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 **ПРОГРАММНАЯ ИНЖЕНЕРИЯ (ИУ7)**

О Т Ч Е Т

По лабораторной работе № 7

Название: Сбалансированные деревья, хеш–таблицы

Дисциплина: Типы и структуры данных

Вариант: 6

Студент ИУ7-31Б (Группа) Савинова М. Г. (Фамилия И. О.)

Преподаватель Силантьева А. В.

Москва, 2022

Оглавление

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 и внутреннем хешировании.

Условие задачи

Используя бинарное дерево следующего выражения: **9+(8*(7+(6*(5+4)-(3-2))+1))**, и процедуру постфиксного обхода дерева:

- вычислить значение каждого узла и результат записать в его вершину;
- получить массив, используя процедуру инфиксного обхода 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.

Описание ТЗ

1. Описание входных данных

Исходными данными, хранящимися в во всех структурах данных, являются **целые числа**.

Ввод данных происходит с помощью выбора 0-8 пункта в меню.

```
Действия:
          1 - Ввод коэффициентов
          2 - Запись выражения в дерево выражений
          3 - Создание ДДП/АВЛ-дерева/Хеш-таблицы
          4 - Поиск значения в ДДП/АВЛ-дереве/Хеш-таблице
          5 - Вывод ДДП/АВЛ-дерева в файлы
          6 - Вывод Хеш-таблиц в файл
          7 - Сравнение эффективностей СД
          0 - Выход

Введите выбранный пункт:
```

Некорректный ввод вызывает завершение программы с ненулевым кодом возврата.

2. Ограничения

- Элементы дерева выражения:
  - **Данные:** целые попарно различные числа **от -65536 до 65535;**
- Элементы двоичного дерева поиска:
  - **Данные:** целые попарно различные числа **от -65536 до 65535;**
- Элементы АВЛ-дерева:
  - **Данные:** целые попарно различные числа **от -65536 до 65535;** знак операции;
- Элементы хеш-таблицы:
  - **Данные:** целые попарно различные числа от **-65536 до 65535;**
- Пункт меню: целое число **от 0 до 8;**

3. Описание выходных данных

Опции 1-3:

В результате выполнения программы будет сформировано дерево выражения, переведено выражение в постфиксную форму.

Опция 4:

В результате выполнения программы будет осуществлен поиск введенного значения.

Опция 5-6:

В результате выполнения программы имеющиеся СД будут записаны в файлы.

Опция 7:

В результате выполнения программы будет осуществлены замерные эксперименты.

4. Описание задачи, реализуемой в программе

Программа реализует:

- формирование различных СД на основе массива;

- поиск значения в различных СД;
- сравнение эффективности использования тех или иных СД;
- вывод СД в файл.

5. Способ обращения к программе

Обращение к программе происходит через консоль, путём запуска *.exe файла.

6. Описание возможных аварийных ситуаций и ошибок пользователя

Аварийная ситуация	Код завершения	Сообщение
Дерево не инициализировано	2	`Пустое дерево!`
Неудачное выделение памяти	5	`Ошибки выделения памяти!`
Неверный выбор пункта меню	6	`Неверный пункт меню!`
Некорректные данные	7	`Неверное значение!`
Добавление в дерево уже имеющегося значения	8	`Одинаковые значения в дереве!`
Запись в файл пустой таблицы	9	`Пустая таблица!`

Описание внутренних структур данных

```
// структура для описания узла дерева
typedef struct node_tree node_tree_t;
struct node_tree
{
    int value;          // значение
    char operand;       // является ли операцией

    int height;         // высота поддерева
    node_tree_t *left;  // указатель на левого «потомка»
    node_tree_t *right; // указатель на правого «потомка»
};
// ячейка таблицы
typedef struct
```

```
{
    int key;                    // значение
    unsigned short int empty; // является ли ячейка занятой
} cell_t;

// ячейка таблицы (метод цепочкой)
typedef struct list_cell list_cell_t;
struct list_cell
{
    int key;            // значение

    list_cell_t *next; // указатель на следующий элемент
};

// структкура самой таблицы
typedef struct
{
    int max_size; // максимальный размер таблицы

    void *cells;  // указатель на таблицу
} hash_table_t;
```

Описание алгоритма

1. Выводится меню данной программы
2. Пользователь выбирает пункт меню
3. Выход из программы осуществляется в случае ошибки или пункта меню - «0»

Тестовые данные

1. Позитивные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Ввод к-тов выражения	1 0 1 2 3 4 5 6 7 8	1 2 3 4 5 6 + * 7 8 - - + 9 + * +

2.	Запись выражения в дерево; получение массива значений	1 0 1 2 3 4 5 6 7 8 2	Выражение записано в дерево! Значение, вычисленное с помощью дерева: 115 Массив, полученный инфиксным обходом до удаления повторов: 1 115 2 114 3 48 4 44 5 11 6 45 7 -1 8 57 9 Массив, полученный инфиксным обходом после удаления повторов: 1 115 2 114 3 48 4 44 5 11 6 45 7 -1 8 57 9
3.	Ввод к-тов и создание СД	1 0 1 2 3 4 5 6 7 8 2 3	ДДП создано! АВЛ-дерево создано! Хеш-таблица с открытой адресацией создана! Хеш-таблица методом цепочек создана!
4.	Запись полученных СД в файлы	1 0 1 2 3 4 5 6 7 8 2 3 5	Дерево выражений записано в файл! ДДП записано в файл! АВЛ-дерево записано в файл! Хеш-таблица с внутренним хешрованием записано в файл! Хеш-таблица с внешним хешрованием записано в файл!

		6	
5.	Сравнение эффективности различных СД	1 0 1 2 3 4 5 6 7 8 2 3 7	*Вывод программы далее

2. Негативные тесты

№	Наименование теста	Пользовательский ввод	Вывод
1.	Неверно выбранный пункт меню	18	Неверный пункт меню!
2.	Инициализация дерева без введенных к-тов	2	Неверное значение!
3.	Получение остальных СД без к-тов	3	Пустое дерево!
5.	Сравнение эффективностей без введенных к-тов	7	Пустое дерево!
5.	Запись неинициализированных СД в файлы	5 6	Пустое дерево!
6.	Ввод одинаковых значений в дерево	10 5 1 1 1	Неверное значение!

		1 1	

Оценка эффективности

Были проведены замеры времени по вычислению значения исходного выражения:

A + (B * (C + (D * (E + F) - (G-H)) + I))

Результаты замеров представлены в виде таблицы:

Для более наглядной разницы в полученных результатах, поиск каждого значения осуществляется 10.000 раз:

```
Введите среднее кол-во сравнений для поиска в таблице: 1
* Кол-во итераций: 10000

+-------------------------+-----------------------------+----------------+--------------------------+
|                         | Время поиска 17 эл-ов(мкс.) | Объем (байтов) | Среднее кол-во сравнений |
+-------------------------+-----------------------------+----------------+--------------------------+
| Двоичное дерево поиска  |                        1386 |            476 |                       11 |
+-------------------------+-----------------------------+----------------+--------------------------+
| Сбалансированное ДДП    |                         538 |            544 |                        4 |
+-------------------------+-----------------------------+----------------+--------------------------+
| Внутреннее хеширование  |                         697 |            152 |                        3 |
+-------------------------+-----------------------------+----------------+--------------------------+

* После реструктуризации внутреннего хеширования:
+-------------------------+-----------------------------+----------------+--------------------------+
| Внутреннее хеширование  |                         410 |            184 |                        1 |
+-------------------------+-----------------------------+----------------+--------------------------+

+-------------------------+-----------------------------+----------------+--------------------------+
| Внешнее хеширование     |                         438 |            304 |                        2 |
+-------------------------+-----------------------------+----------------+--------------------------+

* После реструктуризации внешнего хеширования:
+-------------------------+-----------------------------+----------------+--------------------------+
| Внешнее хеширование     |                         487 |            368 |                        1 |
+-------------------------+-----------------------------+----------------+--------------------------+
```

В результате первого замера можно сделать вывод об эффективности использования хеш-таблицы по **памяти** по сравнению с деревом, поскольку помимо самого значения, им необходимо еще содержать доп. информацию о «потомках», высоте поддерева.

Ввиду превышения среднего кол-ва сравнений была произведена **реструктуризация** таблицы как с внешним, так и с внутренним хешированием:

в результате её выполнения кол-во сравнений уменьшилось ($3 \rightarrow 1$; $2 \rightarrow 1$), а также

время поиска для внутреннего хеширования.

При повторном запуске замеров реструктуризация не производилась, так как среднее кол-во сравнений соответствует введенному кол-ву:

```
Введите среднее кол-во сравнений для поиска в таблице: 1
* Кол-во итераций: 10000

+-------------------------+-----------------------------+----------------+--------------------------+
|                         | Время поиска 17 эл-ов(мкс.) | Объем (байтов) | Среднее кол-во сравнений |
+-------------------------+-----------------------------+----------------+--------------------------+
| Двоичное дерево поиска  |                         588|             476|                         4|
+-------------------------+-----------------------------+----------------+--------------------------+
| Сбалансированное ДДП    |                         464|             544|                         3|
+-------------------------+-----------------------------+----------------+--------------------------+
| Внутреннее хеширование  |                         407|             184|                         1|
+-------------------------+-----------------------------+----------------+--------------------------+
+-------------------------+-----------------------------+----------------+--------------------------+
| Внешнее хеширование     |                         410|             368|                         1|
+-------------------------+-----------------------------+----------------+--------------------------+
```

Из полученных данных можно сделать вывод об эффективности использования АВЛ дерева по сравнению с ДДП. Наименьшее кол-во сравнений дает хеш-таблицы.

Выводы

В ходе работы были изучены способы и алгоритмы работы с хеш-таблицами и деревьями, а также проведены замерные эксперименты, по результатам которых можно сделать вывод, что реализованный алгоритм выполняет поиск элемента быстрее всего в хеш-таблице. Так же немаловажную роль в этом играет выбор корректной хеш-функции.

*Ответы на контрольные вопросы

1. Чем отличается идеально сбалансированное дерево от АВЛ дерева?

Узлы при добавлении в идеально сбалансированное дерево располагаются равномерно слева и справа. Получается дерево, у которого
число вершин в левом и правом поддеревьях отличается не более, чем на единицу. В то время как АВЛ-дерево – сбалансированное двоичное дерево, у каждого узла которого высота двух поддеревьев отличается не более чем на единицу.

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

Временная сложность поиска элемента в АВЛ дереве – **O(log2n)**
Временная сложность поиска элемента в дереве двоичного поиска – **от O(log2n) до O(n).**

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

Массив, заполненный в порядке, который определяется хеш-функцией, называется **хеш-таблицей**. Функция, которая позволяет вычислить этот индекс, называется **хеш-функцией**. Принято считать, что хорошей является такая функция, которая удовлетворяет следующим условиям:

- функция должна быть простой с вычислительной точки зрения;
- функция должна обеспечить равномерное распределение ключей в хеш-таблице;
- функция должна минимизировать число коллизий.

4. 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 ≠ K2.**

Первый метод – внешнее(открытое) хеширование (метод цепочек). В случае, когда элемент таблицы с индексом, который вернула хеш-функция, уже занят, к нему присоединяется **связный список**.

Второй метод - внутреннее (закрытое) хеширование (открытая адресация). Оно состоит в том, чтобы полностью отказаться от ссылок. В этом случае, если ячейка с вычисленным индексом занята, то можно просто просматривать следующие записи таблицы по порядку (с шагом 1), до тех пор, пока не будет найден ключ K или пустая позиция в таблице.

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

В случае большого числа коллизий. Тогда вместо ожидаемой сложности **O(1)** получаем сложность **O(n).**

В случае **открытого хеширования** (цепочки) поиск в списке

осуществляется простым перебором, так как при грамотном выборе хеш-функции любой из списков оказывается достаточно коротким.
Если же хеширование закрытое, необходимо просматривать все ячейки, если есть много коллизий.

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

Хеш-таблица - от O(1) до O(n);

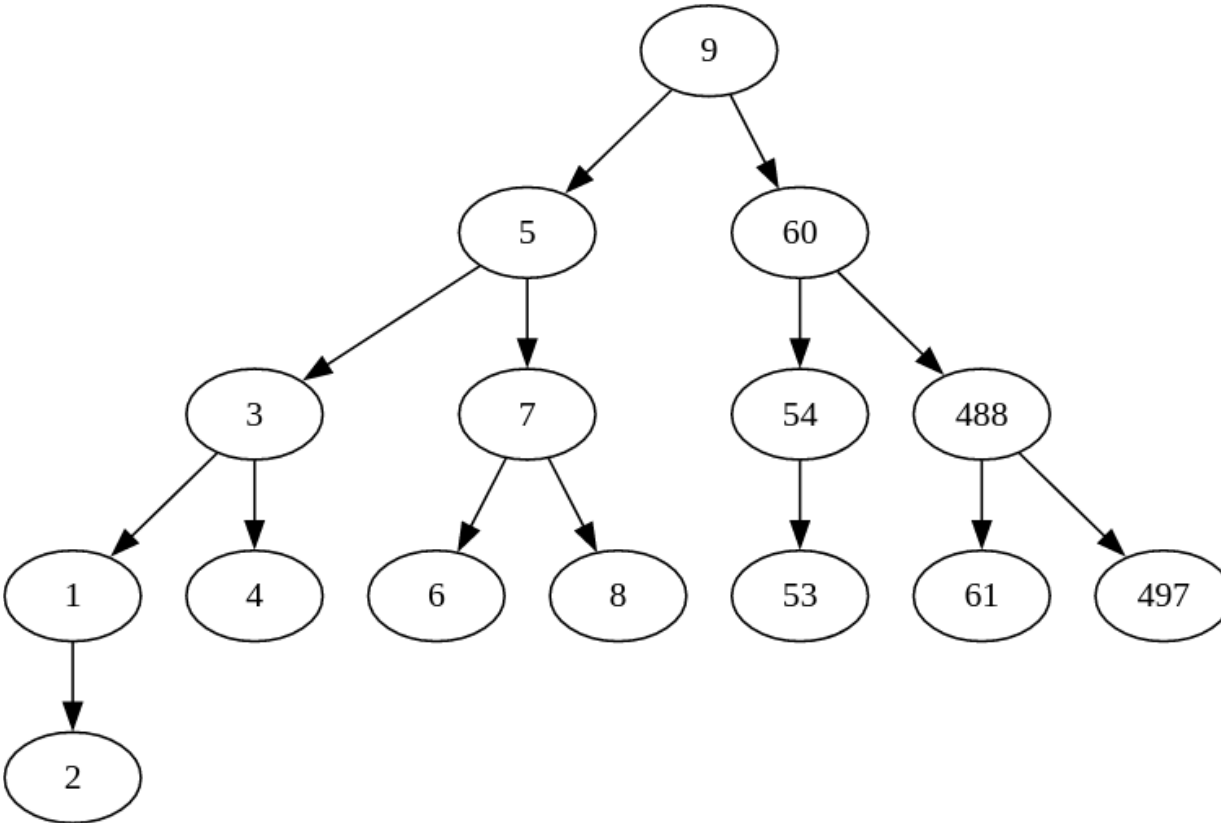
АВЛ-дерево — O(log2n);

Дерево двоичного поиска – от O(log2n) до O(n);

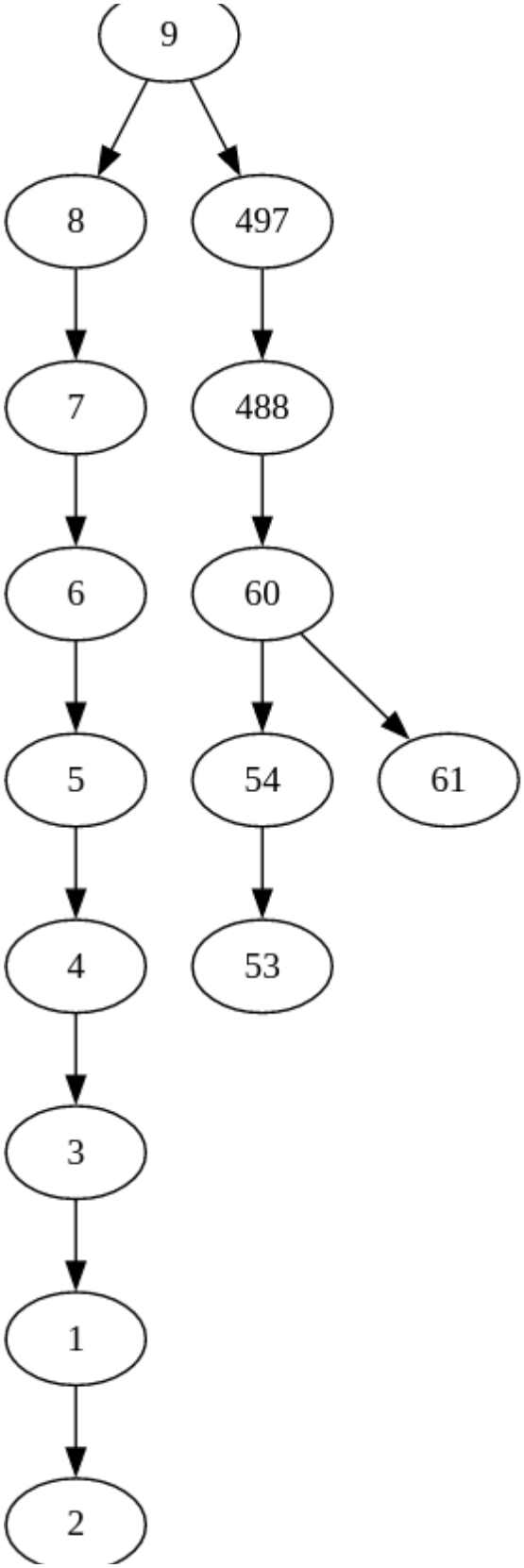
***Приложение**

9+(8*(7+(6*(5+4)-(3-2))+1))

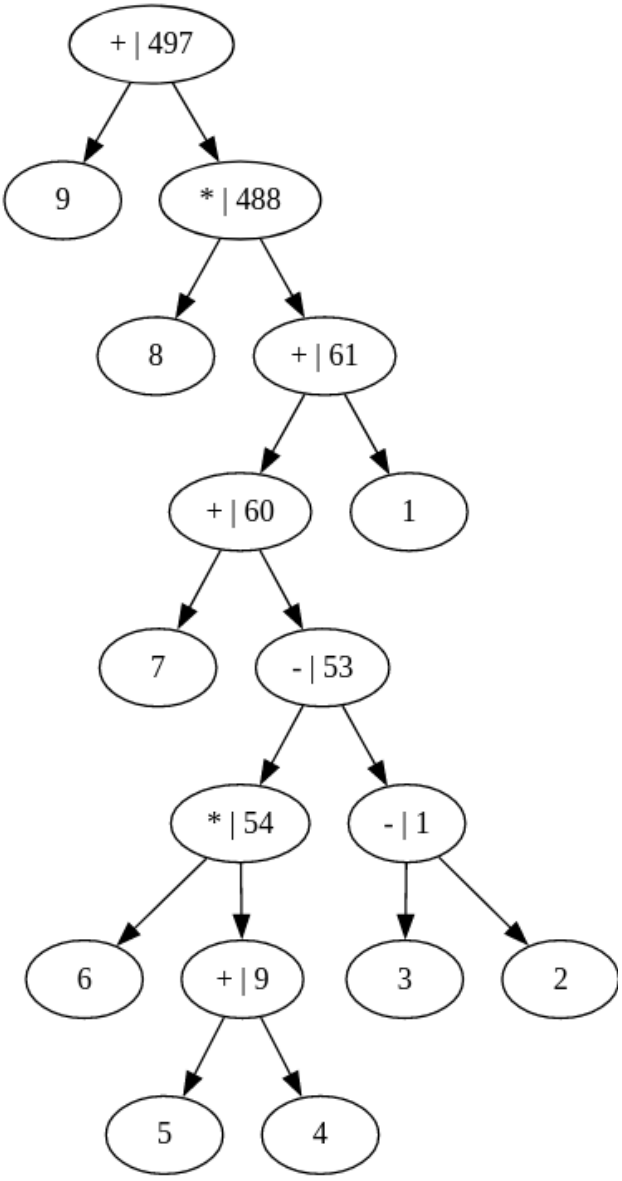
1. АВЛ-дерево



2. ДДП

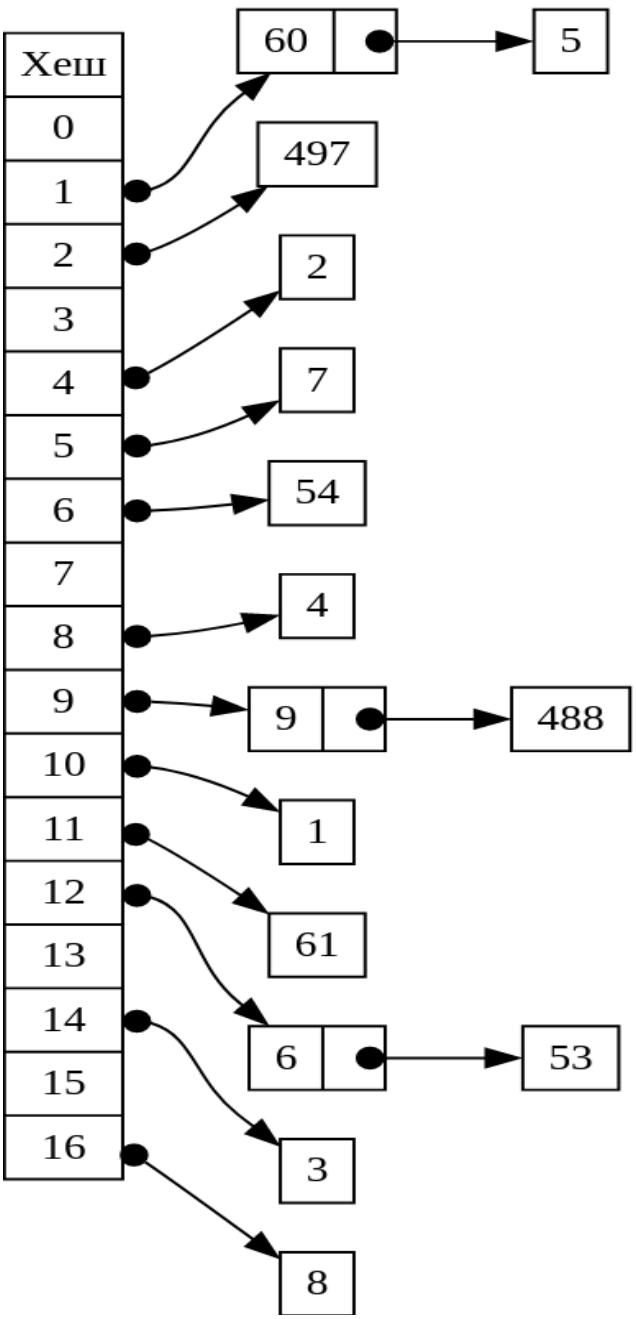


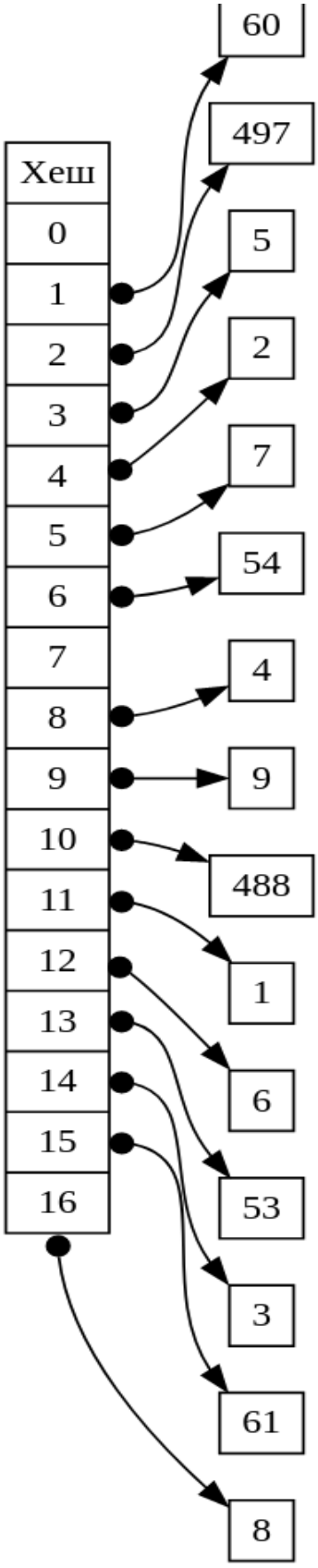
3. Дерево выражений



4. Внешнее хеширование

5. Внутреннее хеширование


Хеш
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
60
5
497
2
7
54
4
9
488
1
61
6
53
3
8


Хеш
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
60
497
5
2
7
54
4
9
488
1
6
53
3
61
8