B+ Tree Final Project Report

CSCI 651 Algorithm Concepts

Emily Ramkissoon (ID: 1291736), Preethi Sridhar (ID: 1282972), Nawrin Sikder (ID: 1341806) & Neelambari Mathivathanan (ID: 1345603)

Fall 2024

Dr. Tao Zhang

Due Date: December 4th, 2024

Introduction:

Databases are renowned for their versatility and efficiency in storing and retrieving data, serving as the backbone for countless applications. However, not all systems rely on traditional database management systems. This project explores an alternative approach by implementing a parts catalog system using a B+ tree data structure in Java to manage and manipulate a dataset of part records. Each record includes a Part ID and a Description, and the program provides functionalities for searching, inserting, updating, deleting, displaying the next 10 records, and retrieving statistics about the B+ Tree. The primary objective is to ensure efficient data operations and maintain data integrity while adhering to the B+ Tree's properties, such as balanced nodes and sorted order. By leveraging the B+ Tree's inherent data storage and retrieval efficiency, the project demonstrates how a database-like system can be constructed using alternative data structures, ensuring performance and scalability in a single-user environment.

This project consists of two distinct components: a B+ Tree for storing parts data and a user interface for interacting with the data. These components are designed to operate independently, ensuring that future enhancements, such as replacing the storage mechanism or modifying the interface, can be achieved with minimal effort. The application enables users to load part records from a text file into the B+ Tree and perform various operations including querying parts, updating descriptions, adding and deleting entries, and viewing sequential data. Upon exiting, the system prompts the user to save changes to the text file, preserving the updated catalog. The B+ tree used in our project is optimized for main memory operations and adheres to specific constraints, such as allowing 2 to 4 keys per index node and up to 16 records per leaf node. By the end of the program's execution, the number of splits, parent splits, fusions, parent fusions, and tree depth will be displayed which offers insights into the efficiency and behavior of the data structure.

Design:

During the project's design phase, we developed and configured six Java class implementation files. These classes include: ‘BPlusIndexNode.java’, ‘BPlusLeafNode.java’, ‘BPlusNode.java’, ‘BPlusTree.java’, ‘Main.java’, and ‘Product.java’.

The BPlusIndexNode.java class represents the nodes (excluding all leaf nodes) in the B+ tree. It holds the keys and pointers to the child nodes. Adding, removing, shifting, and merging are completed with sibling nodes in this Java class.

The BPlusLeafNode.java class exemplifies leaf nodes that store Product objects. Operations such as inserting, deleting, and sorting products are performed. The node is also checked to determine whether or not it is full or if the node can be merged with its siblings. Due to this, tree traversal is maintained in this Java class.

The BPlusNode.java class acts as a base class for other types of nodes in a B+ Tree. It provides a common structure by referencing the parent node (BPlusIndexNode). This allows child classes like BPlusIndexNode and BPlusLeafNode to inherit this functionality and maintain hierarchical relationships within the tree.

The BPlusTree.java class implements a B+ tree data structure for storing and managing Product entries efficiently. It supports operations like inserting, searching, deleting, and updating products and retrieving a list of products based on specific conditions. The tree automatically handles node splits and merges to maintain balance.

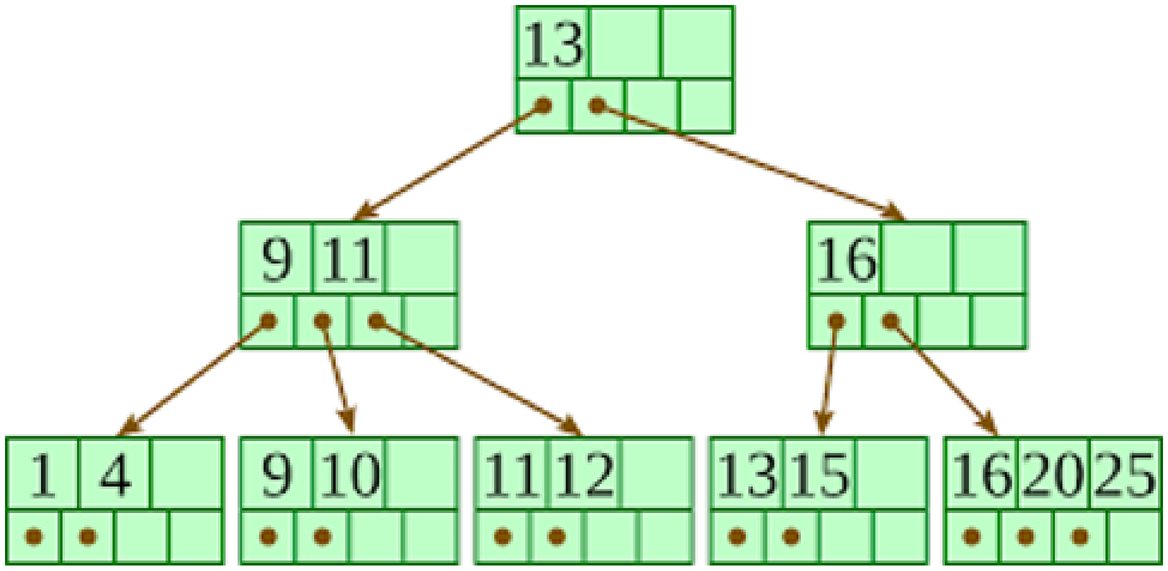
The Main.java class in the B+ tree program initializes the B+ tree in a specific order and then adds test data to demonstrate how it can be grown and split. This Java class runs searches to ensure the tree finds the correct keys and establishes if the tree is adjusted properly.

The Product.java class in a B+ tree program is used to define the structure of the data stored in the tree. It provides ways to create, update, and retrieve product details. This Java class ensures each product can work with the B+ tree, as it will utilize a key to store and organize them.

The system design revolves around a B+ Tree, which is used for indexed storage of part records. The B+ Tree ensures that data retrieval, insertion, and deletion operations are efficient with a running time of *O(log n)* complexity. The main design is interacting with the user via a menu-based interface. The key functionalities include:

1. Searching for a part by its ID.
2. Inserting new records after checking if it is unique.
3. Updating existing records.
4. Deleting records after checking for their existence.
5. Displaying the next 10 records from a given part.
6. Printing tree statistics, such as splits, fusions, and depth.

Diagram:



Source: http://www.cburch.com/cs/340/reading/btree/

Pseudocode:

The primary algorithms utilized in our B+ tree are insertion and deletion. The pseudocode for these algorithms is provided below:

For Tree Insertion

```
case 2:
   System.out.println("Enter Part ID for New Record:");
   String partID = scanner.nextLine().trim();
   if (bPlusTree.searchNode(partID) != null) {
       System.out.println("Part ID " + partID + " already exists. Insertion 
aborted. Enter a different Part ID.");
   } else {
       System.out.println("Enter Part Description for New Record:");
       description = scanner.nextLine().trim();
       bPlusTree.insertNode(partID, description);
       System.out.println("Part ID " + partID + " added successfully.");
       changesMade = true;
   }
   break;
```

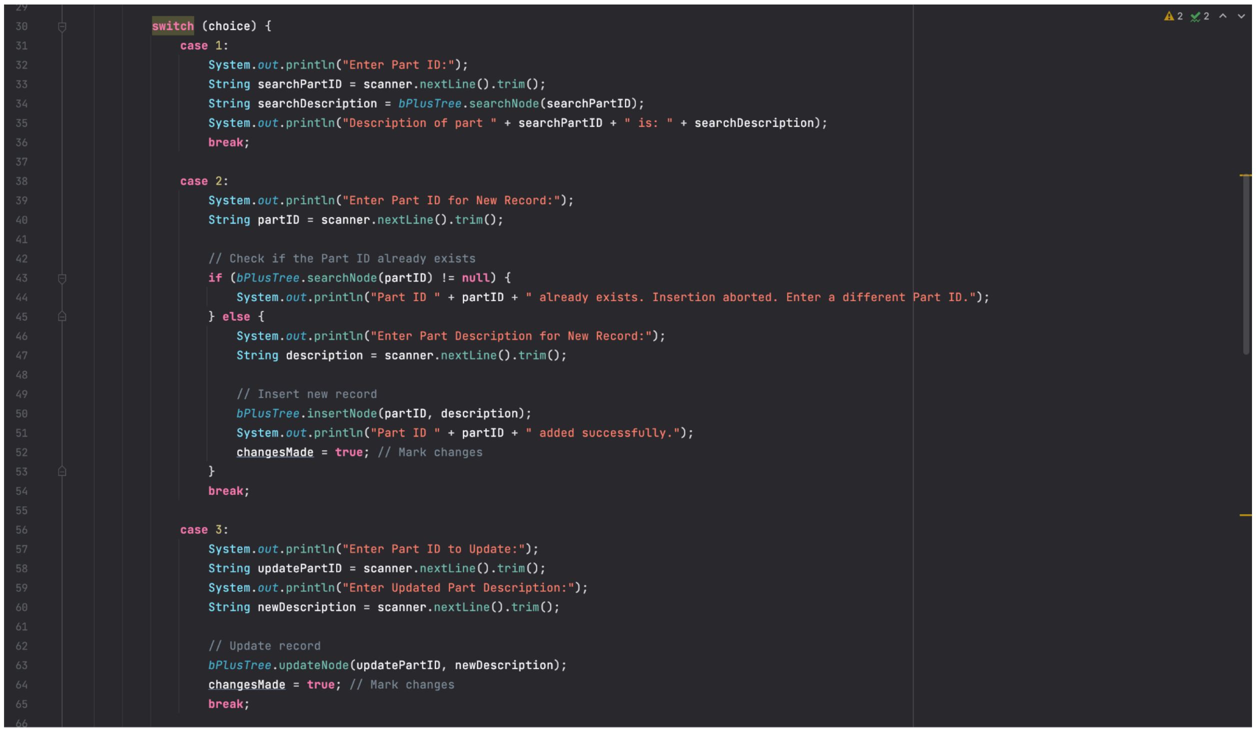
For tree insertion, under case 2, we have a printed statement "Enter Part ID for New Record." The user can enter a Part ID. The if statement shows that if the PartID already exists, it will return null and state that the "Part ID already exists. Insertion aborted. Enter a different Part ID." If the PartID does not exist, the if statement will go under the else statement. If the PartID does not already exist, it will also print out the statement "Enter the Part Description for New Record." The user will be able to insert a new record and if added successfully, they will be greeted with the statement "Part ID added successfully."

<u>For Tree Deletion</u>

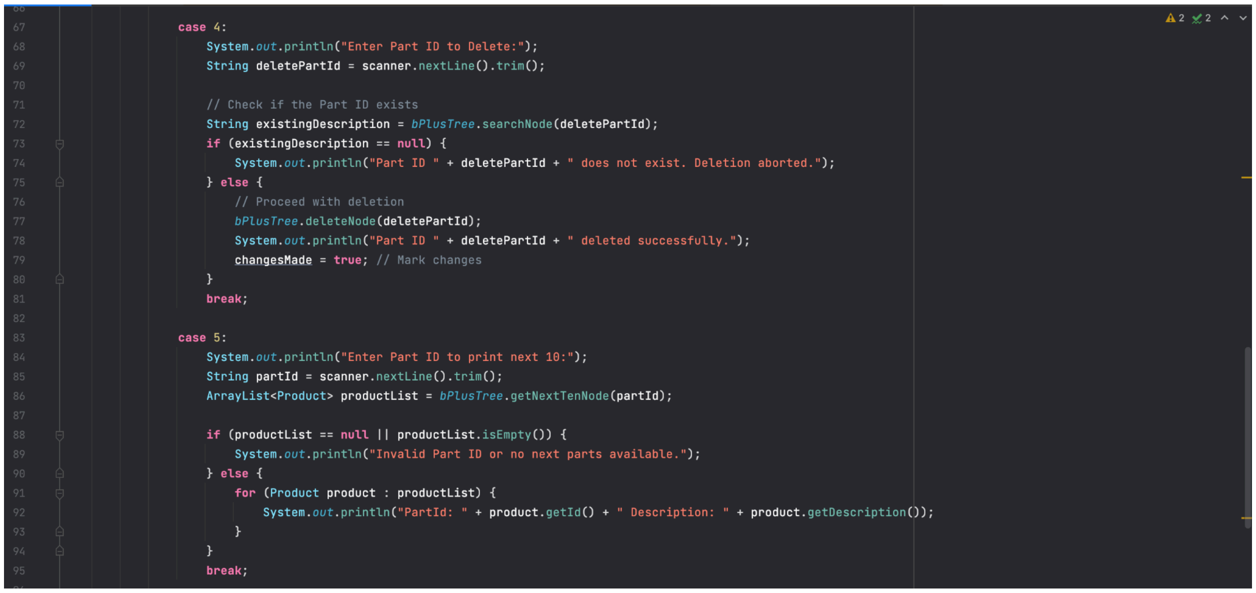
```
case 4:
    System.out.println("Enter Part ID to Delete:");
    String deletePartId = scanner.nextLine().trim();
    String existingDescription = bPlusTree.searchNode(deletePartId);
    if (existingDescription == null) {
        System.out.println("Part ID " + deletePartId + " does not exist. 
Deletion aborted.");
    } else {
        bPlusTree.deleteNode(deletePartId);
        System.out.println("Part ID " + deletePartId + " deleted 
successfully.");
        changesMade = true;
    }
    break;
```

For tree deletion, under case 4, we have a printed statement “Enter Part ID to Delete,” The user is then prompted to enter the PartID they wish to delete. The if statement shows that the deletion is aborted if the user enters a PartID that does not exist. The user is returned to the main menu, to try another option. If the PartID does exist, the deletion is executed and the user is greeted with the statement “Part ID deleted successfully.”

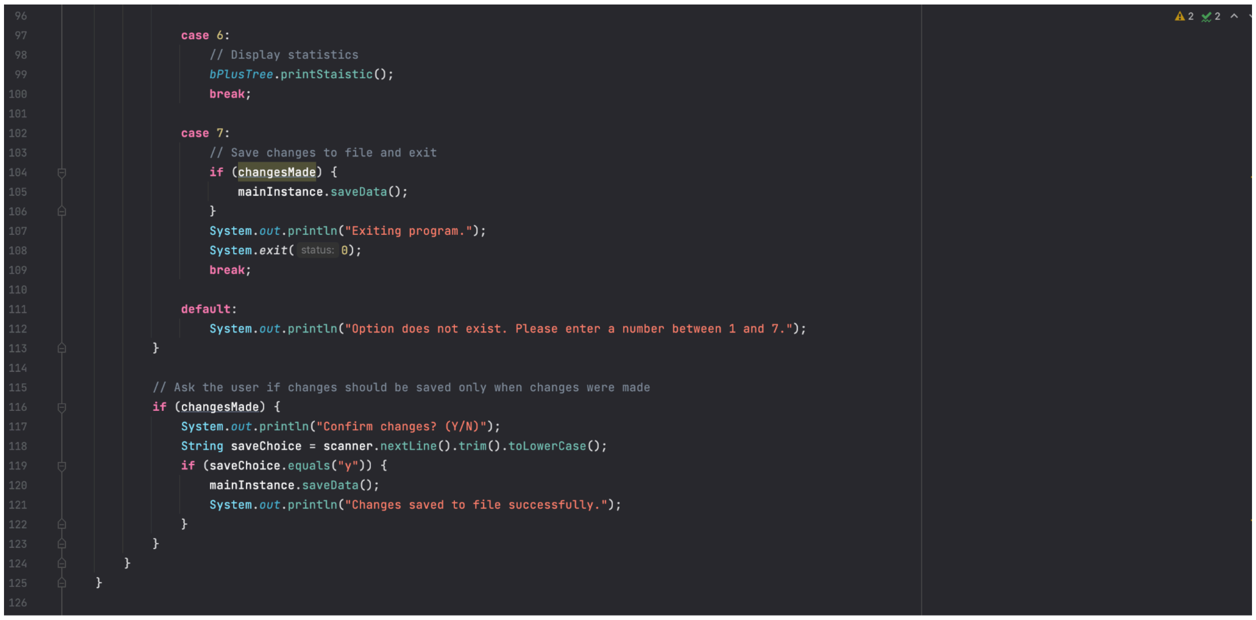
Code:



```java
        switch (choice) {
            case 1:
                System.out.println("Enter Part ID:");
                String searchPartID = scanner.nextLine().trim();
                String searchDescription = bPlusTree.searchNode(searchPartID);
                System.out.println("Description of part " + searchPartID + " is: " + searchDescription);
                break;

            case 2:
                System.out.println("Enter Part ID for New Record:");
                String partID = scanner.nextLine().trim();

                // Check if the Part ID already exists
                if (bPlusTree.searchNode(partID) != null) {
                    System.out.println("Part ID " + partID + " already exists. Insertion aborted. Enter a different Part ID.");
                } else {
                    System.out.println("Enter Part Description for New Record:");
                    String description = scanner.nextLine().trim();

                    // Insert new record
                    bPlusTree.insertNode(partID, description);
                    System.out.println("Part ID " + partID + " added successfully.");
                    changesMade = true; // Mark changes
                }
                break;

            case 3:
                System.out.println("Enter Part ID to Update:");
                String updatePartID = scanner.nextLine().trim();
                System.out.println("Enter Updated Part Description:");
                String newDescription = scanner.nextLine().trim();

                // Update record
                bPlusTree.updateNode(updatePartID, newDescription);
                changesMade = true; // Mark changes
                break;
```

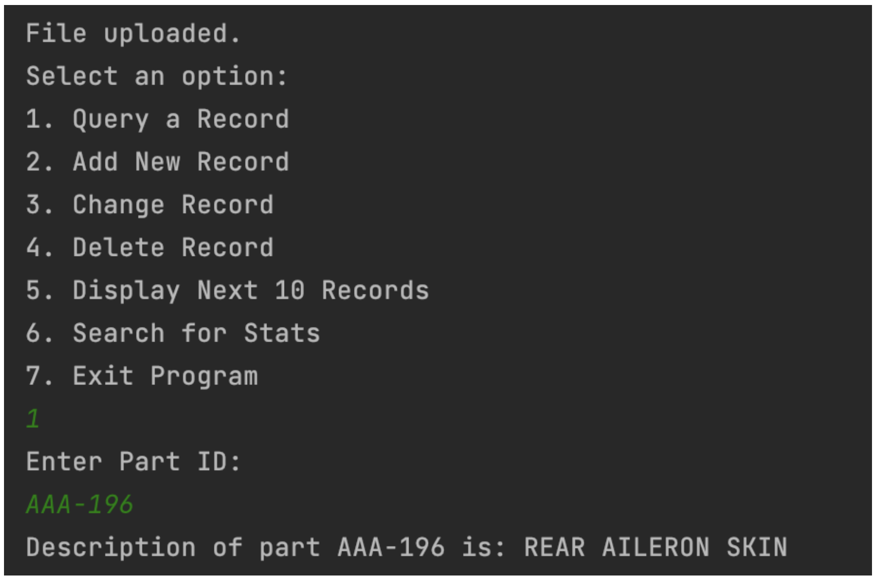


```
                case 4:
                    System.out.println("Enter Part ID to Delete:");
                    String deletePartId = scanner.nextLine().trim();

                    // Check if the Part ID exists
                    String existingDescription = bPlusTree.searchNode(deletePartId);
                    if (existingDescription == null) {
                        System.out.println("Part ID " + deletePartId + " does not exist. Deletion aborted.");
                    } else {
                        // Proceed with deletion
                        bPlusTree.deleteNode(deletePartId);
                        System.out.println("Part ID " + deletePartId + " deleted successfully.");
                        changesMade = true; // Mark changes
                    }
                    break;

                case 5:
                    System.out.println("Enter Part ID to print next 10:");
                    String partId = scanner.nextLine().trim();
                    ArrayList<Product> productList = bPlusTree.getNextTenNode(partId);

                    if (productList == null || productList.isEmpty()) {
                        System.out.println("Invalid Part ID or no next parts available.");
                    } else {
                        for (Product product : productList) {
                            System.out.println("PartId: " + product.getId() + " Description: " + product.getDescription());
                        }
                    }
                    break;
```

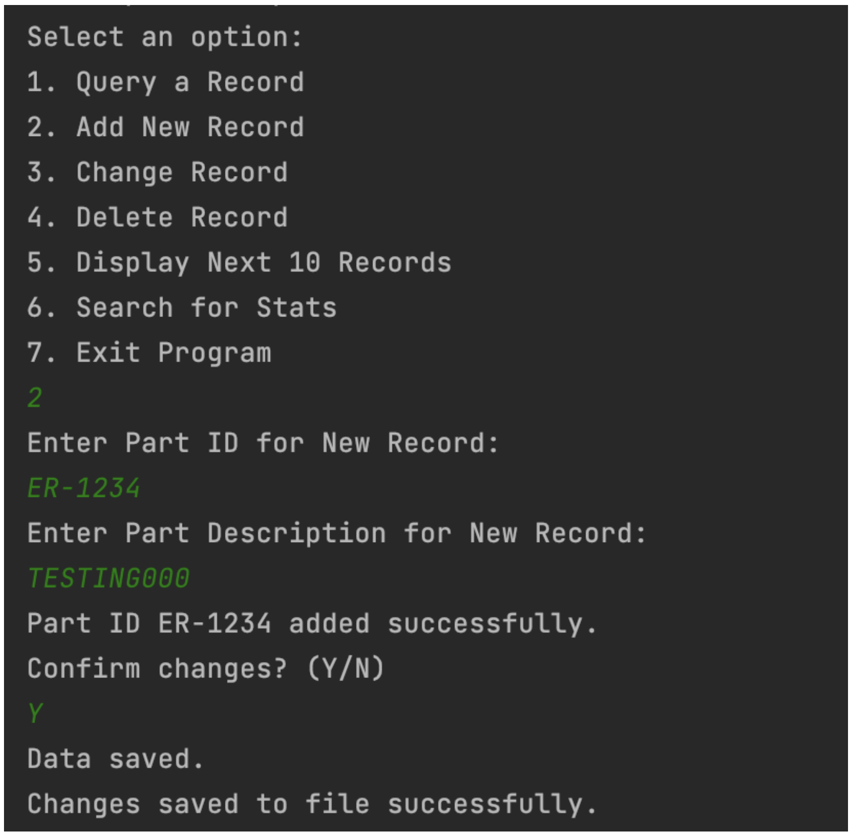


```

                case 6:
                    // Display statistics
                    bPlusTree.printStaistic();
                    break;

                case 7:
                    // Save changes to file and exit
                    if (changesMade) {
                        mainInstance.saveData();
                    }
                    System.out.println("Exiting program.");
                    System.exit( status: 0);
                    break;

                default:
                    System.out.println("Option does not exist. Please enter a number between 1 and 7.");
            }

            // Ask the user if changes should be saved only when changes were made
            if (changesMade) {
                System.out.println("Confirm changes? (Y/N)");
                String saveChoice = scanner.nextLine().trim().toLowerCase();
                if (saveChoice.equals("y")) {
                    mainInstance.saveData();
                    System.out.println("Changes saved to file successfully.");
                }
            }
        }
    }
```

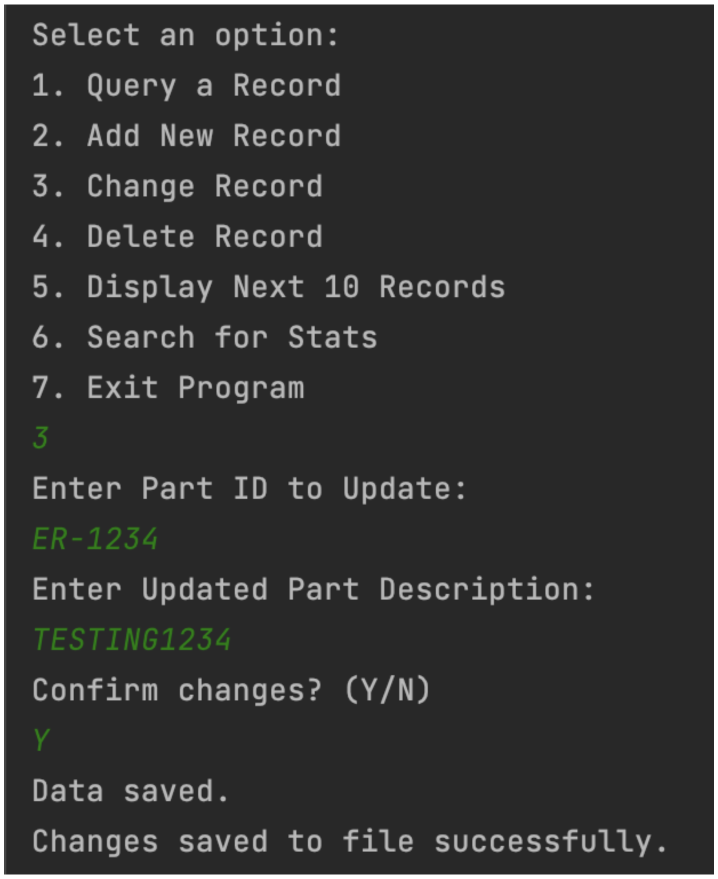
Output:



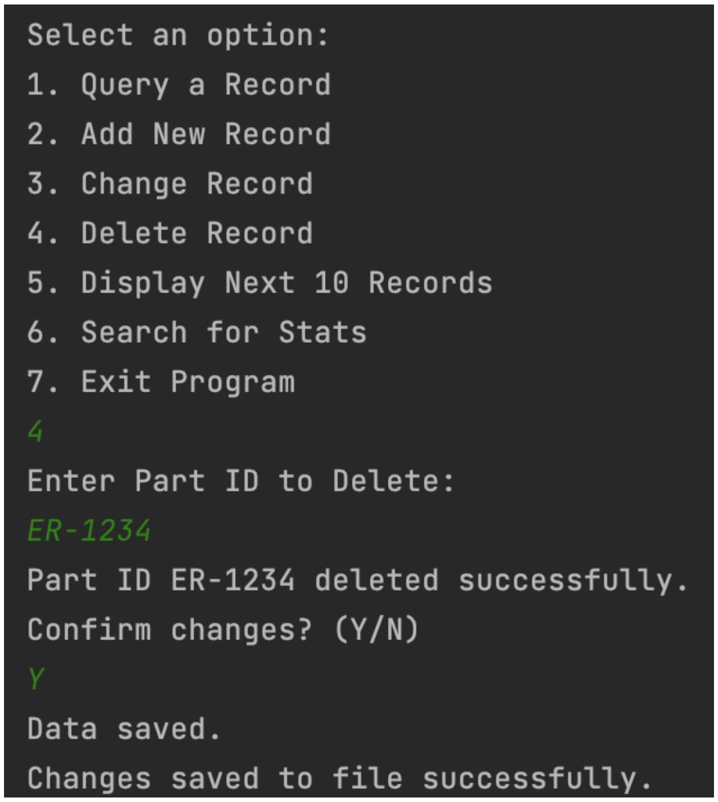
```
File uploaded.
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
1
Enter Part ID:
AAA-196
Description of part AAA-196 is: REAR AILERON SKIN
```



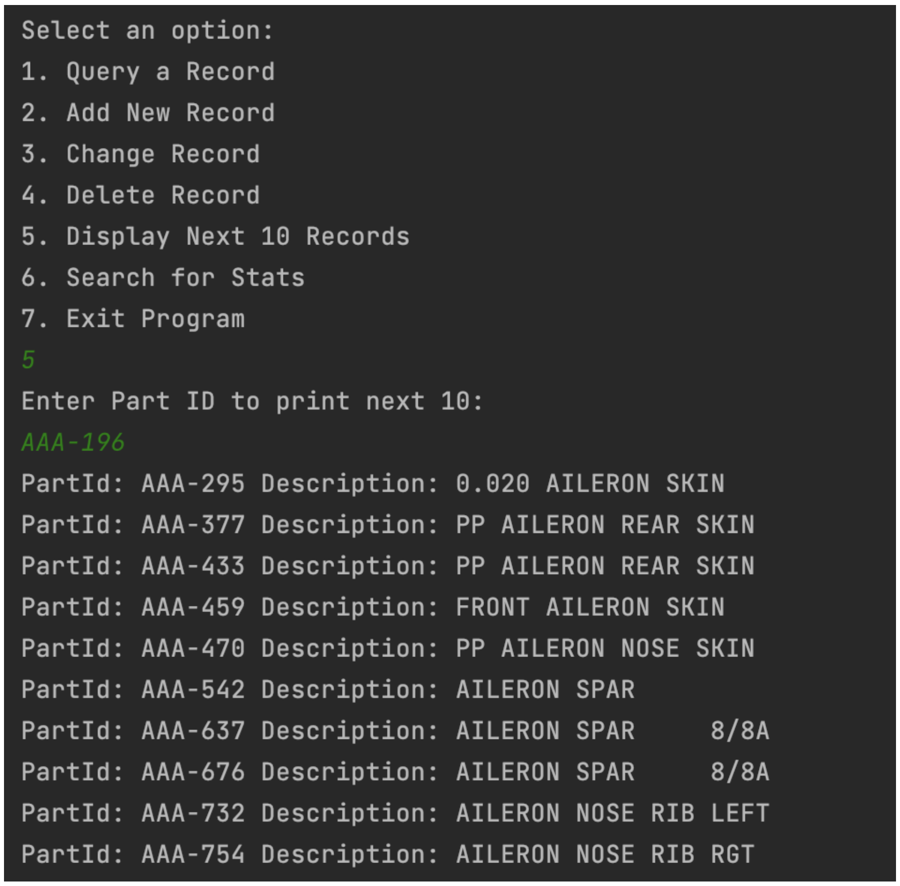
```
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
2
Enter Part ID for New Record:
ER-1234
Enter Part Description for New Record:
TESTING000
Part ID ER-1234 added successfully.
Confirm changes? (Y/N)
Y
Data saved.
Changes saved to file successfully.
```



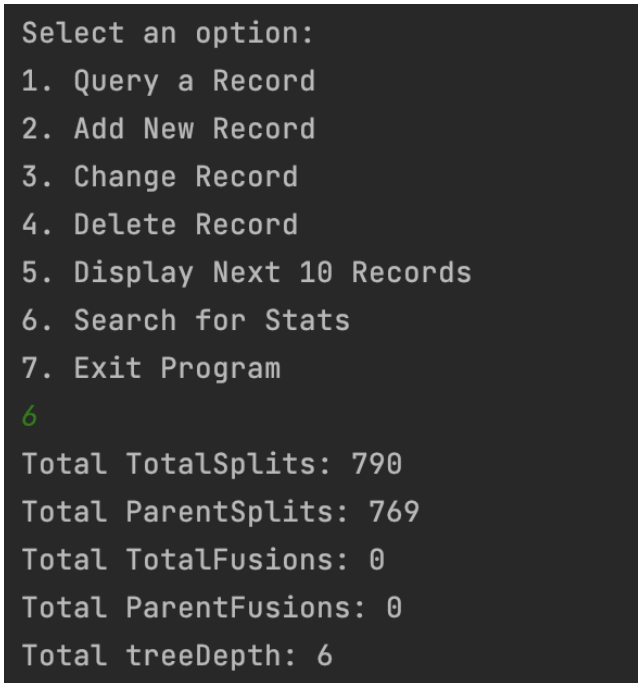
```
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
3
Enter Part ID to Update:
ER-1234
Enter Updated Part Description:
TESTING1234
Confirm changes? (Y/N)
Y
Data saved.
Changes saved to file successfully.
```



```
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
4
Enter Part ID to Delete:
ER-1234
Part ID ER-1234 deleted successfully.
Confirm changes? (Y/N)
Y
Data saved.
Changes saved to file successfully.
```



```
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
5
Enter Part ID to print next 10:
AAA-196
PartId: AAA-295 Description: 0.020 AILERON SKIN
PartId: AAA-377 Description: PP AILERON REAR SKIN
PartId: AAA-433 Description: PP AILERON REAR SKIN
PartId: AAA-459 Description: FRONT AILERON SKIN
PartId: AAA-470 Description: PP AILERON NOSE SKIN
PartId: AAA-542 Description: AILERON SPAR
PartId: AAA-637 Description: AILERON SPAR     8/8A
PartId: AAA-676 Description: AILERON SPAR     8/8A
PartId: AAA-732 Description: AILERON NOSE RIB LEFT
PartId: AAA-754 Description: AILERON NOSE RIB RGT
```



```
Select an option:
1. Query a Record
2. Add New Record
3. Change Record
4. Delete Record
5. Display Next 10 Records
6. Search for Stats
7. Exit Program
6
Total TotalSplits: 790
Total ParentSplits: 769
Total TotalFusions: 0
Total ParentFusions: 0
Total treeDepth: 6
```

Concluding Statement:

Implementing the B+ Tree for this project presented several challenges that required careful consideration and problem-solving. At the beginning of the project, our team struggled with implementing the insert and delete operations. To further elaborate, both operations involved modifying the tree's structure, which required us to maintain the tree's properties. In the case of insertion, adding a new key increased the node size, which caused us to ensure that the node size stays within the required limits. This meant that we had to manage node splits to maintain balance carefully. For the delete operation, we had to ensure that nodes did not fall below a minimum size. If a node became too small, we had to figure out how to merge or redistribute the data to maintain the tree's integrity. Another challenge encountered, was when we executed the code to display the statistics, the TotalFusions and ParentFusions resulted in 0. The amount of deletions performed did not impact the TotalFusions and ParentFusions operation.

In conclusion, our B+ Tree project was a highly valuable learning experience. We successfully implemented all required functionalities, including insertion, deletion, searching, and displaying records. The challenges we encountered, particularly with the insert and delete operations, helped us deepen our understanding of how B+ Trees work and how they can be implemented in Java. We also learned a great deal about the tree's structure, such as maintaining balance and ensuring data integrity. Despite our obstacles, we overcame these challenges through teamwork and problem-solving. This project gave us practical experience working with tree structures, an essential concept in computer science. We are confident as a team that moving forward, we will utilize the lessons learned from this project, and thus enhance our programming skills to help us avoid similar challenges in future endeavors.