**Тема**:  Реализация деревьев

**Цель:** Приобрести навыки программирования и использования в прикладных задачах динамических иерархических структур данных: бинарных деревьев.

# Краткая теория:

# Основным способом организации иерархической структуры связанных отношениями подчиненности элементов является древовидная модель данных, или просто дерево. Эта модель представляет собой важнейший частный случай графа, используемый для представления динамических данных при решении разных задач, в том числе быстрого поиска информации.

# Деревом называется совокупность элементов – узлов , называемых вершинами, и связывающих их «родительских» отношений таким образом, что формируется иерархическая структура данных. Узлы, так же как и элементы списка или любой другой динамической структуры АТД, могут представлять собой элементы любого типа, атомарного или структурированного. Начальный узел является узлом нулевого уровня и называется корнем дерева. Элемент, в который входит одна ветвь, но не выходит ни одной, назван листом дерева. Древовидная иерархическая структура обладает такими отличительными свойствами:

# − имеется узел, в который не входит ни одна дуг а, это корень;

# − в любую вершину, кроме корневой, входит только одна дуга;

# − имеются концевые узлы, у которых нет выходящих дуг, это листья

# Поскольку в иерархиях принято использовать принцип «родственной» подчиненности, то узлы дерева могут быть представлены как связанные совокупности потомков и предков, причем предок может иметь несколько потомков, корень предка не имеет, а у листа нет потомков. Переходом от предка к потомку осуществляется перемещение на следующий уровень дерева. Количество уровней, на которых размещены вершины, принято называть высотой (глубиной) дерева. Обособленная часть иерархической структуры данных может быть представлена в виде самостоятельного дерева и являться поддеревом. Часто выделение поддеревьев используется в рекурсивных алгоритмах обработки информации, поэтому дерево может считаться рекурсивной структурой данных. Максимальное количество дуг, выходящих из вершины, определяет степень вершины. Обычно различают два типа деревьев по этому показателю: двоичные со степенью дерева не более двух и сильноветвящиеся с произвольной степенью.

# Дерево, у которого ветви упорядочены по определенному критерию, так и называется – упорядоченным. Реализацию связей в дереве удобнее всего делать на основе списочного представления, при этом каждый узел имеет одно или несколько полей данных и как минимум два поля указателей: указатель на начало списка потомков и указатель на текущий элемент в этом списке. При этом обязателен глобальный указатель на корень дерева. Для выполнения заданной операции над всеми узлами следует их обработать путем обхода дерева, то есть составления определенной последовательности вершин, где каждая встречается только один раз. Обход дерева может быть реализован разными способами, из которых наиболее распространены следующие: прямой, симметричный и обратный.

Деревья относятся к структурам данных, имеющим большое многообразие. Известны такие древовидные структуры, как бинарные, красно-черные деревья, В-деревья, АВЛ-деревья, матричные, смешанные и другие. Для задач программирования наиболее характерным является бинарное (двоичное) дерево, которое представляет собой дерево, где каждая вершина имеет не более двух потомков. Элемент бинарного дерева содержит информационное поле, возможны служебное поле и две ссылки на различные бинарные поддеревья, при этом на каждый элемент дерева есть только одна ссылка.

Основными операциями, которые позволяют работать с бинарными деревьями, как уже отмечалось при определении АТД «Дерево», являются следующие:

− создание бинарного дерева;

− печать бинарного дерева;

− обход бинарного дерева;

− вставка элемента в бинарное дерево;

− удаление элемента из бинарного дерева;

− проверка пустоты бинарного дерева;

− удаление бинарного дерева.

Далее показаны примеры объявления и некоторых функций на языке

# Создание бинарного дерева

# Прямой обход (Pre-Order)[​](https://stavis-dev.github.io/python/types/binary-tree/?ysclid=m3h6u9xd7s588894686#%D0%BF%D1%80%D1%8F%D0%BC%D0%BE%D0%B9-%D0%BE%D0%B1%D1%85%D0%BE%D0%B4-pre-order)

Написать обход просто. Прямой обход — это посещение родительского узла, а затем каждого из его потомков. Мы «посетим» родительский узел, выведя его на экран, а затем «обойдем» детей, вызывая этот метод рекурсивно для каждого узла-потомка.

**Вставка элемента** в бинарное дерево может быть реализована с помощью следующего алгоритма:

Начать с корневого узла.

Сравнить ключ нового узла с ключом текущего узла.

Если новый ключ меньше, перейти к левому поддереву, если больше — к правому.

Повторять процесс до тех пор, пока не найдётся подходящее место для вставки нового узла (либо левый, либо правый потомок отсутствует).

Алгоритм:

Если дерево пустое, новый узел становится корневым узлом.

Иначе рекурсивно находим правильное место и добавляем новый узел.

2. Удаление (Deletion):

Принцип работы:

Найти узел для удаления.

Рассмотреть три случая:

Узел является листом (не имеет детей): просто удаляем узел.

Узел имеет одного ребёнка: заменяем узел его ребёнком.

Узел имеет двух детей: находим наименьший узел в правом поддереве (или наибольший в левом), копируем его значение в удаляемый узел и рекурсивно удаляем наименьший узел в правом поддереве.

Алгоритм:

Найти узел с заданным ключом.

В зависимости от случая выполнить соответствующее удаление и перераспределение узлов.

3. Поиск (Search):

Принцип работы:

Начинаем с корневого узла.

Сравниваем ключ искомого узла с ключом текущего узла.

Если ключ совпадает, возвращаем узел.

Если ключ меньше, переходим к левому поддереву, если больше — к правому.

Повторяем процесс до тех пор, пока не найдём узел с искомым ключом или не достигнем конца дерева (в этом случае узел не найден).

Алгоритм:

Если дерево пустое или ключ узла совпадает с искомым, возвращаем узел.

Если ключ искомого узла меньше, рекурсивно ищем в левом поддереве.

Если ключ искомого узла больше, рекурсивно ищем в правом поддереве.

# Задание:

Выполнение работы заключается в разработке программы с использованием консольного или графического пользовательского интерфейса на усмотрение студента. Создание бинарного дерева и работа с ним должны осуществляться с использованием указателей; применение объектных типов данных типа TTree, встроенных библиотечных компонентов типа TreeView, JTree, TreeNode и иных средств, применяемых в различных современных языках программирования, не допускается. При выборе языка программирования основным требованием является поддержка работы с указателями. Разработанная программа должна выполнять последовательно две задачи.

Дано число N (>0) и произвольный набор из N чисел.

1. Сформировать начальную структуру числовых данных в виде двоичного дерева заданного вида. Вывести указатель на корень созданного дерева.
2. Имея указатель на корень сформированного непустого дерева, выполнить преобразование сформированного двоичного дерева к иному виду согласно заданию.

Программа должна обеспечивать диалог с помощью меню, вывод промежуточного и итогового результатов работы и контроль ошибок при вводе.

**Варианты заданий:**

|  |  |  |
| --- | --- | --- |
| **№** | **Начальная структура** | **Преобразование дерева** |
|  | Создать дерево из N вершин, в котором каждая вершина (кроме корня) является правой дочерней вершиной. Каждой создаваемой вершине присваивать очередное значение из исходного набора | Ко всем вершинам дерева, которые содержат ровно по одной дочерней вершине, добавить еще одну дочернюю вершину-лист. Значение каждой добавленной вершины положить равным удвоенному значению ее родительской вершины |
|  | Создать дерево из N вершин, в котором каждая внутренняя вершина имеет только одну дочернюю вершину, причем правые и левые дочерние вершины чередуются (корень имеет левую дочернюю вершину). Каждой создаваемой вершине присваивать очередное значение из исходного набор | Не изменяя глубины L исходного дерева, дополнить его до полного дерева, то есть дерева, все листья которого находятся на уровне L. Значения всех добавленных вершин положить равными –1 |
|  | Создать дерево из N вершин, в котором каждая внутренняя вершина имеет только одну дочернюю вершину, причем если значение вершины является нечетным, то она имеет левую дочернюю вершину, а если четным, то–правую. Каждой создаваемой вершине присваивать очередное значение из исходного набора | Удалить вершины дерева, имеющие значения, меньшие значения корня, вместе со всеми их дочерними вершинами. При удалении вершин освобождать память, которую они занимали |
|  | Создать дерево из N вершин, в котором каждая левая дочерняя вершина является листом, а правая дочерняя вершина является внутренней. Для каждой внутренней вершины вначале создавать левую дочернюю вершину, а затем – правую (если она существует); каждой создаваемой вершине присваивать очередное значение из исходного набора | Ко всем вершинам дерева, которые являются листьями, добавить по одной дочерней вершине-листу; при этом к исходной вер-шине с нечетным значением добавляется левая дочерняя верши-на, а к вершине с четным значением – правая. Значение каждой добавленной вершины положить равным значению ее родительской вершины |
|  | Создать дерево из N вершин со следующей структурой: если вершина дерева является внутренней, то в случае, если она имеет нечетное значение, ее левая дочерняя вер-шина должна быть листом, а в случае, если она имеет четное значение, листом должна быть ее правая вершина. Для каждой внутренней вершины вначале создавать дочернюю вершину-лист, а затем – дочернюю внутреннюю вершину (если данная вер-шина существует); каждой создаваемой вершине присваивать очередное значение из исходного набора | Ко всем вершинам дерева, которые являются листьями, добавить по две дочерние вершины-листа: левую – со значением 10 и правую – со значением 11 |
|  | Создать дерево, корень которого имеет значение N, а вершины обладают следующими свойствами: вершина с четным значением K имеет левую дочернюю вершину со значением K/2 и не имеет правой дочерней вершины; вершина со значением 1 является листом; вершина с любым другим нечетным значением K имеет две дочерние вершины: левую со значением K/2 и правую со значением K –K/2 (символ «/» обозначает операцию деления нацело) | Для вершин дерева, имеющих две дочерние вершины, удалить одну из дочерних вершин: правую, если родительская вершина имеет четное значение, и левую – в противном случае (вершины дерева перебирать в префиксном порядке, при удалении вершины удалять и всех ее потомков). Для удаленных вершин освобождать память, которую они занимали |
|  | Создать дерево глубиной L, содержащее вершины со значениями из исходного набора. Вершины добавлять к дереву в префиксном порядке, используя алгоритм, который для каждой вершины уровня, не превышающего L, вначале создает саму вершину с очередным значением из исходного набора, затем ее левое поддерево соответствующей глубины, а затем ее правое поддерево. Если для заполнения дерева глубиной L требуется менее N вершин, то оставшиеся числа из исходного набора не использовать | Удалить каждую вершину дерева, являющуюся листом; при этом со ответствующее поле родительской вершины (Left или Right) следует положить равным nil. При удалении вершин освобождать память, которую они занимали |
|  | Создать идеально сбалансированное дерево из N вершин с заданными значениями (то есть дерево, для каждой вершины которого количество вершин в его левом и правом поддереве отличается не более чем на 1) и вывести указатель на его корень. Для создания дерева использовать рекурсивный алгоритм, который создает вершину дерева с очередным значением, после чего создается левое поддерево с N/2 вершинами и правое поддерево с N –1–N/2 вершинами (символ «/» обозначает операцию деления нацело) | Удалить из дерева все вершины, кроме корня, и освободить па-мять, которую занимали удаленные вершины (полям Left и Right корня следует присвоить значение nil) |
|  | Создать идеально сбалансированное дере-во из N вершин и вывести указатель на его корень. Значение каждой вершины поло-жить равным уровню этой вершины (например, корень дерева должен иметь значение 0, его дочерние вершины – значение 1 и т. д.). При формировании дерева использовать алгоритм, описанный в вари-анте 8 | Для всех внутренних вершин де-рева поменять местами их левые и правые дочерние вершины (то есть значения полей Left и Right) |
|  | Создать дерево из N вершин, в котором каждая вершина (кроме корня) является левой дочерней вершиной. Каждой создаваемой вершине присваивать очередное значение из исходного набора. | Изменить порядок заполнения значений дерева на обратный, за-писав в корень последнее значение, и т. д |
|  | Создать дерево из N вершин, в котором каждая внутренняя вершина имеет только одну дочернюю вершину, причем правые и левые дочерние вершины чередуются (корень имеет правую дочернюю вершину). Каждой создаваемой вершине присваивать очередное значение из исходного набора | Добавить 1 к значению каждого листа дерева и вычесть 1 из значения каждой внутренней вершины |
|  | Создать дерево из N вершин, в котором каждая правая дочерняя вершина является листом, а левая – внутренней. Для каждой внутренней вершины вначале создавать правую дочернюю вершину, а затем левую (если она существует); каждой создаваемой вершине присваивать очередное значение из исходного набора | Для каждой вершины дерева с четным значением уменьшить ее значение в два раза |
|  | Создать дерево из N вершин со следующей структурой: если вершина дерева является внутренней, то в случае, если она имеет четное значение, ее правая дочерняя вершина должна быть листом, а в случае, если она имеет нечетное значение, листом должна быть ее левая вершина. Для каждой внутренней вершины вначале создавать дочернюю вершину-лист, а затем дочернюю внутреннюю вершину (если данная вершина существует); каждой создаваемой вершине присваивать очередное значение из исходного набора | Для каждой вершины дерева, имеющей две дочерние вершины, поменять местами значения дочерних вершин (то есть значения их полей Data) |
|  | Создать дерево из N вершин, в котором каждая внутренняя вершина имеет только одну дочернюю вершину, причем правые и левые дочерние вершины определяются случайным образом. Каждой создаваемой вершине присваивать очередное значение из исходного набора | Для каждой вершины дерева с нечетным значением увеличить ее значение в два раза |
|  | Создать дерево из N вершин, в котором каждая одна дочерняя вершина является листом, а другая дочерняя вершина является внутренней, выбор вершины-листа определяется случайным образом. Для каждой внутренней вершины дочерние создаются аналогично случайным выбором. Каждой создаваемой вершине присваивать очередное значение из исходного набора | Добавить к значению каждого листа дерева значение соседней дочерней вершины и оставить без изменения значения внутренних вершин |