Eats

Данные содержат 18 58 328 уникальных номеров телефонов с именем клиента (20 символов в среднем)

Определимся с форматом записи номера телефона, пусть номера записаны в таком виде: **7XXXXXXXXXX**, то есть 11 символов на один номер телефона (к такому формату всегда можно заранее преобразовать базу)

В каком формате хранить номера?

1. В виде строки
   Для выбранного формата записи телефонного номера получим 11 байт на номер или 11 * 18 758 328 = 206,3mb
2. В виде 64 битного числа
   Для числа 79 999 999 999 потребуется 37 бит = 4,652 байт

Например, в формате long (Java), получим 8 байт на 1 номер телефона или 150,1mb на всю базу номеров

Сколько занимают имена?

Исходя из условия, получим в среднем 20 байт на имя пользователя, то есть 18 758 328 * 20 = 375.2mb

Средний размер всех пар (телефон: имя)

Если номера телефонов хранятся в строках, тогда предполагаемый объем данных будет равен $581.5MB$
Если же в формате числа - $525.3MB$

В базах данных телефонные номера обычно хранятся в строковых форматах данных, поэтому пока остановимся на варианте с 581.5mb, будем работать со строками

Реализация поиска за постоянное время

Для решения данной задачи будем использовать хеш таблицу, ключи - хеши от номеров телефона, в узлах лежат пары (номер : имя)

В качестве хэш функции возьмем $hash(string) = abs(hashCode(string))\%tableSize$, где hashCode - стандартная операция хеширования в Java.

Размер таблицы возьмем равным 18 758 329 - простое число для более эффективного распределения по "ячейкам" таблицы, в качестве самой таблицы используем массив LinkedList[tableSize]. Вставка будет осуществляться следующим образом:

```
if (table[h] == null) {
    table[h] = new LinkedList<Data>()
}
table[h].add(Data)
```

Размер одного узла LinkedList 24 байта, если выбранная хеш функция достаточно хорошая, чтобы равномерно респределить значения по ячейкам, тогда самое наибольшее, что можно получить, это 18 758 328 объектов LinkedList + 1 неисопользованный указатель (сслыка), размер которой 4 байта, то есть в сумме - $18758328 * 24 + 4 = 450.2MB$

Корректирование первоначальных размеров данных

Размер 1 символа в Java = 2 байта, поэтому подсчитанный размер базы данных нужно умножить на 2, более того, размер объекта String на 64битных системах помимо самого размера массива символов = 16 байт, то есть только данные будут занимать: $18758328 * (20 + 11 + 16) * 2 = 1.763GB$

Таким образом, все вместе, то есть сама таблица + данные будут занимать около $2.213GB$

Оптимизация потребляемой памяти

Первое, что можно оптимизировать, это объем памяти, потребляемый самой таблицей. Вместо LinkedList можно использовать ArrayList, который будет потреблять 4 байта на элемент вместо 24, то есть суммарно таблицу можно уместить в $75.03MB$ вместо $450.2MB$

Далее вместо класса String можно использовать более легковесный вариант - массив примитивов char[], с явным указанияем байта - символа конца строки, в таком случае размер имени в среднем станет равным 21, а номер телефона 12 символов. тогда общий размер станет равен
$(21 + 12) * 2 * 18758328 = 1.238GB$ вместо $1.763GB$

Минимизировав, получаем $1.238GB + 75.03MB = 1.313GB$