P3
Reporting the leading party in state S1:
P2
Reporting leading party across all constituencies:
P2
Reporting vote share of party P1 in state S1:
20.8
Discarding the info of all candidates in constituency C1:
```

## 7 Submission Guidelines

- Submit your code in a *.zip* file named in the format < EntryNo >.zip. Make sure that when we run **unzip < yourfile >.zip**, a folder < YourEntryNo > should be produced in the current working directory. You will be penalized for any submissions that do not confirm to this requirements.
- The exact directory structure will be detailed on the moodle submission link. Your submission will be auto-graded. This means that it is essential to make sure that your code follows the specifications of the assignment precisely.
- Your code must compile and run on our VMs. They run amd64 Linux version ubuntu 16.04 running Java JDK 11.0.5 `https://www.oracle.com/technetwork/java/javase/downloads/jdk11-downloads-5066655.html`
- You will be able to check if your submission complies with the proper format and runs as expected in the test environment by submitting to Moodle. The auto-grading in Moodle will run your code and verify if it passes the preliminary test cases.
- Your submission will only be accepted once it passes the initial test cases. For this, you must at least complete the Minimal submission tests of the assignment (will be announced soon). Further test cases will be used for Evaluation.

## 8 To Do/Not To Do

- You must work on this assignment individually.
- The programming language to be used is Java.
- You are not allowed to use any other external libraries for this assignment.
- Your code must be your own. You are not to take guidance from any general-purpose or problem-specific code meant to solve this or related problem.
- We will run a plagiarism detection check. Any person found guilty will be awarded a suitable penalty as announced in the class/course website. Please read academic integrity guidelines on the course home page and follow them carefully.