Deutsche Bank
Group Technology & Operations

Коллекции в Java
Изобретаем заново!

Алексей Рагозин

Passion to Perform

Чем плохи java.util коллекции?

Что можно улучшить в стандартных коллекциях?

- Удобство и функциональность
  - ✓ multi map /read-though / invertible map и т.п.
- Эффективность
  - ✓ Производительность
  - ✓ Ёмкость (в т.ч. работа с примитивными типами)
- Многопоточность
  - ✓ Lock free
  - ✓ Copy on write

Чем плохи java.util коллекции?

Что можно улучшить в стандартных коллекциях?

- ~~Удобство и функциональность~~
  - ✓ ~~multi map /read-though / invertible map и т.п.~~
- Эффективность
  - ✓ Производительность
  - ✓ Ёмкость (в т.ч. работа с примитивными типами)
- Многопоточность
  - ✓ Lock free
  - ✓ Copy on write

Синтаксический сахар

Кому это нужно?

Heap
Vs.
Off heap

Коллекции и многопоточность

Конкурентные коллекции

`java.util.concurrent.ConcurrentHashMap`

- Сегментированная хэш таблица
- Read/write lock на доступ к сегменту

До Java8

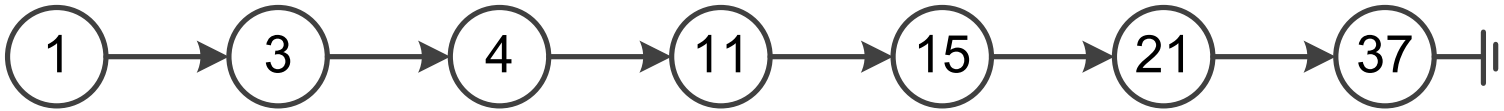
`java.util.concurrent.ConcurrentSkipListMap`

- Стохастическая “дерево подобная” структура
- Упорядоченное
- Атомарные операции с указателями
- Lock free чтение

Java 6

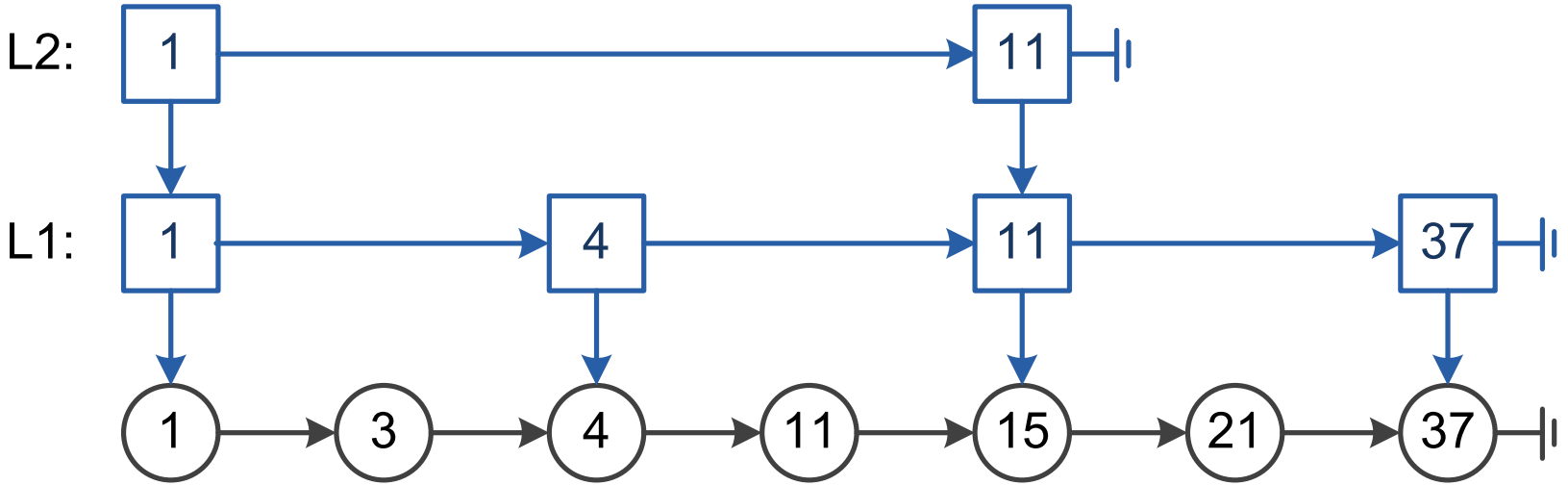
ConcurentSkipList

http://en.wikipedia.org/wiki/Skip_list



ConcurentSkipList

http://en.wikipedia.org/wiki/Skip_list



High scale lib

https://github.com/stephenc/high-scale-lib

http://sourceforge.net/projects/high-scale-lib/

- “Drop in” замена ConcurrentHashMap
- На основе атомарных операций с памятью
- Lock free

Courtesy of Cliff Click (Azul Systems)

Конкурентность на практике

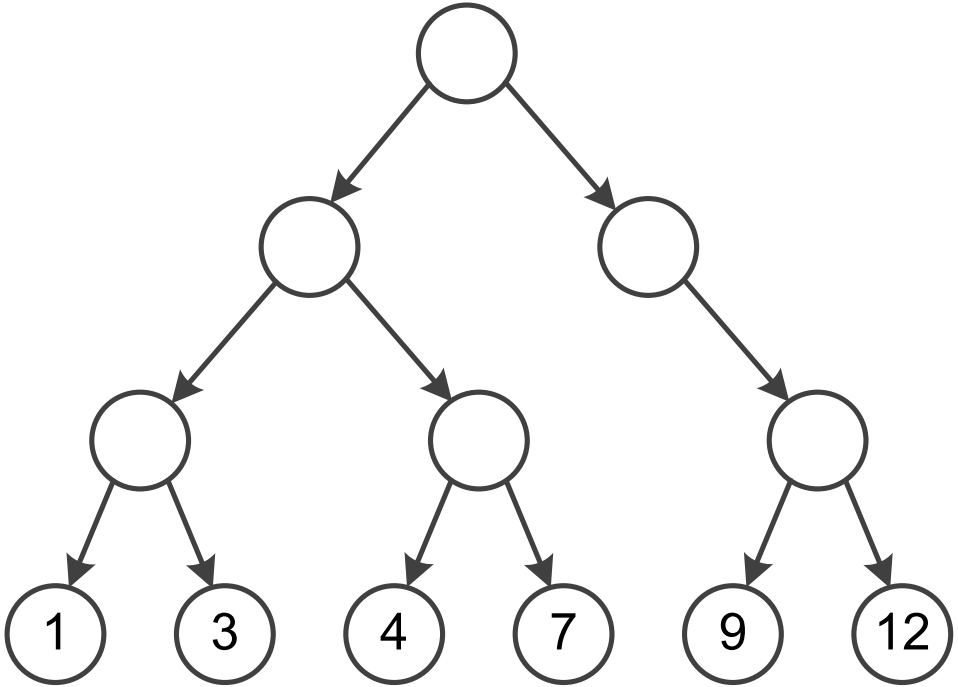
За исключением простых кэшей мы имеем …

- Одиного писателя
- Много читателей
- Атомарные апдейты нескольких структур

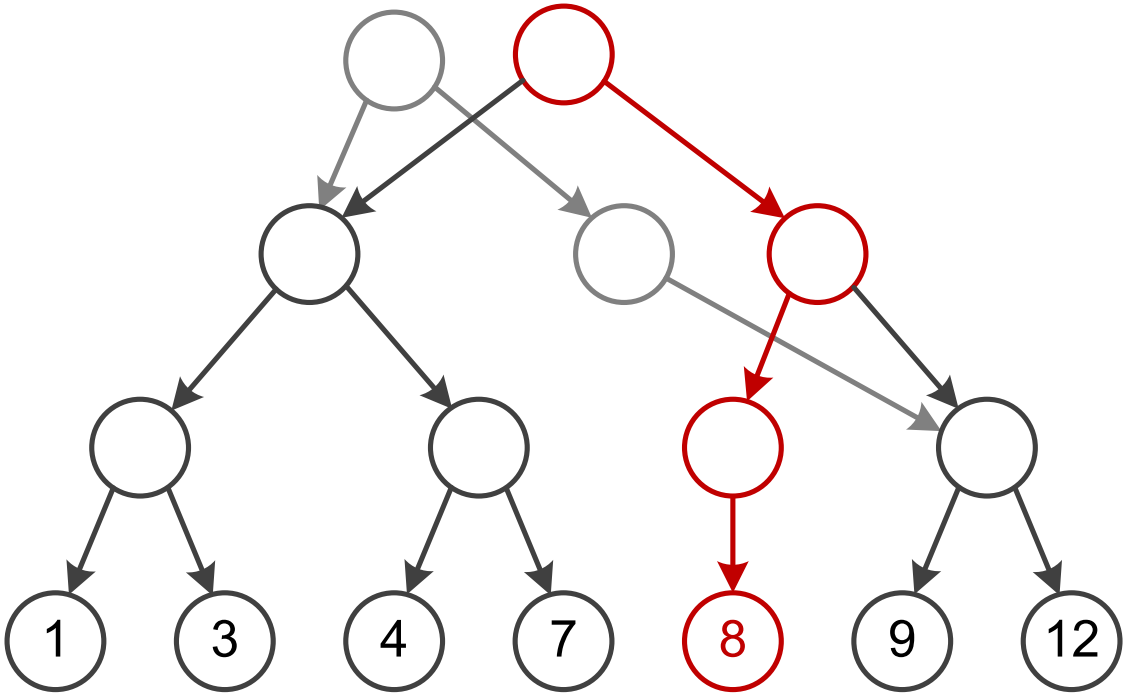
Значит нужен Copy on Write

Значит нужны деревья

Copy on Write и деревья



Copy on Write и деревья



“Конкурентные деревья”

Clojure

http://clojure.org/data_structures#Collections

- Отличная библиотека immutable коллекций
- Можно использовать из “plain Java”
  с некоторой доработкой

“Конкурентные хэш деревья”

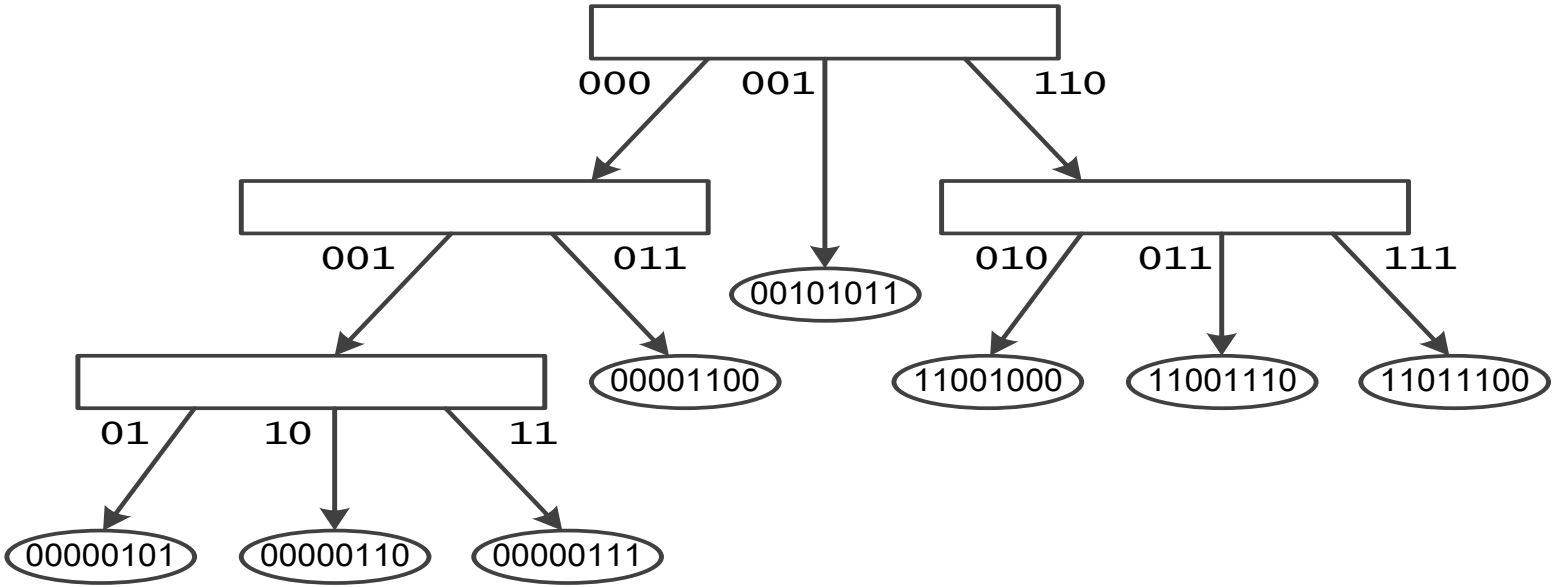
ElasticSearch

`org.elasticsearch.common.collect.CopyOnWriteHashMap`

https://github.com/elasticsearch/elasticsearch

- Hash trie (префиксное дерево) структура
- Эффективное снапшотное чтение
- Lock free

Hash trie от ElasticSearch



Коллекции примитивных типов

Примитивные коллекции

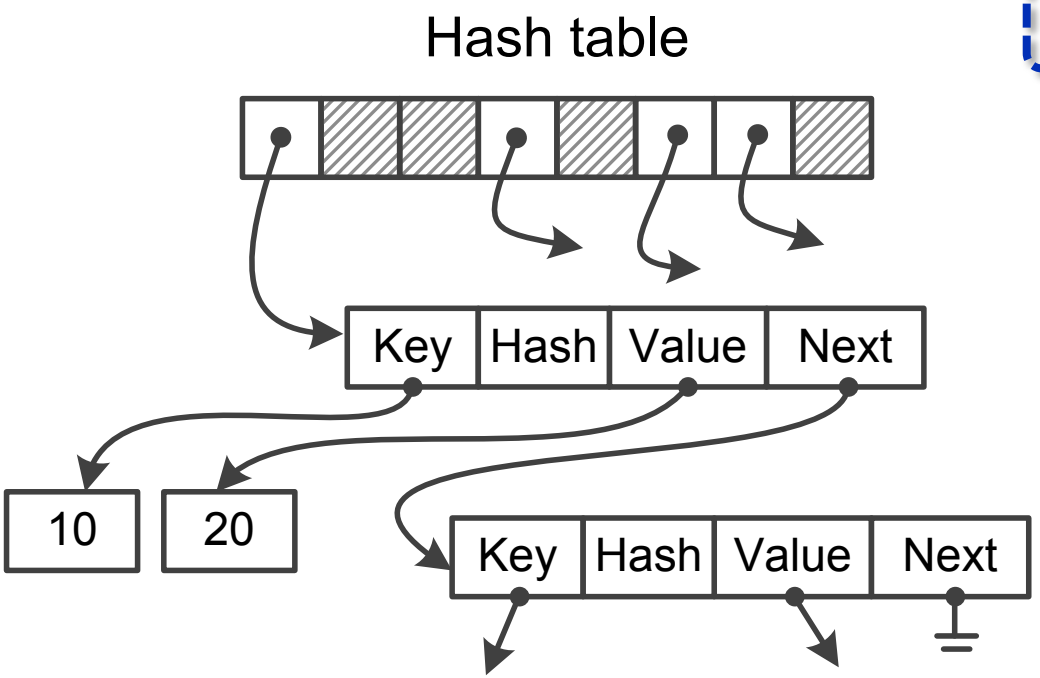
fastutils
http://fastutil.di.unimi.it/

Trove
http://trove.starlight-systems.com/

HPPC
http://labs.carrotsearch.com/hppc.html

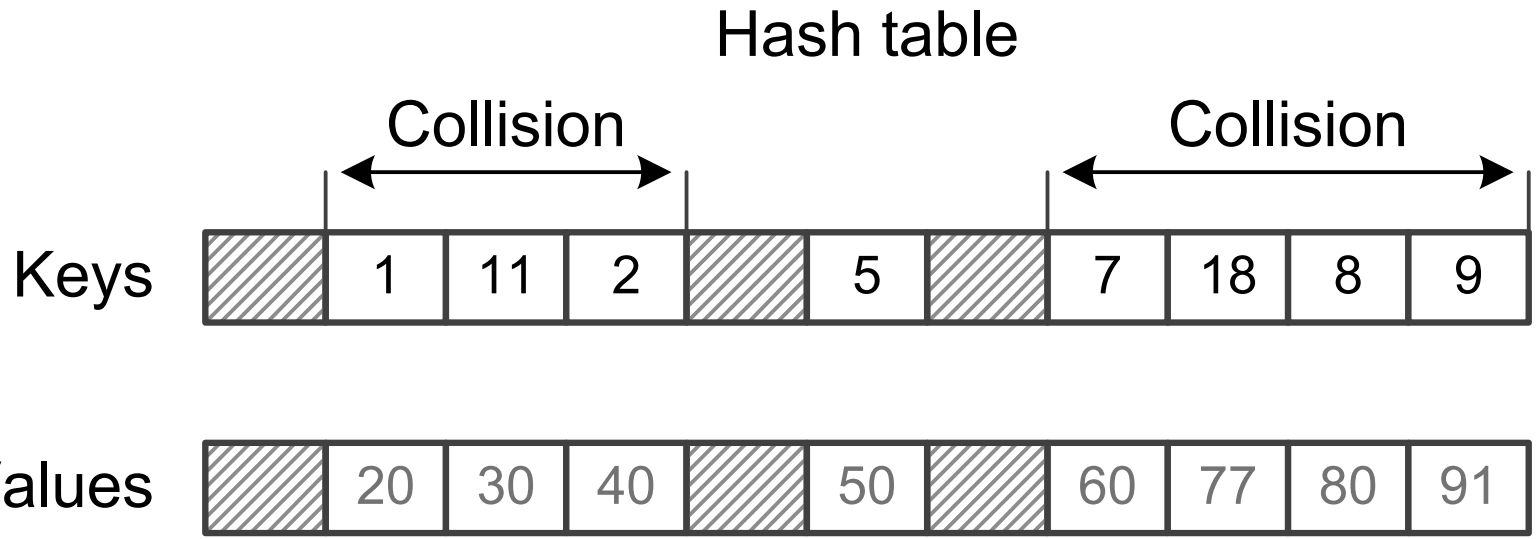
Mahout Math
https://mahout.apache.org/

Старый добрый HashMap

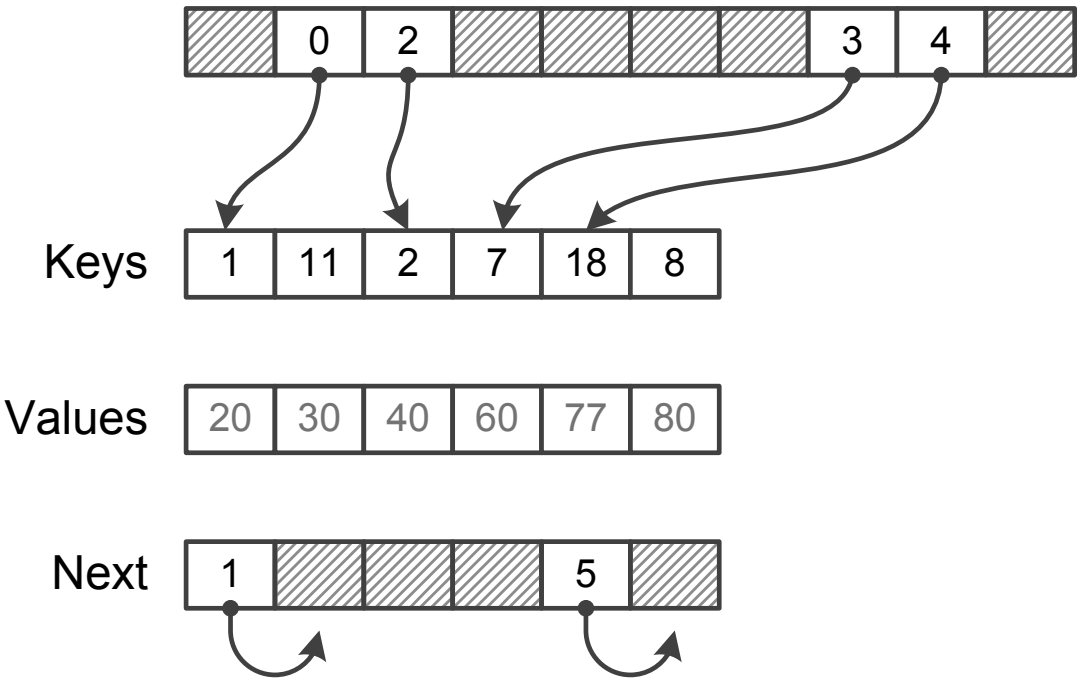


До Java8

Open Addressing hash table int –> int



Closed addressing parallel array hash table



Расход памяти: Хеш таблица long -> long

▪ java.util.HashMap (32 бита)	31.7 / 63.7
▪ java.util.HashMap (64 бита)	62.4 / 110.4
▪ fastutil open hash map	35.6
▪ Chained hash map на параллельных массивах	50.3

Поиск по нескольким полям

Коллекции примитивов это хорошо,

но что если ключём является структура?

Java объект в качестве ключа

- Лишние байты на структуру объекта (~32)
- Нагрузка на сборщик мусора

64 бита

Паралельные массивы

- Нет оверхеда по памяти
- Лишний cache miss на каждый сегмент ключа

Линейное хеширование

Linear hashing

http://en.wikipedia.org/wiki/Linear_hashing

- Позволяет расширять и сокращать хеш таблицу по одному элементу за раз
- Расширение: один слот рехешируется в два
- Сокращение: два слота рехешируется в один

А что с деревьями?

Бинарные деревья

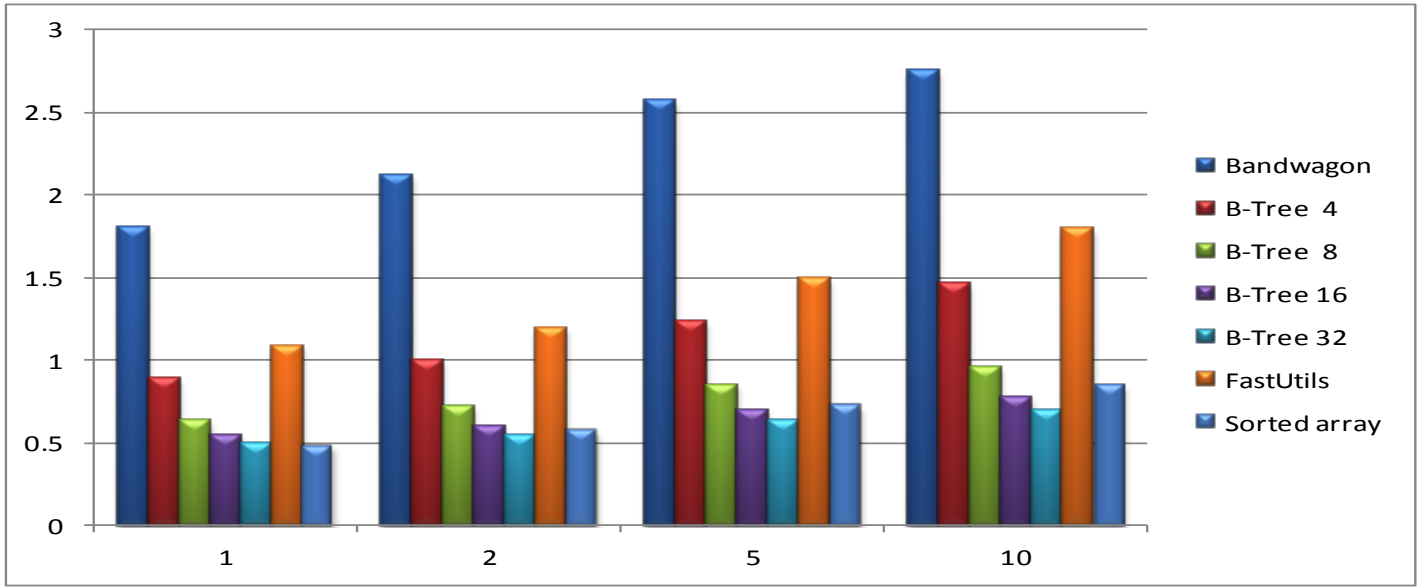
- Очень плохая локальность кэширования
- 100k -> 17 уровней -> 15 гарантированных кэш мисов

Бинарные деревья на параллельных массивах

- Ещё в два раза больше кэш мисов

B-деревья и деревья фиксированной глубины более эффективны

А что с деревьями?



А что со строками?

java.lang.String

- Immutable, можно смело отдавать указатель
- Глобальный пул существенно оптимизировали в Java 7

Альтернативы

- Постоянное копирование массивов символов
- Всё равно создавать java.lang.String для внешнего кода

Нет смысла делать структуры оптимизированные на строки

ArrayTernaryTrie @ Jetty 9

http://git.eclipse.org/c/jetty/org.eclipse.jetty.project.git/tree/jetty-util/src/main/java/org/eclipse/jetty/util/ArrayTernaryTrie.java

- Используется для поиска контекста для URL
- API позволяет передавать ключ как ByteBuffer

Другие полезные структуры

Битовые массивы

- Очень эффективные для хранения сэтов

Сортированные массивы

- Эффективны на небольших объёмах (< 500)
- Лениво сортированные массивы

Многомерные массивы ... (деревья для бедных)

- Простые префиксные деревья (2 – 3 уровня)
- Комбинация с сортировкой

NGram инедкс

Задача

- Хранение списка doc id для каждого три-грама

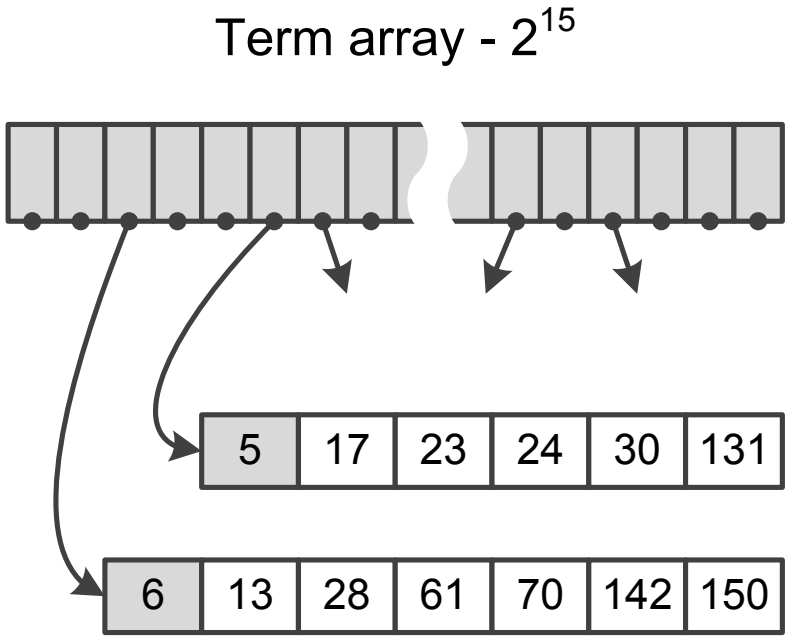
Решение “в лоб”

- Три-грам упаковывается + doc id упаковывается в long
- Используем чёрно-красное дерево для хранения упакованных значение

NGram инедкс

Оптимизированная Структура данных

- 30 раз быстрее
- 3 раза компактнее
- Вектора более 1024 элементов хранятся в виде битовых массивов



Заключение

- Создание библиотек коллекций дело не благодарное
- Оптимизация структур данных может дать многократный перформанс буст
- А может и ничего не дать
- Никому не верьте, всё надо проверять!


Deutsche Bank
Group Technology & Operations
Спасибо
Алексей Рагозин
Passion to Perform
alexey.ragozin@gmail.com
http://blog.ragozin.info