Дерево идентификаторов

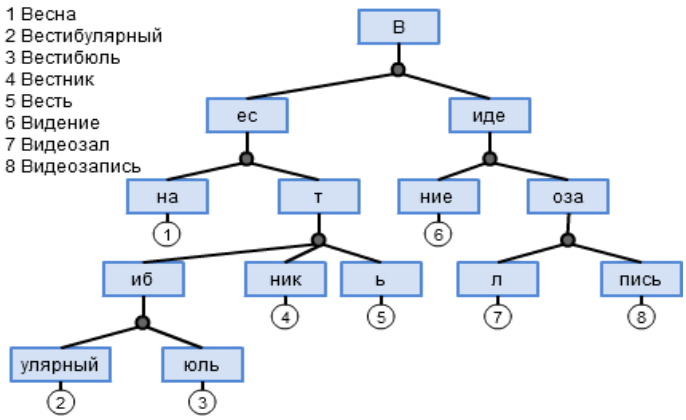
Thursday, August 29, 2019 9:33 AM

Объект содержащий публичные ключи(идентификаторы) индивидов.
Размер ключа 20 байт.

Требование :

- Быстрый поиск по ключу
- Возможность хранить возраст ключа
- Возможность однонаправленой синхронизации в условиях - "я не знаю что у меня и у тебя обновлено на текущий момент времени"
- Возможность хранить множество ключей для каждого жителя планеты
- Компактность (размер базы должен соответствовать возможностям гаджета среднестатистического пользователя)

Реализация - RadixTrie с бинарной координатной сеткой:



Структура данных:

```
//ROOT(content BRANCHs & LEAFs)
//age(2)/hash sha256hash160(20) /ChildPointArray(256*8)
//0000 /00*20 /0000000000000000 max 2070 byte
//BRANCH(content child BRANCHs & LEAFs)
//age(2)/type(1)/key size(1)/key(0-18)/hash sha256hash160(20)/childsMap(32) /ChildPointArray(1-256*8)
//0000 /00      /00         /00*18    /00*20                /00*32         /00*8 max 2104 byte
 /(ideal 153 185(branchs) =~307Mb)
//LEAF(content accounts key suffix & age)
//age(2)/type(1)/key size(1)/key(0-18)/hash sha256hash160(20)/childsMap(32) /age Array(1-256*2)/
//0000 /00     /00         /00*18    /00*20                /00*32         /00*2 max 568 byte /(ideal 39
062 500(leafs)=20GB)
//total 21GB(ideal trie for 10 000 000 000 accounts)
//
```

Где :

1. ROOT - корень дерева, содержит:
    - age - (2 байта) - содержит информацию о возрасте самого старого ключа (age - 2 байта) в дереве - это позволяет приложению определять есть ли необходимость найти самый старый ключ и удалить его из базы.
    - Hash - (20 байт) - содержит хеш сумму всех дочерних узлов, хеш сумма формируется из хешей дочерних узлов. Позволяет определить синхронизированы ли базы данных пользователей.
    - ChildPointArray (2048 байт) - массив позиций в базе где расположены данные о дочерних узлах.
2. BRANCH - ветвь дерева, содержит:
    - age - (2 байта) содержит информацию о возрасте самого старого ключа (age - 2 байта) в ветви - это позволяет приложению определять есть ли необходимость найти самый старый ключ и удалить его из ветви.
    - Type - (1 байт) - тип узла. Необходим для отличия ветвей от листьев дерева.
    - KeySize - (1 байт) - размер префикса ключа число от 0 до 18
    - Key - (0 – 18 байт) - префикс ключа
    - Hash - (20 байт) содержит хеш сумму всех дочерних узлов, хеш сумма формируется из хешей дочерних узлов.
    - ChildsMap - (32 байта) - битовая координатная сетка позволяющая определить какие суффиксы задействованы в узле(какие суффиксы листьев или ветвей есть в ветви)
    - ChildPointArray - (1 - 2048 байт) - массив позиций в базе где расположены данные о дочерних узлах
3. LEAF- Лист дерева, содержит:
    - age - (2 байта) содержит информацию о возрасте самого старого ключа (age - 2 байта) на листе - это позволяет приложению определять есть ли необходимость найти самый старый ключ и удалить его из листа.
    - Type - (1 байт) - тип узла. Необходим для отличия ветвей от листьев дерева.
    - KeySize - (1 байт) - размер префикса ключа число от 0 до 18
    - Key - (0 – 18 байт) - префикс ключа
    - Hash - (20 байт) содержит хеш сумму всех суффиксов, хеш сумма формируется из суффиксов ключей на листе(координатной сетки) и их возрастов.
    - ChildsMap - (32 байта) - битовая координатная сетка позволяющая определить какие суффиксы задействованы в узле(какие ключи есть на листе)
    - AgeArray - (1 - 2048 байт) - массив возрастов ключей расположенных на листе записываемых в порядке определенном на координатной сетке.

Физически база располагается в файле trie.dat в виде строки байтов, приложение при работе оперирует кусками этой строки используя координаты которые отражены в структуре данных узлов базы.

Новые данные вносятся в файл либо в конце файла, либо в неиспользуемое пространство(кусок). Перед записью программа ищет подходящий по размеру кусок неиспользуемого пространства и если не находит то делает запись в конец файла, если подходящий свободный участок файла найден то запись делается туда. Если найденный участок больше чем то, что нужно записать, то координаты остатка неиспользованного пространства записывается в вспомогательную базу(см. ниже).

Данные об высвобожденном от использования пространстве хранятся в отдельной вспомогательной базе данных `SQLiteDatabase`.

Приложение постоянно оптимизирует данные об этих кусках пространства стараясь объединять в большие куски(фрагментировать).

Поиск ключа потребителя услуг:
Благодаря технологии построения базы данных в виде дерева RadixTrie, поиск наличия ключа потребителя услуг, приложение производителя услуг может осуществить за 20 итераций, при том что содержать такое дерево может 1 461 501 637 330 903 000 000 000 000 000 000 000 000 000 000 000 вариантов ключей, что стиховой хватит на всех жителей земли на все времена.

Синхронизация:

Благодаря тому, что узлы дерева базы данных содержат хеш суммы возрастов и ключей, то итоговый корневой хеш всего дерева можно легко сравнить с хешем дерева другой такой же базы(с базой другого пользователя)
И если хеш суммы не совпадают, то можно проходить по дочерним узлам дерева ветвей дерева и листьям на ветвях дерева и быстро определять где именно, в кой ветви дерева на каком листе дерева произошли изменения. В случае если эти изменения соответствуют требованиям протокола синхронизации "Жидкая экономика", то они вносятся в базу данных потребителя(самим потребителем).

Другими словами приложение потребителя при синхронизации запрашивает хеши дерева ветвей и листьев производителя услуг и сравнивает со своими и по мере нахождения несоответствия в отдельных частях дерева углубляется в ветви и листья чтобы понять в чем именно несоответствие, и если оно требует обновления то обновляет свою базу данных исходя из данных производителя услуг.

Удаление устаревших ключей(идентификаторов пользователей):
Благодаря тому, что узлы дерева базы данных содержат возрасты ключей и каждая ветвь и листья на нем содержит возраст самого старого ключа, приложение может легко находить внутри своей базы данных устаревшие ключи и удалять их. Кроме того возрасты позволяют приложению потребителя услуг определять есть ли необходимость обновить или добавить ключ который оно нашло в дереве производителя.