# CSI 508. Database Systems I – Spring 2017

## Programming Assignment II

The total grade for this assignment is 100 points. The deadline for this assignment is **11:59 PM, April 17, 2017**. *Submissions after this deadline will not be accepted.* Students are required to enter the UAlbany Blackboard system and then upload a .zip file (in the form of [first name]_[last name].zip) that contains the Eclipse project directory and a short document describing:

- any missing or incomplete elements of the code
- any changes made to the original API
- the amount of time spent for this assignment
- suggestions or comments if any

---

In this programming assignment, you need to complete a Java program that demonstrates how B+-trees run. For example, Figures 1 and 2 show how a B+tree can change as a key 'b' is inserted into the tree.

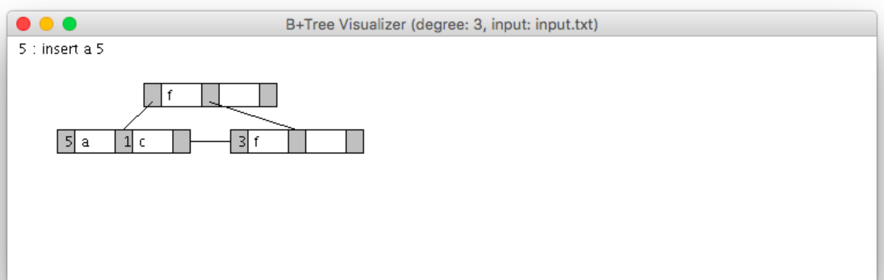


Figure 1: Before Inserting b

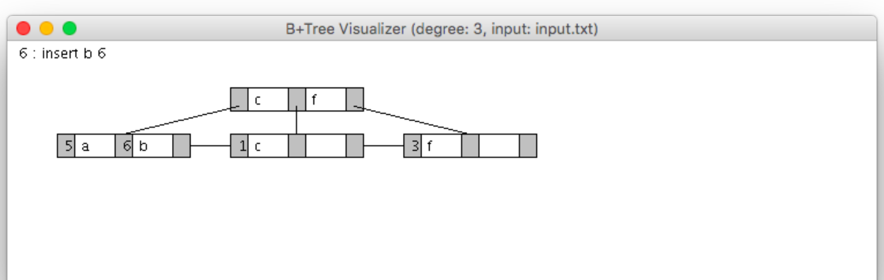


Figure 2: After Inserting b

You first need to run Eclipse on your machine and import the "`bplus_tree`" project (see Appendix A). This assignment provides an eclipse project that contains a B+-tree visualizer (see

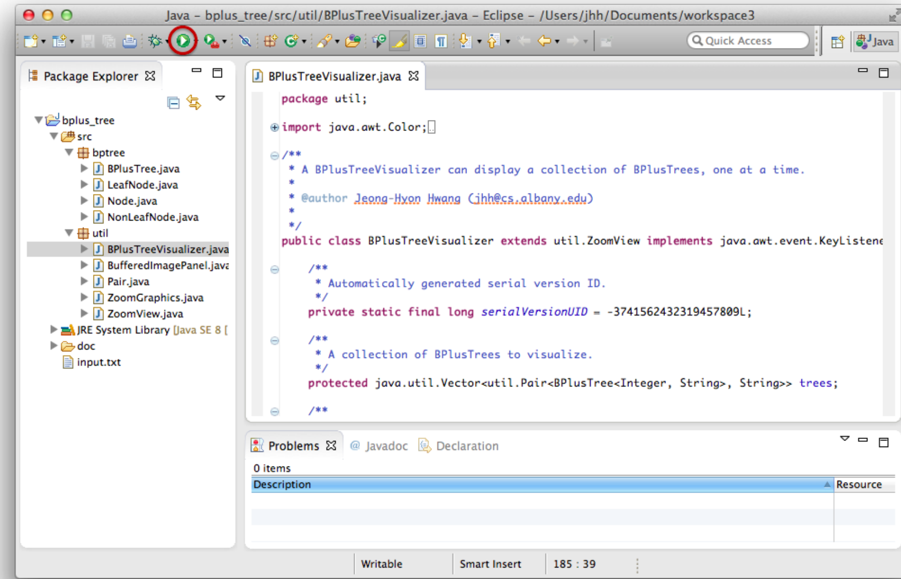

Figure 3: Eclipse

BPlusTreeVisualizer.java) and a partially implemented B+-tree class (see `BPlusTree.java`). In `BPlusTree.java`, methods for finding/inserting a key-pointer pair are already implemented (see `insert(K k, P p)` as well as Algorithms 1, 2, and 3 in Appendix B). For this assignment, you need to implement the `delete(K k, P p)` method so that we can also remove key-pointer pairs from the tree (refer to Algorithms 4, 5, and 6 in Appendix C). Within your code, please add *detailed comments* so that every step of deletion can be clearly understood. Insufficient comments will negatively affect your grade.

As a side note, the visualizer executes both *insert* and *delete* commands defined in the `input.txt` file (see the "`bplus_tree`" project directory). Pleasure ensure that your code works well even if you change the *degree* of the B+-tree and the *insert* and *delete* commands in `input.txt`.

You can play with the visualizer as follows:

1. move to the previous/next frame (left and right arrow keys, respectively)
2. zoom in/out (up and down arrow keys, respectively, or left and right double clicks)
3. panning ([ctrl] key + left/right/up/down arrow keys, or mouse drag and drop without pressing any key)

Good luck! I hope this assignment will help you better understand how B+-trees run.

## Appendix A. Importing the `bplus_tree` Project

1. Run Eclipse. In the menu bar, choose "File" and then "Import". Next, select "General" and "Existing Projects into Workspace". Then, click the "Browse" button and select the "`bplus_tree.zip`" file contained in this assignment package.
2. Once the project is imported, you can choose `BPlusTreeVisualizer.java` and then run the program by clicking the icon highlighted in Figure 3.

## Appendix B. Pseudo-code for Inserting an Entry in a B+-tree

**Algorithm 1:** insert(*key* $K$, *pointer* $P$)

**if** *tree is empty* **then**
  Create an empty leaf node $L$, which is also the root
**else**
  Find the leaf node $L$ that should contain key $K$
**if** $L$ *has less than* $n-1$ *keys* **then**
  insert_in_leaf($L$, $K$, $P$)
**else**
  Copy $L.P_0 \cdots L.K_{n-2}$ to a block of memory $T$ that can hold $n$ (pointer, key) pairs
  insert_in_leaf($T$, $K$, $P$)
  Create node $L'$
  Set $L'.P_{n-1} = L.P_{n-1}$
  Erase $L.P_0$ through $L.K_{n-2}$ from $L$
  Set $L.P_{n-1} = L'$
  Copy $T.P_0$ through $T.K_{\lceil n/2 \rceil -1}$ from $T$ into $L$ starting at $L.P_0$
  Copy $T.P_{\lceil n/2 \rceil}$ through $T.K_{n-1}$ from $T$ into $L'$ starting at $L'.P_0$
  Let $K'$ be the smallest key in $L'$
  insert_in_parent($L$, $K'$, $L'$)

**Algorithm 2:** insert_in_leaf(*node* $L$, *key* $K$, *pointer* $P$)

**if** $K < L.K_0$ **then**
  Insert $P$, $K$ into $L$ just before $L.P_0$
**else**
  Let $K_i$ be the highest key in L that is less than $K$
  Insert $P$, $K$ into $L$ just after $T.K_i$

**Algorithm 3:** insert_in_parent(*node* $N$, *key* $K$, *node* $N'$)

**if** $N$ *is the root of the tree* **then**
  Create a new node $R$ containing $N$, $K$, $N'$
  Make $R$ the root of the tree
  **return**
Let $P = parent(N)$
**if** $P$ *has less than* $n$ *pointers* **then**
  Insert($K$, $N'$) in $P$ just after $N$
**else**
  Copy $P$ to a block of memory $T$ that can hold $P$ and ($K$, $N'$)
  Insert ($K$, $N'$) into $T$ just after $N$
  Erase all entries from $P$
  Create node $P'$
  Copy $T.P_0 \cdots T.P_{\lceil n/2 \rceil -1}$ into $P$
  Copy $T.P_{\lceil n/2 \rceil} \cdots T.P_n$ into $P'$
  Let $K' = T.K_{\lceil n/2 \rceil -1}$
  insert_in_parent($P$, $K'$, $P'$)

## Appendix C. Pseudo-code for Deleting an Entry in a B+-tree

**Algorithm 4:** delete(*key K*, *pointer P*)

```
find the leaf node L that contains (K, P)
delete_entry(L, K, P)
```

**Algorithm 5:** delete_entry(*node N*, *key K*, *pointer P*)

```
delete (K, P) from N
if N is the root then
    if N has only one remaining child then
        Make the child of N the new root of the tree and delete N
else if N has too few keys/pointers then
    Let N' be the previous or next child of parent(N)
    Let K' be the key between pointers N and N' in parent(N)
    if entries in N and N' can fit in a single node then
        if N' is the predecessor of N then
            merge(N', K', N)
        else
            merge(N, K', N')
    else
        // redistribution: move a key and a pointer from node N' to node N
        if N' is the predecessor of N then
            if N is a non-leaf node then
                Let m be such that N'.P_m is the last pointer in N'
                Insert (N'.P_m, K') as the first pointer and key in N by shifting other pointers and
                  keys right
                Remove (N'.K_{m-1}, N'.P_m) from N'
                Replace K' in parent(N) by N'.K_{m-1}
            else
                Let m be such that (N'.P_m, N'.K_m) is the last pointer/key pair in N'
                Insert (N'.P_m, N'.K_m) as the first pointer and key in N by shifting other pointers
                  and keys right
                Remove (N'.P_m, N'.K_m) from N'
                Replace K' in parent(N) by N'.K_m
        else
            ... symmetric to the then case ...
```

**Algorithm 6:** merge(*node N'*, *key K'*, *node N*)

```
if N is not a leaf node then
    Append K' and all pointers and keys in N to N'
else
    Append all (K_i, P_i) pairs in N to N'
    Set N'.P_{n-1} = N.P_{n-1}
delete_entry(parent(N), K', N)
delete node N
```