Отчет о выполнении задания 2 «Бинарные деревья поиска и хэш-таблицы»

Мингазеев Роман Лутфиевич

Группа ИВ-521

direnol@yandex.ru

Описание алгоритмов

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 T и ключ K.

Задача: проверить, есть ли узел с ключом K в дереве T, 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 T.
- Если K<X, рекурсивно искать ключ K в левом поддереве T.

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 T и пара (K, V).

Задача: вставить пару (K, V) в дерево T (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 T.
- Если K<X, циклически добавить (K, V) в левое поддерево T.
- 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).

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 $i = \text{hash}(key)$ 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: $i = \text{hash}(K)$.
- 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 связногосписка.

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

Вычислительная сложность в среднем случае $O(1)$.

Экспериментальное исследование

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 искомого ключа следует выбрать случайное слово, которое уже было добавлено в словарь.

Эксперимент 2 Сравнение эффективности добавления элементов в бинарное дерево поиска и хеш-таблицу

Требуется заполнить таблицу 2 и построить графики зависимости времени t выполнения операции добавления (add) элемента в бинарное дерево поиска и хеш-таблицу от числа n элементов уже вставленных в словарь.

Эксперимент 6 Анализ эффективности хеш-функций

Требуется заполнить таблицу 6 и построить:

- графики зависимости времени t выполнения операции поиска элемента в хеш-таблице от числа n элементов в ней для заданных хеш-функций X и Y (см. распределение вариантов)
- графики зависимости числа q коллизий от количества n элементов в хеш-таблице для заданных хеш-функций X и Y (см. распределение вариантов)

Организация экспериментов

- Эксперименты проводились на ПК DNS (CPU: QuadCore Intel Core i5-3550, 3500 MHz (35 x 100), RAM: 8GB, Hitachi HDS721010CLA330 ATA Device (931 ГБ, IDE))
- Операционная система Ubuntu 14.04 x64
- Ключи компиляции программы: -Wall -o

Результаты экспериментов

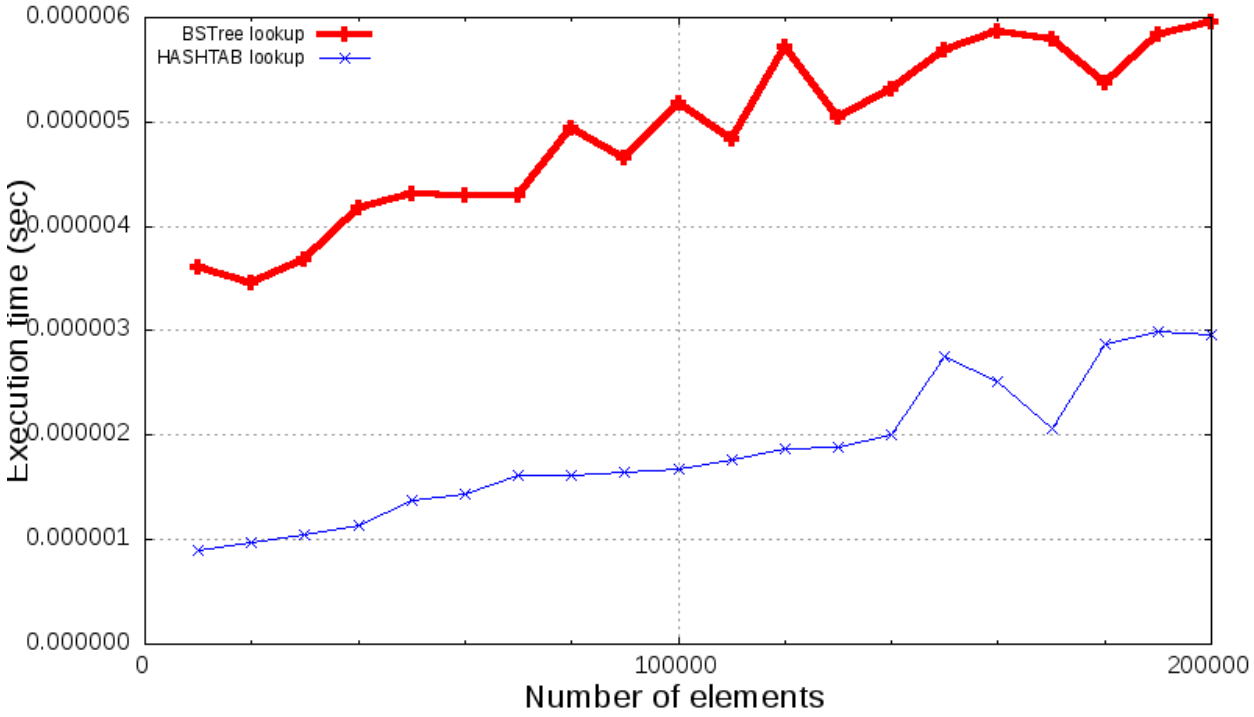
Эксперимент 1

Таблица 1

#	Количество элементов в словаре	Время выполнения функции bstree_lookup, с	Время выполнения функции hashtab_lookup, с
1	10 000	0.00000360	0.00000090
2	20 000	0.00000345	0.00000097
3	30 000	0.00000368	0.00000104
4	40 000	0.00000417	0.00000113
5	50 000	0.00000431	0.00000138
6	60 000	0.00000430	0.00000143
7	70 000	0.00000430	0.00000162
8	80 000	0.00000494	0.00000161
9	90 000	0.00000466	0.00000164
10	100 000	0.00000517	0.00000168
11	110 000	0.00000483	0.00000177
12	120 000	0.00000571	0.00000187
13	130 000	0.00000504	0.00000189
14	140 000	0.00000531	0.00000201
15	150 000	0.00000568	0.00000275
16	160 000	0.00000587	0.00000252
17	170 000	0.00000579	0.00000206
18	180 000	0.00000537	0.00000287

19	**190 000**	**0.00000583**	**0.00000299**
20	**200 000**	**0.00000595**	**0.00000306**

Рис. 1 bstree and hashtab lookup

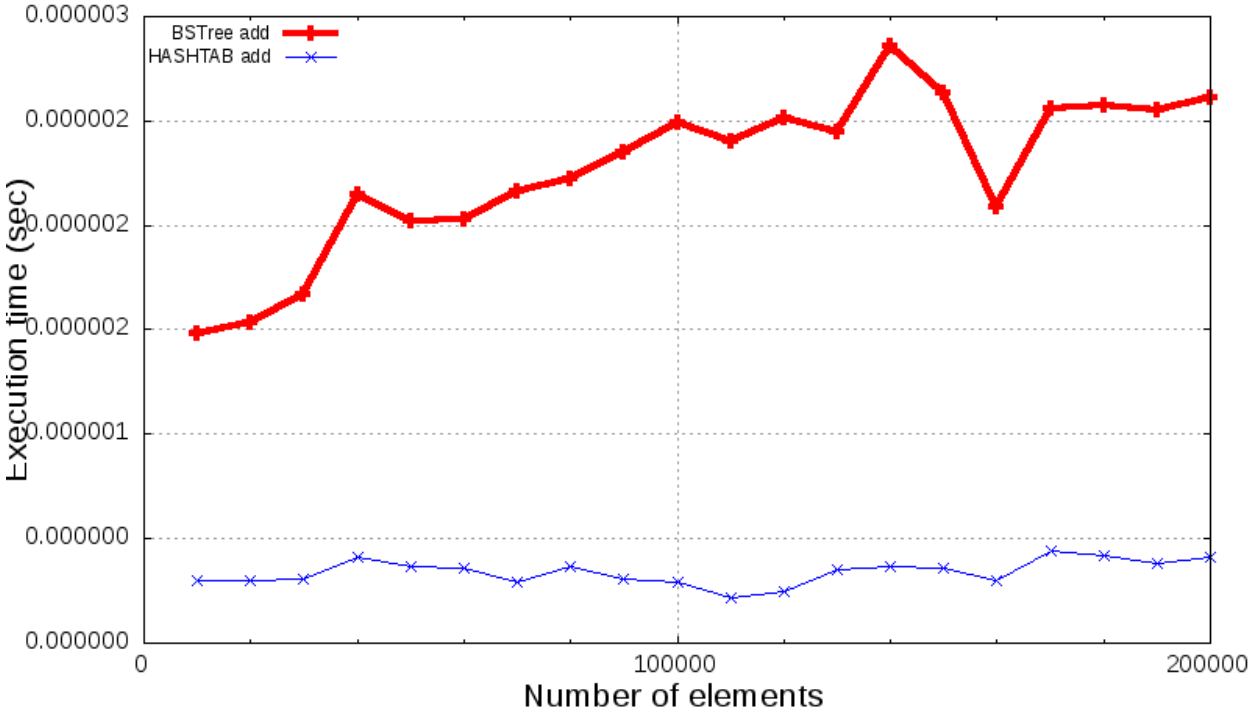


Эксперимент 2

Таблица 2

#	Количество элементов в словаре	Время выполнения функции bstree_add, с	Время выполнения функции hashtab_add, с
1	**10 000**	**0.00000148**	**0.00000030**
2	**20 000**	**0.00000153**	**0.00000030**
3	**30 000**	**0.00000167**	**0.00000031**
4	**40 000**	**0.00000215**	**0.00000041**
5	**50 000**	**0.00000202**	**0.00000037**
6	**60 000**	**0.00000203**	**0.00000036**
7	**70 000**	**0.00000216**	**0.00000029**
8	**80 000**	**0.00000222**	**0.00000037**
9	**90 000**	**0.00000235**	**0.00000031**
10	**100 000**	**0.00000249**	**0.00000029**
11	**110 000**	**0.00000240**	**0.00000022**
12	**120 000**	**0.00000251**	**0.00000025**
13	**130 000**	**0.00000245**	**0.00000035**
14	**140 000**	**0.00000286**	**0.00000037**
15	**150 000**	**0.00000263**	**0.00000036**
16	**160 000**	**0.00000209**	**0.00000030**
17	**170 000**	**0.00000256**	**0.00000044**
18	**180 000**	**0.00000257**	**0.00000042**
19	**190 000**	**0.00000255**	**0.00000038**
20	**200 000**	**0.00000261**	**0.00000041**

Рис. 2 bstree and hashtab add



Эксперимент 6

#	Количество элементов в словаре	Хеш-функция KPHash		Хеш-функция JENKINS	
		Время выполнения функции hashtab_lookup, с	Число коллизий	Время выполнения функции hashtab_lookup, с	Число коллизий
1	10 000	0.00000090	3018	0.00000160	3040
2	20 000	0.00000097	9757	0.00000170	9721
3	30 000	0.00000104	18231	0.00000170	18282
4	40 000	0.00000113	27558	0.00000179	27575
5	50 000	0.00000138	37235	0.00000222	37228
6	60 000	0.00000143	47104	0.00000216	47089
7	70 000	0.00000162	57036	0.00000220	57023
8	80 000	0.00000161	67004	0.00000228	66995
9	90 000	0.00000164	76988	0.00000243	76989
10	100 000	0.00000168	86981	0.00000242	86981
11	110 000	0.00000177	96976	0.00000253	96976
12	120 000	0.00000187	106974	0.00000276	106975
13	130 000	0.00000189	116974	0.00000296	116973
14	140 000	0.00000201	126974	0.00000294	126973
15	150 000	0.00000275	136973	0.00000327	136973
16	160 000	0.00000252	146973	0.00000353	146973
17	170 000	0.00000206	156973	0.00000255	156973

18	180 000	0.00000287	166973	0.00000365	166973
19	190 000	0.00000299	176973	0.00000360	176973
20	200 000	0.00000306	186973	0.00000368	186973

Рис. 3 KPHash and ELFHash lookup

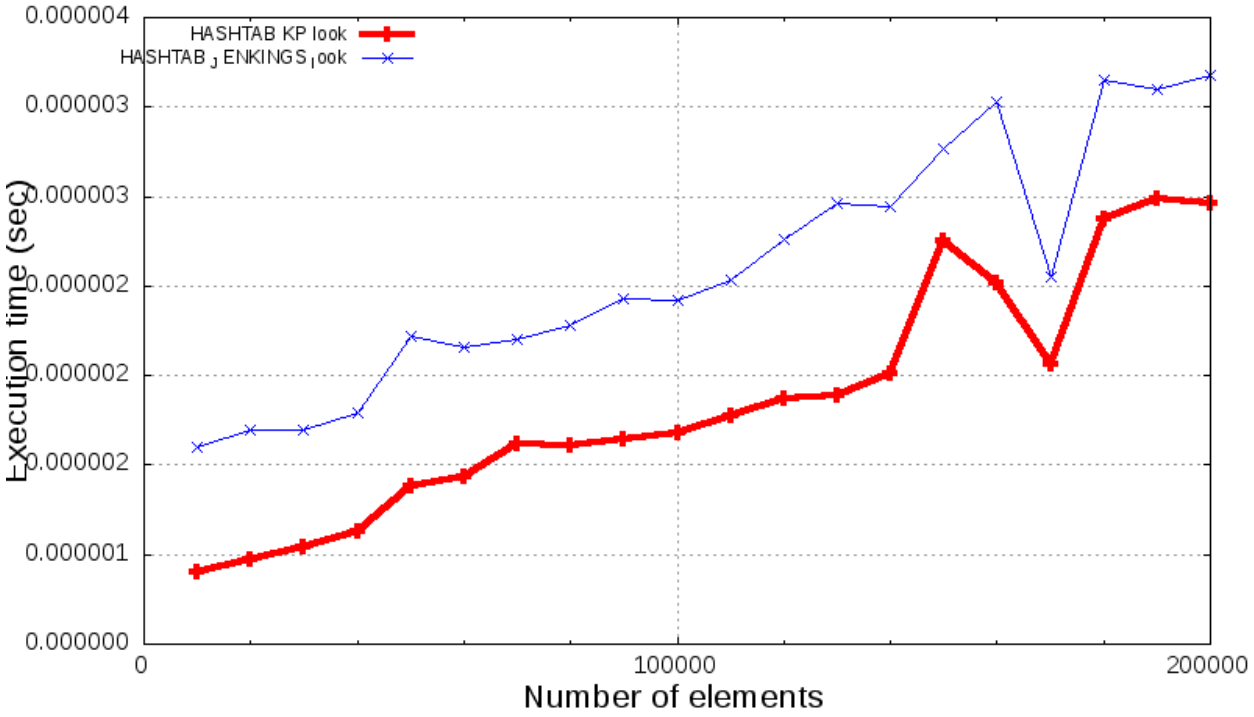
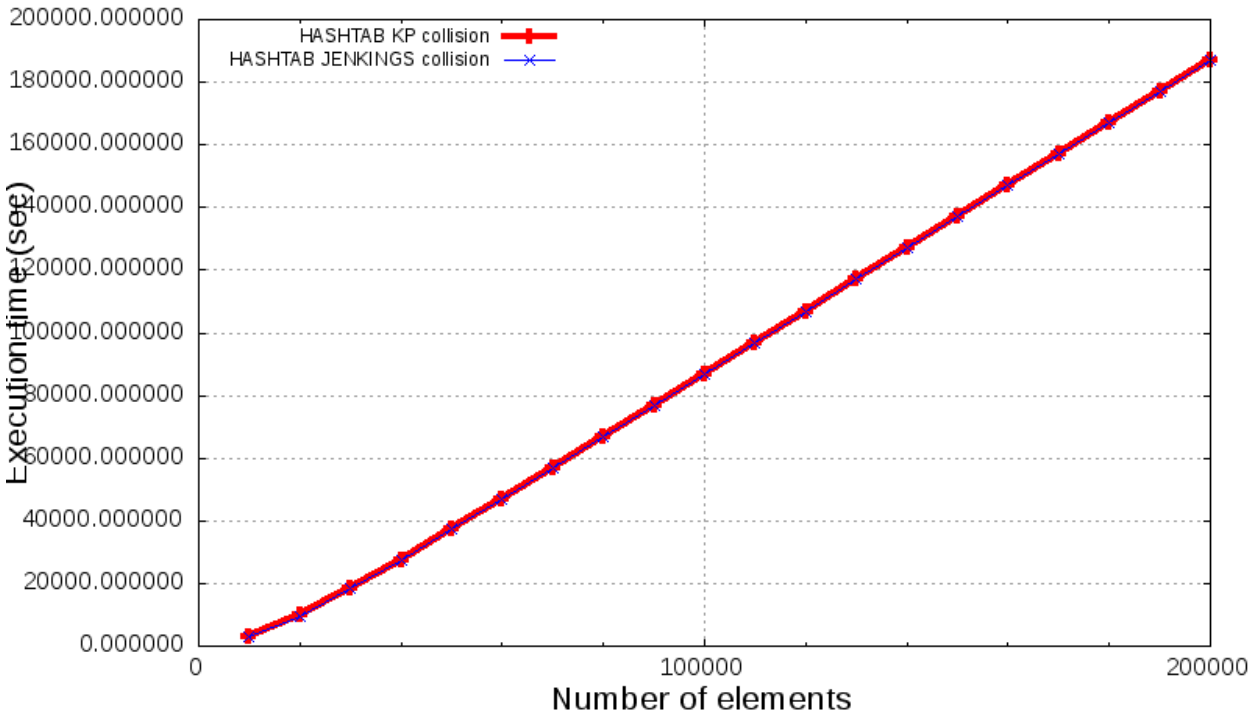


Рис. 4 KPHash and ELFHash collision



Вывод

Из результатов экспериментов можено сделать следущие выводы:

1) Сложность добавления элементов в BST $\Theta(\log n)$ – средний случай, и $O(n)$ в худшей. Поиск имеет аналогичную сложность с добавлением. Если использовать данную реализацию словаря, то необходимо позабоиться о том, что BST будет более менее сбалансированным, или чтобы добавление элементов было не по возрастанию(убыванию).

2) Сложность добавления элементов в hashtab $O(1)$ – так как доавбление происходит в начало списка. Поиск же происходит со сложностью $O((n + m)$, где n – кол-во элементов, а m – размер таблицы. Если необходимо только хранит и добавлять элементы, то для увелечения эффективности hashtab – нужно позаботиться о наименьшем кол-во коллизий, то есть использовать “хорошую” хэш – функцию.

3) Хеш функция JENKINS работает хуже чем KP функция.

Ссылки

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