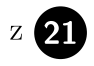
Ian Kaiserman 00867173 **Homework 3** CSCD 320
October 21, 2022

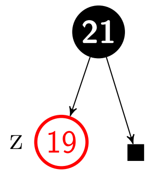
1 Problem 1 (20 points).

Show the trace of the construction of the Red-Black tree for the sequence $21, 19, 17, 12, 15, 9$. That is, you need to draw the state of the tree after inserting each number.

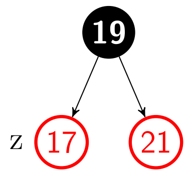
Insert node 21, apply rule 0, color(root) = black



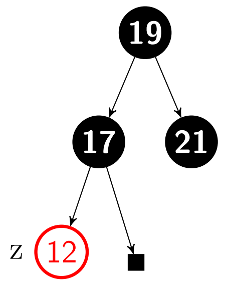
Insert node 19



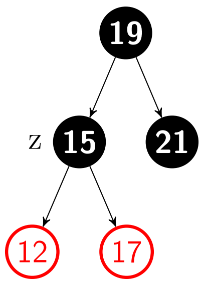
Insert node 17, Rule 3: rotate right on node 21, swap colors of 21 and 19



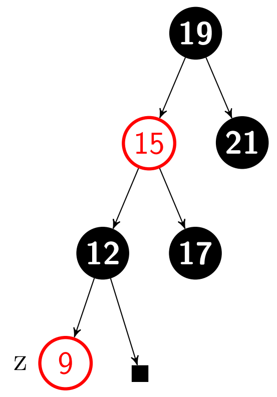
Insert node 12, Rule 1: set nodes 17 and 21 to black, 19 to red, Rule 0: set root node 19 to black



Insert node 15, Rule 2: Rotate left on 12, Rule 3: Rotate right on 17, swap colors of 17 and 15



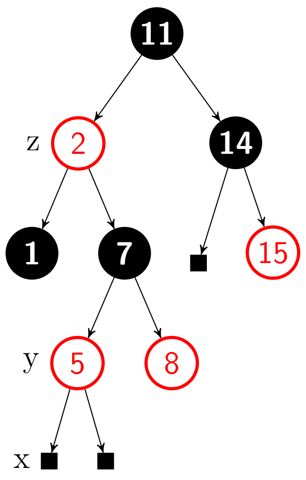
Insert node 9, Rule 1: set nodes 12 and 17 to black, node 15 to red



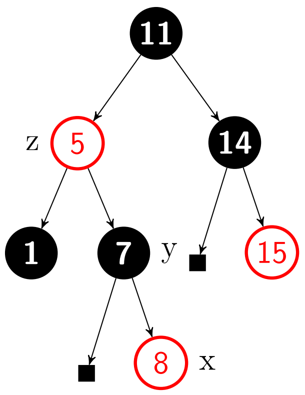
2 Problem 2 (20 points).

Show the trace of deleting the nodes from the Red-Black tree below in the order of $2, 5, 1, 14, 11, 15, 7, 8$. That is, show the state of the tree after deleting each node.

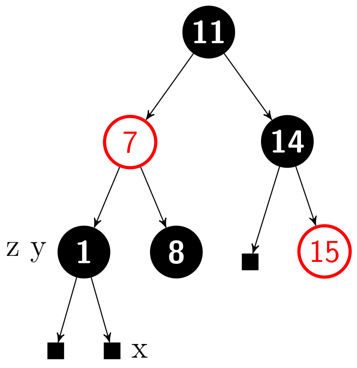
HOMEWORK NOTE: Any extra black squares are not meant to represent all null leaves/extra black nodes, they are simply placeholders to make the tree show up correctly.



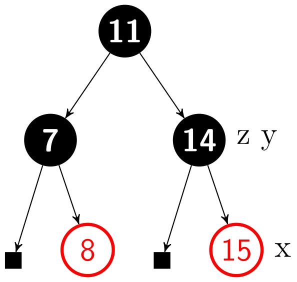
Delete node 2 using BST deletion, y was red before deletion, no further steps



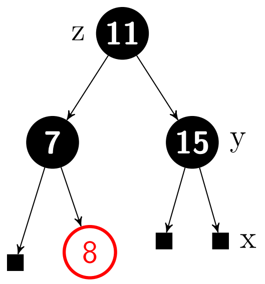
Delete node 5 using BST deletion, y changed from black to red (z old color), y was black before deletion but x is red, change x color to black, no further steps



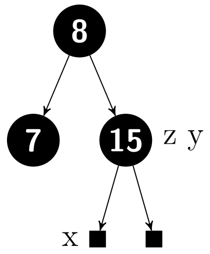
Delete node 1 using BST deletion, y was black before deletion, x was black before deletion, x goes to old y (and z) location marked double black, Rule 2 applies because 8's children are two black NIL leaves, set color of 8 to red, move double black and x to 7, 7 becomes black, double black removed, no further steps



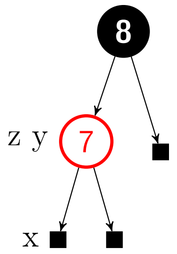
Delete node 14 using BST deletion, move x to old y location, y was black before deletion, but x was red, change x color to black, no further steps



Delete node 11 using BST deletion, y is already same color as z, y and x were both black before deletion, x is double black at old y location, Rule 3 applies because 8 is red and closer to x, left rotation on 7, set 7 to red and 8 to black, now Rule 4 applies because 7 (red) is further from x, right rotation on 15, 15 and 8 already same color, set 7 to black, no further steps



Delete node 15 using BST deletion, move x to old y location, y and x were both black before deletion, double black on x, Rule 2 applies because 7 is black and has two black NIL leaf children, set color of 7 to red, move double black to 8, 8 is root, discard double black, no further steps



Delete node 7 using BST deletion, move x to old y location, y was red before deletion, no further steps

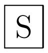
Delete node 8 using BST deletion, tree is now empty, no further steps

3 Problem 3 (20 points).

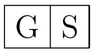
Trace the ONE-PASS construction of the B-tree for the sequence
$S, G, W, H, O, U, M, A, C, X, P$. Draw the configuration of the B-tree after inserting each

letter. We use $t = 2$ as the branching degree threshold of the B-tree, so that: (1) all the non-leaf node must have at least $t - 1 = 1$ key and at most $2t - 1 = 3$ keys; and (2) The root node of an non-empty B-tree must have at least one key and at most $2t - 1 = 3$ keys.

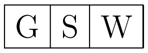
Insert S



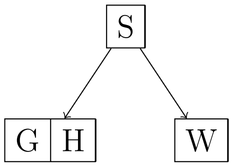
Insert G



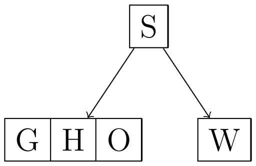
Insert W



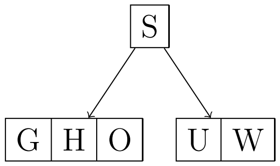
Split root node, insert H



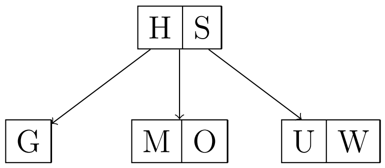
Insert O



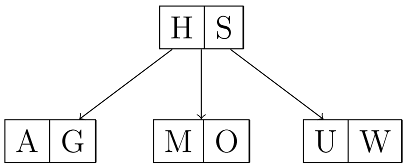
Insert U



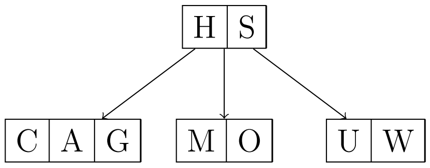
Split first node on level 2, Insert M



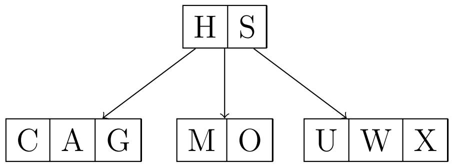
Insert A



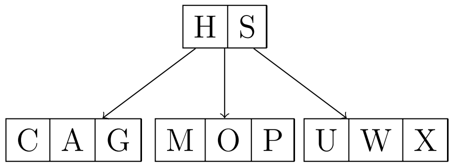
Insert C



Insert X

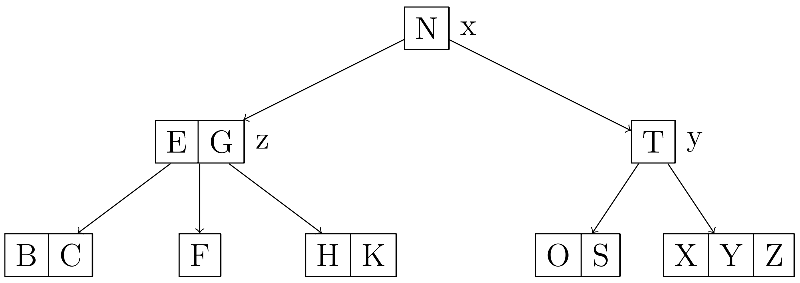


Insert P



4 Problem 4 (20 points).

Trace the deletion the sequence of keys T, G, F, O from the B-tree below. Draw the configuration of the B-tree after each deletion. We use $t = 2$ as the branching degree threshold of the B-tree, so that: (1) all the non-leaf node must have at least $t - 1 = 1$ key and at most $2t - 1 = 3$ keys; and (2) The root node of an non-empty B-tree must have at least one key and at most $2t - 1 = 3$ keys.

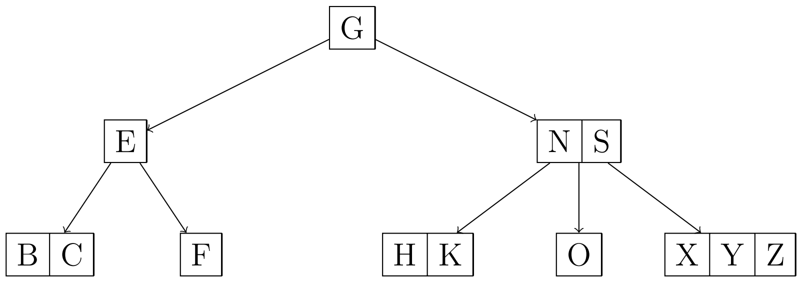


Delete T

Rule 3: Case 2: $y.n == 1$ and $z.n > 1$, k' is N, k" is G, move k' to first element of y, and k" to original position of k', re-link child of k" to right side of k'. Call delete on T in y

Rule 2: Case 1: $y.n > 1$, k is in x, replace k with predecessor (S) in y, call delete on S in y

Rule 1: Delete S in y, deletion done

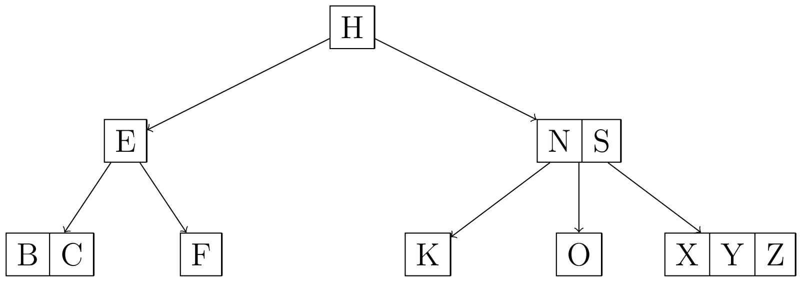


Delete G

Rule 2: Case 1: $z.n > 1$, k is in x, replace k with successor (H) in z subtree, call delete on H in z

Rule 3: Case 1: $y.n > 1$, k is not in x, call delete on H in y

Rule 1: Delete H in y, deletion done

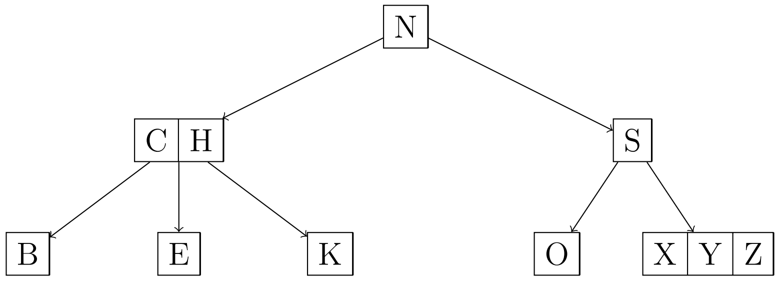


Delete F

Rule 3 Case 2: $y.n == 1$, $z.n > 1$. k' is H, k" is N. Move k' to last element of y, move k" to original position of k'. Re-link child tree of k" to right side of k'. Call delete on F in y.

Rule 3 Case 2: $y.n == 1$, $z.n > 1$. k' is E, k" is C. Move k' to first element of y, move k" to original position of k'. Call delete on F in y.

Rule 1: Delete F in y, deletion done.

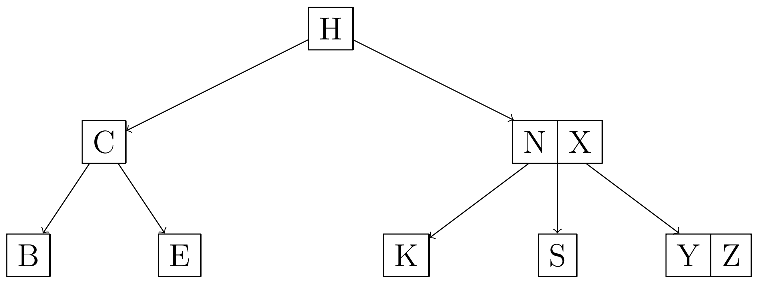


Delete O

Rule 3 Case 2: $y.n == 1$, $z.n > 1$. k' is H, k" is E. Move k' to first element of y, move k" to original position of k'. Re-link child tree of k" to left side of k'. Call delete on O in y.

Rule 3 Case 2: $y.n == 1$, $z.n > 1$. k' is S, k" is X. Move k' to last element of y, move k" to original position of k'. Call delete on O in y.

Rule 1: Delete O in y, deletion done.



5 Problem 5 (20 points).

5.a Algorithm Ideas

The primary idea for the minimum key function is to recursively call itself using the current node's leftmost link until we hit the bottom, and return the leftmost value of the node at the very bottom of the tree. This is not too expensive in time and resources because it only needs to iterate the tree as many times as the height of the tree, given that B-trees are naturally balanced.

The successor function would be able to use the minimum key function as a subroutine, because the successor of a given value is going to be the minimum key of the subtree in the link directly next to it, or simply the next value if the link next to it is null. If i is at the very end of a node, the algorithm would read the parent node of the current node, try to read the next value

5.b Pseudocode

```
B_tree_min(root)
  if root == null, return null
  //return smallest value if root is at the bottom of the tree
  else if root is leaf, return root.value[0]
  // else, recursively call on the first link of the current node
  else
    DISK_READ(root.child[0])
    return B_tree_min(root.child[0])

B_tree_successor(node, i)
  // if node is bottom of tree, next value
  if node is leaf, return node.value[i+1]
    // covers if the i location is the last element in the node.
    // this could continue being called until a parent is found
    // with a valid "next" value after the current link
    if node.value[i+1] == null
      DISK_READ(node.parent)
      return node.parent.value[next]
      *keep reading next parent if there is no next value
  // find smallest next value using link directly next to i value
  else
    DISK_READ(node.child[i+1])
    return B_tree_min(node.child[i+1])
```