```
// Inserting a key on a B-tree in Java

import java.util.Stack;
public class BTree {
  private int T;
  public class Node {
    int n;
    int key[] = new int[2 * T - 1];
    Node child[] = new Node[2 * T];
    boolean leaf = true;
    public int Find(int k) {
      for (int i = 0; i < this.n; i++) {
        if (this.key[i] == k) {
          return i;
        }
      }
      return -1;
    };
  }
  public BTree(int t) {
    T = t;
    root = new Node();
    root.n = 0;
    root.leaf = true;
  }

  private Node root;
  // Search the key
  private Node Search(Node x, int key) {
    int i = 0;
    if (x == null)
      return x;
    for (i = 0; i < x.n; i++) {
      if (key < x.key[i]) {
        break;
      }
      if (key == x.key[i]) {
        return x;
      }
    }
    if (x.leaf) {
      return null;
    } else {
      return Search(x.child[i], key);
    }
  }

  // Split function
```

```
private void Split(Node x, int pos, Node y) {
  Node z = new Node();
  z.leaf = y.leaf;
  z.n = T - 1;
  for (int j = 0; j < T - 1; j++) {
    z.key[j] = y.key[j + T];
  }
  if (!y.leaf) {
    for (int j = 0; j < T; j++) {
      z.child[j] = y.child[j + T];
    }
  }
  y.n = T - 1;
  for (int j = x.n; j >= pos + 1; j--) {
    x.child[j + 1] = x.child[j];
  }
  x.child[pos + 1] = z;

  for (int j = x.n - 1; j >= pos; j--) {
    x.key[j + 1] = x.key[j];
  }
  x.key[pos] = y.key[T - 1];
  x.n = x.n + 1;
}

// Insert the key
public void Insert(final int key) {
  Node r = root;
  if (r.n == 2 * T - 1) {
    Node s = new Node();
    root = s;
    s.leaf = false;
    s.n = 0;
    s.child[0] = r;
    Split(s, 0, r);
    _Insert(s, key);
  } else {
    _Insert(r, key);
  }
}

// Insert the node
final private void _Insert(Node x, int k) {

  if (x.leaf) {
    int i = 0;
    for (i = x.n - 1; i >= 0 && k < x.key[i]; i--) {
      x.key[i + 1] = x.key[i];
```

```
    }
    x.key[i + 1] = k;
    x.n = x.n + 1;
   } else {
    int i = 0;
    for (i = x.n - 1; i >= 0 && k < x.key[i]; i--) {
    }
    ;
    i++;
    Node tmp = x.child[i];
    if (tmp.n == 2 * T - 1) {
     Split(x, i, tmp);
     if (k > x.key[i]) {
      i++;
     }
    }
    _Insert(x.child[i], k);
   }

}

public void Show() {
  Show(root);
}

private void Remove(Node x, int key) {
  int pos = x.Find(key);
  if (pos != -1) {
    if (x.leaf) {
      int i = 0;
      for (i = 0; i < x.n && x.key[i] != key; i++) {
      }
      ;
      for (; i < x.n; i++) {
        if (i != 2 * T - 2) {
          x.key[i] = x.key[i + 1];
        }
      }
      x.n--;
      return;
    }
    if (!x.leaf) {

      Node pred = x.child[pos];
      int predKey = 0;
      if (pred.n >= T) {
        for (;;) {
          if (pred.leaf) {
```

```
      System.out.println(pred.n);
      predKey = pred.key[pred.n - 1];
      break;
    } else {
      pred = pred.child[pred.n];
    }
  }
  Remove(pred, predKey);
  x.key[pos] = predKey;
  return;
}

Node nextNode = x.child[pos + 1];
if (nextNode.n >= T) {
  int nextKey = nextNode.key[0];
  if (!nextNode.leaf) {
    nextNode = nextNode.child[0];
    for (;;) {
      if (nextNode.leaf) {
        nextKey = nextNode.key[nextNode.n - 1];
        break;
      } else {
        nextNode = nextNode.child[nextNode.n];
      }
    }
  }
  Remove(nextNode, nextKey);
  x.key[pos] = nextKey;
  return;
}

int temp = pred.n + 1;
pred.key[pred.n++] = x.key[pos];
for (int i = 0, j = pred.n; i < nextNode.n; i++) {
  pred.key[j++] = nextNode.key[i];
  pred.n++;
}
for (int i = 0; i < nextNode.n + 1; i++) {
  pred.child[temp++] = nextNode.child[i];
}

x.child[pos] = pred;
for (int i = pos; i < x.n; i++) {
  if (i != 2 * T - 2) {
    x.key[i] = x.key[i + 1];
  }
}
for (int i = pos + 1; i < x.n + 1; i++) {
```

```
      if (i != 2 * T - 1) {
        x.child[i] = x.child[i + 1];
      }
    }
    x.n--;
    if (x.n == 0) {
      if (x == root) {
        root = x.child[0];
      }
      x = x.child[0];
    }
    Remove(pred, key);
    return;
  }
 } else {
  for (pos = 0; pos < x.n; pos++) {
    if (x.key[pos] > key) {
      break;
    }
  }
  Node tmp = x.child[pos];
  if (tmp.n >= T) {
    Remove(tmp, key);
    return;
  }
  if (true) {
    Node nb = null;
    int devider = -1;

    if (pos != x.n && x.child[pos + 1].n >= T) {
      devider = x.key[pos];
      nb = x.child[pos + 1];
      x.key[pos] = nb.key[0];
      tmp.key[tmp.n++] = devider;
      tmp.child[tmp.n] = nb.child[0];
      for (int i = 1; i < nb.n; i++) {
        nb.key[i - 1] = nb.key[i];
      }
      for (int i = 1; i <= nb.n; i++) {
        nb.child[i - 1] = nb.child[i];
      }
      nb.n--;
      Remove(tmp, key);
      return;
    } else if (pos != 0 && x.child[pos - 1].n >= T) {

      devider = x.key[pos - 1];
      nb = x.child[pos - 1];
```

```
  x.key[pos - 1] = nb.key[nb.n - 1];
  Node child = nb.child[nb.n];
  nb.n--;

  for (int i = tmp.n; i > 0; i--) {
    tmp.key[i] = tmp.key[i - 1];
  }
  tmp.key[0] = devider;
  for (int i = tmp.n + 1; i > 0; i--) {
    tmp.child[i] = tmp.child[i - 1];
  }
  tmp.child[0] = child;
  tmp.n++;
  Remove(tmp, key);
  return;
 } else {
  Node lt = null;
  Node rt = null;
  boolean last = false;
  if (pos != x.n) {
    devider = x.key[pos];
    lt = x.child[pos];
    rt = x.child[pos + 1];
  } else {
    devider = x.key[pos - 1];
    rt = x.child[pos];
    lt = x.child[pos - 1];
    last = true;
    pos--;
  }
  for (int i = pos; i < x.n - 1; i++) {
    x.key[i] = x.key[i + 1];
  }
  for (int i = pos + 1; i < x.n; i++) {
    x.child[i] = x.child[i + 1];
  }
  x.n--;
  lt.key[lt.n++] = devider;

  for (int i = 0, j = lt.n; i < rt.n + 1; i++, j++) {
    if (i < rt.n) {
      lt.key[j] = rt.key[i];
    }
    lt.child[j] = rt.child[i];
  }
  lt.n += rt.n;
  if (x.n == 0) {
    if (x == root) {
```

```
          root = x.child[0];
        }
        x = x.child[0];
      }
      Remove(lt, key);
      return;
    }
  }
 }
}

public void Remove(int key) {
 Node x = Search(root, key);
 if (x == null) {
  return;
 }
 Remove(root, key);
}

public void Task(int a, int b) {
 Stack<Integer> st = new Stack<>();
 FindKeys(a, b, root, st);
 while (st.isEmpty() == false) {
  this.Remove(root, st.pop());
 }
}

private void FindKeys(int a, int b, Node x, Stack<Integer> st) {
 int i = 0;
 for (i = 0; i < x.n && x.key[i] < b; i++) {
  if (x.key[i] > a) {
   st.push(x.key[i]);
  }
 }
 if (!x.leaf) {
  for (int j = 0; j < i + 1; j++) {
   FindKeys(a, b, x.child[j], st);
  }
 }
}

public boolean Contain(int k) {
 if (this.Search(root, k) != null) {
  return true;
 } else {
  return false;
 }
}
```

```
  // Show the node
  private void Show(Node x) {
    assert (x == null);
    for (int i = 0; i < x.n; i++) {
      System.out.print(x.key[i] + " ");
    }
    if (!x.leaf) {
      for (int i = 0; i < x.n + 1; i++) {
        Show(x.child[i]);
      }
    }
  }

  public static void main(String[] args) {
    BTree b = new BTree(3);
    b.Insert(8);
    b.Insert(9);
    b.Insert(10);
    b.Insert(11);
    b.Insert(15);
    b.Insert(20);
    b.Insert(17);

    b.Show();

    b.Remove(10);
    System.out.println();
    b.Show();
  }
}
```