|  |  |
| --- | --- |
| 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)** |

**Типы и структуры данных.**

**Лабораторная работа №6.**

**«Деревья. Хеш-таблицы»**

Студент **Леонов Владислав Вячеславович**

Группа **ИУ7-36Б**

Студент **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Леонов В.В.

*подпись, дата фамилия, и.о.*

Преподаватель **\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_** Силантьева А.В.

*подпись, дата фамилия, и.о.*

*2020 г.*

Оглавление

[Условие и техническое задание 3](#_Toc59541794)

[Цель работы 3](#_Toc59541795)

[Задание 3](#_Toc59541796)

[Входные данные 4](#_Toc59541797)

[Выходные данные 4](#_Toc59541798)

[Обращение к программе 4](#_Toc59541799)

[Функции программы 5](#_Toc59541800)

[Аварийные ситуации 5](#_Toc59541801)

[Реализация 6](#_Toc59541802)

[Структуры данных 6](#_Toc59541803)

[Структура](#_Toc59541804) ***[vector\_t](#_Toc59541804)*** [6](#_Toc59541804)

[Структура](#_Toc59541805) ***[vertex\_t](#_Toc59541805)*** [7](#_Toc59541805)

[Структура](#_Toc59541806) ***[node\_t](#_Toc59541806)*** [7](#_Toc59541806)

[Струтура](#_Toc59541807) ***[hash\_table\_t](#_Toc59541807)*** [8](#_Toc59541807)

[Алгоритм 9](#_Toc59541808)

[Тестирование 10](#_Toc59541809)

[Аварийные ситуации 10](#_Toc59541810)

[Штатное поведение 11](#_Toc59541811)

[Оценка эффективности использования ДДП, АВЛ-деревьев и хеш-таблиц 14](#_Toc59541812)

[Оценка эффективности по времени (время в тактах процессора). 14](#_Toc59541813)

[Оценка эффективности по памяти (в байтах). 15](#_Toc59541814)

[Контрольные вопросы 16](#_Toc59541815)

[Вывод о проделанной работе 18](#_Toc59541816)

# Условие и техническое задание

## Цель работы

Построить дерево, вывести его на экран в виде дерева, реализовать основные операции работы с деревом: обход дерева, включение, исключение и поиск узлов, сбалансировать дерево, сравнить эффективность алгоритмов сортировки и поиска в зависимости от высоты деревьев и степени их ветвления; построить хеш-таблицу и вывести ее на экран, устранить коллизии, если они достигли указанного предела, выбрав другую хеш-функцию и реструктуризировав таблицу; сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска (ДДП), в хеш-таблицах и в файлах. Сравнить эффективность реструктуризации таблицы для устранения коллизий и поиска в ней с эффективностью поиска в исходной таблице.

## Задание

Построить ДДП, сбалансированное двоичное дерево (АВЛ) и хеш-таблицу по указанным данным. Сравнить эффективность поиска в ДДП в АВЛ дереве и в хеш-таблице (используя открытую или закрытую адресацию) и в файле. Вывести на экран деревья и хеш-таблицу. Подсчитать среднее количество сравнений для поиска данных в указанных структурах. Произвести реструктуризацию хеш-таблицы, если среднее количество сравнений больше указанного. Оценить эффективность использования этих структур (по времени и памяти) для поставленной задачи. Оценить эффективность поиска в хеш-таблице при различном количестве коллизий.

Построить ДДП, в вершинах которого находятся слова из текстового файла. Вывести его на экран в виде дерева. Сбалансировать полученное дерево и вывести его на экран. Построить хеш-таблицу из слов текстового файла. Использовать метод цепочек для устранения коллизий. Осуществить поиск введенного слова в ДДП, в сбалансированном дереве, в хеш-таблице и в файле. Сравнить время поиска, 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.

## Входные данные

1. **Имя файла** для чтения данных.
2. **Слова** при добавлении, удалении или поиске в деревьях и хэш-таблицах.
3. **Максимальное количество коллизий.**
4. **Опция меню.**

## Выходные данные

1. Время создания двоичного дерева поиска.
2. Время создания сбалансированного дерева поиска.
3. Время создания хеш-таблицы.
4. Время вставки в двоичное дерево поиска.
5. Время вставки в сбалансированное дерево поиска.
6. Время вставки в хэш-таблицу.
7. Время удаления из двоичного дерева поиска.
8. Время удаления из сбалансированного дерева поиска.
9. Время удаления из хэш-таблицы.
10. Время поиска и кол-во сравнений в двоичном дереве поиска.
11. Время поиска и кол-во сравнений в сбалансированном дереве поиска.
12. Время поиска и кол-во сравнений в хэш-таблице.
13. Печать состояния деревьев в виде картинки (формат .png)
14. Печать состояния хэш-таблицы в консоль.
15. Сравнительная таблица по времени операций для различных структур.
16. Объем памяти ДДП, АВЛ дерева, хеш-таблицы.

## Обращение к программе

Обращение к программе **app.exe** осуществляется через запуск из терминала.

## Функции программы

1. Чтение данных из файла.
2. Поиск в файле.
3. Создание ДДП, АВЛ дерева и хеш-таблицы.
4. Добавление элемента ДДП, АВЛ дерево и хеш-таблицу.
5. Удаление элемента ДДП, АВЛ дерева и хеш-таблицы.
6. Поиск элемента в ДДП, АВЛ дереве и хэш-таблице.
7. Печать состояния ДДП, АВЛ дерева и хеш-таблицы.
8. Устранение коллизий в хеш-таблице.
9. Сравнительный анализ структур.
10. Выход из программы.

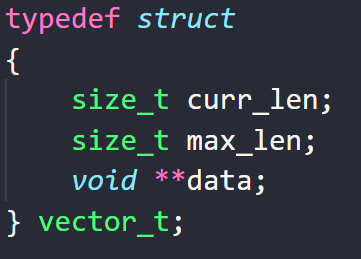
## Аварийные ситуации

1. Некорректная опция меню (буква или значение, несоответствующее ни одному пункту из меню.
2. Некорректное название файла.
3. Отсутствие доступа к файлу.
4. Некорректный уровень коллизий.
5. Ошибка выделения памяти операционной системой.

# Реализация

## Структуры данных

### Структура ***vector\_t***



В данной программе используется не прямой вид хранения (структуры хранят лишь указатели на данные, но самими данными не владеют). Для хранения данных используется агрегированный тип ***vector\_t.***

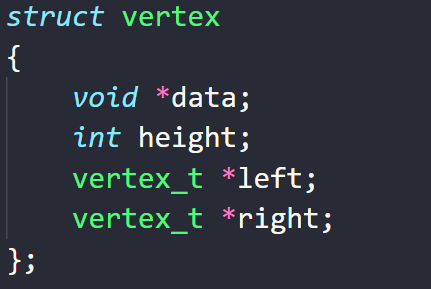
Поле ***curr\_len*** представляет собой беззнаковое целое и служит для хранения текущей длины вектора.

Поле ***max\_len*** представляет собой беззнаковое целое и служит для хранения максимальной длины вектора.

Поле ***data*** представляет собой динамический массив и служит для хранения данных в программе.

При данной реализации вектора размер будет увеличиваться в 2 раза в случае его заполнения.

### Структура ***vertex\_t***



Для реализации ДДП и АВЛ дерева используется агрегированный тип ***vertex\_t***.

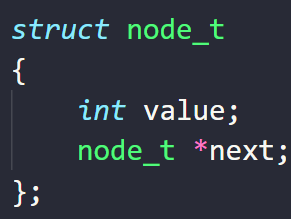
Поле ***data*** представляет собой указатель на данные, соответствующие данной вершине дерева.

Поле ***height*** представляет целое число и служит для хранения высоты поддерева с корнем в данном узле.

Поле ***left*** представляет собой указатель на левое поддерево.

Поле ***right*** представляет собой указатель на правое поддерево.

### Структура ***node\_t***

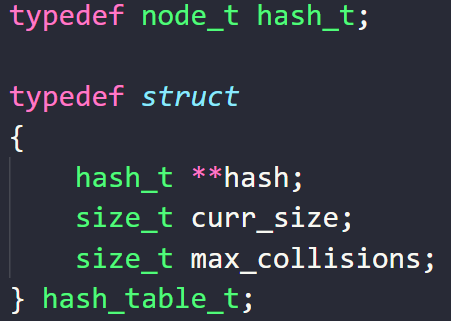


Для хранения узла списка используется агрегированный тип ***node\_t.***

Поле ***value*** представляет собой целое число и служит для хранения данных в узле списка.

Поле ***next*** представляет собой указатель и служит для хранения адреса следующего узла списка.

### Струтура ***hash\_table\_t***



Для реализации хеш-таблицы используется агрегированный тип ***hash\_table\_t***.

Поле ***hash*** представляет собой динамический массив указателей на списки и служит для хранения элементов хеш-таблицы.

Поле ***curr\_size*** представляет собой беззнаковое целое и служит для хранения текущего размера хеш-таблицы.

Поле ***max\_collisions*** представляет собой беззнаковое целое и служит для хранения максимального количества коллизий в хеш-таблице.

## Алгоритм

1. Пользователь вводит номер команды из меню.
2. Пока пользователь не введет 0 (выход из программы), ему будет предложено выполнять действия из меню программы.

При реализации поиска в файле выполняется построчный просмотр слов со сравнением ключа поиска с прочитанным (функция **search\_file**).

При реализации поиска в деревьях выполняется классический алгоритм просмотра деревьев (функция **tree\_search**).

При реализации поиска в хеш-таблицы происходит вычисление значение хеш-функции для заданного ключа с просмотром списка, соответствующего индексу в хэш-таблице (функция **hash\_find**).

В качестве хеш-функции происходит вычисление многочлена вида

,

где **str[n]** код буквы в таблице символов, n – порядковый номер буквы слове, k – количество букв в слове.

В ходе оценки результатов был получен вывод о том, что данная функция обеспечивает не равномерное распределение, поэтому была добавлена другая хеш-функция.

**,**

где n – номер буквы в слове.

# Тестирование

## Аварийные ситуации

|  |  |  |
| --- | --- | --- |
| № | Описание теста | Поведение программы |
| 1 | Некорректная опция из меню |  |
| 2 | Некорректный файл |  |
| 3 | Пустой файл |  |
| 4 | Некорректное максимальное количество коллизий |  |

## Штатное поведение

|  |  |  |
| --- | --- | --- |
| № | Описание теста | Демонстрация работы программы |
| 1 | Чтение данных из файла. |  |
| 2 | Создание дерева поиска |  |
| 3 | Создание сбалансированного дерева поиска |  |
| 4 | Создание хеш-таблицы |  |
| 5 | Поиск в дереве поиска  (без результатов) |  |
| 6 | Поиск в сбалансированном дереве  (с результатами) |  |
| 7 | Поиск в хеш-таблице |  |
| 8 | Печать не сбалансированного  дерева |  |
| 9 | Печать сбалансированного дерева |  |
| 10 | Печать хеш-таблицы |  |

# Оценка эффективности использования ДДП, АВЛ-деревьев и хеш-таблиц

## Оценка эффективности по времени (время в тактах процессора).

50 элементов

|  |  |  |  |
| --- | --- | --- | --- |
|  | ДДП | АВЛ дерево | Хеш-таблица |
| Создание | 9684 | 27523 | 30345 |
| Добавление | 193 | 550 | 606 |
| Поиск | 405 | 317 | 275 |
| Удаление | 421 | 677 | 506 |

100 элементов

|  |  |  |  |
| --- | --- | --- | --- |
|  | ДДП | АВЛ дерево | Хеш-таблица |
| Создание | 41422 | 63545 | 53656 |
| Добавление | 414 | 635 | 536 |
| Поиск | 371 | 315 | 212 |
| Удаление | 394 | 674 | 371 |

500 элементов

|  |  |  |  |
| --- | --- | --- | --- |
|  | ДДП | АВЛ дерево | Хеш-таблица |
| Создание | 247906 | 499298 | 282331 |
| Добавление | 495 | 998 | 564 |
| Поиск | 710 | 548 | 269 |
| Удаление | 748 | 1258 | 481 |

1000 элементов

|  |  |  |  |
| --- | --- | --- | --- |
|  | ДДП | АВЛ дерево | Хеш-таблица |
| Создание | 605411 | 1120418 | 563497 |
| Добавление | 605 | 1120 | 563 |
| Поиск | 701 | 518 | 218 |
| Удаление | 726 | 1143 | 367 |

100000 элементов

|  |  |  |  |
| --- | --- | --- | --- |
|  | ДДП | АВЛ дерево | Хеш-таблица |
| Создание | 278456893 | 361009349 | 72497560 |
| Добавление | 2784 | 3610 | 724 |
| Поиск | 2381 | 1868 | 376 |
| Удаление | 2039 | 2747 | 598 |

Исходя из приведенных выше данных, что хеш-таблица является более эффективной по времени структурой данной, чем ДДП и АВЛ деревья, так как при грамотном выборе хеш-функции получаем равномерное распределение и отсутствует необходимость в большом количестве сравнений вовремя операция добавления, удаления и поиска элементов. Заметим, что при увеличении размера обрабатываемых данных, время на выполнение операция остается в одном и том же временном диапазоне, что подтверждает ожидаемый результат, что время выполнения операция не зависит от размера обрабатываемых данных. Однако следует отметить, что при малых данных хеш-таблица может проигрывать деревьям, ввиду необходимости выполнения операции вычисления хеш-функции.

В свою очередь АВЛ деревья более эффективны по времени поиска, чем не сбалансированные (порядка 25-30% ), однако в тоже время операции создания (порядка 1.3-2 раза), удаления, добавления (порядка 1.3-2 раза) элементов получаются менее эффективны ввиду необходимости проведения балансировки.

## Оценка эффективности по памяти (в байтах).

|  |  |  |  |
| --- | --- | --- | --- |
| Кол-во элементов | ДДП | АВЛ дерево | Хеш-таблица |
| 10 элементов | 320 | 320 | 192 |
| 20 элементов | 640 | 640 | 432 |
| 30 элементов | 960 | 960 | 656 |
| 50 элементов | 1600 | 1600 | 1072 |

Таким образом, можно сделать вывод, что АВЛ дерево не требует дополнительной памяти относительно ДДП. В свою очередь хеш-таблица более эффективна по памяти (порядка 1.5 раз).

Следует отметить, что при устранении коллизий, объем памяти хеш-таблицы возрастает, что еще раз подтверждает важность выбора правильного и эффективного алгоритма хеширования.

# Контрольные вопросы

**1. Что такое дерево?**

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

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

Память под дерево выделяется подобно списку, под каждую вершину отдельно.

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

Обход дерева, поиск по дереву, включение в дерево, исключение из дерева.

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

Двоичное дерево поиска - двоичное дерево, для каждого узла которого сохраняется условие: левый потомок больше или равен родителю, правый потомок строго меньше родителя (либо наоборот).

**5. Чем отличается идеально сбалансированное дерево от АВЛ дерева**

У АВЛ дерева для каждой его вершины высота двух её поддеревьев различается не более чем на 1, а у идеально сбалансированного дерева различается количество вершин в каждом поддереве не более чем на 1.

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

Поиск в АВЛ дереве быстрее, чем в ДДП

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

Хеш-таблицей называется массив, заполненный элементами в порядке, определяемом хеш-функцией. Хеш-функция каждому элементу таблицы ставит в соответствие некоторый индекс. Функция должна быть простой для вычисления, распределять ключи в таблице равномерно и давать минимум коллизий.

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

Коллизия – ситуация, когда разным ключам хеш-функция ставит в соответствие один и тот же индекс. Основные методы устранения коллизий: открытое и закрытое хеширование. При открытом хешировании к ячейке по данному ключу прибавляется связанны список, при закрытом – новый элемент кладется в ближайшую свободную ячейку после данной.

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

Поиск в хеш-таблице становится неэффективен при большом числе коллизий –сложность поиска возрастает по сравнению с О(1). В этом случае требуется реструктуризация таблицы – заполнение её с использованием новой хеш-функции.

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

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

# Вывод о проделанной работе

В результате лабораторной работы были изучены структуры данных – ДДП, АВЛ-дерево, хеш-таблицы, методы их реализации на основе известных ранее структур (список и массив), а также особенности обработки подобных структур данных. Были закреплены навыки работы с динамической памятью, а также освоены методы и способы представления деревьев в графическом виде на языке DOT.

Согласно полученным экспериментальным путем данных использование АВЛ деревьев над ДДП имеет как плюсы так и минусы: а именно в случае необходимости хранения большого количества данных без частого их изменения лучше использовать АВЛ-деревья (выигрыш по времени порядка 25%-30% раза), в случае же если данные часто динамически меняются нет необходимости в использовании АВЛ деревьев, лучше использовать не сбалансированные в среднем.

Использование хеш-таблиц более эффективно, чем деревья как по времени, так и по памяти (1.5 раз), т.к. вычислительная сложность выполнения операций не зависит от размера таблицы. Однако в свою очередь необходимо грамотно и осторожно подойти к выбору хеш-функции, она должно быть простой алгоритмически и при этом обеспечивать минимальное количество коллизий.