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

**Тарасенко Григорий Германович**

Группа ИВ-521

fle4er@yandex.ru

# Описание алгоритмов

В задании 2 требовалось реализовать и исследовать эффективность бинарных деревьев поиска и хеш-таблиц. Все алгоритмы реализованы на языке C под операционной системой GNU/Linux.

# Бинарное дерево поиска[1]

Бинарное дерево поиска ([англ.](https://ru.wikipedia.org/wiki/%D0%90%D0%BD%D0%B3%D0%BB%D0%B8%D0%B9%D1%81%D0%BA%D0%B8%D0%B9_%D1%8F%D0%B7%D1%8B%D0%BA" \o "Английский язык) binary search tree, BST) — это [двоичное дерево](https://ru.wikipedia.org/wiki/%D0%94%D0%B2%D0%BE%D0%B8%D1%87%D0%BD%D0%BE%D0%B5_%D0%B4%D0%B5%D1%80%D0%B5%D0%B2%D0%BE" \o "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.

В работе реализована версия алгоритма с вычислительной сложностью следующих операций:

* Создание элемента дерева (bstree\_create): *O*(1);
* Добавление элемента (bstree\_add): *O*(*n*);
* Поиск элемента (bstree\_lookup): *O*(*n*);
* Поиск минимального значения (bstree\_min): *O*(*n*);
* Поиск максимального значения (bstree\_max): *O*(*n*);

Расход по памяти *O*(*n*), где *n* — это количество внесенных элементов в дерево.

# Хеш-таблицы[2]

Хеш-таблица — это [структура данных](https://ru.wikipedia.org/wiki/%D0%A1%D1%82%D1%80%D1%83%D0%BA%D1%82%D1%83%D1%80%D0%B0_%D0%B4%D0%B0%D0%BD%D0%BD%D1%8B%D1%85" \o "Структура данных), реализующая интерфейс [ассоциативного массива](https://ru.wikipedia.org/wiki/%D0%90%D1%81%D1%81%D0%BE%D1%86%D0%B8%D0%B0%D1%82%D0%B8%D0%B2%D0%BD%D1%8B%D0%B9_%D0%BC%D0%B0%D1%81%D1%81%D0%B8%D0%B2" \o "Ассоциативный массив), а именно, она позволяет хранить пары (ключ, значение) и выполнять три операции: операцию добавления новой пары, операцию поиска и операцию удаления пары по ключу.

В работе реализована версия алгоритма с вычислительной сложностью следующих операций:

* Создание хеша (hashtab\_hash): *O*(s);
* Добавление элемента (hashtab\_add): *O*(1);
* Поиск элемента (bstree\_lookup): *O*(*n*);
* Удаление элемента: *O*(*n*);

Расход по памяти *O*(*n*), где *n* — это количество внесенных элементов в таблицу.

# Организация экспериментов

* Эксперименты проводились на настольном компьютере  
  (CPU: AMD A8-7600, RAM: 8GB, HDD Western Digital Blue 1 TB)
* Операционная система ArchLinux x86\_64 (компилятор gcc 5.3.0)
* Ключи компиляции программы (см. README): -o

# Результаты экспериментов

Результаты экспериментов приведены в таблицах 1, 2 и 3 и в графиках 1, 2, 3.

*Таблица 1. Результаты поиска элемента и бинарном дереве и в хеш-таблице*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Количество элементов** | **Бинарное дерево** | **Хеш-таблица** |
| 1 | 10000 | 0,00009635 | 0,00011385 |
| 2 | 20000 | 0,00018115 | 0,00021498 |
| 3 | 30000 | 0,00028477 | 0,00033914 |
| 4 | 40000 | 0,00036332 | 0,00044467 |
| 5 | 50000 | 0,00047202 | 0,00058207 |
| 6 | 60000 | 0,00050642 | 0,00071978 |
| 7 | 70000 | 0,00064009 | 0,00085785 |
| 8 | 80000 | 0,00071453 | 0,00089446 |
| 9 | 90000 | 0,00071652 | 0,00105858 |
| 10 | 100000 | 0,00081167 | 0,00102402 |
| 11 | 110000 | 0,00100876 | 0,00108384 |
| 12 | 120000 | 0,00107521 | 0,00138215 |
| 13 | 130000 | 0,00114293 | 0,00157982 |
| 14 | 140000 | 0,00112576 | 0,00157089 |
| 15 | 150000 | 0,00129492 | 0,00166828 |
| 16 | 160000 | 0,0013754 | 0,00169534 |
| 17 | 170000 | 0,00124693 | 0,00184687 |
| 18 | 180000 | 0,00166473 | 0,00196512 |
| 19 | 190000 | 0,00149487 | 0,00230543 |
| 20 | 200000 | 0,00152363 | 0,00217288 |

В данной таблице мы можем увидеть, что поиск в бинарном дереве занимает меньше времени, относительно поиск хеш-таблицах.

*График 1. Результаты поиска элемента и бинарном дереве и в хеш-таблице*

*Таблица 2. Время поиска в бинарном дереве в худшем и в среднем случаях*

|  |  |  |  |
| --- | --- | --- | --- |
| **№** | **Количество элементов** | **Худший случай** | **Средний случай** |
| 1 | 10000 | 0,00009 | 0,000078 |
| 2 | 20000 | 0,000274 | 0,000228 |
| 3 | 30000 | 0,000504 | 0,00044 |
| 4 | 40000 | 0,000721 | 0,00065 |
| 5 | 50000 | 0,000878 | 0,000723 |
| 6 | 60000 | 0,000938 | 0,000807 |
| 7 | 70000 | 0,001106 | 0,000913 |
| 8 | 80000 | 0,001302 | 0,001095 |
| 9 | 90000 | 0,001447 | 0,001181 |
| 10 | 100000 | 0,001632 | 0,001364 |
| 11 | 110000 | 0,001679 | 0,00147 |
| 12 | 120000 | 0,001822 | 0,001499 |
| 13 | 130000 | 0,002103 | 0,00168 |
| 14 | 140000 | 0,002178 | 0,001827 |
| 15 | 150000 | 0,002441 | 0,002022 |
| 16 | 160000 | 0,002494 | 0,002187 |
| 17 | 170000 | 0,002763 | 0,002336 |
| 18 | 180000 | 0,002762 | 0,00238 |
| 19 | 190000 | 0,002902 | 0,002503 |
| 20 | 200000 | 0,0031 | 0,002563 |

Из таблицы 2 видно, что худший случай есть худший. Это связано с тем, что в худшем случае мы получаем несбалансированное дерево, нежели в среднем случае.

*График 2. Время поиска в бинарном дереве в худшем и в среднем случаях*

*Таблица 3. Время поиска в хеш-таблицах в алгоритмах KP и XOR*

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **№** | **Количество элементов** | **Хеш-функция KP** | | **Хеш-функция XOR** | |
| Время поиска | Коллизии | Время поиска | Коллизии |
| 1 | 10000 | 0,000002 | 1001 | 0,000004 | 124 |
| 2 | 20000 | 0,000002 | 1001 | 0,00001 | 128 |
| 3 | 30000 | 0,000004 | 1001 | 0,000027 | 128 |
| 4 | 40000 | 0,000003 | 1001 | 0,000028 | 128 |
| 5 | 50000 | 0,000004 | 1001 | 0,000033 | 128 |
| 6 | 60000 | 0,000008 | 1001 | 0,000055 | 128 |
| 7 | 70000 | 0,000008 | 1001 | 0,000058 | 128 |
| 8 | 80000 | 0,000009 | 1001 | 0,00007 | 128 |
| 9 | 90000 | 0,000011 | 1001 | 0,000092 | 128 |
| 10 | 100000 | 0,000011 | 1001 | 0,000083 | 128 |
| 11 | 110000 | 0,000009 | 1001 | 0,000085 | 128 |
| 12 | 120000 | 0,000009 | 1001 | 0,000106 | 128 |
| 13 | 130000 | 0,000012 | 1001 | 0,000107 | 128 |
| 14 | 140000 | 0,000009 | 1001 | 0,000115 | 128 |
| 15 | 150000 | 0,000009 | 1001 | 0,000122 | 128 |
| 16 | 160000 | 0,000013 | 1001 | 0,000059 | 128 |
| 17 | 170000 | 0,000014 | 1001 | 0,000125 | 128 |
| 18 | 180000 | 0,000012 | 1001 | 0,000141 | 128 |
| 19 | 190000 | 0,000014 | 1001 | 0,000165 | 128 |
| 20 | 200000 | 0,00002 | 1001 | 0,000168 | 128 |

По таблице видно, что в алгоритме хеш-функции XOR коллизий хоть и меньше, но поиск в данной хеш-таблице заметно медленнее, чем в хеш-таблице, построенной по алгоритму KP.

*График 3. Время поиска в хеш-таблицах в алгоритмах KP и XOR*

# Ссылки

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