**Типы и структуры данных.**

**Лабораторная работа №7**

# Деревья, хеш-таблицы

**Выполнил**: Кузнецов Александр

**Группа**: ИУ7-33

**Цель работы** – построить и обработать хеш-таблицы, сравнить эффективность поиска в сбалансированных деревьях, в двоичных деревьях поиска и в хеш-таблицах.

Построить хеш-таблицу по указанным данным. Сравнить эффективность поиска в сбалансированном двоичном дереве, в двоичном дереве поиска и в хеш-таблице. 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.

**задача 4**

Построить хеш-таблицу для зарезервированных слов языка С++ (не менее 20 слов), содержащую HELP для каждого слова. Выдать на экран подсказку по введенному слову. Выполнить программу для различных размерностей таблицы и сравнить время поиска и количество сравнений. Для указанных данных создать сбалансированное дерево. Добавить подсказку по вновь введенному слову, используя при необходимости реструктуризацию таблицы. Сравнить эффективность добавления ключа в таблицу или ее реструктуризацию для различной степени заполненности таблицы.

**Входные данные:**

Номер выполнения команды;

Ввод слова поиска(keyword), удаления из хеш-таблицы/дерева

Вставка в хеш-таблицу/дерево

Ввод размера хеш-таблиц(Открытой, Закрытой)

**Пример:**

//ввод команды для хеш-таблицы/дерева

>>1

//ввод слова поиска/удаления/добавления  
>> (keyword) int

//ввод размеров хеш-таблиц

>>(Open) 2

>>(Close) 25

**Выходные данные**

Дерево поиска в PreOrder

Информация о совершенных действия

Количество сравнений в дереве/хеш-таблице

Сбалансированное дерево

DOT файл(с балансированным деревом и деревом бинарного поиска)

Хеш-таблица(Открытая и Закрытая)

**Функция программы**

Обработка хеш-таблиц, деревьев и сбалансированных деревьев, измерение времени поиска в вышеуказанных способах хранения информации, измерение количества сравнений при поиске элемента, балансировка дерева, работа с DOT-файлами.

**Аварийные ситуации**

**-** Нехватка места для формирования закрытой хеш-таблице

- Удаление несуществующего тега

**-** Вставка тега, который уже существует

**Описание использованных структур данных**

//структура (узел дерева)

struct node\_tree{

char keyword[20];

char HELP[100];

struct node\_tree \*left, \*right, \*parent;

};

//Узел хеш-таблицы

struct node\_hash{

char keyword[20];

char HELP[100];

struct node\_hash \*next;

};

**Алгоритм**

**Обработка ввода:**

1. Вывести интерфейс программы
2. Проверить корректность ввода(Интерфейс)
3. Запустить выбранную команду

-//////- Вывод сообщение об ошибке, если она есть -//////-

Иначе

**Обработка операций над хеш-таблицей и деревом:**

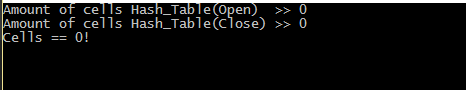
1. Вставка тега в дерево, хеш-таблицу
2. Удаление тега из хеш-таблица и дерева
3. Вывод дерева в консоль и в DOT
4. Вывод открытой хеш-таблицы
5. Вывод закрытой хеш-таблицы
6. Балансировка дерева
7. Изменение размерности хеш-таблицы
8. Поиск в бинарном дереве
9. Поиск в Открытой хеш-таблицы
10. Поиск в закрытой хеш-таблице
11. Обработка времени
12. О меню
13. Выход

**Обработка времени:**

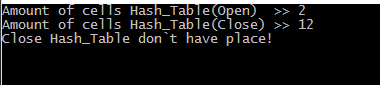
1. Расчет времени поиска в двоичном дереве
2. Расчет времени поиска в открытой хеш-таблице
3. Расчет времени поиска в закрытой хеш-таблице

**Тесты**

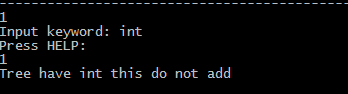
**Тест ввода:**Создание хеш-таблицы размером 0



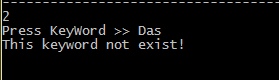
Создание закрытой хеш-таблицы размером меньше, чем хранимая информация.



Вставка элемента, который уже существует в дереве

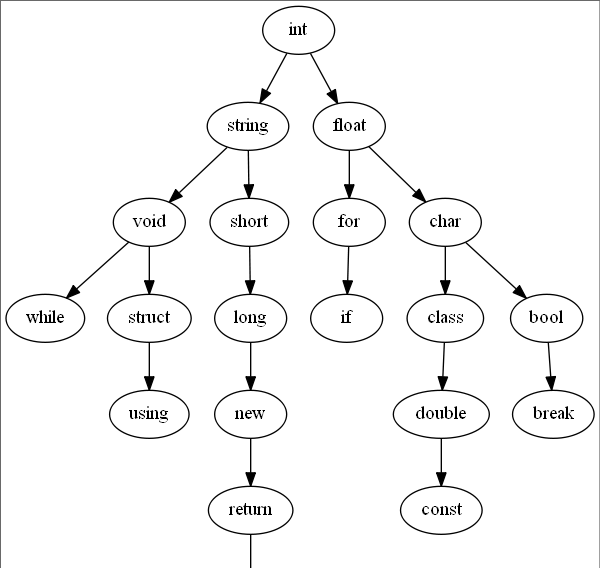


Удаление несуществующего тега

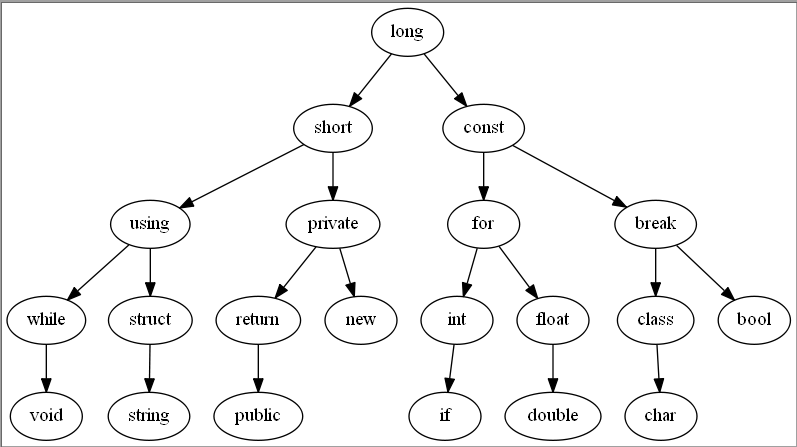


**Результат:**

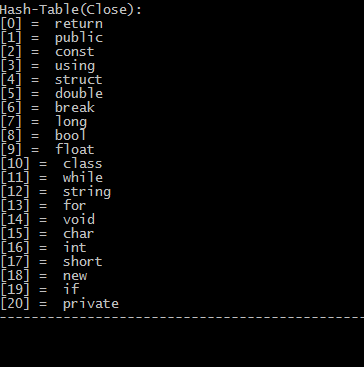
**Несбалансированное деревья**



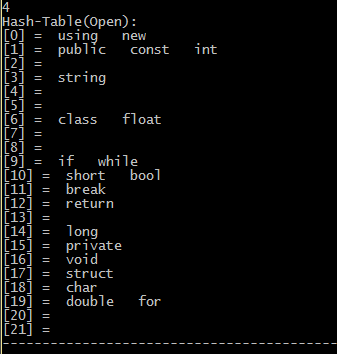
**Сбалансированное дерево**



**Закрытая хеш-таблица**



**Открытая хеш-таблица**



**Сравнение методов по времени и сравнениям.**

**Несбалансированное дерево**

|  |  |  |  |
| --- | --- | --- | --- |
| Iterations | Tree(ms) | Open(ms) | Close(ms) |
| 2000000 | 0.041 | 0.131 | 0.087 |
| Iterations | Tree(comp) | Open(comp) | Close(comp) |
| 2000000 | 1 | 3 | 1 |

**Сбалансированное дерево**

|  |  |  |  |
| --- | --- | --- | --- |
| Iterations | Tree(ms) | Open(ms) | Close(ms) |
| 2000000 | 0.15 | 0.135 | 0.086 |
| Iterations | Tree(comp) | Open(comp) | Close(comp) |
| 2000000 | 4 | 3 | 1 |

Поиск в сбалансированном дереве осуществляется медленнее, потому что при балансировке слово, которое мы ищем сместилось(сбалансировалось) от своего первоначального положения в несбалансированном дереве, поэтому количество сравнений элементов увеличилось, а следственно и время поиска.

В общем случае, время выполнения почти одинаковое, однако поиск в открытой хеш-таблице затраты по времени немного больше, т.к. происходят коллизии при добавлении элемента в открытую хеш-таблицу.

При использовании хеш-таблиц, результат напрямую зависит от используемой хеш-функции – значительное время может затрачиваться вычисление индекса. Кроме того, при закрытом хешировании результат критически зависит от порядка слов в файле – если в самом начале произойдёт коллизия на одном из первых индексов (элемент будет записан в таблице ниже чем должен), то коллизии могут возникать далее и со всеми остальными элементами. По сути, использование хеш-функции при закрытом хешировании в общем случае даёт лишь «начальное приближение» для поиска элемента.

**Вопрос – Ответ**

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**.

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 в АВЛ дереве имеет трудоёмкость О(log2n), в то время как в обычном ДДП может иметь O(n). АВЛ дерево никогда не вырождается в линейный список (исключение – дерево из двух элементов), в то время как «внешний вид» ДДП может зависеть от того, в каком порядке в него добавлялись элементы.

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

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*. Минимальная трудоемкость поиска в хеш-таблице равна О(1)!

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. В этом случае, очевидно, необходимо найти новое место для хранения ключей, претендующих на одну и ту же ячейку хеш-таблицы. Причем количество коллизий необходимо минимизировать. Таким образом, хорошая хеш-функция должна удовлетворять еще одному требованию, а именно: она должна минимизировать число коллизий.

Если ключей меньше, чем элементов массива, то в качестве хеш-функции можно принять вычисление остатка от деления целочисленного ключа на размерность массива (m), то есть:

**h(ki) = (ki mod m),**

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

Поиск в ХТ 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.

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

В хеш-таблице минимальная трудоёмкость поиска равна О(1). В АВЛ: О(log2n). В дереве двоичного поиска может достигать О(n),