Отчет о выполнении задания 1  
«**Бинарные деревья поиска и хеш-таблицы**»

**Хотылев Александр Сергеевич**

Группа ИВ-621

[Zerling14@gmail.com](mailto:hegemonies@yandex.ru)

# Описание алгоритмов

В задании 2 требовалось реализовать на языке C две библиотеки для работы с бинарным дере- вом поиска (Binary search tree) и хеш-таблицей (Hash table). Ключом в обоих слу- чаях является строка (char []), а значением целое число (int). Провести эксперименты 1 и 4 для модуля с бинарными деревьями поиска и эксперимент 6 (хеш-функции KP, ELF) для модуля с хеш-таблицами, в соответствии с вариантом.

1. Бинарное дерево поиска

Бинарное дерево поиска (англ. binary search tree, BST) — это двоичное дерево, для которого выполняются следующие дополнительные условия (свойства дерева поиска):

\* Оба поддерева — левое и правое — являются двоичными деревьями поиска.

\* У всех узлов левого поддерева произвольного узла X значения ключей данных меньше, нежели значение ключа данных самого узла X.

\* В то время, как значения ключей данных у всех узлов правого поддерева (того же узла X) больше, нежели значение ключа данных узла X.

Поиск элемента (lookup)

Дано: дерево Т и ключ K.

Задача: проверить, есть ли узел с ключом K в дереве Т, и если да, то вернуть ссылку на этот узел.

Алгоритм:

\* Если дерево пусто, сообщить, что узел не найден, и остановиться.

\* Иначе сравнить K со значением ключа корневого узла X.

\* Если K=X, выдать ссылку на этот узел и остановиться.

\* Если K>X, рекурсивно искать ключ K в правом поддереве Т.

\* Если K<X, рекурсивно искать ключ K в левом поддереве Т.

Вычислительная сложность в худшем случае O(n), где n — это количество ключей

Вычислительная сложность в среднем случае O(log n).

Сложность по памяти O(n).

Добавление элемента (add)

Дано: дерево Т и пара (K, V).

Задача: вставить пару (K, V) в дерево Т (при совпадении K, заменить V).

Алгоритм:

\* Если дерево пусто, заменить его на дерево с одним корневым узлом ((K, V), null, null) и остановиться.

\* Иначе сравнить K с ключом корневого узла X.

\* Если K>X, циклически добавить (K, V) в правое поддерево Т.

\* Если K<X, циклически добавить (K, V) в левое поддерево Т.

\* Если K=X, заменить V текущего узла новым значением (хотя можно и организовать список значений V, но это другая тема).

Вычислительная сложность в худшем случае O(n), где n — это количество ключей

Вычислительная сложность в среднем случае O(log n).

Сложность по памяти O(n)

2. Хеш-табли́ца

Хеш-табли́ца (англ. Hash table) — это структура данных, реализующая интерфейс ассоциативного массива, а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

Выполнение операции в хеш-таблице начинается с вычисления хеш-функции от ключа. Получающееся хеш-значение i = hash(key) играет роль индекса в массиве H. Затем выполняемая операция (добавление, удаление или поиск) перенаправляется объекту, который хранится в соответствующей ячейке массива H[i].

Каждая ячейка массива H является указателем на связный список (цепочку) пар ключ-значение, соответствующих одному и тому же хеш-значению ключа. Коллизии просто приводят к тому, что появляются цепочки длиной более одного элемента.

Поиск элемента (lookup)

Дано: хеш-таблица H и ключ K.

Задача: проверить, есть ли узел с ключом K в хеш-таблице H, и если да, то вернуть ссылку на этот узел.

Алгоритм:

\* Вычислить хеш от ключа K: i = hash(K).

\* Обратиться к элементу массива с индексом i.

\* Выполнить поиск в связном списке по ключу K.

Вычислительная сложность в худшем случае O(n), где n — это количество ключей

Вычислительная сложность в среднем случае O(1 + n/h).

Сложность по памяти O(n).

Добавление элемента (add)

Дано: хеш-таблица H и пара (K, V).

Задача: вставить пару (K, V) в хеш-таблицу H.

Алгоритм:

\* Вычислить хеш от ключа K: i = hash(K).

\* Обратиться к элементу массива с индексом i.

\* Добавить элемент (K, V) в начало связного списка.

Вычислительная сложность в худшем случае O(1), где n — это количество ключей

Вычислительная сложность в среднем случае O(1).

# Организация экспериментов

* Эксперименты проводились на ПК.  
  (CPU: Intel Core i5-3210M 2.5 Ghz, RAM: 8GB, HDD 150GB)
* Операционная система Windows 7 (компилятор gcc 5.3.0)
* Ключи компиляции программы (см. README): gcc -o -c

**Эксперимент 1. Сравнение эффективности поиска элементов в бинарном дереве поиска и хеш-таблице в среднем случае (average case)**

Требуется заполнить таблицу 1 и построить графики зависимости времени t выполнения операции поиска (lookup) элемента в бинарном дереве поиска и хеш- таблице от числа n элементов уже вставленных в словарь. В качестве ключей использовать слова из романа Л.Н. Толстого «Война и Мир» (можно использовать любой текстовый файл с большим числом слов). Файл включен в архив.

Количество элементов в словаре Binary search tree Hash table

|  |  |  |
| --- | --- | --- |
| **N** | **T (BST)** | **T(Hash)** |
| 10000 | 0,000008 | 0,000004 |
| 20000 | 0,00001 | 0,000007 |
| 30000 | 0,000012 | 0,000008 |
| 40000 | 0,000016 | 0,000009 |
| 50000 | 0,000017 | 0,00001 |

**Эксперимент 3. Сравнение эффективности поиска элементов в бинарном дереве поиска и хеш-таблице в худшем случае (worst case)**

Требуется заполнить таблицу 3 и построить графики зависимости времени t выполнения операции поиска (lookup) элемента в бинарном дереве поиска и хеш- таблице от числа n элементов уже вставленных в словарь. Добавляем в словарь n слов - от меньших к большим (например, слова «aaaaa», «bbbbb», ...). В качестве искомого ключа следует выбрать слово, которое вставлено последним.

|  |  |  |
| --- | --- | --- |
| **N** | **T (BST)** | **T(Hash)** |
| 10000 | 0,00002 | 0,000005 |
| 20000 | 0,000028 | 0,000009 |
| 30000 | 0,000035 | 0,000011 |
| 40000 | 0,00004 | 0,000012 |
| 50000 | 0,000045 | 0,000014 |

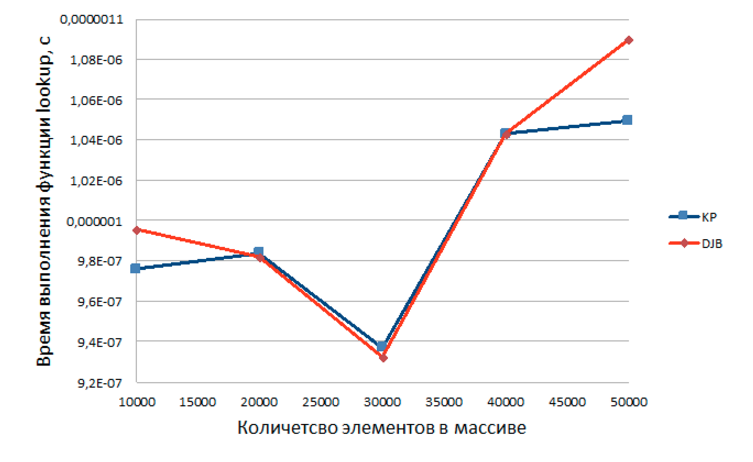
**Эксперимент 6. Анализ эффективности хеш-функций**

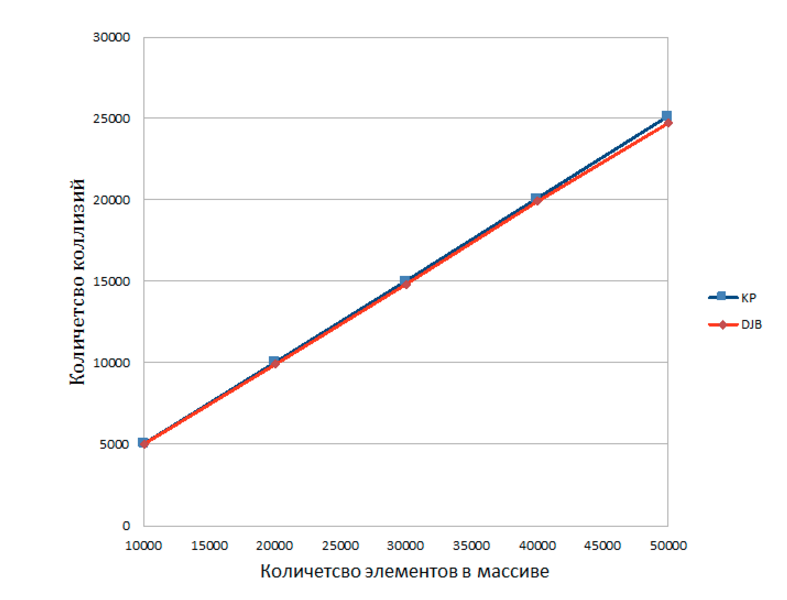
Требуется заполнить таблицу и построить:

▪ графики зависимости времени t выполнения операции поиска элемента в хеш-таблице от числа n элементов в ней для заданных хеш-функций KP и DJB.

▪ графики зависимости числа q коллизий от количества n элементов в хеш-таблице для заданных хеш-функций KP и DJB.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| # | Количество элементов в словаре | Хеш-функция KP | | Хеш-функция DJB | |
| Время выполнения функции hashtab\_lookup, с | Число коллизий | Время выполнения функции hashtab\_lookup, с | Число коллизий |
| 1 | 10000 | 0,000000976 | 5035 | 9,957E-07 | 5014 |
| 2 | 20000 | 9,841E-07 | 10037 | 9,821E-07 | 9904 |
| 3 | 30000 | 0,000000937 | 15005 | 9,325E-07 | 14787 |
| 4 | 40000 | 1,0432E-06 | 20086 | 1,0432E-06 | 19913 |
| 5 | 50000 | 1,0497E-06 | 25032 | 1,0899E-06 | 24814 |



Из результатов экспериментов можно сделать вывод, что бинарное дерево поиска работает медленнее, чем Хэш-таблицы. Это объясняется тем, что хэш-таблица имеет константную вычислительную сложность O(1), а у бинарного дерева O(log(n)).

Ссылки

1. <https://ru.wikipedia.org/wiki/Двоичное_дерево_поиска>

2. https://ru.wikipedia.org/wiki/Хеш-таблица