**Advanced Algorithms**

**Exercise for Lecture 5,6,7**

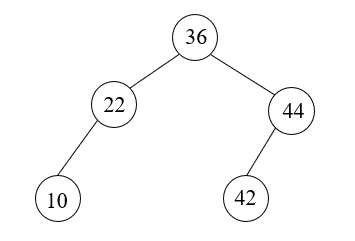
|  |  |  |  |
| --- | --- | --- | --- |
| **Student Name** |  | **Student ID** |  |
| **Lecture 5** |  | | |
| **Lecture 6** |  | | |
| **Lecture 7** |  | | |
| **Total Score** |  | | |
| **Notes** | Deadline: **2024-10-10 24:00**  Submission Format: ‘**Lecture567\_Name\_ID.docx**’, and please send to: **3459996503@qq.com**.  This assignment is meant to be an evaluation of your **individual** understanding coming into the course and should be completed **without collaboration** or outside help. | | |

**Lecture 5**

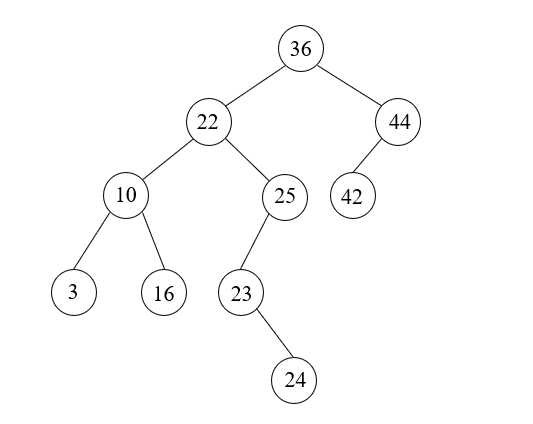
**Problem 5.1[20 points]** Draw the BST where the data value at each node is an integer and the values are entered in the following order: 36, 22, 10, 44, 42.Then, add 16, 25, 3, 23, 24 in this order, and again draw the tree.Then draw the tree after deletions of 42, 23 and 22 in this order.

**Solution**:

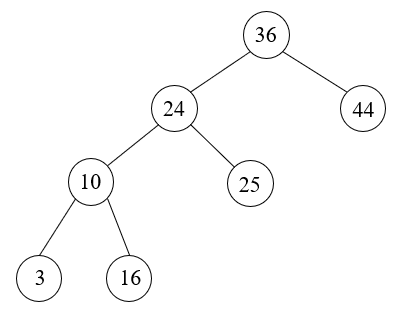
Insert 36, 22, 10, 44, 42 in order,



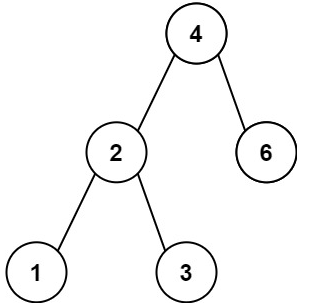
add 16, 25, 3, 23, 24 in order,



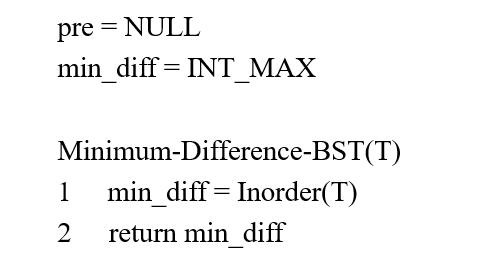
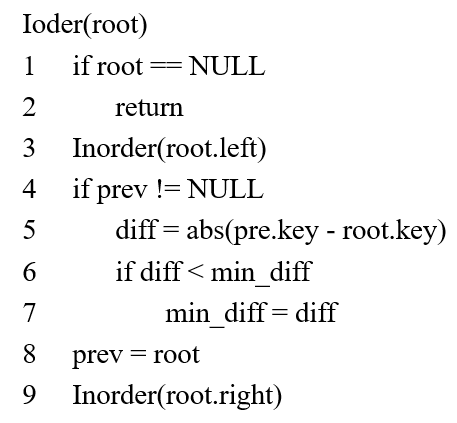
delete 42, 23 and 22 in order,



**Problem 5.2[15 points]** Given the root of a binary search tree T, write an algorithm Minimum-Difference-BST(T) which returns the minimum difference between the values of any two different nodes in the tree.The difference is a positive number, equal to the absolute value of the difference between the two values. For example, the minimum difference of the following tree is 1.



**Solution**:

**Lecture 6**

**Problem 6.1[20 points]**

(1) Please insert key value 100, 99, 75, 50, 25, 5 into a red black tree one by one. Please show each intermediate state of the red black tree.

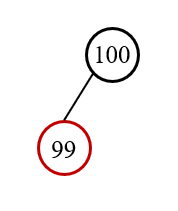
(2) Please delete key value 5, 25, 50, 75, 99, from the red black tree one by one which you get in (1). Please show each intermediate state of the red black tree.

**Solution:**

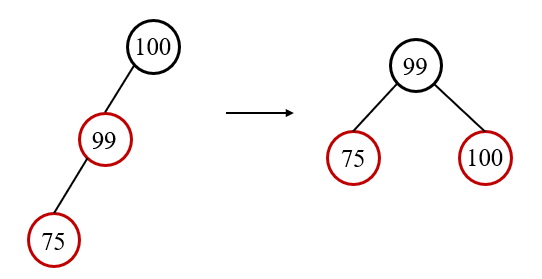
Insert 100:



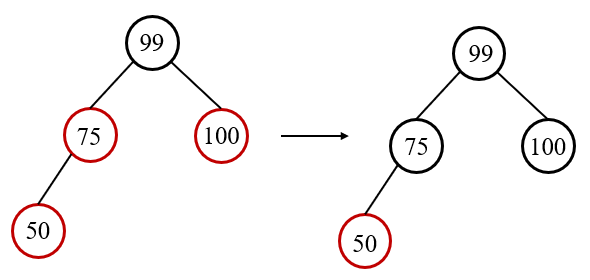
Insert 99:



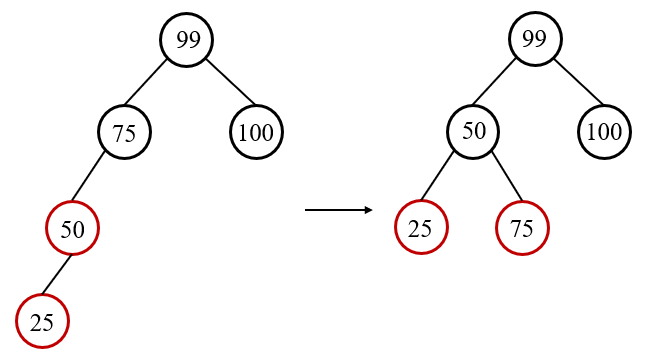
Insert 75:



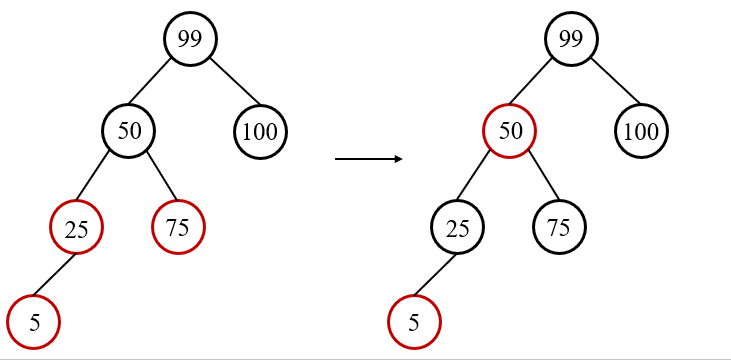
Insert 50:



Insert 25:

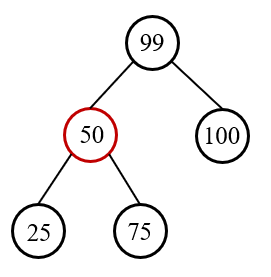


Insert 5:

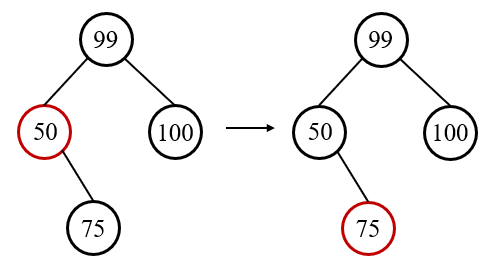


(2)

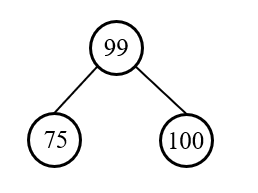
Delete 5:



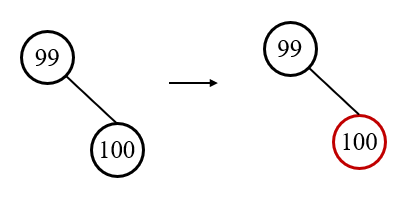
Delete 25:



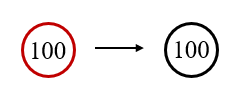
Delete 50:



Delete 75:



Delete 99:

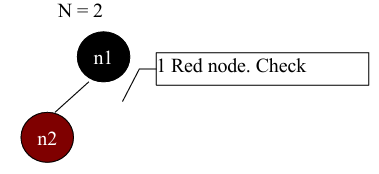


**Problem 6.2[15 points]** Consider a red-black tree formed by inserting n nodes into an initially empty red-black tree. Argue that if n > 1, the tree has at least one red node. Your argument must be in the form of a proof by induction. There are two cases you will need to take into consideration when making your inductive step. The trivial case when Red Node N+1 is inserted as a child of a black node does not need to be solved.

**Solution**:

Proof by Induction:

Base Case:



Inductive Hypothesis: Assume that for a red-black tree of n nodes, where 1<n≤N , there exists at least 1 red node.

Inductive Step: Prove that for a red-black tree of N+1 nodes there exists at least 1 red node.

There are 2 cases we have to of interest on the N+1 th insertion.

Case 1: Red Node N+1 is inserted as a child of a black node. This is the base case which we proved to be true already. This is the trivial case.

Case 2: Red Node N+1 is inserted as a child of a red node. We must look at the insertion cases that have X as a child of a red node P.

Insertion Case 1: X remains red after the recoloring of P, G, U(Parent, Grandparent, Uncle). It also remains the same color after we move reference X to Grandparent of X. Thus we have at least 1 red node.

Insertion Case 2: The parent P of X remains red after the Zig-Zag rotation of X about P and then G. P remains red after recoloring X and G. Thus we have at lease 1 red node.

Insertion Case 3: X remains red after the rotation of P about G. X remains red after the recoloring of P and G. Thus we have at least 1 red node.

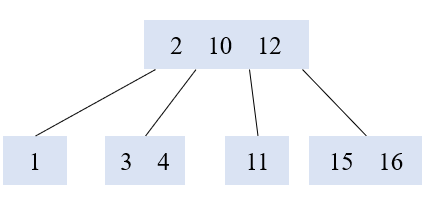
Insertion Case 0: Not included since n!=1 , Insertion Cases 2 & 3 are terminating conditions, and Insertion Case 1 still produces 1 red node.

**Lecture 7**

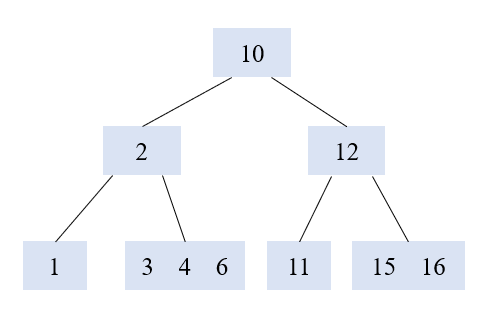
**Problem 7.1[20 points]** Show the results of inserting the keys 10, 11, 1, 2, 12, 16, 15, 3, 4, 6, 22, 17, 36, 9, 23 in order into an empty B-tree with minimum degree 2. Draw the configurations of inserting 4,6 and the final configuration.

**Solution**:

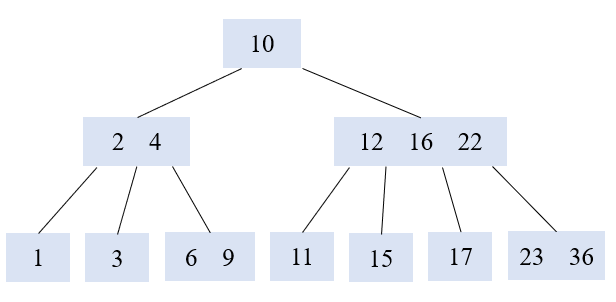
The configurations of inserting 4:



The configurations of inserting 6:



The final configurations:



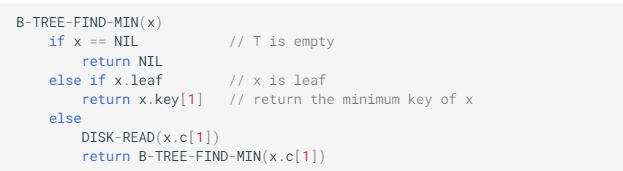
**Problem 7.2[10 points]** Explain how to find the minimum key stored in a B-tree and how to find the predecessor of a given key stored in a B-tree. Provide an explanation and include relevant pseudocode.

**Solution**:

Finding the minimum in a B-tree is quite similar to finding a minimum in a binary search tree. We need to find the left most leaf for the given root, and return the first key.

**PRE:** *x* is a node on the B-tree *T*. The top level call is B-TREE-FIND-MIN(*T*.*root*).

**POST:** FCTVAL is the minimum key stored in the subtree rooted at *x*.



Finding the predecessor of a given key x.keyi​ is according to the following rules:

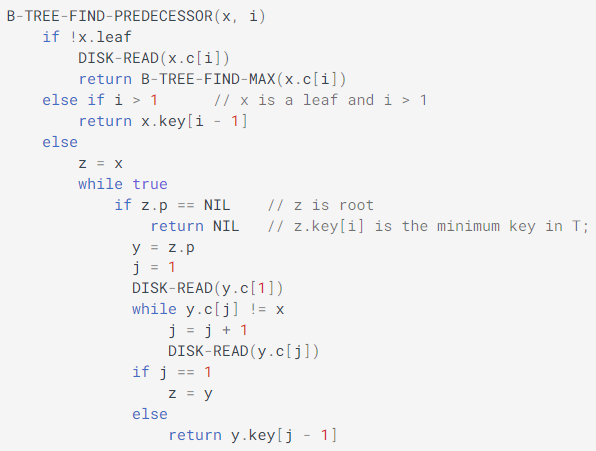
a. If *x* is not a leaf, return the maximum key in the *i*-th child of *x*, which is also the maximum key of the subtree rooted at *x*.*ci*​.

b. If *x* is a leaf and *i*>1, return the  (*i*−1)st key of *x*, i.e., *x*.*keyi*−1​.

c. Otherwise, look for the last node y (from the bottom up) and *j*>0, such that *x*.*keyi*​ is the leftmost key in  *y*.*cj*​; if *j*=1, return NIL since *x*.*keyi*​is the minimum key in the tree; otherwise we return *y*.*keyj*−1​.

**PRE:** *x* is a node on the B-tree *T*. *i* is the index of the key.

**POST:**  FCTVAL is the predecessor of *x*.*keyi*​.



**PRE:** *x* is a node on the B-tree *T*. The top level call is  B-TREE-FIND-MAX(*T*.*root*).

**POST:**  FCTVAL is the maximum key stored in the subtree rooted at *x*.

