Задача A. [Fixed Set] (75 баллов)

Ограничение по времени:	[1.5 секунды]
Ограничение по памяти:	[64Mb]

Реализуйте следующий класс для хранения множества целых чисел:

```
class FixedSet {
 public:
  FixedSet();
  void Initialize(const vector<int>& numbers);
  bool Contains(int number) const;
};
```

`FixedSet` получает при вызове `Initialize` набор целых чисел, который впоследствии и будет хранить. Набор чисел не будет изменяться с течением времени (до следующего вызова `Initialize`). Операция `Contains` возвращает `true`, если число `number` содержится в наборе. Мат. ожидание времени работы `Initialize` должно составлять $O(n)$, где n — количество чисел в `numbers`. Затраты памяти должны быть порядка $O(n)$ в худшем случае. Операция `Contains` должна выполняться за $O(1)$ в худшем случае.

С помощью этого класса решите модельную задачу: во входе будет дано множество различных чисел, а затем множество запросов — целых чисел. Необходимо для каждого запроса определить, лежит ли число из запроса в множестве.

В первой строке входа — число n — размер множества. $1 \leqslant n \leqslant 100\,000$. В следующей строке n различных целых чисел, по модулю не превосходящих 10^9. В следующей строке — число q — количество запросов. $1 \leqslant q \leqslant 1\,000\,000$. В следующей строке q целых чисел, по модулю не превосходящих 10^9. Для каждого запроса нужно вывести в отдельную строку “Yes” (без кавычек), если число из запроса есть в множестве и “No” (без кавычек), если числа из запроса нет в множестве. См. примеры.

тест	ответ
3 1 2 3 4 1 2 3 4	Yes Yes Yes No
3 3 1 2 4 10 1 5 2	No Yes No Yes

Решение

Нужно реализовать либо вариант `FixedSet` с помощью двухуровнего хеширования, либо вариант с использованием случайного графа. На ревью принимаем только двухуровневое хеширование.

1. FixedSet. Решение и анализ

Будем для нашей задачи реализовывать хеширование по методу *FKS*.

Описание хеш-таблицы.

Обозначим через $\mathcal{K} = \{0, 1, \ldots, k-1\}$ диапазон, из которого берутся числа, подающиеся на вход, а через n – их количество. Пусть $\mathcal{H}$ – некоторое универсальное семейство функций, действующих из $\mathcal{K}$ в $M = \{0, 1, \ldots, n-1\}$. Заметим, что в нашем случае количество значений хеш-функции равно количеству ключей.

Опишем подробно устройство хеш-таблицы, т.е. опишем метод `FixedSet::Initialize`.

1. Строим таблицу первого уровня. Для этого в цикле делаем следующее:
   (a) Выбираем случайную хеш-функцию h из $\mathcal{H}$. Строим для нее стандартную хеш-таблицу. Обозначим через b_i количество ключей в i-м `bucket`'е. Обозначим $S = \sum_{i=1}^{n} b_i^2$. Если $S < 4n$, то завершаем цикл и переходим к следующему пункту. Иначе удаляем таблицу и переходим к началу пункта.
2. Для каждого `bucket`'а выполняем следующее:
   (a) Выберем универсальное сем-во хеш-функций $\mathcal{H}_i$, действующее из $\mathcal{K}$ в $\{0, 1, \ldots, b_i^2 - 1\}$.
   (b) В цикле выбираем случайно равномерно функцию g_i из семейства $\mathcal{H}_i$ и проверяем, что у нее нет коллизий внутри нашего `bucket`'а. Если коллизий нет, то строим хеш-таблицу размера b_i^2 с функцей g_i для ключей `bucket`'а. Если же у g_i есть коллизии, то повторяем текущий пункт.

Таким образом, мы построим двухуровневую хеш-таблицу.

Опишем действия при запросе `FixedSet::Contains`.

По числу $v \in \mathcal{K}$ мы считаем $h(v)$, идем в соответсвующий j-й `bucket` и возвращаем результат запроса `Contains` для внутренней хеш-таблицы и ключа v.

Корректность.

Тот факт, что методы работают корректно, следует из свойств хеш-таблиц.

Требуется обоснование того, что метод `FixedSet::Initialize` не уходит в бесконечный цикл ни на одном из циклов случайного независимого выбора хеш-функций из универсального семейства. В теории это может произойти на всех этапах выбора хеш-функций. Однако на практике такого не происходит, поскольку вероятность долго выбирать функцию эксоненциально убывает с ростом количества шагов случайного перевыбора функции (См. анализ затрат по памяти и времени.)

Лемма 1. $\mathcal{H}$ – семейство универсальных функций. Пусть размер таблицы равен l^2, где l – количество ключей, которые необходимо хранить. Тогда:

$$P\left(h \text{ имеет коллизии}\right) \leqslant \frac{1}{2}$$

Доказательство. Обозначим $\alpha_{ij} = \mathbb{1}[h(a_i) = h(a_j)]$. По свойствам матожидания и из свойств универсальной хеш-функции имеем:

$$E(\alpha_{ij}) = \frac{1}{l^2}$$

Найдем матожидание количества коллизий β у h на ключах. Заметим, что $\beta = \sum_{i<j} \alpha ij$, а отсюда

$$E\beta = \binom{l}{2} * \frac{1}{l^2} = \frac{2(l-1)}{l} < \frac{1}{2}$$

Отсюда по неравенству Чебышева $P(\beta \geqslant 1) \leqslant \dfrac{E\beta}{1} < \dfrac{1}{2}$, а значит:

$$P(\beta = 0) = 1 - P(\beta \geqslant 1) > 1 - \frac{1}{2} = \frac{1}{2}$$

■

Лемма 2. Пусть дана стандартная хеш-таблица с универсальной хеш-функцией $h \in \mathcal{H}$, размер которой равен количеству ключей ($m = n$). Обозначим через b_i количество ключей в i-м `bucket`'е (т.е. том, в который попали ключи v со свойством $h(v) = i$). Тогда:

$$E\left(\sum_{i=1}^{n} l_i^2\right) = 2n - 1$$

Доказательство. Перепишем сумму внутри матожидания:

$$\sum_{i=1}^{n} l_i^2 = \sum_{i=1}^{n} l_i + 2\sum_{i=1}^{n} \binom{l_i}{2}$$

Заметим, что $\sum_{i=1}^{n} \binom{l_i}{2}$ – суть число коллизий β функции h на наших ключах. Матожидание для β, по аналогии с предудущим пунктом, равно $\dfrac{\binom{n}{2}}{n} = \dfrac{n-1}{2}$. Имеем:

$$E\left(\sum_{i=1}^{n} l_i^2\right) = E\left(\sum_{i=1}^{n} l_i\right) + 2E\beta = n + n - 1 = 2n - 1$$

■

Таким образом, из леммы 1 и свойств распределения Бернулли следует, что матожидание количества выборов функций для построения хеш-таблиц второго уровня для каждого `bucket`'а не более двух.

Осталось обосновать, почему генерация хеш-функции для построения таблицы первого уровня не займет много времени.

Оценим, применив неравенство Чебышева и лемму 2, вероятность того условия, при котором функция генерируется заново:

$$P\left(\sum_{i=1}^{n} l_i^2 \geqslant 4n\right) \leqslant \frac{E\left(\sum_{i=1}^{n} l_i^2\right)}{4n} = \frac{2n-1}{4n} < \frac{1}{2}$$

Аналогично, получаем, что матожидание количества выборов хеш-функций для построения хеш-таблицы первого уровня не больше двух.

Анализ затрат по времени и памяти.

Затраты по памяти. Из алгоритма следует, что сумма размеров всех таблиц не превосходит $5n$ – не более $4n$ для таблиц второго уровня и n для таблицы первого уровня. Для каждого случайного выбора хеш-функции из универсального семейства мы используем константое количество памяти.

Следовательно, алгоритм использует $O(n)$ памяти в худшем случае.

Затраты по времени. Матожидание времени работы `FixedSet::Initialize` пропорционально количеству операций выбора хеш-функций. На каждом шаге матожидание количества выборов функции не превосходит двух, а значит время не более чем линейно по n.

Поскольку коллизий в таблицах второго уровня нет, то для каждого числа метод `Contains` для таблицы второго уровня работает за константу, а значит и метод `FixedSet::Contains` гарантировано работает за $O(1)$.

Замечание об универсальном семействе. По ходу алгоритма нам также необходимо будет генерировать функцию из универсального семейства хеш-функций.

В задаче мы имеем дело с числами из отрезка $\left[-10^9, 10^9\right]$. Для удобства переведем каждое из них в диапазон $\left[0, 2 \cdot 10^9\right]$, прибавив в каждому 10^9. Воспользуемся стандартным универсальным семейством хеш-функций, которое после нашего преобразования будет иметь вид:

$$\mathcal{H}_{(p,m)} = \{h_{ab}(x) = ((a(x + 10^9) + b) \mod p) \mod m \mid a, b \in \mathbb{F}_p,\ a \neq 0\},$$

В качестве простого модуля возьмем `2’000’000’009`. Такой выбор обоснован тем, что это первый (или один из первых) простой модуль, превосходящий $2 \cdot 10^9$ – наибольшее возможное число после преобразования. На каждом шаге (включая первый и второй уровень) выбора хеш-функций будем брать функцию случайно равномерно (и независимо от выборов на других шагах) из семейства $\mathcal{H}_{(p,m)}$, подставляя каждый раз желаемый размер m таблицы.