|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ Информатика и системы управления

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ № 6**

**«Деревья»**

Вариант 6

Студент Пермякова Екатерина Дмитриевна

Группа ИУ7 – 32Б

Преподаватель Силантьева Александра Васильевна

2023 г.

Оглавление

[Условие задачи 3](#_Toc151936871)

[Техническое задание 4](#_Toc151936872)

[Описание внутренних структур данных 6](#_Toc151936873)

[Алгоритм 8](#_Toc151936874)

[Эффективность 11](#_Toc151936875)

[Вывод 12](#_Toc151936876)

[Ответы на вопросы 12](#_Toc151936877)

# Условие задачи

Построить дерево в соответствии со своим вариантом задания. Вывести его на экран в виде дерева. Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. Сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления.

Ввести значения переменных: от A до I. Построить и вывести на экран бинарное дерево следующего выражения: A + (B \* (C + (D \* (E + F) - (G - H)) + I)). Написать процедуры постфиксного, инфиксного и префиксного обхода дерева и вывести соответствующие выражения на экран. Подсчитать результат. Используя «польскую» запись, ввести данное выражение в стек. Сравнить время вычисления выражения с использованием дерева и стека.

# Техническое задание

***Описание программы:***

Вводное окно:

|  |
| --- |
| Программа производит работу бинарным деревом поиска и  деревом выражения A + (B \* (C + (D \* (E + F) - (G - H)) + I))  Возможности программы:  1 - Работать со своим бинарным деревом поиска  2 - Эффективность функций создания и поиска в бинарном дереве поиска  3 - Работать с деревом выражения  0 - Завершить программу  Введите номер действия, которого хотите совершить: |

Пользователю надо выбрать номер действия, которое он хочет совершить.

***Результат выбора действия:***

Описание программы в зависимости от выбора номера действия:

**Действие 1:**

***Исходные данные:***

Номер действия с деревом, и если потребуется данные узла дерева

***Результат:***

|  |
| --- |
| Действия с бинарным деревом поиска целых чисел:  1 - Добавить узел  2 - Удалить узел по содержимому  3 - Найти элемент по содержимому  4 - Показать дерево  5 - Очистить дерево  0 - Завершить работу  Введите номер действия, которого хотите совершить: |

1. Добавляет узел дерева с данными, которые введет пользователь.
2. Удаляет элемент дерева, соответствующий содержимому которое введет пользователь
3. Ищет элемент дерева по введенным пользователем данным
4. Выводит дерево в префиксной форме и записывает данные о нем в файл graph/graph.gv для дальнейшего преобразования его через graphviz
5. Очищает дерево
6. Выводит из опции работы с бинарным деревом поиска

**Действие 2:**

нет

***Результат:***

Выводит таблицу эффективности алгоритмов заполнения дерева (аналог сортировки) и поиска в дереве в зависимости от высоты дерева и его ветсвистости.

**Действие 3:**

***Исходные данные:***

Числа, соответствующие буквам в выражении A + (B \* (C + (D \* (E + F) - (G - H)) + I))

***Результат:***

Считает результат выражения с введенными числами, выводит выражение в постфиксной, инфиксной и префиксной форме, записывает данные о дереве выражения в файл graph/exp\_graph.gv для дальнейшего преобразования его через graphviz, выводит таблицу эффективности подсчета выражения используя дерево и стек

**Действие 0:**

***Исходные данные:*** нет

***Результат:***

Завершается программа

***Описание задачи, реализуемой программой:***

Производит действия, описанные в предыдущем пункте при работе с бинарным деревом поиска и деревом указанного выражения.

***Способ обращения к программе:***

После запуска программы ./app.exe пользователь выбирает номер действия, которое хочет выполнить (нумерация действий описана в программе) и далее взаимодействует с программой в зависимости от выбранного действия.

***Описание возможных аварийных ситуаций и ошибок пользователя:***

|  |  |
| --- | --- |
| **Ошибка** | **Вывод программы** |
| Неверный ввод данных в узел | ERR\_DATA: ошибка в данных узла списка |
| Попытка добавить в бинарное дерево поиска повторяющийся элемент | ERR\_NODE\_EXIST: такой узел уже существует |
| Ошибка записи в файл | ERR\_FILE: ошибка записи в файл |
| Не найден искомый элемент | ERR\_NOT\_FOUND: элемент не найден |
| введен неверный номер действия | ERR\_NUM\_ACT\_INPUT: введен неверный номер действия |
| Попытка удалить элемент из пустого дерева | ERR\_EMPTY\_TREE: дерево пустое |

# Описание внутренних структур данных

Бинарное дерево поиска

|  |
| --- |
| typedef struct st\_tree\_t node\_t;  struct st\_tree\_t  {  int data;  node\_t \*left;  node\_t \*right;  }; |

Где:

* data – целое число
* left – указатель на левого предка
* right – указатель на правого предка

Дерево выражения

|  |
| --- |
| #define PLUS\_EXP '+'  #define MINUS\_EXP '-'  #define MULT\_EXP '\*'  typedef enum { EXP, NUM } type\_data\_t;  typedef struct st\_exp\_tree\_t enode\_t;  struct st\_exp\_tree\_t  {  type\_data\_t type;  union data\_t  {  int num;  char exp;  } data;  char id[2];  enode\_t \*left;  enode\_t \*right;  }; |

Где:

* type – тип данных
* data (num, exp) – данные (число или выражение (PLUS\_EXP, MINUS\_EXP, MULT\_EXP))
* id - неповторяющаяся строка описание содержимого (для числе это буква которая им соответствует в маске выражения, для выражения (+ - \*) это его номер). id нужен для того чтобы graphviz мог строить дерево с повторяющимися элементами
* left – указатель на левого предка
* right – указатель на правого предка

Список на основе, которого сделан стек

|  |
| --- |
| typedef struct st\_list\_node\_t lnode\_t;  struct st\_list\_node\_t  {  type\_data\_t type;  union st\_data\_t  {  int num;  char exp;  } data;  lnode\_t \*next;  }; |

Где:

* type – тип данных
* data (num, exp) – данные (число или выражение (PLUS\_EXP, MINUS\_EXP, MULT\_EXP))
* next – указатель на следующий элемент

# Алгоритм

**Алгоритм добавления элемента в бинарное дерево поиска**

Выделяет память под новый узел. Если корень указывает на NULL, то новый узел становится корнем. Если нет, то заводится временная переменная - указатель на узел списка, с помощью которой мы проходимся по узлам дерева и сравниваем содержимое с новым узлом, если новый узел больше данного, то переходим к правому потомку, если меньше к левому, если равен, то выкидывается ошибка, и так пока не дойдем до листа и уже к нему прикрепим новый узел.

**Алгоритм поиска узла в дереве**

Этот алгоритм похож на алгоритм добавления узла, когда мы ищем конец ветки, только здесь мы ищем узел с таким же содержимым, и если доходим до конца ветки так и не найдя его, то такого узла в дереве нет

**Алгоритм удаления элемента в узле дерева**

Находим удаляемый элемент с помощью функции поиска. Если у этого элемента предков нет, то просто удаляем его. Если у него 1 предок, то этот предок встанет на место удаляемого элемента. Если есть 2 предка, то в правом поддереве ищем минимальный элемент и ставим его на место удаляемого.

**Алгоритм обхода дерева**

Рекурсивно запускаем функцию обхода дерева. Если переданный ей указатель на узел указывает на NULL, то выходим из нее, иначе обрабатываем переданный указатель, потом рекурсивно запускаем эту же функцию обхода для левого и правого предка переданного узла. Порядок обработки зависит от порядка обхода, здесь был описан префиксный.

**Алгоритм построения дерева выражения**

В массив целых чисел записываются введенные пользователем данные. Потом на основе этого массива строится дерево, порядок каждого узла прописан внутри программы.

**Алгоритм подсчета значения выражения через стек**

В постфиксной форме в стек подаются данные из списка. Если встречается число, то ого записывается в стек, если операнд, то берется 2 числа в обратном порядке и считаются с использованием этого операнда, получившееся число записывается обратно в стек. В конце обработки всех данных из списка, в стеке остается 1 число - результат подсчета.

**Алгоритм подсчета значения выражения через дерево**

Осуществляется рекурсивный обход дерева, если левый и правый предок узла числа, то сам узел содержит операнд, на основе этого считается выражение и в узел на место операнда записывается значение выражения. После обработки всего дерева, в его корне будет лежать результат подсчета

**Алгоритм определения времени работы функции**

Создаем массив целых числе t, в который будем записывать результат определения времени работы функции рассматриваемой функции

Записываем результат определения времени работы функции при одних и тех же параметрах минимальное количество раз, определенное как константа внутри программы (в данном случае 100 раз)

На основе данных их массива t, вычисляем RSE

Если RSE > 5%, то повторяем замер времени пока она не станет меньше 5%

В итоге временем работы функции для данных параметром считается среднее значение данных в массиве t

При вычислении времени работы функции поиска, искомый элемент всегда являлся листом дерева

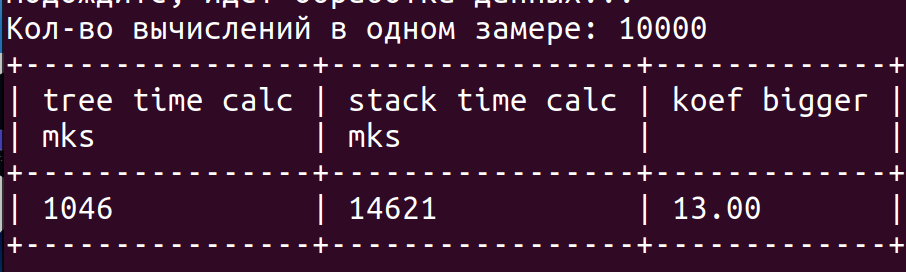
**Тесты:**

Тесты ввода числа:

|  |  |  |
| --- | --- | --- |
| **Ввод** | **Вывод** | **Что проверяется** |
| 1  1 2  1 1  1 3  3 3  4  5 | Добавлен новый узел с содержимым 2  Добавлен новый узел с содержимым 1  Добавлен новый узел с содержимым 3  Найден элемент - 3, его предок - 2  Дерево в префиксной форме: 2 1 3  Данные о дереве записаны в файл graph/graph.gv | Создали дерево  Нашли его элемент  Очистили дерево |
| 2 | Таблица с оценкой эффективности работы функции создания дерева и поиска в нем | Оценка эффективности |
| 3 |  | Работа с деревом выражения |
| 0 | Программа завершена | Выход из программы |

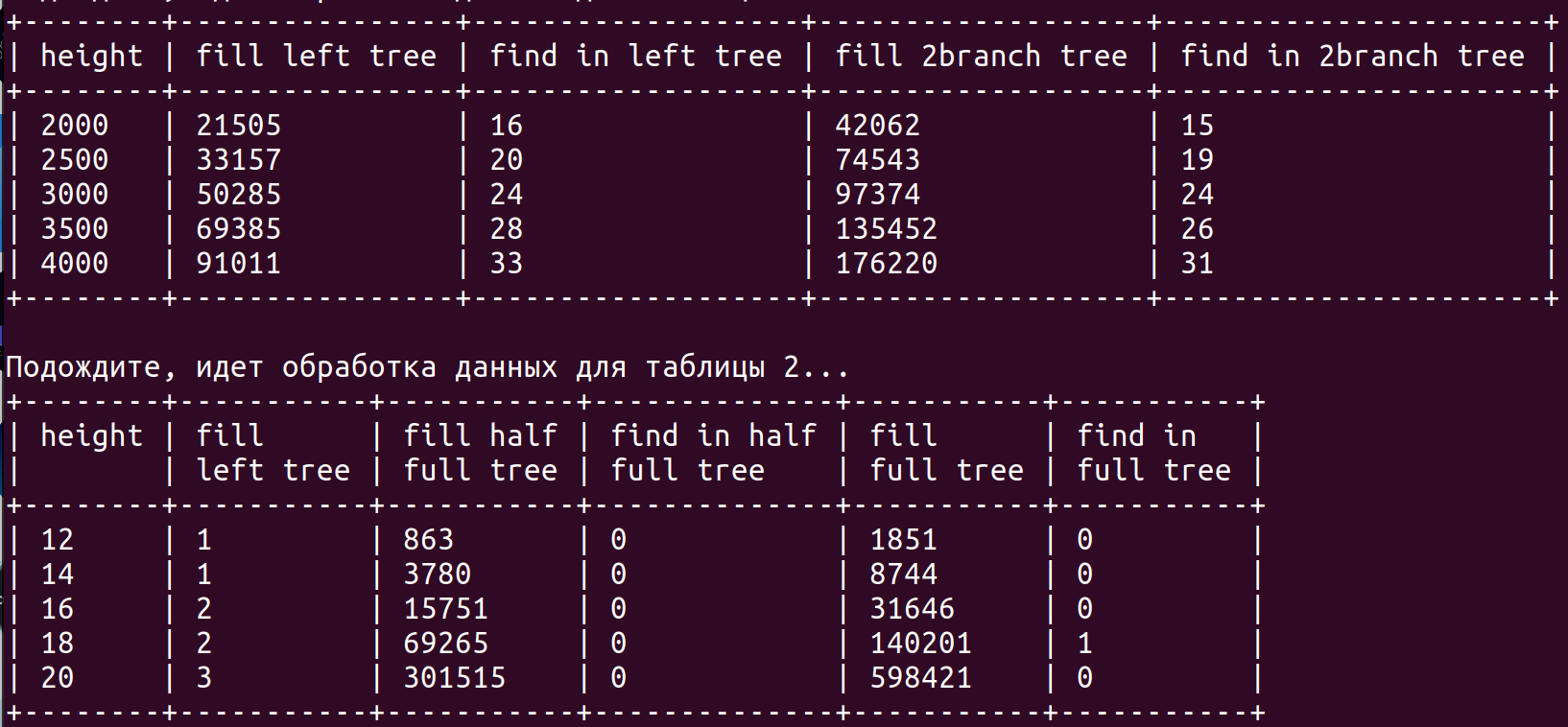
# Эффективность

Минимальное количество измерений на 1 замер – 100, после высчитывается rse, пока оно не станет меньше 5%



Время вычисления выражения через стек в среднем в 13 раз больше, чем время вычисления этого же выражения через дерево. Так как при использовании стека, представленного списком, при вычислении используется выделение памяти под новые узлы списка при заполнении стека, что очень затратно по времени.

Минимальное количество измерений на 1 замер – 5, после высчитывается rse, пока оно не станет меньше 5%



Четко видно, что функция заполнения дерева работает во много раз больше, чем функция поиска не зависимо от высоты и ветвистости дерева, это происходит в основном из-за процесса выделения памяти под новые узлы при заполнении массива, что занимает достаточно много времени.

Из первой таблицы видно, что заполнение дерева, состоящего из двух веток одинаковой высоты, занимает в 2 раза больше времени чем заполнение 1 ветки, что вполне логично, так как выполняется в 2 раза больше одинаковых действий.

Так же видно, что время работы функции поиска не зависит от ветвистости дерева, что тоже логично, так как в ней происходит исключительно спуск по определенной ветке вниз. (При вычислении времени работы функции поиска, искомый элемент всегда являлся листом дерева)

Из второй таблицы видно, что время заполнения дерева, состоящего из одной ветки во много раз меньше, чем время полного заполнения дерева. Так же четко прослеживается, что время заполнения дерева на половину в 2 раза меньше, чем время заполнения дерева полностью, то есть время работы функции прямо пропорционально ветвистости дерева.

Так же стоит отметить, что при полном заполнении бинарного дерева невозможно сделать высоту много больше 20, так как происходит переполнение по памяти, вследствие этого не получилось вычислить точное время работы функции поиска для полностью или на половину заполненных деревьев, так как эта функция работает достаточно быстро.

# Вывод

Написана программа, которая с бинарным деревом поиска и с деревом выражения.

Произведено сравнение эффективности работы функций заполнения бинарного дерева поиска и поиска элемента в нем. Так же произведено сравнение эффективности работы функции вычисления значения выражения с использованием дерева выражения и стека.

# Ответы на вопросы

1. Что такое дерево? Как выделяется память под представление деревьев?

Дерево – это нелинейная структура данных, используемая для представления

иерархических связей, имеющих отношение «один ко многим».

Память выделяет под каждый узел, рекурсивно.

2. Какие бывают типы деревьев?

Бинарное дерево поиска, дерево у которого 2 и более предков, сбалансированные деревья (АВЛ, красно-черное)

3. Какие стандартные операции возможны над деревьями?

* Обход (посещение) вершин
* Поиск по дереву
* Включение узла в дерево
* Удаление узла

4. Что такое дерево двоичного поиска?Дерево, в котором каждая вершина имеет значение, которое больше, чем содержание любой из вершин его левого поддерева и меньше, чем содержание любой из вершин его правого поддерева.