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)

ФАКУЛЬТЕТ «ИНФОРМАТИКА И СИСТЕМЫ УПРАВЛЕНИЯ»

КАФЕДРА «ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ»

ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №7

Студент **Паламарчук А.Н.**

Группа **ИУ7-33Б**

Предмет **Типы и структуры данных**

Название предприятия **НУК ИУ МГТУ им. Н. Э. Баумана**

Студент _____ **Паламарчук А.Н.**

Преподаватель _____ **Никульшина Т. А.**

Преподаватель _____ **Барышникова М. Ю.**

2023 г.

Условие задачи
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. Оценить эффективность использования этих структур (по времени и по памяти) для поставленной задачи.

Техническое задание
Используя предыдущую программу (задача №6), построить дерево, например, для следующего выражения: 9+(8*(7+(6*(5+4)-(3-2))+1)). При постфиксном обходе дерева, вычислить значение каждого узла и результат записать в его вершину. Получить массив, используя инфиксный обход полученного дерева. Построить для этих данных дерево двоичного поиска (ДДП), сбалансировать его. Построить хеш-таблицу для значений этого массива. Осуществить поиск указанного значения. Сравнить время поиска, объем памяти и количество сравнений при использовании ДДП, сбалансированных деревьев и хеш-таблиц.

Входные данные

- Команда (число) из меню программы (см. меню программы) – обозначение необходимой операции.

Выходные данные

- Таблицы эффективности
- Файл в DOT формате
- Файлы png с деревьями
- Хеш-таблицы
- Результат поиска в двоичном дереве поиска
- Результат поиска в открытой хеш-таблице
- Результаты поиска в закрытой хеш-таблице

Возможные аварийные ситуации

- Некорректный исходный файл
- Ошибка открытия файла
- Отсутствие исходного файла
- Некорректная команда

Описание внутренних структур данных

```
typedef struct tree_node
{
    const char *name;
    int val;
    // родитель
    struct tree_node *parent;
    // меньшие
    struct tree_node *left;
    // большие
    struct tree_node *right;
} tree_node_t;

typedef struct hash_node
{
    int key;
    int value;
    struct hash_node *node;
} hash_node_t;

typedef struct hash_table
{
    size_t size;
    hash_node_t **array;
} hash_table_t;
```

Константы

```
 #define _POSIX_C_SOURCE 199309L
 #define COUNT_TESTS 100
#define FILE_DOT "graph.gv"
#define FILE_PNG "graph.png"
#define FILE_BST_DOT "graph_BST.gv"
#define FILE_BST_PNG "graph_BST.png"
```

Используемые функции

```
int fill_tree(struct val_nodes *tree);
```

Заполнение дерева значениями пользователя

```
void export_to_dot(FILE *f, const char *tree_name, struct tree_node
*tree);
```

Перевод дерева в DOT формат

```
void apply_pre(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Префиксный обход дерева

```
void apply_in(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Инфиксный обход дерева

```
void apply_post(tree_node_t *tree, void (*f)(tree_node_t*, void*), void
*arg);
```

Постфиксный обход дерева

```
tree_node_t* balance_BST(int *arr, size_t asize);
```
Построение сбалансированного дерева

```
int create_node(tree_node_t **new_node, int data);
```
Создает элемент хеш-таблицы

```
void fill_hash_table(hash_table_t *table, int *a, size_t asize);
```
Заполнение закрытой хеш-таблицы

```
void fill_hash_table_open(hash_table_t *table, int *a, size_t asize);
```
Заполнение открытой хеш-таблицы

```
tree_node_t* search_from_tree(tree_node_t *node, int data, size_t *cmp);
```
Поиск в двоичном дереве поиска

```
hash_node_t* search_from_hash_table(hash_table_t* table, int key, size_t
*cmp);
```
Поиск в закрытой хеш-таблице

```
hash_node_t* open_htable_search(hash_table_t* table, int key, size_t
*cmp);
```
Поиск в открытой хеш-таблице

```
int cmp_tree_vs_table(void);
```
Сравнение двоичного дерева поиска и хеш-таблиц

Меню программы

Программа обрабатывает нужную команду:

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы
```

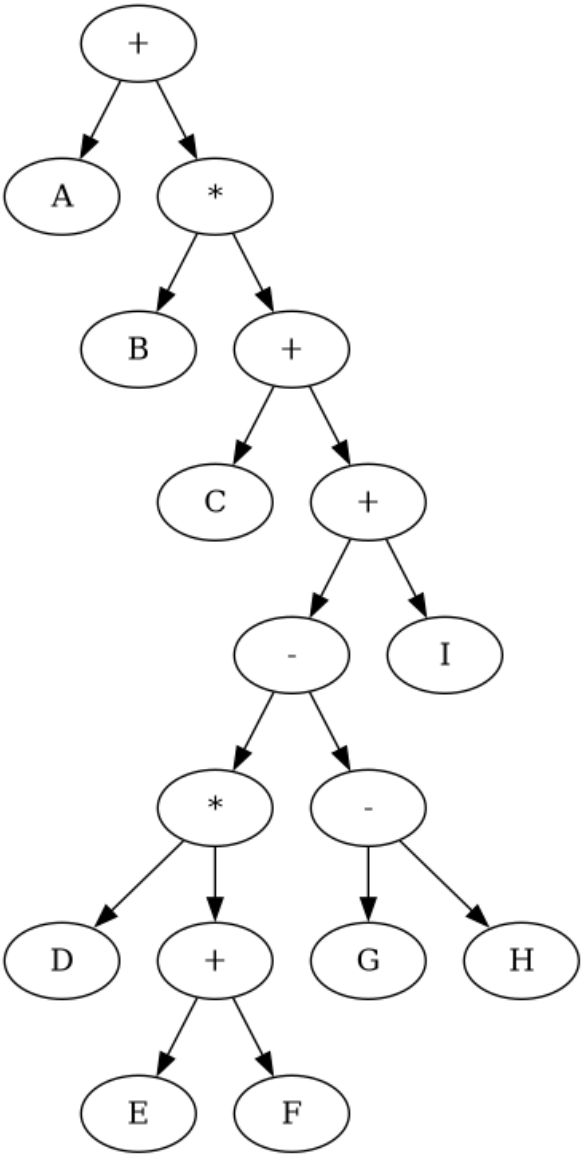
Хеш-функция

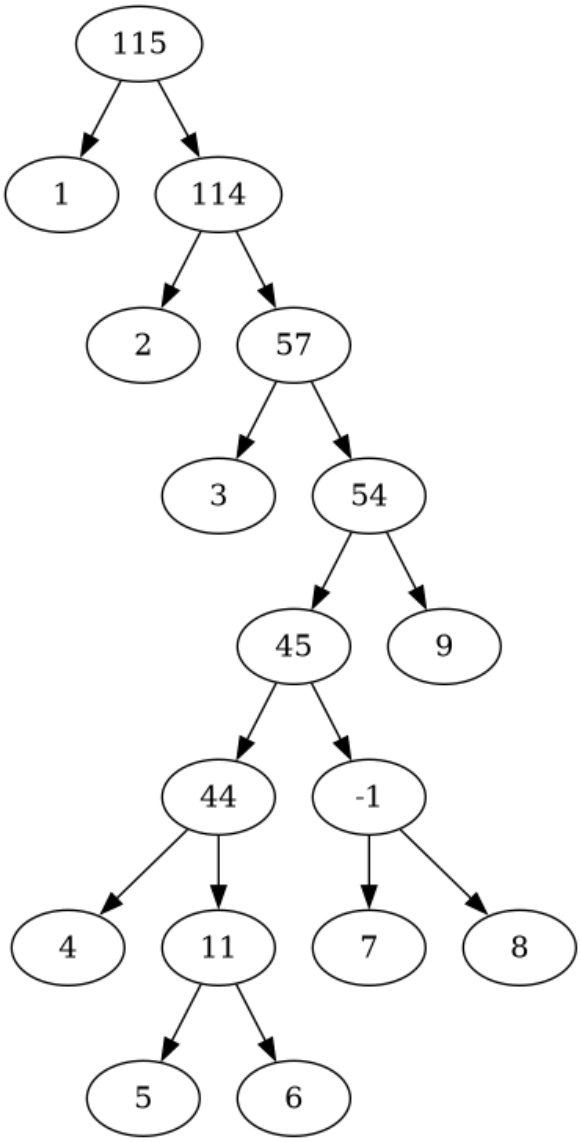
```
// key – значение для хеширования
// size – размер хеш-таблицы
int hash(int key, int size)
{
    return key % size;
}
```

Реструктуризация

При коллизии (более 10 элементов с одним хешем) в хеш-таблице, выполняется реструктуризация. Увеличивается размер хеш-таблицы в 2 раза и перезаписывается в новую хеш-таблицу.

Пример работы программы





```
6
Хеш-таблица закрытого типа
---------------------------------------
|   №|                 Key|               Value|
---------------------------------------
|   0|                   -|                   -|
|   1|                   1|                   1|
|   2|                   2|                   2|
|   3|                   3|                   3|
|   4|                   4|                   4|
|   5|                   5|                   5|
|   6|                   6|                  57|
|   7|                   7|                   6|
|   8|                   8|                   7|
|   9|                   9|                   8|
|  10|                  10|                  44|
|  11|                  11|                  11|
|  12|                  12|                 114|
|  13|                  13|                 115|
|  14|                   -|                   -|
|  15|                  15|                  45|
|  16|                   -|                   -|
|  17|                  17|                  -1|
|  18|                  18|                   9|
|  19|                  19|                  54|
|  20|                   -|                   -|
|  21|                   -|                   -|
|  22|                   -|                   -|
|  23|                   -|                   -|
|  24|                   -|                   -|
|  25|                   -|                   -|
|  26|                   -|                   -|
|  27|                   -|                   -|
|  28|                   -|                   -|
|  29|                   -|                   -|
|  30|                   -|                   -|
|  31|                   -|                   -|
|  32|                   -|                   -|
|  33|                   -|                   -|
---------------------------------------
```

```
Хеш-таблица открытого типа
---------------------------------------
|   №|                 Key|               Value|
---------------------------------------
|   0|                   -|                   -|
|   1|                   1|                   1|
|    |                   1|                  -1|
|   2|                   2|                   2|
|   3|                   3|                   3|
|    |                   3|                  54|
|   4|                   4|                   4|
|   5|                   5|                   5|
|   6|                   6|                  57|
|    |                   6|                   6|
|   7|                   7|                   7|
|   8|                   8|                   8|
|   9|                   9|                   9|
|  10|                  10|                  44|
|  11|                  11|                  11|
|    |                  11|                  45|
|  12|                  12|                 114|
|  13|                  13|                 115|
|  14|                   -|                   -|
|  15|                   -|                   -|
|  16|                   -|                   -|
---------------------------------------
```

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

7
Введите искомое значение: 44
Элемент найден, его название: 12

Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

7
Введите искомое значение: 98
Элемент не найден

Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

8
Введите искомое значение: 115
Элемент найден, его хеш: 13

Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

8
Введите искомое значение: 67
Элемент не найден
```

```
Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

9
Введите искомое значение: 54
Элемент найден, его хеш: 3

Комманды:
1 - Ввести значения переменных: от A до I
2 - Вывести бинарное дерево выражения A + (B * (C + (D * (E + F) - (G - H)) + I))
3 - Вычислить значение каждого узла и результат записать в его вершину
4 - Построить сбалансированное дерево двоичного поиска
5 - Сформировать хеш-таблицы
6 - Вывести хеш-таблицы
7 - Поиск в дереве двоичного поиска
8 - Поиск в хеш-таблице (закрытого типа)
9 - Поиск в хеш-таблице (открытого типа)
10 - Сравнить поиск в дереве двоичного поиска и в хеш-таблицах
0 - Завершить работу программы

9
Введите искомое значение: 59
Элемент не найден
```

Производительность

- Замер для структур, построенных с помощью массива, упорядоченного по возрастанию.

```
TIME
-------------------------------------------------------------------------------------------------------
|      |                          Tree|              Close hash table|               Open hash table|
-------------------------------------------------------------------------------------------------------
|     N|          Time|       Compare|          Time|       Compare|          Time|       Compare|
-------------------------------------------------------------------------------------------------------
|   100|           107|             7|            86|             2|            80|             2|
|   200|           126|             8|            92|             2|            82|             2|
|   300|           125|             9|            85|             2|            80|             2|
|   400|           123|             9|            85|             2|            82|             2|
|   500|           122|             9|            86|             2|            77|             2|
|   600|           133|            10|            86|             2|            79|             2|
|   700|           131|            10|            86|             2|            78|             2|
|   800|           132|            10|            86|             2|            78|             2|
|   900|           131|            10|            85|             2|            79|             2|
|  1000|           131|            10|            85|             2|            80|             2|
-------------------------------------------------------------------------------------------------------
```

MEMORY

N	Tree	Close table	Open table
100	4000	3216	2416
200	8000	6416	5616
300	12000	9616	9616
400	16000	12816	14416
500	20000	16016	20016
600	24000	19216	26416
700	28000	22416	33616
800	32000	25616	41616
900	36000	28816	50416
1000	40000	32016	60016

- Замер для структур, построенных с помощью неупорядоченного массива.

Тестирование проведено 100 раз
Время указано в наносекундах, затраты памяти в байтах

TIME

	Tree		Close hash table		Open hash table	
N	Time	Compare	Time	Compare	Time	Compare
100	108	7	130	8	80	2
200	115	8	123	7	80	2
300	124	9	121	7	79	2
400	123	9	139	9	80	2
500	123	9	115	6	80	2
600	133	10	137	9	80	2
700	132	10	160	12	80	2
800	132	10	149	10	80	2
900	133	10	163	12	80	2
1000	131	10	172	13	80	2

MEMORY

N	Tree	Close table	Open table
100	4000	3216	2304
200	8000	6416	5136
300	12000	9616	8704
400	16000	12816	12976
500	20000	16016	17888
600	24000	19216	23296
700	28000	22416	29280
800	32000	25616	35872
900	36000	28816	43152
1000	40000	32016	51184

Сравним скорость поиска элемента. Скорость поиска по хеш-таблицам выше скорости поиска по бинарному дереву поиска и хеш-таблицы занимают меньше места (каждый узел дерева больше узла списка). Однако при возникновении достаточно большого числа коллизий закрытая хеш-таблица начинает уступать бинарному дереву поиска.

Выводы по проделанной работе

Для быстрого поиска данных можно использовать такие структуры, как деревья поиска и хеш-таблицы. Деревья являются хорошим вариантом поиска, но они занимают (сравнительно) большое количество памяти и поиск по ним происходит с минимальной сложностью O(log n).
Для ещё более быстрого поиска данных можно использовать хеш-таблицы, которые позволяют не только ускорить поиск, но и уменьшить количество занимаемой памяти. Однако в хеш-таблицах могут возникать коллизии, для избавления от которых можно использовать списки или алгоритмы поиска новых мест в массиве.

Поэтому для хранения данных, в которых нужно будет выполнять быстрый поиск, выгоднее всего использовать хеш-таблицы, единственный недостаток которых - необходимость в реструктуризации при большом количестве коллизий.

Контрольные вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Идеально сбалансированное дерево заполняется путём равномерного распределения вершин по ветвям и, в отличие от АВЛ дерева не является отсортированным (то есть не всегда является деревом поиска)

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

Алгоритм поиска ничем не отличается. Просто в среднем сравнений придётся произвести меньше.

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

Массив, позиция элементов в котором определяется хеш-функцией, которая позволяет определить позицию каждого элемента.

4. Что такое коллизии? Каковы методы их устранения.

Коллизия - это ситуация, когда хеш-функция присваивает двум разным элементам одинаковый индекс. В зависимости от того, закрытая или открытая хеш-таблица, коллизии устраняются
для первого случая - поиском новой позиции чуть дальше (index + k^2), где k - номер попытки поиска нового места.

для второго случая - записью элемента в конец списка, который содержится в ячейке по хеш индексу.

5. В каком случае поиск в хеш-таблицах становится неэффективен? В случаях, когда количество коллизий становится слишком большим. В таких случаях хеш-таблицу реструктуризируют.

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

- в хеш-таблицах минимальное - O(1)
- в АВЛ - O(log n)
- в двоичном дереве поиска от O(log n) до O(n)