Отчет по лабораторной работе № 6

по дисциплине

“Типы и структуры данных”

Сушина А.Д. ИУ7-31б

Работа № 6

Обработка очередей

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) и в хеш-таблицах. Сравнить эффективность реструктуризации таблицы для устранения коллизий с эффективностью поиска в исходной таблице.

Описание условия задачи

Вариант 3.
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. Удалить указанное слово в исходном и сбалансированном дереве. Сравнить время удаления и объем памяти. Построить хеш-таблицу из слов текстового файла, задав размерность таблицы с экрана, используя метод цепочек для устранения коллизий. Вывести построенную таблицу слов на экран. Осуществить удаление введенного слова, вывести таблицу. Сравнить время удаления, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев, хеш-таблиц и файла.

Описание ТЗ

Общая концепция системы

Программа выполняет операции над деревом, AVL-деревом и хеш-таблицей. Данные в деревья и таблицу считываются из текстового файла, но при необходимости могут быть введены с клавиатуры. Пользователь может выбрать действие, которое он хочет осуществить.

Требования к функциональным характеристикам

Программа должна выполнять следующие функции:

- Добавление элемента в дерево
- Удаление элемента из дерева
- Балансировка дерева
- Построение хеш-таблицы
- Удаление слова из дерева

- Удаление слова из хеш-таблицы
- Вывод дерева в графическом виде (или предложить иную визуализацию в виде дерева)
- Вывод на экран хеш-таблицы;
- Вывод времени и количества сравнений при поиске одних и тех же данных в различных структурах данных.

На **вход** программа получает размерность хеш-таблицы, максимальное среднее значение коллизий, затем действие (номер пункта меню). При необходимости элементы типа char[30], т.е. сами элементы дерева, над которыми нужно произвести действие.

Выход должен быть представлен в виде таблицы или дерева. Также программа должна вывести данные о времени выполнения алгоритма.

Программа должна выдавать корректный ответ при вводе любых данных. Если произошла ошибка ввода, программа должна сообщить об этом.

Аварийные ситуации

- Ввод несуществующего пункта меню
    - Программа выведет сообщение “Input error”.
- Ввод значений другого типа
    - Программа выведет сообщение “Input error”.
- Отсутствие удаляемого элемента в дереве
    - Программа выведет NO THIS ELEMENT IN TREE
- Повторный ввод элемента, который уже есть в дереве
    - Программа пропустит добавление
- Ввод размерности или количества коллизий < 0
    - Программа выведет сообщение “Input error”

Способ обращения к программе

Программа представляет собой файл app.exe. Запускается в консоли. Для запуска достаточно команды ./app.exe. Если файл отсутствует можно собрать его с помощью утилиты make.

Описание структур данных

```
typedef struct st_t st;
struct st_t
{
 char word[N];
 st *next;
};  - Структура для хранения элементов хеш-таблицы.
```

```
typedef struct node_t node;
struct node_t
{
 char word[N];
 unsigned char height;
 node* left;
 node* right;
}; - Структура для хранения элементов AVL-дерева

typedef struct tree_element t_el;
struct tree_element
{
 char word[N];
 t_el *left;
 t_el *right;
}; - Структура для хренения элементов дерева

 t_el *head = NULL; - голова дерева
 node *head_node = NULL; - голова AVL-дерева
 st **htable; - хеш-таблица
 char buf[31]; - буфер для хранения введенных слов
 long int tb, te; - время
 long int t1, t2, t3; - время выполнение алгоритма
```

Описание алгоритма

Добавление элемента:
ДДП:
Пока не пустая вершина
    Если элемент равен вершине
        выйти из цикла
    Если элемент больше вершины
        рассмотреть правую вершину
    Иначе
        рассмотреть левую вершину
Вставить элемент на пустое место
AVL:
Пока не пустая вершина
    Если элемент равен вершине
        выйти из цикла
    Если элемент больше вершины
        рассмотреть правую вершину
    Иначе
        рассмотреть левую вершину
Вставить элемент на пустое место
Провести балансировку дерева
Хеш-функция:

Найти индекс элемента
Записать новый элемент последним в списке, соответствующем индексу

Удаление элемента:
ДДП:
Найти элемент и его предка
Если элемент последний
    Удалить элемент
Если у элемента один потомок
    Заменить элемент на потомка
Если у элемента два потомка
    Найти наименьший элемент в правом поддереве
    Заменить элемент на найденный
    Рекурсивно удалить найденный элемент
AVL:
Найти элемент и его предка
Если элемент последний
    Удалить элемент
Если у элемента один потомок
    Заменить элемент на потомка
Если у элемента два потомка
    Найти наименьший элемент в правом поддереве
    Заменить элемент на найденный
    Рекурсивно удалить найденный элемент
Провести балансировку дерева
Хеш-функция:
Найти индекс элемента
Найти элемент в списке
Удалить элемент из списка

Балансировка:
Если разность высот правого и левого поддеревьев стала равна 2
    Повернуть влево
Если разность высот правого и левого стала равна -2
    Повернуть вправо

Тесты

1. Несуществующий пункт меню
    Вход: 10
    Вывод: Input error
2. Некорректный ввод max (max < 0)
    Вход:
    max: -1
    Вывод: Input error
3. Отсутствие элемента в дереве
    Вход: word1
    Вывод: NO THIS ELEMENT IN TREE

Пусть дерево представлено в виде:
d->e
d->a
a->c

4. Удаление “листа”
   Вход: 3 c
   Вывод:
   d->e
   d->a

   Удаление узла с одним предком
   Вход: 3 a
   Выход:
   d->e
   d->c

   Удаление узла с двумя предками
   Вход: 3 d
   Выход:
   e->a
   a->c

   Добавление
   Вход: 2 g
   Вывод:
   d->e
   e->g
   d->a
   a->c

Оценка эффективности

Удаление элемента ДДП t, такты	Удаление элемента AVL-дерево t, такты	Удаление элемента Хеш-таблица t, такты	Удаление элемента Файл
29010	6548	4590	-
Память ДДП	Память AVL-дерево	Память Хеш-таблица	Память файл
1344	1344	1128	-
Среднее количество сравнений ДДП	Среднее количество сравнений АВЛ-дерево	Среднее количество сравнений Хеш-таблица	Среднее количество сравнений Файл
4.6	4.1	2.6	13.5

Выводы по проделанной работе

В ходе выполнения работы была реализована программа, работающая с двумя типами деревьев и хеш-таблицами. Были реализованы функции добавления, удаления и поиска для деревьев и хеш-таблицы.
Удаление элемента в хеш-таблице происходит на 84% быстрее, чем в обычном дереве, и на 30% быстрее, чем в АВЛ-дереве.
ДДП и АВЛ-дерево требуют примерно одинаковое количество памяти. а Хеш-таблица требует на 14% меньше памяти.
Эффективность хеш-таблицы зависит от среднего количества допустимых коллизий. При большом значении этого показателя, хеш-таблицы становятся не эффективными, увеличивается время удаления и среднее количество необходимых сравнений.
Таким образом, представление данных в виде хеш-таблицы эффективнее, чем в виде деревьев, а АВЛ-дерево эффективнее, чем дерево двоичного поиска.

Ответы на вопросы

1. Что такое дерево?

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

2. Как выделяется память под представление деревьев?

Память под деревья выделяется динамически. Память выделяется под элемент и два указателя на правое и левое поддерево.

3. Какие стандартные операции возможны над деревьями?

Обход, поиск, добавление и удаление элемента.

4. Что такое дерево двоичного поиска?

Дерево двоичного поиска - это бинарное дерево(у каждого элемента не больше двух потомков), в котором все левые потомки меньше предка, а все правые больше.

5. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

В идеально сбалансированном дереве количество элементов в правом и левом поддеревьях отличается не больше, чем на единицу. В АВЛ дереве, не больше, чем на единицу отличается высота дерева.

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

Поиск в АВЛ дереве гораздо быстрее посика в ДДП, так как зачастую приходится применять меньше операций сравнения, чтобы найти элемент.

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

Массив, заполненный в порядке, определенным хеш-функцией, называется хеш-таблицей.

8. Что такое коллизии? Каковы методы их устранения.

Коллизия - ситуация, когда разным ключам соответствует одно значение хеш-функции, то есть, когда $h(K1)=h(K2)$, в то время как $K1 \neq K2$.
Методы устранения: внешнее(метод цепочек) и внутреннее(открытая адресация) хеширование.

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

Когда среднее количество коллизий становится больше 3-4, поиск в хеш-таблицах становится неэффективен.

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

Эффективность хеш-таблицы зависит от среднего количества допустимых коллизий. При большом значении этого показателя, хеш-таблицы становятся не эффективными, увеличивается время удаления и среднее количество необходимых сравнений.
Авл-деревья эффективнее, чем деревья двоичного поиска, так как требуется меньше времени и сравнений, чтобы найти каждый элемент.