Date: 05/02/2021

Practical 3

2CSDE75 - Advanced Data Structures

Name: Shrey Viradiya

Roll No: 18BCE259

Aim:

Re-balancing operation can be delayed until a certain threshold is attained. Scapegoat tree uses partial rebuilding for balancing a search tree. Implement scapegoat tree to demonstrate the partial rebuilding operation.

Code:

Prac3_ScapeGoat.cpp

```cpp
// Rebalancing operation can be delayed until a certain threshold is attained.
// Scapegoat tree uses partial rebuilding for balancing a search tree.
// Implement scapegoat tree to demonstrate the partial rebuilding operation.

#include "ScapeGoat.h"
#include <iostream>

int main(){
    ScapeGoat data("Ajino Motado");

    // data.AddData("data.csv", 1);
    for (int i = 0; i < 10; i++)
    {
        data.insert(i,i*10);
        data.PrettyPrinting();
        std::cout << std::endl;
    }

    // data.traverse();
    data.PrettyPrinting();

    std::cout << "Searching" << std::endl;
    std::cout << "================" << std::endl;
    std::cout << data.search(809) << std::endl;

    std::cout << "After Deleting Some Keys" << std::endl;
    data.deleteKey(5);
    data.deleteKey(3);
    data.deleteKey(0);
    data.deleteKey(1);
    data.PrettyPrinting();
    data.deleteKey(9);
    data.deleteKey(7);
    data.PrettyPrinting();

    return 0;
}
```

ScapeGoat.h

```cpp
#pragma once
#include <iostream>
#include <cstring>
#include <string>
```

```cpp
#include <fstream>
#include <vector>
#include <stdexcept> // std::runtime_error
#include <sstream>
#include "ScapeGoatUtilities.h"
// This header file contains the code for ScapeGoat tree

class ScapeGoat
{
    private:
        char name[50];
        int n, q;
        ScapeGoatNode *root;
    public:
        ScapeGoat(const char n[50]);
        ~ScapeGoat();
        void AddData(std::string filename, int isHeading);
        void insert(int key, int object);
        void traverse(int mode);
        void deleteKey(int key);
        int search(int key);
        void PrettyPrinting();
};

ScapeGoat::ScapeGoat(const char nameinput[50]){
    strcpy(name, nameinput);
    n = 0;
    q = 0;
    root = nullptr;
}

ScapeGoat::~ScapeGoat()
{
    using namespace std;
    delete root;
    cout << "Memory Released of " << name << endl;
}

int ScapeGoat::search(int key){
    ScapeGoatNode* node = root;

    while (node)
    {
        if (node->key == key)
        {
            return node->object;
        }
        else if(key < node->key){
```

```
            node = node->left;
        }
        else{
            node = node->right;
        }
    }
    return 0;
}

void ScapeGoat::AddData(std::string filename, int isHeading = 1){
    using namespace std;
    // working with csv in CPP
    // https://www.gormanalysis.com/blog/reading-and-writing-csv-files-with-cpp/

    ifstream myFile(filename);
    // if(!myFile.is_open()) throw runtime_error("Could not open file");

    string line, word;
    int val;

    if (isHeading)  getline(myFile, line);

    // Read data, line by line
    while(getline(myFile, line))
    {
        // Create a stringstream of the current line
        stringstream ss(line);
        pair<int, int> data;

        // add the column data
        // of a row to a pair
        getline(ss, word, ',');
        data.first = stoi(word);

        getline(ss, word, ',');
        data.second = stoi(word);

        insert(data.first, data.second);
    }

    // Close file
    myFile.close();
}

void ScapeGoat::traverse(int mode = 1){
    using namespace std;

    cout << "\n\nPrinting The ScapeGoatTree: " << name << endl;
```

```
    cout << "============================" << endl;
    cout << "Key --> Value" << endl;
    cout << "============================" << endl;
    if (mode == 0){
        cout << "Preorder" << endl;
        traversePreorder(root);
    }
    else if (mode == 1){
        cout << "Inorder" << endl;
        traverseInorder(root);
    }
    else if (mode == 2){
        cout << "Postorder" << endl;
        traversePostorder(root);
    }
    else{
        cout << "Invalid Mode" << endl;
        cout << "Inorder" << endl;
        traverseInorder(root);
    }
    cout << "============================\n" << endl;
}

void ScapeGoat::PrettyPrinting(){
    using namespace std;

    cout << "\n\nPrinting The ScapeGoatTree: " << name << endl;
    cout << "n: " << n << endl;
    cout << "q: " << q << endl;
    cout << "============================" << endl;
    cout << "Key --> Value" << endl;
    cout << "============================" << endl;
    printBT("", root, false);
}

void ScapeGoat::insert(int k, int o){
    ScapeGoatNode *newnode = new ScapeGoatNode(k, o);

    ScapeGoatNode *iter = root;
    if (iter == nullptr)
    {
        root = newnode;
        n++;
        q++;
        return;
    }

    bool done = false;
```

```
int d = 0;
do
{
    if (newnode->key < iter->key)
    {
        if (iter->left == nullptr)
        {
            iter->left = newnode;
            newnode->parent = iter;
            done = true;
        }
        else
        {
            iter= iter->left;
        }
    }
    else if (newnode->key > iter->key)
    {
        if (iter->right == nullptr)
        {
            iter->right = newnode;
            newnode->parent = iter;
            done = true;
        }
        else
        {
            iter = iter->right;
        }
    }
    else
    {
        iter->object = newnode->object;
        delete newnode;
        return;
    }
    d++;
}
while (!done);

n++;
q++;

if (d > log32(q))
{
    ScapeGoatNode *iter2 = newnode->parent;
    while (3 * size(iter2) <= 2 * size(iter2->parent))
        iter2 = iter2->parent;
```

```
        ScapeGoatNode *u = iter2->parent;
        int ns = size(u);
        ScapeGoatNode *p = u->parent;
        ScapeGoatNode **a = new ScapeGoatNode* [ns];
        packIntoArray(u, a, 0);
        if (p == nullptr)
        {
            root = buildBalanced(a, 0, ns);
            root->parent = nullptr;
        }
        else if (p->right == u)
        {
            p->right = buildBalanced(a, 0, ns);
            p->right->parent = p;
        }
        else
        {
            p->left = buildBalanced(a, 0, ns);
            p->left->parent = p;
        }
    }
}

void ScapeGoat::deleteKey(int key){
    ScapeGoatNode * iter = root;

    while (iter->key != key && iter != nullptr )
    {
        if(key < iter->key ){
            iter = iter->left;
        }
        else if(key > iter->key){
            iter = iter->right;
        }
    }

    if (iter == nullptr)
    {
        std::cout << "Key Not Found" << std::endl;
        return;
    }

    if( (iter->left == nullptr) || (iter->right == nullptr) )
        {
            ScapeGoatNode * temp = iter->left ?
                        iter->left :
                        iter->right;
```

```
        if (temp == nullptr)
        {
            // iter
            if (iter->parent->left == iter)
            {
                iter->parent->left = nullptr;
            }
            else{
                iter->parent->right = nullptr;
            }
            temp = iter;
            iter = nullptr;
        }
        else {
            temp->parent = iter->parent;
            if (iter->parent != nullptr){
                if (iter->parent->left == iter)
                {
                    iter->parent->left = temp;
                }
                else{
                    iter->parent->right = temp;
                }
            }
            else{
                root = temp;
            }
            iter = nullptr;
        }

    }
    else
    {
        std::pair<int, int> minimum = findMin(iter->right);
        iter->key = minimum.first;
        iter->object = minimum.second;
    }

if ((2*n) > q && n <= q)
{
    n--;
}
else{
    ScapeGoatNode *u = root;
    int ns = size(u);
    ScapeGoatNode *p = u->parent;
    ScapeGoatNode **a = new ScapeGoatNode* [ns];
    packIntoArray(u, a, 0);
```

```
        if (p == nullptr)
        {
            root = buildBalanced(a, 0, ns);
            root->parent = nullptr;
        }
        else if (p->right == u)
        {
            p->right = buildBalanced(a, 0, ns);
            p->right->parent = p;
        }
        else
        {
            p->left = buildBalanced(a, 0, ns);
            p->left->parent = p;
        }
        n--;
        q=n;
    }
}
```

ScapeGoatUtilities.h

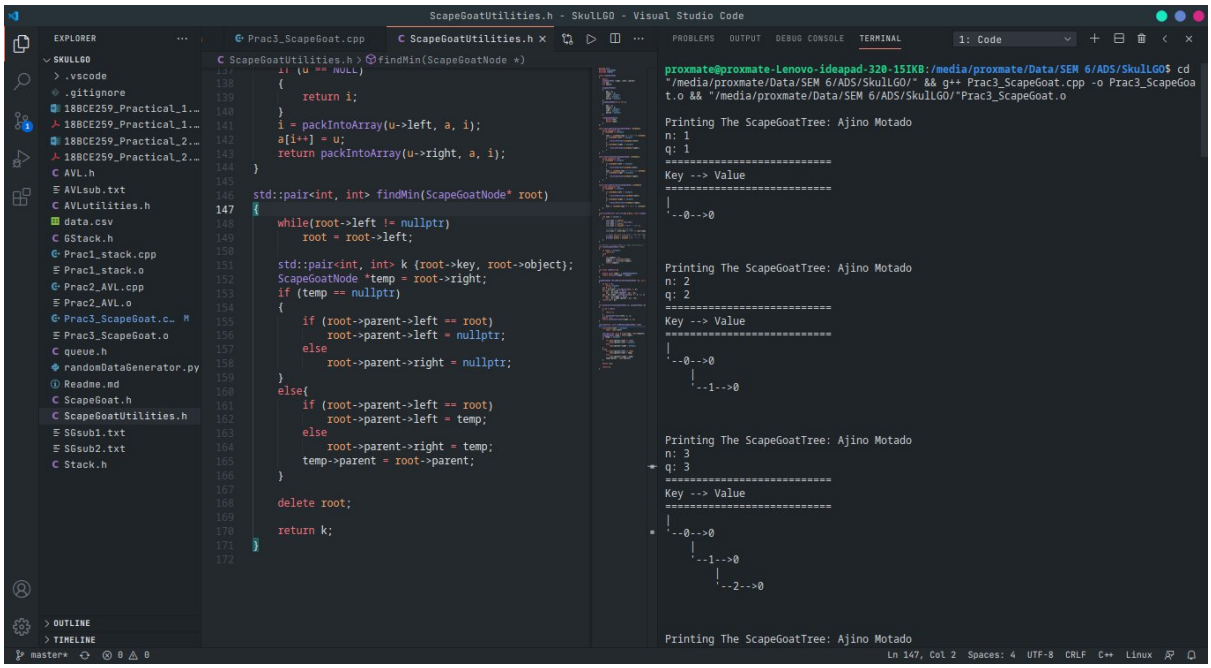
```
#pragma once
#include <iostream>
#include <cmath>

class ScapeGoatNode
{
    public:
    ScapeGoatNode *right, *left, *parent;
    int key;
    int object;

    ScapeGoatNode()
    {
        key = 0;
        object = 0;
        right = nullptr;
        left = nullptr;
        parent = nullptr;
    }
    ScapeGoatNode(int k, int o)
    {
        key = k;
        object = 0;
        right = nullptr;
        left = nullptr;
        parent = nullptr;
```

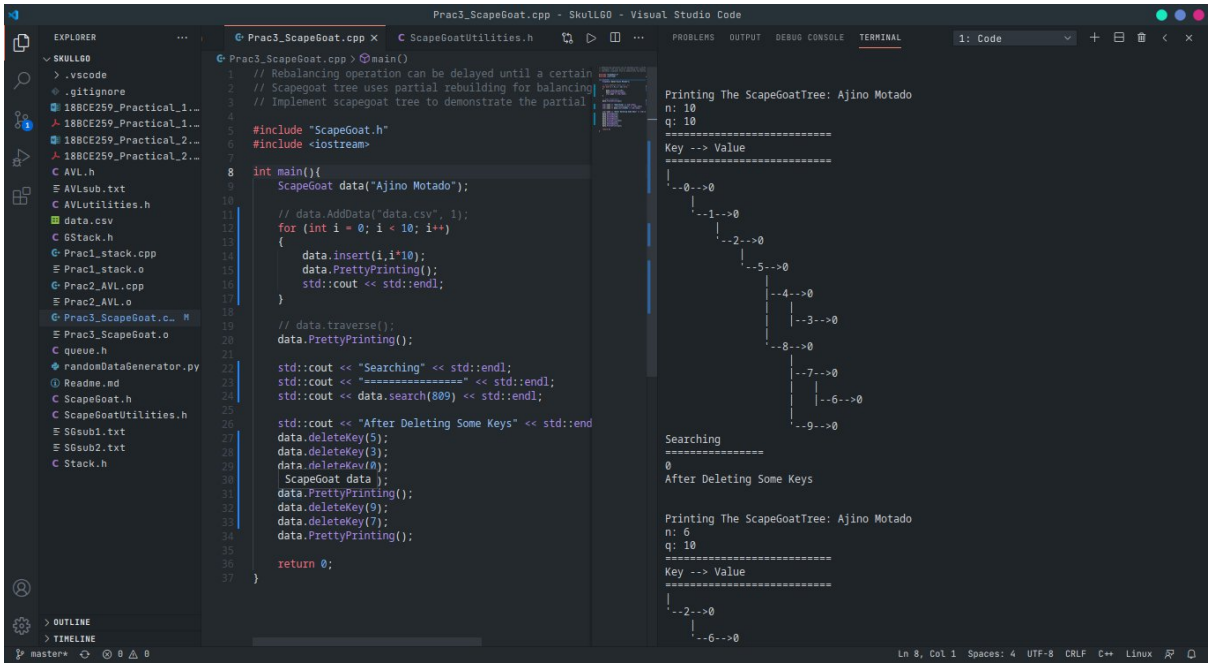
```
    }

    ~ScapeGoatNode(){
        delete left;
        delete right;
    }
};

void traversePreorder(ScapeGoatNode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        cout << rootNode->key << " --> " << rootNode->object << endl;
        if (rootNode->left != nullptr)
        {
            traversePreorder(rootNode->left);
        }
        if (rootNode->right != nullptr)
        {
            traversePreorder(rootNode->right);
        }
    }
}

void traverseInorder(ScapeGoatNode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        if (rootNode->left != nullptr)
        {
            traverseInorder(rootNode->left);
        }
        cout << rootNode->key << " --> " << rootNode->object << endl;
        if (rootNode->right != nullptr)
        {
            traverseInorder(rootNode->right);
        }
    }
}

void traversePostorder(ScapeGoatNode* rootNode){
    using namespace std;
    if (rootNode != nullptr)
    {
        if (rootNode->left != nullptr)
        {
            traversePostorder(rootNode->left);
        }
```

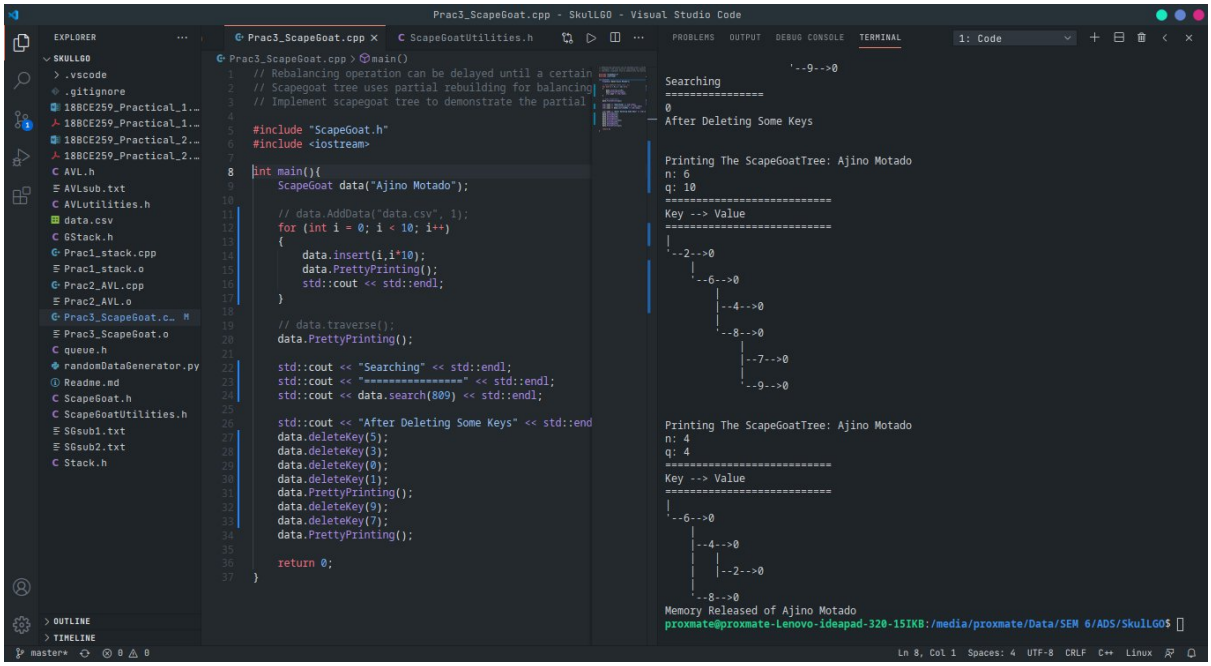
```
        if (rootNode->right != nullptr)
        {
            traversePostorder(rootNode->right);
        }
        cout << rootNode->key << " --> " << rootNode->object << endl;
    }
}

void printBT(const std::string& prefix, const ScapeGoatNode* node, bool isLeft)
{
    if( node != nullptr )
    {
        std::cout << prefix;
        std::cout << "|" << std::endl;
        std::cout << prefix;
        std::cout << (isLeft ? "|--" : "'--" );

        // print the value of the node
        std::cout << node->key << "-->" << node->object << std::endl;

        // enter the next tree level - left and right branch
        printBT( prefix + (isLeft ? "|   " : "    ") , node->left, true);
        printBT( prefix + (isLeft ? "|   " : "    ") , node->right, false);
    }
}

/* Function to count number of nodes recursively */
int size(ScapeGoatNode *root)
{
    if (root == nullptr)
        return 0;
    else
    {
        int numbers = 1;
        numbers += size(root->left);
        numbers += size(root->right);
        return numbers;
    }
}

int const log32(int q)
{
    double const log23 = 2.4663034623764317;
    return (int)ceil(log23 * log(q));
}

ScapeGoatNode *buildBalanced(ScapeGoatNode **a, int i, int ns)
{
```

```
    if (ns == 0)
        return nullptr;
    int m = ns / 2;
    a[i + m]->left = buildBalanced(a, i, m);
    if (a[i + m]->left != NULL)
        a[i + m]->left->parent = a[i + m];
    a[i + m]->right = buildBalanced(a, i + m + 1, ns - m - 1);\
    if (a[i + m]->right != NULL)
        a[i + m]->right->parent = a[i + m];
    return a[i + m];
}

int packIntoArray(ScapeGoatNode *u, ScapeGoatNode *a[], int i)
{
    if (u == NULL)
    {
        return i;
    }
    i = packIntoArray(u->left, a, i);
    a[i++] = u;
    return packIntoArray(u->right, a, i);
}

std::pair<int, int> findMin(ScapeGoatNode* root)
{
    while(root->left != nullptr)
        root = root->left;
    std::pair<int, int> k {root->key, root->object};
    ScapeGoatNode *temp = root->right;
    if (temp == nullptr)
    {
        if (root->parent->left == root)
            root->parent->left = nullptr;
        else
            root->parent->right = nullptr;
    }
    else{
        if (root->parent->left == root)
            root->parent->left = temp;
        else
            root->parent->right = temp;
        temp->parent = root->parent;
    }

    delete root;
    return k;
}
```

Snapshot of the output:







Conclusion:

With delayed re-balancing, unnecessary overhead on the system is avoided. This makes ScapeGoat tree a better implementation.