Решения

Eats

> Условие:
> 30_000_000 записей всего
> 18_758_328 уникальных номеров телефонов
> ~ 20 длина полного имени

Пусть оперативная память ничем не ограничена (В условии не указано). Для примеров будем использовать Java 8.

Решение задачи 1)

Каждый номер телефона состоит из кода страны + кода оператора + кода клиента. Пусть есть таблицы с кодами страны(они известны) их N штук, аналогично есть таблица с кодом оператора, пусть K. Остается 7 цифр, на каждой позиции может быть одна из 10 цифр => 10^7 может быть всего вариантов. Итого всего может быть N * K * 10^7 вариантов номеров телефонов.

Чтобы поиск выполнялся со сложностью в среднем ~ O(1) необходимо использовать структуру данных - хеш-таблица (метод перемешивания или сцепления). Желательно будет оптимизировать хеш - функцию, чтобы она потребляла меньше ресурсов и возникало как можно меньше коллизий и сохранять ее значения вместе с ключом + не будем сильно задумывать можно ли будет.

Будем хранить все записи в таблице и использовать в качестве ключа - значение хеш функции(номер телефона смапленный в число), а в качестве значения - полное имя.

Не будем хранить записи с одинаковыми номерами телефонов (запись будет создавать коллизию поскольку будет браться хеш от такого же номера телефона). Т.е. по сути нам нужен словарь с хеш функцией.

В Java хеш таблица представлена как HashMap.

Тогда нам потребуется вот столько памяти на полную запись:

1. В одной элементе будем использовать две строки типа (String). Заголовок в String занимает - 8 байт
2. Длина массива (int) - 4 байта

3. Ссылка на char[] (1 символ занимает 2 байта), а у нас в среднем 20 символов - 40 байт
4. Сслыка на номер телефона (его тоже будем держать в памяти) ~ 10 символов по 2 байта = 20 байт
5. Суммарно на одно полное имя - 8 + 4 + 40 + 20 = 72 байта

Хеш можно хранить как int и одни хеш будет занимать 4 байта

Суммарно одна запись будет занимать 56 байт, таких записей будет 18_758_328 * 72 = 1_350_599_616 байт памяти для хранения нашей базы.

Ps Плюс еще различный системные элементы, которые требует сам язык/платформа и его структуры.

Решение задачи 2)

Решение: можно использовать сжатые префиксные деревья или схожие структуры или алгоритмы для сжатия данных. Префиксные деревья более предпочтительны посколько у многих номеров телефонов совпадает код страны и код оператора. а также совпадают начальные цифры, так что можно хранить как минимум в два раза меньше информации, и использовать только метки. Также у многих пользователей совпадают имена.

Итого: можно рассматривать одну строку номер телефона + 20 символов информация о пользователе. и раскладывать все это в сжатое префиксное дерево, тогда получится примерно 2 мб на один символ

1. 30 символов одна строка * 9_000_000 = 270_000_000 байт

Ps Плюс еще различный системные элементы которые требует сам язык/платформа и его структуры (в них входит ссылка на объект). Можно также использовать префиксные деревья.

I got you

Описание решения: Будем использовать принцип Producer - Consumer. Возможно можно было решить данную задачу с помощью блокировок, но мое решение более общее и может быть применимо к разного рода схожим задачам. Основной принцип работы будет вот такой:

> Первый поток "Cher" будет итеративно идти по именам и выводить строчки, если его имя совпадает с "Cher", печатаем строчки текста. Если первый поток нашел "Sonny", то

> передается информация во второй поток и выводится с помощью него. Если встретилось "Sonny, Cher", то будем выводить сначала первым потоком, а потом выводить его во вторым.

Сделаем класс с песней (чтобы точно был ООП стиль):

```
package IGotYouBabe;


public final class Song {
    public final
    String[][] lyrics = {
            {"Cher", "They say we're young and we don't know \nWe won't find out until \
            {"Sonny", "Well I don't know if all that's true \n'Cause you got me, and bal
            {"Sonny", "Babe"},
            {"Sonny, Cher", "I got you babe \nI got you babe"},
            {"Cher", "They say our love won't pay the rent \nBefore it's earned, our mor
            {"Sonny", "I guess that's so, we don't have a pot \nBut at least I'm sure o
            {"Sonny", "Babe"},
            {"Sonny, Cher", "I got you babe \nI got you babe"},
            {"Sonny", "I got flowers in the spring \nI got you to wear my ring"},
            {"Cher", "And when I'm sad, you're a clown \nAnd if I get scared, you're alw
            {"Cher", "So let them say your hair's too long \n'Cause I don't care, with y
            {"Sonny", "Then put your little hand in mine \nThere ain't no hill or mounta
            {"Sonny", "Babe"},
            {"Sonny, Cher", "I got you babe \nI got you babe"},
            {"Sonny", "I got you to hold my hand"},
            {"Cher", "I got you to understand"},
            {"Sonny", "I got you to walk with me"},
            {"Cher", "I got you to talk with me"},
            {"Sonny", "I got you to kiss goodnight"},
            {"Cher", "I got you to hold me tight"},
            {"Sonny", "I got you, I won't let go"},
            {"Cher", "I got you to love me so"},
            {"Sonny, Cher", "I got you babe \nI got you babe \nI got you babe \nI got yo
    };
}
```

В роле Prodecer - Sender, вот его описани:

```java
package Sender;

import IGotYouBabe.Song;
import Model.Packet;

import java.util.Objects;

public class Sender implements Runnable {
    private final String name = "Cher";
    private final Packet pack;

    public Sender(Packet pack) {
        this.pack = pack;
    }

    @Override
    public void run() {
        Song song = new Song();
        String[][] lyrics = song.lyrics;

        for (String[] lyric : lyrics) {
            if (Objects.equals(lyric[0], "Cher")) {
                System.out.println(this.name + ": " + lyric[1]);
            }
            else if (Objects.equals(lyric[0], "Sonny")) {
                pack.send(lyric[1]);
            }
            else if (Objects.equals(lyric[0], "Sonny, Cher")
            {
                System.out.println(this.name + ": " + lyric[1]);
                pack.send(lyric[1]);
            }

            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }

        }
    }
}
```

В роли Consumer - Receiver, он будет получаеть информацию и выводить ее.

```java
package Receiver;

import Model.Packet;

import java.util.Objects;

public class Receiver implements Runnable {
    private final String name = "Sonny";
    private final Packet loadPacket;

    public Receiver(Packet loadPacket) {
        this.loadPacket = loadPacket;
    }

    @Override
    public void run() {
        for (String receivedMessage = loadPacket.receive(); !Objects.equals(receivedMes
             receivedMessage = loadPacket.receive()) {

            System.out.println(this.name + ": " + receivedMessage);
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }

        }
    }
}
```

Общаться треды будут с помощью класса Packet. В этот класс будет записывать информацию поток 1 и ждать пока поток 2 ее прочитает. Для этого добавлены Thread.sleep(1000).

```
package Model;


public class Packet {

    private boolean transfer = true;
    private String phrase;


    public synchronized void send(String data) {
        while (!transfer) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("Thread Interrupted");
            }
        }
        this.transfer = false;

        this.phrase = data;

        notifyAll();
    }

    public synchronized String receive() {

        while (transfer) {
            try {
                wait();
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                System.out.println("Thread Interrupted");
            }
        }

        transfer = true;
        String result = this.phrase;
        notifyAll();
        return result;
    }

}
```

В классе Solver создадим два треда и запустим их.

```java
import Model.Packet;
import Receiver.Receiver;
import Sender.Sender;

public class Solver {

    public static void main(String[] args) {

        Packet pack = new Packet();
        Thread sender = new Thread(new Sender(pack));
        Thread receiver = new Thread(new Receiver(pack));

        sender.start();
        receiver.start();
    }
}
```

Ссылка на гит с решением:

https://github.com/CtrAtlDel/Data

Если на гите нет доступа или иные проблемы, напишите мне:

https://t.me/didyoumissmee