МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МО ЭВМ

ОТЧЕТ

по лабораторной работе №5
по дисциплине «Алгоритмы и структуры данных»
Тема: БДП и хеш-таблицы

Студент гр. 9381 Матвеев А. Н.

Преподаватель Фирсов М. А.

Санкт-Петербург
2020

Цель работы.

Изучение и реализация случайного бинарного дерева поиска.

Задание.

Вариант 9.

БДП: случайное* БДП; действие: 1+2в. **1.

**1 БДП должно быть реализовано на базе указателей, если в ЛР3 был в-вариант; БДП должно быть реализовано на базе массива, если в ЛР3 был д-вариант.

В вариантах заданий 2-ой группы (БДП и хеш-таблицы) требуется:

1) По заданной последовательности элементов Elem построить структуру данных определённого типа – БДП или хеш-таблицу;

2) Выполнить одно из следующих действий:

а) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem,

и если входит, то в скольких экземплярах. Добавить элемент e в структуру данных.

Предусмотреть возможность повторного выполнения с другим элементом.

б) Для построенной структуры данных проверить, входит ли в неё элемент e типа Elem,

и если входит, то удалить элемент e из структуры данных (первое обнаруженное

вхождение). Предусмотреть возможность повторного выполнения с другим элементом.

в) Записать в файл элементы построенного БДП в порядке их возрастания; вывести построенное БДП на экран в наглядном виде.

г) Другое действие.

Примечание.

Задача состоит в восстановлении случайного БДП по входной последовательности, формировании возрастающей последовательности, записи её в файл и выводе на экран БДП в наглядном виде. Также необходимо реализовать БДП на базе массива. В качестве элемента типа Elem выбран тип int.

Описание алгоритма.

Случайное бинарное дерево поиска - динамическая структура данных, представляющая собой бинарное дерево, в котором для каждого узла в левом поддереве находятся элементы, меньшие корня данного узла, а в правом - большие, причём в каждом узле есть счётчик попыток вставки данного элемента в узел. Структура случайного БДП полностью зависит от того («случайного») порядка, в котором элементы расположены во входной последовательности (во входном файле).

Построение случайного БДП по последовательности чисел.

В цикле поэлементно вводятся числа, и на каждой итерации запускается функция построения. Сначала узел дерева проверяется на пустоту. Если он пуст, то туда записывается текущий элемент из входной строки. Если же узел не пуст, то поданный элемент сравнивается с весом данного узла.

Если первый меньше, то функция проверяет левого сына данного узла на пустоту. Если левое поддерево пусто, то ему присваивается ссылка на некоторое место в массиве, задаваемое формулой 2*index + 1 (index - индекс текущего узла в массиве). Такая формула вставки нужна для избежания конфликтов при расстановке ссылок. Если не пусто, то сразу рекурсивно вызывается данная функция для левого поддерева (она вызывается в любом случае).

Если же поданный элемент больше веса узла, то функция проверяет правого сына данного узла на пустоту. Если правое поддерево пусто, то ему

присваивается ссылка на некоторое место в массиве, задаваемое формулой 2*index + 2 (index - индекс текущего узла в массиве). Если не пусто, то сразу рекурсивно вызывается данная функция для правого поддерева. (она вызывается в любом случае).

Если элемент и вес узла совпадают, то просто увеличивается на 1 счетчик попыток вставки данного элемента в данный узел.

1) Построение последовательности элементов случайного БДП в порядке возрастания.

Т.к. в левом поддереве для каждого узла находятся меньшие элементы, а в правом - большие, чтобы сформировать элементы случайного БДП в порядке возрастания, достаточно обойти дерево в порядке левый -> корень -> правый, записывая в выходной массив элементы. Стоит отметить, что условие захода в узел - счётчик попыток вставки в этот узел != 0 (проверка узла на пустоту в массиве).

2) Вывод дерева на экран в наглядном виде.

Это тоже разновидность обхода дерева, но на этот раз в формате: правый -> корень -> левый. Сначала выполняется рекурсивный заход в правое поддерево для каждого узла, затем выводится корень с количеством отступов, пропорциональных глубине рекурсии, после чего рекурсивно выполняется обход левого поддерева для каждого узла. На выходе получается наглядное БДП, повёрнутое на 90 градусов влево.

Описание классов.

Ключевой класс - BinSTree. Представляет собой случайное бинарное дерево поиска. Имеет приватные поля int count (количество попыток вставки), Elem info (вес узла - целое число) , int leftTree (ссылка на левое поддерево, представленная индексом массива), int rightTree (ссылка на правое поддерево, представленная индексом массива). В качестве публичных методов помимо конструктора и деструктора по умолчанию реализованы статические методы,

которые и представляют собой набор необходимых функций, нужных для работы программы. Среди них:

- static void searchAndInsert(Elem x, int index, ostream & out, int n) - функция, позволяющая по записи построить случайное БДП;
- static void printBST(int index, int move, std::ostream &out) - функция, изображающая бинарное дерево поиска в наглядном виде;
- static void getSeqenceOfElements(int index, std::ostream &out, vector<int> & sequence, int n) - функция, создающая последовательность элементов БДП в порядке возрастания;
- static void printIndent(int n, std::ostream & out) - функция, печатающая необходимое количество отступов (нужна в отладочных выводах).

Также имеется пространство имён vectorBST, в котором находится экземпляр библиотечного вектора, представляющий собой в данной реализации массив элементов типа BinSTree. Хранит узлы БДП.

Описание функций.

static void searchAndInsert(Elem x, int index, ostream & out, int n) - функция, позволяющая по записи построить случайное БДП. Принимает на вход текущий элемент x(который предполагается вставить), индекс массива index (ссылку на узел, начиная с которой происходит обработка дерева), ссылку на поток вывода out и целое число n, отражающее нужное количество отступов при рекурсивном выводе. Этот статический метод ничего не возвращает.

Стоит заметить, что вектор на протяжении всего хода программы содержит некоторое количество нулевых узлов (все поля узла = 0 изначально), и если текущий индекс при заполнении больше вместимости вектора, то к вектору применяется метод resize() и вместимость увеличивается на 20 элементов.

Если поле count текущего узла нулевое (изначально все поля узла нулевые), то оно устанавливается в 1 и полю info присваивается поданный элемент (далее - x).

Иначе если x < info, следует зайти в левое поддерево и проверить его на возможность вставки (эта функция вызывается рекурсивно для левого поддерева). Причём если поле leftTree не инициализировано (= 0), его надо инициализировать индексом массива по формуле 2*index + 1, чтобы не возникало конфликтов перезаписи ссылок на узлы. (index - ссылка, представленная индексом, на текущий узел).

Если x > info, указанные выше действия выполняются для правого поддерева и они совершенно аналогичны за тем исключением, что инициализация rightTree происходит по формуле 2*index + 2.

В случае если значение x совпало с значением узла, то поле узла count увеличивается на 1.

static void printBST(int index, int move, std::ostream &out) - функция, изображающая бинарное дерево поиска в наглядном виде. Принимает на вход индекс массива index (ссылку на узел, начиная с которого производится обход), целочисленную переменную move, отвечающую за количество отступов при визуализации БДП и ссылку на поток вывода out. Ничего не возвращает. Если индекс больше размера массива, то происходит возврат из функции. Если поле count не равно 0 для конкретного узла, то, при ненулевом левом или правом поддереве выполняется обход сначала правого поддерева (функция вызывает саму себя для поля rightTree), затем выводится корень с соответствующим количеством отступов, а потом производится обход левого поддерева (функция вызывает саму себя для поля leftTree).

static void getSeqenceOfElements(int index, std::ostream &out, vector<int> & sequence, int n) - функция, создающая последовательность элементов БДП в порядке возрастания. Принимает на вход индекс массива index (ссылку на узел, начиная с которого производится обход), ссылку на поток вывода out, ссылку на экземпляр библиотечного вектора для элементов типа int, в который будут записаны элементы БДП в порядке возрастания (sequence), целочисленную переменную n. Ничего не возвращает. Если индекс превосходит количество элементов в массиве, выводится сообщение об ошибке.

Далее для каждого узла рекурсивно обходится левое поддерево (если leftTree != 0), после чего в конец вектора-последовательности записывается значение узла, после чего обходится правое поддерево (если rightTree != 0). После завершения работы функции в векторе sequence находятся элементы БДП в порядке возрастания.

static void printIndent(int n, std::ostream & out) - печать отступов в отладочных выводах. Принимает на вход целое число, отвечающее за количество отступов и ссылку на поток вывода. Ничего не возвращает.

В функции **main()** пользователю предоставляется возможность ввести данные с файла или с консоли. Обязательно требуется указать путь до выходного файла, куда будет записана возрастающая последовательность (этого требует формулировка задания). Предусмотрена обработка некоторых ошибок (таких как некорректный входной путь или некорректная входная последовательность).

После того как пользователь ввёл данные с консоли в строку elements (или же они были считаны из файла), они записываются в строковый поток strStream, из которого поэлементно считываются значения в цикле while и подаются на вход функции searchAndInsert(), заполняющей ими БДП. После этого вызывается getSeqenceOfElements(), которая записывает элементы БДП в вектор sequence, после чего в цикле они выводятся в выходной файл (это обязательное требование задания) и в консоль для удобства. После этого запускается printBST(), дерево выводится на экран в наглядном виде, после чего программа завершается.

Также имеется пространство имён file, где объявлены потоки ввода из файла и записи в файл.

Исходный код программы смотреть в Приложении А.

Тестирование.

Результаты тестирования представлены в таблице 1.

Таблица 1. – Результаты тестирования

№ п/п	Входные данные	Выходные данные
1.	1	Обработка элемента (1) Обнаружен свободный узел. Вставка элемента (1) Процесс получения последовательности элементов дерева в порядке возрастания: Значение текущего узла (1) Элементы случайного БДП в порядке возрастания: 1 Случайное БДП имеет вид: 1
2.	5 6 1 3 4 5	Обработка элемента (5) Обнаружен свободный узел. Вставка элемента (5) Обработка элемента (6) Элемент (6) > (5) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 2 Обнаружен свободный узел. Вставка элемента (6) Обработка элемента (1) Элемент (1) < (5) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 1 Обнаружен свободный узел. Вставка элемента (1) Обработка элемента (3) Элемент (3) < (5) - значения узла. Переход к левому поддереву. Элемент (3) > (1) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 4

		Обнаружен свободный узел. Вставка элемента (3) Обработка элемента (4) Элемент (4) < (5) - значения узла. Переход к левому поддереву. Элемент (4) > (1) - значения узла. Переход к правому поддереву. Элемент (4) > (3) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 10 Обнаружен свободный узел. Вставка элемента (4) Обработка элемента (5) Такой элемент уже есть в дереве и встречался уже 1 раз(-а). Процесс получения последовательности элементов дерева в порядке возрастания: Обнаружено непустое левое поддерево -> Переход по ссылке 1 Значение текущего узла (1) Обнаружено непустое правое поддерево -> Переход по ссылке 4 Значение текущего узла (3) Обнаружено непустое правое поддерево -> Переход по ссылке 10 Значение текущего узла (4) Значение текущего узла (5) Обнаружено непустое правое поддерево -> Переход по ссылке 2 Значение текущего узла (6) Элементы случайного БДП в порядке возрастания: 1 3 4 5 6

		Случайное БДП имеет вид: __6 5 _____4 ____3 __1
3.	5 2 10 3 6 96 78 0	Обработка элемента (5) Обнаружен свободный узел. Вставка элемента (5) Обработка элемента (2) Элемент (2) < (5) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 1 Обнаружен свободный узел. Вставка элемента (2) Обработка элемента (10) Элемент (10) > (5) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 2 Обнаружен свободный узел. Вставка элемента (10) Обработка элемента (3) Элемент (3) < (5) - значения узла. Переход к левому поддереву. Элемент (3) > (2) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 4 Обнаружен свободный узел. Вставка элемента (3) Обработка элемента (6) Элемент (6) > (5) - значения узла. Переход к правому поддереву.

		Элемент (6) < (10) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 5 Обнаружен свободный узел. Вставка элемента (6) Обработка элемента (96) Элемент (96) > (5) - значения узла. Переход к правому поддереву. Элемент (96) > (10) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 6 Обнаружен свободный узел. Вставка элемента (96) Обработка элемента (78) Элемент (78) > (5) - значения узла. Переход к правому поддереву. Элемент (78) > (10) - значения узла. Переход к правому поддереву. Элемент (78) < (96) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 13 Обнаружен свободный узел. Вставка элемента (78) Обработка элемента (0) Элемент (0) < (5) - значения узла. Переход к левому поддереву. Элемент (0) < (2) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 3 Обнаружен свободный узел. Вставка элемента (0)

		Процесс получения последовательности элементов дерева в порядке возрастания: Обнаружено непустое левое поддерево -> Переход по ссылке 1 Обнаружено непустое левое поддерево -> Переход по ссылке 3 Значение текущего узла (0) Значение текущего узла (2) Обнаружено непустое правое поддерево -> Переход по ссылке 4 Значение текущего узла (3) Значение текущего узла (5) Обнаружено непустое правое поддерево -> Переход по ссылке 2 Обнаружено непустое левое поддерево -> Переход по ссылке 5 Значение текущего узла (6) Значение текущего узла (10) Обнаружено непустое правое поддерево -> Переход по ссылке 6 Обнаружено непустое левое поддерево -> Переход по ссылке 13 Значение текущего узла (78) Значение текущего узла (96) Элементы случайного БДП в порядке возрастания: 0 2 3 5 6 10 78 96 Случайное БДП имеет вид: ____96 ______78 __10 ____6 5 ____3

		__2 ____0
4.	1 -9 5 3 -80 53 60 -20	Обработка элемента (1) Обнаружен свободный узел. Вставка элемента (1) Обработка элемента (-9) Элемент (-9) < (1) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 1 Обнаружен свободный узел. Вставка элемента (-9) Обработка элемента (5) Элемент (5) > (1) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 2 Обнаружен свободный узел. Вставка элемента (5) Обработка элемента (3) Элемент (3) > (1) - значения узла. Переход к правому поддереву. Элемент (3) < (5) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 5 Обнаружен свободный узел. Вставка элемента (3) Обработка элемента (-80) Элемент (-80) < (1) - значения узла. Переход к левому поддереву. Элемент (-80) < (-9) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 3

		Обнаружен свободный узел. Вставка элемента (-80) Обработка элемента (53) Элемент (53) > (1) - значения узла. Переход к правому поддереву. Элемент (53) > (5) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 6 Обнаружен свободный узел. Вставка элемента (53) Обработка элемента (60) Элемент (60) > (1) - значения узла. Переход к правому поддереву. Элемент (60) > (5) - значения узла. Переход к правому поддереву. Элемент (60) > (53) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 14 Обнаружен свободный узел. Вставка элемента (60) Обработка элемента (-20) Элемент (-20) < (1) - значения узла. Переход к левому поддереву. Элемент (-20) < (-9) - значения узла. Переход к левому поддереву. Элемент (-20) > (-80) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 8 Обнаружен свободный узел. Вставка элемента (-20)

		Процесс получения последовательности элементов дерева в порядке возрастания: Обнаружено непустое левое поддерево -> Переход по ссылке 1 Обнаружено непустое левое поддерево -> Переход по ссылке 3 Значение текущего узла (-80) Обнаружено непустое правое поддерево -> Переход по ссылке 8 Значение текущего узла (-20) Значение текущего узла (-9) Значение текущего узла (1) Обнаружено непустое правое поддерево -> Переход по ссылке 2 Обнаружено непустое левое поддерево -> Переход по ссылке 5 Значение текущего узла (3) Значение текущего узла (5) Обнаружено непустое правое поддерево -> Переход по ссылке 6 Значение текущего узла (53) Обнаружено непустое правое поддерево -> Переход по ссылке 14 Значение текущего узла (60) Элементы случайного БДП в порядке возрастания: -80 -20 -9 1 3 5 53 60 Случайное БДП имеет вид: _____60 ____53 __5 ____3 1 __-9 _____-20

		____-80
5.	3 -1 0 0 0	Обработка элемента (3) Обнаружен свободный узел. Вставка элемента (3) Обработка элемента (-1) Элемент (-1) < (3) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 1 Обнаружен свободный узел. Вставка элемента (-1) Обработка элемента (0) Элемент (0) < (3) - значения узла. Переход к левому поддереву. Элемент (0) > (-1) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 4 Обнаружен свободный узел. Вставка элемента (0) Обработка элемента (0) Элемент (0) < (3) - значения узла. Переход к левому поддереву. Элемент (0) > (-1) - значения узла. Переход к правому поддереву. Такой элемент уже есть в дереве и встречался уже 1 раз(-а). Обработка элемента (0) Элемент (0) < (3) - значения узла. Переход к левому поддереву. Элемент (0) > (-1) - значения узла. Переход к правому поддереву. Такой элемент уже есть в дереве и встречался уже 2 раз(-а).

		Процесс получения последовательности элементов дерева в порядке возрастания: Обнаружено непустое левое поддерево -> Переход по ссылке 1 Значение текущего узла (-1) Обнаружено непустое правое поддерево -> Переход по ссылке 4 Значение текущего узла (0) Значение текущего узла (3) Элементы случайного БДП в порядке возрастания: -1 0 3 Случайное БДП имеет вид: 3 ____0 __-1
6.	2 4 -9	Обработка элемента (2) Обнаружен свободный узел. Вставка элемента (2) Обработка элемента (4) Элемент (4) > (2) - значения узла. Переход к правому поддереву. Ссылка на правое поддерево текущего корня теперь равна 2 Обнаружен свободный узел. Вставка элемента (4) Обработка элемента (-9) Элемент (-9) < (2) - значения узла. Переход к левому поддереву. Ссылка на левое поддерево текущего корня теперь равна 1 Обнаружен свободный узел. Вставка элемента (-9) Процесс получения последовательности элементов дерева в порядке возрастания:

		Обнаружено непустое левое поддерево -> Переход по ссылке 1 Значение текущего узла (-9) Значение текущего узла (2) Обнаружено непустое правое поддерево -> Переход по ссылке 2 Значение текущего узла (4) Элементы случайного БДП в порядке возрастания: -9 2 4 Случайное БДП имеет вид: __4 2 __-9

Тестирование на некорректных данных.

Результаты тестирования на некорректных данных представлены в таблице 2.

Таблица 2 - результаты тестирования на некорректных данных

№ п/п	Входные данные	Выходные данные
1.	1+-2	Во входной строке есть недопустимые символы
2.	цкущпьтцуп	Во входной строке есть недопустимые символы
3.	Выберите способ ввода данных: 1. Из консоли 2. Из файла (нажмите 1 или 2) **2** Введите путь до входного файла: **rjg**	Не удалось открыть файл

Выводы.

Изучена и реализована в коде структура случайного бинарного дерева поиска, а также функции для работы с ней.

ПРИЛОЖЕНИЕ А
ИСХОДНЫЙ КОД ПРОГРАММЫ

Файл main.cpp:

```
#include <iostream>
#include <cstdlib>
#include <fstream>
#include <string>
#include <sstream>
#include <vector>

using namespace std;
typedef int Elem;
class BinSTree;

namespace vectorBST{ // пространство имён, содержащее вектор, в котором будут храниться элементы БДП
    vector<BinSTree> array;
}



class BinSTree{// случайное бинарное дерево поиска
private:
    int count = 0; // количество попыток вставки
    Elem info = 0;// данные узла
    int leftTree = 0; // левое поддерево
    int rightTree = 0; // правое поддерево

public:
    BinSTree() = default;// конструктор класса БД
    ~BinSTree() = default; // деструктор класса БД
     static void searchAndInsert(Elem x, int index, ostream & out, int n) { // функция, позволяющая по записи построить случайное БДП
        using namespace vectorBST;

        if(index >= array.capacity()) {
            array.resize(index + 20); // если требуется, размер массива увеличивается
        }
```

```
        BinSTree & tmp = array.at(index);
        if (tmp.count == 0){ // если поле count текущего узла = 0 (узел свободен), то инициализируем его поданным на вход элементом
            printIndent(n, out);
             out << "Обнаружен свободный узел. Вставка элемента (" << x << ")\n";
            tmp.count = 1;
            tmp.info = x;
        }
          else if(x < tmp.info){ // если поданное значение < значения узла, то переходим к его левому сыну
            printIndent(n, out);
               out << "Элемент (" << x << ") < (" << tmp.info << ") - значения узла. Переход к левому поддереву.\n";
               if(tmp.leftTree == 0) { /* если левого сына ещё нет, то даём узлу ссылку в виде индекса на место в векторе, где будет находиться
                 его левое поддерево , в массиве эта ссылка задаётся формулой 2*(индекс узла) + 1 */
                tmp.leftTree = 2 * index + 1;
                printIndent(n, out);
                out << "Ссылка на левое поддерево текущего корня теперь равна " << tmp.leftTree << "\n";
            }
                   searchAndInsert(x, tmp.leftTree, out, n + 1); // рекурсивный вызов для левого сына текущего узла
        }
          else if(x > tmp.info){ // если поданное значение > значения узла, то переходим к его правому сыну
            printIndent(n, out);
               out << "Элемент (" << x << ") > (" << tmp.info << ") - значения узла. Переход к правому поддереву.\n";
               if(tmp.rightTree == 0){/* если правого сына ещё нет, то даём узлу ссылку в виде индекса на место в векторе, где будет находиться
                 его правое поддерево , в массиве эта ссылка задаётся формулой 2*(индекс узла) + 2 */
                tmp.rightTree = 2 * index + 2;
                printIndent(n, out);
```

```
                    out << "Ссылка на правое поддерево текущего корня
теперь равна " << tmp.rightTree << "\n";
            }
                searchAndInsert(x, tmp.rightTree, out, n + 1); //
рекурсивный вызов для правого сына текущего узла
        }
        else{ // поданное значение уже есть в дереве
            printIndent(n, out);
             out << "Такой элемент уже есть в дереве и встречался уже "
<< tmp.count << " раз(-а).\n";
              array.at(index).count++; // увеличиваем счётчик попыток
вставки на 1
        }
    }

     static void printBST(int index, int move, std::ostream &out) //
функция, изображающая бинарное дерево поиска (повернутое) в наглядном
виде
    { // БДП выводится справа налево с необходимым количеством отступов
        // это позволяет представить БДП в привычном виде
         // эта функция также изображает обход дерева в порядке правый
-> корень -> левый
        using namespace vectorBST;
        if(index >= array.size())
            return;
        BinSTree & tmp = vectorBST::array.at(index);
        if(tmp.count != 0) { // если текущий узел непуст
                if(tmp.rightTree != 0) // если правый сын узла непуст,
производится рекурсивный обход правого поддерева
                printBST(tmp.rightTree, move + 2, out);
                    for (int i = 0; i < move; i++)
                        out << "_";
                         out << tmp.info << std::endl; // когда правое
дерево полностью просмотрено, выводится корень
                 if(tmp.leftTree != 0) // если левый сын узла непуст,
производится рекурсивный обход левого поддерева
                printBST(tmp.leftTree, move + 2, out);
        }
    }
```

```
      static void getSeqenceOfElements(int index, std::ostream &out,
vector<int> & sequence, int n) { // функция, создающая
последовательность элементов БДП в порядке возрастания
        using namespace vectorBST;
        if(index >= array.size()){
            printIndent(n, out);
                 out << "Индекс превосходит размер массива - Действие
невозможно\n";
            return;
        }
         /* т.к. в левом поддереве узла находятся меньшие элементы, а в
правом - большие, чтобы сформировать
            элементы  случайного БДП в порядке возрастания, достаточно
обойти дерево в порядке левый -> корень -> правый */
        BinSTree & tmp = vectorBST::array.at(index);
        if(tmp.count != 0) {  // если узел непуст
              if(tmp.leftTree != 0) { // если левое поддерево непусто,
данная функция запускается рекурсивно в левом поддереве
                printIndent(n, out);
                  out << "Обнаружено непустое левое поддерево -> Переход
по ссылке " << tmp.leftTree << endl;
                   getSeqenceOfElements(tmp.leftTree, out, sequence, n +
1);
            }
            printIndent(n, out);
            out << "Значение текущего узла ("<< tmp.info << ")\n";
                      sequence.push_back(tmp.info);  //  найденный  узел
записывается в конец последовательности
             if(tmp.rightTree != 0) { // если правое поддерево непусто,
данная функция запускается рекурсивно в правом поддереве
                printIndent(n, out);
                 out << "Обнаружено непустое правое поддерево -> Переход
по ссылке " << tmp.rightTree << endl;
                  getSeqenceOfElements(tmp.rightTree, out, sequence, n +
1);
            }
        }
    }
```

```
      static void printIndent(int n, std::ostream & out){ // печать
отступов (нужно в отладочных выводах)
        for(int i = 0; i < n; i++ )
            out << "\t";
    }

};



namespace file{
    ifstream inFile;
    ofstream outFile;
}

int main()
{
    int flag; // флаг, определяющий источник ввода
    string path; // путь до файла
    string elements; // входная строка
       cout << "Выберите способ ввода данных:\n1. Из консоли\n2. Из
файла\n(нажмите 1 или 2)\n";
    cin >> flag;
    switch(flag){
        case 1:
            cout << "Введите элементы БДП через пробел:\n";
            break;
        case 2:
            cout << "Введите путь до входного файла:\n";
            cin >> path;
            file::inFile.open(path);
            if(!file::inFile.is_open()) {
                cerr << "Не удалось открыть файл\n";
                return 0;
            }
            break;
```

```
        default:
            cerr << "Некорректный ввод\n";
            return 0;
    }
    istream & in = (flag == 1) ? cin : file::inFile;
    if(flag == 1)
        in.ignore();

    MaybeRepeat:
    getline(in, elements);

    for(char element : elements){
        if(!isdigit(element) && !isspace(element) && element != '-'){
            cerr << "Во входной строке есть недопустимые символы\n";
            return 0;
        }
    }

     if(elements.empty() && flag == 1){ // повторная попытка продолжить работу программы допускается, только если пользователь забыл ввести строку элементов
        cerr << "Вы не ввели ни одного символа\n";
        goto MaybeRepeat;
    }
     else if(elements.empty() && flag == 2){ // если входной файл пуст, то программа завершается
        cerr << "Файл пуст\n";
        return 0;
    }

      cout << "Введите путь до выходного файла (куда будет записана последовательность элементов БДП в порядке возрастания):\n";
    cin >> path;
    file::outFile.open(path);
    if(!file::outFile.is_open()){
        cerr << "Не удалось открыть файл\n";
        return 0;
    }
    ostream & out = cout;
```

```
    int x;
    unsigned int index = 0;
        stringstream   strStream; // строковый поток для упрощения поэлементного считывания исходной последовательности
    strStream << elements;
    while(strStream >> x) {
        out << "Обработка элемента (" << x << ")\n";
           BinSTree::searchAndInsert(x, 0, out, 0); // на каждом шаге запускается функция заполнения и дерево пополняется на 1 элемент
    }
     out << "\nПроцесс получения последовательности элементов дерева в порядке возрастания:\n";
    vector <int> sequence;
     BinSTree::getSeqenceOfElements(0, out, sequence, 0); // в sequence будет записана последовательность элементов в порядке возрастания
    out << "Элементы случайного БДП в порядке возрастания: ";
    for(auto it : sequence){
          file::outFile << it << " "; // запись в последовательности в файл согласно заданию
        out << it << " ";
    }
    out << "\n\nСлучайное БДП имеет вид:\n";
     BinSTree::printBST(0, 0, out); // вывод БДП на экран в наглядном виде согласно заданию
    vectorBST::array.clear();
    return 0;
}
```