Fixed Set

Чесаков Д. Г.

19 октября 2019 г.

1 Условие задачи

При вызове Initialize FixedSet получает 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: на вход будет дано множество различных чисел, а затем множество запросов — целых чисел. Необходимо для каждого запроса определить, лежит ли число из запроса в множестве. В задаче строится множество из n различных чисел, после чего выполняется q запросов, $1 \leq n \leq 100000$, $1 \leq q \leq 1000000$. Все числа в задаче по модулю не превосходят 10^9.

2 Алгоритм решения

Для построения вышеуказанной структуры нам надо будет воспользоваться следующим алгоритмом двухуровневого хеширования. На первом уровне мы хешируем наши n значений в m ячеек с помощью функции H вида $h(k) = ((a \times k + b)mod(p))mod(m)$, выбранной из семейства универсальных хеш функций, где m равно n, а p достаточно большое простое число, большее чем $2 * 10^9$, поскольку числа в задаче по модулю не превосходят 10^9. Числа a, b выбираются случайно из поля вычетов по модулю p, с условием также, что $a \neq 0$. На данном этапе у нас могут случиться коллизии, однако мы выбираем функцию так, чтобы сумма квадратов размеров бакетов была линейна по n, где в коде данное условие будет реализовано как то, что сумма квадратов размеров бакетов будет меньше, чем $10 * N$. Тогда посмотрим на элементы, которые под воздействием нашей хеш фунеции перешли в одну и ту же ячейку хеш таблицы, и для каждой ячейки $0, ..., m-1$ реализуем еще одну соответствующую хеш функцию H_j, генерируя ее пока не будет выполняться условие отсутствия коллизий. Данная функция уже без коллизий отобразит множество элементов в бакете b_j в другую хеш таблицу, множество ячеек которой m_j равно квадрату количества элементов в бакете b_j.

3 Доказательство правильности

Количество памяти которое нам потребуется на первом уровне будет O(n) так как m линейно зависит от n, число памяти на втором уровне будет равно $\sum_{j=0}^{m-1} b_j^2 = 2 \times S + \sum_{j=0}^{m-1} b_j = O(n^2/m + n)$, где $S = \sum_{i<j} \mathbb{1}[H(a_i) = H(a_j)]$, по доказанному в лекции, если выбрать такую H, чтобы сумма квадратов размеров бакетов была линейна по n при m линейному по n.

Значит затраты памяти будут линейны по n в худшем случае. Также по доказанному в лекции $P[S \geq 1] \leq ES = O(n^2/m)$. Тогда мат ожидание времени поиска таких функций H_j на каждом этапе, чтобы не было коллизий будет $O(1)$ (как я уже говорил m_j квадратично по размеру бакета b_j), а суммарное время будет тогда $O(n)$. А мат. ожидание количества попыток, чтобы найти такую функцию H, чтобы сумма квадратов бакетов была линейна по n, будет O(1) (по неравенству маркова и показано в лекции), время проверки каждой функции – O(n). Значит мат. ожидание требуемого времени будет O(n), а памяти O(n) в худшем случае. Операция Contains будет занимать O(1) в худшем случае, поскольку это выполнение двух хэш функций над элементом и проверка, занята ли получившаяся ячейка в соответствующей хэш таблице второго уровня или нет.

4 Временная сложность – ассимптотика

Мат. ожидание времени работы Initialize $O(n)$. Операция Contains выполняется за O(1) в худшем случае.

5 Затраты памяти – ассимптотика

O(n) в худшем случае.