|  |  |
| --- | --- |
| **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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.03.04** Программная инженерия

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 6 |

Моделирование очереди

**Дисциплина:** Типы и структуры данных

|  |  |  |  |
| --- | --- | --- | --- |
| Студент | ИУ7-35Б |  | А. В. Толмачев |
|  | (Группа) |  | (И.О. Фамилия) |
|  |  |  |  |

Москва, 2022

**Описание условия задачи**

Цель:

Реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов. В текстовом файле содержатся целые числа. Построить двоичное дерево из чисел файла. Вывести его на экран в виде дерева. Используя процедуру, определить количество узлов дерева на каждом уровне. Добавить число в дерево и в файл. Сравнить время добавления чисел в указанные структуры.

**Описание ТЗ**

1. **Описание исходных данных**

Исходными данными является путь до текстового файла с целыми числами, записанными каждый с новой строки.

Ограничения исходных данных:

1. Путь до файла должен быть указан как аргумент командной строки
2. Допускается указание пустого файла
3. Указание несуществующего файла является некорректным вводом
4. Если файл содержит любые символы, кроме цифр и переводов строк, то он считается некорректным.
5. Дубликаты элементов не поддерживаются
6. Недопустима работа с нецелыми числами
7. Выполнение любых операций над пустым деревом является невозможным.
8. Пункт меню задается целым числом. Возможен выбор одновременно только 1 пункта меню.
9. Ввод отсутствующего пункта меню является некорректным вводом.
10. **Описание задачи, реализуемой программой**

Программа позволяет выполнять операции над деревьями:

* Добавление узла дерева
* Удаление узла дерева
* Поиска узла дерева
* Префиксный обход
* Постфиксный обход
* Инфиксный обход
* Просмотр дерева
* Определение количества узлов на каждом уровне дерева
* Определение эффективности

1. **Выходные данные**

* Сравнение эффективности
* Построенное дерево
* Последовательность вершин при префиксном/постфиксном/инфиксном обходе
* Количество узлов на каждом уровне
* Результат поиска узла в дереве
* Результат добавления/удаления узла из дерева

1. **Способ обращения к программе**

Для запуска программы необходимо из консоли вызвать исполняемый файл ./app.exe.

1. **Описание возможных аварийных ситуаций и ошибок пользователя**

Аварийные ситуации:

1. Ошибка выделения памяти

Ошибки пользователя:

1. Ввод данных, не удовлетворяющих ограничениям исходных данных, описанных ранее
2. Попытка удалить узел из пустого дерева
3. Попытка найти узел в пустом дереве
4. Попытка посмотреть пустое дерево
5. Выбор несуществующего пункта меню
6. Ввод не целого числа при добавлении узла в дерево

**Описание внутренних структур данных**

Дерево:

**struct** **tree\_node\_t**

{

**int** num;

**struct** **tree\_node\_t** \*left;

**struct** **tree\_node\_t** \*right;

};

**Набор тестов**

1. Позитивные тесты
   1. Просмотр непустого дерева
   2. Добавление узла в дерево
   3. Удаление узла из дерева
   4. Корректный выбор пункта меню
2. Негативные тесты

Тесты с входными данными, не удовлетворяющими ограничениям исходных данных, записанным ранее.

**Алгоритмы:**

**1) Поиск элемента в дереве**

Дерево рекурсивно обрабатывается, начиная с головы. Значение узла сравнивается с искомым. Если искомое значение меньше, то поиск рекурсивно продолжается в левом поддереве. Если искомое значение больше, то поиск рекурсивно продолжается в правом поддереве. Поиск заканчивается, когда искомый элемент совпадает со значением узла, либо нужное для поиска поддерево равно 0 — в этом случае искомый элемент отсутствует в дереве.

**2) Вставка элемента в дерево**

1. Значение вставляемого элемента сравнивается со значением корня

2. Если значение меньше, то функция вставки вызывается для левого поддерева

3. Если значение больше, то функция вставки вызывается для правого поддерева

4. Если правое или левое поддерево пусто — заменить его на дерево с одним узлом. Закончить вставку.

**3) Удаление узла n с ключом k**

1. Ключ k сравнивается с ключом корневого узла

2. Если k>x — рекурсивно удалить k из правого поддерева, иначе из левого поддерева.

3. Если k=x:

1) Если обоих детей нет: удаляем узел, и обнуляем ссылку у родительского узла.

2) Если нет одного из детей — настроить ссылку родительского узла на ребенка, после чего удалить узел.

3) Если есть оба ребенка — найти самый левый узел правого поддерева узла. Скопируем данные в узел. Рекурсивно удалим найденный узел.

**4) Обход дерева(инфиксный)**

1. Рекурсивно обходим левое поддерево

2. Применяем функцию к корневому узлу

3. Рекурсивно обходим правое поддерево

**Временная эффективность**

Обработка элементов в правостороннем дереве:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Место | Время вставки | Время удаления | Время поиска |
| 1000 | Середина | 6.091000 | 5.019000 | 4.005000 |
| Конец | 14.001000 | 12.103000 | 10.009000 |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Количество элементов | Место | Время вставки | Время удаления | Время поиска |
| 10000 | Середина | 92.728000 | 92.156000 | 64.036000 |
| Конец | 359.075000 | 427.094000 | 273.262000 |

Сортировка

|  |  |  |
| --- | --- | --- |
|  | Количество элементов | Время сортировки, мк |
| Разветвленное дерево | 100 | 89.383000 |
| Правостороннее дерево | 100 | 111.218000 |
| Разветвленное дерево | 1000 | 479.850000 |
| Правостороннее дерево | 1000 | 704.399000 |

Вставка в дерево и в файл

|  |  |  |  |
| --- | --- | --- | --- |
|  | Количество элементов | Время вставки в файл | Время вставки в дерево |
| Разветвленное дерево | 100 | 45.339000 | 2.000000 |
| Правостороннее дерево | 100 | 44.287000 | 2.100000 |

Время поиска в дереве

|  |  |  |
| --- | --- | --- |
|  | Количество элементов | Время поиска |
| Правостороннее дерево | 1000 | 11.26200 |
| Разветвленное дерево | 1000 | 2.132000 |

**Вывод:**

В рамках данной лабораторной работы были получен навыки по работе с двоичными деревьями поиска. Двоичные деревья поиска являются наиболее эффективными в случае, если дерево сбалансированно. Сортировка такого дерева будет занимать O(nlogn), а поиск и вставка — O(logn); Худший случай — когда дерево является правосторонним или левосторонним деревом, т. е. вырождается в связанный список. В этом случае время обработки узлов дерева будет занимать O(n).

**Контрольные вопросы:**

1. Что такое дерево?

Дерево – способ организации данных в виде иерархической структуры

1. Как выделяется память под представление деревьев?

Память выделяется под каждый узел дерева в отдельности

1. Какие бывают типы деревьев?

Бинарные деревья, красно-черные деревья, B-деревья, АВЛ-деревья

1. Какие стандартные операции над деревьями?

Обход дерева, вставка в дерево, удаление узла из дерева, поиск в дереве

1. Что такое дерево двоичного поиска?

Двоичное дерево поиска – двоичное дерево, организованное таким образом, что для каждого узла t все ключе в его левом поддереве меньше, чем ключ узла, а ключи в правом поддереве больше, чем ключ t.