|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **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 (ИУ)

КАФЕДРА Программное обеспечение ЭВМ и информационные технологии (ИУ7)

**ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7  
«Сбалансированные деревья, хеш–таблицы»**

Студент Фам Минь Хиеу

Группа ИУ7 – 32Б

Преподаватель Никульшина Т.А.

*2022 г.*

**Цель работы**

построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах. Сравнить эффективность устранения коллизий при внешнем хешировании.

**Описание условия задачи**

Сбалансировать полученное дерево в ЛР6. Вывести его на экран в виде дерева. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана. Осуществить поиск введенного слова в двоичном дереве поиска, в сбалансированном дереве и в хеш-таблице. Сравнить время поиска, объем памяти и количество сравнений при использовании различных структур данных.

**Входные данные:**

1. **Имя файла с деревом:** строка, содержащая имя файла.
2. **Максимальное значение допустимых коллизий:** целое число.
3. **Слово для поиска:** строка, содержащая слово, которое планируется найти в дереве.

**Выходные данные:**

1. ДДП в виде дерево.
2. АВЛ в виде дерево.
3. Хеш-таблица вершин дерева.
4. Эффективность при использовании различных структуров.

**Функция программы:** программа выполняет ряд функций в порядке:

1. Вывод бинарного дерева.
2. Вывод сбалансированного бинарного дерева.
3. Вывод хеш-таблицы.
4. Вывод эффективность при использовании различных структуров.

**Обращение к программе:** запускается из терминала с аргументом в виде файла, типа **./app.exe file\_name.txt**

**Аварийные ситуации:**

1. Некорректный ввод имени файла.  
   На входе: имя файла, несуществующего в системе.   
   На выходе: сообщение «Неверное имя файла! Повторите попытку.»
2. Пустой файл в качестве аргумента программы.  
   На входе: пустой файл.  
   На выходе: сообщение «Файл пуст! Проверьте файл.»
3. Некорректный ввод максимального числа коллизий.  
   На входе: буква или, любой другой нечисловой символ или отрицательное число.  
   На выходе: сообщение «Введено недопустимое значение! Повторите попытку.»

**Структуры данных**

Реализация листа дерева:

**typedef struct tree\_node**

**{**

***char* \*value;**

***unsigned char* height;**

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

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

**} tree\_node;**

Поля структуры:

* ***char* \*value *–*** указатель на массив символов;
* ***unsigned char \**height –** Разность высоты между двумя потомками;
* ***struct tree\_node* \*left *–***указатель на левого потомка;
* ***struct tree\_node* \*right *–***указатель на правого потомка.

Реализация динамического массива (для реализации хеш-таблицы):

**typedef struct**

**{**

***tree\_node* \*\*arr;**

***int* size;**

***int* mem\_size;**

**} dynarr\_t;**

Поля структуры:

* ***tree\_node* \*\*arr *–*** указатель на массив указателей листов дерева;
* ***int* size *–***фактическая ёмкость массива;
* ***int* mem\_size –** аллоцированная ёмкость массива.

**Алгоритм**

Сначала происходит балансировка дерева методом поворота. После постройки дерева строится хеш-таблица с помощью примитивной функции по дереву. В случае, если количество коллизий превосходит допустимое, таблица самоперестраивается на основе улучшенной хеш-функции).

**Тесты**

|  |  |  |  |
| --- | --- | --- | --- |
|  | Тест | Пользовательский ввод | Результат |
| 1 | Некорректный ввод имени файла | ./app.exe abc.txt (файл не существует) | Неверное имя файла! Повторите попытку. |
| 2 | Пустой файл | ./app.exe sth.txt (пустой файл) | Файл пуст! Проверьте файл. |
| 3 | Некорректный ввод максимального количества коллизий | A | Введено недопустимое значение! Повторите поппытку |
| 4 | Некорректный ввод максимального количества коллизий | 0 | Введено недопустимое значение! Повторите поппытку |
| 5 | Некорректный ввод максимального количества коллизий | -1 | Введено недопустимое значение! Повторите поппытку |
| 6 | Ввод несуществующего слова | iu7 (слова нет в дереве) | Слово “iu7” не найдено. |
| 7 | Ввод количества коллизий, большего, чем текущее максимальное | 3 (при максимальном 2) | Результат достижим за введенное количество коллизий.  Пересоздание таблицы не требуется. |
| 8 | Ввод количества коллизий, меньшего, чем текущее максимальное | 1 (при максимальном 2) | Пересоздание хеш-таблицы |
| 9 | Корректный ввод всех характеристик | Корректный файл, корректный ввод числа коллизий | Количественная характеристика моделирования |

**Оценка эффективности**

Измерения эффективности реализаций решения задачи на дереве будут производиться в единицах измерения – тактах.

**Поиск слова (в тактах)\*:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Количество элементов дерева** | **Бинарное дерево** | **Сбалансированное бинарное дерево** | **Хеш-таблица**  **(открытое)** | **Хеш-таблица**  **(закрытое)** | **Файл** |
| 16 | 1140 | 976 | 636 | 615 | 7841 |
| 32 | 1563 | 923 | 720 | 742 | 11362 |
| 50 | 1767 | 1351 | 785 | 773 | 21022 |
| 500 | 1989 | 1610 | 842 | 835 | 43696 |
| 1000 | 2017 | 1724 | 966 | 970 | 86655 |

**Объём занимаемой памяти (в байтах):**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Количество элементов дерева** | **Бинарное дерево** | **Сбалансированное бинарное дерево** | **Хеш-таблица**  **(открытое)** | **Хеш-таблица**  **(закрытое)** | **Файл** |
| 16 | 384 | 384 | 488 | 510 | 119 |
| 32 | 768 | 768 | 824 | 1110 | 222 |
| 50 | 1200 | 1200 | 1596 | 1590 | 379 |
| 500 | 12000 | 12000 | 11160 | 15030 | 3737 |
| 1000 | 24000 | 24000 | 22120 | 30030 | 7624 |

**Контрольные вопросы**

**1. 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.

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

Поиск в АВЛ дереве происходит быстрее, чем в ДДП.

**3. 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.

**4. 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.

**5. 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.

**6. 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**

Использование хеш-таблицы всегда эффективно по времени, но не всегда эффективно по памяти (в случае хорошей дистрибуции функции распределение будет не плотным), так как требует выделенной памяти под каждый хеш (если отсутствуют коллизии, хорошая дистрибуция). В случае деревьев, АВЛ дерево не всегда выигрывает по времени поиска у несбалансированного дерева, так как порядок вершин при балансировке меняется, но всегда выигрывает по среднему значению количества сравнений и среднему времени поиска по дереву.