**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра информационных систем**

отчет

**по курсовой работе**

**по дисциплине «Программирование»**

Тема: АВЛ-Дерево

|  |  |  |
| --- | --- | --- |
| Студент гр. |  | Полуянов В. Н. |
| Преподаватель |  | Глущенко А. Г. |

Санкт-Петербург

2023

**Цель работы.**

Изучение свойств и организации сбалансированных деревьев; получение практических навыков в работе с АВЛ-деревьями; определение преимуществ и недостатков подобных структур данных; проведение сравнительной характеристики скорости вставки, удаления и поиска элемента в АВЛ-деревьях.

**Основные теоретические положения.**

Дерево – это связный граф без циклов. То есть граф, в котором из любой вершины можно попасть в любую другую, при этом такой путь будет единственным. Древовидная модель применяется, когда стоит цель быстрого поиска информации (например, базы данных).

Каждый элемент – это вершина или узел дерева. Соединенные узлы называются ветвями. Начальный узел (верхний) – это корень дерева (корневой узел). Листья – это узлы, в которые входит одна ветвь, причем не выходит ни одной.

Для выполнения операции с деревом необходимо просмотреть некоторые или все его узлы. Данную задачу называют обходом дерева. В процессе обхода все узлы посещаются в заранее определенном порядке. Способов обхода много, но выделяют три основных: прямой обход (префиксный, preorder), симметричный (инфиксный, inorder), обратный (постфиксный, postorder).

Двоичное дерево поиска, в котором высота каждого из поддеревьев, имеющих общий корень, отличается не более чем на некоторую константу, называется сбалансированным.

АВЛ-дерево – это сбалансированное двоичное дерево поиска с константой отличия, равной одному. Для узлов определяется коэффициент сбалансированности – разность высот правого и левого поддеревьев, принимающая одно значения из множества {-1, 0, 1}. АВЛ-дерево является эффективным в обработке: так как поддеревья почти или полностью симметричны, высота дерева сводится к оптимальному минимуму, что ускоряет поиск.

Как было сказано выше, АВЛ-дерево имеют коэффициент сбалансированности. В процессе обработки дерева балансировка может нарушаться, тогда требуется осуществить балансировку. Для этого после операций вставки или удаления выполняют поворот (малый правый, большой правый, малый левый или большой левый).

Операция вставки нового узла выполняется рекурсивно. По ключу данного узла определяется место вставка: выполняя обход, алгоритм сравнивает ключ добавляемого узла со встречающимися ключами, после чего вставляет новый элемент. По возвращению из рекурсии выполняется балансировка. Операция удаления также выполняется рекурсивно, но является более сложной, чем в случае работы с несбалансированным бинарным деревом. Сначала выполняется поиск узла, в процессе которого запоминаются пройденные узлы для выполнения балансировки). Удаляется искомый узел. Выполняется балансировка и производится перерасчет коэффициентов сбалансированности.

**Постановка задачи.**

Необходимо написать программу, которая:

1. Реализует формирование АВЛ-дерева из N элементов двумя способами
   1. Заполнением случайными числами после определения размера N;
   2. Ручным заполнением с автоматическим определением размера N;
2. Реализует вывод в консоль или файл (по выбору пользователя) дерева;
3. Реализует вставку, удаление и получение элемента дерева;
4. Определяет время вставки, удаления и получения элемента дерева.

**Выполнение работы.**

Код программы представлен в приложении А.

1. При запуске программы пользователю выводится основное меню, предлагающее выбрать вид работы (заполнение случайными числами, ручное заполнение, работа с файлом). (рис. 1).

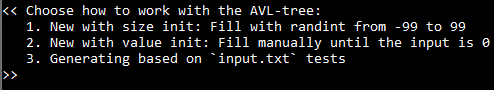


Рисунок 1. Главное меню

1. Следующий шаг зависит от введенной команды, если пользователь ввёл:
   1. “1”, то ожидается ввод пользователем размера дерева, после чего формируется АВЛ-дерево, заполненное случайными значениями в диапазоне от -99 до 99 (рис. 2).

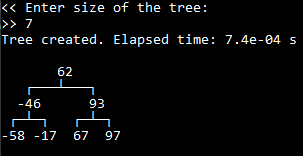


Рисунок 2. Заполнение случайными числами

* 1. “2”, то ожидается ввод пользователем значений узлов, после чего формируется АВЛ-дерево (рис. 3).

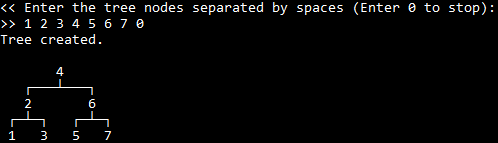


Рисунок 3. Ручное заполнение

* 1. “3”, то инструкции считываются с файла *input.txt* и результаты вычислений записываются в файлы *output\_ans.txt* и *output\_key.txt* с пошаговымрешением и с ответами соответственно (рис. 4).

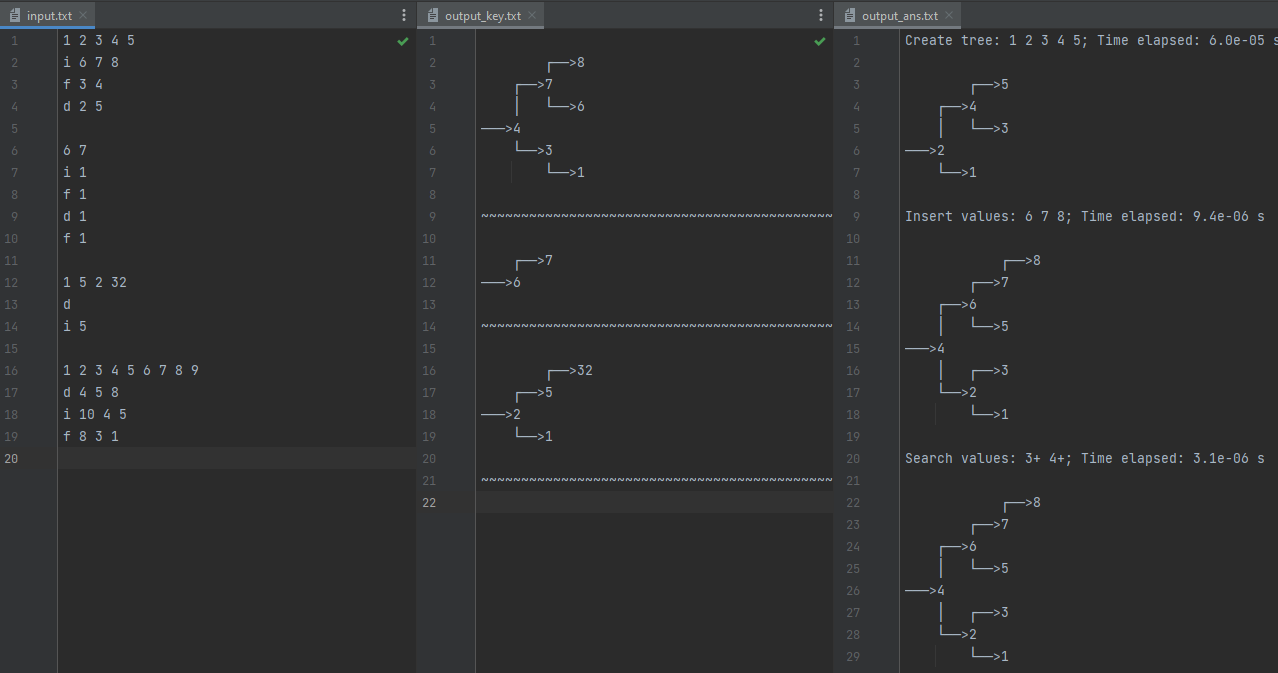


Рисунок 4. Работа с файлом

1. Если пользователь ввёл число, отличное от “3”, то ожидается ввод пользователем команды по обработке дерева (рис. 5). Если пользователь ввёл:
   1. “i”, то ожидается ввод значения нового узла, после чего производится вставка в дерево с расчётом затраченного времени (рис. 5).

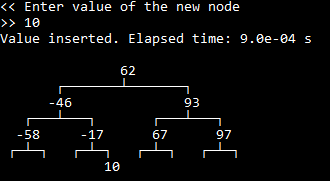


Рисунок 5. Вставка узла

* 1. “f”, то ожидается ввод искомого значения, после чего производится поиск в дереве с расчётом затраченного времени (рис. 6).

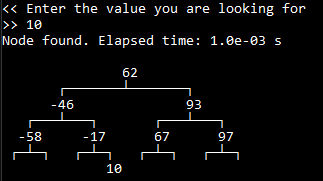


Рисунок 6. Поиск узла

* 1. “d”, то ожидается ввод удаляемого значения, после чего производится удаление узла из дерева с расчётом затраченного времени (рис. 7).

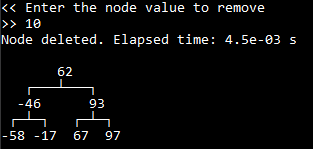


Рисунок 7. Удаление узла

* 1. “p”, то дерево выводится в консоль (рис. 8).

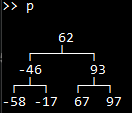
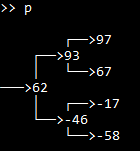


Рисунок 8. Вывод дерева (вертикально слева и горизонтально справа)

* 1. “s”, то меняется режим вывода дерева (вертикальный или горизонтальный) (рис. 9).



Рисунок 9. Смена режима вывода дерева

* 1. “0”, то выполнение программы завершается.

**Выводы.**

В ходе работы были изучены свойства и организация сбалансированных деревьев, получены практические навыки в работе с АВЛ-деревьями. Определены преимущества и недостатки подобных структур данных. Проведены подсчёты времени, затрачиваемой на вставку, удаление и поиск элементов дерева, содержащем 10 элементов (табл. 1).

|  |  |
| --- | --- |
| Операция | Время, миллисекунды |
| Вставка | 0,73 |
| Поиск | 1,30 |
| Удаление | 4,10 |

Таблица . Скорость операций

Приложение А

рабочий код

#include <iostream>  
#include <iomanip>  
#include <vector>  
#include <chrono>  
#include <string>  
#include "Tree.h"  
  
*/\*\*  
 \* Reads the user action input via cin  
 \* @param[out] userAction Reference to a declared variable  
 \* @return True if input was correct, else false  
 \*/*bool inputAction(char &userAction) {  
 std::cout << "<< Action:\n"  
 " i. Insert new node\n"  
 " f. Find the node by it`s value\n"  
 " d. Delete the node by it`s value\n"  
 " p. Print the tree\n"  
 " s. Switch the print method\n"  
 << std::setw(28) << std::setfill('-') << '\n'  
 << std::setfill(' ') <<  
 " 0. Exit\n"  
 ">> ";  
 std::cin >> userAction;  
  
 *// Error handler (i.e. more than one symbol input)* if (std::cin.fail() || std::cin.get() != '\n') {  
 std::cout << "RuntimeError. Unknown command\n";  
 std::cin.clear();  
 std::cin.ignore(100000, '\n');  
 return false;  
 }  
 return true;  
}  
  
*/\*\*  
 \* Reads the integer input via cin  
 \* @param[out] variable Reference to a declared variable  
 \* @param[in] isSpaceSep If true, it does not check the last character of the input for newline  
 \* @param[in] isUnsigned If true, then negative numbers will lead to an error  
 \* @return True if input was correct, else false  
 \*/*bool inputInt(int &variable, bool isSpaceSep = false, bool isUnsigned = false) {  
 std::cin >> variable;  
 if (std::cin.fail() || (isUnsigned && variable < 0) || (isSpaceSep && std::cin.peek() != '\n')) {  
 std::cout << "Invalid input\n";  
 std::cin.clear();  
 std::cin.ignore(10000, '\n');  
 return false;  
 }  
 return true;  
}  
  
*/\*\*  
 \* Gets the start time and calculates the time elapsed  
 \* @param[in] start Start time\_point  
 \* @param[in] outStream Output stream  
 \* @param[in] isEndOfLine If true, puts a newline character at the end of line  
 \*/*void printTimeDurationCast(auto start, std::ostream &outStream = std::cout, bool isEndOfLine = true) {  
 auto end = std::chrono::steady\_clock::now();  
 auto elapsed = std::chrono::duration\_cast<std::chrono::nanoseconds>(end - start);  
 outStream << std::scientific << std::setprecision(1);  
 outStream << elapsed.count() / 1e9 << " s";  
 if (isEndOfLine) outStream << std::endl;  
 outStream << std::defaultfloat;  
}  
  
*/\*\*  
 \* Fill the tree of a given size with random numbers from -99 to 99  
 \* @param[out] tree Reference to the tree object  
 \*/*void fillTreeRandom(Tree &tree) {  
 std::cout << "<< Enter size of the tree:\n>> ";  
 int size;  
 while (!inputInt(size, true, true) || size == 0) std::cout << ">> ";  
 auto start = std::chrono::steady\_clock::now();  
 for (int i = 0; i < size; ++i) tree.insert(-99 + std::rand() % 198);  
 std::cout << "Tree created. Elapsed time: ";  
 printTimeDurationCast(start);  
}  
  
*/\*\*  
 \* Fill the tree with numbers entered by the user  
 \* @param[out] tree Reference to the tree object  
 \*/*void fillTreeManual(Tree &tree) {  
 std::cout << "<< Enter the tree nodes separated by spaces (Enter 0 to stop):\n";  
 int value;  
 std::vector<int> values;  
 while (values.empty()) {  
 std::cout << ">> ";  
 while (true) {  
 if (!inputInt(value)) {  
 values.clear();  
 break;  
 }  
 if (value == 0) {  
 if (values.empty()) std::cout << "Can't create empty tree\n";  
 break;  
 }  
 values.push\_back(value);  
 }  
 }  
 for (auto el : values) tree.insert(el);  
 std::cout << "Tree created.\n";  
}  
  
*/\*\*  
 \* Take instructions from the INPUT\_PATH file, execute them, and write the results to OUTPUT\_KEY\_PATH and OUTPUT\_ANS\_PATH  
 \* @warning Be sure to specify environment variables before using  
 \*/*void generateAnswers() {  
 *// Environment variables* const char\* INPUT\_PATH = std::getenv("INPUT\_PATH");  
 const char\* OUTPUT\_KEY\_PATH = std::getenv("OUTPUT\_KEY\_PATH");  
 const char\* OUTPUT\_ANS\_PATH = std::getenv("OUTPUT\_ANS\_PATH");  
  
 std::ifstream in(INPUT\_PATH); *// Open input file* std::ofstream out\_key(OUTPUT\_KEY\_PATH); *// Open output key file* std::ofstream out\_ans(OUTPUT\_ANS\_PATH); *// Open output ans file  
  
 // File not open handler* if (!in.is\_open()) {  
 std::string errorMessage = "FileNotFoundError. No such file: `" + std::string(INPUT\_PATH) + "`";  
 throw errorMessage;  
 }  
 if (!out\_key.is\_open()) {  
 std::string errorMessage = "FileNotFoundError. No such file: `" + std::string(OUTPUT\_KEY\_PATH) + "`";  
 throw errorMessage;  
 }  
 if (!out\_ans.is\_open()) {  
 std::string errorMessage = "FileNotFoundError. No such file: `" + std::string(OUTPUT\_ANS\_PATH) + "`";  
 throw errorMessage;  
 }  
  
  
 *// Work* std::string line; *// Current line in file* unsigned int lineNumber = 0; *// Current line index* Tree tree; *// Tree object* while (std::getline(in, line)) {  
 lineNumber += 1;  
  
 *// Line is empty: end of tree commands. Continue* if (line.empty()) {  
 tree.print(true, out\_key);  
 out\_ans << '\n' << std::setw(64) << std::setfill('~') << '\n' << std::setfill(' ') << '\n';  
 out\_key << std::setw(64) << std::setfill('~') << '\n' << std::setfill(' ');  
 tree.clear();  
 continue;  
 }  
  
 *// Split string* std::vector<std::string> tokens;  
 auto \*number = new std::string;  
 auto \*symbolCount = new unsigned int(0);  
 for (char c: line) {  
 \*symbolCount += 1;  
 std::string s(1, c);  
 if (c == ' ') {  
 if (!number->empty()) {  
 tokens.push\_back(\*number);  
 number->clear();  
 }  
 } else if (\*symbolCount == 1 && (c == 'i' || c == 'f' || c == 'd')) {  
 if (!number->empty()) {  
 tokens.push\_back(\*number);  
 number->clear();  
 }  
 tokens.push\_back(s);  
 } else if (isdigit(c)) {  
 \*number += c;  
 } else {  
 in.close();  
 out\_key.close();  
 out\_ans.close();  
 std::string errorMessage = "Invalid syntax. Unknown symbol `" + s + "`. [Line " +  
 std::to\_string(lineNumber) + ", symbol " + std::to\_string(\*symbolCount) +  
 "]";  
 number->erase();  
 delete symbolCount;  
 throw errorMessage;  
 }  
 }  
 if (!number->empty()) tokens.push\_back(\*number);  
 number->erase();  
 delete symbolCount;  
  
 *// Create the tree* if (!tree.root) {  
 *// Another command or syntax error handler* if (line[0] != '-' && !isdigit(line[0])) {  
 in.close();  
 out\_key.close();  
 out\_ans.close();  
 std::string errorMessage = "RuntimeError. An array of tree values was expected, but something "  
 "else came in. [Line " + std::to\_string(lineNumber) + ", symbol 1]";  
 throw errorMessage;  
 }  
 auto start = std::chrono::steady\_clock::now();  
 for (auto &token: tokens) tree.insert(std::stoi(token));  
 out\_ans << "Create tree: " << line << "; Time elapsed: ";  
 printTimeDurationCast(start, out\_ans);  
 tree.print(true, out\_ans);  
 continue;  
 }  
  
 *// Action* auto start = std::chrono::steady\_clock::now();  
 switch (line[0]) {  
 *// Insert new node* case 'i':  
 out\_ans << "Insert values:";  
 for (int i = 1; i < tokens.size(); ++i) {  
 tree.insert(std::stoi(tokens[i]));  
 out\_ans << " " << tokens[i];  
 }  
 break;  
 *// Search the node* case 'f':  
 out\_ans << "Search values:";  
 for (int i = 1; i < tokens.size(); ++i) {  
 Node \*node = tree.search(std::stoi(tokens[i]));  
 out\_ans << " " << tokens[i] << (node ? "+" : "-");  
 }  
 break;  
 *// Delete the node* case 'd':  
 out\_ans << "Delete values:";  
 for (int i = 1; i < tokens.size(); ++i) {  
 tree.deleteNode(std::stoi(tokens[i]));  
 out\_ans << " " << tokens[i];  
 }  
 break;  
 *// Unknown symbol* default:  
 in.close();  
 out\_key.close();  
 out\_ans.close();  
 throw "RuntimeError.";  
 }  
 out\_ans << "; Time elapsed: ";  
 printTimeDurationCast(start, out\_ans);  
 tree.print(true, out\_ans);  
 }  
  
 in.close();  
 out\_key.close();  
 out\_ans.close();  
}  
  
int main() {  
  
 bool isPrintVertical = false;  
  
 *// Create tree or generate answers* int input;  
 while (true) {  
 std::cout << "<< Choose how to work with the AVL-tree:\n"  
 " 1. New with size init: Fill with randint from -99 to 99\n"  
 " 2. New with value init: Fill manually until the input is 0\n"  
 " 3. Generating based on `input.txt` tests\n"  
 ">> ";  
 if (!inputInt(input, true)) continue;  
 if (input != 3 && input != 2 && input != 1) {  
 std::cout << "Your input must be just 1, 2 or 3. Isn't it simple?\n";  
 continue;  
 }  
 break;  
 }  
 if (input == 3) { *// Answer generation, working with .txt* try {  
 generateAnswers();  
 }  
 catch (std::string error) {  
 std::cerr << error << std::endl;  
 return -1;  
 }  
 std::cout << "Answers generated. Check the files";  
 return 0;  
 }  
 Tree tree;  
 if (input == 1) fillTreeRandom(tree); *// Random fill, working in terminal* else fillTreeManual(tree); *// Manual fill, working in terminal* tree.print(isPrintVertical);  
  
 while (true) {  
 char userAction; *// Get command from user* if (!inputAction(userAction)) continue; *// Error occurred* if (userAction == '0') break; *// Exit command* switch (userAction) {  
 *// Insert new node* case 'i': {  
 std::cout << "<< Enter value of the new node\n>> ";  
 int key;  
 if (!inputInt(key, true)) continue;  
  
 auto start = std::chrono::steady\_clock::now();  
 tree.insert(key);  
 std::cout << "Value inserted. Elapsed time: ";  
 printTimeDurationCast(start);  
 tree.print(isPrintVertical);  
 break;  
 }  
  
 *// Get the node by it`s value* case 'f': {  
 std::cout << "<< Enter the value you are looking for\n>> ";  
 int key;  
 if (!inputInt(key, true)) continue;  
  
 auto start = std::chrono::steady\_clock::now();  
 Node \*node = tree.search(key);  
 if (node) std::cout << "Node found. Elapsed time: ";  
 else std::cout << "Node not found. Elapsed time: ";  
 printTimeDurationCast(start);  
 tree.print(isPrintVertical);  
 break;  
 }  
  
 *// Delete the node by it`s value* case 'd': {  
 std::cout << "<< Enter the node value to remove\n>> ";  
 int key;  
 if (!inputInt(key, true)) continue;  
  
 auto start = std::chrono::steady\_clock::now();  
 tree.deleteNode(key);  
 std::cout << "Node deleted. Elapsed time: ";  
 printTimeDurationCast(start);  
 tree.print(isPrintVertical);  
 break;  
 }  
  
 *// Print the tree* case 'p': {  
 tree.print(isPrintVertical);  
 break;  
 }  
  
 *// Switch the tree print method* case 's': {  
 isPrintVertical = not isPrintVertical;  
 std::cout << "The tree print method has been switched to ";  
 if (isPrintVertical) std::cout << "vertical\n";  
 else std::cout << "horizontal\n";  
 break;  
 }  
  
 *// Runtime error. Unknown command* default: std::cout << "RuntimeError. Unknown command\n";  
 }  
 system("pause");  
 }  
 return 0;  
}