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 ноутбуке Samsung NP300V5A  
  (CPU: Intel 2310m, RAM: 3GB, HDD Western Digital Blue 320 GB)
* 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 | 0,000058 | 0.0000011920928955078125 |
| 2 | 20000 | 0,000102 | 0.00000095367431640625 |
| 3 | 30000 | 0,000038 | 0.0000011920928955078125 |
| 4 | 40000 | 0,0002 | 0.0000011920928955078125 |
| 5 | 50000 | 0,000288 | 0.00000095367431640625 |
| 6 | 60000 | 0,000337 | 0.00000095367431640625 |
| 7 | 70000 | 0,000297 | 0.00000095367431640625 |
| 8 | 80000 | 0,000229 | 0.00000095367431640625 |
| 9 | 90000 | 0,000266 | 0.00000000000000000000 |
| 10 | 100000 | 0,000636 | 0.00000000000000000000 |
| 11 | 110000 | 0,000418 | 0.00000000000000000000 |
| 12 | 120000 | 0,000192 | 0.0000011920928955078125 |
| 13 | 130000 | 0,000777 | 0.00000095367431640625 |
| 14 | 140000 | 0,000258 | 0.00000095367431640625 |
| 15 | 150000 | 0,000782 | 0.00000095367431640625 |
| 16 | 160000 | 0,001027 | 0.00000095367431640625 |
| 17 | 170000 | 0,001262 | 0.00000000000000000000 |
| 18 | 180000 | 0,000697 | 0.00000095367431640625 |
| 19 | 190000 | 0,001285 | 0.0000011920928955078125 |
| 20 | 200000 | 0,001185 | 0.00000095367431640625 |

В данной таблице мы можем увидеть, что хеш-таблицы абсолютный победитель по быстрому поиску элемента.

*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 | 0.000082 | 0.000088 |
| 2 | 20000 | 0.000173 | 0.000184 |
| 3 | 30000 | 0.000335 | 0.000343 |
| 4 | 40000 | 0.000509 | 0.000506 |
| 5 | 50000 | 0.000651 | 0.000649 |
| 6 | 60000 | 0.000783 | 0.000797 |
| 7 | 70000 | 0.000932 | 0.000896 |
| 8 | 80000 | 0.001024 | 0.001045 |
| 9 | 90000 | 0.001146 | 0.001141 |
| 10 | 100000 | 0.001267 | 0.001302 |
| 11 | 110000 | 0.001395 | 0.001423 |
| 12 | 120000 | 0.001512 | 0.001528 |
| 13 | 130000 | 0.001651 | 0.001706 |
| 14 | 140000 | 0.001816 | 0.001788 |
| 15 | 150000 | 0.001954 | 0.001903 |
| 16 | 160000 | 0.002050 | 0.002035 |
| 17 | 170000 | 0.002164 | 0.002141 |
| 18 | 180000 | 0.002355 | 0.002320 |
| 19 | 190000 | 0.002411 | 0.002436 |
| 20 | 200000 | 0.002575 | 0.002620 |

Из таблицы 2 видно, что худший случай не есть худший.

*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 | 0.0000000000000 | 11 | 0.0000040531158447265625 | 124 |
| 2 | 20000 | 0.0000000000000 | 85 | 0.000011920928955078125 | 128 |
| 3 | 30000 | 0.00000095367431640625 | 295 | 0.0000150203704833984375 | 128 |
| 4 | 40000 | 0.0000011920928955078125 | 648 | 0.000030994415283203125 | 128 |
| 5 | 50000 | 0.000000000000000000000 | 1303 | 0.0000460147857666015625 | 128 |
| 6 | 60000 | 0.0000000000000000000000000 | 2235 | 0.0000360012054443359375 | 128 |
| 7 | 70000 | 0.0000009536743164062500000 | 3460 | 0.0000419616699218750000000 | 128 |
| 8 | 80000 | 0.0000009536743164062500000 | 5000 | 0.0000460147857666015625000 | 128 |
| 9 | 90000 | 0.0000009536743164062500000 | 6819 | 0.0000569820404052734375000 | 128 |
| 10 | 100000 | 0.0000000000000000000000000 | 8972 | 0.0000691413879394531250000 | 128 |
| 11 | 110000 | 0.0000000000000000000000000 | 11516 | 0.0000691413879394531250000 | 128 |
| 12 | 120000 | 0.0000009536743164062500000 | 14439 | 0.0000801086425781250000000 | 128 |
| 13 | 130000 | 0.0000009536743164062500000 | 17830 | 0.0000829696655273437500000 | 128 |
| 14 | 140000 | 0.0000009536743164062500000 | 21645 | 0.0000967979431152343750000 | 128 |
| 15 | 150000 | 0.0000009536743164062500000 | 25855 | 0.0001299381256103515625000 | 128 |
| 16 | 160000 | 0.0000009536743164062500000 | 30421 | 0.0001361370086669921875000 | 128 |
| 17 | 170000 | 0.0000000000000000000000000 | 35520 | 0.0001609325408935546875000 | 128 |
| 18 | 180000 | 0.0000009536743164062500000 | 40896 | 0.0001630783081054687500000 | 128 |
| 19 | 190000 | 0.0000009536743164062500000 | 46702 | 0.0001690387725830078125000 | 128 |
| 20 | 200000 | 0.0000011920928955078125000 | 52830 | 0.0001749992370605468750000 | 128 |

По таблице видно, что в алгоритме хеш-функции XOR коллизий хоть и меньше, но поиск в данной хеш-таблице заметно меньше, 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>