Министерство науки и высшего образования Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования

ПЕРМСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ

(ФГАОУ ВО ПНИПУ)

ЭЛЕКТРОТЕХНИЧЕСКИЙ ФАКУЛЬТЕТ

КАФЕДРА ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ И АВТОМАТИЗИРОВАННЫЕ СИСТЕМЫ

УДК 004.65

ОТЧЕТ

О ЛАБОРАТОРНОЙ РАБОТЕ

По дисциплине «Учебно-исследовательская работа» на тему:

МЕТОД ВЕРТИКАЛЬНОГО ВЫВОДА БИНАРНОГО ПОИСКОВОГО ДЕРЕВА

Выполнил:

Студент группы РИС-23-2б

Борисов Н.А.

Проверил:

Доцент кафедры ИТАС

Петренко А. А.

Пермь 2024

ВВЕДЕНИЕ

Известный метод вертикального вывода бинарного поискового дерева является сложно выполнимой задачей, состоящей из большого количества шагов, которые должны наиболее корректно представить вертикальное отображение дерева. Метод, который используется повсеместно, является сложно читаемым, поэтому было решено создать универсальный и легко читаемый метод вертикального вывода для бинарного поискового дерева.

Цель лабораторной работы заключается в изучении структуры бинарного поискового дерева, выявлении закономерностей значений в древе и создании на их основе своего универсального метода для вывода значений узлов деревьев в консоль в вертикальном формате

Для достижения этой цели перед студентами ставятся следующие задачи:

1. Проанализировать структуру бинарного поискового дерева

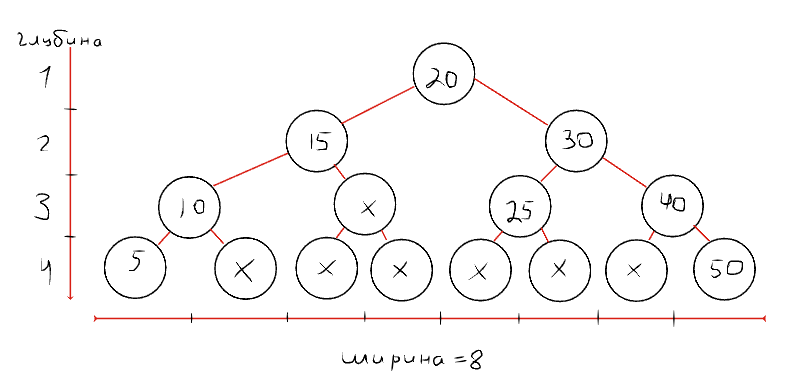
2. Написать программный код, реализующий вертикальный вывод бинарного поискового дерева в консоли

**1 Анализ структуры бинарного поискового дерева в контексте его последующей реализации**

В качестве результата работы метода вертикальной печати будет выступать равномерный построчный вывод значений узлов на соответствующих им местам бинарного поискового древа в консоли.

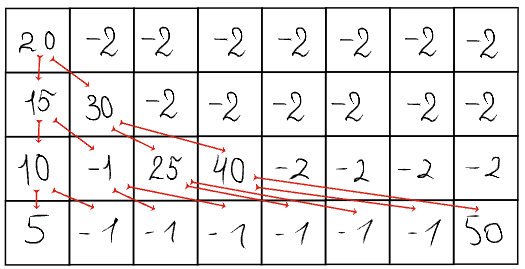
Для большей наглядности, было решено использовать матрицу, которая копирует значения идентификаторов из узлов дерева и хранит их в ячейках, которые соответствуют уровням глубины дерева. Количество строк в матрице равно высоте дерева. Ширина матрицы равна максимально возможному количеству элементов на последнем уровне матрицы.

Рассматриваемое дерево представлено на рисунке 1.1.



*Рисунок 1.1. – Концепция дерева поиска, копируемого в матрицу*

Рассматриваемая матрица представлена на рисунке 1.2.



*Рисунок 1.2. – Соответствующая матрица*

Матрица создается по следующему принципу:

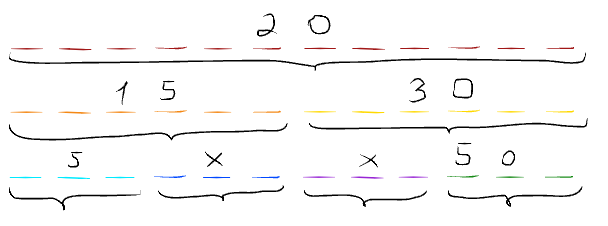
1. Создается пустая матрица и заполняется “-2”, которые означают, что ячейка пустая и не участвует в построении древа
2. Вызывается рекурсивный метод заполнения матрицы, который основан на прямом обходе узлов дерева. В качестве основных параметров рекурсии будет передаваться указатель на матрицу, указатель на текущий узел дерева, значение текущей глубины дерева.
3. Ячейки матрицы в строке заполняются по принципу: если в ячейке -2, то она пустая, заполняем ее значением идентификатора узла. Если значение узла равно nullptr, то заполняем пустую ячейку -1. Если значение ячейки больше или равно 0, то переходим к следующей ячейке в строке. После каждого действия, вызываем рекурсию. Выход из нее осуществляется, если глубина, на которой находится функция превышает высоту дерева.

Следующим шагом является вывод матрицы с помощью двух циклов.

Первый цикл проходится по строкам матрицы, второй вложенный цикл проходится по первым элементам строки матрицы, которые попадают под условие element\_index < pow(2, current\_level - 1). Если рассматриваемый элемент матрицы < 0, то вызывается функция, которая возвращает строку с нужным количеством пробелов, иначе вызывается функция, которая помещает в центр ячейки, размер которой высчитан с помощью специальной функции, идентификатор узла.

Размер ячейки на последнем уровне матрицы вычисляется как длина наибольшего числа + 1. Наибольшее число получается засчет прохода по правым веткам дерева. Длина ячеек матрицы, которые находятся выше последнего уровня дерева вычисляются по формуле int cur\_width\_cell = min\_length\_cell \* pow(2, d - cd) , где d – глубина дерева, cd – текущий уровень, на котором находится ячейка.

Визуализация вывода матрицы представлена на рисунке 1.3.



*Рисунок 1.3. – Пример визуализации вывода матрицы*

**Вывод**

Данный метод вертикального вывода бинарного дерева позволяет корректно и наглядно визуализировать дерево через консоль в вертикальном формате.

**2 Метод вертикального вывода бинарного дерева в консоль**

Для корректной работы метода VerticalPrint были написаны и использованы следующие функции

1. int OurTree::getDepth(OurTree\* current\_node, int number\_of\_current\_level, int maximal\_level)  
   (Находит высоту дерева, а именно количество уровней, на которых расположены узлы)
2. string OurTree::placeNumberInCell(unsigned long long number, int lenght\_of\_cell)  
   (Помещает символы числа в центр ячейки)
3. string OurTree::convertNumberToString(int number)  
   (Конвертирует число в строку)
4. void OurTree::addLeaf(OurTree\* current\_node, int value)  
   (Создаёт лист для дерева, по алгоритму, согласно которому получается дерево поиска)
5. void OurTree::demolishCollapse(OurTree\* current\_node)  
   (Удаление части дерева от указанного узла до его листов)

Использование данных методов позволило упростить чтение кода и упростить работу с данными.

**Вывод**

Разработанная программа успешно решает задачу вертикального вывода бинарного поискового дерева в консоли

ЗАКЛЮЧЕНИЕ

В ходе учебно-исследовательской была подробно рассмотрена структура и устройство дерева. На основе изученной информации был создан оригинальный метод вертикального вывода бинарного поискового дерева.

Предложенная программа ставит перед собой конкретные цели улучшения учебного процесса студентов первого курса Пермского Национального Политехнического Университета (ПНИПУ). С ее помощью студенты могут более эффективно усвоить материал по теме “бинарные поисковые деревья”. Данная работа демонстрирует совершенно новый способ вывода бинарного дерева в вертикальном формате, с понятным описанием и структурой

**ПРИЛОЖЕНИЕ А**

**Исходный код программы**

#include "..\Structures\OurTree.h"

#include <math.h>

#include "..\Structures\OurTree.h"

void OurTree::verticalPrint(OurTree\* root\_of\_tree) {

// Создание матрицы и внесение в неё значений узлов дерева

const int height = getDepth(root\_of\_tree, 1, 1); // Глубина дерева

const int width = exponentiationOfNumberTwo(height - 1); // Количество ячеек на последней строке вывода

int\*\* matrix = new int\* [height];

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

int\* level = new int[width];

for (int j = 0; j < width; ++j)

level[j] = -2; // Означает свободное место

matrix[i] = level;

}

cout << "Вывод значений узлов дерева поиска:\n";

directWay(root\_of\_tree);

cout << "\nИсходное состояние матрицы:\n";

printMatrix(matrix, height, width);

fillMatrix(matrix, root\_of\_tree, height, 1);

cout << "Матрица со значениями узлов дерева:\n";

printMatrix(matrix, height, width);

cout << endl;

int length\_of\_cell = getLenghtOfNumber(findNodeWithMaximalKey(root\_of\_tree)->key) + 1; // Найти длину для ячейки последнего уровня

string temp;

int depth = height;

int min\_length\_cell = length\_of\_cell;

for (int current\_level = 1; current\_level <= depth; ++current\_level)

{

for (int element\_index = 0; element\_index < pow(2, current\_level - 1); ++element\_index)

{

if (matrix[current\_level - 1][element\_index] < 0)

{

temp = createEmptyCell(min\_length\_cell, current\_level, depth);

}

else

{

int number = matrix[current\_level - 1][element\_index];

temp = createKeepCell(min\_length\_cell, number, current\_level, depth);

}

cout << temp;

}

cout << endl;

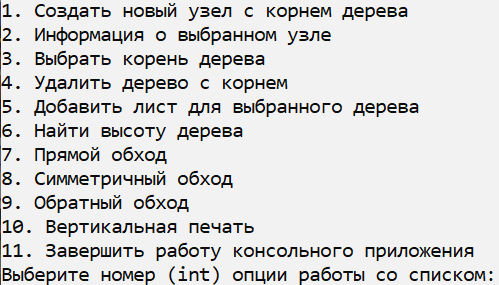
}

destroyMatrix(matrix, height);

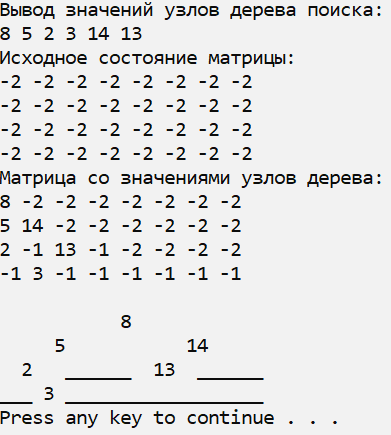
}

**ПРИЛОЖЕНИЕ Б**

**Скриншоты результатов программы**



*Рисунок 2.1. – Главное меню программы*

****

*Рисунок 2.2. – Результат работы программы для введенного дерева*

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. C/C++. Программирование на языке высокого уровня / Т. А. Павловская. - СПб.: Питер, 2013. - 461 с.: ил.