

**Министерство образования и науки 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**

### Студент Завойских Евгения Васильевна

*фамилия, имя, отчество*

### Группа ИУ7-33Б

Студент Завойских Е.В.

*подпись, дата фамилия и.о.*

### Руководитель практики Барышникова М.Ю.

*подпись, дата фамилия, и.о.*

### Оценка

*2021 г.*

**Условие задачи**

В текстовом файле содержатся целые числа. Построить ДДП из чисел файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из чисел файла. Использовать закрытое хеширование для устранения коллизий. Осуществить добавление введенного целого числа, если его там нет, в ДДП, в сбалансированное дерево, в хеш-таблицу и в файл. Сравнить время добавления, объем памяти и количество сравнений при использовании различных (4-х) структур данных. Если количество сравнений в хеш-таблице больше указанного (вводить), то произвести реструктуризацию таблицы, выбрав другую функцию.

**Внешняя спецификация программы**

1. Входные данные

При вызове программы выводится меню, каждый раз пользователь выбирает один из пунктов.

При добавлении в структуры элемента вводится целое число с клавиатуры. При определении максимального числа сравнений вводится целое число с клавиатуры. Также с клавиатуры вводится название файла для считывания чисел.

1. Выходные данные

По запросу выводятся ДДП, АВЛ дерево (создаются .png файлы) и хеш-таблица.

По запросу выводятся замеры времени добавления числа (а так же памяти и количества сравнений) в деревья, хеш-таблицу и файл.

В случае некорректного ввода команды меню выводится сообщение об ошибке и предоставляется возможность выбрать команду еще раз. В случае некорректного ввода числа или некорректных данных в файле также выводится соответствующее сообщение об ошибке. При ошибке открытия исходного файла выводится соответствующее сообщение. При ошибке выделения памяти выводится соответствующее сообщение об ошибке. При попытке вывести пустое дерево или хеш-таблицу выводится соответствующее сообщение. При попытке добавить уже имеющееся число – соответствующее сообщение.

1. Задача, реализуемая программой

Данная программа предназначена для работы с ДДП, АВЛ деревом и хеш-таблицей, и представляет собой приложение со следующими возможными операциями:

1. Ввести имя исходного файла.
2. Вывести ДДП.
3. Вывести АВЛ дерево.
4. Вывести хеш-таблицу.
5. Добавить целое число в деревья, таблицу и файл.
6. Задать максимальное число сравнений для хеш-таблицы.
7. Сравнить время добавления, объем памяти и количество сравнений при использовании 4-х структур данных.

После каждого добавления если количество сравнений в хеш-таблице больше указанного числа, то производится реструктуризация таблицы.

1. Способ обращения к программе

Программа может быть вызвана из консоли или запущена в любой среде разработки, поддерживающей язык С.

1. Возможные аварийные ситуации и ошибки пользователя

В случае аварийной ситуации программа выводит сообщение о произошедшей ошибке. Ошибка может возникнуть в результате некорректного ввода (не соответствующего формату), при открытии файла, при попытке вывести пустое дерево или хеш-таблицу, выделении динамической памяти, добавить уже имеющееся число.

**Внутренние структуры данных**

В программе используются структуры для хранения узла дерева, элемента хеш-таблицы и самой хеш-таблицы.

Для дерева:

// структура для определения узла двоичного дерева поиска

struct tree\_node

{

    int val; // значение узла

    struct tree\_node \*left; // указатель на левое поддерево

    struct tree\_node \*right; // указатель на правое поддерево

    int bal; // показатель сбалансированности вершины

};

typedef struct tree\_node tree\_node\_s;

Для хеш-таблицы:

// структура для определения элемента хеш-таблицы

struct ht\_elem

{

    int val; // значение элемента

    int busy; // занят ли элемент таблицы

};

typedef struct ht\_elem ht\_elem\_s;

// структура для определения хеш-таблицы

struct hash\_table

{

    int size; // максимальный размер таблицы

    int cur\_size; // текущий размер таблицы

    ht\_elem\_s \*arr; // массив с элементами таблицы

};

typedef struct hash\_table hash\_table\_s;

**Алгоритм**

Пока пользователь не решит завершить программу, выполняется действие, выбранное из меню:

1. Ввести имя исходного файла: вводится имя файла с целыми числами, далее числа по одному добавляются в деревья (каждый раз динамически выделяется память под новый элемент дерева) и хеш-таблицу (используется закрытое хеширование для устранения коллизий; хеш-функция – остаток от деления числа на размер таблицы), размер хеш-таблицы изначально в 2 раза больше количества чисел в файле
2. Вывести ДДП: в файл записывается ДДП на языке DOT, по нему формируется .png файл с изображением дерева
3. Вывести АВЛ дерево: в файл записывается АВЛ на языке DOT, по нему формируется .png файл с изображением дерева
4. Вывести хеш-таблицу: построчно выводятся индекс элемента и его значение в виде таблицы
5. Добавить целое число в деревья, таблицу и файл: вводится число, если его нет в деревьях и хеш-таблице, оно добавляется в деревья (динамически выделяется память под новый элемент), хеш-таблицу (если она уже целиком заполнена, то ее размер увеличивается в 2 раза) и файл (записывается в конец)
6. Задать максимальное число сравнений для хеш-таблицы: вводится целое число больше 0, оно сохраняется как максимальное число сравнений
7. Сравнить время добавления, объем памяти и количество сравнений при использовании 4-х структур данных: выводится таблица со средним временем добавления (количеством сравнений и памятью) элемента в деревья, хеш-таблицу и файл

После выполнения одного из пунктов меню идет проверка: если количество сравнений при добавлении элемента в хеш-таблицу оказалось больше указанного числа, то производится реструктуризация таблицы: создается новая хеш-таблица с прежними числами, но с использованием другой хеш-функции.

**Тесты**

Негативы:

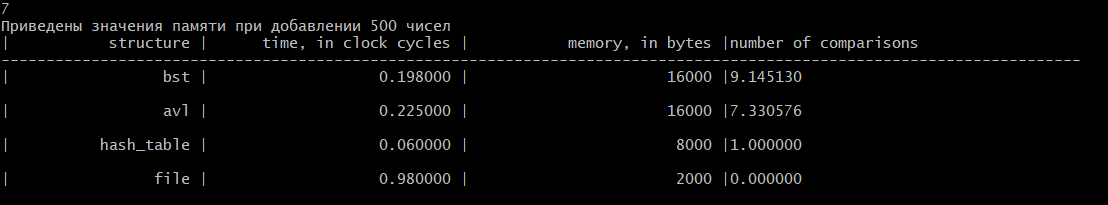
|  |  |  |
| --- | --- | --- |
| Входные данные | Вывод | Что проверяется |
| Возможные действия (введите номер нужной команды):  0. Завершить работу  1. Ввести имя исходного файла  2. Вывести ДДП  3. Вывести АВЛ дерево  4. Вывести хеш-таблицу  5. Добавить целое число в деревья, таблицу и файл  6. Задать максимальное число сравнений для хеш-таблицы  7. Сравнить время добавления, объем памяти и количество сравнений при использовании 4-х структур данных  12 | Нет команды с введенным номером | Неверно введена команда меню |
| Возможные действия (введите номер нужной команды):  0. Завершить работу  1. Ввести имя исходного файла  2. Вывести ДДП  3. Вывести АВЛ дерево  4. Вывести хеш-таблицу  5. Добавить целое число в деревья, таблицу и файл  6. Задать максимальное число сравнений для хеш-таблицы  7. Сравнить время добавления, объем памяти и количество сравнений при использовании 4-х структур данных  2 | Дерево не создано | Попытка вывести пустое ддп |
| 3 | Дерево не создано | Попытка вывести пустое авл дерево |
| 4 | Таблица не создана | Попытка вывести пустую хеш-таблицу |
| 1  e.txt | Ошибка открытия файла | Входной файл не существует |
| 1  2.txt  В файле:  ф | Ошибка чтения из файла | В файле не числа |
| 1  1.txt  Выполнено  5  Введите целое число: 0  В файле:  1  3  -5  6  7  2  0  10  17  19 | Такое число уже есть | Попытка добавить уже имеющееся число |
| 5  Введите целое число: ф | Ошибка ввода | Вместо целого числа введена буква |
| 6  Введите положительное целое число: -11 | Ошибка ввода | Максимальное число сравнений - отрицательно |

Позитивы:

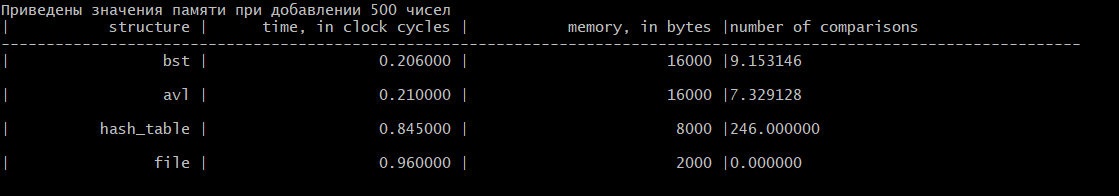
|  |  |  |
| --- | --- | --- |
| Входные данные | Вывод | Что проверяется |
| 1  3.txt  В файле:  0  1  2  3  4 | Выполнено | Создание структур по числам файла |
| 4 | | index | value |  | 0 | 0 |  | 1 | 1 |  | 2 | 2 |  | 3 | 3 |  | 4 | 4 | | Вывод хеш-таблицы |
| 3 | avl.png | Вывод авл дерева |
| 2 | bst.png | Вывод ддп |
| 6  Введите положительное целое число: 3 |  | Задание максимального числа сравнений |
| 5  Введите целое число: 10 | Количество сравнений в хеш-таблице превышено, попытка реструктуризировать хеш-таблицу  Удалось реструктуризировать хеш-таблицу  | index | value |  | 0 | 0 |  | 1 | 10 |  | 2 | 2 |  | 4 | 4 |  | 6 | 1 |  | 8 | 3 | | Добавление числа |
| 7 | Снимок.PNG | Вывод таблицы по сравнению добавления в 4 разные структуры |

**Вывод**

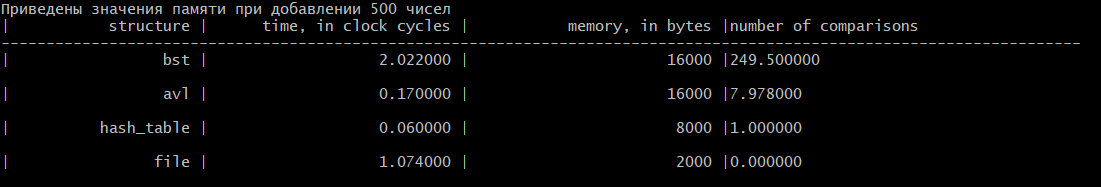
Замеры времени (в тактах), количества сравнений и замеры памяти (в байтах) для 4-х структур при добавлении числа:



Добавление в АВЛ дерево работает медленнее всех из-за процесса балансировки. Хеш-таблица добавляет элемент быстрее всех при отсутствии коллизий, но при большом количестве коллизий (если хеш-функция подобрана неудачно), число сравнений сильно возрастает (вместе со временем добавления), и использование хеш-таблицы становится невыгодным (ниже пример с “плохой” хеш-функцией).



Нам выгодно использовать по времени хеш-таблицу, если хеш-функция подобрана так, чтобы минимизировать число коллизий, иначе выгоднее использовать АВЛ или ДДП. АВЛ дерево выигрывает по времени у ДДП при добавлении, если ДДП становится сильно несбалансированным (в примере ниже – вырождается в односвязный список), так как при добавлении нам приходится проходить по все элементам ДДП.



**Ответы на вопросы**

1. **Что такое дерево?**

Дерево – это нелинейная структура данных, используемая для представления иерархических связей, имеющих отношение «один ко многим».

1. **Как выделяется память под представление деревьев?**

В виде односвязного списка – динамически под каждый узел.

1. **Какие стандартные операции возможны над деревьями?**

Обход дерева, поиск элемента, добавление и удаление узла.

1. **Что такое дерево двоичного поиска?**

Если у каждой вершины дерева имеется не более двух потомков (левые и правые поддеревья), то такое дерево называется двоичным или бинарным. Дерево двоичного поиска – это такое дерево, в котором все левые потомки моложе предка, а все правые – старше.

1. **Чем отличается идеально сбалансированное дерево от АВЛ дерева?**

Если при добавлении узлов в дерево мы будем их равномерно располагать слева и справа, то получится дерево, у которого число вершин в левом и правом поддеревьях отличается не более, чем на единицу. Такое дерево называется идеально сбалансированным.   
 Адельсон-Вельский и Ландис сформулировали менее жесткий критерий сбалансированности таким образом: двоичное дерево называется сбалансированным, если у каждого узла дерева высота двух поддеревьев отличается не более чем на единицу. Такое дерево называется АВЛ-деревом.

1. **Чем отличается поиск в АВЛ-дереве от поиска в дереве двоичного поиска?**

Поиск осуществляется аналогично, но быстрее благодаря сбалансированности АВЛ дерева.

1. **Что такое хеш-таблица, каков принцип ее построения?**

То есть необходимо создать такую функцию, по которой можно вычислить этот индекс. Такая функция называется хеш-функцией (от англ. To hash - крошить, рубить) и она ставит в соответствие каждому ключу ki индекс ячейки j, где расположен элемент с этим ключом, таким образом: h(ki) = j, если j=(1,m), где j принадлежит множеству от 1 до m, а m. – размерность массива. Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей.

1. **Что такое коллизии? Каковы методы их** **устранения?**

Может возникнуть ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда h(K1)=h(K2), в то время как K1 ≠ K2. Такая ситуация называется коллизией.

 Основные методы устранения коллизий:

* при открытом хешировании к ячейке по данному ключу прибавляется связанный список,
* при закрытом – новый элемент кладется в ближайшую свободную ячейку после данной.

1. **В каком случае поиск в хеш-таблицах становится неэффективен?**

Поиск в хеш-таблице становится неэффективен при большом числе коллизий. В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции или расширение таблицы.

1. Эффективность поиска в АВЛ деревьях, в дереве двоичного поиска и в хеш-таблицах**?**

* В хеш-таблице минимальное время поиска О(1).
* В АВЛ дереве: O(log2 n).
* В ДДП О(h), где h - высота дерева (от log2n до n).