**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ**

Федеральное государственное бюджетное образовательное учреждение   
высшего образования

**«Сибирский государственный университет науки и технологий   
имени академика М.Ф. Решетнева»**

Институт информатики и телекоммуникаций

Кафедра информатики и вычислительной техники

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ**

Алгоритмы и структуры данных

|  |
| --- |
| Лабораторная 4 – Бинарные деревья |

Руководитель В. В. Тынченко

подпись, дата инициалы, фамилия

Обучающийся БПИ20-02, 201219047 Р. А. Сухачев

номер группы, зачетной книжки подпись, дата инициалы, фамилия

Красноярск 2021 г.

# ЦЕЛЬ РАБОТЫ

Изучение метода рекурсивной обработки данных, знакомство с принципами организации и обработки бинарных деревьев.

# порядок выполнения работы

1. Изучите теоретические сведения к данной лабораторной работе (см. прикрепленный файл "Бинарное дерево").
2. Ознакомьтесь с постановкой задачи (см. прикрепленный файл "Задание Лабораторная 4). Вариант задания соответствует вашему номеру в списке группы.
3. Составьте программу на языке С++ и выполните ее тестирование.

Подготовьте отчет по лабораторной работе. Отчет должен включать в себя:

-  титульный лист;

-  цель лабораторной работы;

-  постановку задачи из варианта;

-    текст программы с комментариями;

-   наборы тестовых исходных данных с соответствующими результатами работы программы (снимки экрана);

-    краткие ответы на контрольные вопросы (перед ответом надо включать в отчет формулировку вопроса);

-    выводы по лабораторной работе.

4. В качестве ответа на задание требуется прикрепить в систему следующие файлы:

    1) отчет в формате pdf;

  2) программный проект (архив).

5. Защитите работу перед преподавателем.

# постановка задачи

Выполнить задание в соответствии с вариантом. Для решения поставленной задачи разработать и использовать шаблон класса «Бинарное дерево поиска».

***Для оценки «удовлетворительно».***

Шаблон класса «Бинарное дерево поиска» должен включать в себя необходимый минимум методов, обеспечивающий полноценное функционирование объектов указанного класса, а именно:

* конструктор по умолчанию;
* конструктор копирования;
* деструктор;
* добавление узла;
* удаление узла;
* поиск узла с заданным значением ключа;

Используя механизм перегрузки стандартных операций, реализовать:

* вывод дерева на экран в удобном для восприятия человеком виде;
* ввод данных в дерево с клавиатуры;
* сравнение двух деревьев;
* присваивание данных одного дерева другому дереву;
* решение задачи согласно варианту задания.

***Для оценки «хорошо».***

Дополнить разработанный класс следующими функциональными возможностями:

* прямой обход дерева (корень, левое поддерево, правое поддерево);
* обратный обход дерева (левое поддерево, корень, правое поддерево);
* концевой обход дерева (левое поддерево, правое поддерево, корень).
* вывод дерева в файл в удобном для восприятия человеком виде;
* ввод данных в дерево из файла;

***Для оценки «отлично».***

На базе разработанного класса «Бинарное дерево поиска», реализовать производный класс «Сбалансированное бинарное дерево», добавив показатель балансировки для узлов дерева и метод, выполняющий при необходимости повторную балансировку дерева.

# ХОД РАБОТЫ

Выполнил задание в соответствии с вариантом. Для решения поставленной задачи разработал и использовал шаблон класса «Бинарное дерево поиска».

Шаблон класса «Бинарное дерево поиска» включает в себя необходимый минимум методов, обеспечивающий полноценное функционирование объектов указанного класса, а именно:

* конструктор по умолчанию;
* конструктор копирования;
* деструктор;
* добавление узла;
* удаление узла;
* поиск узла с заданным значением ключа;

Используя механизм перегрузки стандартных операций, реализовать:

* вывод дерева на экран в удобном для восприятия человеком виде;
* ввод данных в дерево с клавиатуры;
* сравнение двух деревьев;
* присваивание данных одного дерева другому дереву;
* решение задачи согласно варианту задания.

Дополнил разработанный класс следующими функциональными возможностями:

* прямой обход дерева (корень, левое поддерево, правое поддерево);
* обратный обход дерева (левое поддерево, корень, правое поддерево);
* концевой обход дерева (левое поддерево, правое поддерево, корень).
* вывод дерева в файл в удобном для восприятия человеком виде;
* ввод данных в дерево из файла;

На базе разработанного класса «Бинарное дерево поиска», реализовал производный класс «Сбалансированное бинарное дерево», добавив показатель балансировки для узлов дерева и метод, выполняющий при необходимости повторную балансировку дерева.

**BinaryTree.сpp:**

#pragma once

#include <iostream>

#include <vector>

#define SPACE 10

using namespace std;

template <class Data>

class TreeNode

{

public:

Data value; // data(key)

TreeNode \*left;

TreeNode \*right;

TreeNode()

{

value = 0;

left = nullptr;

right = nullptr;

}

TreeNode(Data v)

{

value = v;

left = nullptr;

right = nullptr;

}

~TreeNode() {}

};

template <class Type>

class BinaryTree

{

public:

TreeNode<Type> \*root;

BinaryTree()

{

root = nullptr;

}

bool isTreeEmpty() // check if empty

{

if (root == nullptr)

return true;

else

return false;

}

void add(TreeNode<Type> \*&t, int n)

{ // add node

if (t == NULL)

{

t = new TreeNode<Type>(n);

}

else

{

if (n < t->value)

{

add(t->left, n);

}

else

{

add(t->right, n);

}

}

}

void addFarray(int n)

{ // add from array

add(root, n);

}

TreeNode<Type> \*insertRecursive(TreeNode<Type> \*r, TreeNode<Type> \*new\_node)

{

if (r == nullptr)

{

r = new\_node;

return r;

}

if (new\_node->value < r->value)

{

r->left = insertRecursive(r->left, new\_node);

}

else if (new\_node->value > r->value)

{

r->right = insertRecursive(r->right, new\_node);

}

else

{

cout << "No duplicate values allowed!\n";

return r;

}

return r;

}

void print2D(TreeNode<Type> \*r, int space) // Print tree

{

if (r == nullptr)

return;

space += SPACE;

print2D(r->right, space);

cout << endl;

for (int i = SPACE; i < space; i++)

cout << " ";

cout << r->value << "\n";

print2D(r->left, space);

}

void print2DFile(TreeNode<Type> \*r, int space, ofstream &fout) // print in FILE

{

if (r == nullptr)

return;

space += SPACE;

print2DFile(r->right, space, fout);

fout << endl;

for (int i = SPACE; i < space; i++)

fout << " ";

fout << r->value << "\n";

print2DFile(r->left, space, fout);

}

void printPreorder(TreeNode<Type> \*r) // Root, Left, Right

{

int count = 0;

if (r == nullptr)

return;

cout << r->value << " ";

printPreorder(r->left);

printPreorder(r->right);

}

void printInorder(TreeNode<Type> \*r) // Left, Root, Right

{

if (r == nullptr)

return;

printInorder(r->left);

cout << r->value << " ";

printInorder(r->right);

}

void printPostorder(TreeNode<Type> \*r) // Left, Right, Root

{

if (r == nullptr)

return;

printPostorder(r->left);

printPostorder(r->right);

cout << r->value << " ";

}

TreeNode<Type> \*iterativeSearch(Type val) // search

{

if (root == nullptr)

{

return root;

}

else

{

TreeNode<Type> \*temp = root;

while (temp != nullptr)

{

if (val == temp->value)

{

return temp;

}

else if (val < temp->value)

{

temp = temp->left;

}

else

{

temp = temp->right;

}

}

return nullptr;

}

}

TreeNode<Type> \*minValueNode(TreeNode<Type> \*node) // min node

{

TreeNode<Type> \*current = node;

while (current->left != nullptr)

{

current = current->left;

}

return current;

}

TreeNode<Type> \*deleteNode(TreeNode<Type> \*r, Type val) // delete node

{

if (r == nullptr)

{

return nullptr;

}

else if (val < r->value)

{

r->left = deleteNode(r->left, val);

}

else if (val > r->value)

{

r->right = deleteNode(r->right, val);

}

else

{

if (r->left == nullptr)

{

TreeNode<Type> \*temp = r->right;

delete r;

return temp;

}

else if (r->right == nullptr)

{

TreeNode<Type> \*temp = r->left;

delete r;

return temp;

}

else

{

TreeNode<Type> \*temp = minValueNode(r->right);

r->value = temp->value;

r->right = deleteNode(r->right, temp->value);

}

}

return r;

}

Type getLevelUtil(TreeNode<Type> \*r, Type value, int level)

{

if (r == nullptr)

return 1;

if (r->value == value)

return level;

int downlevel = getLevelUtil(r->left, value, level + 1);

if (downlevel != 0)

return downlevel;

downlevel = getLevelUtil(r->right, value, level + 1);

return downlevel;

}

Type getLevel(TreeNode<Type> \*r, Type value) // nodes levels

{

return getLevelUtil(r, value, 0);

}

void getReverseNode(TreeNode<Type> \*r, vector<int> &arr) // get Palindroms

{

if (r == nullptr)

return;

Type n = r->value;

Type tmp = n;

Type a = n % 10;

while (tmp /= 10)

{

a = a \* 10 + tmp % 10;

}

if (a == n)

{

arr.push\_back(n);

}

getReverseNode(r->left, arr);

getReverseNode(r->right, arr);

}

~BinaryTree() {}

};

**Main.cpp:**

#include <iostream>

#include <fstream>

#include <vector>

#include "BinaryTree.h"

using namespace std;

int main()

{

BinaryTree<int> obj;

int answer;

int val;

int size;

int \*arr;

string fileName;

ofstream fout;

ifstream fin;

vector<int> Palindroms;

do

{

cout << "1. Writing a tree from an array" << endl;

cout << "2. Insert Node" << endl;

cout << "3. Search Node" << endl;

cout << "4. Delete Node" << endl;

cout << "5. Print BinaryTree values and TRAVERSE" << endl;

cout << "6. Input Tree from FILE" << endl;

cout << "7. Output Tree in FILE" << endl;

cout << "8. Palindroms and NODE level" << endl;

cout << "0. Exit Program" << endl;

cin >> answer;

TreeNode<int> \*new\_node = new TreeNode<int>();

switch (answer)

{

case 0:

break;

case 1:

cout << "Enter size of the array: ";

cin >> size;

arr = new int[size];

for (int i = 0; i < size; i++)

{

cout << i << " element = ";

cin >> arr[i];

}

cout << "\n";

for (int i = 0; i < size; i++)

{

cout << arr[i] << " ";

}

cout << "\n\n";

for (int i = 0; i < size; i++)

{

obj.addFarray(arr[i]);

}

delete arr;

break;

case 2:

cout << "INSERT" << endl;

cout << "Enter VALUE of TREE NODE to INSERT in BST: ";

cin >> val;

new\_node->value = val;

obj.root = obj.insertRecursive(obj.root, new\_node);

cout << endl;

break;

case 3:

cout << "SEARCH" << endl;

cout << "Enter VALUE of TREE NODE to Search in BST: ";

cin >> val;

new\_node = obj.iterativeSearch(val);

if (new\_node != nullptr)

cout << "\nValue found\n\n";

else

cout << "\nValue NOT found!\n\n";

break;

case 4:

cout << "DELETE" << endl;

cout << "Enter VALUE of TREE NODE to Delete in BST: ";

cin >> val;

new\_node = obj.iterativeSearch(val);

if (new\_node != nullptr)

{

obj.deleteNode(obj.root, val);

cout << "\nValue DELETED\n\n";

}

else

{

cout << "\nValue NOT found!\n\n";

}

break;

case 5:

cout << "PRINT AND TRAVERSE" << endl;

obj.print2D(obj.root, 5);

cout << "Preorder Traversal:\n";

obj.printPreorder(obj.root);

cout << endl;

cout << "Inorder Traversal:\n";

obj.printInorder(obj.root);

cout << endl;

cout << "Postorder Traversal:\n";

obj.printPostorder(obj.root);

cout << "\n\n";

break;

case 6:

cout << "Enter file name: ";

cin >> fileName;

fin.open(fileName);

if (fin.is\_open())

{

fin >> size;

arr = new int[size];

cout << "\n";

for (int i = 0; i < size; i++)

{

fin >> arr[i];

cout << arr[i] << " ";

}

for (int i = 0; i < size; i++)

{

obj.addFarray(arr[i]);

}

cout << "\n\n";

delete arr;

}

else

{

cout << "\nERROR! FILE isn't open!\n\n";

}

fin.close();

break;

case 7:

cout << "Enter file name: ";

cin >> fileName;

fout.open(fileName);

obj.print2DFile(obj.root, 5, fout);

cout << "\nThe file is filled in successfully\n\n";

fout.close();

break;

case 8:

obj.getReverseNode(obj.root, Palindroms);

for (int i = 0; i < Palindroms.size(); i++)

{

cout << Palindroms[i] << " ";

}

cout << "\n\n";

for (int i = 0; i < Palindroms.size(); i++)

{

cout << Palindroms[i] << ": Level - " << obj.getLevel(obj.root, Palindroms[i]) << "\n";

}

cout << "\n\n";

break;

default:

cout << "Enter correct number " << endl;

}

} while (answer != 0);

return 0;

}

# ОТВЕТЫ НА КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Опишите общий принцип организации бинарного дерева поиска. В чем заключается преимущество подобной организации данных?

Двоичное дерево поиска — двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

* оба поддерева — левое и правое — являются двоичными деревьями поиска;
* у всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X;
* у всех узлов правого поддерева произвольного узла X значения ключей данных больше либо равны, нежели значение ключа данных самого узла X.

Очевидно, данные в каждом узле должны обладать ключами, на которых определена операция сравнения меньше.

Как правило, информация, представляющая каждый узел, является записью, а не единственным полем данных.

Основным преимуществом двоичного дерева поиска перед другими структурами данных является возможная высокая эффективность реализации основанных на нём алгоритмов поиска и сортировки.

Двоичное дерево поиска применяется для построения более абстрактных структур, таких, как множества, мультимножества, ассоциативные массивы.

1. Опишите алгоритм построения бинарного дерева поиска?

Дерево строится из входной последовательности чисел, и эти числа используются как ключ при построении. Следует заметить, что каждый узел дерева содержит четыре поля: info - число из входной последовательности, с - счетчик числа вхождений данного ключа в исходную последовательность чисел, rlink и llink - указатели на левое и правое поддерево соответственно.

Функция построения дерева tree рекурсивна. Положение нового узла в дереве определяется по следующему правилу. Значение элемента, добавляемого к дереву, сравнивается со значением корневого узла. Если ключ элемента меньше, чем ключ корневого узла, то нужно переместиться к левому потомку узла, в противном случае следует переместиться к правому потомку. Сравнения повторяются, причем каждый раз ключ элемента сопоставляется либо с ключом левого потомка, либо с ключом правого потомка.

1. Как выполняется поиск данных в бинарном дереве?

Алгоритм поиска в бинарном дереве поиска, как простом, так и сбалансированном, достаточно прост. Поиск осуществляется целенаправленным движением по ветвям дерева. Если ключ поиска равен ключу в вершине, значит, ключ найден и его адрес возвращается через параметр-указатель.

1. Опишите алгоритм удаления узла из бинарного дерева.

Для удаления узла из бинарного дерева поиска нужно рассмотреть три возможные ситуации. Если у узла нет дочерних узлов, то у его родителя нужно просто заменить указатель на null. Если у узла есть только один дочерний узел, то нужно создать новую связь между родителем удаляемого узла и его дочерним узлом. Наконец, если у узла два дочерних узла, то нужно найти следующий за ним элемент (у этого элемента не будет левого потомка), его правого потомка подвесить на место найденного элемента, а удаляемый узел заменить найденным узлом. Таким образом, свойство бинарного дерева поиска не будет нарушено. Данная реализация удаления не увеличивает высоту дерева.

1. Перечислите и охарактеризуйте различные способы обхода бинарного дерева.

Функция обхода двоичного дерева тоже рекурсивна, и в данном случае входная последовательность будет выведена в порядке неубывания.

Обычно рассматриваются три способа обхода дерева:

1. Прямой порядок прохождения (префиксная запись /+ a b - c d):
   * попасть в корень;
   * пройти левое поддерево;
   * пройти правое поддерево.
2. Обратный порядок прохождения (инфиксная запись (a+b)/(c-d)):
   * пройти левое поддерево;
   * попасть в корень;
   * пройти правое поддерево.
3. Концевой порядок прохождения (постфиксная запись a b + c d - / ):
   * пройти левое поддерево;
   * пройти правое поддерево;
   * попасть в корень.

# ВЫВОДЫ

Изучил метод рекурсивной обработки данных, познакомился с принципами организации и обработки бинарных деревьев.