**РЕФЕРАТ**

Ключевые слова: ПОИСК, МЕТОДЫ ОБХОДА ДЕРЕВЬЕВ, ДЕРЕВЬЯ.

Цель работы: исследование методов обхода по дереву, написание программы для обучения методу обхода с использованием языка C#.

Объект исследования: методы поиска.

Предмет исследования: методы обхода деревьев.

Результат работы: изучение деревьев как структуры данных и методов обхода.

ОГЛАВЛЕНИЕ

[1 ВВЕДЕНИЕ 8](#_Toc533272494)

[2 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ 10](#_Toc533272495)

[2.1 Деревья 10](#_Toc533272496)

[2.2 Бинарные деревья 12](#_Toc533272497)

[2.3 Методы обхода деревьев 14](#_Toc533272498)

[3 РЕАЛИЗАЦИЯ ПРОГРАММЫ 18](#_Toc533272499)

[3.1 Описание разработанной программы 18](#_Toc533272500)

[3.2 Руководство пользователя 26](#_Toc533272501)

[3.2.1 Запуск программы 26](#_Toc533272502)

[3.2.2 Работа в программе 26](#_Toc533272503)

[4 ЗАКЛЮЧЕНИЕ 29](#_Toc533272504)

[5 СПИСОК ЛИТЕРАТУРЫ 30](#_Toc533272505)

[6 ПРИЛОЖЕНИЕ 31](#_Toc533272506)

# ВВЕДЕНИЕ

В наше время, время информационного общества, очень много профессий связаны с работой с большими массивами данных. Да и не только профессий. Каждый человек изо дня в день сталкивается с потребностью в какой-либо конкретной информации.

Поиск данных, являясь одним из приоритетных направлений работы с данными, предполагает использование соответствующих алгоритмов в зависимости от ряда факторов: способ представления данных, упорядоченность множества поиска, объем данных, расположение их во внешней или во внутренней памяти. И тут конечно же появляется вопрос: как его осуществлять?

Исходя из вышеизложенного, данная курсовая работа посвящена актуальной теме: «Исследование задач поиска по дереву».

Целью работы является изучение методов обхода деревьев и создание программы, которая будет объяснять принцип работы этих методов, реализованных на бинарном дереве.

Объектом исследования являются задачи поиска по дереву. Предметом исследования – методы обходов деревьев .

Для достижения цели в работе поставлены следующие задачи:

* изучение деревьев;
* изучение классов в C#;
* разработка функций вставки элементов в дерево и обхода по нему;
* сборка программы.

Содержание курсовой работы отражает реализацию поставленных задач.

В первой главе представлена краткая теория о деревьях, их представлениях и использовании.

Во второй главе описывается программа и принцип ее работы.

В заключении подведены итоги работы, сделаны выводы по решению поставленных задач и сформулированы предложения и рекомендации по дальнейшему применению полученных результатов.

В приложении представлены разработанные программные коды.

Для решения поставленных задач исследования в работе использовались методы объектно-ориентированного программирования, обхода деревьев.

Практическая значимость работы заключается в разработке программы, которая будет отображать принцип работы алгоритмов обхода деревьев.

Курсовая работа состоит из введения, двух разделов, заключения, списка использованных источников и приложения. Ее общий объем составляет 36 страниц машинописного текста. Работа содержит 15 рисунков, список источников из 7 наименований.

# ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ

## Деревья

Дерево – это нелинейная связанная структура данных или же, говоря другим языком, граф, не имеющий цикличности.

Для такой структуры данных характерны следующие признаки:

* самый первый элемент дерева (корень) не имеет ссылок;
* для того чтобы обратится к элементу нужно пройти от корня до искомого переходя от одного к другому по ссылкам;
* каждый элемент имеет связь только со следующим и предыдущим элементом.

На рисунке 1 представлен простой его пример.

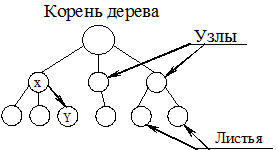


Рисунок 1 – Дерево

Дерево – это одна из самых распространенных структур данных, которая своей структурой напоминает дерево, и является набором связанных узлов.

Узел является экземпляром одного из двух типов элементов графа, соответствующим объекту некоторой фиксированной природы. Узел может содержать значение, состояние или представление отдельной информационной структуры или самого дерева. Каждый узел дерева имеет ноль или более узлов-потомков, которые располагаются ниже по дереву (по соглашению, деревья 'растут' вниз, а не вверх, как это происходит с настоящими деревьями). Узел, имеющий потомка, называется узлом-родителем относительно своего потомка (или узлом-предшественником, или старшим). Каждый узел имеет не больше одного предка. Высота узла – это максимальная длина нисходящего пути от этого узла к самому нижнему узлу (краевому узлу), называемому листом. Высота корневого узла равна высоте всего дерева. Глубина вложенности узла равна длине пути до корневого узла.

Узел, не имеющий предков (самый верхний), называется корневым узлом (корнем). Это узел, на котором начинается выполнение большинства операций над деревом. Все прочие узлы могут быть достигнуты путём перехода от корневого узла по рёбрам (или ссылкам). В диаграммах он обычно изображается на самой вершине. В некоторых деревьях, корневой узел обладает особыми свойствами. Каждый узел дерева можно рассматривать как корневой узел поддерева, «растущего» из этого узла.

Графически дерево можно представить разными способами, это показано на рисунке 2 и 3.

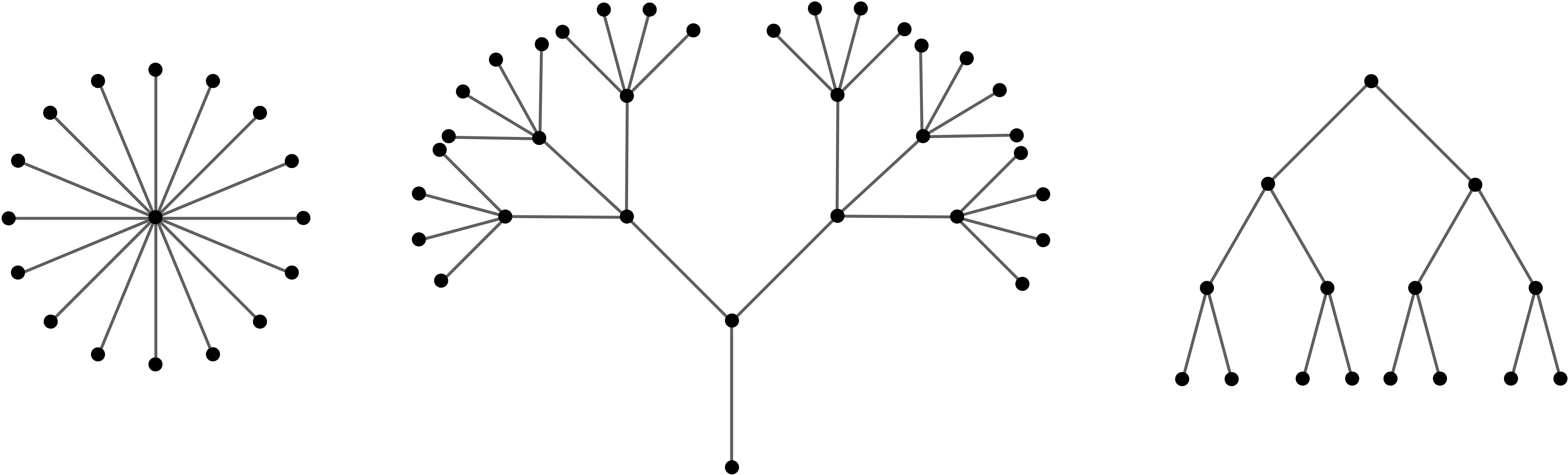


Рисунок 2- Виды деревьев

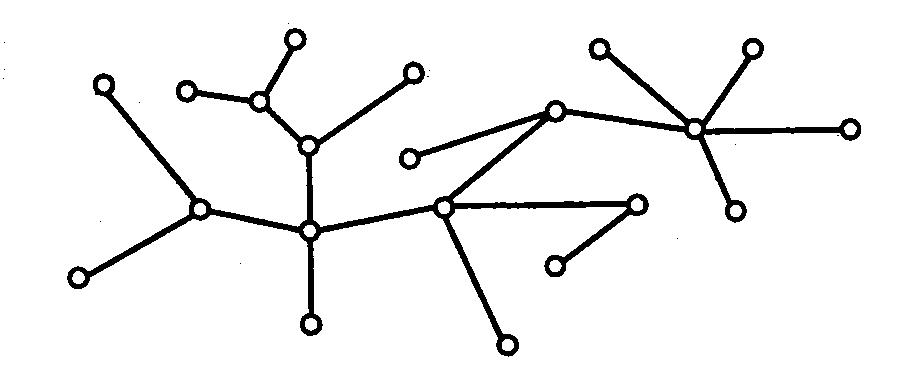


Рисунок 3 – Виды деревьев

Однако же их в основном изображают в стандартном виде, напоминающем перевернутое дерево. (На рисунке 4 представлен стандартный вид).

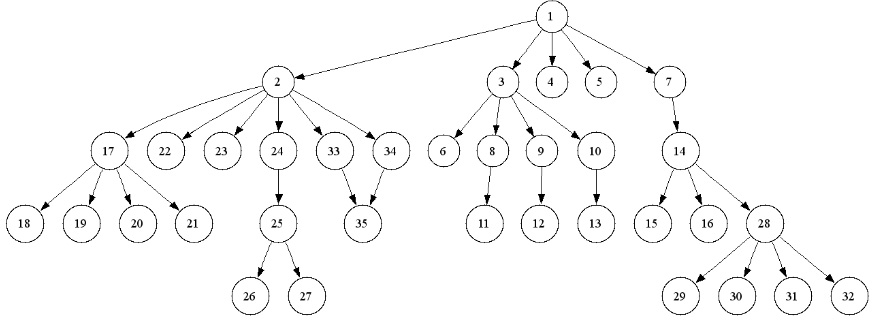


Рисунок 4 – Стандартный вид дерева

## Бинарные деревья

Так как, программа реализована на бинарном дереве, в теоретических сведениях рассматриваются и бинарные деревья.

Одним из видов деревьев является бинарное дерево (или двоичное дерево). Двоичное дерево состоит из вершин и связей между ними. Конкретнее, у дерева есть выделенная вершина-корень и у каждой вершины может быть левый и правый сыновья. Такое дерево представлено на рисунке 5.

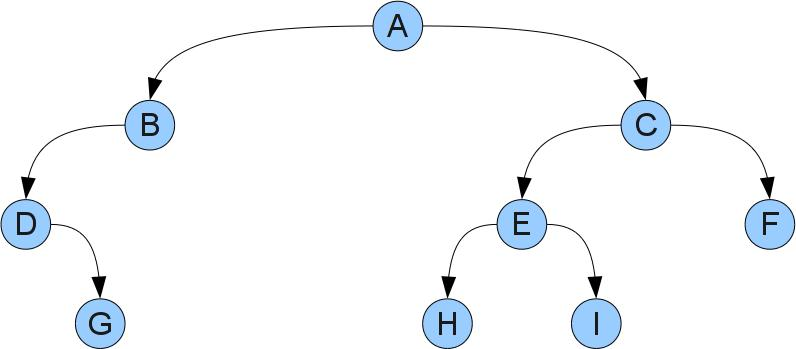


Рисунок 5 – Бинарное дерево

У этого дерева корнем будет вершина A. Видно, что у вершины D отсутствует левый сын, у вершины B – правый, а у вершин G, H, F и I – оба. Вершины без сыновей принято называть листьями.

Каждой вершине X можно сопоставить свое дерево, состоящее из вершины, ее сыновей, сыновей ее сыновей, и т.д. Такое дерево называют поддеревом с корнем X. Левым и правым поддеревьями X называют поддеревья с корнями соответственно в левом и правом сыновьях X. Заметим, что такие поддеревья могут оказаться пустыми, если у X нет соответствующего сына.

Данные в дереве хранятся в его вершинах. В программах вершины дерева обычно представляют структурой, хранящей данные и две ссылки на левого и правого сына. Отсутствующие вершины обозначают null.

Так же у бинарных деревьев существуют определенные правила построения. Именно они и отличают их от других деревьев.

Рассмотрим их на примере: если в вершине X хранится ключ x, то в левом (правом) поддереве должны храниться только ключи меньшие (и большие соответственно) чем x. Если представить это графически, то это будет выглядеть так (рисунок 6)

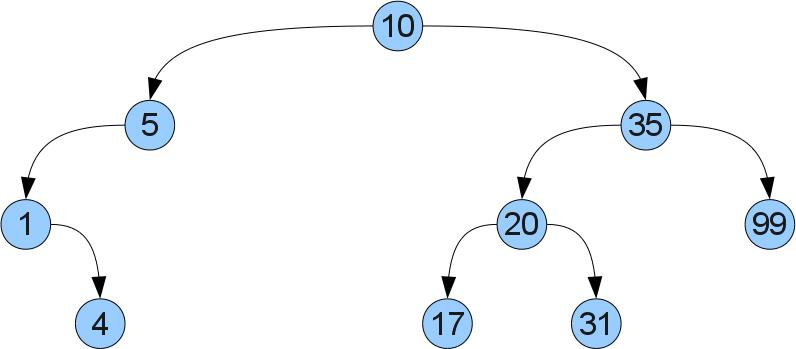


Рисунок 6 – Бинарное дерево с ключами

## Методы обхода деревьев

Обход дерева (известный также как поиск по дереву) — это вид [обхода графа](https://ru.wikipedia.org/w/index.php?title=%D0%9E%D0%B1%D1%85%D0%BE%D0%B4_%D0%B3%D1%80%D0%B0%D1%84%D0%B0&action=edit&redlink=1) и он обозначает процесс посещения (проверки) каждого узла в [структуре данных дерева](https://ru.wikipedia.org/wiki/%D0%94%D0%B5%D1%80%D0%B5%D0%B2%D0%BE_(%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85)) в точности один раз. Такие обходы классифицируются по порядку, в котором узлы посещаются.

В отличие от [связных списков](https://ru.wikipedia.org/wiki/%D0%A1%D0%B2%D1%8F%D0%B7%D0%BD%D1%8B%D0%B9_%D1%81%D0%BF%D0%B8%D1%81%D0%BE%D0%BA), одномерных [массивов](https://ru.wikipedia.org/wiki/%D0%9C%D0%B0%D1%81%D1%81%D0%B8%D0%B2_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) и других [линейных структур данных](https://ru.wikipedia.org/wiki/%D0%9F%D1%80%D0%BE%D0%B5%D0%BA%D1%82:%D0%98%D0%BD%D1%84%D0%BE%D1%80%D0%BC%D0%B0%D1%86%D0%B8%D0%BE%D0%BD%D0%BD%D1%8B%D0%B5_%D1%82%D0%B5%D1%85%D0%BD%D0%BE%D0%BB%D0%BE%D0%B3%D0%B8%D0%B8/%D0%A1%D0%BF%D0%B8%D1%81%D0%BA%D0%B8/%D0%A1%D0%BF%D0%B8%D1%81%D0%BE%D0%BA_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85#%D0%9B%D0%B8%D0%BD%D0%B5%D0%B9%D0%BD%D1%8B%D0%B5_%D1%81%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D1%8B_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85_(Linear_data_structures)), которые канонически обходятся в линейном порядке, деревья можно обходить различными путями. Деревья можно обходить «в глубину» или «в ширину». Существует три основных способа обхода «в глубину»

* прямой (pre-order)
* центрированный (in-order)
* обратный (post-order).

Обход дерева итеративно проходит по всем узлам согласно некоторому алгоритму. Поскольку из данного узла имеется более одного следующего узла (это не линейная структуры данных), то, в предположении последовательных вычислений (а не параллельных), некоторые узлы должны быть отложены, то есть запомнены некоторым способом для дальнейшего посещения. Часто это делается с помощью [стека](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA) (LIFO = последний вошёл — первый вышел) или [очереди](https://ru.wikipedia.org/wiki/%D0%9E%D1%87%D0%B5%D1%80%D0%B5%D0%B4%D1%8C_(%D0%BF%D1%80%D0%BE%D0%B3%D1%80%D0%B0%D0%BC%D0%BC%D0%B8%D1%80%D0%BE%D0%B2%D0%B0%D0%BD%D0%B8%D0%B5)) (FIFO = первый вошёл — первым вышел), в программе реализовано через строку, в которой накапливается результат. Так как дерево самореферентная (ссылающаяся на себя, определённая рекурсивно) структура данных, обход может быть определён [рекурсией](https://ru.wikipedia.org/wiki/%D0%A0%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) или, более тонко, [корекурсией](https://ru.wikipedia.org/wiki/%D0%9A%D0%BE%D1%80%D0%B5%D0%BA%D1%83%D1%80%D1%81%D0%B8%D1%8F) естественным и ясным образом. В этих случаях отложенные узлы запоминаются явно [стеке вызовов](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D0%B5%D0%BA_%D0%B2%D1%8B%D0%B7%D0%BE%D0%B2%D0%BE%D0%B2).

Поиск в ширину легко имплементируется через очередь, включая имплементацию через корекурсию.

Эти поиски называются поиском в глубину ввиду того, что дерево поиска проходится вниз насколько это можно на каждом потомке прежде чем переходить к следующей родственной ветке. Для двоичного дерева они определяются как операции обработки вершины рекурсивно в каждом узле, начиная с корня. Алгоритм обхода следующий

Основной рекурсивный подход для обхода (непустого) бинарного дерева: Начиная с узла N делаем следующее:

(L) Рекурсивно обходим левое поддерево. Этот шаг завершается при попадании опять в узел N.

(R) Рекурсивно обходим правое поддерево. Этот шаг завершается при попадании опять в узел N.

(N) Обрабатываем сам узел N.

Эти шаги могут быть проделаны в любом порядке. Если (L) осуществляется перед (R), процесс называется обходом слева направо, в противном случае — обходом справа налево. Следующие методы показывают обходы слева направо:

Прямой обход (NLR)(Рисунок 7)

1. Проверяем, не является ли текущий узел пустым или null.
2. Показываем поле данных корня (или текущего узла).
3. Обходим левое поддерево рекурсивно, вызвав функцию прямого обхода.
4. Обходим правое поддерево рекурсивно, вызвав функцию прямого обхода.

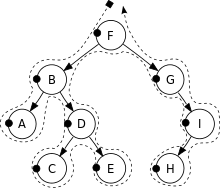
[](https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_preorder.svg?uselang=ru)

Рисунок 7 – прямой обход

Прямой обход: F, B, A, D, C, E, G, I, H.

Центрированный обход (LNR)(Рисунок 8)

1. Проверяем, не является ли текущий узел пустым или null.
2. Обходим левое поддерево рекурсивно, вызвав функцию центрированного обхода.
3. Показываем поле данных корня (или текущего узла).
4. Обходим правое поддерево рекурсивно, вызвав функцию центрированного обхода.

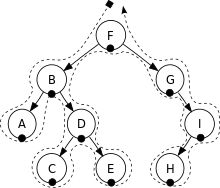
[](https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_inorder.svg?uselang=ru)

Рисунок 8 – централизованный обход

Центрированный обход: A, B, C, D, E, F, G, H, I.

Обратный обход (LRN)

1. Проверяем, не является ли текущий узел пустым или null.
2. Обходим левое поддерево рекурсивно, вызвав функцию обратного обхода.
3. Обходим правое поддерево рекурсивно, вызвав функцию обратного обхода.
4. Показываем поле данных корня (или текущего узла).

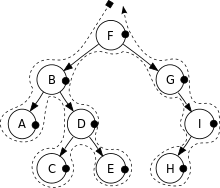
[](https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_postorder.svg?uselang=ru)

Рисунок 9 – обратный обход

Обратный обход: A, C, E, D, B, H, I, G, F.

Последовательность обхода называется секвенциализацией дерева. Последовательность обхода — это список всех посещённых узлов. Ни одна из секвенциализацией согласно прямому, обратному или центрированному порядку не описывает дерево однозначно. Если задано дерево с различными элементами, прямой или обратный обход вместе с центрированным обходом достаточны для описания дерева однозначно. Однако прямой обход вместе с обратным оставляет некоторую неоднозначность в структуре дерева.

Поиск в ширину

Деревья можно обходить также в порядке уровней, где мы посещаем каждый узел на уровне прежде чем перейти на следующий уровень. Такой поиск называется поиском в ширину(Рисунок 10).

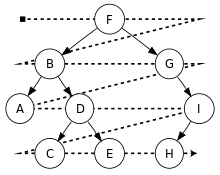
[](https://commons.wikimedia.org/wiki/File:Sorted_binary_tree_breadth-first_traversal.svg?uselang=ru)

Рисунок 10 – поиск в ширину

Уровневый порядок: F, B, G, A, D, I, C, E, H.

# РЕАЛИЗАЦИЯ ПРОГРАММЫ

## Описание разработанной программы

При открытии программы перед пользователем появляется окно, на котором имеется 3 кнопки (button),3 textbox, с помощью которых будет вводиться требуемая информация(элементы дерева, количество элементов дерева при случайном заполнении дерева, искомое значение), 6 элементов Label, с помощью которых отображается информация (теория, просьбы), 1 checkBox (флажок) и 2 элемента listBox, в которых происходит распечатка и обход дерева (см Рисунок 11).

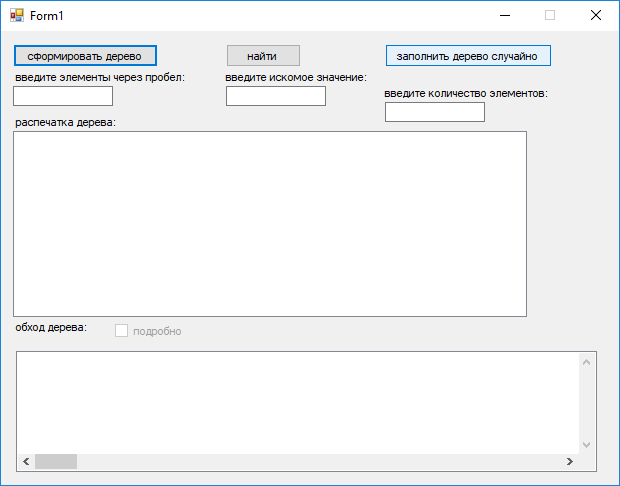


Рисунок 11 – Окно программы

В программе создается отдельный класс для реализации структуры бинарного дерева:

class TreeElement

{

public int data;

public TreeElement left, right;

public TreeElement(int d)

{

data = d;

}

}

class BinaryTree

{

public TreeElement root;

public int Length = 0;

protected int lvl;

private void Print(TreeElement link, ListBox list)

public void PrintTree(ListBox list)

private void Add\_rec(int d, TreeElement link)

public void Add(int d)

public void CLR(TreeElement node, ref string str, ref string s, ref int iskznach,ref int k)

public void LCR(TreeElement node, ref string str, ref string s)

public void RCL(TreeElement node, ref string str, ref string s)

public void Across(TreeElement node, ref string str, ref string s)

}

Для рекурсивной распечатки дерева написаны функции Print и PrintTree:

private void Print(TreeElement link, ListBox list)

{

if (link.left != null)

{

lvl++;

list.Items.Add('|' + new string('\_', lvl) + link.left.data.ToString());

Print(link.left,list);

}

if (link.right != null)

{

lvl++;

list.Items.Add('|' + new string('\_', lvl) + link.right.data.ToString());

Print(link.right,list);

}

lvl--;

}

public void PrintTree(ListBox list)

{

if (root != null)

{

list.Items.Add('|' + root.data.ToString());

Print(root,list);

}

else

list.Items.Add("Дерево пусто");

lvl = 0;

}

Для рекурсивного добавления элементов в бинарное дерево используются функции Add и Add\_rec:

private void Add\_rec(int d, TreeElement link)

{

if (d < link.data)

if (link.left == null)

{

link.left = new TreeElement(d);

Length++;

}

else

Add\_rec(d, link.left);

if (d >= link.data)

if (link.right == null)

{

link.right = new TreeElement(d);

Length++;

}

else

Add\_rec(d, link.right);

}

public void Add(int d)

{

if (root == null)

{

root = new TreeElement(d);

Length++;

}

else

Add\_rec(d, root);

}

Обходы деревьев описаны с помощью следующих функций:

Прямой обход, в котором реализуется поиск элементов дерева:

public void CLR(TreeElement node, ref string str, ref string s, ref int iskznach,ref int k)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " получили значение " + node.data.ToString() + Environment.NewLine;

if (node.data == iskznach) k++;

s += node.data.ToString() + " ";

str += " обходим левое поддерево" + Environment.NewLine;

CLR(node.left, ref str, ref s, ref iskznach, ref k); // обойти левое поддерево

/\*if (detailed)\*/

str += " обходим правое поддерево" + Environment.NewLine;

CLR(node.right, ref str, ref s, ref iskznach, ref k); // обойти правое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

Централизованный обход:

public void LCR(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref - передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " обходим левое поддерево" + Environment.NewLine;

LCR(node.left, ref str, ref s); // обойти левое поддерево

str += " получили значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

str += " обходим правое поддерево" + Environment.NewLine;

LCR(node.right, ref str, ref s); // обойти правое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

Обратный обход:

public void RCL(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " обходим правое поддерево" + Environment.NewLine;

RCL(node.right, ref str, ref s); // обойти правое поддерево

str += " получили значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

str += " обходим левое поддерево" + Environment.NewLine;

RCL(node.left, ref str, ref s); // обойти левое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

Обход в ширину:

public void Across(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

var queue = new Queue<TreeElement>(); // создать новую очередь

str += " заносим в очередь значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

queue.Enqueue(node); // поместить в очередь первый уровень

while (queue.Count != 0) // пока очередь не пуста

{

//если у текущей ветви есть листья, их тоже добавить в очередь

if (queue.Peek().left != null)

{

str += " заносим в очередь значение " + queue.Peek().left.data.ToString() + " из левого поддерева" + Environment.NewLine;

s += queue.Peek().left.data.ToString() + " ";

queue.Enqueue(queue.Peek().left);

}

if (queue.Peek().right != null)

{

str += " заносим в очередь значение " + queue.Peek().right.data.ToString() + " из правого поддерева" + Environment.NewLine;

s += queue.Peek().right.data.ToString() + " ";

queue.Enqueue(queue.Peek().right);

}

//извлечь из очереди информационное поле последнего элемента

str += " извлекаем значение из очереди: " + queue.Peek().data.ToString() + Environment.NewLine;

queue.Dequeue();

}

}

## Руководство пользователя

### Запуск программы

Для запуска программы необходимо выбрать файл «поиск по дереву.exe». После запуска отроется окно как показано на рисунке 11.

### Работа в программе

Для начала работы пользователь должен создать дерево. Это можно сделать двумя способами:

* Добавлять элементы в дерево вручную, введя элементы через пробел в поле ввода и нажав кнопку «сформировать дерево» (Рисунок 12)

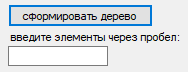


Рисунок 12 – Кнопка «сформировать дерево»

* Создать дерево, заполнив его случайными элементами, введя количество элементов в дереве и нажав кнопку «заполнить дерево случайно» (Рисунок 13).

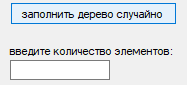


Рисунок 13 – Кнопка «заполнить дерево случайно»

После этого сгенерированное дерево распечатается в поле «распечатка поля», а поле «обход дерева» отобразятся обходы данного дерева (Рисунок 14).

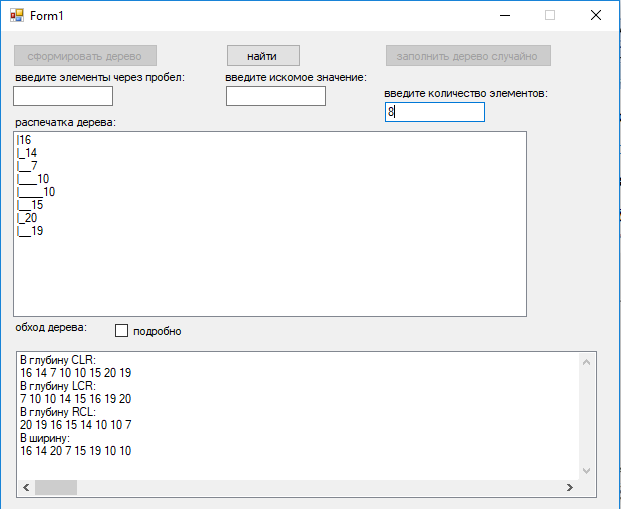


Рисунок 14 – Окно программы с распечатанным деревом

Если поставить флажок в элементе «подробно», то обходы отобразятся пошагово (Рисунок 14).

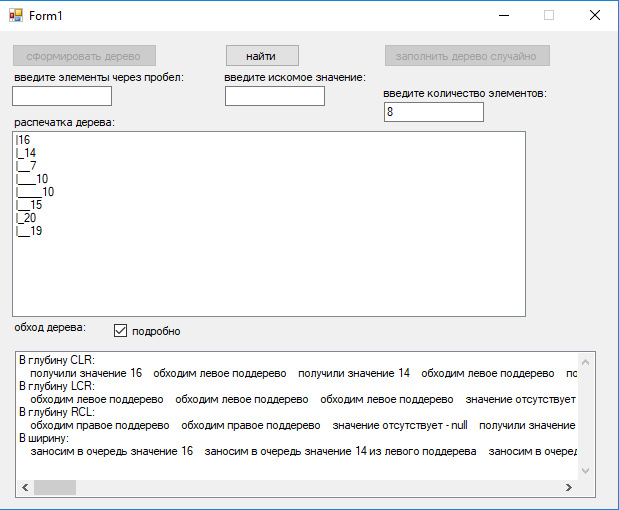


Рисунок 14 – Окно с подробным описанием обходов

Для поиска элемента в дереве, введите в поле «введите искомое значение» и нажмите кнопку «поиск» как на рисунке 15.

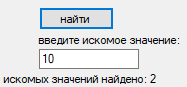


Рисунок 15 – поиск значения

# ЗАКЛЮЧЕНИЕ

Поставленные задачи в работе, а именно:

* изучение классов в C#;
* разработка функций вставки элементов в дерево и обхода по нему;
* сборка программы.

были выполнены. Кроме того, в ходе работы были изучены деревья, как структуры данных, и алгоритмы обхода деревьев. Также была написана программа, предназначенная для объяснения данных методов, что говорит о полном достижении цели, поставленной в курсовой работе.

В программе были реализованы функции добавления в дерево элементов, методы обхода деревьев, структура дерева.

Программа написана на языке C# посредством Microsoft Visual Studio.

# СПИСОК ЛИТЕРАТУРЫ

1. C#. Программирование на языке высокого уровня / Т.А. Павловская. – М.: Питер, 2014. – 427 с.
2. C# 6.0. Справочник. Полное описание языка. / Джозеф Албахари, Бен Албахари – 2016. - 1040 с.
3. Алгоритмы и структуры данных: Курс лекций. / В.И. Лойко – Краснодар: КубГАУ, 2006. – 120 с.
4. C# 6.0 and the .NET 4.6 Framework. / Andrew Troelsen, Philip Japikse – 2015.- 1660 с.
5. Beginning C# Object-Oriented Programming. / Dan Clark – 2013. 384 с.
6. C# 5.0 All-in-One For Dummies. / Bill Sempf, Chuck Sphar, Stephen R. Davis – 2014. 843 с.
7. Design Patterns via C#. Приемы объектно-ориентированного проектирования. / Александр Шевчук, Дмитрий Охрименко, Андрей – 2015. 288 с.

# ПРИЛОЖЕНИЕ

Листинг программы

using System;

using System.Collections.Generic;

using System.ComponentModel;

using System.Data;

using System.Drawing;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

using System.Windows.Forms;

namespace WindowsFormsApp5

{

public partial class Form1 : Form

{

public Form1()

{

InitializeComponent();

}

private int k = 0;

private int iskznach;

private string str1 = null;

private string str2 = null;

private string str3 = null;

private string str4 = null;

private string s1 = null;

private string s2 = null;

private string s3 = null;

private string s4 = null;

private string o1 = "В глубину CLR:";

private string o2 = "В глубину LCR:";

private string o3 = "В глубину RCL:";

private string o4 = "В ширину:";

private BinaryTree Tree = new BinaryTree();

private Random rand = new Random();

private void button1\_Click(object sender, EventArgs e)

{

checkBox1.Enabled = true;

string[] words = textBox1.Text.Split(new char[] { ' ' });

for (int i = 0; i < words.Length; i++)

Tree.Add(Convert.ToInt32(words[i]));

Tree.PrintTree(listBox1);

ObhodTree();

Kratko();

button1.Enabled = false;

button2.Enabled = false;

}

private void checkBox1\_CheckedChanged(object sender, EventArgs e)

{

listBox2.Items.Clear();

if (checkBox1.Checked == true)

{

Podrobno();

}

else

{

Kratko();

}

}

private void button2\_Click(object sender, EventArgs e)

{

checkBox1.Enabled = true;

int n = Convert.ToInt32(textBox2.Text);

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

Tree.Add(rand.Next(0, 21));

Tree.PrintTree(listBox1);

ObhodTree();

Kratko();

button1.Enabled = false;

button2.Enabled = false;

}

private void Kratko()

{

listBox2.Items.Add(o1);

listBox2.Items.Add(s1);

listBox2.Items.Add(o2);

listBox2.Items.Add(s2);

listBox2.Items.Add(o3);

listBox2.Items.Add(s3);

listBox2.Items.Add(o4);

listBox2.Items.Add(s4);

}

private void Podrobno()

{

listBox2.Items.Add(o1);

listBox2.Items.Add(str1);

listBox2.Items.Add(o2);

listBox2.Items.Add(str2);

listBox2.Items.Add(o3);

listBox2.Items.Add(str3);

listBox2.Items.Add(o4);

listBox2.Items.Add(str4);

}

private void ObhodTree()

{

Tree.CLR(Tree.root, ref str1, ref s1, ref iskznach, ref k);

Tree.LCR(Tree.root, ref str2, ref s2);

Tree.RCL(Tree.root, ref str3, ref s3);

Tree.Across(Tree.root, ref str4, ref s4);

}

private void button3\_Click(object sender, EventArgs e)

{

if (Tree.root.data == iskznach) k--;

iskznach = Convert.ToInt32(textBox3.Text);

Tree.CLR(Tree.root, ref str1, ref s1, ref iskznach, ref k);

label5.Text = k.ToString();

if (k != 0)

label7.Text = "искомых значений найдено: " + k;

else

label7.Text = "искомое значение не найдено";

}

}

class TreeElement

{

public int data;

public TreeElement left, right;

public TreeElement(int d)

{

data = d;

}

}

class BinaryTree

{

public TreeElement root;

public int Length = 0;

protected int lvl;

private void Print(TreeElement link, ListBox list)

{

if (link.left != null)

{

lvl++;

list.Items.Add('|' + new string('\_', lvl) + link.left.data.ToString());

Print(link.left,list);

}

if (link.right != null)

{

lvl++;

list.Items.Add('|' + new string('\_', lvl) + link.right.data.ToString());

Print(link.right,list);

}

lvl--;

}

public void PrintTree(ListBox list)

{

if (root != null)

{

list.Items.Add('|' + root.data.ToString());

Print(root,list);

}

else

list.Items.Add("Дерево пусто");

lvl = 0;

}

private void Add\_rec(int d, TreeElement link)

{

if (d < link.data)

if (link.left == null)

{

link.left = new TreeElement(d);

Length++;

}

else

Add\_rec(d, link.left);

if (d >= link.data)

if (link.right == null)

{

link.right = new TreeElement(d);

Length++;

}

else

Add\_rec(d, link.right);

}

public void Add(int d)

{

if (root == null)

{

root = new TreeElement(d);

Length++;

}

else

Add\_rec(d, root);

}

// прямой обход (CLR - center, left, right)

public void CLR(TreeElement node, ref string str, ref string s, ref int iskznach,ref int k)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " получили значение " + node.data.ToString() + Environment.NewLine;

if (node.data == iskznach) k++;

s += node.data.ToString() + " ";

str += " обходим левое поддерево" + Environment.NewLine;

CLR(node.left, ref str, ref s, ref iskznach, ref k); // обойти левое поддерево

/\*if (detailed)\*/

str += " обходим правое поддерево" + Environment.NewLine;

CLR(node.right, ref str, ref s, ref iskznach, ref k); // обойти правое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

// централизованный обход (LCR - left, center, right)

public void LCR(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref - передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " обходим левое поддерево" + Environment.NewLine;

LCR(node.left, ref str, ref s); // обойти левое поддерево

str += " получили значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

str += " обходим правое поддерево" + Environment.NewLine;

LCR(node.right, ref str, ref s); // обойти правое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

// обратный обход (RCL -right, center, left)

public void RCL(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

if (node != null)

{

str += " обходим правое поддерево" + Environment.NewLine;

RCL(node.right, ref str, ref s); // обойти правое поддерево

str += " получили значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

str += " обходим левое поддерево" + Environment.NewLine;

RCL(node.left, ref str, ref s); // обойти левое поддерево

}

else str += " значение отсутствует - null" + Environment.NewLine;

}

// обход дерева в ширину (итерационно, используется очередь)

public void Across(TreeElement node, ref string str, ref string s)

{

/\*

Аргументы метода:

1. TreeNode node - текущий "элемент дерева" (ref передача по ссылке)

2. ref string s - строка, в которой накапливается результат (ref - передача по ссылке)

\*/

var queue = new Queue<TreeElement>(); // создать новую очередь

str += " заносим в очередь значение " + node.data.ToString() + Environment.NewLine;

s += node.data.ToString() + " ";

queue.Enqueue(node); // поместить в очередь первый уровень

while (queue.Count != 0) // пока очередь не пуста

{

//если у текущей ветви есть листья, их тоже добавить в очередь

if (queue.Peek().left != null)

{

str += " заносим в очередь значение " + queue.Peek().left.data.ToString() + " из левого поддерева" + Environment.NewLine;

s += queue.Peek().left.data.ToString() + " ";

queue.Enqueue(queue.Peek().left);

}

if (queue.Peek().right != null)

{

str += " заносим в очередь значение " + queue.Peek().right.data.ToString() + " из правого поддерева" + Environment.NewLine;

s += queue.Peek().right.data.ToString() + " ";

queue.Enqueue(queue.Peek().right);

}

//извлечь из очереди информационное поле последнего элемента

str += " извлекаем значение из очереди: " + queue.Peek().data.ToString() + Environment.NewLine;

queue.Dequeue();

}

}

}

}