Muhammad Hammad (23K-2005)

Data Structures

Assignment 02

Trees Database System

Code:

```cpp
#include <iostream>
#include <stdlib.h>
#include <string>
#include <limits>
#include <chrono>
#include <iomanip>
#include <ctime>
#include <sstream>
using namespace std;

// dummy record limit
#define limit 10000

int getInt() {
    int n;
    for (;;) {
        if (cin >> n) {
            return n;
        }
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Invalid entry. Please enter an integer: ";
    }
}

// ------------------------------------------------- Binary Search Tree --------------------------------
--------------------
class RecordBST {
    public:
        static int totalIDs;
        int id;
        string name;
        int age;
        RecordBST() : id(++totalIDs), name(""), age(0) {}
        RecordBST(string n, int a) : id(++totalIDs), name(n), age(a) {}

        // copy constructor for deep copy
        RecordBST (const RecordBST &other) : id(other.id), name(other.name), age(other.age) {}

        // copy assignment operator
        RecordBST& operator=(const RecordBST &other) {
            // self assignment check
            if (this == &other) {
                return *this;
```

```
            }
            this->id = other.id;
            this->name = other.name;
            this->age = other.age;
            return *this;
        }
};
int RecordBST::totalIDs = 0;

class NodeBST {
    public:
        RecordBST record;
        NodeBST *leftChild;
        NodeBST *rightChild;

        NodeBST (RecordBST newRecord) : record(newRecord), leftChild(NULL), rightChild(NULL) {}
};

class BST_Table {
    public:
        NodeBST *root;

        BST_Table () : root(NULL) {}

        void createRecord () {
            string name;
            int age;
            cout << "__________ Create Record __________" << endl;
            cout << "Enter the name: ";
            getline(cin, name);
            cout << "Enter the age: ";
            age = getInt();
            cin.ignore();

            RecordBST newRecord(name, age);
            insertNode(newRecord);
        }

        void insertNode(RecordBST newRecord) {
            if (root == NULL) {
                root = new NodeBST(newRecord);
                return;
            }
            insertHelper(root, newRecord);
        }

        void insertHelper(NodeBST *temp, RecordBST newRecord, NodeBST *parent = NULL) {
            if (temp == NULL) {
                if (newRecord.id <= parent->record.id)
                    parent->leftChild = new NodeBST(newRecord);
                else
                    parent->rightChild = new NodeBST(newRecord);
                return;
            }
```

```
        parent = temp;
        if (newRecord.id <= parent->record.id)
            insertHelper(temp->leftChild, newRecord, parent);
        else
            insertHelper(temp->rightChild, newRecord, parent);
    }

    void deleteNode() {
        if (root == NULL) {
            cout << "Unable to delete. The BST is empty." << endl;
            return;
        }

        int target;
        cout << "Enter the ID that you want to delete: ";
        target = getInt();
        cin.ignore();

        bool deleted = deleteHelper(root, target);
        if (deleted) {
            cout << "ID " << target << " has been deleted." << endl;
        } else {
            cout << "Delete failed. The ID was not found in the database." << endl;
        }
    }

    bool deleteHelper(NodeBST*& temp, int target) {
        if (temp == NULL) return false; // base case: not found

        // find the node
        if (target < temp->record.id) {
            return deleteHelper(temp->leftChild, target);
        }
        else if (target > temp->record.id) {
            return deleteHelper(temp->rightChild, target);
        }
        else { // node found
            // case 1: no children
            if (temp->leftChild == NULL && temp->rightChild == NULL) {
                delete temp;
                temp = NULL;
                return true;
            }

            // case 2: 01 children
            else if (temp->rightChild == NULL) { // node has left child
                NodeBST* left = temp->leftChild;
                delete temp;
                temp = left;
                return true;
            }
            else if (temp->leftChild == NULL) { // node has right child
                NodeBST* right = temp->rightChild;
```

```
            delete temp;
            temp = right;
            return true;
        }

        // case 3: 02 children
        NodeBST* inSuccessor = findInSuccessor(temp->rightChild);
        temp->record.id = inSuccessor->record.id;
        return deleteHelper(temp->rightChild, inSuccessor->record.id);
    }
    return false;
}

NodeBST* findInSuccessor(NodeBST* right) {
    while (right->leftChild != NULL) {
        right = right->leftChild;
    }
    return right;
}

void search() {
    if (root == NULL) {
        cout << "Unable to search. The BST is empty." << endl;
        return;
    }

    int target;
    cout << "__________ Search Record __________" << endl;
    cout << "Enter the ID that you want to search: ";
    target = getInt();
    cin.ignore();

    NodeBST* result = searchHelper(root, target);
    if (result == NULL) {
        cout << "ID " << target << " not found in the database." << endl;
        return;
    }
    cout << "ID " << target << " found in the database." << endl;
    cout << "Details of ID " << target << ":" << endl;;
    cout << "Name: " << result->record.name << endl;
    cout << "Age: " << result->record.age << endl;
    return;
}

NodeBST* searchHelper(NodeBST *temp, int target) {
    if (temp == NULL) {
        return NULL;
    }

    if (target == temp->record.id) {
        return temp;
    }

    if (target <= temp->record.id)
```

```
            return searchHelper(temp->leftChild, target);
        else
            return searchHelper(temp->rightChild, target);
    }

    void update() {
        if (root == NULL) {
            cout << "Unable to update. The BST is empty." << endl;
            return;
        }

        int target;
        cout << "__________ Update Record __________" << endl;
        cout << "Enter the ID that you want to update: ";
        target = getInt();
        cin.ignore();

        NodeBST* result = searchHelper(root, target);
        if (result == NULL) {
            cout << "Update failed. The ID was not found in the database." << endl;
            return;
        }

        string newName;
        int newAge;
        cout << "Enter new name for ID " << target << ": ";
        getline(cin, newName);
        cout << "Enter new age for ID " << target << ": ";
        newAge = getInt();
        cin.ignore();

        result->record.name = newName;
        result->record.age = newAge;

        cout << "Record for ID " << target << " has been updated." << endl;
    }

    void display() {
        if (root == NULL) {
            cout << "Unable to display records. The BST database is empty right now." << endl;
            return;
        }
        cout << "__________ Display Record __________" << endl;
        inOrder(root);
    }

    void inOrder (NodeBST *temp) {
        if (temp == NULL) return;

        inOrder(temp->leftChild);
        cout << "ID: " << temp->record.id << endl;
        cout << "Name: " << temp->record.name << endl;
        cout << "Age: " << temp->record.age << endl << endl;
        inOrder(temp->rightChild);
```

```
}

void createDummyRecord() {
    cout << "\n__________ Create Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < limit; i++) {
        string name = "Dummy";
        name += to_string(i + 1);
        srand(time(0));
        int age = rand() % 42 + 18;
        RecordBST newRecord(name, age);
        insertNode(newRecord);
    }

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "Dummy Record of " << limit << " people have been created." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void searchDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Search Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    cout << "Details of the searched IDs:" << endl;
    for (int i=0; i<5; i++) {
        NodeBST* temp = searchHelper(root, idsArr[i]);
        if (temp != NULL) {
            cout << "ID: " << temp->record.id << endl;
            cout << "Name: " << temp->record.name << endl;
            cout << "Age: " << temp->record.age << endl << endl;
        }
        else {
            cout << "ID " << idsArr[i] << " not found in the database." << endl;
        }
    }
    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs ("<< ids << ") have been searched and displayed." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void updateDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Update Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    for (int i=0; i<5; i++) {
        updateDummyHelper(idsArr[i]);
    }
    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs ("<< ids << ") have been updated." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void updateDummyHelper(int target) {
```

```
            NodeBST* temp = searchHelper(root, target);

            if (temp != NULL) {
                temp->record.name += " ka naya naam";
                srand(time(0));
                int age = rand() % 42 + 18;
                temp->record.age = age;
                cout << "------- Updated Details for ID " << target << " -------" << endl;
                cout << "Name: " << temp->record.name << endl;
                cout << "Age: " <<  temp->record.age << endl << endl;;
            }
            else {
                cout << "ID " << target << " not found in the database." << endl;
            }
        }

        void deleteDummyRecord (string ids, int idsArr[]) {
            cout << "\n__________ Delete Dummy Record __________" << endl;
            auto startTime = std::chrono::high_resolution_clock::now();
            for (int i=0; i<5; i++) {
                if(deleteHelper(root, idsArr[i])) {
                    cout << "ID " << idsArr[i] << " has been deleted." << endl;
                }
                else {
                    cout << "Delete failed." << idsArr[i] << " not found in the dummy database." << endl;
                }
            }
            auto endTime = std::chrono::high_resolution_clock::now();
            auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
            cout << "5 IDs ("<< ids << ") have been deleted." << endl;
            cout << "Duration: " << duration.count() << " milliseconds." << endl;
        }

        void performDummyOperations () {
            createDummyRecord();
            if (root == NULL) {
                cout << "Unable to perform operations ahead since there is no dummy record in the database."
<< endl;
                return;
            }

            int idsArray[5] = {limit/5, (limit/5) * 2, (limit/5) * 3, (limit/5) * 4, (limit/5) * 5};
            string ids = "";
            for (int i=0; i<5; i++) {
                ids = ids + to_string(idsArray[i]) + ", ";
            }
            ids.erase(ids.length() - 2 ); // remove comma and space
            searchDummyRecord(ids, idsArray);
            updateDummyRecord(ids, idsArray);
            deleteDummyRecord(ids, idsArray);
        }
};
```

```
// ------------------------------------------------- AVL Tree ----------------------------------------
----------
class RecordAVL {
    public:
        static int totalIDs;
        int id;
        string name;
        int age;
        RecordAVL() : id(++totalIDs), name(""), age(0) {}
        RecordAVL(string n, int a) : id(++totalIDs), name(n), age(a) {}

        // copy constructor for deep copy
        RecordAVL (const RecordAVL &other) : id(other.id), name(other.name), age(other.age) {}

        // copy assignment operator
        RecordAVL& operator=(const RecordAVL &other) {
            // self assignment check
            if (this == &other) {
                return *this;
            }
            this->id = other.id;
            this->name = other.name;
            this->age = other.age;
            return *this;
        }
};
int RecordAVL::totalIDs = 0;

class NodeAVL {
    public:
        RecordAVL record;
        NodeAVL *leftChild;
        NodeAVL *rightChild;
        int height;

        NodeAVL (RecordAVL newRecord) : record(newRecord), height(0), leftChild(NULL), rightChild(NULL) {}
};

class AVL_Table {
    public:
        NodeAVL *root;

        AVL_Table () : root(NULL) {}

        int getHeight (NodeAVL* node) {
            if (node == NULL) return 0;
            return node->height;
        }

        void updateHeight(NodeAVL* node) {
            node->height = 1 + max(getHeight(node->leftChild), getHeight(node->rightChild));
        }

        int getBalanceFactor (NodeAVL* node) {
```

```
    if (node == NULL) return 0;
    return (getHeight(node->leftChild) - getHeight(node->rightChild));
}

NodeAVL* rotateRight (NodeAVL* y) {
    NodeAVL *x = y->leftChild;
    NodeAVL *T2 = x->rightChild;

    x->rightChild = y;
    y->leftChild = T2;

    updateHeight(y);
    updateHeight(x);
    return x;
}

NodeAVL* rotateLeft (NodeAVL* x) {
    NodeAVL* y = x->rightChild;
    NodeAVL *T2 = y->leftChild;

    y->leftChild = x;
    x->rightChild = T2;

    updateHeight(x);
    updateHeight(y);

    return y;
}

void createRecord () {
    string name;
    int age;
    cout << "__________ Create Record __________" << endl;
    cout << "Enter the name: ";
    getline(cin, name);
    cout << "Enter the age: ";
    age = getInt();
    cin.ignore();

    RecordAVL newRecord(name, age);
    insertNode(newRecord);
}

void insertNode(RecordAVL newRecord) {
    if (root == NULL) {
        root = new NodeAVL(newRecord);
        return;
    }
    root = insertHelper(root, newRecord);
}

NodeAVL* insertHelper(NodeAVL* temp, RecordAVL newRecord) {
    // base case: if tree is empty, then this node becomes the root
    if (temp == NULL) {
```

```
            return new NodeAVL(newRecord);
        }

        if (newRecord.id < temp->record.id) {
            temp->leftChild = insertHelper(temp->leftChild, newRecord);
        } else if (newRecord.id > temp->record.id) {
            temp->rightChild = insertHelper(temp->rightChild, newRecord);
        } else {
            return temp;
        }

        updateHeight(temp);
        int balanceFactor = getBalanceFactor(temp);

        // Left Left Case
        if (balanceFactor > 1 && newRecord.id < temp->leftChild->record.id) {
            return rotateRight(temp);
        }

        // Right Right Case
        if (balanceFactor < -1 && newRecord.id > temp->rightChild->record.id) {
            return rotateLeft(temp);
        }

        // Left Right Case
        if (balanceFactor > 1 && newRecord.id > temp->leftChild->record.id) {
            temp->leftChild = rotateLeft(temp->leftChild);
            return rotateRight(temp);
        }

        // Right Left Case
        if (balanceFactor < -1 && newRecord.id < temp->rightChild->record.id) {
            temp->rightChild = rotateRight(temp->rightChild);
            return rotateLeft(temp);
        }

        return temp;
    }

    void deleteNode() {
        if (root == NULL) {
            cout << "Unable to delete. The AVL Tree is empty." << endl;
            return;
        }

        int target;
        cout << "Enter the ID that you want to delete: ";
        target = getInt();
        cin.ignore();

        bool deleted = false; // Track whether deletion was successful
        root = deleteHelper(root, target, deleted); // Pass the deleted flag by reference

        if (deleted) {
```

```
            cout << "ID " << target << " has been deleted successfully." << endl;
        } else {
            cout << "Deletion failed. ID not found in the database." << endl;
        }
    }

    NodeAVL* deleteHelper(NodeAVL* temp, int target, bool& deleted) {
        if (temp == NULL) return NULL;

        // find the target node
        if (target < temp->record.id) {
            temp->leftChild = deleteHelper(temp->leftChild, target, deleted);
        }
        else if (target > temp->record.id) {
            temp->rightChild = deleteHelper(temp->rightChild, target, deleted);
        }
        else { // node found
            deleted = true;

            // case 1: node with no children
            if (temp->leftChild == NULL && temp->rightChild == NULL) {
                delete temp;
                return NULL;
            }
            // case 2: node with one child
            else if (temp->leftChild == NULL) { // only right child
                NodeAVL* right = temp->rightChild;
                delete temp;
                return right;
            } else if (temp->rightChild == NULL) { // only left child
                NodeAVL* left = temp->leftChild;
                delete temp;
                return left;
            }
            // case 3: node with two children
            else {
                NodeAVL* inSuccessor = findInSuccessor(temp->rightChild);
                temp->record = inSuccessor->record;
                temp->rightChild = deleteHelper(temp->rightChild, inSuccessor->record.id, deleted);
            }
        }

        updateHeight(temp);
        int bf = getBalanceFactor(temp);

        // Left Left Case
        if (bf > 1 && getBalanceFactor(temp->leftChild) >= 0) {
            return rotateRight(temp);
        }
        // Left Right Case
        if (bf > 1 && getBalanceFactor(temp->leftChild) < 0) {
            temp->leftChild = rotateLeft(temp->leftChild);
            return rotateRight(temp);
        }
```

```
        // Right Right Case
        if (bf < -1 && getBalanceFactor(temp->rightChild) <= 0) {
            return rotateLeft(temp);
        }
        // Right Left Case
        if (bf < -1 && getBalanceFactor(temp->rightChild) > 0) {
            temp->rightChild = rotateRight(temp->rightChild);
            return rotateLeft(temp);
        }
        return temp;
    }

    NodeAVL* findInSuccessor(NodeAVL* right) {
        while (right->leftChild != NULL) {
            right = right->leftChild;
        }
        return right;
    }

    void search() {
        if (root == NULL) {
            cout << "Unable to search. The AVL Tree is empty." << endl;
            return;
        }

        int target;
        cout << "__________ Search Record __________" << endl;
        cout << "Enter the ID that you want to search: ";
        target = getInt();
        cin.ignore();

        NodeAVL* result = searchHelper(root, target);
        if (result == NULL) {
            cout << "ID " << target << " not found in the database." << endl;
            return;
        }
        cout << "ID " << target << " found in the database." << endl;
        cout << "Details of ID " << target << ":" << endl;;
        cout << "Name: " << result->record.name << endl;
        cout << "Age: " << result->record.age << endl;
        return;
    }

    NodeAVL* searchHelper(NodeAVL *temp, int target) {
        if (temp == NULL) {
            return NULL;


        }

        if (target == temp->record.id) {
            return temp;
        }
```

```
        if (target <= temp->record.id) searchHelper(temp->leftChild, target);
        else searchHelper(temp->rightChild, target);
    }

    void update() {
        if (root == NULL) {
            cout << "Unable to update. The AVL Tree is empty." << endl;
            return;
        }

        int target;
        cout << "__________ Update Record __________" << endl;
        cout << "Enter the ID that you want to update: ";
        target = getInt();
        cin.ignore();

        NodeAVL* result = searchHelper(root, target);
        if (result == NULL) {
            cout << "Update failed. The ID was not found in the database." << endl;
            return;
        }

        string newName;
        int newAge;
        cout << "Enter new name for ID " << target << ": ";
        getline(cin, newName);
        cout << "Enter new age for ID " << target << ": ";
        newAge = getInt();
        cin.ignore();

        result->record.name = newName;
        result->record.age = newAge;

        cout << "Record for ID " << target << " has been updated." << endl;
        return;
    }

    void display() {
        if (root == NULL) {
        cout << "Unable to display records. The AVL Tree database is empty right now." << endl;
        return;
    }
        cout << "__________ Display Record __________" << endl;
        inOrder(root);
    }

    void inOrder (NodeAVL *temp) {
        if (temp == NULL) return;

        inOrder(temp->leftChild);
        cout << "ID: " << temp->record.id << endl;
        cout << "Name: " << temp->record.name << endl;
        cout << "Age: " << temp->record.age << endl << endl;
        inOrder(temp->rightChild);
```

```
}

void createDummyRecord () {
    cout << "__________ Create Dummy Record __________" << endl;
    for (int i=0; i<limit; i++) {
        string name = "Dummy";
        name += to_string(i+1);
        srand(time(0));
        int age = rand() % 42 + 18;
        RecordAVL newRecord(name, age);
        insertNode(newRecord);
    }
    cout << "Dummy Record of " << limit << " people have been created." << endl;
}

void searchDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Search Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    cout << "Details of the searched IDs:" << endl;
    for (int i = 0; i < 5; i++) {
        NodeAVL* temp = searchHelper(root, idsArr[i]);
        if (temp != NULL) {
            cout << "ID: " << temp->record.id << endl;
            cout << "Name: " << temp->record.name << endl;
            cout << "Age: " << temp->record.age << endl << endl;
        } else {
            cout << "ID " << idsArr[i] << " not found in the database." << endl;
        }
    }
    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs (" << ids << ") have been searched and displayed." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void updateDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Update Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    for (int i = 0; i < 5; i++) {
        updateDummyHelper(idsArr[i]);
    }
    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs (" << ids << ") have been updated." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void updateDummyHelper(int target) {
    NodeAVL* temp = searchHelper(root, target);
    if (temp != NULL) {
        temp->record.name += " ka naya naam";
        srand(time(0));
        int age = rand() % 42 + 18;
        temp->record.age = age;
```

```
            cout << "------- Updated Details for ID " << target << " -------" << endl;
            cout << "Name: " << temp->record.name << endl;
            cout << "Age: " << temp->record.age << endl << endl;
        } else {
            cout << "ID " << target << " not found in the database." << endl;
        }
    }

    void deleteDummyRecord(string ids, int idsArr[]) {
        cout << "\n__________ Delete Dummy Record __________" << endl;
        auto startTime = std::chrono::high_resolution_clock::now();
        for (int i = 0; i < 5; i++) {
            bool deleted = false;
            root = deleteHelper(root, idsArr[i], deleted);

            if (deleted) {
                cout << "ID " << idsArr[i] << " has been deleted." << endl;
            } else {
                cout << "Delete failed. " << idsArr[i] << " not found in the dummy database." << endl;
            }
        }
        auto endTime = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
        cout << "5 IDs (" << ids << ") have been deleted." << endl;
        cout << "Duration: " << duration.count() << " milliseconds." << endl;
    }

    void performDummyOperations() {
        createDummyRecord();
        if (root == NULL) {
            cout << "Unable to perform operations ahead since there is no dummy record in the database."
<< endl;
            return;
        }

        int idsArray[5] = {limit / 5, (limit / 5) * 2, (limit / 5) * 3, (limit / 5) * 4, (limit / 5) * 5};
        string ids = "";
        for (int i = 0; i < 5; i++) {
            ids = ids + to_string(idsArray[i]) + ", ";
        }
        ids.erase(ids.length() - 2); // remove comma and space
        searchDummyRecord(ids, idsArray);
        updateDummyRecord(ids, idsArray);
        deleteDummyRecord(ids, idsArray);
    }

};


// Considering a B Tree of Order 4
const int MAX_KEYS = 3;
const int MIN_KEYS = MAX_KEYS / 2;
```

```
// ------------------------------------------------- B Tree ----------------------------------------
---------
class RecordBTree {
    public:
        static int totalIDs;
        int id;
        string name;
        int age;

        RecordBTree() : id(0), name(""), age(0) {}

        RecordBTree(string n, int a) : name(n), age(a) {
            id = ++totalIDs;
        }
};

int RecordBTree::totalIDs = 0;

class NodeBTree {
    public:
        RecordBTree records[MAX_KEYS];
        NodeBTree* children[MAX_KEYS + 1];
        int numKeys;
        bool isLeaf;

        NodeBTree(bool leaf) : numKeys(0), isLeaf(leaf) {
            for (int i = 0; i < MAX_KEYS + 1; i++) {
                children[i] = NULL;
            }
        }
};

class BTree_Table {
    private:
        NodeBTree* root;

    public:
        BTree_Table() : root(NULL) {}

        void createRecord() {
            string name;
            int age;
            cout << "__________ Create Record __________" << endl;
            cout << "Enter the name: ";
            getline(cin, name);
            cout << "Enter the age: ";
            age = getInt();
            cin.ignore();

            RecordBTree newRecord(name, age);
            insert(newRecord);
        }

        void insert(RecordBTree newRecord) {
```

```
        if (root == NULL) {
            root = new NodeBTree(true);
            root->records[0] = newRecord;
            root->numKeys = 1;
        } else {
            if (root->numKeys == MAX_KEYS) {
                NodeBTree* newNode = new NodeBTree(false);
                newNode->children[0] = root;
                splitChild(newNode, 0, root);
                root = newNode;
            }
            insertNonFull(root, newRecord);
        }
    }

    void insertNonFull(NodeBTree* node, RecordBTree newRecord) {
        int i = node->numKeys - 1;

        if (node->isLeaf) {
            while (i >= 0 && newRecord.id < node->records[i].id) {
                node->records[i + 1] = node->records[i];
                i--;
            }
            node->records[i + 1] = newRecord;
            node->numKeys++;
        } else {
            while (i >= 0 && newRecord.id < node->records[i].id) {
                i--;
            }
            i++;
            if (node->children[i]->numKeys == MAX_KEYS) {
                splitChild(node, i, node->children[i]);
                if (newRecord.id > node->records[i].id) {
                    i++;
                }
            }
            insertNonFull(node->children[i], newRecord);
        }
    }

    void splitChild(NodeBTree* parent, int index, NodeBTree* fullChild) {
        NodeBTree* newChild = new NodeBTree(fullChild->isLeaf);
        newChild->numKeys = MIN_KEYS;

        for (int j = 0; j < MIN_KEYS; j++) {
            newChild->records[j] = fullChild->records[j + MIN_KEYS + 1];
        }

        if (!fullChild->isLeaf) {
            for (int j = 0; j < MIN_KEYS + 1; j++) {
                newChild->children[j] = fullChild->children[j + MIN_KEYS + 1];
            }
        }
```

```
        fullChild->numKeys = MIN_KEYS;

        for (int j = parent->numKeys; j >= index + 1; j--) {
            parent->children[j + 1] = parent->children[j];
        }
        parent->children[index + 1] = newChild;

        for (int j = parent->numKeys - 1; j >= index; j--) {
            parent->records[j + 1] = parent->records[j];
        }
        parent->records[index] = fullChild->records[MIN_KEYS];
        parent->numKeys++;
    }

    void deleteNode() {
        if (root == NULL) {
            cout << "Unable to delete. The B-tree is empty." << endl;
            return;
        }

        int id;
        cout << "Enter the ID that you want to delete: ";
        id = getInt();

        bool deleted = deleteHelper(root, id);

        if (deleted) {
            cout << "ID " << id << " has been deleted." << endl;
        } else {
            cout << "Delete failed. The ID was not found in the database." << endl;
        }

        // if the root has no keys after deletion, make its first child the new root
        if (root->numKeys == 0) {
            NodeBTree* oldRoot = root;
            if (root->isLeaf) {
                root = NULL;
            } else {
                root = root->children[0];
            }
            delete oldRoot;
        }
    }

    bool deleteHelper(NodeBTree* node, int id) {
        int i = 0;
        while (i < node->numKeys && id > node->records[i].id) {
            i++;
        }

        // If the id is found
        if (i < node->numKeys && id == node->records[i].id) {
            if (node->isLeaf) {
                // case 1: node is a leaf, simply delete the record
```

```
                for (int j = i; j < node->numKeys - 1; j++) {
                    node->records[j] = node->records[j + 1];
                }
                node->numKeys--;
                return true;
            } else {
                // case 2: node is not a leaf, find in-order predecessor or successor
                if (node->children[i]->numKeys >= MIN_KEYS) {
                    // Replace with in-order predecessor
                    RecordBTree pred = getInOrderPredecessor(node, i);
                    node->records[i] = pred;
                    return deleteHelper(node->children[i], pred.id);
                } else if (node->children[i + 1]->numKeys >= MIN_KEYS) {
                    // Replace with in-order successor
                    RecordBTree succ = getInOrderSuccessor(node, i);
                    node->records[i] = succ;
                    return deleteHelper(node->children[i + 1], succ.id);
                } else {
                    // Merge the children
                    mergeChildren(node, i);
                    return deleteHelper(node->children[i], id);
                }
            }
        } else {
            if (node->isLeaf) {
                return false; // ID not found
            } else {
                bool deleted = deleteHelper(node->children[i], id);
                if (deleted && node->children[i]->numKeys < MIN_KEYS) {
                    handleUnderflow(node, i);
                }
                return deleted;
            }
        }
    }

    void mergeChildren(NodeBTree* parent, int index) {
        NodeBTree* leftChild = parent->children[index];
        NodeBTree* rightChild = parent->children[index + 1];

        // Move the parent's key down to the left child
        leftChild->records[leftChild->numKeys] = parent->records[index];
        leftChild->numKeys++;

        // Move all records from the right child to the left child
        for (int j = 0; j < rightChild->numKeys; j++) {
            leftChild->records[leftChild->numKeys + j] = rightChild->records[j];
        }
        leftChild->numKeys += rightChild->numKeys;

        // Move all children from the right child to the left child
        if (!rightChild->isLeaf) {
            for (int j = 0; j <= rightChild->numKeys; j++) {
                leftChild->children[leftChild->numKeys + j] = rightChild->children[j];
```

```
        }
    }

    // Shift the parent's children and keys to fill the gap
    for (int j = index; j < parent->numKeys - 1; j++) {
        parent->records[j] = parent->records[j + 1];
        parent->children[j + 1] = parent->children[j + 2];
    }

    parent->numKeys--;
    delete rightChild;
}

RecordBTree getInOrderPredecessor(NodeBTree* node, int index) {
    NodeBTree* current = node->children[index];
    while (!current->isLeaf) {
        current = current->children[current->numKeys];
    }
    return current->records[current->numKeys - 1];
}

RecordBTree getInOrderSuccessor(NodeBTree* node, int index) {
    NodeBTree* current = node->children[index + 1];
    while (!current->isLeaf) {
        current = current->children[0];
    }
    return current->records[0];
}

void handleUnderflow(NodeBTree* node, int index) {
    if (index > 0 && node->children[index - 1]->numKeys >= MIN_KEYS) {
        // Borrow from left sibling
        borrowFromLeft(node, index);
    } else if (index < node->numKeys && node->children[index + 1]->numKeys >= MIN_KEYS) {
        // Borrow from right sibling
        borrowFromRight(node, index);
    } else {
        // Merge with sibling
        if (index < node->numKeys) {
            mergeChildren(node, index);
        } else {
            mergeChildren(node, index - 1);
        }
    }
}

void borrowFromLeft(NodeBTree* parent, int index) {
    NodeBTree* child = parent->children[index];
    NodeBTree* leftSibling = parent->children[index - 1];

    // Shift all records and children in the child to the right
    for (int i = child->numKeys - 1; i >= 0; i--) {
        child->records[i + 1] = child->records[i];
    }
```

```
        if (!child->isLeaf) {
            for (int i = child->numKeys; i >= 0; i--) {
                child->children[i + 1] = child->children[i];
            }
        }

        // Move a record from the parent to the child
        child->records[0] = parent->records[index - 1];

        // Move a child from the left sibling to the child
        if (!child->isLeaf) {
            child->children[0] = leftSibling->children[leftSibling->numKeys];
        }

        // Update parent key
        parent->records[index - 1] = leftSibling->records[leftSibling->numKeys - 1];

        child->numKeys++;
        leftSibling->numKeys--;
    }

    void borrowFromRight(NodeBTree* parent, int index) {
        NodeBTree* child = parent->children[index];
        NodeBTree* rightSibling = parent->children[index + 1];

        // Copy the record from the parent to the child
        child->records[child->numKeys] = parent->records[index];

        if (!child->isLeaf) {
            child->children[child->numKeys + 1] = rightSibling->children[0];
        }

        parent->records[index] = rightSibling->records[0];

        // Shift all records and children in the right sibling to the left
        for (int i = 1; i < rightSibling->numKeys; i++) {
            rightSibling->records[i - 1] = rightSibling->records[i];
        }
        if (!rightSibling->isLeaf) {
            for (int i = 1; i <= rightSibling->numKeys; i++) {
                rightSibling->children[i - 1] = rightSibling->children[i];
            }
        }

        child->numKeys++;
        rightSibling->numKeys--;
    }

    void display() const {
        if (root == NULL) {
            cout << "Unable to display records. The B-Tree database is empty right now." << endl;
            return;
        }
        cout << "__________ Display Record __________" << endl;
```

```
        displayHelper(root);
    }

    void displayHelper(NodeBTree* node) const {
        if (node == NULL) return;

        for (int i = 0; i < node->numKeys; i++) {
            if (!node->isLeaf) {
                displayHelper(node->children[i]);
            }
            cout << "ID: " << node->records[i].id << endl;
            cout << "Name: " << node->records[i].name << endl;
            cout << "Age: " << node->records[i].age << endl << endl;
        }
        if (!node->isLeaf) {
            displayHelper(node->children[node->numKeys]);
        }
    }

    void search() const {
        if (root == NULL) {
            cout << "Unable to search. The B-Tree is empty." << endl;
            return;
        }
        cout << "__________ Search Record __________" << endl;
        int target;
        cout << "Enter the ID that you want to search in the database: ";
        target = getInt();

        NodeBTree* result = searchHelper(root, target);
        if (result == NULL) {
            cout << "ID " << target << " not found in the database." << endl;
        } else {
            cout << "ID " << target << " found in the database." << endl;
            for (int i = 0; i < result->numKeys; i++) {
                if (result->records[i].id == target) {
                    cout << "Details of ID " << result->records[i].id << ":" << endl;
                    cout << "Name: " << result->records[i].name << endl;
                    cout << "Age: " << result->records[i].age << endl;
                }
            }
        }
    }

    NodeBTree* searchHelper(NodeBTree* node, int id) const {
        if (node == NULL) return NULL;

        int i = 0;
        while (i < node->numKeys && id > node->records[i].id) {
            i++;
        }
        if (i < node->numKeys && id == node->records[i].id) {
            return node;
        }
```

```
        if (node->isLeaf) {
            return NULL;
        }
        return searchHelper(node->children[i], id);
    }

    void update() {
        if (root == NULL) {
            cout << "Unable to update. The B-Tree is empty." << endl;
            return;
        }

        int id;
        cout << "__________ Update Record __________" << endl;
        cout << "Enter the ID that you want to update: ";
        id = getInt();

        NodeBTree* node = searchHelper(root, id);
        if (node == NULL) {
            cout << "Update failed. The ID was not found in the database." << endl;
            return;
        }

        string newName;
        int newAge;
        cout << "Enter new name for ID " << id << ": ";
        cin.ignore();
        getline(cin, newName);
        cout << "Enter new age for ID " << id << ": ";
        newAge = getInt();

        for (int i = 0; i < node->numKeys; i++) {
            if (node->records[i].id == id) {
                node->records[i].name = newName;
                node->records[i].age = newAge;
                cout << "Record updated successfully." << endl;
                return;
            }
        }
    }

    void createDummyRecord () {
        cout << "__________ Create Dummy Record __________" << endl;
        auto startTime = std::chrono::high_resolution_clock::now();
        for (int i=0; i<limit; i++) {
            string name = "Dummy";
            name += to_string(i+1);
            srand(time(0));
            int age = rand() % 42 + 18;
            RecordBTree newRecord(name, age);
            insert(newRecord);
        }
        auto endTime = std::chrono::high_resolution_clock::now();
        auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
```

```
    cout << "Dummy Record of " << limit << " people have been created." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void searchDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Search Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();
    cout << "Details of the searched IDs:" << endl;

    for (int i = 0; i < 5; i++) {
        NodeBTree* temp = searchHelper(root, idsArr[i]);
        if (temp != NULL) {
            for (int j = 0; j < temp->numKeys; j++) {
                if (temp->records[j].id == idsArr[i]) {
                    cout << "ID: " << temp->records[j].id << endl;
                    cout << "Name: " << temp->records[j].name << endl;
                    cout << "Age: " << temp->records[j].age << endl << endl;
                    break;
                }
            }
        } else {
            cout << "ID " << idsArr[i] << " not found in the database." << endl;
        }
    }

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs (" << ids << ") have been searched and displayed." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

NodeBTree* searchDummyRecordHelper(NodeBTree* node, int target) {
    if (node == NULL) {
        return NULL;
    }

    int i = 0;
    while (i < node->numKeys && target > node->records[i].id) {
        i++;
    }

    // if the key is found in the current node
    if (i < node->numKeys && target == node->records[i].id) {
        return node;
    }

    // if the node is a leaf, return NULL
    if (node->isLeaf) {
        return NULL;
    }

    // otherwise, recurse on the appropriate child
    return searchDummyRecordHelper(node->children[i], target);
}
```

```
void updateDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Update Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < 5; i++) {
        updateDummyHelper(idsArr[i]);
    }

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs (" << ids << ") have been updated." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void updateDummyHelper(int target) {
    NodeBTree* temp = searchHelper(root, target);

    if (temp != NULL) {
        for (int i = 0; i < temp->numKeys; i++) {
            if (temp->records[i].id == target) {
                temp->records[i].name += " ka naya naam";
                srand(time(0));
                int age = rand() % 42 + 18;
                temp->records[i].age = age;
                cout << "------- Updated Details for ID " << target << " -------" << endl;
                cout << "Name: " << temp->records[i].name << endl;
                cout << "Age: " << temp->records[i].age << endl << endl;
                break;
            }
        }
    } else {
        cout << "ID " << target << " not found in the database." << endl;
    }
}

void deleteDummyRecord(string ids, int idsArr[]) {
    cout << "\n__________ Delete Dummy Record __________" << endl;
    auto startTime = std::chrono::high_resolution_clock::now();

    for (int i = 0; i < 5; i++) {
        deleteDummyRecordHelper(root, idsArr[i]);
    }

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime);
    cout << "5 IDs (" << ids << ") have been deleted." << endl;
    cout << "Duration: " << duration.count() << " milliseconds." << endl;
}

void deleteDummyRecordHelper(NodeBTree* node, int target) {
    if (node == NULL) {
        return;
    }
```

```
            int i = 0;
            while (i < node->numKeys && target > node->records[i].id) {
                i++;
            }

            if (i < node->numKeys && target == node->records[i].id) {
                // if targetID is found, remove it from the node
                for (int j = i; j < node->numKeys - 1; j++) {
                    node->records[j] = node->records[j + 1];
                }
                node->numKeys--;

                cout << "ID " << target << " has been deleted." << endl;
            } else if (!node->isLeaf) {
                // If it's not a leaf, move to the appropriate child
                deleteDummyRecordHelper(node->children[i], target);
            }
        }

        void performDummyOperations() {
            createDummyRecord();
            if (root == NULL) {
                cout << "Unable to perform operations ahead since there is no dummy record in the database."
<< endl;
                return;
            }

            int idsArray[5] = {limit / 5, (limit / 5) * 2, (limit / 5) * 3, (limit / 5) * 4, (limit / 5) * 5};
            string ids = "";
            for (int i = 0; i < 5; i++) {
                ids = ids + to_string(idsArray[i]) + ", ";
            }
            ids.erase(ids.length() - 2); // remove comma and space
            searchDummyRecord(ids, idsArray);
            updateDummyRecord(ids, idsArray);
            deleteDummyRecord(ids, idsArray);
        }
};

void printOptions () {
    cout << "1. Perform Dummy Operations" << endl;
    cout << "2. Create new/original record" << endl;
    cout << "3. Delete record" << endl;
    cout << "4. Display all records" << endl;
    cout << "5. Search a record" << endl;
    cout << "6. Update a record" << endl;
    cout << "7. Go back to select table" << endl;
    cout << "8. Exit Program" << endl;
    cout << "Your choice: ";
}

int main() {
    int tableChoice, choice;
```

```
    BST_Table bst;
    AVL_Table avl;
    BTree_Table btree;

    do {
        cout << "\n------------------ Muhammad Hammad || 23K-2005 -------------------";
        cout << "\n---------------------- Trees Database System ----------------------";
        cout << "\nSelect which table to work with:" << endl;
        cout << "1. Binary Search Tree (BST)" << endl;
        cout << "2. AVL Tree" << endl;
        cout << "3. B-Tree" << endl;
        cout << "4. Exit Program" << endl;
        cout << "Your choice: ";
        tableChoice = getInt();
        cin.ignore();

        if (tableChoice == 1) {
            do {
                cout << "\n------------------ Muhammad Hammad || 23K-2005 -------------------";
                cout << "\n---------------------- Trees Database System ----------------------";
                cout << "\nBST Table Operations:" << endl;
                printOptions();
                choice = getInt();
                cin.ignore();

                switch (choice) {
                    case 1:
                        // bst.createDummyRecord();
                        bst.performDummyOperations();
                        break;
                    case 2:
                        bst.createRecord();
                        break;
                    case 3:
                        bst.deleteNode();
                        break;
                    case 4:
                        bst.display();
                        break;
                    case 5:
                        bst.search();
                        break;
                    case 6:
                        bst.update();
                        break;
                    case 7:
                        break;
                    case 8:
                        cout << "\nExiting Program... Ba-bye!" << endl;
                        return 0;
                    default:
                        cout << "Invalid choice, try again." << endl;
                        break;
                }
```

```
        } while (choice != 7);
    }
    else if (tableChoice == 2) {
        do {
            cout << "\n------------------ Muhammad Hammad || 23K-2005 -------------------";
            cout << "\n---------------------- Trees Database System ----------------------";
            cout << "\nAVL Table Operations:" << endl;
            printOptions();
            choice = getInt();
            cin.ignore();

            switch (choice) {
                case 1:
                    avl.performDummyOperations();
                    break;
                case 2:
                    avl.createRecord();
                    break;
                case 3:
                    avl.deleteNode();
                    break;
                case 4:
                    avl.display();
                    break;
                case 5:
                    avl.search();
                    break;
                case 6:
                    avl.update();
                    break;
                case 7:
                    break;
                case 8:
                    cout << "\nExiting Program... Ba-bye!" << endl;
                    return 0;
                default:
                    cout << "Invalid choice, try again." << endl;
                    break;
            }
        } while (choice != 7);
    }
    else if (tableChoice == 3) {
        do {
            cout << "\n------------------ Muhammad Hammad || 23K-2005 -------------------";
            cout << "\n---------------------- Trees Database System ----------------------";
            cout << "\nBTree Table Operations:" << endl;
            printOptions();
            choice = getInt();
            cin.ignore();

            switch (choice) {
                case 1:
                    btree.performDummyOperations();
                    break;
```

```
                case 2:
                    btree.createRecord();
                    break;
                case 3:
                    btree.deleteNode();
                    break;
                case 4:
                    btree.display();
                    break;
                case 5:
                    btree.search();
                    break;
                case 6:
                    btree.update();
                    break;
                case 7:
                    break;
                case 8:
                    cout << "\nExiting Program... Ba-bye!" << endl;
                    return 0;
                default:
                    cout << "Invalid choice, try again." << endl;
                    break;
            }
        } while (choice != 7);
    }
    else if (tableChoice == 4) {
        cout << "\nExiting program... Ba-bye!" << endl;
        break;
    } else {
        cout << "Invalid choice, try again." << endl;
    }

} while (true);

return 0;
}
```

Muhammad Hammad (23K-2005)

Output:

BST Database Operations

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 1

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Hammad
Enter the age: 18

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Sami
Enter the age: 19
```

Muhammad Hammad (23K-2005)

```
----------------- Muhammad Hammad || 23K-2005 ------------------
--------------------- Trees Database System ---------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Faiq
Enter the age: 20

----------------- Muhammad Hammad || 23K-2005 ------------------
--------------------- Trees Database System ---------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Talal
Enter the age: 19

----------------- Muhammad Hammad || 23K-2005 ------------------
--------------------- Trees Database System ---------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Hammad
Age: 18

ID: 2
Name: Sami
Age: 19

ID: 3
```

```
ID: 3
Name: Faiq
Age: 20

ID: 4
Name: Talal
Age: 19


------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 6
__________ Update Record __________
Enter the ID that you want to update: 2
Enter new name for ID 2: Daniyal
Enter new age for ID 2: 20
Record for ID 2 has been updated.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Hammad
Age: 18

ID: 2
Name: Daniyal
Age: 20

ID: 3
Name: Faiq
Age: 20

ID: 4
Name: Talal
```

```
ID: 4
Name: Talal
Age: 19


------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 5
Delete failed. The ID was not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 3
ID 3 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 4
ID 4 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 2
ID 2 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 1
ID 1 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Unable to delete. The BST is empty.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
Unable to search. The BST is empty.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Sami
Enter the age: 19

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Hammad
Enter the age: 19
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 5
Name: Sami
Age: 19

ID: 6
Name: Hammad
Age: 19


------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search: 3
ID 3 not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 5
Name: Sami
Age: 19
```

```
__________ Display Record __________
ID: 5
Name: Sami
Age: 19

ID: 6
Name: Hammad
Age: 19


------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search: 1
ID 1 not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 5
Name: Sami
Age: 19

ID: 6
Name: Hammad
Age: 19


------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
```

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search: 5
ID 5 found in the database.
Details of ID 5:
Name: Sami
Age: 19

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 8

Exiting Program... Ba-bye!
```

AVL Tree Database Operations

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 2

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Hammad
Enter the age: Sami
Invalid entry. Please enter an integer: 18

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Sami
Enter the age: 19
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Faiq
Enter the age: 20

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Talal
Enter the age: 21

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Hammad
Age: 18

ID: 2
Name: Sami
Age: 19

ID: 3
```

Muhammad Hammad (23K-2005)

```
__________ Display Record __________
ID: 1
Name: Hammad
Age: 18

ID: 2
Name: Sami
Age: 19

ID: 3
Name: Faiq
Age: 20

ID: 4
Name: Talal
Age: 21


------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search: 3
ID 3 found in the database.
Details of ID 3:
Name: Faiq
Age: 20

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search: 10
ID 10 not found in the database.
```

Muhammad Hammad (23K-2005)

```
----------------- Muhammad Hammad || 23K-2005 ------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 6
__________ Update Record __________
Enter the ID that you want to update: 2
Enter new name for ID 2: Muhib
Enter new age for ID 2: 21
Record for ID 2 has been updated.

----------------- Muhammad Hammad || 23K-2005 ------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Hammad
Age: 18

ID: 2
Name: Muhib
Age: 21

ID: 3
Name: Faiq
Age: 20

ID: 4
Name: Talal
Age: 21
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 10
Deletion failed. ID not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 3
ID 3 has been deleted successfully.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 2
ID 2 has been deleted successfully.
```

Muhammad Hammad (23K-2005)

```
----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 1
ID 1 has been deleted successfully.

----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 4
ID 4 has been deleted successfully.

----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Unable to delete. The AVL Tree is empty.
```

Muhammad Hammad (23K-2005)

```
----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
Unable to search. The AVL Tree is empty.

----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Unable to delete. The AVL Tree is empty.

----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 7

----------------- Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 4

Exiting program... Ba-bye!
```

Muhammad Hammad (23K-2005)

B-Tree Database Operations

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 3

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Mike
Enter the age: 25

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: John
Enter the age: 15
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Carlos
Enter the age: 23

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 2
__________ Create Record __________
Enter the name: Kevin
Enter the age: 38

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Mike
Age: 25

ID: 2
Name: John
Age: 15

ID: 3
```

```
Age: 15

ID: 3
Name: Carlos
Age: 23

ID: 4
Name: Kevin
Age: 38


------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search in the database: 3
ID 3 found in the database.
Details of ID 3:
Name: Carlos
Age: 23

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 5
__________ Search Record __________
Enter the ID that you want to search in the database: 6
ID 6 not found in the database.
```

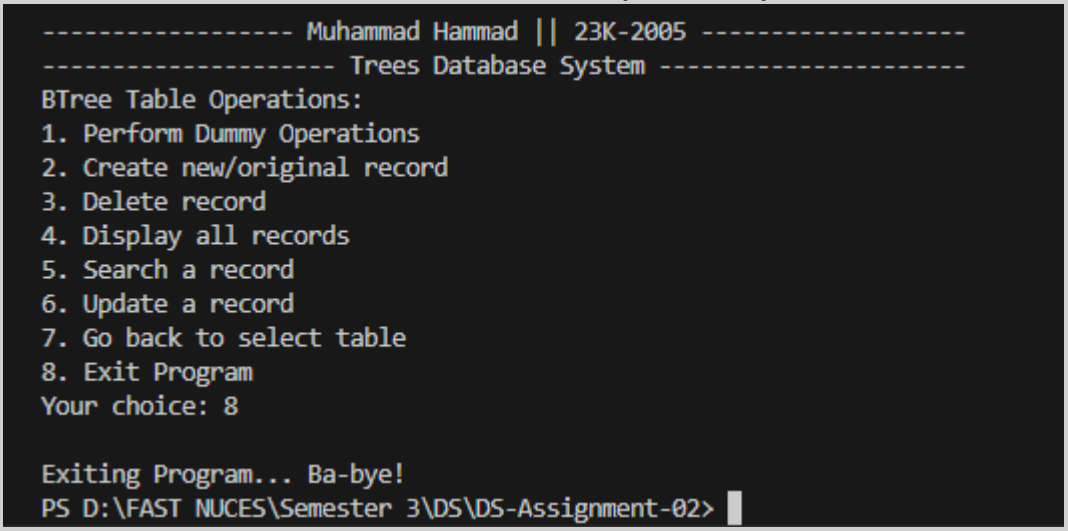
Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 6
__________ Update Record __________
Enter the ID that you want to update: 5
Update failed. The ID was not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 6
__________ Update Record __________
Enter the ID that you want to update: 3
Enter new name for ID 3: Mandela
Enter new age for ID 3: 26
Record updated successfully.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
__________ Display Record __________
ID: 1
Name: Mike
Age: 25

ID: 2
Name: John
Age: 15
```

```
ID: 3
Name: Mandela
Age: 26

ID: 4
Name: Kevin
Age: 38


------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 2
ID 2 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 1
ID 1 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 3
ID 3 has been deleted.
```

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 4
ID 4 has been deleted.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 3
Enter the ID that you want to delete: 10
Delete failed. The ID was not found in the database.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 4
Unable to display records. The B-Tree database is empty right now.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
```

Muhammad Hammad (23K-2005)



Muhammad Hammad (23K-2005)

Dummy Data Operations

```
------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 1

------------------ Muhammad Hammad || 23K-2005 -------------------
---------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 1

__________ Create Dummy Record __________
Dummy Record of 10000 people have been created.
Duration: 7016 milliseconds.

__________ Search Dummy Record __________
Details of the searched IDs:
ID: 500
Name: Dummy500
Age: 21

ID: 1000
Name: Dummy1000
Age: 21

ID: 1500
Name: Dummy1500
Age: 21

ID: 2000
Name: Dummy2000
Age: 24

ID: 2500
Name: Dummy2500
Age: 24

ID: 3000
Name: Dummy3000
Age: 24
```

```
ID: 3500
Name: Dummy3500
Age: 24

ID: 4000
Name: Dummy4000
Age: 24

ID: 4500
Name: Dummy4500
Age: 28

ID: 5000
Name: Dummy5000
Age: 28

ID: 5500
Name: Dummy5500
Age: 28

ID: 6000
Name: Dummy6000
Age: 31

ID: 6500
Name: Dummy6500
Age: 31

ID: 7000
Name: Dummy7000
Age: 34

ID: 7500
Name: Dummy7500
Age: 34

ID: 8000
Name: Dummy8000
Age: 37

ID: 8500
Name: Dummy8500
Age: 37

ID: 9000
Name: Dummy9000
Age: 41

ID: 9500
Name: Dummy9500
Age: 44
```

```
ID: 10000
Name: Dummy10000
Age: 44

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been searched and displayed.
Duration: 27 milliseconds.

__________ Update Dummy Record __________
------- Updated Details for ID 500 -------
Name: Dummy500 ka naya naam
Age: 44

------- Updated Details for ID 1000 -------
Name: Dummy1000 ka naya naam
Age: 44

------- Updated Details for ID 1500 -------
Name: Dummy1500 ka naya naam
Age: 44

------- Updated Details for ID 2000 -------
Name: Dummy2000 ka naya naam
Age: 44

------- Updated Details for ID 2500 -------
Name: Dummy2500 ka naya naam
Age: 44

------- Updated Details for ID 3000 -------
Name: Dummy3000 ka naya naam
Age: 44

------- Updated Details for ID 3500 -------
Name: Dummy3500 ka naya naam
Age: 44

------- Updated Details for ID 4000 -------
Name: Dummy4000 ka naya naam
Age: 44

------- Updated Details for ID 4500 -------
Name: Dummy4500 ka naya naam
Age: 44

------- Updated Details for ID 5000 -------
Name: Dummy5000 ka naya naam
Age: 44

------- Updated Details for ID 5500 -------
Name: Dummy5500 ka naya naam
Age: 44
```

```
------- Updated Details for ID 6000 -------
Name: Dummy6000 ka naya naam
Age: 44

------- Updated Details for ID 6500 -------
Name: Dummy6500 ka naya naam
Age: 44

------- Updated Details for ID 7000 -------
Name: Dummy7000 ka naya naam
Age: 44

------- Updated Details for ID 7500 -------
Name: Dummy7500 ka naya naam
Age: 44

------- Updated Details for ID 8000 -------
Name: Dummy8000 ka naya naam
Age: 44

------- Updated Details for ID 8500 -------
Name: Dummy8500 ka naya naam
Age: 44

------- Updated Details for ID 9000 -------
Name: Dummy9000 ka naya naam
Age: 44

------- Updated Details for ID 9500 -------
Name: Dummy9500 ka naya naam
Age: 44

------- Updated Details for ID 10000 -------
Name: Dummy10000 ka naya naam
Age: 44

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been updated.
Duration: 30 milliseconds.

__________ Delete Dummy Record __________
ID 500 has been deleted.
ID 1000 has been deleted.
ID 1500 has been deleted.
ID 2000 has been deleted.
ID 2500 has been deleted.
ID 3000 has been deleted.
ID 3500 has been deleted.
ID 4000 has been deleted.
ID 4500 has been deleted.
ID 5000 has been deleted.
ID 5500 has been deleted.
ID 6000 has been deleted.
ID 6500 has been deleted.
```

```
ID 6500 has been deleted.
ID 7000 has been deleted.
ID 7500 has been deleted.
ID 8000 has been deleted.
ID 8500 has been deleted.
ID 9000 has been deleted.
ID 9500 has been deleted.
ID 10000 has been deleted.
20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been deleted.
Duration: 8 milliseconds.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BST Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 7

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 2

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 1
__________ Create Dummy Record __________
Dummy Record of 10000 people have been created.

__________ Search Dummy Record __________
Details of the searched IDs:
ID: 500
Name: Dummy500
Age: 25
```

```
ID: 1000
Name: Dummy1000
Age: 25

ID: 1500
Name: Dummy1500
Age: 25

ID: 2000
Name: Dummy2000
Age: 25

ID: 2500
Name: Dummy2500
Age: 25

ID: 3000
Name: Dummy3000
Age: 25

ID: 3500
Name: Dummy3500
Age: 25

ID: 4000
Name: Dummy4000
Age: 25

ID: 4500
Name: Dummy4500
Age: 25

ID: 5000
Name: Dummy5000
Age: 25

ID: 5500
Name: Dummy5500
Age: 25

ID: 6000
Name: Dummy6000
Age: 25

ID: 6500
Name: Dummy6500
Age: 25

ID: 7000
Name: Dummy7000
Age: 25

ID: 7500
```

```
ID: 7500
Name: Dummy7500
Age: 25

ID: 8000
Name: Dummy8000
Age: 25

ID: 8500
Name: Dummy8500
Age: 25

ID: 9000
Name: Dummy9000
Age: 25

ID: 9500
Name: Dummy9500
Age: 25

ID: 10000
Name: Dummy10000
Age: 25

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been searched and displayed.
Duration: 25 milliseconds.

__________ Update Dummy Record __________
------- Updated Details for ID 500 -------
Name: Dummy500 ka naya naam
Age: 25

------- Updated Details for ID 1000 -------
Name: Dummy1000 ka naya naam
Age: 25

------- Updated Details for ID 1500 -------
Name: Dummy1500 ka naya naam
Age: 25

------- Updated Details for ID 2000 -------
Name: Dummy2000 ka naya naam
Age: 25

------- Updated Details for ID 2500 -------
Name: Dummy2500 ka naya naam
Age: 25

------- Updated Details for ID 3000 -------
Name: Dummy3000 ka naya naam
Age: 25
```

```
------- Updated Details for ID 3500 -------
Name: Dummy3500 ka naya naam
Age: 25

------- Updated Details for ID 4000 -------
Name: Dummy4000 ka naya naam
Age: 25

------- Updated Details for ID 4500 -------
Name: Dummy4500 ka naya naam
Age: 25

------- Updated Details for ID 5000 -------
Name: Dummy5000 ka naya naam
Age: 25

------- Updated Details for ID 5500 -------
Name: Dummy5500 ka naya naam
Age: 25

------- Updated Details for ID 6000 -------
Name: Dummy6000 ka naya naam
Age: 25

------- Updated Details for ID 6500 -------
Name: Dummy6500 ka naya naam
Age: 25

------- Updated Details for ID 7000 -------
Name: Dummy7000 ka naya naam
Age: 25

------- Updated Details for ID 7500 -------
Name: Dummy7500 ka naya naam
Age: 25

------- Updated Details for ID 8000 -------
Name: Dummy8000 ka naya naam
Age: 25

------- Updated Details for ID 8500 -------
Name: Dummy8500 ka naya naam
Age: 25

------- Updated Details for ID 9000 -------
Name: Dummy9000 ka naya naam
Age: 25

------- Updated Details for ID 9500 -------
Name: Dummy9500 ka naya naam
Age: 25
```

```
------- Updated Details for ID 10000 -------
Name: Dummy10000 ka naya naam
Age: 25

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been updated.
Duration: 25 milliseconds.

__________ Delete Dummy Record __________
ID 500 has been deleted.
ID 1000 has been deleted.
ID 1500 has been deleted.
ID 2000 has been deleted.
ID 2500 has been deleted.
ID 3000 has been deleted.
ID 3500 has been deleted.
ID 4000 has been deleted.
ID 4500 has been deleted.
ID 5000 has been deleted.
ID 5500 has been deleted.
ID 6000 has been deleted.
ID 6500 has been deleted.
ID 7000 has been deleted.
ID 7500 has been deleted.
ID 8000 has been deleted.
ID 8500 has been deleted.
ID 9000 has been deleted.
ID 9500 has been deleted.
ID 10000 has been deleted.
20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been deleted.
Duration: 9 milliseconds.

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
AVL Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 7

------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
Select which table to work with:
1. Binary Search Tree (BST)
2. AVL Tree
3. B-Tree
4. Exit Program
Your choice: 3
```

Muhammad Hammad (23K-2005)

```
------------------ Muhammad Hammad || 23K-2005 -------------------
--------------------- Trees Database System ----------------------
BTree Table Operations:
1. Perform Dummy Operations
2. Create new/original record
3. Delete record
4. Display all records
5. Search a record
6. Update a record
7. Go back to select table
8. Exit Program
Your choice: 1
__________ Create Dummy Record __________
Dummy Record of 10000 people have been created.
Duration: 52 milliseconds.

__________ Search Dummy Record __________
Details of the searched IDs:
ID: 500
Name: Dummy500
Age: 48

ID: 1000
Name: Dummy1000
Age: 48

ID: 1500
Name: Dummy1500
Age: 48

ID: 2000
Name: Dummy2000
Age: 48

ID: 2500
Name: Dummy2500
Age: 48

ID: 3000
Name: Dummy3000
Age: 48

ID: 3500
Name: Dummy3500
Age: 48

ID: 4000
Name: Dummy4000
Age: 48

ID: 4500
Name: Dummy4500
Age: 48
```

```
ID: 5000
Name: Dummy5000
Age: 48

ID: 5500
Name: Dummy5500
Age: 48

ID: 6000
Name: Dummy6000
Age: 48

ID: 6500
Name: Dummy6500
Age: 48

ID: 7000
Name: Dummy7000
Age: 48

ID: 7500
Name: Dummy7500
Age: 48

ID: 8000
Name: Dummy8000
Age: 48

ID: 8500
Name: Dummy8500
Age: 48

ID: 9000
Name: Dummy9000
Age: 48

ID: 9500
Name: Dummy9500
Age: 48

ID: 10000
Name: Dummy10000
Age: 48

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been searched and displayed.
Duration: 26 milliseconds.

__________ Update Dummy Record __________
------- Updated Details for ID 500 -------
Name: Dummy500 ka naya naam
Age: 48
```

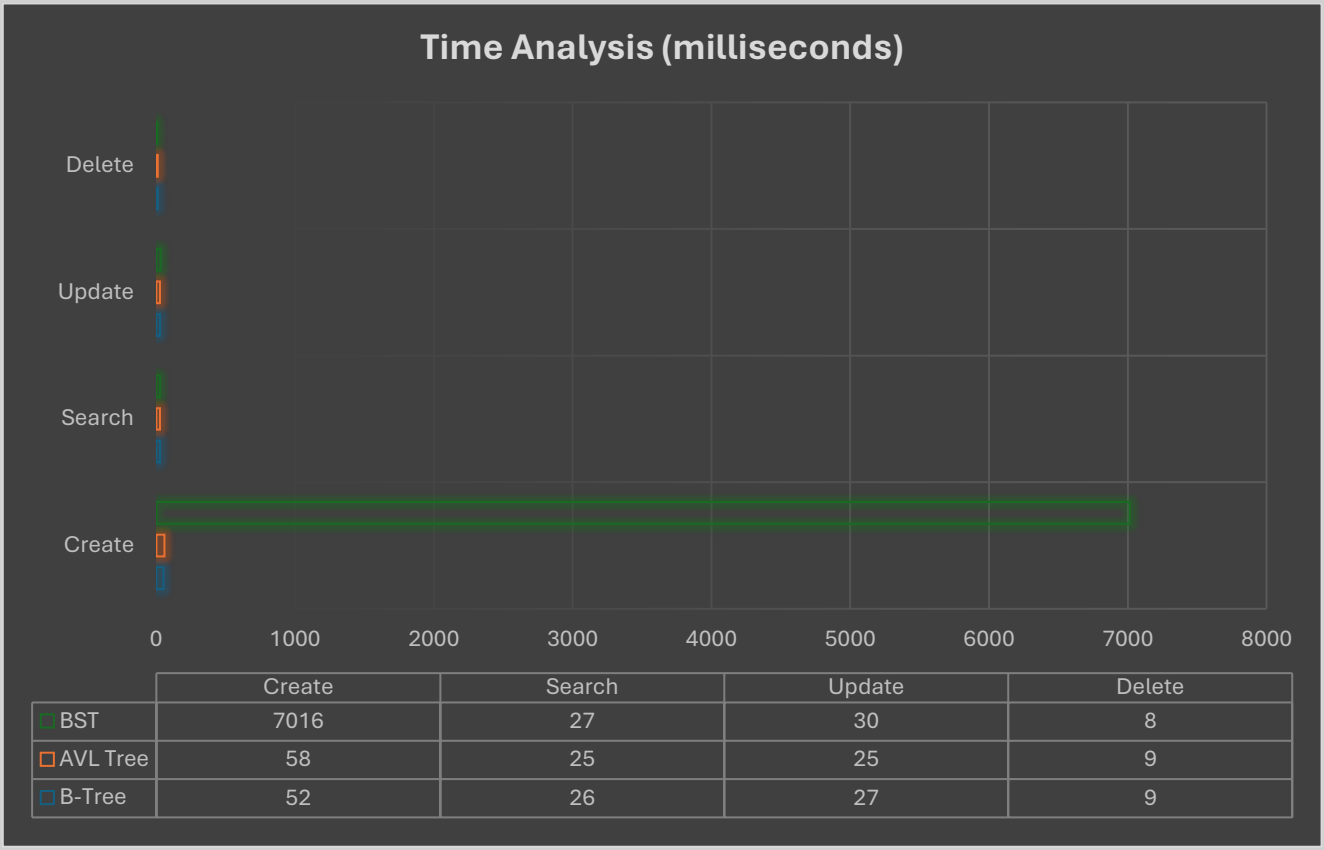
```
------- Updated Details for ID 1000 -------
Name: Dummy1000 ka naya naam
Age: 48

------- Updated Details for ID 1500 -------
Name: Dummy1500 ka naya naam
Age: 48

------- Updated Details for ID 2000 -------
Name: Dummy2000 ka naya naam
Age: 48

------- Updated Details for ID 2500 -------
Name: Dummy2500 ka naya naam
Age: 48

------- Updated Details for ID 3000 -------
Name: Dummy3000 ka naya naam
Age: 48

------- Updated Details for ID 3500 -------
Name: Dummy3500 ka naya naam
Age: 48

------- Updated Details for ID 4000 -------
Name: Dummy4000 ka naya naam
Age: 48

------- Updated Details for ID 4500 -------
Name: Dummy4500 ka naya naam
Age: 48

------- Updated Details for ID 5000 -------
Name: Dummy5000 ka naya naam
Age: 48

------- Updated Details for ID 5500 -------
Name: Dummy5500 ka naya naam
Age: 48

------- Updated Details for ID 6000 -------
Name: Dummy6000 ka naya naam
Age: 48

------- Updated Details for ID 6500 -------
Name: Dummy6500 ka naya naam
Age: 48

------- Updated Details for ID 7000 -------
Name: Dummy7000 ka naya naam
Age: 48

------- Updated Details for ID 7500 -------
Name: Dummy7500 ka naya naam
```

```
------- Updated Details for ID 7500 -------
Name: Dummy7500 ka naya naam
Age: 48

------- Updated Details for ID 8000 -------
Name: Dummy8000 ka naya naam
Age: 48

------- Updated Details for ID 8500 -------
Name: Dummy8500 ka naya naam
Age: 48

------- Updated Details for ID 9000 -------
Name: Dummy9000 ka naya naam
Age: 48

------- Updated Details for ID 9500 -------
Name: Dummy9500 ka naya naam
Age: 48

------- Updated Details for ID 10000 -------
Name: Dummy10000 ka naya naam
Age: 48

20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been updated.
Duration: 27 milliseconds.

__________ Delete Dummy Record __________
ID 500 has been deleted.
ID 1000 has been deleted.
ID 1500 has been deleted.
ID 2000 has been deleted.
ID 2500 has been deleted.
ID 3000 has been deleted.
ID 3500 has been deleted.
ID 4000 has been deleted.
ID 4500 has been deleted.
ID 5000 has been deleted.
ID 5500 has been deleted.
ID 6000 has been deleted.
ID 6500 has been deleted.
ID 7000 has been deleted.
ID 7500 has been deleted.
ID 8000 has been deleted.
ID 8500 has been deleted.
ID 9000 has been deleted.
ID 9500 has been deleted.
ID 10000 has been deleted.
20 IDs (500, 1000, 1500, 2000, 2500, 3000, 3500, 4000, 4500, 5000, 5500, 6000, 6500, 7000, 7500, 8000, 8500, 9000, 9500, 10000) have been deleted.
Duration: 9 milliseconds.
```

Muhammad Hammad (23K-2005)

Time Analysis:

Graph:



	Create	Search	Update	Delete
BST	7016	27	30	8
AVL Tree	58	25	25	9
B-Tree	52	26	27	9

Analysis Summary:

- **Create:** AVL and B-Trees are much faster (~50 ms) than BST (7016 ms).
- **Search:** Similar times; AVL is slightly faster (25 ms).
- **Update:** AVL is fastest (25 ms), followed by B-Trees (27 ms) and BST (30 ms).
- **Delete:** All perform equally (~9 ms).

Submitted by: Muhammad Hammad

Roll no.: 23K-2005