Министерство науки и высшего образования Российской Федерации
федеральное государственное бюджетное
образовательное учреждение высшего образования
«Российский экономический университет имени Г.В. Плеханова»

Институт цифровой экономики и информационных технологий

Базовая кафедра цифровой экономики института развития информационного общества

КУРСОВАЯ РАБОТА

по дисциплине «Алгоритмизация и программирование»
на тему «Определения количества узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками в бинарном дереве»

Выполнил
обучающийся группы
15.11д-мо11/19б
очной формы обучения
института цифровой экономики и
информационных технологий
Гришанин Вячеслав Валерьевич
Научный руководитель:
к.э.н., доцент Комлева Н.В.,

Москва – 2020

Содержание:

Введение

Основная задача компьютеров — это обработка данных. Для того, чтобы компьютер имел возможность обработать данные, их необходимо поместить в память компьютера. То, как организованно хранение данных в памяти компьютера, определяется структурой данных. Причиной, по которой существуют множество различных структур данных, является то, что не существует такой универсальной структуры данных, с помощью которой можно наиболее оптимально по затрачиваемым ресурсам (времени и памяти) решить любую задачу, которая ставится перед данными. Поэтому оптимальность выбора структуры данных для хранения информации зависит от конкретного назначения и типа данных, определенных целей и задач их обработки.

Например, в одних случаях может требоваться хранить данные, являющиеся определенной последовательностью равно размерных элементов и иметь возможность получить быстрый доступ к произвольному элементу данных. Примером такого рода данных является текст определенного объема. Текст – это строго определенная последовательность символов. Следовательно, элементами текста являются символы, расположенные последовательно. Также, при работе с текстом может требоваться получить доступ к произвольным частям текста. Например, к произвольным предложениям или абзацам. В таком случае можно использовать структуру данных, элементы которой храниться в памяти смежно, то есть располагаются друг за другом. Это позволяет сохранить информацию о взаимном расположении элементов данных (символов в этом примере), а также, зная каким по счету является элемент данных(символ) - быстро получить произвольный элемент данных(символ) за счет вычисления смещения относительно начала контейнера, равного произведению порядкового номера элемента(символа) и объема (размера

отрезка) памяти, который занимает элемент(символ). Примером такой структуры данных является массив.

Но в другом случае может также потребоваться хранить данные, являющиеся определенной последовательностью равно размерных элементов, но вместо произвольного доступа к элементам данных может требоваться иметь возможность быстро включать и исключать любые элементы данных. Примером такого рода данных является информация об очереди, например, людей, которые могут в любой момент покинуть эту очередь. Все, что знает человек в очереди(элемент), это определенная цель, за которой он в ней стоит (данные) и то, за кем он стоит (ссылка на следующий узел). Все, что следует же знать об очереди, это то, кто в ней первый и последний (информация о том, что является первым и последним элементом контейнера). В этом случае структура, которая хранит информацию в памяти смежно, не будет оптимальной для решения этой задачи, так как для произвольной вставки и удаления потребуется реорганизовать контейнер, создав фактически новый контейнер, элементы которого от начала и до вставляемого/удаляемого будут те же, что и у реорганизуемого контейнера, после которых в случае вставки следует вставляемый элемент. После чего следуют элементы контейнера от вставляемого/удаляемого контейнера до конца реорганизуемого контейнера. На практике из-за сегментации памяти в случае, если нет возможности расположить получившийся контейнер на прежнем месте, может потребоваться дополнительно найти в памяти новое место, которое может полностью уместить новый контейнер. Все эти действия затрачивают ресурсы в виде памяти и времени. Поэтому более оптимальным решением будет организация данных в виде связанного списка, представляющего из себя структуру данных, элементами которой выступают узлы, хранящие данные и ссылку на следующий узел списка. Эта структура данных, за счет простого редактирования связей между

узлами, дает возможность более быстрой и простой по сравнению с массивом вставки и удаления элемента за или перед произвольным элементом. Также эта структура решает проблему, связанную с сегментацией памяти, так как в отличии от массива, связанному списку для размещения в памяти не нужен непрерывный блок памяти, равный размеру контейнера, так как отдельные элементы связанного списка(узлы) могут быть разбросаны по всей памяти. Однако недостатком этой структуры данных является то, что, например, в отличии от массива, нет возможности получить прямой доступ к произвольному элементу по порядковому номеру.

Таким образом для каждой определенной задачи существует своя определенная структура данных. В этой курсовой работе рассматривается структура данных, которая решает задачи, в которых требуется возможность скоростного поиска элемента в контейнере по значению одновременно со скоростным добавлением новых элементов в контейнер. Этой структурой данных является бинарное дерево или двоичное дерево.

Актуальность темы обусловлена широким спектром “повседневных” задач (неформально выражаясь - задач по составлению словаря) при разработке программ, для решения которых требуется такая структура данных как бинарное дерево. Примером такой задачи является обработка телефонной книги или базы данных теле-абонентов. Такого рода данные обычно имеют большой объем и программе требуется быстро получать данные об абонентах по номеру телефона. Если просто размещать данные в памяти последовательно в виде массива или списка, то чтобы найти абонента по номеру телефона придется перебрать всех абонентов начиная с первого и заканчивая искомым абонентом. В худшем случае, если искомый абонент является последним в такой телефонной книге, то придется перебрать всех абонентов. Можно отсортировать по возрастанию или убыванию всех абонентов и с помощью бинарного поиска быстро

находить нужного абонента, но в таком случае при добавлении нового абонента для того, чтобы элементы в контейнере остались отсортированы потребуется заново сортировать и реорганизовать весь контейнер, который содержит абонентов. Такая структура данных как двоичное дерево поиска, основанная на бинарном дереве в свою очередь позволяет использовать алгоритм бинарного поиска, а также имеет возможность скоростного добавления элементов в контейнер, без необходимости заново сортировать и перестраивать контейнер.

Также важность и востребованность такой структуры данных как бинарное дерево подтверждает, то что во многих популярных языках программирования на основе бинарного дерева есть встроенные(стандартные) контейнеры. Примером такого языка программирования является C++, стандартная библиотека которого – STL, содержит четыре контейнера на основе бинарного дерева (множество, мультимножество, отображение и мультиотображение)

Целью данной курсовой работы является исследование такой структуры данных как двоичное дерево поиска, систематизация и расширение теоретических знаний и их практическое применение в процессе разработки.

Задание на курсовую работу

Для достижения поставленной цели необходимо решить следующие **задачи:**

- рассмотреть понятие, сущность и необходимость двоичного дерева поиска
- спроектировать и разработать программу сложной структуры, которая реализует основные действия с двоичным деревом поиска

- реализовать индивидуальную задачу: определить количество узлов двоичного дерева поиска, лежащих на пути между двумя его узлами, заданными своими ключевыми признаками.

Объектом данного исследования является бинарное дерево

Предметом исследования являются операции двоичным деревом поиска

Метод исследования. В данной курсовой работе применяются такие общенаучные методы исследования, как анализ, эксперимент и моделирование.

Глава 1. Двоичное дерево. Двоичное дерево поиска.

1.1. Двоичное дерево

Бинарное дерево – это разновидность древовидной структуры, которая в свою очередь является графом без циклов, то есть без замкнутых цепей. Среди древовидных структур бинарное дерево выделяется тем, что каждый узел дерева имеет до двух прямых потомков. В теории множеств двоичное дерево определяется рекурсивно через упорядоченный набор фиксированной длинны (L, S, R), где L и R - это бинарные деревья или пустое множество, а S - это множество из одного элемента, являющегося бинарным деревом. В некоторых случаях бинарное дерево может быть пустым множеством. Один из узлов дерева не имеет родительского узла и называется корневым узлом. В данной курсовой работе используется именно это определение бинарного дерева.

Существует два подхода использования бинарных деревьев. Первый подход основан на доступе к узлам по их ключевому признаку, но расположения узла в дереве не имеет значения и не учитывается как часть данных, которые представляет структура. Примерами структур, которые основанные на этом подходе являются, например, двоичное дерево поиска и двоичная куча. Как будет показано далее расположение узлов в таких бинарных деревьях зависит только от порядка, в котором эти узлы добавлялись в двоичное дерево и могут быть перестроены без изменения

данных, которые они представляют. Это может быть использовано, например, для балансировки бинарного дерева. Виды бинарных деревьев, основанные на данном подходе используются для эффективного поиска и сортировки.

Второй подход основан на том, что расположение узла в дереве имеет значение и то, находится ли некорневой узел слева или справа относительно своего родителя, учитывается как часть данных, которые представляет структура. Примерами данных, которые может представлять эта структура, являются кладограммы или родословные деревья, так как любой человек имеет два биологических родителя (одну мать и одного отца), матери в таких деревья располагаются всегда с одной стронны (обычно с правой стороны от потомка), а отцы всегда с другой стороны (обычно с левой стороны от потомка). Деревья, основанные на данном подходе, используются в Алгоритме кодирования Хаффмана, который в свою очередь широко применяется при сжатии данных. Другим примером который использует данный подход является код Морзе, где тире соответствует одному направлению (обычно переходу на правое поддерево), а точка другому (обычно переходу на левое поддерево) и каждая буква соответствует только одной определенной последовательности точек и тире, то есть узел с символом может находится только в одном определенном месте двоичного дерева поиска.

Далее приведены некоторые из наиболее востребованных видов бинарных деревьев и их краткие характеристики.

Двоичное дерево поиска - бинарное дерево, особенностью которого является, то, что выделяются левое поддерево узла, ключевые признаки узлов которого меньше ключевого признака узла, выступающего по отношению к поддереву корневым узлом и правое поддерево узла, ключевые признаки узлов которого больше ключевого признака узла, выступающего по отношению к поддереву корневым узлом. Если ключевые признаки узла-потомка и узла-предка равны, то такой узел может не является отдельным узлом дерева или является всегда либо левым потомком, либо правым потомком. Для такого дерева должна быть определенна операция сравнения для ключевого

признака. Данный вид бинарного дерева позволяет использовать алгоритм бинарного поиска для эффективного извлечения, добавления и удаления элементов, данных(узлов). Бинарное дерево поиска используется для реализации таких контейнеров как множество и отображение (таблица поиска). Подробнее двоичное дерево поиска будет рассмотрено далее в разделе 1.2.

АВЛ-дерево

Как будет показано далее, эффективность алгоритмов при работе с двоичным деревом поиска зависит от сбалансированности дерева, то есть от разности высот поддеревьев. Под высотой поддерева подразумевается количество узлов между листом поддерева (узлом не имеющего потомков) и корнем поддерева. Для исправления проблемы скошенных двоичных деревьев поиска были разработаны самобалансирующиеся виды двоичных деревьев поиска. Одним из таких бинарных деревьев является АВЛ-дерево. Суть АВЛ-дерева в том, что при операциях, изменяющих структуру дерева, таких как например вставка нового элемента или удаление узла, структура выполняет операцию балансировки дерева при условии, если после выполнения операции высота поддеревьев отличается более, чем на единицу. Балансировка АВЛ-дерева осуществляется с помощью четырех возможных операций: малого левого вращения, малого правого вращения, большого левого вращения (композиция малого правого вращения и малого левого вращения), большого правого вращения (композиция малого левого и малого правого вращения), которые по сути, как бы перетягивают поддеревья так, чтобы дерево вновь стало сбалансированным. Также существует модификация АВЛ-дерева – дерево Фибоначчи. Особенностью данного АВЛ-дерева является то, что число возможных вершин дерева минимально, что экономит размер памяти, выделяемый на хранения адреса узла. Также для более эффективной работы алгоритмов в узлах АВЛ-дерева могут хранится разницы высот. В данной курсовой работе не приводится реализация АВЛ-дерева и особенности его

работы. Но со всем этим можно ознакомится в работе Адельсон-Вельский Г. М., Ландис Е. М. «Один алгоритм организации информации».

Крано-черные деревья.

Это другой вид самобалансирующегося двоичного дерева поиска. Особенностью красно-черных двоичных деревьев является то, что каждый узел также характеризуется так называемым цветом (бинарным состоянием), который может быть либо красным, либо черным. Это состояние используется алгоритмами обработки красно-черного дерева.

Данный вид самобалансирующегося двоичного дерева, возможно, более эффективен при больших объемах данных, но обычно красно-черные деревья проигрывают АВЛ-деревьям по скорости. Однако красно-черные деревья являются одним из наиболее популярных видов самобалансирующихся деревьев и используются в таких C++ библиотеках структур данных как Boost (multi-index containers) и во многих реализациях стандартной библиотеки STL(set, map). Также данная структура используется для реализации TreeMap в Java. Тема реализации и особенностей красно-черного дерева выходит за пределы данной курсовой работы. Подробнее с крано-черными деревьями можно ознакомится в литературе на соответствующую тему.

Двоичная куча.

Для решения некоторых задач требуется такой тип данных как приоритетная очередь. Приоритетная очередь — это такой абстрактный тип данных, для которого определенны две операции извлечение наиболее приоритетного элемента (например, максимального или минимального по значению) и добавления нового элемента в приоритетную очередь. Примерами задач, в которых требуется приоритетная очередь.

Существует множество различных способов реализовать приоритетную очередь. Простейший из них - это, например, реализация с помощью отсортированного массива или, например, реализация с помощью связанного списка. Но при использовании массива вставка в приоритетную очередь будет выполнятся не самым эффективным образом, так как каждый раз придется

пересортировывать массив. Связанный список также не является самой эффективной структурой данных для реализации приоритетной очереди, так как в худшем случае потребуется линейная вычислительная сложность, чтобы найти место в которое вставляется элемент. Намного эффективнее приоритетная очередь будет работать при реализации в виде двоичной кучи. Двоичная куча - это структура данных на основе двоичного дерева, которая реализует такой тип данных как приоритетная очередь. Отличительной особенность двоичной кучи, как бинарного дерева, является то, что для каждого узла дерева выполняется условие, согласно которому его значение приоритетней (больше или меньше), чем значения его поддеревьев. То есть, в то время, как у бинарного дерева поиска значения одного поддерева меньше значения корня, а значения другого – больше, у бинарной кучи значения обоих поддеревьев либо больше, либо меньше значения корня. Другой же особенностью бинарное кучи как бинарного дерева является то, что бинарная куча должна быть так называемым совершенным двоичным деревом, то есть таким деревом, в котором все уровни, кроме последнего, полностью заполнены (то есть узлы, находящиеся на одинаковой высоте имеют поддеревья с обоих сторон), а узлы последнего уровня находятся как можно левее (то есть, если последний уровень не заполонён полностью, то все узлы этого уровня располагаются левее на сколько это возможно. Алгоритм добавления элемента в бинарную кучу следующий: добавляем элемент на последний уровень как можно левее, после чего если добавленный элемент приоритетней элемента, по отношению к которому он стал дочерним (то есть было нарушено правило кучи), то меняем родительский и дочерний(добавленный) элемент. Рекурсивно выполняем последнюю операцию до тех пор, пока родительский узел элемента не станет приоритетней текучего дочернего(добавленного) элемента. Таким образом добавляемый узел встает на свое место в куче. Алгоритм извлечения наиболее приоритетного элемента похож на алгоритм вставки, только обход идёт вниз, то есть если в алгоритме вставки элемент всплывал, то при удалении он тонет.

Алгоритм извлечения, следующий: в начале заменяем корень (наиболее приоритетный элемент) элементом который находится в конце очереди (элемент на последнем уровне, который располагается правее всех остальных). Сравниваем новый узел с одним из дочерних узлов. Если дочерний узел текущего узла приоритетней, то меняем их местами. Рекурсивно выполняем последнюю операцию до тех пор, пока текущий элемент не станет приоритетнее обоих своих дочерних элементов. Таким образом после удаления корня дерева восстанавливается свойство кучи. Эти операции достаточно эффективны за счет того, что бинарное куча – эта совершенное бинарное дерево и операции восстанавливают это свойство если оно было нарушено. Так операция удаления имеет логарифмическую сложность, а операция вставки в среднем – логарифмическую и в лучшем константную.

За счет того, что двоичная куча — это совершенное бинарное дерево, то данную структуру данных обычно реализуют в виде массива. Первый элемент массива обычно выступает корнем. Если i – индекс элемента в массиве, то элемент по индексу 2*i+1 его первый дочерний элемент, а 2*i+1 – второй. При этом индекс родительского элемента по отношению к элементу с индексом i находится по формуле (i - 1) div 2. Данный способ представить бинарную кучу помимо реализации приоритетной очереди (где массив - динамический) также используется для реализации алгоритма сортировки кучей (массив – статический, так как можно посчитать количество элементов, участвующих в сортировке).

Нет единого мнения почему свойство кучи получило именно такое название. Данное свойство так называют с тех пор, как в 1964 году Джон Вильям Джозеф Вильямс создал алгоритм сортировки кучей и двоичную кучу и дал всему этому такое название. Возможно, это свойство получило именно такое название потому, что результат сортировки кучей на первый взгляд, если не знать, как интерпретировать его, напоминает беспорядочный набор данных - кучу из данных. Стоит отметить, что термин куча также относится к способу

организации памяти, но при этом этот способ не имеет ничего общего с математическим свойством кучи Вильямса.

Бинарная куча намного эффективнее двоичных деревьев поиска в задачах, где требуется получить наибольший или наименьший по значению элемент, а также для ее реализации не требуется дополнительного места для хранения указателей. Но данная структура данных в отличии от двоичных деревьев поиска плохо справляется с поиском произвольного по значению элемента.

Курево

Как уже было сказано самобалансирующиеся деревья, как например упомянутые АВЛ-деревья или красно-черные деревья, не просты в реализации, так как требуется учитывать много особых случаев. Однако существует структура данных, которая хоть и не является самобалансирующимся деревом, но с очень высокой вероятностью вычислительная сложность операций этого дерева будет как и у приведённых видов самобалансирующихся деревьев - логарифмической. Для реализации такой структуры используется курево(дуча). Данный термин отражает суть структуры данных, которая сочетает в себе кучу и дерево поиска (ку-ча + де-рево или д-ерево + ку-ча). Отличительной особенностью данного двоичного дерева поиска является то, что каждый узел двоичного дерева поиска содержит информацию о так называемом приоритете узла и приоритеты узлов подчиняются свойству математической кучи. Значения приоритетов должны быть случайными значениями из большого диапазона. Дерево с высокой вероятностью будет сбалансированным за счет того, что даже если вставляемые или удаляемые значения в бинарном дереве поиска будут через некоторые промежутки упорядочены (причина, по которой двоичное дерево поиска скашивается), то если значения приоритетов случайны (никак не упорядочены), то чтобы сохранилось свойство кучи для приоритетов, дерево будет реорганизовано для восстановления свойства кучи, что приводит к псевдо-балансировке дерева. Чтобы проиллюстрировать это рассмотрим что

может произойти с куревом целых чисел, если вставляемые числа — это числа от 1 до 5. Если просто вставлять значения в дерева по алгоритму формирования двоичного дерева поиска, то высота дерева станет равной 5, хотя совершенная версия этого дерева имеет высоту 3. Поэтому каждому вставляемому значению зададим случайный приоритет:

1 – 123, 2 - 321, 3 - 999, 4 - 300 , 5 – 222;

Теперь если вставлять последовательно эти значения в курево соблюдая для ключей правила двоичного дерева поиска, а для приоритетов - правило двоичной кучи, то получится дерево высотой 3. Стоит отметить, что в данном случае получилось наилучшее по высоте двоичное дерево поиска, хотя если бы случайные значения были бы другими, могло бы получится дерево и другой высоты, начиная от высоты соответствующего совершенного дерева и заканчивая максимальной возможной высотой для данного дерева. Однако в любом случае с достаточно большой вероятностью (которая зависит от степени случайности значений приоритетов) получится сбалансированное дерево, то есть дерево, разница высот которого отличается не более, чем на единицу. Важной особенностью курева, которая находит практическое применение — это то, что пары ключ-приоритет задают только одно возможное, уникальное дерево поиска (дерево только одной определенной формы). То есть, например, по куреву по причине этого свойства не может быть восстановлена последовательность в которой элементы добавлялись в узел. Это может быть использовано в задачах, в которых требуется скрыть историю формирования двоичного дерева поиска. Также существуют эффективные алгоритмы обмена поддеревьями между куревами. Тема реализации и особенностей курева выходит за пределы данной курсовой работы и подробнее с куревом можно ознакомиться в литературе на соответствующую тему.

Объема курсовой работы недостаточно, чтобы охватить тему бинарных деревьев и их видов, поэтому далее будет рассмотрен только один из видов бинарного дерева – двоичное дерево поиска

1.2. Двоичное дерево поиска

Одним из самых эффективных алгоритмов поиска является классический алгоритм бинарного поиска. Этот алгоритм заключается в следующем: чтобы найти значения в отсортированном по возрастанию массиве, в качестве текущего элемента возьмем элемент по середине массива; Если искомое значение меньше текущего элемента, то возьмем в качестве текущего элемент, который находится по середине отрезка от начала массива и до текущего элемента; если больше, то в качестве текущего элемента возьмем элемент по середине отрезка от текущего элемента и до конца массива. Последняя операция повторяется до тех пор, пока текущий элемент не станет искомым элементом, либо пока не останется элементов, которые не были текущим.

Как уже было сказано, алгоритм бинарного поиска работает только с отсортированным массивом. Но часто приходится работать с динамическими структурами данных, то есть добавлять в массив и удалять из него значения. Это приводит к тому, что если хранить данные в таком массиве, то чтобы алгоритм бинарного поиска мог работать с данным массивом, то при каждом добавлении элемента в массив или удаления элемента из массива его необходимо отсортировать. Чтобы избежать вынужденной сортировки данных структуры при включении и исключении данных, но при этом иметь возможность использовать алгоритм бинарного поиска, можно использовать двоичное дерево поиска, определение которого было дано ранее в разделе 1.1.

В отсортированном массиве поиск элемента в худшем случае имеет логарифмическую вычислительную сложность, а вычислительная сложность вставки и удаления элемента в зависимости от используемого алгоритма сортировки может колебаться, но в лучшем случае (алгоритм быстрой сортировки Хоара) имеет сложность O(n log n).

В двоичном дереве поиск, вставка и удаление элемента в среднем имеют логарифмическую вычислительную сложность, а в худшем – линейную.

Таким образом, можно сделать вывод, что для хранения динамических данных, которые имеют много элементов, двоичное дерево поиска будет эффективнее, чем отсортированный массив.

Но недостатком двоичного дерева поиска является то что средняя вычислительная сложность зависит от сбалансированности дерева, то есть от того в каком порядке добавлялись элементы. Иллюстрацией этого утверждения будет двоичное дерево поиска все n элементов которого добавлялись в него по возрастанию. В таком случае все узлы двоичного дерева поиска будут иметь только правого потомка, и фактически получится связанный список из n элементов. Тогда если потребуется получить n элемент который был добавлен в дерево последним (то есть в данном случае самый большой по значению элемент), то потребуется перебрать начиная с корня все n элементов двоичного дерева поиска. То есть, получение этого n-ого элемента будет иметь линейную вычислительную сложность – худшую для двоичного дерева поиска. Это был пример так называемого скошенного дерева (причем скошенного на столько, на сколько это возможно).

Данную проблему решают самобалансирующиеся виды двоичных деревьев поиска, элементы в которые добавляются таким образом, чтобы бинарное дерево не становилось скошенным. Примерами таких бинарных деревьев являются крано-черные деревья и АВЛ-деревья. Краткие характеристики данных видов бинарного дерева приведены ранее в разделе 1.1.

Перейдем к рассмотрению операций, выполняемых в двоичном дереве поиска. Существует множество различных операций, которые можно совершать над бинарным деревом, но ограничимся рассмотрением некоторых наиболее востребованных основных операций.

Операция поиска элемента – операция которая принимает ключ узла и возвращает ссылку на узел, которая имеет равный ключ. Существует различные способы реализовать данную операцию, рассмотрим классическую и простейший алгоритм реализации. Для того, чтобы найти узел дерева с определённым ключом, необходимо сравнить заданный ключ с ключом корня

текущего поддерева. Если заданный ключ больше - то рекурсивно искать ключ в правом поддереве, если меньше - то в левом. Выход из рекурсии осуществляется в том случае, если текущий ключ равен ключу корня текущего поддерева или есть этот корень не имеет потомков. Как уже было упомянуто ранее, вычислительная сложность операции поиска в двоичном дереве поиска сильно зависит от степени его сбалансированности, то есть от того на сколько оно скошено.

Операция добавления элемента – алгоритм добавления элемента в бинарное дерева похож на алгоритм поиска элемента. Его также можно реализовать с помощью рекурсии, но приведём нерекурсивную версию алгоритма на псевдокоде(данный алгоритм может быть упрощен, если есть возможность напрямую работать с памятью компьютера, то есть, если есть например такой тип данных, как указатели и операции для работы с ними):

```
Пусть добавляемыйЭлемент – элемент который добавляется в дерево;
Пусть текущУзл – корневой узел дерева;
Пока(
добавляемыйЭлемент.ключ меньше текущийУзел.ключ
И
текущУзл имеет поддерево слева
ИЛИ
добавляемыйЭлемент.ключ больше текущийУзел.ключ
И
текущУзл имеет поддерево справа
)
    Если(добавляемыйЭлемент.ключ равен текущийУзел.ключ) то
        Завершить операцию;
    Конец Если;
    Если(добавляемыйЭлемент.ключ больше текущийУзел.ключ) то
        Пусть текущУзл это текущийУзел.правоеПоддерево;
    Конец Если;
```

```
        Если(добавляемыйЭлемент.ключ меньше текущийУзел.ключ) то
            Пусть текущУзл это текущийУзел.левоеПоддерево;
        Конец Если;
    Конец Пока;
    Если(
    добавляемыйЭлемент.ключ меньше текущийУзел.ключ
    И
    текущУзл НЕ имеет поддерево слева
    ) то
        текущУзл.создатьЛевоеПоддеревоСКорнем(добавляемыйЭлемент);
    Иначе Если(
    добавляемыйЭлемент.ключ больше текущийУзел.ключ
    И
    текущУзл НЕ имеет поддерево справа
    ) то
        текущУзл.создатьПравоеПоддеревоСКорнем(добавляемыйЭлемент);
    Конец Если;
```

Операция удаления узла – также, как и в случае операций добавления и удаления элементов в бинарное дерево поиска, существует множество различных способов реализовать операцию удаления элемента дерева. Основная сложность этой операции заключается в том, чтобы сохранить упорядоченность бинарного дерева после удаления произвольного узла. Томас Н. Хиббард в 1962 году предложил такой алгоритм, который также изменяет высоту (наибольшее число улов между корнем и узлом не имеющего потомков) поддеревьев не более, чем на один узел, что позволяет сохранить сбалансированность дерева. Суть предложенного им алгоритма заключалась в том, чтобы удалять узлы, имеющие поддеревья, заменяя их так называемым преемником (successor), то есть заменять значение удаляемого узла значением узла-преемника, после чего рекурсивно удалять сам узел-преемник. Алгоритм

рассматривает следующие три случая. Первый случай - если удаляемый узел не имеет поддеревьев. В таком случае просто удаляем узел. Второй случай - если удаляемый узел имеет поддерево с одной из сторон. В таком случае делаем так, чтобы корень поддерева с этой стороны (с левой или справой) удаляемого узла стал прямым предком (дочерним узлом) узла, родительского по отношению к удаляемому узлу, при чем с той же стороны, с которой находится удаляемый узел по отношению к этому родителю. Третий случай – если удаляемый узел имеет поддеревья с обоих сторон. В таком случае можно найти минимальный по ключевому признаку узел в правом по отношению к удаляемому узлу, поддереве. Эта операция сводится к тому, чтобы перейти в данном случае к правому поддереву удаляемого узла и далее переходить в данном случае к левым поддеревьям до тех пор, пока не будет определен узел не имеющий поддеревьев. Этот минимальный по ключевому признаку узел правого поддерева становится так называемым преемником удаляемого узла и его значения замещают значения удаляемого узла, после чего узел-преемник рекурсивно удаляется. Двоичное дерево поиска при этом остается упорядоченным, так как между ключом удаляемого узла и ключом узла-преемника нет ключевых признаков, узлы которых существуют в данном бинарном дереве. Иными словами узел-преемник - это такой узел одного из поддеревьев, который не имеет поддеревьев (это необходимо, чтобы те не перекрывали поддеревья удаляемого узла) и при этом его ключевой признак больше ключевого признака корня левого поддерева удаляемого узла и меньше ключевого признака корня правого поддерева удаляемого узла(это необходимо чтобы сохранялась упорядоченность двоичного дерева поиска, то есть оставалось верным правило, согласно которому любой узел по ключу должен быть больше дочернего узла с левой стороны и меньше дочернего узла с правой стороны)

Рассмотрим некоторые из методов обхода бинарного дерева.

Существует множество различных способов обойти дерево, но чтобы не превысить максимальный допустимый объем курсовой работы, ограничимся только обходом в глубину и в ширину.

Отличительной особенностью обхода в ширину является то, что дерево рекурсивно обходится сначала по левым потомкам пока не будет достигнут узел, не имеющий левых потомков и затем рекурсивно по правым потомкам, пока не будет достигнут узел, не имеющий правых потомков. Или наоборот – сначала по правым и затем по левым. Обходы в глубину по мимо того - по потомкам с какой стороны сначала осуществляется обход, классифицируется также по тому, в какой момент будет извлечен сам узел: до перехода к потомкам, после обхода потомков с одной из сторон или после обхода потомков с обоих сторон.

В некоторых источниках для уточнения вида обхода в глубину используются следующие обозначения: L – рекурсивный обход левого поддерева, R - рекурсивный обход правого поддерева, N – извлечение узла.

Наиболее актуальными обходами в глубину являются:

Центрированный обход – LNR, если узлы бинарного дерева— это числа, а дерево упорядоченно по возрастанию, то обход узлов произойдет по возрастанию их ключевых признаков, то есть чисел.

Обратный центрированному обход – RNL, если узлы двоичного дерева— это числа, а дерево упорядоченно по возрастанию, то обход узлов произойдет по убыванию их ключевых признаков, то есть чисел.

Прямой обход – NLR, данный обход используется для топологического обхода узлов дерева по возрастанию. Суть топологической сортировки заключается в том, чтобы представить такой линейный порядок узлов однонаправленного нециклического графа, в которой узлы упорядочены по возрастанию числа всех узлов из которых можно попасть в данный узел. Это широко используется в задачах, в которых, например, требуется выполнить действия, которые могут зависеть от других действий только после того, как

все действия, от которых зависит текущее, будут выполнены. Примером такой задачи является установление правильного порядка установки программ пакетным менеджером, так как некоторые программы могут быть неработоспособными без программ, от которых они зависят. Аналогичным примером будет задача нахождения порядка сборки исходного текста программы компилятором, так как функции в одном файле программы могут содержать вызовы функций из других файлов и получается, что для их работы требуется сначала вставить функции, от которых они зависят. Другим примером топологического порядка является допустимый порядок прохождения курсов в Европейских Высших учебных заведениях, в которых студенты самостоятельно определяют курсы, которые они будут проходить, но порядок их прохождения обязан быть таким, чтобы к началу курса у студента были все необходимые для его освоения знания, которые студент в свою очередь может получить пройдя другие курсы, указанные в требованиях к курсу. И таким образом достигается то, что, например, студент не станет проходить курс трехмерной графики, до курса аналитической геометрии, освоение которого в свою очередь требует прохождения курса линейной алгебры.

Обратный обход – LRN, обход обратный прямому (NLR) обходу. Данный обход используется для топологического обхода узлов дерева по убыванию. То есть обход узлов дерева по убыванию числа всех узлов из которых можно попасть в данный узел (бинарное дерево при этом считается однонаправленным, то есть игнорируется направления по ребрам ведущие от потомка к предку). Данный обход используется в алгоритмах балансировки дерева. Также как будет продемонстрированно ниже данный обход удобно использовать для удаления дерева, так как чтобы удалить дерево не переставляя узлы необходимо удалять узлы, не имеющие потомков.

Отличительной особенностью обхода в ширину является то, что уровни (узлы имеющие одинаковое число узлов, лежащих между ними и корнем)

обходятся последовательно один за другим. Рассмотрим один их вариантов обхода в ширину. Данный алгоритм, в отличии от приведенных ранее алгоритмов обхода в глубину, является не рекурсивным и для его реализации необходимо использовать такую структуру данных как очередь. Приведём неформальное вариант описания на псевдокоде алгоритма обхода двоичного дерева в глубину:

```
Пусть очрдУзлв – очередь для хранения узлов дерева;
Пусть очрдКлчй – очередь узлов для извлечения;
корень – узел с которого начинает обход;
очрдУзлв.добавить(корень);
Пока(есть элементы в очрдУзлв)
    Пусть ткщйУзл = очрдУзлв.извлечь();
    очрдКлчй.добавить(ткщйУзл);
    Если (ткщйУзл имеет левое поддерево) то
        очрдУзлв.добавить(ткщйУзл.левоеПоддерево);
    Конец Если;


    Если (ткщйУзл имеет правое поддерево) то
        очрдУзлв.добавить(ткщйУзл.правоеПоддерево);
    Конец Если;
Конец Пока;
Пока(есть элементы в очрдКлчй)
    очрдКлчй.извлечь(); // Обход узлов в ширину
Конец Пока;
```

Обход в ширину находит применение при решении задач, связанных с более сложными графами чем бинарные деревья (циклическими, имеющие более двух ребер, исходящих из одного узла и.т.п.), но данный алгоритм может быть полезен и для двоичного дерева поиска, например, для определения узлов,

лежащих на пути между двумя узлами. Реализация данной задачи будет продемонстрирована далее.

Обход в глубину можно также использовать для удобного численного представления бинарного дерева (то есть представления бинарного дерева в форме последовательности чисел). Это можно использовать для записи бинарного дерева в файл, который в свою очередь фактически является последовательностью чисел. Запись в файл двоичного дерева существенно отличается, например, от записи массива(за исключением случая когда для реализации бинарного дерева используется курево, так как оно однозначно представляется линейной последовательностью двух ключей(ключей кучи и дерева)), так как не последовательные(ассоциативные) контейнеры реализующие соответствующие структуры данных, такие как например двоичное дерево, обычно (например, в C++) реализованы с использованием указателей, хранящих адреса, разбросанных по всей памяти блоков со значениями узлов. Так как информация о связях (в данном случае ребрах дерева) хранится в форме указателя в одном узле, который хранит адрес блока памяти с узлом, с которым связан текущий, то эта информация теряется при выгрузке бинарного представления такого контейнера в файл и последующей выгрузке из него. Это происходит потому что бинарный ввод и вывод подразумевает ввод и вывод информации о контейнеров в точно таком же виде что и их представление в памяти компьютера. И таким образом значения указателей (и соответственно информация о ребрах дерева) не будет корректной при чтении контейнера из файла в другую область памяти. Поэтому чтобы записать бинарное дерево в файл нужен способ преобразования двоичного дерева в некую линейную форму (которую можно записать в файл) и однозначного обратного преобразования из неё в двоичное дерево. Примером такого способа является следующий метод.

Любое двоичное дерево поиска может быть однозначен описано результатами двух разных обходов – например прямого и центрированного. Пусть $[x_1,\ldots,x_n]$ - результат прямого обхода, а $[z_1,\ldots,z_n]$ – центрированного.

x_1 – корень текущего поддерева.

Пусть k – такой индекс, что $z_k=x_1$. Тогда $[z_1,\ldots,z_{k-1}]$ – центрированный обход левого поддерева, а $[z_{k+1},\ldots,z_n]$ – центрированный обход правого поддерева. Также $[x_2,\ldots,x_k]$ – прямой обход левого поддерева, а $[x_{k+1},\ldots,x_n]$ - прямой обход правого поддерева. Рекурсивно выполнив эту операцию для левого поддерева и правого поддерева получим исходной двоичное дерево поиска.

Стоит отметить, что данный способ является не единственным способом решить поставленную задачу. Например, из преобразуемого дерева можно сделать курево, генерируя приоритеты таким образом, чтобы для приоритетов узлов выполнялось правило кучи. Так как пары ключ-значение уникальным образом задают двоичное дерево, то чтобы преобразовать дерево поиска достаточно сохранить эти пары. После чего из них можно восстановить исходное двоичное дерево поиска по алгоритму формирования курева. Также данную задачу можно решить если вместо NULL использовать заполнители. В таком случае решением данной задачи является результат обхода в ширину.

Вариант реализации на C++ представленных операций, вышеприведённого метода и ранее упомянутой задачи определения узлов, лежащих на пути между двумя узлами продемонстрирована далее.

Глава 2. Проектирование и разработка программы

2.1. Состав и назначение отдельных компонентов программы

Программа написана на C++ и представляет из себя консольное приложение для операционной системы Microsoft Windows. Для реализации

программы использовались только стандартные библиотеки языка, а также интерфейс программирования приложений - Windows API. Как уже было упомянуто ранее двоичное дерево поиска может быть использовано для реализации базы данных телефонных абонентов. Поэтому для более ясного представления практического применения элементов и операций двоичного дерева поиска программа имитирует такую базу данных. Ключевым признаком является номер мобильного телефона, а значением набор данных из того же номера, имени его владельца и информации о тарифе по которому обслуживается данный номер.

При запуске программы отображается заставка, которая содержит информацию с титульного листа данной курсовой работы. После заставки пользователю предлагается меню действий с двоичным деревом поиска, имитирующего базу теле-абонентов. Данное меню предлагает следующие действия:

1 - создать новое бинарное дерево (базу абонентов);

2 - обход бинарного дерева с выдачей на экран содержимого информационных полей (вывод всех абонентов базы);

3 - включить элемент в бинарное дерево (добавить абонента);

4 - удаление заданного узла из дерева (удаление абонента);

5 - удаление дерева (базы абонентов) с освобождением памяти;

6 - определить количество узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками;

7 - открыть бинарное дерево (базу абонентов) из файла;

8 - сохранить текущее состояние бинарного дерева (базы абонентов) в файл;

9 - завершить программу;

Выбор какого-либо действия выполняется нажатием на соответствующую клавишу цифровой клавиатуры, каждая цифра которого, за исключением цифры 9(действию завершающую программу соответствует клавиша с цифрой 0), соответствует определенному действию. Реакция

программы на выбранную пользователем задачу из меню действий реализуется оператором switch. При выборе действий, которые требуют ввода дополнительной информации (добавление/удаление элемента; определения количества узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками; сохранение/открытие базы) осуществляется переход в соответствующие меню, которое содержит форму для ввода необходимых для выполнения действия данных. Формы не позволяют пользователю ввести некорректные данные. Примерами некорректных данных служат пустой номер или попытка определения количества узлов между узлами, один из которых не принадлежит к обрабатываемому дереву. При попытке ввести некорректные данные рядом с соответствующими полями отображается причина по которым данные не могут быть обработаны. Также в этих меню с формами есть возможность вернуться в меню выбора действия не вводя данные. Меню с формами реализованы в виде отдельных функций, которые вызываются из оператора switch.

Само же двоичное дерево поиска реализовано в виде C++ структуры BinSearchTree, которая содержит методы для работы с двоичным деревом поиска и полями с данными, которые относятся ко всей структуре (указатель на корень дерева; указатель на логическую функцию, которая определяет операцию сравнения для ключевых признаков узлов дерева. Также в форме вложенной структуры в BinSearchTree определенна структура BinSearchTree::Node. BinSearchTree::Node используется для представления узлов двоичного дерева поиска. Эта содержит поля данных: ключевого признака; значения узла; указателя на левый дочерний узел; указателя на правый дочерний узел;

В последующей таблице представлены все методы структуры BinSearchTree.

Таблица 1

Спецификация методов

Название метода и его назначение	**Описание вызова метода другими методами**	**Описание входных параметров**	**Описание выходных параметров**	**Описание используемых файлов**	**Список методов, вызываемых данным методом**
find – находит узел дерева по ключу	Метод вызывается из методов get, insert, erase	Ключевой признак	узел с данным ключом	-	-
get – получает значение узла по ключу	-	Ключевой признак	Значение узла с данным ключом. Если узла с данным ключом нет в дереве, то возвращает значение по умолчанию (такое значение, которое принимает значение узла при объявлении в глобальной области видимости)	-	find
insert – создает новый узел в дереве с заданным ключом и значением	-	-	Ключевой признак узла и его значение	-	find
erase – удаляет узел из дерева по ключу	Метод рекурсивно вызывается самим методом erase; Также метод вызывается методом clear	Ключевой признак	-	-	Метод рекурсивно вызывается самим методом erase; Также метод вызывает метод find
clear – уничтожает дерево с освобождением памяти	-	-	-	-	traverseVals erase
traverseVals – осуществляет обход узлов дерева	Метод рекурсивно вызывается самим методом traverseVals	Указатель на узел с которого начинает обход; Вид обхода; Указатель на функцию в которую передаются	-	cstdarg	Метод рекурсивно вызывается самим методом traverseVals

	Также метод вызывается методом clear	значения узлов; количество входных параметров функции, в которую передаются значения узлов; параметры функции в которую передаются значения узлов			

Реализация представленных методов продемонстрирована далее.

2.2. Реализации задач

Создание двоичного дерева поиска

Для создания двоичного дерева поиска необходимо определить операцию сравнения для ключей дерева в виде функции принимающей ключ с для которого выполняется сравнение и ключ узла с которым сравнивается значение и возвращающей болевое значение. Истинна соответствует правому дочернему узлу, ложь - левому

```
// Функция определяющая операцию сравнения для ключей
bool greaterInt(KeyType a, KeyType b)
{
    return (a > b);
}
```

Само же дерево создается с помощью операции new, которая создает дерево с заданным узлом и операцией сравнения.

```
subsBase = new BinSearchTree{ NULL, greaterInt};
```

(локальная переменная) BinSearchTree *subsBase

Поиск в Интернете

Добавление узла в дерева с заданным ключом и значением реализовано в виде метода insert

```
void insert(KeyType key, ValueType val)
{
    // Ключу соответсятует только одно значение,
    // если данный ключ уже есть, то завершаем функцию.
    Node* keyIsValNode = find(key);
    if (keyIsValNode)
        return;
    if (mainRoot) // Если в дереве есть узлы(указатель на корень не нулевой)
    {
        // Начинаем обход с корня, для этого устанавливаем на него указатель
        Node* currNode = mainRoot;
        // Переходим в зависимости от значения ключа создаваемого узла и ключа текущего узла
        // на правое или левое поддерево
        for (Node* nextNode = (sorter(key, currNode->key)) ? currNode->right : currNode->left;
            nextNode;
            nextNode = (sorter(key, currNode->key)) ? currNode->right : currNode->left)
            currNode = nextNode;
        // Найдя родительский узел для создаваемого узла,
        // в зависимости от значений ключей,
        // создаем дочерней узел с права или слева
        if (sorter(key, currNode->key))
            currNode->right = new Node{ key, NULL, NULL, currNode, val };
        else
            currNode->left = new Node{ key, NULL, NULL, currNode, val };
    }
    // Если дерево пустое, то корнем становится создаваемый узел
    else
        mainRoot = new Node{ key, NULL, NULL, NULL, val };
}
```

Обход двоичного дерева поиска, включение элемента в бинарное дерево (согласно алгоритму формирования дерева).

Поиск узла в дереве реализован в виде метода find

```
Node* find(KeyType val)
{
    // Начинаем поиск с корня дерева. Устанавливаем указатель на корень
    Node* currNode = mainRoot;
    // Пока текущий узел существует(указатель не нулевой) и значение текущего узла не равно искомому.
    while (currNode and !isKeyTypeValsEqual(currNode->key, val))
        // Переставляем указатель на правый дочерний узел если значение больше текущего и на левый если меньше
        currNode = (sorter(val, currNode->key)) ? currNode->right : currNode->left;
    // Возвращается NULL, если значение небыло найденно(так как перешли на указатель дочернего узла, который равен нулю)
    // и указатель на узел со значением если он существует
    return currNode;
}
```

Поиск значения, которое соответствует ключу реализован в виде метода get

```
ValueType get(KeyType key)
{
    // Получаем указатель на узел со значением или ноль если такого узла нет
    Node* currNode = find(key);
    // Если указатель не нулевой, то возращаем значение узла.
    // Если нулевой, то значение по умолчанию
    return (currNode) ? currNode->value : ValueType();
}
```

Удаление заданного узла из дерева (без поддерева)

Удаление узла из дерева по ключу реализовано в виде метода erase

```
void erase(KeyType val)
{
    // Если узла с таким значением нет, то завершаем функцию
    Node* currNode = find(val);
    if (!currNode)
        return;
    // Если удаляемый узел имеет дочерние узлы с обоих сторон
    if (currNode->left && currNode->right)
    {
        // Находим минимальный по ключчу узел правого поддерева
        Node* rightTreeMinNode = currNode->right;
        while (rightTreeMinNode->left)
            rightTreeMinNode = rightTreeMinNode->left;
        // Меняем значение удаляемого узла значением найденного узла
        KeyType rtmnKey = rightTreeMinNode->key;
        // Рекурсивно удаляем найденный узел
        erase(rightTreeMinNode->key);
        currNode->key = rtmnKey;
    }
    // Если удаляемый узел имеет один дочерний узел или не имеет потомков
    else
    {
        // Если у удаляемого узла есть левый дочерний узелб
        // то он становится узлом-преемником,
        // иначе узел-преемник - правый узел
        // или, в том случае если удаляемый узел не имеет потомков - нулевой указатель
        Node* successorNode = (currNode->left) ? currNode->left : currNode->right;
        // В том случае если указатель на узел-преемник не нулевой, то
        // то меняем значение указателя дочернего узла на родительский узел,
        // который может быть нулевым, если удаляемый узел - корень
        successorNode && (successorNode->parent = currNode->parent);
        // Если удаляемый узел не корень
        if (currNode->parent)
        {
            // Меняем указатель родительского узла на дочерний узел
            // на указатель узла-преемника
            if (currNode == currNode->parent->left)
                currNode->parent->left = successorNode;
            else
                currNode->parent->right = successorNode;
        }
        // Если удаляемый узел-корень
        else
            // то корнем становится его дочерний узел или нулевой указатель
            mainRoot = successorNode;
        // удаляем удаляемый узел
        delete currNode;
    }
}
```

Удаление дерева с освобождением памяти

Удаление дерева реализовано в виде метода clear

```
void clear(Node*& currNode)
{
    // Оптимальным алгоритмом удаления дерева
    // является такой алгоритм, который в процессе
    // выполнения не требует перестраивания дерева.
    // Для этого нужно удалять только те узлы которые не имею потомков.
    // Для этого можно удалять узлы в обратном топологическом порядке
    // обходя узлы методом обратного(LRN) обхода.

    // Условие выхода из рекурсии
    if (!currNode)
        return;

    // L - рекурсивное удаление левых поддеревьев
    clear(currNode->left);
    // R - рекурсивное удаление правых поддеревьев
    clear(currNode->right);
    // N - удаление самомго узла
    delete currNode;
}
```

Определение количество узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками

Данная задача реализована в виде метода scoutPath

```
int scoutPath(BinSearchTree::Node* currNode, KeyType key, int pathLength, SinglyLinkedList_queue& addOfVisitedNodes)
{
    // Если текущий узел не существует(указатель на него нулевой),
    // то вернуть отрицательный результат, так как
    // данная ветвь рекурсии не смогла обнаружить узел к которому ищется путь
    if (!currNode)
        return -1;
    // Если текущий узел уже был посещен,
    // (его адрес есть в контейнере уже посещенных узлов)
    // то вернуть отрицательный результат, так как
    // данная ветвь рекурсии или уже была обработана или обработается другой ветвью
    for (SinglyLinkedList_queue::Node* iter = addOfVisitedNodes.head; iter != NULL; iter = iter->next)
    {
        //std::cout << "<<< " << iter->data->key << " >>>" << std::endl;
        if (iter->data == currNode)
            return -1;
    }
    // Добавить адресс текущего узла в контейнер,
    /// который сожержит адреса уже посещенных узлов
    addOfVisitedNodes.insert(currNode);
    // Если текущий узел - узел назначения,
    // то вернуть колличество узлов лежищих между ним
    // и начальным узлом
    if (currNode->key == key)
        return pathLength;
    // Возвращаемое значение
    int returnVal;
    // Рекурсивно искать в родительском узле
    returnVal = scoutPath(currNode->parent, key, pathLength + 1, addOfVisitedNodes);
    // Если искомый узел не был найден(результат отрицателен), то искать в левом дочернем узле
    if(returnVal == -1)
        returnVal = scoutPath(currNode->left, key, pathLength + 1, addOfVisitedNodes);
    // Если искомый узел не был найден(результат отрицателен), то искать в правом дочернем узле
    if (returnVal == -1)
        returnVal = scoutPath(currNode->right, key, pathLength + 1, addOfVisitedNodes);
    // Вернуть найденный результат
    return returnVal;
}
int scoutPath(BinSearchTree* bst, KeyType keyA, KeyType keyB)
{
    // Если один из узлов отсутствует в дереве, то возвращаем отрицательный результат
    BinSearchTree::Node* nodeA = bst->find(keyA);
    if (!nodeA || !bst->find(keyB))
        return -1;
    // В качесте контейнера для хранения адрессов уже посещенных узлов,
    // лучше было бы использовать все тот-же BinSearchTree,
    // но чтобы реализовать хранения ключей и значений произвольного типа нужны шаблоны
    int nodeCount = 0;
    bst->traverseVals(countNodes, 1, &nodeCount);
    SinglyLinkedList_queue addOfVisitedNodes{NULL, NULL};
    // Устанавливает изначальные значения переменных
    return scoutPath(nodeA, keyB, -1, addOfVisitedNodes);
}
```

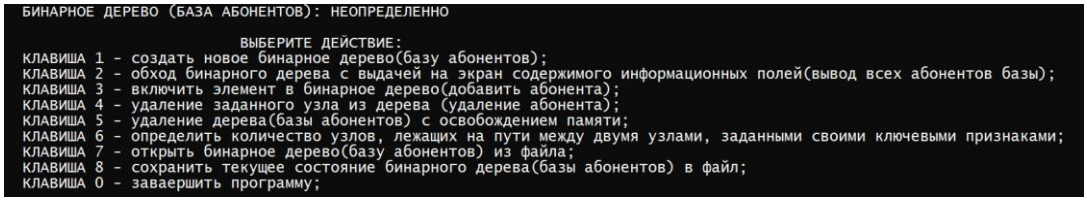
Преобразование прямого и центрированного обхода в исходное дерево.

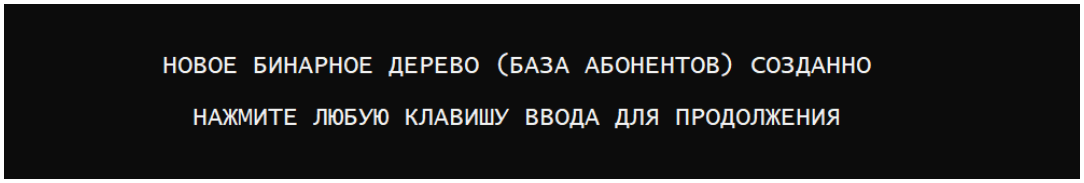
```
void recon_BST_from_NLR_LNR_seq
(
    BinSearchTree::Node*& rootNodeAdd,
    KeyType* nlr,
    int nlrLength,
    int nlrRootIndex,
    KeyType* lnr,
    int lnrLength,
    int lnrBeginIndex,
    ValueType* vals
)
{
    if (lnrLength < 1)
    {
        rootNodeAdd = NULL;
        return;
    }
    rootNodeAdd->key = nlr[nlrRootIndex];
    rootNodeAdd->value = vals[nlrRootIndex];
    rootNodeAdd->left = new BinSearchTree::Node;
    rootNodeAdd->left->parent = rootNodeAdd;
    rootNodeAdd->right = new BinSearchTree::Node;
    rootNodeAdd->right->parent = rootNodeAdd;

    int lnrRootIndex;
    for (lnrRootIndex = 0; lnr[lnrRootIndex] != nlr[nlrRootIndex]; lnrRootIndex++);

    recon_BST_from_NLR_LNR_seq
    (
        rootNodeAdd->left,
        nlr,
        (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex, nlrRootIndex + 1,
        lnr,
        (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex,
        lnrBeginIndex,
        vals
    );
    recon_BST_from_NLR_LNR_seq
    (
        rootNodeAdd->right,
        nlr,
        nlrLength - (( lnrRootIndex+(nlrRootIndex-lnrBeginIndex) ) - nlrRootIndex) - 1,
        (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) + 1,
        lnr,
        nlrLength - ((lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex) - 1,
        lnrRootIndex + 1,
        vals);
}
```

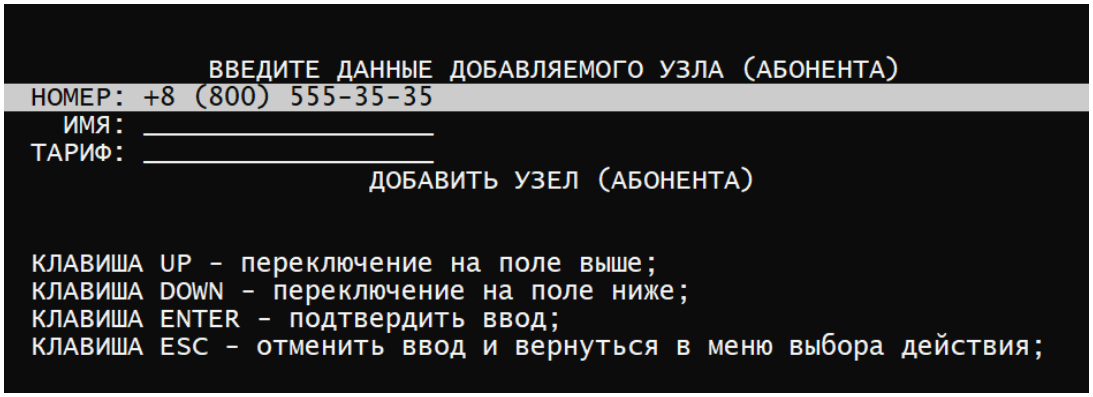
Полный текст программы приведён в приложении.

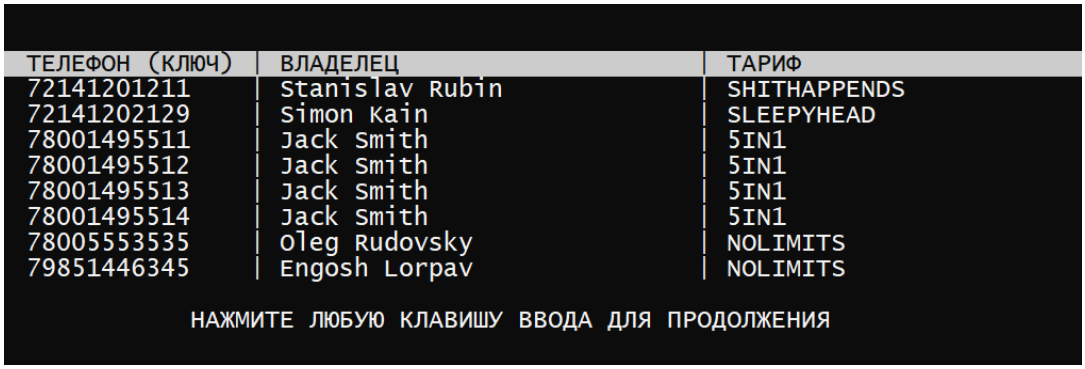
2.3. Описание машинного эксперимента

В качестве примера работы программы выполним ввод нескольких элементов, выведем их на экран, удалим элемент с конца и вставим элемент в начало списка, опять выведем на экран и выйдем из программы.









Результаты проведенного машинного эксперимента свидетельствуют о правильности выбранных методов и средств для решения поставленной задачи.

Заключение

Данная курсовая работа знакомит с понятием бинарного дерева, а также позволяет приобрести теоретические знания и практические навыки, связанные с такой структурой данных как двоичное дерево поиска. Были рассмотрены основные свойства бинарных деревьев, их виды, а также область их применения.

Как пример работы с двоичным деревом поиска была разработана программа, в которой реализовывает ряд задач:
создание двоичного дерева поиска, обход двоичного дерева поиска;
включение элемента в бинарное дерево
(согласно алгоритму формирования дерева);
удаление заданного узла из дерева (без поддерева);
удаление дерева с освобождением памяти;
определения количество узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками;
представление дерева в виде последовательности чисел вместе с обратным преобразованием этой последовательности чисел в прежнее дерево.

Исследование и программная реализация основных действий с двоичным деревом поиска позволили систематизировать и расширить теоретические знания в данной области и применить их на практике.

Список использованных источников

1. Robert Lafore. Object-Oriented Programming in C++, Fourth Edition, ISBN 0-672-32308-7
2. Aditya Y. Bhargava. Grokking Algorithms. An illustrated guide for programmers and other curious people. ISBN 9781617292231.
3. Robert Sedgewick, Kevin Wayne: Algorithms Fourth Edition. Pearson Education, ISBN 978-0-321-57351-3
4. Rowan Garnier, John Taylor. Discrete Mathematics: Proofs, Structures and Applications, Third Edition, ISBN 978-1-4398-1280-8
5. https://www.youtube.com/playlist?list=PLhQjrBD2T381L3iZyDTxRwOBuUt6m1FnW
6. https://www.cplusplus.com/
7. https://www.youtube.com/watch?v=6podLUYinH8
8. https://habr.com/ru/post/100953/
9. https://cs.stackexchange.com/questions/439/which-combinations-of-pre-post-and-in-order-sequentialisation-are-unique

Приложение

```
#include <Windows.h>
#include <iostream>
#include <stdarg.h>
#include <string>
#include <iomanip>
#include <ctime>
#include <conio.h>
#include <fstream>


// Реализация FIFO.
// Очередь нужна для итеративной реализации обхода бинарного дерева в ширину


// Реализация бинарного дерева поиска:
struct TeleSubscriber
{
	unsigned long long number;
	std::string owner;
	std::string tariff;
};
```

```
typedef unsigned long long KeyType;
typedef TeleSubscriber ValueType;


bool isKeyTypeValsEqual(KeyType a, KeyType b)
{
	return (a == b);
}
struct BinSearchTree
{
	struct Node
	{
		KeyType key;
		Node* left, * right, * parent;
		ValueType value;
		//--------------------------------
	};
	enum TraversalType
	{
		// В глубину:
		LNR_REQ = 123, // Центрированный(In-order) обход.
```
Рекурсивная реализация

RNL_REQ = 321, // Обратный центрированный(Reverse in-order) обход. Рекурсивная реализация

NLR_REQ = 213, // Прямой(Pre-order) обход. Рекурсивная реализация

LRN_REQ = 132, // Обратный(Post-order) обход. Рекурсивная реализация

// В ширину

BFS_QUE = 231 // Обход в ширину слева на право. Итеративная реализация с помощью очереди

};

Node* mainRoot;

bool (*sorter)(KeyType a, KeyType b);

//--------------------------

Node* find(KeyType val)

{

// Начинаем поиск с корня дерева. Устанавливаем указатель на корень

Node* currNode = mainRoot;

// Пока текущий узел существует(указатель не нулевой) и значение текущего узла не равно искомому.

while (currNode and !isKeyTypeValsEqual(currNode->key, val))

```
            // Переставляем указатель на правый дочерний узел если значение больше текущего и на левый если меньше
            currNode = (sorter(val, currNode->key)) ? currNode->right : currNode->left;
            // Возращается NULL, если значение небыло найденно(так как перешли на указатель дочернего узла, который равен нулю)
            // и указатель на узел со значением если он существует
            return currNode;
        }

        ValueType get(KeyType key)
        {
            // Получаем указатель на узел со значением или ноль если такого узла нет
            Node* currNode = find(key);
            // Если указатель не нулевой, то возращаем значение узла.
            // Если нулевой, то значение по умолчанию
            return (currNode) ? currNode->value : ValueType();
        }

        void insert(KeyType key, ValueType val)
        {
```

```
            // Ключу соответсятует только одно значение,
            // если данный ключ уже есть, то завершаем функцию.
            Node* keyIsValNode = find(key);
            if (keyIsValNode)
                return;
            if (mainRoot) // Если в дереве есть узлы(указатель на корень не нулевой)
            {
                // Начинаем обход с корня, для этого устанавливаем на него указатель
                Node* currNode = mainRoot;
                // Переходим в зависимости от значения ключа создаваемого узла и ключа текущего узла
                // на правое или левое поддерево
                for (Node* nextNode = (sorter(key, currNode->key)) ? currNode->right : currNode->left;
                        nextNode;
                        nextNode = (sorter(key, currNode->key)) ? currNode->right : currNode->left)
                        currNode = nextNode;
                // Найдя родительский узел для создаваемого узла,
                // в зависимости от значений ключей,
                // создаем дочерней узел с права или слева
                if (sorter(key, currNode->key))
```

```
                currNode->right = new Node{ key, NULL, NULL,
currNode, val };
            else
                currNode->left = new Node{ key, NULL, NULL,
currNode, val };
        }
        // Если дерево пустое, то корнем становится создаваемый узел
        else
            mainRoot = new Node{ key, NULL, NULL, NULL, val };
    }

    void erase(KeyType val)
    {
        // Если узла с таким значением нет, то завершаем функцию
        Node* currNode = find(val);
        if (!currNode)
            return;
        // Если удаляемый узел имеет дочерние узлы с обоих сторон
        if (currNode->left && currNode->right)
        {
            // Находим минимальный по клюжчу узел правого
поддерева
            Node* rightTreeMinNode = currNode->right;
```

```
            while (rightTreeMinNode->left)
                rightTreeMinNode = rightTreeMinNode->left;
            // Меняем значение удаляемого узла значением найденного узла
            KeyType rtmnKey = rightTreeMinNode->key;
            // Рекурсивно удаляем найденный узел
            erase(rightTreeMinNode->key);
            currNode->key = rtmnKey;
        }
        // Если удаляемый узел имеет один дочерний узел или не имеет потомков
        else
        {
            // Если у удаляемого узла есть левый дочерний узелб
            // то он становится узлом-преемником,
            // иначе узел-преемник - правый узел
            // или, в том случае если удаляемый узел не имеет потомков - нулевой указатель
            Node* successorNode = (currNode->left) ? currNode->left : currNode->right;
            // В том случае если указатель на узел-преемник не нулевой, то
            // то меняем значение указателя дочернего узла на родительский узел,
```

```
                // который может быть нулевым, если удаляемый узел - корень
                successorNode && (successorNode->parent = currNode->parent);
                // Если удаляемый узел не корень
                if (currNode->parent)
                {
                        // Меняем указатель родительского узла на дочерний узел
                        // на указатель узла-преемника
                        if (currNode == currNode->parent->left)
                                currNode->parent->left = successorNode;
                        else
                                currNode->parent->right = successorNode;
                }
                // Если удаляемый узел-корень
                else
                        // то корнем становится его дочерний узел или нулевой указатель
                        mainRoot = successorNode;
                // удаляем удаляемый узел
                delete currNode;
        }
```

```
    }
    void clear()
    {
        clear(mainRoot);
    }


    void clear(Node*& currNode)
    {
        // Оптимальным алгоритмом удаления дерева
        // является такой алгоритм, который в процессе
        // выполнения не требует перестраивания дерева.
        // Для этого нужно удалять только те узлы которые не имею потомков.
        // Для этого можно удалять узлы в обратном топологическом порядке
        // обходя узлы методом обратного(LRN) обхода.

        // Условие выхода из рекурсии
        if (!currNode)
            return;

        // L - рекурсивное удаление левых поддеревьев
```

```
            clear(currNode->left);
            // R - рекурсивное удаление правых поддеревьев
            clear(currNode->right);
            // N - удаление самомго узла
            delete currNode;
            currNode = NULL;
        }

        void traverseVals(Node* currNode, TraversalType traversalType, void (*func)(const ValueType&, int, va_list), int argCount, va_list funcArgs)
        {
            if (!currNode)
                return;
            switch (traversalType)
            {
            case LNR_REQ:
                traverseVals(currNode->left, TraversalType::LNR_REQ, func, argCount, funcArgs);
                func(currNode->value, argCount, funcArgs);
                traverseVals(currNode->right, TraversalType::LNR_REQ, func, argCount, funcArgs);
                break;
            case RNL_REQ:
```

```
                traverseVals(currNode->right, TraversalType::RNL_REQ,
func, argCount, funcArgs);
                func(currNode->value, argCount, funcArgs);
                traverseVals(currNode->left, TraversalType::RNL_REQ,
func, argCount, funcArgs);
                break;
            case NLR_REQ:
                func(currNode->value, argCount, funcArgs);
                traverseVals(currNode->left, TraversalType::NLR_REQ,
func, argCount, funcArgs);
                traverseVals(currNode->right, TraversalType::NLR_REQ,
func, argCount, funcArgs);
                break;
            case LRN_REQ:
                traverseVals(currNode->left, TraversalType::LRN_REQ,
func, argCount, funcArgs);
                traverseVals(currNode->right, TraversalType::LRN_REQ,
func, argCount, funcArgs);
                func(currNode->value, argCount, funcArgs);
                break;
            default:
                break;
            }
        }
```

```
void traverseVals_BFS(Node* currNode, TraversalType traversalType, void (*func)(const ValueType&, int, va_list), int argCount, va_list funcArgs)
{


}
void traverseVals(void (*func)(const ValueType&, int, va_list), int argCount = 0, ...)
{
	va_list funcArgs;
	va_start(funcArgs, argCount);
	traverseVals(mainRoot, TraversalType::LNR_REQ, func, argCount, funcArgs);
	va_end(funcArgs);
}
void traverseVals(Node* currNode, TraversalType traversalType, void (*func)(const ValueType&, int, va_list), int argCount = 0, ...)
{
	va_list funcArgs;
	va_start(funcArgs, argCount);
	if(traversalType == TraversalType::BFS_QUE)
		traverseVals_BFS(currNode, traversalType, func, argCount, funcArgs);
	else
```

```
                traverseVals(currNode, traversalType, func, argCount,
funcArgs);
            va_end(funcArgs);
        }



};

typedef BinSearchTree::Node* SinglyLinkedList_queue_ValueType;
struct SinglyLinkedList_queue
{
        struct Node
        {
            Node* next;
            SinglyLinkedList_queue_ValueType data;
        };
        Node* head;
        Node* tail;
        void insert(SinglyLinkedList_queue_ValueType val)
        {

            Node* currNode = new Node;
            currNode->data = val;
```

```
        currNode->next = NULL;
        if (tail == NULL)
        {
                head = currNode;
                tail = currNode;
        }
        else
        {
                tail->next = currNode;
                tail = currNode;
        }
}
SinglyLinkedList_queue_ValueType shift()
{
        if (!head)
                return NULL;
        Node* nextHead = head->next;
        SinglyLinkedList_queue_ValueType popedValue = head->data;
        delete head;
        head = nextHead;
        return popedValue;
}
```

```
        void destroy()
        {
                for (Node* currNode = head, *nex; currNode; currNode = nex)
                {
                        nex = currNode->next;
                        delete currNode;
                }
                head = NULL;
                tail = NULL;
        }
};
//-------------------------------------------------------------------------------------------
-------------------------------------------------------------------

// Функции для форматирования текста в консоли
void coutCenterStr(const std::string&);  //
https://ru.stackoverflow.com/questions/802079/%D0%92%D1%8B%D1%80%
D0%B0%D0%B2%D0%BD%D0%B8%D0%B2%D0%B0%D0%BD%D0%B8
%D0%B5-%D0%BF%D0%BE-
%D1%86%D0%B5%D0%BD%D1%82%D1%80%D1%83-
%D0%B2%D1%8B%D0%B2%D0%BE%D0%B4-%D0%B2-%D1%81
void gotoxy(short, short); // https://forum.vingrad.ru/topic-279072.html
std::string formatNumber(std::string);
```

```
int countNodesBetweenByKeys(BinSearchTree*, KeyType, KeyType);


// Функции экранов меню:
void intro(int argc, char* argv[]);
void mainMenu(int argc, char* argv[]);
void valueInputMenu(bool&, HANDLE&, BinSearchTree*);
void valueEraseManu(bool&, HANDLE&, BinSearchTree*);
void countNodesBetweenByKeysMenu(HANDLE&, BinSearchTree*);
void saveInFileMenu(int argc, char* argv[], bool&, HANDLE&,
BinSearchTree*);
void openFromFileMenu(int argc, char* argv[], bool&, HANDLE&,
BinSearchTree*&);


// Функции для работы с BinTree
// Функция определяющая операцию сравнения для ключей
bool greaterInt(KeyType a, KeyType b)
{
        return (a > b);
}


void coutData(const ValueType& data, int argCount, va_list args)
{
```

```
	std::cout << "  " << std::setw(14) << std::left << data.number << " | " << std::setw(27) << std::left << data.owner << " | " << std::setw(24) << std::left << data.tariff << std::endl;
}
void countNodes(const ValueType& data, int argCount, va_list args)
{
	(*va_arg(args, int*))++;
}
void cpyValsToBuff(const ValueType& data, int argCount, va_list args)
{
	*va_arg(args, ValueType*)++ = data;
}
void cpyKeysToBuff(const ValueType& data, int argCount, va_list args)
{
	*va_arg(args, KeyType*)++ = data.number;
}

void coutSubOwnerByNumber(BinSearchTree& binTree, KeyType number)
{
	TeleSubscriber sub = binTree.get(number);
	std::cout << number << " sub is: " << ((sub.owner.size()) ? sub.owner : "UNDEFINED") << std::endl;
}
```

```
int scoutPath(BinSearchTree::Node* currNode, KeyType key, int pathLength,
SinglyLinkedList_queue& addOfVisitedNodes)
{
	// Если текущий узел не существует(указатель на него нулевой),
	// то вернуть отрицательный результат, так как
	// данная ветвь рекурсии не смогла обнаружить узел к которому ищется путь
	if (!currNode)
		return -1;
	// Если текущий узел уже был посещен,
	// (его адрес есть в контейнере уже посещенных узлов)
	// то вернуть отрицательный результат, так как
	// данная ветвь рекурсии или уже была обработана или обработается другой ветвью
	for (SinglyLinkedList_queue::Node* iter = addOfVisitedNodes.head; iter != NULL; iter = iter->next)
	{
		//std::cout << "<<< " << iter->data->key << " >>>" << std::endl;
		if (iter->data == currNode)
			return -1;
	}
	// Добавить адресс текущего узла в контейнер,
```

```
/// который сожержит адреса уже посещенных узлов
addOfVisitedNodes.insert(currNode);
// Если текущий узел - узел назначения,
// то вернуть колличество узлов лежищих между ним
// и начальным узлом
if (currNode->key == key)
        return pathLength;
// Возращаемое значение
int returnVal;
// Рекурсивно искать в родительском узле
returnVal = scoutPath(currNode->parent, key, pathLength + 1,
addOfVisitedNodes);
// Если искомый узел не был найден(результат отрицателен), то
искать в левом дочернем узле
if(returnVal == -1)
        returnVal = scoutPath(currNode->left, key, pathLength + 1,
addOfVisitedNodes);
// Если искомый узел не был найден(результат отрицателен), то
искать в правом дочернем узле
if (returnVal == -1)
        returnVal = scoutPath(currNode->right, key, pathLength + 1,
addOfVisitedNodes);
// Вернуть найденный результат
return returnVal;
```

```
}
int scoutPath(BinSearchTree* bst, KeyType keyA, KeyType keyB)
{
	// Если один из узлов отсутствует в дереве, то возращаем отрицательный результат
	BinSearchTree::Node* nodeA = bst->find(keyA);
	if (!nodeA || !bst->find(keyB))
		return -1;
	// В качесте контейнера для хранения адрессов уже посещенных узлов,
	// лучше было бы использовать все тот-же BinSearchTree,
	// но чтобы реализовать хранения ключей и значений произвольного типа нужны шаблоны
	int nodeCount = 0;
	bst->traverseVals(countNodes, 1, &nodeCount);
	SinglyLinkedList_queue addOfVisitedNodes{NULL, NULL};
	// Устанавливает изначальные значения переменных
	return scoutPath(nodeA, keyB, -1, addOfVisitedNodes);
}

struct Preinseq
{
	int length;
```

```
        KeyType* nlrArr;
        KeyType* lnrArr;
        ValueType* valsArr;
};
void recon_BST_from_NLR_LNR_seq(BinSearchTree::Node*& rootNodeAdd, KeyType* nlr, int nlrLength, int nlrRootIndex, KeyType* lnr, int lnrLength, int lnrBeginIndex, ValueType* vals)
{
        if (lnrLength < 1)
        {
                rootNodeAdd = NULL;
                return;
        }
        rootNodeAdd->key = nlr[nlrRootIndex];
        rootNodeAdd->value = vals[nlrRootIndex];
        rootNodeAdd->left = new BinSearchTree::Node;
        rootNodeAdd->left->parent = rootNodeAdd;
        rootNodeAdd->right = new BinSearchTree::Node;
        rootNodeAdd->right->parent = rootNodeAdd;

        int lnrRootIndex;
        for (lnrRootIndex = 0; lnr[lnrRootIndex] != nlr[nlrRootIndex]; lnrRootIndex++);
```

```
// https://cs.stackexchange.com/questions/439/which-combinations-of-pre-post-and-in-order-sequentialisation-are-unique

recon_BST_from_NLR_LNR_seq(rootNodeAdd->left, nlr, (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex, nlrRootIndex + 1, lnr, (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex, lnrBeginIndex, vals);

recon_BST_from_NLR_LNR_seq(rootNodeAdd->right, nlr, nlrLength - (( lnrRootIndex+(nlrRootIndex-lnrBeginIndex) ) - nlrRootIndex) - 1, (lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) + 1, lnr, nlrLength - ((lnrRootIndex + (nlrRootIndex - lnrBeginIndex)) - nlrRootIndex) - 1, lnrRootIndex + 1, vals);

}

void recon_BST_from_NLR_LNR_seq(BinSearchTree::Node*& rootNodeAdd, Preinseq& preinseq)

{

recon_BST_from_NLR_LNR_seq(rootNodeAdd, preinseq.nlrArr, preinseq.length, 0, preinseq.lnrArr, preinseq.length, 0, preinseq.valsArr);

}

Preinseq con_NLR_LNR_seq_of_BST(BinSearchTree*& bst)

{

Preinseq preinseq;


preinseq.length = 0;

bst->traverseVals(countNodes, 1, &preinseq.length);
```

```
	preinseq.nlrArr = new KeyType[preinseq.length];
	preinseq.lnrArr = new KeyType[preinseq.length];
	preinseq.valsArr = new ValueType[preinseq.length];

	bst->traverseVals(bst->mainRoot,
BinSearchTree::TraversalType::NLR_REQ, cpyKeysToBuff, 1,
preinseq.nlrArr);
	bst->traverseVals(bst->mainRoot,
BinSearchTree::TraversalType::LNR_REQ, cpyKeysToBuff, 1,
preinseq.lnrArr);
	bst->traverseVals(bst->mainRoot,
BinSearchTree::TraversalType::NLR_REQ, cpyValsToBuff, 1,
preinseq.valsArr);

	return preinseq;
}
bool make_preinseq_file(std::string path, Preinseq& preinseq)
{
	std::ofstream outFile(path + ".preinseq", std::ios::binary);
	if (!outFile)
		return false;
	outFile.write(reinterpret_cast<char*>(&preinseq.length),
sizeof(preinseq.length));
```

```
	for (int i = 0; i < preinseq.length; i++)
		outFile.write(reinterpret_cast<char*>(preinseq.nlrArr + i), sizeof(preinseq.nlrArr[i]));
	for (int i = 0; i < preinseq.length; i++)
		outFile.write(reinterpret_cast<char*>(preinseq.lnrArr + i), sizeof(preinseq.lnrArr[i]));
	for (int i = 0; i < preinseq.length; i++)
		outFile.write(reinterpret_cast<char*>(preinseq.valsArr + i), sizeof(preinseq.valsArr[i]));
	if (!outFile)
		return false;
	outFile.close();
	return true;
}
Preinseq read_preinseq_file(std::string path)
{
	Preinseq preinseq;
	std::ifstream inFile(path + ".preinseq", std::ios::binary);
	inFile.seekg(inFile.beg);
	if (!inFile)
	{
		preinseq.length = -1;
		return preinseq;
```

```
    }
    inFile.read(reinterpret_cast<char*>(&preinseq.length),
sizeof(preinseq.length));
    preinseq.nlrArr = new KeyType[preinseq.length];
    preinseq.lnrArr = new KeyType[preinseq.length];
    preinseq.valsArr = new ValueType[preinseq.length];
    for (int i = 0; i < preinseq.length; i++)
        inFile.read(reinterpret_cast<char*>(preinseq.nlrArr + i),
sizeof(preinseq.nlrArr[i]));
    for (int i = 0; i < preinseq.length; i++)
        inFile.read(reinterpret_cast<char*>(preinseq.lnrArr + i),
sizeof(preinseq.lnrArr[i]));
    for (int i = 0; i < preinseq.length; i++)
        inFile.read(reinterpret_cast<char*>(preinseq.valsArr + i),
sizeof(preinseq.valsArr[i]));

    if (!inFile)
    {
        preinseq.length = -1;
        return preinseq;
    }
    inFile.close();
    return preinseq;
```

```
}


int main(int argc, char* argv[])
{
	intro(argc, argv);
	return 0;
}
void coutCenterStr(const std::string& s)
{
	int width_field = 68;
	std::string v(width_field, ' ');
	std::string v1;

	int l_word = s.size();
	int l_field = v.size();
	int step = (l_field - l_word) / 2;

	for (int i = 0; i < l_word; i++) {
		v1.push_back(s[i]);
	}
	copy(v1.begin(), v1.end(), v.begin() + step);
	for (auto x : v) {
```

```
            std::cout << x;
        }
        std::cout << std::endl;
}
void gotoxy(short x, short y)
{
        HANDLE    StdOut = GetStdHandle(STD_OUTPUT_HANDLE);
        COORD    coord = { x, y };
        SetConsoleCursorPosition(StdOut, coord);
}
int wherex()
{
        HANDLE                  StdOut =
GetStdHandle(STD_OUTPUT_HANDLE);
        CONSOLE_SCREEN_BUFFER_INFO    csbi;
        GetConsoleScreenBufferInfo(StdOut, &csbi);
        return int(csbi.dwCursorPosition.X);
}
int wherey()
{
        HANDLE                  StdOut =
GetStdHandle(STD_OUTPUT_HANDLE);
        CONSOLE_SCREEN_BUFFER_INFO    csbi;
```

```
	GetConsoleScreenBufferInfo(StdOut, &csbi);
	return int(csbi.dwCursorPosition.Y);
}

void pressAnyKeyToContinue()
{
	for (int state = 1, lastClockVal = clock(); !_kbhit(); state = (state > 1000) ? (state % 2) ? 0 : 1 : state + (clock() - lastClockVal) * 2, lastClockVal = clock())
	{
		if (state % 2)
		{
			gotoxy(0, wherey() - 1);
			coutCenterStr("  НАЖМИТЕ ЛЮБУЮ КЛАВИШУ ВВОДА ДЛЯ ПРОДОЛЖЕНИЯ");
		}
		else
		{
			gotoxy(0, wherey() - 1);
			printf("
\n");
		}
	}
```

```
	_getche();
	gotoxy(0, wherey());
	std::cout << " ";
	gotoxy(0, wherey());
}
void intro(int argc, char* argv[])
{
	// https://www.cyberforum.ru/cpp/thread2454842.html
	CONSOLE_CURSOR_INFO curs = { 0 };
	curs.dwSize = sizeof(curs);
	curs.bVisible = FALSE;
	::SetConsoleCursorInfo(::GetStdHandle(STD_OUTPUT_HANDLE),
&curs);

	SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE),
(WORD)((0 << 4) | 15));

	setlocale(LC_ALL, "Russian");

	std::cout << "\n\n";
	coutCenterStr("  КУРСОВАЯ РАБОТА");
	coutCenterStr("  по дисциплине");
	coutCenterStr("  \"Алгоритмизация и программирование\"");
```

```
        coutCenterStr("  на тему");
        coutCenterStr("  \"Определения количества узлов, лежащих на пути между двумя узлами,");
        coutCenterStr("  заданными своими ключевыми признаками в бинарном дереве\"");


        std::cout << std::endl <<
                "  Выполнил" << std::endl <<
                "  обучающийся группы 15.11Д-МО11/19б" << std::endl <<
                "  очной формы обучения" << std::endl <<
                "  института цифровой экономики" << std::endl <<
                "  и информационных технологий" << std::endl <<
                "  Гришанин Вячеслав Валерьевич" << std::endl;
        std::cout << "\n\n";
        pressAnyKeyToContinue();
        mainMenu(argc, argv);
}
void mainMenu(int argc, char* argv[])
{
        HANDLE hConsoleHandle = GetStdHandle(STD_OUTPUT_HANDLE);
        BinSearchTree* subsBase = NULL;
        bool isSaved = false;
        while (true)
```

```
{

    system("cls");
    std::cout << "  БИНАРНОЕ ДЕРЕВО (БАЗА АБОНЕНТОВ): ";
    if (subsBase)
    {
        if (isSaved)
            std::cout << "СОХРАНЕННО В ФАЙЛЕ";
        else
            std::cout << "НЕ ЗАПИСАННО В ФАЙЛ";
    }
    else
        std::cout << "НЕОПРЕДЕЛЕННО";
    std::cout << "\n\n";

    coutCenterStr("  ВЫБЕРИТЕ ДЕЙСТВИЕ:");
    std::cout << "  КЛАВИША 1 - создать новое бинарное дерево(базу абонентов);" << std::endl
                << "  КЛАВИША 2 - обход бинарного дерева с выдачей на экран содержимого информационных полей(вывод всех абонентов базы);" << std::endl
                << "  КЛАВИША 3 - включить элемент в бинарное дерево(добавить абонента);" << std::endl
```

```
                << "  КЛАВИША 4 - удаление заданного узла из дерева (удаление абонента);" << std::endl
                << "  КЛАВИША 5 - удаление дерева(базы абонентов) с освобождением памяти;" << std::endl
                << "  КЛАВИША 6 - определить количество узлов, лежащих на пути между двумя узлами, заданными своими ключевыми признаками;" << std::endl
                << "  КЛАВИША 7 - открыть бинарное дерево(базу абонентов) из файла;" << std::endl
                << "  КЛАВИША 8 - сохранить текущее состояние бинарного дерева(базы абонентов) в файл;" << std::endl
                << "  КЛАВИША 0 - заваершить программу;" << std::endl;

        GET_ACTION:
        char action = _getche();
        gotoxy(0, wherey());
        std::cout << " ";
        gotoxy(0, wherey());
        switch (action - 48)
        {
        case 1:
            system("cls");
            std::cout << "\n\n";
```

```
                if (subsBase)
                {
                        coutCenterStr("  В ДАННЫЙ МОМЕНТ
ОБРАБАТЫВАЕТСЯ СОЗДАННОЕ РАНЕЕ ДЕРЕВО");
                }
                else
                {
                        subsBase = new BinSearchTree{ NULL, greaterInt};
                        coutCenterStr("  НОВОЕ БИНАРНОЕ ДЕРЕВО
(БАЗА АБОНЕНТОВ) СОЗДАННО");
                }
                std::cout << "\n\n";
                pressAnyKeyToContinue();
                break;
        case 2:


                system("cls");
                std::cout << "\n\n";
                if (subsBase)
                {
                        if (subsBase->mainRoot)
                        {
```

```
                    SetConsoleTextAttribute(hConsoleHandle,
(WORD)((7 << 4) | 0));
                    std::cout << "  " << std::setw(14) << std::left <<
"ТЕЛЕФОН (КЛЮЧ)"<< " | " << std::setw(27) << std::left << "ВЛАДЕЛЕЦ"
<< " | " << std::setw(24) << std::left << "ТАРИФ" << std::endl;
                    SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 15));
                    //subsBase->traverseVals(coutData);
                    subsBase->traverseVals(subsBase->mainRoot,
BinSearchTree::TraversalType::LNR_REQ, coutData);

                }
                else
                    coutCenterStr("  БИНАРНОЕ ДЕРЕВО (БАЗА
АБОНЕНТОВ) НЕ ИМЕЕТ ЗНАЧЕНИЙ");
            }
            else
                coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ
ДЕРЕВО (БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");

            std::cout << "\n\n";
            pressAnyKeyToContinue();
            break;
        case 3:
```

```
            valueInputMenu(isSaved, hConsoleHandle, subsBase);
            break;
        case 4:
            valueEraseManu(isSaved, hConsoleHandle, subsBase);
            break;
        case 5:
            system("cls");
            std::cout << "\n\n";
            if (subsBase)
            {
                subsBase->clear();
                delete subsBase;
                subsBase = NULL;
                coutCenterStr("  БИНАРНОЕ ДЕРЕВО(БАЗА АБОНЕНТОВ) УДАЛЕННО");
            }
            else
                coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ ДЕРЕВО (БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");

            std::cout << "\n\n";
            pressAnyKeyToContinue();
            break;
```

```
            case 6:
                countNodesBetweenByKeysMenu(hConsoleHandle,
subsBase);
                break;
            case 7:
                openFromFileMenu(argc, argv, isSaved, hConsoleHandle,
subsBase);
                break;
            case 8:
                saveInFileMenu(argc, argv, isSaved, hConsoleHandle,
subsBase);
                break;
            case 0:
                system("cls");
                return;
                break;
            default:
                goto GET_ACTION;
                break;
            }
        }
}
```

```
void valueInputMenu(bool& isSaved, HANDLE& hConsoleHandle,
BinSearchTree* subsBase)
{
	int selectedMenuItem = 1;
	system("cls");
	std::cout << "\n\n";

	if (!subsBase)
	{
		coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ ДЕРЕВО
(БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");
		std::cout << "\n\n";
		pressAnyKeyToContinue();
		return;
	}
	coutCenterStr("  ВВЕДИТЕ ДАННЫЕ ДОБАВЛЯЕМОГО УЗЛА
(АБОНЕНТА)");
	SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
	std::cout << std::setw(68) << std::left << "  НОМЕР: +_ (___) ___-__-
__" << std::endl;
	SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
	std::cout << "    ИМЯ: _____" << std::endl;
	std::cout << "  ТАРИФ: _____" << std::endl;
```

```
        coutCenterStr("  ДОБАВИТЬ УЗЕЛ (АБОНЕНТА)");
        std::cout << "\n\n"
                << "  КЛАВИША UP - переключение на поле выше;" << std::endl
                << "  КЛАВИША DOWN - переключение на поле ниже;" << std::endl
                << "  КЛАВИША ENTER - подтвердить ввод;" << std::endl
                << "  КЛАВИША ESC - отменить ввод и вернуться в меню выбора действия;" << std::endl;

        std::string number = "", name = "", tariff = "";
        int ch;
        while (GetKeyState(VK_ESCAPE) >= 0)
        {
                std::string formatedNumber = formatNumber(number),
                                formatName = (name.length())?name:"__________________",
                                        formatTariff = (tariff.length()) ? tariff : "__________________";

                if (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0 || GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
                {
                        gotoxy(0, wherey() - 1);
```

```
                SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 <<
4) | 15));
                if (selectedMenuItem == 1)
                        std::cout << std::setw(68) << std::left << ("  НОМЕР: "
+ formatedNumber) << std::endl;
                if (selectedMenuItem == 2)
                        std::cout << std::setw(68) << std::left << "    ИМЯ: " +
formatName;
                if (selectedMenuItem == 3)
                        std::cout << std::setw(68) << std::left << "  ТАРИФ: "
+ formatTariff << std::endl;
                if (selectedMenuItem == 4)
                        coutCenterStr("  ДОБАВИТЬ УЗЕЛ (АБОНЕНТА)");
                if (GetKeyState(VK_UP) < 0 && selectedMenuItem > 1)
                        selectedMenuItem--;

                if (GetKeyState(VK_RETURN) < 0 && selectedMenuItem
== 4)
                {
                        gotoxy(27, 3);
                        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0
<< 4) | 4));
                        if (number.length())
                        {
```

```
                    if(subsBase->find(stoull(number)))
                        std::cout << "(ДАННЫЙ НОМЕР УЖЕ
ЕСТЬ В БАЗЕ)" << std::endl;
                    else
                    {
                        subsBase->insert(stoull(number),
TeleSubscriber{ stoull(number), name, tariff });
                        SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 15));
                        isSaved = false;
                        return;
                    }
                }
                else
                    std::cout << "(ПОЛЕ С НОМЕРОМ НЕ
ДОЛЖНО БЫТЬ ПУСТЫМ)" << std::endl;
            }

            if ((GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0) && selectedMenuItem < 4)
                selectedMenuItem++;

            if(GetKeyState(VK_BACK) < 0)
                switch (selectedMenuItem)
```

```
                {
                case 1:
                    if(number.length())
                        number.pop_back();
                    break;
                case 2:
                    if (name.length())
                        name.pop_back();
                    break;
                case 3:
                    if (tariff.length())
                        tariff.pop_back();
                    break;
                default:
                    break;
                }
            }
            while (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0
|| GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
            { }

            SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
            gotoxy(0, selectedMenuItem + 2);
```

```
            if (selectedMenuItem == 1)
                std::cout << std::setw(68) << std::left << ("  НОМЕР: " +
formatedNumber) << std::endl;
            if (selectedMenuItem == 2)
                std::cout << std::setw(68) << std::left << "    ИМЯ: " +
formatName << std::endl;
            if (selectedMenuItem == 3)
                std::cout << std::setw(68) << std::left << "  ТАРИФ: " +
formatTariff << std::endl;
            if (selectedMenuItem == 4)
                coutCenterStr("  ДОБАВИТЬ УЗЕЛ (АБОНЕНТА)");
            SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) |
15));

            if (_kbhit())
            {
                ch = _getche();
                gotoxy(0, wherey());
                std::cout << " ";
                gotoxy(0, wherey());
                if (ch == 0xE0 || ch == 0x00) //
https://ru.stackoverflow.com/questions/1141080/%d0%9f%d1%80%d0%be%d
0%b1%d0%bb%d0%b5%d0%bc%d0%b0-%d1%81-
%d0%bf%d0%be%d0%bb%d1%83%d1%87%d0%b5%d0%bd%d0%b8%d0%b
```

5%d0%bc-%d1%81%d0%b8%d0%bc%d0%b2%d0%be%d0%bb%d0%b0-%d1%81-%d0%bf%d0%be%d0%bc%d0%be%d1%89%d1%8c%d1%8e-getche

```
                {
                        ch = _getch();
                        if (ch == 0x48 || ch == 0x50 || ch == 0x4B || ch == 0x4D)
                                continue;
                }
                switch (selectedMenuItem)
                {
                case 1:
                        if (number.length() < 11 && ch > 47 && ch < 58)
                                number += ch;
                        break;
                case 2:
                        if ((ch > 96 && ch < 123) || (ch > 64 && ch < 91) || ch == ' ')
                                name += ch;
                        break;
                case 3:
                        if ((ch > 32 && ch < 127) || ch == ' ')
                                tariff += ch;
                        break;
```

```
                }
            }

        }

}
void valueEraseManu(bool& isSaved, HANDLE& hConsoleHandle,
BinSearchTree* subsBase)
{
        int selectedMenuItem = 1;
        system("cls");
        std::cout << "\n\n";
        if (subsBase)
        {
            if (!subsBase->mainRoot)
            {
                coutCenterStr("  БИНАРНОЕ ДЕРЕВО (БАЗА
АБОНЕНТОВ) НЕ ИМЕЕТ ЗНАЧЕНИЙ");
                std::cout << "\n\n";
                pressAnyKeyToContinue();
                return;
            }
        }
```

```
        else
        {
            coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ ДЕРЕВО (БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");
            std::cout << "\n\n";
            pressAnyKeyToContinue();
            return;
        }
        coutCenterStr("  ВВЕДИТЕ КЛЮЧ (НОМЕР) УДАЛЯЕМОГО УЗЛА (АБОНЕНТА)");
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        std::cout << std::setw(68) << std::left << "  НОМЕР: +_ (___) ___-__-__" << std::endl;
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
        coutCenterStr("  УДАЛИТЬ УЗЕЛ (АБОНЕНТА)");
        std::cout << "\n\n"
            << "  КЛАВИША UP - переключение на поле выше;" << std::endl
            << "  КЛАВИША DOWN - переключение на поле ниже;" << std::endl
            << "  КЛАВИША ENTER - подтвердить ввод;" << std::endl
            << "  КЛАВИША ESC - отменить ввод и вернуться в меню выбора действия;" << std::endl;
```

```
        std::string number = "";
        int ch;
        while (GetKeyState(VK_ESCAPE) >= 0)
        {
                std::string formatedNumber = formatNumber(number);

                if (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
                {
                        gotoxy(0, wherey() - 1);
                        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 <<
4) | 15));
                        if (selectedMenuItem == 1)
                                std::cout << std::setw(68) << std::left << ("  НОМЕР: "
+ formatedNumber) << std::endl;
                        if (selectedMenuItem == 2)
                                coutCenterStr("  УДАЛИТЬ УЗЕЛ (АБОНЕНТА)");
                        if (GetKeyState(VK_UP) < 0 && selectedMenuItem > 1)
                                selectedMenuItem--;

                        if (GetKeyState(VK_RETURN) < 0 && selectedMenuItem
== 2)
                        {
```

```
                    gotoxy(27, 3);
                    SetConsoleTextAttribute(hConsoleHandle, (WORD)((0
<< 4) | 4));
                    if (number.length())
                    {
                        if (!subsBase->find(stoull(number)))
                            std::cout << "(ДАННЫЙ НОМЕР
ОТСУТСТВУЕТ В БАЗЕ)" << std::endl;
                        else
                        {
                            subsBase->erase(stoull(number));
                            SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 15));
                            isSaved = false;
                            return;
                        }
                    }
                    else
                        std::cout << "(ПОЛЕ С НОМЕРОМ НЕ
ДОЛЖНО БЫТЬ ПУСТЫМ)" << std::endl;
                }

                if ((GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0) && selectedMenuItem < 2)
```

```
                selectedMenuItem++;


            if (GetKeyState(VK_BACK) < 0)
                switch (selectedMenuItem)
                {
                case 1:
                    if (number.length())
                        number.pop_back();
                    break;
                default:
                    break;
                }
        }
        while (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0
|| GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
        { }


        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        gotoxy(0, selectedMenuItem + 2);
        if (selectedMenuItem == 1)
            std::cout << std::setw(68) << std::left << ("  HOMEP: " +
formatedNumber) << std::endl;
        if (selectedMenuItem == 2)
```

```
            coutCenterStr("  УДАЛИТЬ УЗЕЛ (АБОНЕНТА)");
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) |
15));

        if (_kbhit())
        {
            ch = _getche();
            gotoxy(0, wherey());
            std::cout << " ";
            gotoxy(0, wherey());
            if (ch == 0xE0 || ch == 0x00)
            {
                ch = _getch();
                if (ch == 0x48 || ch == 0x50 || ch == 0x4B || ch ==
0x4D)
                    continue;
            }
            switch (selectedMenuItem)
            {
            case 1:
                if (number.length() < 11 && ch > 47 && ch < 58)
                    number += ch;
                break;
```

```
                }
            }

        }
}
void countNodesBetweenByKeysMenu(HANDLE& hConsoleHandle,
BinSearchTree* subsBase)
{
        int selectedMenuItem = 1;
        system("cls");
        std::cout << "\n\n";
        if (subsBase)
        {
            int nodesCount = 0;
            subsBase->traverseVals(countNodes, 1, &nodesCount);
            if (nodesCount < 2)
            {
                coutCenterStr("  В БИНАРНОМ ДЕРЕВЕ (БАЗЕ
АБОНЕНТОВ) МЕНЕЕ ДВУХ ЗНАЧЕНИЙ");
                std::cout << "\n\n";
                pressAnyKeyToContinue();
                return;
            }
```

```
        }
        else
        {
            coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ ДЕРЕВО (БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");
            std::cout << "\n\n";
            pressAnyKeyToContinue();
            return;
        }
        coutCenterStr("  ВВЕДИТЕ КЛЮЧИ УЗЛОВ ДЛЯ ОПРЕДЕЛЕНИЯ КОЛЛИЧЕСТВА УЗЛОВ МЕЖДУ НИМИ");
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        std::cout << std::setw(68) << std::left << "  КЛЮЧ A: +_ (___) ___-__-__" << std::endl;
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
        std::cout << std::setw(68) << std::left << "  КЛЮЧ B: +_ (___) ___-__-__" << std::endl;
        coutCenterStr("  РАССЧИТАТЬ");
        std::cout << std::endl << std::setw(68) << std::left << "  УЗЛОВ В БИНАРНОМ ДЕРЕВЕ МЕЖДУ 'A' И 'B': НЕОПРЕДЕЛЕННО" << std::endl;
        std::cout << "\n\n"
            << "  КЛАВИША UP - переключение на поле выше;" << std::endl
```

```
<< "  КЛАВИША DOWN - переключение на поле ниже;" << std::endl
<< "  КЛАВИША ENTER - подтвердить ввод;" << std::endl
<< "  КЛАВИША ESC - отменить ввод и вернуться в меню выбора действия;" << std::endl;

std::string numberA = "", numberB = "";
int ch;
while (GetKeyState(VK_ESCAPE) >= 0)
{
    std::string formatNumberA = formatNumber(numberA),
                formatNumberB = formatNumber(numberB);

    if (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0 || GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
    {
        gotoxy(0, wherey() - 1);
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
        if (selectedMenuItem == 1)
            std::cout << std::setw(68) << std::left << ("  КЛЮЧ A: " + formatNumberA) << std::endl;
        if (selectedMenuItem == 2)
```

```
                std::cout << std::setw(68) << std::left << ("  КЛЮЧ B: " + formatNumberB) << std::endl;
            if (selectedMenuItem == 3)
                coutCenterStr("  РАССЧИТАТЬ");
            if (GetKeyState(VK_UP) < 0 && selectedMenuItem > 1)
                selectedMenuItem--;

            if (GetKeyState(VK_RETURN) < 0 && selectedMenuItem == 3)
            {
                bool isKeysRelevant = true;
                SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 4));
                gotoxy(29, 3);
                if (numberA.length())
                {
                    if (!subsBase->find(stoull(numberA)))
                    {
                        std::cout << "(ДАННЫЙ НОМЕР ОТСУТСТВУЕТ В БАЗЕ)" << std::endl;
                        isKeysRelevant = false;
                    }
```

```
                }
                else
                {
                        std::cout << "(ПОЛЕ С НОМЕРОМ НЕ
ДОЛЖНО БЫТЬ ПУСТЫМ)" << std::endl;
                        isKeysRelevant = false;
                }

                gotoxy(29, 4);
                if (numberB.length())
                {
                        if (!subsBase->find(stoull(numberB)))
                        {
                                std::cout << "(ДАННЫЙ НОМЕР
ОТСУТСТВУЕТ В БАЗЕ)" << std::endl;
                                isKeysRelevant = false;
                        }
                }
                else
                {
                        std::cout << "(ПОЛЕ С НОМЕРОМ НЕ
ДОЛЖНО БЫТЬ ПУСТЫМ)" << std::endl;
                        isKeysRelevant = false;
```

```
                }


                if (isKeysRelevant)
                {
                    SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 15));
                    int count = scoutPath(subsBase,
stoull(numberA), stoull(numberB));
                    if (count != -1)
                    {
                        gotoxy(43, 7);
                        std::cout << std::setw(25) << std::left <<
count << std::endl;
                        gotoxy(0, 6);
                    }
                }
            }


            if ((GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0) && selectedMenuItem < 3)
                selectedMenuItem++;


            if (GetKeyState(VK_BACK) < 0)
```

```
                switch (selectedMenuItem)
                {
                case 1:
                    if (numberA.length())
                        numberA.pop_back();
                    gotoxy(43, 7);
                    std::cout << std::setw(25) << std::left <<
"НЕОПРЕДЕЛЕННО" << std::endl;
                    gotoxy(0, 4);
                    break;
                case 2:
                    if (numberB.length())
                        numberB.pop_back();
                    gotoxy(43, 7);
                    std::cout << std::setw(25) << std::left <<
"НЕОПРЕДЕЛЕННО" << std::endl;
                    gotoxy(0, 5);
                    break;
                }
            }
            while (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0
|| GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
            { }
```

```
            SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
            gotoxy(0, selectedMenuItem + 2);
            if (selectedMenuItem == 1)
                std::cout << std::setw(68) << std::left << ("  КЛЮЧ A: " +
formatNumberA) << std::endl;
            if (selectedMenuItem == 2)
                std::cout << std::setw(68) << std::left << ("  КЛЮЧ B: " +
formatNumberB) << std::endl;
            if (selectedMenuItem == 3)
                coutCenterStr("  РАССЧИТАТЬ");
            SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) |
15));

            if (_kbhit())
            {
                ch = _getche();
                gotoxy(0, wherey());
                std::cout << " ";
                gotoxy(0, wherey());
                if (ch == 0xE0 || ch == 0x00)
                {
                    ch = _getch();
```

```
                if (ch == 0x48 || ch == 0x50 || ch == 0x4B || ch ==
0x4D)
                        continue;
                }
                switch (selectedMenuItem)
                {
                case 1:
                        if (numberA.length() < 11 && ch > 47 && ch < 58)
                                numberA += ch;
                        gotoxy(43, 7);
                        std::cout << std::setw(25) << std::left <<
"НЕОПРЕДЕЛЕННО" << std::endl;
                        gotoxy(0, 4);
                        break;
                case 2:
                        if (numberB.length() < 11 && ch > 47 && ch < 58)
                                numberB += ch;
                        gotoxy(43, 7);
                        std::cout << std::setw(25) << std::left <<
"НЕОПРЕДЕЛЕННО" << std::endl;
                        gotoxy(0, 5);
                        break;
                }
```

```
            }


        }
}
void saveInFileMenu(int argc, char* argv[], bool& isSaved, HANDLE&
hConsoleHandle, BinSearchTree* subsBase)
{
        int selectedMenuItem = 1;
        system("cls");
        std::cout << "\n\n";
        if (subsBase)
        {
                if (!subsBase->mainRoot)
                {
                        coutCenterStr("  БИНАРНОЕ ДЕРЕВО (БАЗА
АБОНЕНТОВ) НЕ ИМЕЕТ ЗНАЧЕНИЙ");
                        std::cout << "\n\n";
                        pressAnyKeyToContinue();
                        return;
                }
        }
        else
        {
```

```
            coutCenterStr("  В ДАННЫЙ МОМЕНТ БИНАРНОЕ ДЕРЕВО (БАЗА АБОНЕНТОВ) НЕ ОБРАБАТЫВАЕТСЯ");
            std::cout << "\n\n";
            pressAnyKeyToContinue();
            return;
        }
        coutCenterStr("  ВВЕДИТЕ ПОЛНЫЙ ПУТЬ СОХРАНЯЕМОГО ФАЙЛА (БЕЗ ФОРМАТА)");
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        std::cout << std::setw(68) << std::left << "  ПУТЬ:" << std::endl;
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
        coutCenterStr("  ЗАПИСАТЬ");
        std::cout << std::endl << std::setw(68) << std::left << "  СТАТУС: БИНАРНОЕ ДЕРЕВО НЕ ЗАПИСАННО В ФАЙЛ" << std::endl;
        std::cout << "\n\n"
            << "  КЛАВИША UP - переключение на поле выше;" << std::endl
            << "  КЛАВИША DOWN - переключение на поле ниже;" << std::endl
            << "  КЛАВИША ENTER - подтвердить ввод;" << std::endl
            << "  КЛАВИША ESC - отменить ввод и вернуться в меню выбора действия;" << std::endl;

        std::string fullPath = argv[0];
```

```
        fullPath = fullPath.substr(0, fullPath.length() - 11);
        int ch;
        while (GetKeyState(VK_ESCAPE) >= 0)
        {
            if (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
            {
                gotoxy(0, wherey() - 1);
                SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 <<
4) | 15));
                if (selectedMenuItem == 1)
                    std::cout << std::setw(68) << std::left << ("  ПУТЬ: " +
fullPath) << std::endl;
                if (selectedMenuItem == 2)
                    coutCenterStr("  ЗАПИСАТЬ");
                if (GetKeyState(VK_UP) < 0 && selectedMenuItem > 1)
                    selectedMenuItem--;

                if (GetKeyState(VK_RETURN) < 0 && selectedMenuItem
== 2)
                {
                    Preinseq preinseq =
con_NLR_LNR_seq_of_BST(subsBase);
```

```
                bool fileStatus = make_preinseq_file(fullPath,
preinseq);
                if (!fileStatus)
                {
                        gotoxy(0, 5);
                        SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 4));
                        std::cout << std::endl << std::setw(68) <<
std::left << "  СТАТУС: ПРОИЗОШЛО ЧТО-ТО УЖАСНОЕ" << std::endl;
                        gotoxy(0, 5);
                        continue;
                }
                else
                {
                        gotoxy(0, 5);
                        SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 2));
                        std::cout << std::endl << std::setw(68) <<
std::left << "  СТАТУС: ФАЙЛ " + fullPath.substr(strlen(argv[0]) - 11,
fullPath.length()) + " ЗАПИСАН" << std::endl;
                        gotoxy(0, 5);
                        isSaved = true;
                        continue;
                }
```

```
                }


                if ((GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0) && selectedMenuItem < 3)
                        selectedMenuItem++;


                if (GetKeyState(VK_BACK) < 0)
                        switch (selectedMenuItem)
                        {
                        case 1:
                                if (fullPath.length())
                                        fullPath.pop_back();
                                break;
                        }
        }
        while (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0
|| GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
        { }
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        gotoxy(0, selectedMenuItem + 2);
        if (selectedMenuItem == 1)
                std::cout << std::setw(68) << std::left << ("  ПУТЬ: " +
fullPath) << std::endl;
```

```
if (selectedMenuItem == 2)
	coutCenterStr("  ЗАПИСАТЬ");
SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) |
15));


if (_kbhit())
{
	ch = _getche();
	gotoxy(0, wherey());
	std::cout << " ";
	gotoxy(0, wherey());
	if (ch == 0xE0 || ch == 0x00)
	{
		ch = _getch();
		if (ch == 0x48 || ch == 0x50 || ch == 0x4B || ch ==
0x4D)
			continue;
	}
	switch (selectedMenuItem)
	{
	case 1:
		if ((ch > 32 && ch < 127) || ch == ' ')
			fullPath += ch;
```

```
                    break;
                }
            }

        }
}
void openFromFileMenu(int argc, char* argv[], bool& isSaved, HANDLE& hConsoleHandle, BinSearchTree*& subsBase)
{
    int selectedMenuItem = 1;
    system("cls");
    std::cout << "\n\n";
    if (subsBase)
    {
        coutCenterStr("  В ДАННЫЙ МОМЕНТ УЖЕ ОБРАБАТЫВАЕТСЯ БИНАРНОЕ ДЕРЕВО");
        std::cout << "\n\n";
        pressAnyKeyToContinue();
        return;
    }
    coutCenterStr("  ВВЕДИТЕ ПОЛНЫЙ ПУТЬ ОТКРЫВАЕМОГО ФАЙЛА (БЕЗ ФОРМАТА)");
    SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
```

```
        std::cout << std::setw(68) << std::left << "  ПУТЬ:" << std::endl;
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) | 15));
        coutCenterStr("  ОТКРЫТЬ");
        std::cout << "\n\n";
        std::cout << "\n\n"
                << "  КЛАВИША UP - переключение на поле выше;" <<
std::endl
                << "  КЛАВИША DOWN - переключение на поле ниже;" <<
std::endl
                << "  КЛАВИША ENTER - подтвердить ввод;" << std::endl
                << "  КЛАВИША ESC - отменить ввод и вернуться в меню
выбора действия;" << std::endl;


        std::string fullPath = argv[0];
        fullPath = fullPath.substr(0, fullPath.length() - 11);
        int ch;
        while (GetKeyState(VK_ESCAPE) >= 0)
        {
                if (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
                {
                        gotoxy(0, wherey() - 1);
                        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 <<
4) | 15));
```

```
                if (selectedMenuItem == 1)
                        std::cout << std::setw(68) << std::left << ("  ПУТЬ: " +
fullPath) << std::endl;
                if (selectedMenuItem == 2)
                        coutCenterStr("  ОТКРЫТЬ");
                if (GetKeyState(VK_UP) < 0 && selectedMenuItem > 1)
                        selectedMenuItem--;


                if (GetKeyState(VK_RETURN) < 0 && selectedMenuItem
== 2)
                {
                        Preinseq preinseq = read_preinseq_file(fullPath);
                        if (preinseq.length == -1)
                        {
                                gotoxy(0, 5);
                                SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 4));
                                std::cout << std::endl << std::setw(68) <<
std::left << "  СТАТУС: ПРОИЗОШЛО ЧТО-ТО УЖАСНОЕ" << std::endl;
                                gotoxy(0, 5);
                                continue;
                        }
                        else
```

```
                {
                    if (subsBase && subsBase->mainRoot)
                        subsBase->clear();
                    if (!subsBase)
                        subsBase = new BinSearchTree{ NULL,
greaterInt };
                    if (preinseq.length > 0)
                        subsBase->insert(0, TeleSubscriber{ 0,
"PLACEHOLDER", "PLACEHOLDER" });
                    recon_BST_from_NLR_LNR_seq(subsBase-
>mainRoot, preinseq);

                    gotoxy(0, 5);
                    SetConsoleTextAttribute(hConsoleHandle,
(WORD)((0 << 4) | 2));
                    std::cout << std::endl << std::setw(68) <<
std::left << "  СТАТУС: ФАЙЛ " + fullPath.substr(strlen(argv[0]) - 11,
fullPath.length()) + " ПРОЧИТАН" << std::endl;
                    gotoxy(0, 5);
                    isSaved = false;
                    continue;
                }
            }
```

```
                if ((GetKeyState(VK_DOWN) < 0 ||
GetKeyState(VK_RETURN) < 0) && selectedMenuItem < 3)
                        selectedMenuItem++;


                if (GetKeyState(VK_BACK) < 0)
                        switch (selectedMenuItem)
                        {
                        case 1:
                                if (fullPath.length())
                                        fullPath.pop_back();
                                break;
                        }
        }
        while (GetKeyState(VK_UP) < 0 || GetKeyState(VK_DOWN) < 0
|| GetKeyState(VK_RETURN) < 0 || GetKeyState(VK_BACK) < 0)
        {
        }
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((7 << 4) | 0));
        gotoxy(0, selectedMenuItem + 2);
        if (selectedMenuItem == 1)
                std::cout << std::setw(68) << std::left << ("  ПУТЬ: " +
fullPath) << std::endl;
        if (selectedMenuItem == 2)
```

```
            coutCenterStr("  ОТКРЫТЬ");
        SetConsoleTextAttribute(hConsoleHandle, (WORD)((0 << 4) |
15));

        if (_kbhit())
        {
            ch = _getche();
            gotoxy(0, wherey());
            std::cout << " ";
            gotoxy(0, wherey());
            if (ch == 0xE0 || ch == 0x00)
            {
                ch = _getch();
                if (ch == 0x48 || ch == 0x50 || ch == 0x4B || ch ==
0x4D)
                    continue;
            }
            switch (selectedMenuItem)
            {
            case 1:
                if ((ch > 32 && ch < 127) || ch == ' ')
                    fullPath += ch;
                break;
```

```
                }
            }


        }
}
int countNodesBetweenByKeys(BinSearchTree* subsBase, KeyType keyA,
KeyType keyB)
{
        BinSearchTree::Node *nodeA, *nodeB;
        int count;
        for(count = -1, nodeA = subsBase->find(keyA), nodeB = subsBase-
>find(keyB); nodeA && nodeA != nodeB; nodeA=nodeA->parent, count++);
        if (nodeA)
                return count;
        for (count = -1, nodeB = subsBase->find(keyB), nodeA = subsBase-
>find(keyA); nodeB && nodeB != nodeA; nodeB = nodeB->parent, count++);
        if (nodeB)
                return count;
        return -1;
}
std::string formatNumber(std::string number)
{
        std::string formatNumber = "+_ (___) ___-__-__";
```

```
if (number.length() > 0)
        formatNumber[1] = number[0];
if (number.length() > 1)
        formatNumber[4] = number[1];
if (number.length() > 2)
        formatNumber[5] = number[2];
if (number.length() > 3)
        formatNumber[6] = number[3];
if (number.length() > 4)
        formatNumber[9] = number[4];
if (number.length() > 5)
        formatNumber[10] = number[5];
if (number.length() > 6)
        formatNumber[11] = number[6];
if (number.length() > 7)
        formatNumber[13] = number[7];
if (number.length() > 8)
        formatNumber[14] = number[8];
if (number.length() > 9)
        formatNumber[16] = number[9];
if (number.length() > 10)
        formatNumber[17] = number[10];
```

```
        return formatNumber;
    }
```