Задание 2.

Представим на время, что вы black hat, и у вас в распоряжении оказались данные ~30 млн. заказов некоторого сервиса доставки еды за некоторый период 2021 и 2022 года.

Поверхностный анализ показал, что данные содержат 18 758 328 уникальных телефонов с полным именем клиента, а средняя длина полного имени - 20 символов (латинских или кириллических).

Допустим, вы хотите развернуть веб-сервис, который позволит по номеру телефона найти полное имя клиента. Но вы не хотите оставлять следы на диске или в базе данных - придется все держать в памяти. Но еще вы не хотите зря тратить крипту на слишком большой сервер. Поэтому давайте оценим, сколько памяти займут эти данные:

- если мы хотим реализовать поиск за постоянное время, т.е. O(1)?
- если мы хотим занять как можно меньше памяти?

Самое главное - объяснить, как вы пришли к той или иной числовой оценке.

Можно выбрать любой язык программирования/платформу.

Решение:

За основу возьмем язык Java и JVM 64-bit.

1. Поиск за постоянное время O(1).

Для достижения постоянного времени доступа к данным подходит структура данных HashMap. В HashMap ключом будет номер телефона, а значением — полное имя клиента.

Структура HashMap

- Таблица бакетов (buckets): HashMap хранит пары ключ-значение в массиве бакетов. Каждый бакет может содержать одну или несколько пар (в случае коллизий).

- Node/Entry: Каждая пара ключ-значение представлена объектом Node (в более старых версиях Java – Entry). Этот объект содержит: хэш ключа, ключ, значение, ссылка на следующий Node (в случае коллизий).

Оценка объема памяти:

Ключ и значение.

- **Ключ (номер телефона).** Возьмём за основу российские номера телефонов, состоящие из 11 (например, 79123456789). В Java каждый символ в формате UTF-16 занимает 2 байта. Итого на ключ потребуется 11 символов * 2 байта/символ = 22 байта.
- **Значение (полное имя).** Средняя длина имени — 20 символов. В UTF-16 это 20 символов * 2 байта/символ = 40 байт.

Служебные данные HashMap.

- Ссылка на ключ (8 байт);
- Ссылка на значение (8 байт);
- Хэш-код ключа (4 байта);
- Ссылка на следующий узел (8 байт);
- Выравнивание объекта (возможно, до 8 байт).
- Ссылка на первый узел в бакете (8 байт)

Итого на одну запись: 22 (ключ) + 40 (значение) + 8 + 8 + 4 + 8 + 8 + 8= 106 байт

Общий объем: 106 байт/запись * 18 758 328 записей = 1 988 382 768 байт ≈ 2 ГБ

2. Минимальное потребление памяти.

Если приоритет — минимизировать объем памяти, можно использовать более компактные структуры данных. Например, Trie (бор) или DAWG (Directed Acyclic Word Graph). Эти структуры позволяют эффективно хранить строки, используя общие префиксы.

Оценка объема памяти:

Префиксное дерево для номеров телефонов. Эффективно сжимает общие префиксы номеров телефонов, но каждый узел требует хранения

дополнительной информации (указатели на дочерние узлы, возможно, флаги состояния). Тем не менее, размер в байтах может быть снижен примерно на 20-30% по сравнению с хеш-таблицей.

Префиксное дерево для имен. Аналогично, но учитывая возможные общие начала имен (например, распространенные имена и фамилии), можно добиться более значительной экономии памяти

Таким образом, минимально возможное использование памяти может быть на уровне 70% от изначально расчитанного объема при использовании хеш-таблицы.

Возможный объем: 2 ГБ * 0.7 = 1,4 ГБ