Добавление элемента в список

График 1. Добавление элемента в 1М список

Тесты на добавление одного элемента в начало и конец списка, состоящего из 1М элементов, примерно одинаковые, в отличие от добавления во вторую половину списка. Связано это с тем, что в java.util.LinkedList классе, чтобы достичь некоторой позиции через цикл, рассматриваются две половины списка. В первой половине движение идет от начала до индекса, во второй половине - от конца до индекса.

Проверка if (index < size >> 1) смотрит в какой половине лежит индекс. После изменения MyLinkedList по этому алгоритму, результаты стали примерно одинаковыми.

График 2. Добавление элементов в пустой список

Таблица скучных цифр

Добавление 10к элементов в 1М список (в мс)

|  |  |  |
| --- | --- | --- |
|  | MyLinkedList | LinkedList |
| Begin | 66,457 | 64,3589 |
| Middle | 269453,845 | 267048,894 |
| End | 551,3686 | 573,2823 |

Удаление элемента

График 3. Удаление элемента из 1М списка

Из начала списка: MLL – 0,00241мс, LL – 0,00213мс

Из конца списка: MLL – 0,00266мс, LL – 0,00213мс

С позиции 500000: MLL – 27,1655мс, LL – 26,0306мс

Получение элемента

График 4. Получение элемента из 1М списка

Из начала списка: MLL - 0,00158мс, LL – 0,00217мс

Из конца списка: MLL – 0,00156мс, LL – 0,00226мс

С позиции 500000: MLL – 1,79306мс, LL – 1,66539мс

Вывод некоторый.

Очень нерационально сравнивать время выполнения команд с помощью System.nanoTime(), при каждом новом пробеге получался разный результат. В целом, при выполнении одной команды время приблизительно равное, но в при выполнение большого количества команд время будет расти, что логично. Так при добавлении 1 элемента в середину списка время 27,073 мс и 26,381 мс для MLL и LL соответственно. Кажется незаметной такая разница.

Но если добавить 1М элементов в середину списка, то разница будет 692 c или 11 минут 32 секунды, что в каких-то ситуациях очень значительная и неприемлимая разница.