Universitat Politècnica De Catalunya

Barcelona Tech - UPC

Advanced Data Structures

Assignment

External Memory: B-Trees

Author:
Leonardo Menti

June 2022

Contents

1 Introduction

A binary search tree is a tree in which the values of the children of a node are ordered, usually having values lower than those of the starting node in the children on the left and higher values in the children on the right. B-Trees are data structure that generalize the idea of binary search trees, allowing nodes to have more than two children and more than one representing element. Each key belonging to the left subtree of a node has a lower value than any key belonging to the subtree to its right; moreover, their structure guarantees their balance: for each node, the heights of the right and left subtree differ by more than one unit. This is the main advantage of the B-tree, and allows you to perform insertion, deletion and search operations in logarithmically amortized times. B-Trees are commonly used in databases and filesystems.

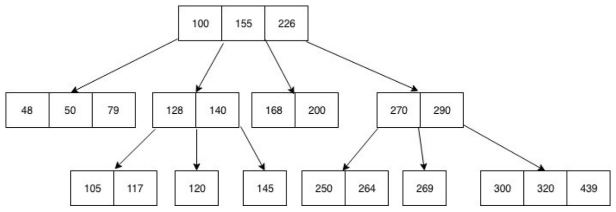


Figure 1: An example of B-Tree

B-Tree properties

- For each node x, the keys are stored in increasing order
- In each node, there is a boolean value x.leaf which is true if x is a leaf
- If n is the order of the tree, each internal node can contain at most n - 1 keys along with a pointer to each child
- Each node except root can have at most n children and at least n/2 children
- The root has at least 2 children and contains a minimum of 1 key

Insert and Split operations In this work we're going to analyze the performance and features of the B Trees after the insertion of elements, so this is a quick introduction to the operations of *insert* and the related *split*.

Inserting a new key can be more difficult than the same procedure for a binary tree as it's essential to keep the tree balanced. A preliminary operation in order to create a function for inserting a key in a B-Tree is the division operation of a node. A node of a B-Tree is defined full if it contains exactly t-1 keys, where t is the order of the tree. The division operation is performed at the median key of the full node. After the split, the solid node is split into two different nodes each with (t-1)/2 keys. Specifically, the median key of node y is moved to the parent of node y. The operation of inserting a key is carried out thanks to a visit to the tree. After searching the appropriate node for the insertion, if the node is full the new key is inserted in increasing order and then the split operation is performed.

2 Implementation

I chose to implement the B-Tree data structure in Java. I produced a single class named `BTree` with the following private attributes:

- `private int B` : order of the B-Tree
- `private Node root` : root node of the B-Tree
- `private int height` : height of the B-Tree
- `private int nsplit` : number of split made by the B-Tree

When we construct a new B-Tree we can pass the parameter to set the order. At the beginning the height is 1, the number of split is obviously 0 and we construct a new Node as the root one. `Node` is an inner class inside `BTree`, that represent a generic node of the B-Tree. A Node can contain at most *2*B-1* keys and *2*B* children. I chose to duplicate the order of the tree because it's easier to perform the split operation if the number of keys in a full node is always even. Going back to the main class `BTree`, I have implemented the following functions (not mentioning getter and setters):

- `private void split(Node x, int pos, Node y)`
- `public void insert(final int key)`
- `private void insertValue(Node x, int k)`

Talking about the `Node` inner class I have implementend the following fields and methods:

- `private int n`
- `private boolean leaf`
- `private final int[] keys = new int[2 * B - 1]`
- `private final Node[] children = new Node[2 * B]`
- `public void insertChild(int i, Node newNode)`

3 Experiments

One of the main reason of using B tree is its capability to store large number of keys in a single node and large key values by keeping the height of the tree relatively small. I analyzed the relationship between the order of the tree and other characteristics after the insertion of a lot of elements. In order to do it, I followed the same schema for each of the three experiments that I made: for each one I created B trees of different order and for each of them I inserted a great amount of numbers and, after all the insertions, I analyzed the features of my interests. I chose to perform two kinds of insertions in order to see the different behaviour of the tree: the first is the insertion of a huge amount of random numbers in a specified range, while the second is the insertion of the same amount of numbers, but as a strictly growing sequence.

3.1 Relationship between the order and the height

In order to make this experiments I created the script `HeightExperiment.java`. The script creates 100 trees with orders from 2 to 100 (remember, as mentioned in section 2, that I duplicate the order of the tree in order to easy perform the split operation). For each tree I insert a great amount of numbers and then I store the results in a txt file to analyze in a second moment. I used the `height` field to track this information: the height at the moment of the creation of the tree is equal to 1 and then I increment it everytime the root node is full and we perform the split of it.

3.2 Relationship between the order and the time

I created the script `TimeExperiment.java`. As before, the script creates 100 trees with orders from 2 to 100 and inserts numbers in each of them. In order to analyze the time, I calculate the delta for each tree from the insertion of the first element and the insertion of the last element. I saved the output in a txt file to analyze later.

3.3 Relationship between the order and the number of splits

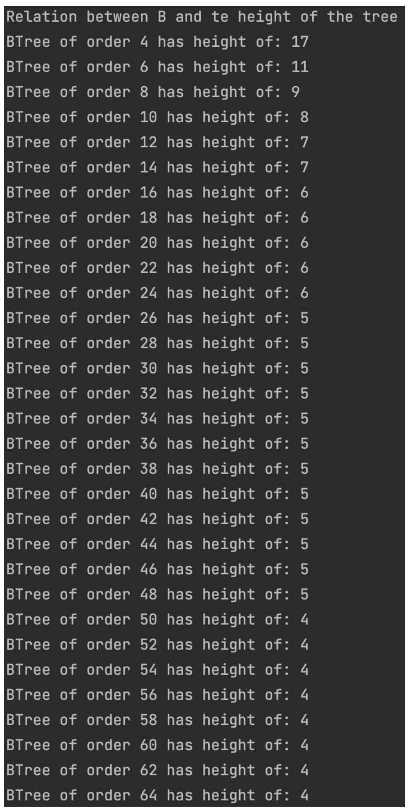
I created the script `SplitExperiment.java`. As before, the script creates 100 trees with orders from 2 to 100 and inserts numbers in each of them. I used the `nsplit` field in order to keep track of the number of the splits. At the moment of the creation of the tree the number of splits is equal to 0 and then I increment it everytime the `split` function is called. I saved the result in a txt file.

4 Results

In this section I'm going to show the result I obtained from all the experiments I made. Having performed the experiments with in two different ways (inserting random numbers and a sequence of numbers), I produced for each experiment a plot that compare the results.

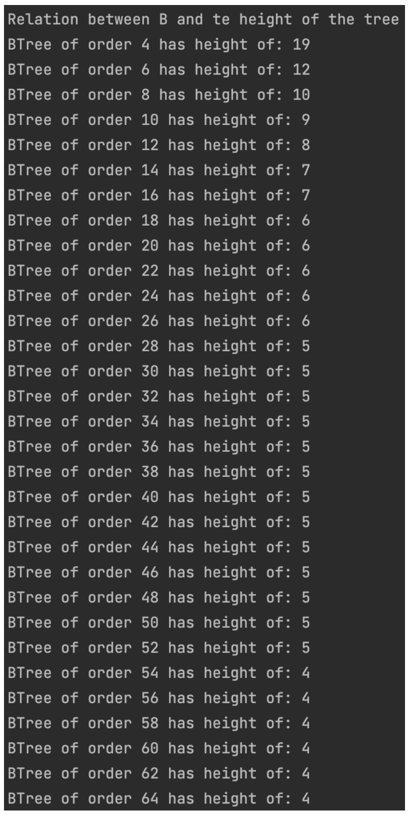
4.1 Relationship between the order and the height

As said before, this script creates 100 trees of different order and, for each one, inserts 1000000 elements. The first time the script inserts random numbers (from 0 to 500), while the second time it inserts the sequence from 0 to 1000000. We can see from Figure 2a and Figure 2b that there aren't big differences between the two situations: a B tree of order x in which we insert random values grows at the same way (in terms of height) of a B tree of the same order in which we insert a sequence of numbers.



```
Relation between B and te height of the tree
BTree of order 4 has height of: 17
BTree of order 6 has height of: 11
BTree of order 8 has height of: 9
BTree of order 10 has height of: 8
BTree of order 12 has height of: 7
BTree of order 14 has height of: 7
BTree of order 16 has height of: 6
BTree of order 18 has height of: 6
BTree of order 20 has height of: 6
BTree of order 22 has height of: 6
BTree of order 24 has height of: 6
BTree of order 26 has height of: 5
BTree of order 28 has height of: 5
BTree of order 30 has height of: 5
BTree of order 32 has height of: 5
BTree of order 34 has height of: 5
BTree of order 36 has height of: 5
BTree of order 38 has height of: 5
BTree of order 40 has height of: 5
BTree of order 42 has height of: 5
BTree of order 44 has height of: 5
BTree of order 46 has height of: 5
BTree of order 48 has height of: 5
BTree of order 50 has height of: 4
BTree of order 52 has height of: 4
BTree of order 54 has height of: 4
BTree of order 56 has height of: 4
BTree of order 58 has height of: 4
BTree of order 60 has height of: 4
BTree of order 62 has height of: 4
BTree of order 64 has height of: 4
```

(a) Random



```
Relation between B and te height of the tree
BTree of order 4 has height of: 19
BTree of order 6 has height of: 12
BTree of order 8 has height of: 10
BTree of order 10 has height of: 9
BTree of order 12 has height of: 8
BTree of order 14 has height of: 7
BTree of order 16 has height of: 7
BTree of order 18 has height of: 6
BTree of order 20 has height of: 6
BTree of order 22 has height of: 6
BTree of order 24 has height of: 6
BTree of order 26 has height of: 6
BTree of order 28 has height of: 5
BTree of order 30 has height of: 5
BTree of order 32 has height of: 5
BTree of order 34 has height of: 5
BTree of order 36 has height of: 5
BTree of order 38 has height of: 5
BTree of order 40 has height of: 5
BTree of order 42 has height of: 5
BTree of order 44 has height of: 5
BTree of order 46 has height of: 5
BTree of order 48 has height of: 5
BTree of order 50 has height of: 5
BTree of order 52 has height of: 5
BTree of order 54 has height of: 4
BTree of order 56 has height of: 4
BTree of order 58 has height of: 4
BTree of order 60 has height of: 4
BTree of order 62 has height of: 4
BTree of order 64 has height of: 4
```

(b) Sequence

Figure 2: Height

Observations The results makes sense because B trees are self balancing and the height increments only if the root node is split. Of course the height of a single B tree decrease as the order increases, because a single node can store more elements, so we'll have less splits and so the height will be lower. From the Figure 3 we can see the comparison in a plot. The two results follows the same trend. As said before, as the order grows the height of the B tree decrease, because a single node can store more elements. Order and height are inversely proportional.

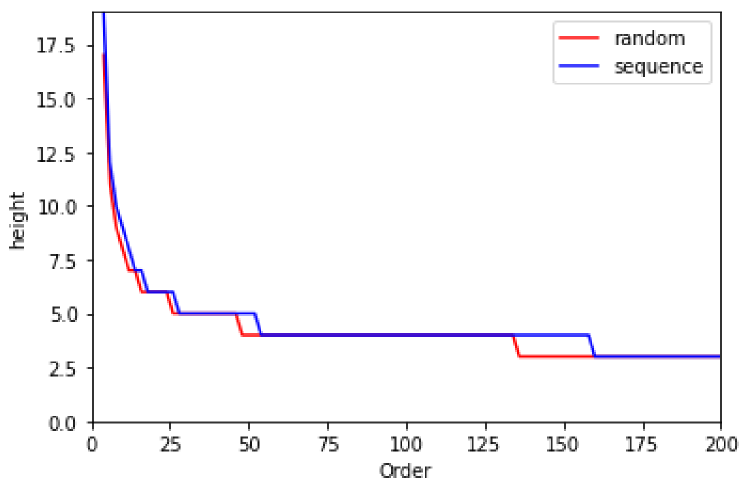


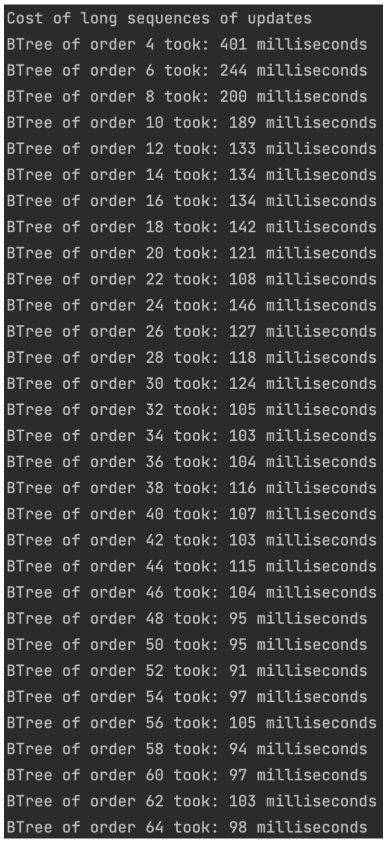
Figure 3: Relationship between the order and the height

4.2 Relationship between the order and the time

As before, this script creates 100 trees of different order and, for each one, inserts 1000000 elements. The first time with random numbers and the second time with the increasing sequence.

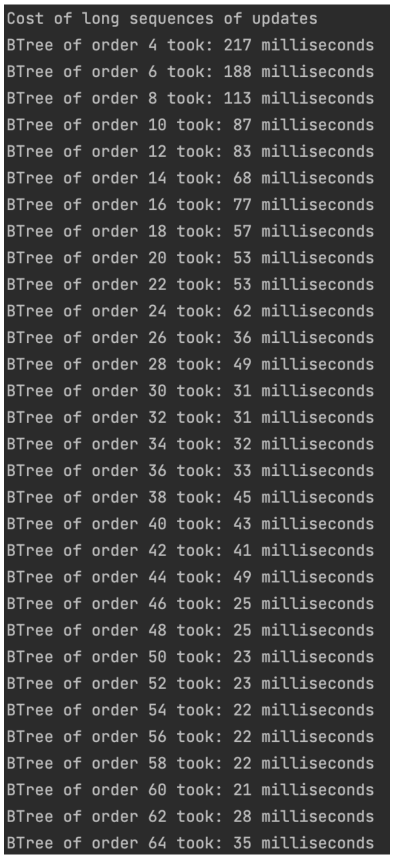
Unlike before, this time the two experiments gave different numbers: the insertion of a long sequence of values took less time that the insertion of random values. We can see from the Figure 4a and Figure 4b that the elapsed time is different and we can notice better the difference from the Figure 5.

Another interesting information is that the trend is not fully inversely proportional. We can obviously see, from Figure 5, that with low orders the insertion of the same amount of numbers took much more time, but we can also notice that sometimes a tree with a lower order took less time that a tree with a higher order.



```
Cost of long sequences of updates
BTree of order 4 took: 401 milliseconds
BTree of order 6 took: 244 milliseconds
BTree of order 8 took: 200 milliseconds
BTree of order 10 took: 189 milliseconds
BTree of order 12 took: 133 milliseconds
BTree of order 14 took: 134 milliseconds
BTree of order 16 took: 134 milliseconds
BTree of order 18 took: 142 milliseconds
BTree of order 20 took: 121 milliseconds
BTree of order 22 took: 108 milliseconds
BTree of order 24 took: 146 milliseconds
BTree of order 26 took: 127 milliseconds
BTree of order 28 took: 118 milliseconds
BTree of order 30 took: 124 milliseconds
BTree of order 32 took: 105 milliseconds
BTree of order 34 took: 103 milliseconds
BTree of order 36 took: 104 milliseconds
BTree of order 38 took: 116 milliseconds
BTree of order 40 took: 107 milliseconds
BTree of order 42 took: 103 milliseconds
BTree of order 44 took: 115 milliseconds
BTree of order 46 took: 104 milliseconds
BTree of order 48 took: 95 milliseconds
BTree of order 50 took: 95 milliseconds
BTree of order 52 took: 91 milliseconds
BTree of order 54 took: 97 milliseconds
BTree of order 56 took: 105 milliseconds
BTree of order 58 took: 94 milliseconds
BTree of order 60 took: 97 milliseconds
BTree of order 62 took: 103 milliseconds
BTree of order 64 took: 98 milliseconds
```

(a) Random



```
Cost of long sequences of updates
BTree of order 4 took: 217 milliseconds
BTree of order 6 took: 188 milliseconds
BTree of order 8 took: 113 milliseconds
BTree of order 10 took: 87 milliseconds
BTree of order 12 took: 83 milliseconds
BTree of order 14 took: 68 milliseconds
BTree of order 16 took: 77 milliseconds
BTree of order 18 took: 57 milliseconds
BTree of order 20 took: 53 milliseconds
BTree of order 22 took: 53 milliseconds
BTree of order 24 took: 62 milliseconds
BTree of order 26 took: 36 milliseconds
BTree of order 28 took: 49 milliseconds
BTree of order 30 took: 31 milliseconds
BTree of order 32 took: 31 milliseconds
BTree of order 34 took: 32 milliseconds
BTree of order 36 took: 33 milliseconds
BTree of order 38 took: 45 milliseconds
BTree of order 40 took: 43 milliseconds
BTree of order 42 took: 41 milliseconds
BTree of order 44 took: 49 milliseconds
BTree of order 46 took: 25 milliseconds
BTree of order 48 took: 25 milliseconds
BTree of order 50 took: 23 milliseconds
BTree of order 52 took: 23 milliseconds
BTree of order 54 took: 22 milliseconds
BTree of order 56 took: 22 milliseconds
BTree of order 58 took: 22 milliseconds
BTree of order 60 took: 21 milliseconds
BTree of order 62 took: 28 milliseconds
BTree of order 64 took: 35 milliseconds
```

(b) Sequence

Figure 4: Elapsed Time

Observations The information retrieved about the time don't convince me at all. Inserting an increasing sequence of numbers should make the B tree performing more splits than inserting random values. I'm going to explain better this in the section 4.3, but the general idea is that inserting increasing value forces the tree to update only its right side.

We have also to say that time information could be interesting, but not fully informative. For example when we insert the random values, we should every time count the time for the creation of the random number. I solved this problem pre-processing all the random values. Furthermore, the time depends also about the machine that run the script and this can produce different time results for the same operation did in different machines.

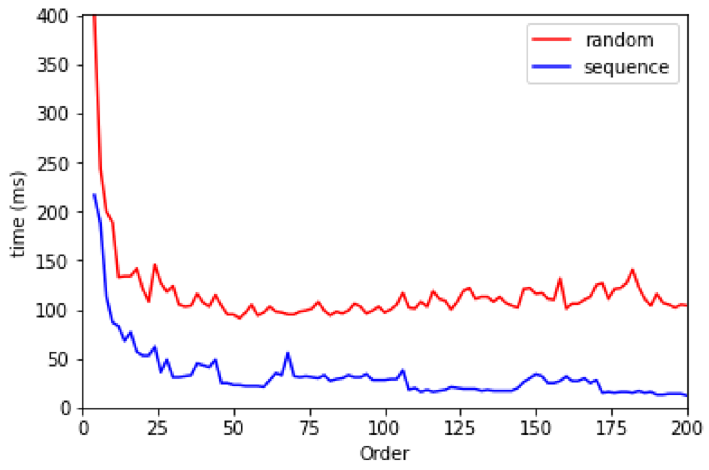


Figure 5: Relationship between the order and the time

4.3 Relationship between the order and the number of splits

The schema of the experiment is the same as the other two: 100 trees and insertion of random values and a sequence.

We can see from the Figure 7a and Figure 7b that inserting the sequence of values in the B tree let it performs much more splits than inserting the random numbers. However we can see from 6 that the trend of the number of splits in basically the same in both situations.

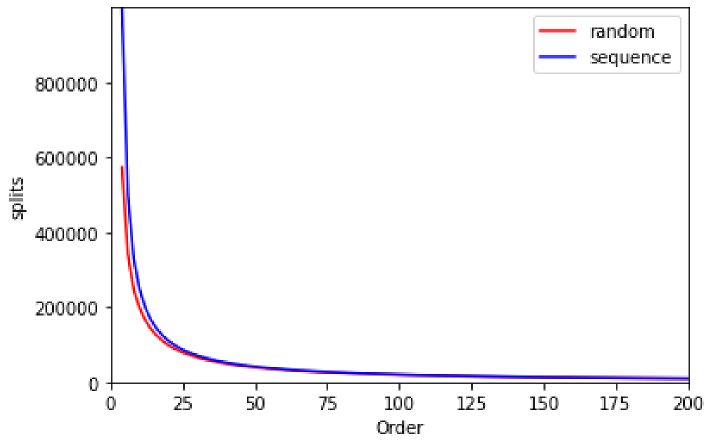
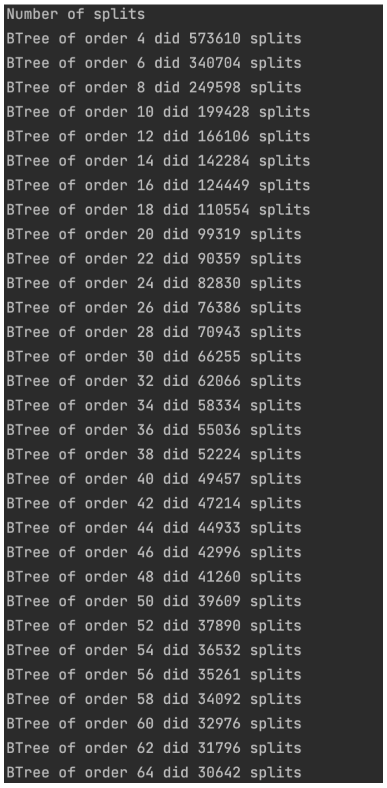
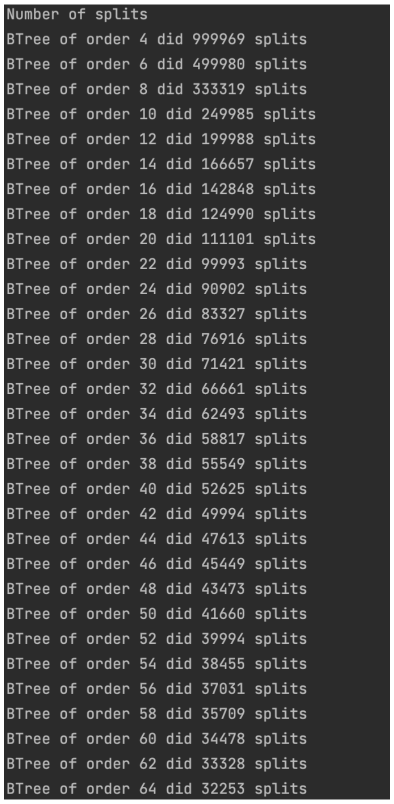


Figure 6: Relationship between the order and the number of splits



```
Number of splits
BTree of order 4 did 573610 splits
BTree of order 6 did 340704 splits
BTree of order 8 did 249598 splits
BTree of order 10 did 199428 splits
BTree of order 12 did 166106 splits
BTree of order 14 did 142284 splits
BTree of order 16 did 124449 splits
BTree of order 18 did 110554 splits
BTree of order 20 did 99319 splits
BTree of order 22 did 90359 splits
BTree of order 24 did 82830 splits
BTree of order 26 did 76386 splits
BTree of order 28 did 70943 splits
BTree of order 30 did 66255 splits
BTree of order 32 did 62066 splits
BTree of order 34 did 58334 splits
BTree of order 36 did 55036 splits
BTree of order 38 did 52224 splits
BTree of order 40 did 49457 splits
BTree of order 42 did 47214 splits
BTree of order 44 did 44933 splits
BTree of order 46 did 42996 splits
BTree of order 48 did 41260 splits
BTree of order 50 did 39609 splits
BTree of order 52 did 37890 splits
BTree of order 54 did 36532 splits
BTree of order 56 did 35261 splits
BTree of order 58 did 34092 splits
BTree of order 60 did 32976 splits
BTree of order 62 did 31796 splits
BTree of order 64 did 30642 splits
```

(a) Random



```
Number of splits
BTree of order 4 did 999969 splits
BTree of order 6 did 499980 splits
BTree of order 8 did 333319 splits
BTree of order 10 did 249985 splits
BTree of order 12 did 199988 splits
BTree of order 14 did 166657 splits
BTree of order 16 did 142848 splits
BTree of order 18 did 124990 splits
BTree of order 20 did 111101 splits
BTree of order 22 did 99993 splits
BTree of order 24 did 90902 splits
BTree of order 26 did 83327 splits
BTree of order 28 did 76916 splits
BTree of order 30 did 71421 splits
BTree of order 32 did 66661 splits
BTree of order 34 did 62493 splits
BTree of order 36 did 58817 splits
BTree of order 38 did 55549 splits
BTree of order 40 did 52625 splits
BTree of order 42 did 49994 splits
BTree of order 44 did 47613 splits
BTree of order 46 did 45449 splits
BTree of order 48 did 43473 splits
BTree of order 50 did 41660 splits
BTree of order 52 did 39994 splits
BTree of order 54 did 38455 splits
BTree of order 56 did 37031 splits
BTree of order 58 did 35709 splits
BTree of order 60 did 34478 splits
BTree of order 62 did 33328 splits
BTree of order 64 did 32253 splits
```

(b) Sequence

Figure 7: Number of splits

Observations Looking at the Figure 6, we confirm what we said before about the difference in number of splits. The blue line start with higher values but then, from a certain order, the trend is more or less the same. It makes sense because if the order is very high the tree will perform less splits in both cases, because the nodes will be able to store much more elements before being split. Order and number of splits are inversely proportional.

Looking again at Figure 7a and Figure 7b we see that the number of splits is much higher when we insert the strictly increasing sequence instead of the random values. It makes sense because, using a sequence, the tree is going to insert the numbers always in its right side and the number of splits increases. Inserting a increasing sequence of number let the tree use only its right side and never the left side (nodes on the left). On the other hand, when we insert random values the key could be stored in any side of the tree, so we can perform a lower number of splits in total.

5 Conclusion

To sum up, I've implemented a Java version of **B Trees**, with operations of `insert` and `split`. I created three different scripts in order to analyze some features of the tree in relation to its order. Being specifically, I analyzed the relationship between the order and the height, the order and the time and the order and the number of splits. In order to make the analysis, for each experiment I created different B Trees of different orders and for each of them I inserted a great number of values. After all the insertions I retrieved the information that I wanted. I repeated every experiment twice, the first time inserting random values and the second time inserting a strictly increasing sequence of numbers. In order to generate the plot, I stored the result of the experiments scripts and then I created a python notebook that generates the plots with the previous results.

The result I obtained makes sense with the data structure I'm analyzing. As the order grows the height and the number of splits goes down, because they are inversely proportional. Usually inserting a sequence let the tree perform worst than inserting random values. The only complicated topic was the comparison in time between the sequence and the random values, because I was expecting an higher time for the sequence instead of the random numbers, due to the highest number of splits.

6 Appendix

6.1 BTree.java

```java
public class BTree {

  private final int B;
  private Node root;
  private int height;
  private int n_split;

  public BTree(int b) {
    B = b;
    root = new Node(0, true);
    height = 1;
    n_split = 0;
  }

  public int getHeight() { return height; }
  public int getN_split() { return n_split; }

  // Splitting the node
  private void split(Node x, int pos, Node y) {
    n_split++;
    Node z = new Node(B-1, y.leaf);
    if (B - 1 >= 0) System.arraycopy(y.keys, 0 + B, z.keys, 0, B - 1);

    if (!y.leaf) {
      if (B >= 0) System.arraycopy(y.children, 0 + B, z.children, 0, B);
    }
    y.n = B - 1;
    if (x.n + 1 - (pos + 1) >= 0) System.arraycopy(x.children, pos + 1,
        x.children, pos + 1 + 1, x.n + 1 - (pos + 1));

    x.children[pos + 1] = z;

    if (x.n - pos >= 0) System.arraycopy(x.keys, pos, x.keys, pos + 1,
        x.n - pos);

    x.keys[pos] = y.keys[B - 1];
    x.n = x.n + 1;
  }

  // Inserting a value
  public void insert(int key) {
    Node curr = this.root;
    if (curr.n == 2 * B - 1) {
      Node s = new Node(0, false);
      root = s;
      this.height++;
      s.insertChild(0, curr);
      split(s, 0, curr);
      insertValue(s, key);
    }
    else {
      insertValue(curr, key);
    }
  }

  // Insert a value inside the node
  private void insertValue(Node x, int k) {
    if (x.leaf) {
      int i = 0;
      for (i = x.n - 1; i >= 0 && k < x.keys[i]; i--)
        x.keys[i + 1] = x.keys[i];
      x.keys[i + 1] = k;
      x.n = x.n + 1;
    }
    else {
      int i = 1;
      Node tmp = x.children[i];
      if (tmp.n == 2 * B - 1) {
        split(x, i, tmp);
        if (k > x.keys[i])
          i++;
      }
      insertValue(x.children[i], k);
    }
  }

  // Node
  public class Node {

    private int n;
    private final boolean leaf;
    private final int[] keys = new int[2 * B - 1];
    private final Node[] children = new Node[2 * B];

    public Node(int n, boolean leaf) {
      this.n = n;
      this.leaf = leaf;
    }

    public void insertChild(int i, Node newNode) {
      children[i] = newNode;
    }
  }
}
```

6.2 HeightExperiment.java

```java
import java.io.File;
import java.util.Random;
import java.io.FileWriter;

public class HeightExperiment {
    public static void main(String[] args) {

        System.out.println("Relation between B and te height of the
            tree");
        try {
            File myObj = new File("height_results.txt");
            //File myObj = new File("height_results2.txt");
            FileWriter myWriter = new FileWriter(myObj.getName());

            for(int i=2;i<101;i++){
                BTree btree = new BTree(i);
                for(int j=0;j<1000000;j++){
                    int x = new Random().nextInt(500);
                    btree.insert(x);
                }
                System.out.println("BTree of order " + i*2 + " has height
                    of: " + btree.getHeight());

                myWriter.write(i*2 + ":" + btree.getHeight() + "\n");
            }
            myWriter.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6.3 TimeExperiment.java

```java
import java.io.File;
import java.io.FileWriter;
import java.time.Duration;
import java.time.Instant;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class TimeExperiment {
    public static void main(String[] args) {

        System.out.println("\nCost of long sequences of updates");
        try {
            File myObj = new File("time_results.txt");
            //File myObj = new File("time_results2.txt");
            FileWriter myWriter = new FileWriter(myObj.getName());

            int[] values = new int[1000000];
            int[] sequence = new int[1000000];

            for(int j=0;j<1000000;j++)
                values[j] = new Random().nextInt(500);

            for(int j=0;j<1000000;j++)
                sequence[j] = j;

            for(int i=2;i<101;i++){
                BTree btree = new BTree(i);
                Instant start = Instant.now();
                for(int j=0;j<1000000;j++)
                    btree.insert(values[j]);

                Instant end = Instant.now();
                Duration timeElapsed = Duration.between(start, end);
                System.out.println("BTree of order " + i*2 + " took: "+
                     timeElapsed.toMillis() +" milliseconds");

                myWriter.write(i*2 + ":" + timeElapsed.toMillis() + "\n");
            }
            myWriter.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6.4 SplitExperiment.java

```java
import java.io.File;
import java.util.Random;
import java.io.FileWriter;

public class SplitExperiment {
    public static void main(String[] args) {

        System.out.println("\nNumber of splits");
        try {
            File myObj = new File("split_results.txt");
            //File myObj = new File("split_results2.txt");
            FileWriter myWriter = new FileWriter(myObj.getName());

            for(int i=2;i<101;i++){
                BTree btree = new BTree(i);
                for(int j=0;j<1000000;j++){
                    int x = new Random().nextInt(1000000);
                    btree.insert(x);
                }
                System.out.println("BTree of order " + i*2 + " did "+
                     btree.getN_split() +" splits");

                myWriter.write(i*2 + ":" + btree.getN_split() + "\n");
            }
            myWriter.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

6.5 plot.ipynb

```
import matplotlib.pyplot as plt

orders = list(range(4,201,2))

def get_axes(filename):

    file = open(filename, "r")

    ys = []
    for row in file.readlines():
        x,y = row.split(':')
        ys.append(int(y[:-1]))
    return ys

def plot_results(filename1, filename2, y_name):

    ys1 = get_axes(filename1)
    ys2 = get_axes(filename2)

    ax = plt.gca()
    ax.set_xlim([0, 200])
    ax.set_ylim([0, max(max(ys1), max(ys2))])
    plt.xlabel("Order")
    plt.ylabel(y_name)
    plt.plot(orders, ys1, color='red', label='random')
    plt.plot(orders, ys2, color='blue', label='sequence')
    plt.legend(loc="upper right")

plot_results("time_results.txt", "time_results2.txt", 'time (ms)')

plot_results("height_results.txt", "height_results2.txt", 'height')

plot_results("split_results.txt", "split_results2.txt", 'splits')
```