Skip List

Сегодня расскажу о структуре данных Skip List. Хочу сделать упор не на деталях реализации, а на том, зачем он нужен. И почему в java Skip List есть только в многопоточном варианте.

Кто вообще использует связные списки?

Давным-давно мы сравнивали (https://t.me/java\_fillthegaps/390) ArrayList (список на основе массива) и LinkedList (связный список).

В большинстве энтерпрайзных задач LinkedList проиграл. Вставка в середину редко встречается в бизнес-логике, в основном мы работаем со списком как с хранилищем данных. Но

✅ В инфраструктурных задачах вставка в середину более актуальна. SortedSet в Redis — сортированный список уникальных элементов. Элементы часто добавляются и удаляются из произвольных мест, логично взять за основу именно LinkedList. В очереди с приоритетами и secondary indexes тоже пригодится связный список.

✅ Многие структуры используют связный список косвенно: элементы ссылаются друг на друга для упрощения обхода или дополнительной сортировки.

Взять ту же LinkedHashMap из JDK — хэшмэп, в котором элементы связаны между собой в порядке добавления.

Самая проблемная операция LinkedList — поиск элемента. Даже в сортированном списке приходится ходить по ссылкам последовательно. Это долго.

Как устроен Skip List

Чтобы искать элементы быстрее, добавляем для исходного списка несколько уровней со ссылками.

Пример — на картинке под постом. Чтобы найти пятёрку, движемся от верхнего уровня к нижнему.

В реальности на верхних уровнях промежутки больше, и мы быстрее приходим к нужному месту. В идеальном случае за O(log N).

❓ Это разве не дерево теперь?

Действительно, алгоритм поиска очень похож. Но чем Skip List отличается от дерева:

🔸 Балансировка

В деревьях очень строгие правила, при добавлении-удалении дерево часто перестраивается. Зато мы получаем высокую и стабильную скорость поиска.

В Skip List более расслабленный подход. Новый элемент добавляется в основной список, а вопрос с наличием ссылки на верхних уровнях решается через рандом.

В целом структура получается нормально сбалансированной. Одни элементы будут находиться быстрее, другие медленнее, в среднем время поиска стремится к O(log N).

🔸 Расположение элементов

В Skip List все элементы хранятся на нижнем уровне. Добавление-удаление происходит очень легко — надо переписать всего пару ссылок.

В дереве элементы находятся в узлах со строгой структурой. Вставка и удаление часто приводят к ребалансировке.

Резюме

✍️ Связные списки редко используются в энтерпрайзе, но часто в инфраструктуре (кэши, БД). Основной сценарий здесь — поиск. И сам по себе, и для работы с текущими значениями.

✍️ Дерево не всегда подойдёт, тк часто балансируется. В нагруженных системах лишняя суета ни к чему😑

✍️ Чтобы ускорить поиск в связном списке, добавляем дополнительные уровни. Получаем Skip List!

Сортированные структуры в JDK

В однопоточной среде для хранения сортированных элементов чаще используют TreeMap/Set. Это красно-чёрное дерево. Балансируется при каждом изменении, поиск работает быстро и стабильно.

Поддерживать дерево в многопоточной среде сложно как раз из-за балансировки. Чтобы дерево безопасно сбалансировалось, лучше заблокировать его целиком, что снижает общую пропускную способность.

В многопоточной среде для той же задачи используется ConcurrentSkipListMap/Set. Изменения очень локальные: меняется несколько ссылок, а большая часть списка остаётся неизменной. Поэтому одновременно со структурой могут работать больше потоков.

Ответ на вопрос перед постом

ConcurrentSkipListSet — сортированный список без повторов. Skip List — название базовой структуры данных, Set — признак уникальности элементов.

