Теория к задаче Fixed Set

1 Алгоритм решения

Initialize: Пусть n - количество чисел в переданном векторе.

Если $n == 0$, заканчивает работу. Создаем вектор векторов $table$ чисел размера n, и такой же вектор хэш функций $hash\_functions$.

В цикле генерируем случайную хэш функцию из универсального параметрического семейства $f(x) = ((a * x + b) \mod prime) \mod n$, где $prime$ - простое число, большее любого числа из входных данных, a и b равномерно распределена на $[1..prime)$, пока $\sum l_i^2 > 4n$, где l_i - размеры бакетов, на которые хэш функция делит входной вектор. Записываем получившуюся функцию в $first\_level\_hash$

Затем для каждого бакета l_i строим naive perfect hash function - генерируем случайную функцию $f(x) = ((a*x+b) \mod prime) \mod l_i^2$ с таким же распределением на a, b, пока не получим функцию без коллизий в данном бакете. Записываем ее в $hash\_functions[i]$. Создаем вектор чисел $vector$ размера l_i^2, заполняем значением $prime$. Для каждого числа $number$ из бакета присваиваем $vector[f(number)] = number$, и кладем вектор в таблицу: $table[i] = vector$.

Contains: Если хэш таблица пустая, возвращаем false.

Пусть передано число $number$, тогда $bucket := first\_level\_hash(number)$. Если вектор $table[bucket]$ пустой, возвращаем false.

Иначе возвращаем $table[bucket][hash\_functions[bucket](number)] == number$.

2 Доказательство правильности алгоритма

Если число $number$ было во входном векторе: Поскольку по построению вектор делился на непересекающиеся бакеты, и в каждом из них строилась функция без коллизий, то и в целом функция $number$ -> адрес для этого значения тоже без коллизий. Значит, хэш таблица не пустая, по адресу $table[bucket][hash\_functions[bucket](number)]$ был записан $number$ и его не перезаписали, поскольку коллизий нет. Contains вернет true.

Поскольку Contains выдает true, только если переданное число совпадает с одним из чисел в исходном векторе, то если числа не было в исходном векторе, Contains вернет false, так как иначе по адресу лежит $prime$, который больше переданного number

3 Временная сложность — асимптотика

Initialize:

Создание вектора векторов и вектора функций размера n - сложность $O(n)$

Генерация хэш функции первого уровня: Поскольку построенные в алгоритме хэш функции лежат в универсальном семействе, для которого вероятность коллизии меньше $1/n$ в предположении равномерного и независимого выбора входных данных, где n - количество чисел в векторе, по которому берется модуль в последней операции, то вероятность того, что $\sum l_i^2 > 4n$ меньше $1/2$. Значит, матожидание количества циклов равно 2, в каждом из которых генерируется функция за $O(1)$ и проверяется критерий остановки за $O(n)$. Общее матожидание времени работы - $O(n)$.

Генерация хэш функций второго уровня: Поскольку размеры таблиц второго уровня квадратичны по отношению количества элементов в бакетах, то уже вероятность получить perfect hash function не меньше $1/2$, матожидание циклов для каждого бакета - 2, сложность линейна по размеру каждого бакета, а сумма длин всех бакетов не больше $4n$ по построению, получаем общее матожидание времени работы $O(n)$.

Проход по всей таблице и запись *prime* или *number* тоже не больше суммы длин всех бакетов, значит $O(n)$

Contains:

Все операции элементарны, сложность $O(1)$.

Общая сложность - $O(n)$

4 Затраты памяти — асимптотика

Для вектора векторов - линейна по сумме длин всех бакетов - $O(n)$

Для вектора функций - $O(n)$

Всего $O(n)$