B-Trees

Lawrence Cabbabe, Aaron Levesque, Ben Uthoff,
Sam Zhong

What are B-Trees?

Definition:

A balanced tree data structure used for organizing and storing data.

They allow for searching, insertion, and deletions under logarithmic timing.

Key Features:

- Balanced Structure
- Node Structure
- Degree or Order
- Sorted Data

A Brief History

- B-tree first introduced in Rudolf Bayer and Edward M. McCreight's paper in 1970 named "Organization and Maintenance of Large Ordered Indexes".

- B-Tree was made to address the limitations of binary search trees for large datasets, and to optimize storage and retrieval in secondary storage systems.

- Between 1970s - 1980s, IBM adopted B-trees in their R database management system popularizing them.
- In modern day, B-tree is a fundamental data structure for indexing and storage.
  - B+ Trees are directly evolved and are widely used.
  - Used with hash indexes, LSM trees, to create secured storage.

How B–Tree works (Technical description)

B-Trees are structured with nodes like Binary Search trees.

Each node within the B-Tree can have multiple keys and child pointers.
(Unlike Binary Search Trees which are limited to 2)

Starting with the root node, it contains at least one key.

Internal Nodes contain between T - 1 and 2T - 1 keys in a sorted order.
Internal Node's child pointers are also within the key ranges.

Leaf Nodes store the data and also contain between T - 1 and 2T - 1 keys.

How B-Tree works (Technical cont.)

Search

1. Starts at the root
2. Traverses internal nodes by comparing target key and node's keys.
3. Follows the child pointers based on the key ranges.
4. Continues comparing keys until reaching leaf node.
5. Search for Target Key in Leaf Node

Insert

1. Search the tree comparing keys to place the new key.
2. Check if Leaf node has Space. (Insert if space)
3. If full, split the node, move the middle key to the parent node.
4. Create a new child node with keys greater than the middle.
5. Ensure parent nodes are updated correctly.

Delete

1. Search tree like insert.
2. If key is leaf, delete.
   a. If leaf goes beneath min. key amount, merge or distribute.
3. If found in internal node, replace with predecessor/successor and delete it. (check key amt. again).
4. Ensure parent nodes are updated correctly.

How B-Tree works (Visual depiction)

Max Degree of 3

Our Code

We have 4 unique sets of test cases to show different uses of the program.

```
192.168.23.45
10.42.57.89
172.31.14.208
192.168.76.102
10.128.39.74
172.16.88.33
192.168.200.17
10.73.91.245
172.29.150.62
192.168.112.78
10.51.77.193
172.20.45.221
192.168.5.36
10.19.60.124
172.30.18.77
192.168.99.200
10.221.87.44
172.17.5.168
192.168.33.79
10.66.124.55
172.28.240.13
192.168.72.221
10.91.13.78
172.18.56.104
192.168.150.200
10.38.91.17
172.19.37.82
192.168.2.145
10.11.68.29
172.21.92.176
192.168.115.60
10.254.33.77
172.27.15.98
192.168.220.41
10.84.5.209
172.22.77.34
192.168.44.123
10.145.200.8
172.26.64.55
192.168.170.92
10.99.21.174
172.23.180.37
192.168.88.66
10.31.110.55
172.25.13.245
```

```
John_McMann.txt
Emily_Roberts.doc
Carlos_Santos.pdf
Lisa_Anderson.jpg
Ryan_Wilson.xls
Natasha_Collins.png
Benjamin_Taylor.ppt
Olivia_Cooper.txt
Marcus_Jackson.docx
Mia_Lopez.jpeg
Nathan_Hughes.csv
Victoria_Martin.pdf
Ethan_Wright.mp3
Sophia_Campbell.mp4
Daniel_Hill.xlsx
Ava_Garcia.png
Matthew_Ross.txt
Lily_Fisher.doc
Christopher_Ward.pptx
Grace_Kelly.jpg
Tyler_Cooper.wav
Hailey_Brown.avi
Dylan_Perez.json
Zoe_Johnson.xml
Sebastian_Mitchell.log
Aria_Moore.ini
Joshua_Hall.cfg
Chloe_Watson.ods
Nicholas_King.ods
Mia_Williams.rtf
Lucas_Young.ps
Addison_Robinson.sql
Ethan_Barnes.bmp
```

```
event
poet
classroom
activity
protection
moment
baseball
mixture
possession
internet
interaction
song
wood
satisfaction
city
country
mixture
```

```
28
91
23
45
32
49
10
1
83
62
25
84
29
```

```
Enter input file name:test_1.txt
 ------------------------
Enter degree or order:3
 ------------------------
```

or you can use program arguments with our program instead.

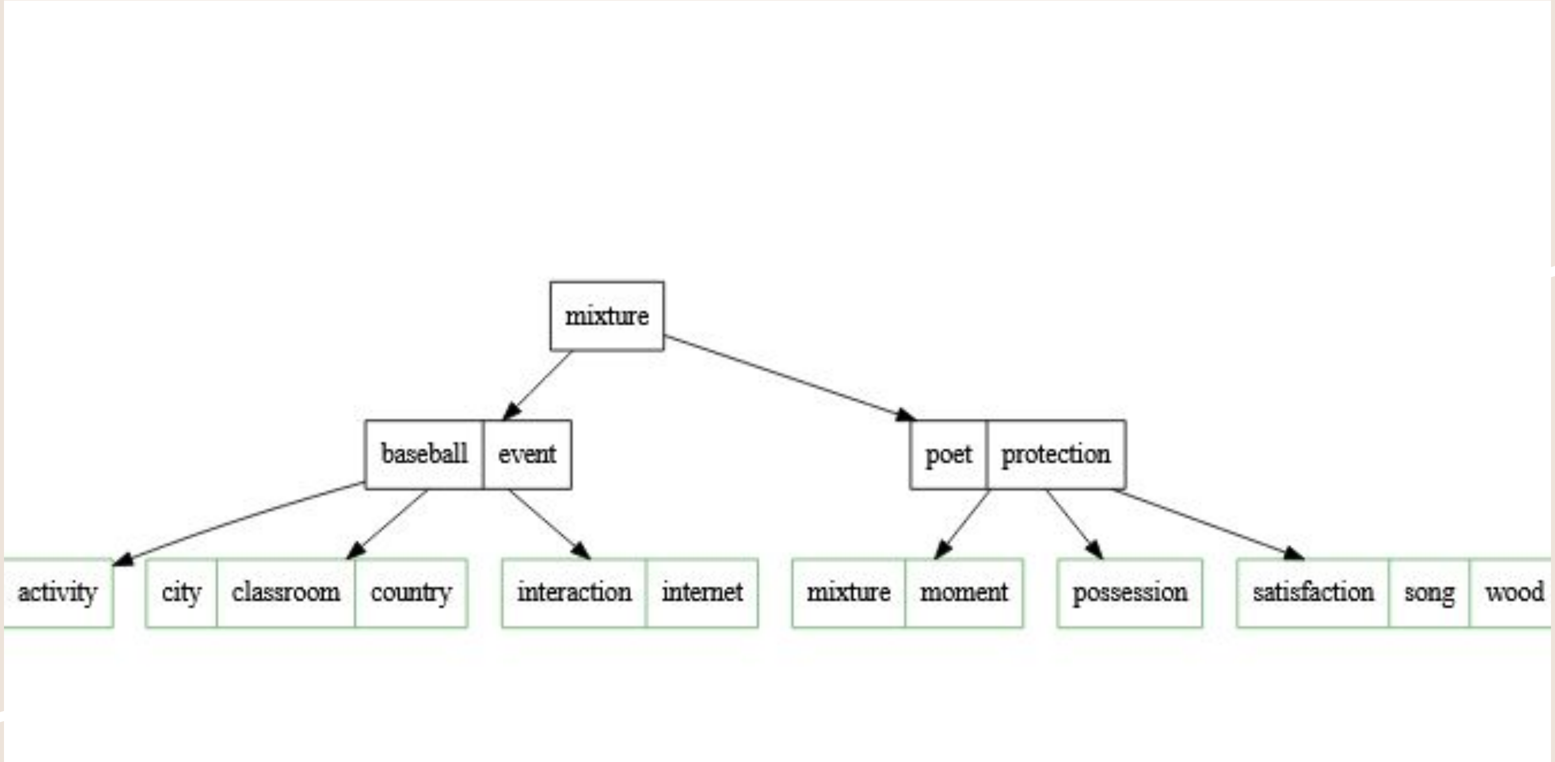
```
Hello!  Welcome to the B-Tree program.
What would you like to do?
(Enter the number of the action would like to perform)
(1) Search for a word.
(2) Insert a word.
(3) Delete a word. (Beta/Unstable)
(4) Generate a .DOT file
(0) Close the program
```

Prompted to the menu.

Our implementation uses a btree header file, btree .cpp file, and a main.cpp

Now for a live demonstration and explanation of our code.

Our Code: Dot File



Dot File visualization created after using test case 1

Time Complexity

	Worst Case	Average Case	Best Case
Search Operation	O(logn)	Θ(logn)	Ω(logn)
Insertion Operation	O(logn)	Θ(logn)	Ω(logn)
Deletion Operation	O(logn)	Θ(logn)	Ω(logn)

Pros:

- Balanced & Ordered Structure
- Versatile
- Efficient Disk usage

Cons:

- Slower Search with Small Data
- Potentially Wasted Space
- Complexity

Applications to the Real World

IBM Implementation in R Database during the 1970s-1980s.

- Popularized B-Trees and their capabilities.

Google Open Source Implementation on 2013.

- Showed increased speed and storage/memory efficiency compared to Red-Black Trees.

Linux Developers Introduction of Maple Trees for Virtual Memory Areas.

- 2021 Implementation by Liam Howlett and Matthew Wilcox based heavily of B-Trees to replace Red-Black Trees to fix memory management issues.

Questions?

FAQ:

What does the ‘B’ in ‘B-Tree’ stand for?

McCreight has said it stands for nothing! He jokingly said that the more you think about it the more you understand B-Trees.

Any others?

Sources

"B-Tree Indexes." Www.ibm.com, 20 July 2022, www.ibm.com/docs/en/informix-servers/14.10?topic=indexes-b-tree

"B*-Trees Implementation in C++." GeeksforGeeks, 30 July 2019, www.geeksforgee ks.org/b-trees-implementation-in-c/#

"B Tree in Data Structure: Learn Working of B Trees in Data Structures." EDUCBA, 14 June 2023, www.educba.com/b-tree-in-data-structure/

Bayer, R., and E. M. McCreight. "Organization and Maintenance of Large Ordered Indexes." Acta Informatica, vol. 1, no. 3, 1972, pp. 173–189, infolab.usc.edu/csci585/Spring2010/den_ar/indexing.pdf

"Deep Understanding of B-TREE Indexing." Www.linkedin.com, www.linkedin.com/pulse/deep-understanding-b-tree-indexing-sohel-rana/

"Google Code Archive - Long-Term Storage for Google Code Project Hosting." code.google.com/archive/p/cpp-btree/

"Introducing Maple Trees [LWN.net]." Lwn.net, lwn.net/Articles/845507/.

"Speedb | LSM vs B-Tree." Www.speedb.io, www.speedb.io/blog-posts/02-lsm-vs-b-tr ee-v2.

Our Code: Search

```
1
Enter a word to search:192.168.200.17
 -----------------------
Found the word "192.168.200.17" a total of 1 time(s).
-----------------------
```

Search simply runs the helper function

SearchHelper traverses every node trying to match the input key. It is able to do this by comparing the key to the key stored within the node leafs.

```
int BTree::search(std::string& key) {
    int count = 0;
    searchHelper( node: root,  &: key,  &: count);
    return count;
}
```

```
void BTree::searchHelper(BTreeNode* node, std::string& key, int& count) {
    // Look through each key in node.
    for (int i=0; i < node->keys.size(); i++) {
        if (key < node->keys[i]) break;
        if (key == node->keys[i]) count=count+1;
    }

    // Check if there are no children.
    if (node->is_leaf == true) return;

    // Place before the first child.
    if (key <= node->keys[0])
        searchHelper( node: node->children[0],  &: key,  &: count);
    // Place inbetween two keys in the node.
    for (int i=1; i < node->children.size()-1; i++) {
        if (key >= node->keys[i-1] && key <= node->keys[i]) {
            searchHelper( node: node->children[i],  &: key,  &: count);
        }
    }
    // place after the last child
    if (key >= node->keys[node->keys.size()-1])
        searchHelper( node: node->children[node->children.size()-1],  &: key,  &: count);

}
```

Our Code: Insert

The splitChild function splits a child node of a B-tree at a specified index within its parent. It creates a new child node, redistributes keys and children between the old and new child nodes, and updates the parent node accordingly. Additionally, it handles the case where the nodes are not leaf nodes by adjusting their child pointers.

```cpp
void BTree::insert(std::string& key) {
    // If there is no root...
    if (root == nullptr) {
        root = new BTreeNode( leaf: true);
        root->keys.push_back(key);
        return;
    }
    // If the root node is reaches max...
    if (root->keys.size() == (2 * degree - 1)) {
        BTreeNode* new_root = new BTreeNode();
        new_root->children.push_back(root);
        splitChild( parent: new_root,  index: 0);
        root = new_root;
    }
    // Recursively insert the function...
    insertNonFull( node: root,  &: key);
}
```

```
2
Welcome to the insert portal.
When finished enter 'Done'
Enter :Saul_Goodman.txt
Saul_Goodman.txt Successfully inserted.
Enter :Done
```

```cpp
void BTree::insertNonFull(BTreeNode* node, std::string& key) {
    // Start at the end of the node...
    int i = node->keys.size() - 1;
    // Go through each key until you find a place to put the new one...
    while (i >= 0 && key < node->keys[i]) {
        i--;
    }
    i++;

    // If the node is a leaf...
    if (node->is_leaf) {
        // Add the key to the node.
        node->keys.insert( position: node->keys.begin() + i,  x: key);

        // The node is not a leaf...
    } else {
        if (node->children[i]->keys.size() == (2 * degree - 1)) {
            splitChild( parent: node,  index: i);
            if (key > node->keys[i])
                i++;
        }
        insertNonFull( node: node->children[i],  &: key);
    }
}
```

The insertNonFull function inserts a key into a non-full B-tree node. It starts at the end of the node, finds the correct position for the new key by traversing existing keys, and inserts the key. If the node is not a leaf and the child at the insertion position is full, it calls the splitChild function to handle the split before recursively continuing the insertion process in the appropriate child node. The insert function serves as the entry point for inserting a key into the B-tree, handling the creation of a new root node if necessary and initiating the recursive insertion process.

```cpp
void BTree::splitChild(BTreeNode* parent, int index) {
    BTreeNode* new_child = new BTreeNode();
    BTreeNode* old_child = parent->children[index];
    parent->keys.insert( position: parent->keys.begin() + index,  x: old_child->keys[degree - 1]);
    parent->children.insert( position: parent->children.begin() + index + 1,  x: new_child);

    new_child->keys.assign( first: old_child->keys.begin() + degree,  last: old_child->keys.end());
    old_child->keys.resize( new_size: degree - 1);

    if (!old_child->is_leaf) {
        new_child->children.assign( first: old_child->children.begin() + degree,  last: old_child->children.end());
        old_child->children.resize( new_size: degree);
    }

    parent->is_leaf = false;
    if (new_child->children.size()) new_child->is_leaf = false;

}
```