Department of Informatics
Database Technology Group
University of Zurich — Prof. Dr. Michael Böhlen

Informatics II
Exercise 9

March 16, 2022

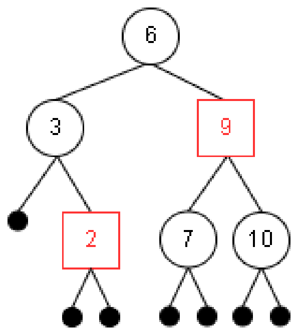
Red Black Tree

Consider the red-black tree in lecture where black nodes are denoted with a circle and red nodes are denoted with a square. We use the key to represent a node. For example, 2 represents the node with key 2. We consider four operations:

1. Creating a new node (left and right are set to NULL):
   Example: create node 9: `create(9)`
2. Setting a property (color, key, left, right) of a node:
   Example: set the key of node 9 to 5: `9->key = 5`. Here, 9 represents the node with key 9.
3. Rotating a node:
   Example: right rotate node 5: `RightRotate(5)`
4. Deleting a node:
   Example: delete node 9: `delete(9)`

Task 1

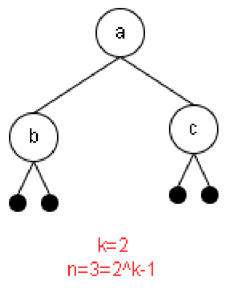
1. Consider the red-black tree below. Determine the number of internal nodes, the black height of node 3 and the black height of the tree?

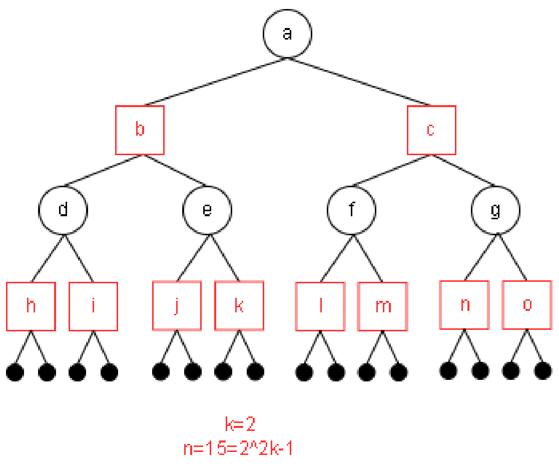


Solution: 6, 1 and 2.

2. What are the largest and smallest possible numbers of internal nodes in a red-black tree with a black-height k? Give examples.

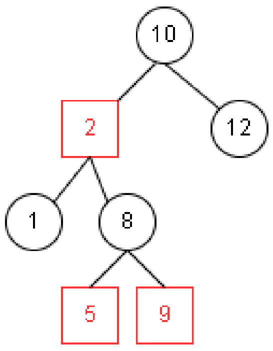
$2^k - 1, 2^{2k} - 1$

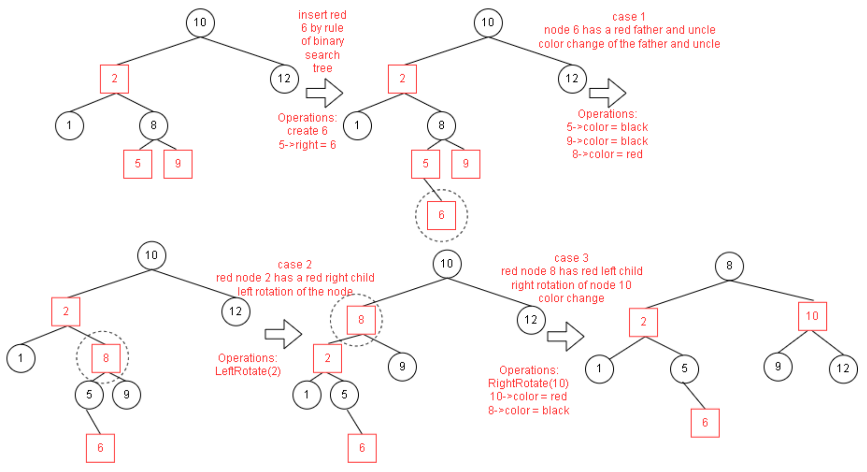




Task 2

1. Consider the red-black tree shown below. State the operations that are required to insert 6 into the red-black tree.





insert red
6 by rule
of binary
search
tree

Operations:
create 6
5->right = 6

case 1
node 6 has a red father and uncle
color change of the father and uncle

Operations:
5->color = black
9->color = black
8->color = red

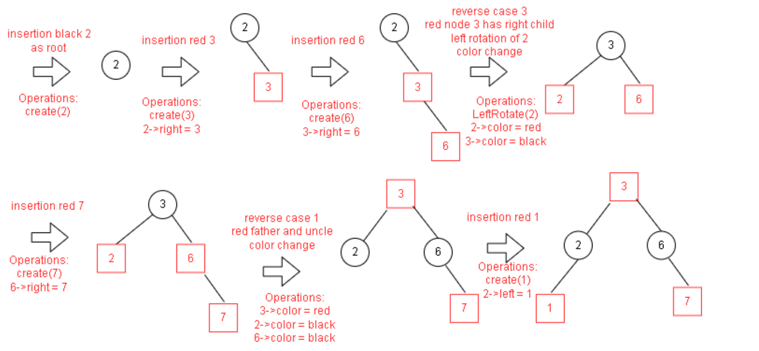
case 2
red node 2 has a red right child
left rotation of the node

Operations:
LeftRotate(2)

case 3
red node 8 has red left child
right rotation of node 10
color change

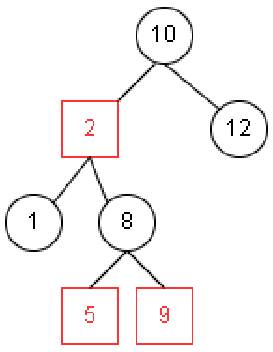
Operations:
RightRotate(10)
10->color = red
8->color = black

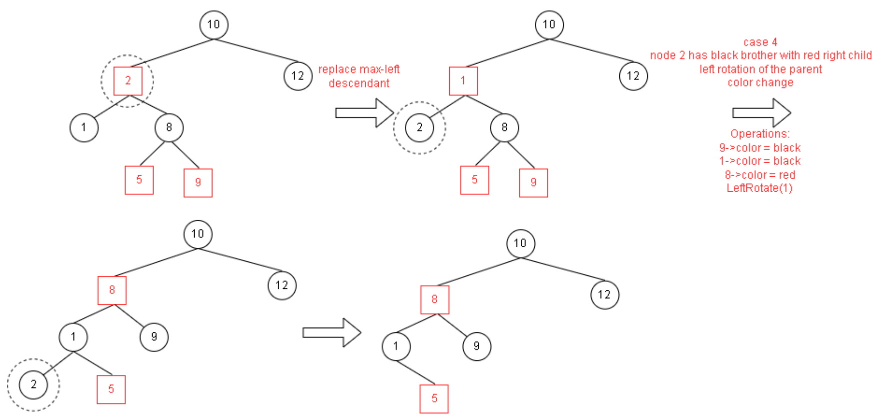
2. Show the red-black tree that results after each of the integer keys 2, 3, 6, 7, and 1 insertion into an empty red black tree step by step.


insertion black 2 as root
Operations: create(2)
insertion red 3
Operations: create(3) 2->right = 3
insertion red 6
Operations: create(6) 3->right = 6
reverse case 3 red node 3 has right child left rotation of 2 color change
Operations: LeftRotate(2) 2->color = red 3->color = black
insertion red 7
Operations: create(7) 6->right = 7
reverse case 1 red father and uncle color change
Operations: 3->color = red 2->color = black 6->color = black
insertion red 1
Operations: create(1) 2->left = 1

Task 3

Delete 2 from the red black tree. Show state of tree after each step by drafts and explain the change.

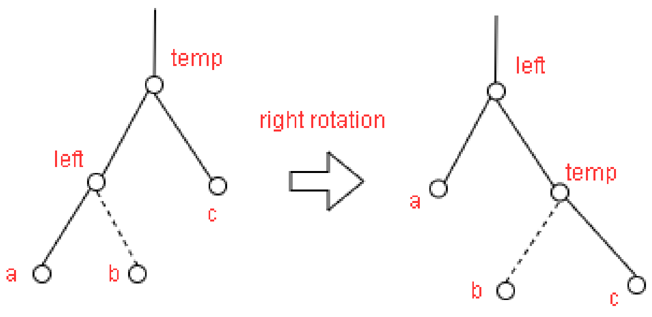



replace max-left descendant
case 4 node 2 has black brother with red right child left rotation of the parent color change
Operations: 9->color = black 1->color = black 8->color = red LeftRotate(1)

Follow-up practises: show the states of deleting other nodes.

Task 4

1. Complete the code segment of `RightRotation(temp)` according to the diagram and instructions. Assume that the left child of node `temp` exists.



```
 1 struct node {
 2 struct node* p; // parent
 3 struct node* r; // right child
 4 struct node* l; // left child
 5 };
 6
 7
 8 void rightrotate(struct node* temp){
 9     struct node * g = temp−>p;
10     struct node* left = temp−>l;
11     (a)__________ // 'b' becomes left child of 'temp'
12     if(left−>r) {
13         (b)__________ // 'temp' becomes parent of 'b'
14     }
15     (c)__________ // 'left' become the child of g
16     if (temp == g−>l){
17         g−>l = left;
18     }
19     else {
20         g−>r = left;
21     }
22     (d)__________ // 'temp' becomes right child of 'left'
23     (e)__________ // 'left' becomes parent of 'temp'
24 }
```

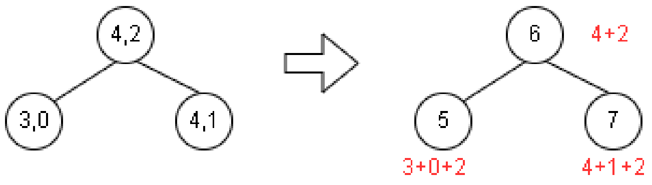
a. temp->l = left->r;
b. left->r->p = temp;
c. left->p = g;
d. left->r = temp;
e. temp->p = left;

2. State what lines 16-21 have done.

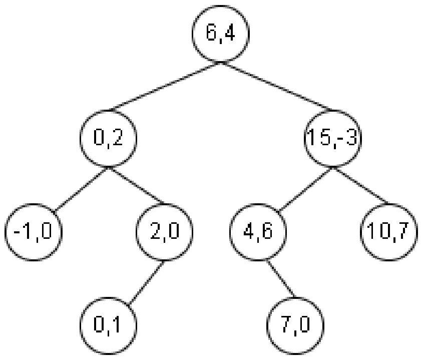
Since node `temp` is replaced by node `left`, these several lines make node `left` the right or left child as node `temp` was.

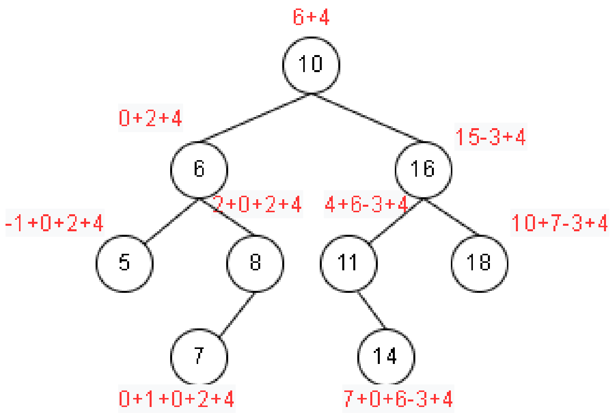
Task 5

Consider an enhanced binary search tree in which each node v contains a key and an value called `addend`. The addend of node v is implicitly added to keys of all nodes in the subtree rooted at v (including v). Let (key, addend) denote the contents of a node. See the following example that transforms an enhanced binary search tree to a binary search tree.

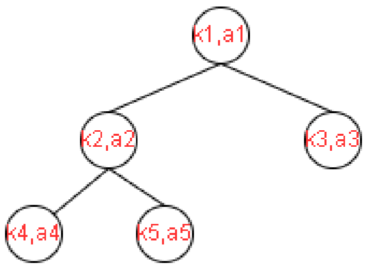


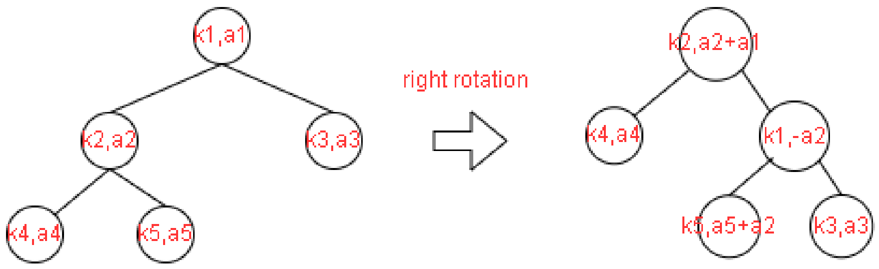
1. Show the binary search tree of the enhanced binary search tree below?





2. Consider the enhanced binary search tree below. Show the state of the enhanced binary search tree after the right rotation of node `k1`.




k1,a1
k2,a2
k3,a3
k4,a4
k5,a5
right rotation
k2,a2+a1
k4,a4
k1,-a2
k5,a5+a2
k3,a3