Скулкина Н

**Отчёт по домашней работе №3**

Оглавление

[**Сравнительный анализ производительности MyLinkedList и java.util.LinkedList** 2](#_Toc71283859)

[**1.** **Добавление элемента** 2](#_Toc71283860)

[**2.** **Поиск элемента** 3](#_Toc71283861)

[**3.** **Удаление элемента** 4](#_Toc71283862)

[**4.** **Выводы** 4](#_Toc71283863)

[**Сравнительный анализ производительности ArrayList и LinkedList** 5](#_Toc71283864)

[**1.** **Добавление элемента** 5](#_Toc71283865)

[**2.** **Вставка элемента** 5](#_Toc71283866)

[**3.** **Удаление элемента** 6](#_Toc71283867)

[**4.** **Поиск элемента** 6](#_Toc71283868)

[**5.** **Выводы** 7](#_Toc71283869)

[**Сравнительный анализ производительности HashSet, LinkedHashSet, TreeSet** 8](#_Toc71283870)

[**1.** **Добавление элемента** 8](#_Toc71283871)

[**2.** **Удаление элемента** 8](#_Toc71283872)

[**3.** **Выводы** 9](#_Toc71283873)

[**Сравнительный анализ производительности HashMap, LinkedHashMap, TreeMap** 10](#_Toc71283874)

[**1.** **Добавление элемента** 10](#_Toc71283875)

[**2.** **Удаление элемента** 10](#_Toc71283876)

[**3.** **Выводы** 11](#_Toc71283877)

# **Сравнительный анализ производительности MyLinkedList и java.util.LinkedList**

## **Добавление элемента**

Создадим объекты MyLinkedList и LinkedList и добавим в них 10000 элементов. На рис 1 показана зависимость времени добавления (по вертикали) от добавляемого элемента (по горизонтали). Здесь красным показан MyLinkedList, синим - LinkedList.

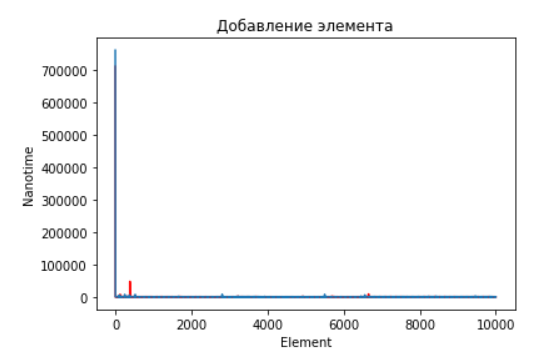


Рисунок 1

Больше всего времени занимает добавление первого элемента в пустой список для обоих списков (там на графике под высокой синей полосочкой можно заметить красную). Среднее значение времени добавления для списков приблизительно одинаковое (рис 2) и немного варьируется от запуска к запуску.

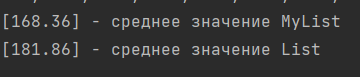


Рисунок 2

Если исключить добавление первого элемента в пустой список, значения времени становятся более равномерными (рис 3), а среднее значение времени добавления уменьшается (рис 4).

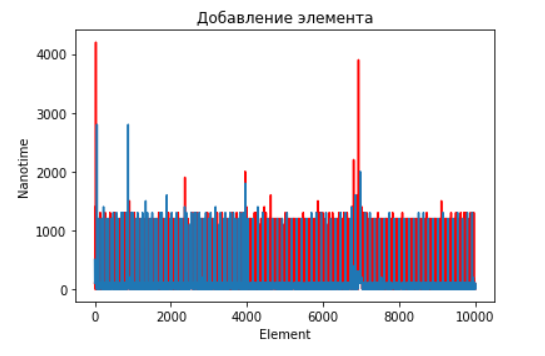


Рисунок 3

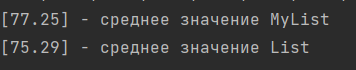


Рисунок 4

## **Поиск элемента**

Продолжаем работу с MyLinkedList и LinkedList. В списках лежат по 10000 элементов типа Circle с радиусами от 0 до 10000 (у i-ого элемента i-ый радиус). Затем 1000 раз генерируем Circle с рандомным радиусом (от 0 до 10000) и ищем его в обоих листах. На рис 5 показана зависимость времени поиска (по вертикали) от радиуса добавляемого элемента (и его расположения в списке соответственно) (по горизонтали). Здесь синим показан MyLinkedList, оранжевым – LinkedList. Среднее время поиска в обоих листах приблизительно одинаковое (рис 6).

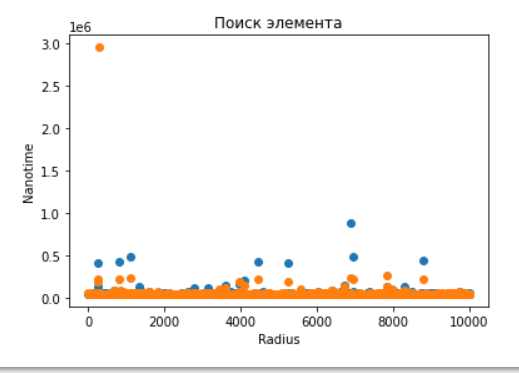


Рисунок 5

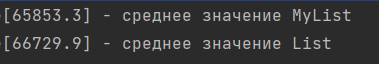


Рисунок 6

## **Удаление элемента**

Продолжаем работу с MyLinkedList и LinkedList. Далее 1000 раз сгенерируем рандомный номер элемента списка и удалим его из обоих списков. На рис 7 показана зависимость времени удаления (по вертикали) от номера удаляемого элемента (по горизонтали). Здесь синим показан MyLinkedList, оранжевым- LinkedList. Среднее время удаления в обоих листах приблизительно одинаковое (рис 8).

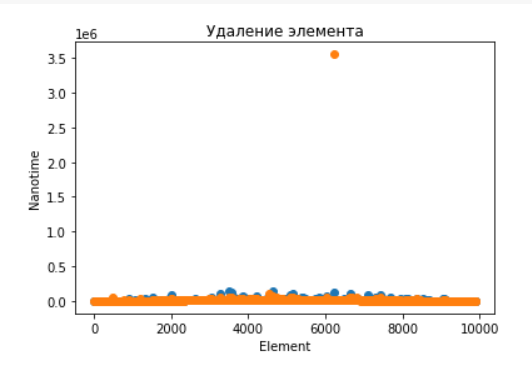


Рисунок 7



Рисунок 8

## **Выводы**

Производительность MyLinkedList и LinkedList приблизительно одинаковая.

# **Сравнительный анализ производительности ArrayList и LinkedList**

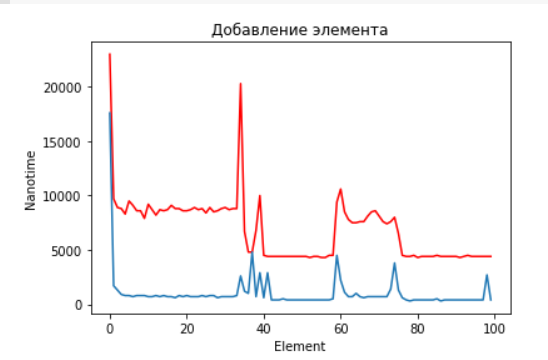
## **Добавление элемента**

Красным обозначен ArrayList, синим LinkedList. При добавлении элементов в конец списка (рис 9-12) время добавления у обоих листов приблизительно одинаковое.

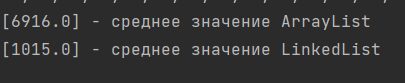
|  |  |
| --- | --- |
| Рисунок 9    Рисунок 10 | Рисунок 11    Рисунок 12 |

## **Вставка элемента**

При добавлении элементов в начало списка (рис 13-14) время добавления в ArrayList больше, чем время добавления в LinkedList. При добавлении элементов в середину списков ситуация аналогичная. Это обусловлено тем, что в ArrayList элементы, находящиеся после вставляемого, необходимо сдвигать, тогда как в LinkedList нужно просто поменять ссылки.



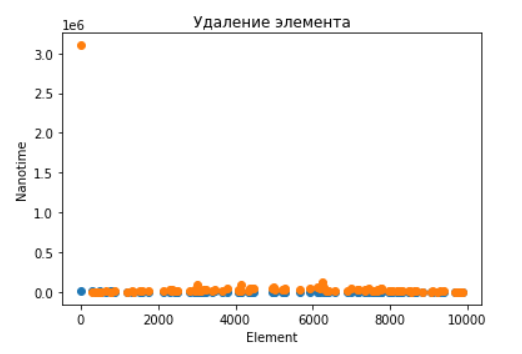
*Рисунок 13*



*Рисунок 14*

## **Удаление элемента**

Здесь синим показан ArrayList, оранжевым – LinkedList. При удалении рандомных элементов списка время удаления из ArrayList меньше, чем время удаления из LinkedList (рис 15-16). В ArrayList трудоёмкой операцией является удаление элемента из начала списка, но в среднем ArrayList выигрывает.



*Рисунок 15*



*Рисунок 16*

## **Поиск элемента**

Здесь на графике (рис 17) изображена зависимость времени поиска элемента от номера этого элемента в списке. Синим обозначен LinkedList, оранжевым ArrayList. Среднее время поиска у ArrayList значительно меньше (рис 18).

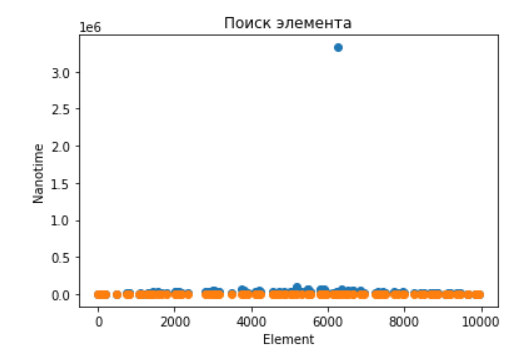


Рисунок 17



Рисунок 18

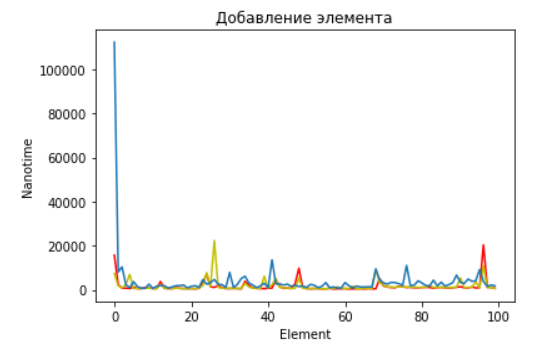
## **Выводы**

LinkedList лучше использовать, когда необходимо работать (добавлять/удалять) с данными из начала списка. ArrayList лучше использовать, когда часто пользуемся поиском элементов (по логике и изменением тоже).

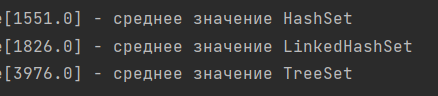
# **Сравнительный анализ производительности HashSet, LinkedHashSet, TreeSet**

## **Добавление элемента**

Здесь красным показан HashSet, жёлтым LinkedHashSet, синим TreeSet. Логично, что быстрее всего происходит операция добавления в HashSet, т.к. он просто добавляет элемент, тогда как в LinkedHashSet необходимо поддерживать порядок элементов, а в TreeSet при добавлении элемента нужно выполнять сортировку.



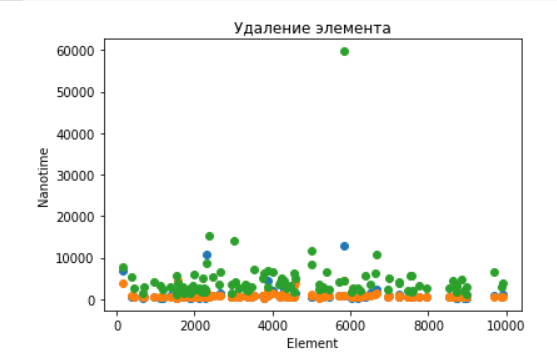
*Рисунок 19*



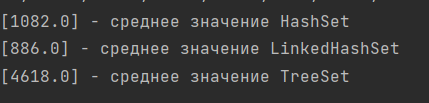
*Рисунок 20*

## **Удаление элемента**

Здесь оранжевым показан HashSet, синим LinkedHashSet, зелёным TreeSet. Аналогично добавлению удаление дольше всего работает в TreeSet. В HashSet и LinkedHashSet приблизительно одинаково.



*Рисунок 21*



*Рисунок 22*

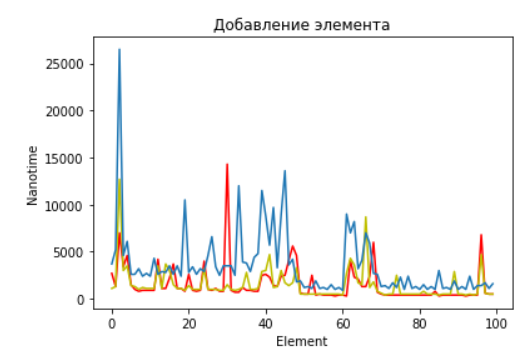
## **Выводы**

HashSet и LinkedHashSet работают приблизительно одинаково, тогда как TreeSet работает сильно медленнее. HashSet хранит данные в случайном порядке, LinkedHashSet сохраняет порядок записи, а TreeSet сохраняет элементы в порядке возрастания ключей. Если нам не нужно хранить данные в отсортированном виде, лучше использовать HashSet или LinkedHashSet. Если нужно, лучше TreeSet, у него есть много полезных методов.

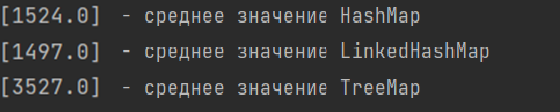
# **Сравнительный анализ производительности HashMap, LinkedHashMap, TreeMap**

## **Добавление элемента**

Опять добавление первого элемента занимает больше всего времени для всех коллекций. При добавлении элементов в уже заполненную таблицу, результат сравнения показан на рис 23. Здесь красным показан HashMap, жёлтым LinkedHashMap, синим TreeMap. Из результатов, показаных на рис 24, можно сделать выводы, что добавление элемента в HashMap и LinkedHashMap занимает приблизительно одинаковое количество времени, а для добавления элемента в TreeMap требуется больше времени, так как там происходит сортировка элементов.



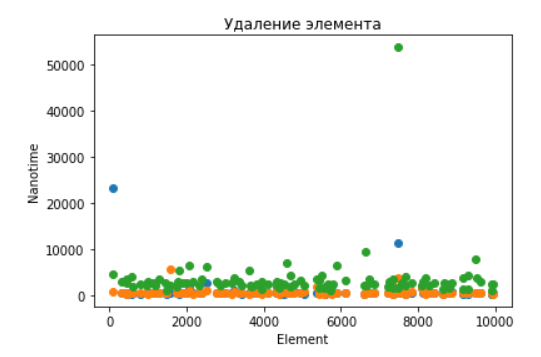
*Рисунок 23*



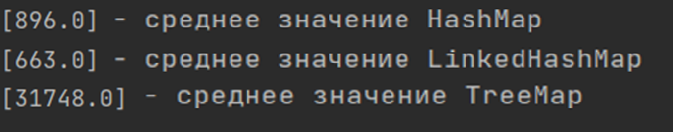
*Рисунок 24*

## **Удаление элемента**

Здесь оранжевым показан HashMap, синим LinkedHashMap, зелёным TreeMap. Аналогично добавлению удаление дольше всего работает в TreeMap. В HashMap и LinkedHashMap приблизительно одинаково.



*Рисунок 25*

**

*Рисунок 26*

## **Выводы**

Тут всё аналогично предыдущему пункту, т.к. HashSet реализуется с помощью HashMap LinkedHashSet реализуется с помощью LinkedHashMap, а TreeSet реализуется с помощью TreeMap.