**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**к курсовой работе**

**по дисциплине «Программирование»**

|  |  |  |
| --- | --- | --- |
| Студент гр. 2373 |  | Чекстер А. А. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

изучение свойств и организации деревьев как структуры данных; получение практических навыков в работе с бинарным деревом поиска; определение преимуществ и недостатков структуры данных вида дерева; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в различных структурах данных.

**Основные теоретические положения.**

Бинарное дерево – это динамическая структура данных, состоящая из узлов, каждый из которых содержит кроме данных не более двух ссылок на различные бинарные деревья. На каждый узел имеется ровно одна ссылка. Начальный узел называется корнем дерева.

Дерево является рекурсивной структурой данных, поскольку каждое поддерево также является деревом. Действия с такими структурами лучше всего описывать с помощью рекурсивных алгоритмов.

Бинарные дерева нужны:

1. Если необходимо расположить информацию, связанную между собой некой иерархией. Примером является файловая система компьютера.
2. Если необходимо хранить данные, составленные в виде определённой структуры. Тогда хранение в виде бинарного дерева позволяет уменьшить скорость поиска данных и доступа к хранимой информации.
3. Если необходима высокая скорость добавления или удаления данных.
4. Если заранее неизвестен хранимый объем данных. Бинарные деревья, также, как и связанные списки, не имеют ограничения на количество узлов, поскольку узлы связаны указателями.

Бинарное дерево поиска – бинарное дерево, которое обладает дополнительными свойствами. Если дерево организовано таким образом, что для каждого узла все ключи левого поддерева меньше ключа этого узла, а все ключи его правого поддерева больше, то его можно называть бинарным деревом поиска.

Граф – это множество вершин и ребер. Ребро – это связь между двумя вершинами. Количество возможных ребер в графе квадратично зависит от количества вершин (для понимания можно представить турнирную таблицу сыгранных матчей).

Дерево – это связный граф без циклов. Связность означает, что из любой вершины в любую другую существует путь по ребрам. Отсутствие циклов означает, что данный путь – единственный.

Обход графа – это систематическое посещение всех его вершин по одному разу каждой. Существует два вида обхода графа:

 1) поиск в глубину;

 2) поиск в ширину.

**Постановка задачи**

Необходимо реализовать программу, которая выполняет следующие действия.

1.   Формирование бинарного дерева из *N* элементов:

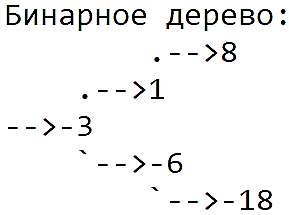
a) пользователь вводит количество элементов *N* бинарного дерева, которое автоматически заполняется случайными числами (–99 до 99);

б) пользователь вводит в консоль элементы массива, *N* определяется автоматически по количеству введенных элементов;

в) элементы считываются с файла, в котором хранится массив из чисел, *N* – количество элементов в файле.

Определение скорости формирования бинарного дерева.

2.   Вывод в консоль и файл tree бинарного дерева. Бинарное дерево должно иметь подобный вид:



3.   Определение скорости вставки, удаления и получения элемента дерева. В отчете сравните скорость работы бинарного дерева с линейной структурой (двусвязным списком или динамическим массивом) и сделайте выводы.

4. Прямой обход, обратный обход и обход в ширину бинарного дерева.

5.  Генерация заданий к практической работе по бинарным деревьям. Необходимо сгенерировать задания в файл output\_task в количестве вариантов, которые введет пользователь. В файл output\_key необходимо вывести короткие ответы к заданиям. В файл output\_ans необходимо вывести развернутые ответы к заданиям. Должны быть представлены задания на создание бинарного дерева поиска из массива значений, удаление и вставка элементов в бинарное дерево поиска.

**Выполнение работы.**

Код программы представлен в приложении А.

**Блок описания кода и использованных алгоритмов:**

1. Поскольку каждый корень дерева должен иметь три части, логичнее всего представить его в виде рекурсивной структуры
2. Для поиска элемента в бинарном дереве поиска можно воспользоваться функцией, которая принимает в качестве параметра заданное значение ключа. Для каждого узла функция сравнивает значение его ключа со значением заданного ключа. Если значения ключей одинаковы, то функция возвращает текущий узел. В противном случае, функция вызывается рекурсивно: для левого поддерева, если заданный ключ меньше ключа узла, для правого поддерева, если заданный ключ больше ключа узла.
3. При вставке элемента в бинарное дерево поиска необходимо учитывать два случая:

* Дерево пустое;
* Дерево непустое.

Если дерево пустое, то создается новый узел, который добавляется в дерево. Если дерево не пустое, то значение ключа добавляемого элемента сравнивается со значением ключа в узле, начиная от корня.

1. С удалением узла дело обстоит немного иначе. Существует несколько возможных ситуаций:

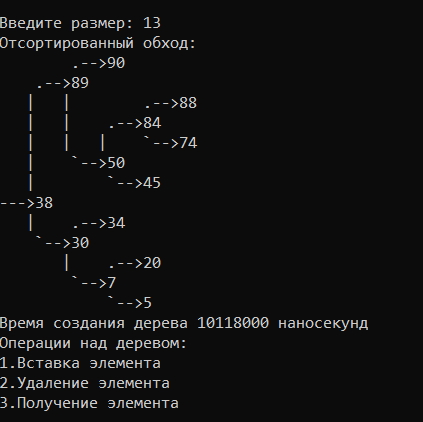
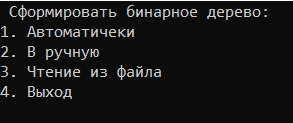
1)      у узла нет наследников (узел является листом);  
2)      у узла имеется наследник, притом только один;  
3)      у узла есть оба наследника.

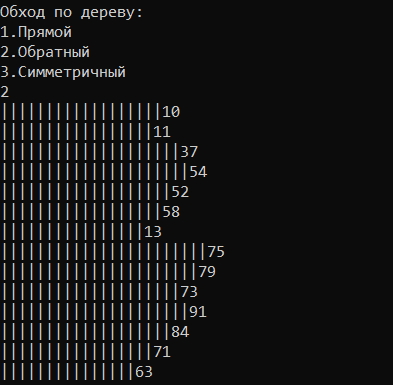
1. Поиск в ширину идет из начальной вершины, посещает сначала все вершины, находящиеся на расстоянии одного ребра от начальной, потом посещает все вершины на расстоянии двух ребер от начальной и так далее. Алгоритм поиска в ширину является по своей природе нерекурсивным (итеративным). Для его реализации применяется структура данных «очередь» (FIFO).

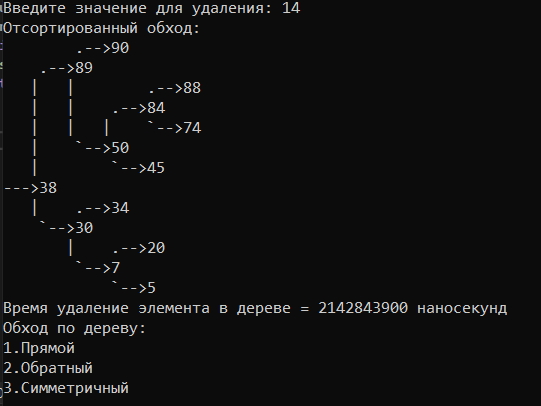
**Выводы.**

Мы научились изучению свойств и организации деревьев как структуры данных; получение практических навыков в работе с бинарным деревом поиска; определение преимуществ и недостатков структуры данных вида дерева; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в различных структурах данных.

**Блок скриншотов работы программы**







Приложение А

рабочий код

#include <iostream>

#include <string>

#include <fstream>

#include <ctime>

#include <chrono>

using namespace std;

using namespace chrono;

struct Node {

int key;

struct Node\* left, \* right;

};

struct Trunk {

Trunk\* prev;

string str;

Trunk(Trunk\* prev, string str) {

this->prev = prev;

this->str = str;

}

};

struct Node\* newNode(int item) {

struct Node\* temp = (struct Node\*)malloc(sizeof(struct Node));

temp->key = item;

temp->left = temp->right = NULL;

return temp;

}

void File\_Reading(Node\* Tree, string Name\_of\_File, int Count) {

ifstream Data;

Data.open(Name\_of\_File);

string str;

for (int i = 0; i < Count; i++) {

getline(Data, str);

Tree[i].key = stoi(str);

}

Data.close();

}

void File\_Reading\_Task(string Name\_of\_File, int N) {

ifstream Data;

Data.open(Name\_of\_File);

string str;

for (int i = 0; i < N; i++) {

getline(Data, str);

Name\_of\_File = str;

}

Data.close();

}

void T\_File(string Name\_of\_File, int N) {

ofstream Data;

Data.open(Name\_of\_File);

const int task = 3;

const char\* taskbank[task] = { "Вставьте в дерево элемент: ", "Удалите из дерева элемент: ", "Создайте бинарное дерево поиска из массива значений: " };

for (int i = 0; i < N; i++) {

int rnd\_task = rand() % task;

if (rnd\_task == 2) {

int numb[10];

Data << taskbank[rnd\_task];

for (int i = 0; i < 10; i++)

numb[i] = rand() % 100;

for (int i = 0; i < 10; i++)

Data << numb[i] << " ";

Data << "\n";

}

else if (rnd\_task == 0) {

int rnd\_task\_num = rand() % 100;

Data << taskbank[rnd\_task] << rnd\_task\_num << '\n';

}

else if (rnd\_task == 1) {

int rnd\_task\_num = rand() % 100;

Data << taskbank[rnd\_task] << rnd\_task\_num << '\n';

}

}

Data.close();

}

void Reflect\_Data(Node\* Tree, int Count) {

for (int i = 0; i < Count; i++) {

cout << Tree[i].key << '\n';

cout << "\n\n";

}

}

void inorder(struct Node\* root) {

if (root != NULL) {

inorder(root->left);

cout << root->key << " -> ";

inorder(root->right);

}

}

void showTrunks(Trunk\*& trunk) {

if (trunk == NULL)

return;

showTrunks(trunk->prev);

cout << trunk->str;

}

void showTree(Node\*& tree, Trunk\* prev, bool isRight) {

if (tree == NULL) return;

string prev\_str = " ";

Trunk\* trunk = new Trunk(prev, prev\_str);

showTree(tree->right, trunk, true);

if (!prev) {

trunk->str = "--->";

}

else if (isRight) {

trunk->str = ".-->";

prev\_str = " |";

}

else {

trunk->str = "`-->";

prev->str = prev\_str;

}

showTrunks(trunk);

cout << tree->key << endl;

if (prev) prev->str = prev\_str;

trunk->str = " |";

showTree(tree->left, trunk, false);

}

void print\_Tree(Node\* p, int level)

{

if (p != NULL) {

if (p)

{

print\_Tree(p->left, level + 1);

for (int i = 0; i < level; i++) cout << " ";

cout << p->key << endl;

print\_Tree(p->right, level + 1);

}

}

}

struct Node\* insert(struct Node\* Tree, int key) {

if (Tree == NULL) return newNode(key);

if (key < Tree->key)

Tree->left = insert(Tree->left, key);

else

Tree->right = insert(Tree->right, key);

return Tree;

}

struct Node\* minValueNode(struct Node\* node) {

struct Node\* current = node;

while (current && current->left != NULL)

current = current->left;

return current;

}

struct Node\* deleteNode(struct Node\* root, int key) {

if (root == NULL) return root;

if (key < root->key)

root->left = deleteNode(root->left, key);

else if (key > root->key)

root->right = deleteNode(root->right, key);

else {

if (root->left == NULL) {

struct Node\* temp = root->right;

free(root);

return temp;

}

else if (root->right == NULL) {

struct Node\* temp = root->left;

free(root);

return temp;

}

struct Node\* temp = minValueNode(root->right);

root->key = temp->key;

root->right = deleteNode(root->right, temp->key);

}

return root;

}

void search(Node\* Tree, int key) {

if (Tree) {

if (key == Tree->key) {

cout << "Элемент " << key << " есть в дереве.\n";

return;

}

else if (key > Tree->key)

search(Tree->right, key);

else if (key < Tree->key)

search(Tree->left, key);

}

else {

cout << "Данного элемента нет в дереве\n";

return;

}

}

void stright(Node\* Tree, int key)

{

if (Tree == NULL) return;

for (int i = 0; i < key; ++i) cout << "|";

cout << Tree->key << endl;

key--;

stright(Tree->left, ++key);

stright(Tree->right, ++key);

}

void reverse(Node\* Tree, int key) {

if (Tree == NULL) return;

else {

reverse(Tree->left, ++key);

reverse(Tree->right, ++key);

for (int i = 0; i < key; ++i) cout << "|";

cout << Tree->key << endl;

key--;

}

}

void symmetry(Node\* Tree, int key) {

if (Tree == NULL) return;

else {

symmetry(Tree->left, ++key);

for (int i = 0; i < key; ++i) cout << "|";

cout << Tree->key << endl;

key--;

}

symmetry(Tree->right, ++key);

}

int Dimensionality(string Name\_of\_File) {

ifstream Data;

Data.open(Name\_of\_File);

if (!Data.is\_open()) {

return 0;

}

int count = 0;

string str;

while (getline(Data, str)) {

count++;

}

Data.close();

return (count);

}

int main() {

setlocale(0, "");

struct Node\* root = NULL;

int choose;

Menu: {

cout << " Сформировать бинарное дерево: \n" <<

"1. Автоматичеки\n" <<

"2. В ручную\n" <<

"3. Чтение из файла\n" <<

"4. Выход\n";

}

cin >> choose;

switch (choose) {

case 1: {

system("cls");

srand(time(0));

int N;

int count = 0;

cout << "\nВведите размер: ";

cin >> N;

auto time1 = steady\_clock::now();

for (int i = 1; i <= N; i++) {

root = insert(root, rand() % 100);

}

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2 = steady\_clock::now();

cout << "Время создания дерева " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

int input;

cout << "Операции над деревом:\n" <<

"1.Вставка элемента\n" <<

"2.Удаление элемента\n" <<

"3.Получение элемента\n";

cin >> input;

if (input == 1) {

auto time1\_1 = steady\_clock::now();

int data;

cout << "Введите значение для вставки: ";

cin >> data;

root = insert(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_1 = steady\_clock::now();

cout << "Время вставки элемента в дерево = " << duration\_cast<nanoseconds>(time2\_1 - time1\_1).count() << " наносекунд\n";

}

else if (input == 2) {

auto time1\_2 = steady\_clock::now();

int data;

cout << "Введите значение для удаления: ";

cin >> data;

root = deleteNode(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_2 = steady\_clock::now();

cout << "Время удаление элемента в дереве = " << duration\_cast<nanoseconds>(time2\_2 - time1\_2).count() << " наносекунд\n";

}

else if (input == 3) {

auto time1\_3 = steady\_clock::now();

int data;

cout << "Введите значение для поиска: ";

cin >> data;

search(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_3 = steady\_clock::now();

cout << "Время поиска элемента в дереве = " << duration\_cast<nanoseconds>(time2\_3 - time1\_3).count() << " наносекунд\n";

}

else

cout << "Incorrect input. Try again" << endl;

int input\_o;

cout << "Обход по дереву:\n" <<

"1.Прямой\n" <<

"2.Обратный\n" <<

"3.Симметричный\n";

cin >> input\_o;

if (input\_o == 1) {

stright(root, N);

}

else if (input\_o == 2) {

reverse(root, N);

}

else if (input\_o == 3) {

symmetry(root, N);

}

else

cout << "Incorrect input. Try again" << endl;

system("pause");

goto Menu;

}

case 2: {

system("cls");

int N, count = 0;

int summ = 0;

bool flag = true;

cout << "\nВведите список: \n";

cout << "\nДля завершения ввода, введите '0': ";

auto time1 = steady\_clock::now();

while (flag == true) {

cin >> N;

if (N != 0) {

count++;

root = insert(root, N);

}

else

flag = false;

}

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, summ);

//inorder(root);

cout << endl;

auto time2 = steady\_clock::now();

cout << "Время создания дерева" << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

int input;

cout << "Операции над деревом:\n" <<

"1.Вставка элемента\n" <<

"2.Удаление элемента\n" <<

"3.Получение элемента\n";

cin >> input;

if (input == 1) {

auto time1\_1 = steady\_clock::now();

int data;

cout << "Введите значение для вставки: ";

cin >> data;

root = insert(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, summ);

//inorder(root);

auto time2\_1 = steady\_clock::now();

cout << "Время вставки элемента в дерево = " << duration\_cast<nanoseconds>(time2\_1 - time1\_1).count() << " наносекунд\n";

}

else if (input == 2) {

auto time1\_2 = steady\_clock::now();

int data;

cout << "Введите значение для удаления: ";

cin >> data;

root = deleteNode(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, summ);

//inorder(root);

auto time2\_2 = steady\_clock::now();

cout << "Время удаление элемента в дереве = " << duration\_cast<nanoseconds>(time2\_2 - time1\_2).count() << " наносекунд\n";

}

else if (input == 3) {

auto time1\_3 = steady\_clock::now();

int data;

cout << "Введите значение для поиска: ";

cin >> data;

search(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, summ);

//inorder(root);

auto time2\_3 = steady\_clock::now();

cout << "Время поиска элемента в дереве = " << duration\_cast<nanoseconds>(time2\_3 - time1\_3).count() << " наносекунд\n";

}

else

cout << "Incorrect input. Try again" << endl;

int input\_o;

cout << "Обход по дереву:\n" <<

"1.Прямой\n" <<

"2.Обратный\n" <<

"3.Симметричный\n";

cin >> input\_o;

if (input\_o == 1) {

stright(root, count);

}

else if (input\_o == 2) {

reverse(root, count);

}

else if (input\_o == 3) {

symmetry(root, count);

}

else

cout << "Incorrect input. Try again" << endl;

system("pause");

goto Menu;

}

case 3: {

system("cls");

int count = 0;

string Name\_of\_File = "C:\\Users\\User\\Desktop\\rurs\\Data.txt";

int Count = Dimensionality(Name\_of\_File);

Node\* Tree = new Node[Count];

auto time1 = steady\_clock::now();

ifstream Data;

Data.open(Name\_of\_File);

string str;

for (int i = 0; i < Count; i++) {

getline(Data, str);

root = insert(root, stoi(str));

}

Data.close();

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2 = steady\_clock::now();

cout << "Время создания дерева " << duration\_cast<nanoseconds>(time2 - time1).count() << " наносекунд\n";

int input;

cout << "Операции над деревом:\n" <<

"1.Вставка элемента\n" <<

"2.Удаление элемента\n" <<

"3.Получение элемента\n" <<

"4.Генерация задания\n";

cin >> input;

if (input == 1) {

auto time1\_1 = steady\_clock::now();

int data;

cout << "Введите значение для вставки: ";

cin >> data;

root = insert(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_1 = steady\_clock::now();

cout << "Время вставки элемента в дерево = " << duration\_cast<nanoseconds>(time2\_1 - time1\_1).count() << " наносекунд\n";

}

else if (input == 2) {

auto time1\_2 = steady\_clock::now();

int data;

cout << "Введите значение для удаления: ";

cin >> data;

root = deleteNode(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_2 = steady\_clock::now();

cout << "Время удаление элемента в дереве = " << duration\_cast<nanoseconds>(time2\_2 - time1\_2).count() << " наносекунд\n";

}

else if (input == 3) {

auto time1\_3 = steady\_clock::now();

int data;

cout << "Введите значение для поиска: ";

cin >> data;

search(root, data);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

auto time2\_3 = steady\_clock::now();

cout << "Время поиска элемента в дереве = " << duration\_cast<nanoseconds>(time2\_3 - time1\_3).count() << " наносекунд\n";

}

else if (input == 4) {

string Task\_File = "C:\\Users\\User\\Desktop\\rurs\\output\_task.txt";

string Key\_File = "C:\\Users\\User\\Desktop\\rurs\\output\_key.txt";

string Ans\_File = "C:\\Users\\User\\Desktop\\rurs\\output\_ans.txt";

int N;

cout << "Введите количество заданий по действиям над Бинарным деревом: ";

cin >> N;

T\_File(Task\_File, N);

int count = Dimensionality(Task\_File);

File\_Reading\_Task(Task\_File, count);

cout << "Отсортированный обход:\n";

showTree(root, 0, false);

//print\_Tree(root, count);

//inorder(root);

}

else

cout << "Incorrect input. Try again" << endl;

int input\_o;

cout << "Обход по дереву:\n" <<

"1.Прямой\n" <<

"2.Обратный\n" <<

"3.Симметричный\n";

cin >> input\_o;

if (input\_o == 1) {

stright(root, Count);

}

else if (input\_o == 2) {

reverse(root, Count);

}

else if (input\_o == 3) {

symmetry(root, Count);

}

else

cout << "Incorrect input. Try again" << endl;

system("pause");

goto Menu;

}

case 4: {

return 0;

}

default: {

cout << "\nВведите значение из меню\n\n";

cin.clear();

while (cin.get() != '\n');

system("pause");

goto Menu;

}

}

}