Теоретическая информатика II
Лекция 1: Хэш-таблицы. Префиксное дерево. Поиск в строке: алгоритм Рабина–Карпа*

Александр Охотин

13 февраля 2017 г.

Содержание

1 Хэш-таблицы

Структуры данных для представления *отображения* — нахождение элемента по ключу. Операции: найти элемент с данным ключом; вставить элемент; удалить элемент. Хочется делать это за среднее время $o(n \log n)$, где n — число элементов; желательно даже за $O(1)$.

Если ключи — небольшие числа, причём все различные, то можно хранить их в заранее выделенном массиве. Пока элемента с ключом i нет, i-я ячейка массива пустует. Такой подход редко когда работает, поскольку возможных значений ключа обычно бывает много, и памяти на все не напасёшься.

Хэш-таблица[1] (hash table): то же самое, но ключи переводятся в номера ячеек с помощью некоторой функции — *хэш-функции*. Множество ключей — U, хэш-функция $h \colon U \to \{0, \ldots, m-1\}$. Когда поступает элемент x, он размещается в ячейке с номером $h(x)$. Когда надо найти элемент x в хэш-таблице, если он там есть, то он может быть только в ячейке $h(x)$.

Что делать если $h(x) = h(x')$, и в хэш-таблицу помещаются оба этих значения? Первый способ: хранить их все в виде списка.

*Краткое содержание лекций, прочитанных студентам СПбГУ, обучающимся по программе «математика», в весеннем семестре 2016–2017 учебного года. Без посещения самих лекций в этом едва ли что-то возможно понять.

[1] Принято называть на птичьем языке, устоявшегося перевода нет. Слово hash обозначает не очень аппетитное блюдо, полученное смешиванием продуктов. Ближайшее русское соответствие — «сборная солянка».

1.1 Реализация со списками

В каждой ячейке хэш-таблицы находится список. Когда элемент x попадает в таблицу, он заносится в начало списка в ячейке $h(x)$ — стало быть, за время $O(1)$.

Время поиска в худшем случае — линейное, что происходит, если все размещаемые элементы имеют одно и то же значение хэш-функции. Но хэш-функции придуманы не для этого.

Хорошая хэш-функция: равномерно распределена, легко вычисляется. Простейшая хэш-функция для хранения чисел в таблице: остаток от деления числа на m, при этом следует взять простое m.

Предполагая хэш-функцию хорошей, можно оценить среднее время поиска. Вводится понятие *коэффициента заполнения* (load factor) таблицы: $\alpha = \frac{n}{m}$, где n — число хранимых в ней элементов. То есть, α — это среднее число элементов в списке.

Лемма 1. *Среднее время поиска элемента, если его в таблице нет* — $\Theta(1+\alpha)$,

Доказательство. Потому что он с равной вероятностью принадлежит любому из списков, а средняя длина списка — α. □

Среднее время поиска элемента x, если он есть в таблице, вычисляется иначе. Чем больше элементов в некотором списке, тем вероятнее, что данный элемент находится именно в этом списке.

Лемма 2. *Среднее время поиска элемента, если он есть в таблице* — $\Theta(1+\alpha)$.

Доказательство. В своём списке искомый элемент x находится после всех элементов с тем же ключом, добавленных после x. Если $x = x_i$, где $x_1, \ldots, x_n$ — вообще все элементы хэш-таблицы в порядке их вставки, то среди элементов $x_{i+1}, \ldots, x_n$, все те, ключ которых совпадает с x, придётся просмотреть в списке. Ключ каждого элемента x_j совпадает с ключом x с вероятностью $1/m$. Отсюда — в среднем $1 + (n-i)/m$ операций. Далее, математическое ожидание числа операций вычисляется, как среднее по i.

$$\frac{1}{n}\sum_{i=1}^{n}\left(1+\frac{n-i}{m}\right) = 1 + \frac{1}{mn}\sum_{i=1}^{n}(n-i) = 1 + \frac{1}{mn}\left(n^2 - \frac{n(n+1)}{2}\right) = 1 + \frac{n-1}{2m} = \Theta(1+\alpha)$$

□

Если использовать хэш-таблицу размера, примерно равного ожидаемому числу элементов, то получится поиск за время $O(1)$.

1.2 Открытая адресация

Второй способ — «открытая адресация» (open addressing). В каждой ячейке хэш-таблицы хранится один элемент. Если при попытке разместить элемент x выясняется, что в ячейке $h(x)$ уже находится другой элемент с тем же значением хэш-функции, то x размещается в какой-то другой ячейке, номер которой можно определить, зная x. В простейшем случае находится минимальное число i, для которого ячейка $h(x)+i$ пустует, и в ней и размещается элемент x. А потом, когда потребуется разместить элемент y, которому положено место $h(y) = h(x) + i$, его придётся размещать не на своём месте. Процедура поиска должна учитывать эту особенность, и процедура стирания тоже. (рассказано на лекции) Достоинство: последовательно расположенные элементы чаще всего можно быстрее просмотреть (кэш-память, и т.д.). Недостаток: будут образовываться пробки. Есть современные исследования о том, что работает всё-таки не так плохо.

Более сложный вариант — двойное хэширование: i-я попытка разместить элемент x будет использовать адрес $h(x) + ih'(x)$ по модулю m, где h' — вторая хэш-функция. Чтобы гарантированно вставить элемент, значение $h'(x)$ должно быть взаимно простым с m.

В общем случае — равномерно распределённая функция $h(x, i)$. *Коэффициент заполнения* $\alpha = \frac{n}{m}$, очевидно, не превосходит 1.

Лемма 3. *При открытой адресации среднее количество просмотренных элементов при безуспешном поиске не превосходит* $\frac{1}{1-\alpha}$.

Доказательство. Поиск ведётся вплоть до нахождения пустой ячейки. С какой вероятностью среди первых i просмотренных ячеек так и не встретится пустая ячейка? На первом шаге — n/m, на втором — $(n-1)/(m-1)$, и т.д. до $(n-i+1)/(m-i+1)$. Вероятность — произведение всего этого, оценивается сверху так.

$$\frac{n}{m} \cdot \frac{n-1}{m-1} \cdot \ldots \cdot \frac{n-i+1}{m-i+1} \leqslant \alpha^i$$

Это функция распределения случайной величины — числа просмотренных ячеек до появления первой пустой ячейки. Плотность её распределения — то есть, вероятность того, что будет просмотрено i ячеек — это $P(i) = \alpha^{i-1} - \alpha^i$. Математическое ожидание числа операций оценивается так.

$$\sum_{i=1}^{n} i(\alpha^{i-1} - \alpha^i) \leqslant \sum_{i=1}^{\infty} i(\alpha^{i-1} - \alpha^i) = \sum_{i=0}^{\infty} \alpha^i = \frac{1}{1-\alpha}$$

□

Лемма 4. *При открытой адресации среднее количество просмотренных элементов при успешном поиске не превосходит* $\frac{1}{\alpha} \ln \frac{1}{1-\alpha}$.

Доказательство. Пусть ведётся поиск элемента x, который есть в таблице. При этом потребуется просмотреть ровно столько ячеек, сколько было просмотрено, когда этот элемент вставлялся. Пусть в тот момент в таблице было i элементов, и x стал $(i+1)$-м. Тогда коэффициент заполнения в тот момент равнялся $\frac{i}{m}$, и в среднем требовалось просмотреть $\frac{1}{1-i/m} = \frac{m}{m-i}$ ячеек (по предыдущей лемме).

Поскольку элемент x с совпадает с каждым i-м из имеющихся элементов с вероятностью $\frac{1}{n}$, среднее время просмотра оценивается так.

$$\sum_{i=0}^{n-1} \frac{1}{n} \cdot \frac{m}{m-i} = \frac{m}{n} \sum_{i=0}^{n-1} \frac{m}{m-i} = \frac{m}{n} \sum_{k=m-n+1}^{m} \frac{1}{k} \leqslant \frac{m}{n} \int_{m-n}^{m} \frac{1}{x} dx = \frac{m}{n}(\ln m - \ln(m-n)) = \frac{1}{\alpha} \ln \frac{1}{1-\alpha}$$

□

1.3 Хэш-функции для строк

Пусть ключи в таблице — это символьные строки над алфавитом Σ. Как лучше определить хэш функцию?

Самое простое, что можно придумать: сложить коды всех символов, взять сумму по модулю размера таблицы. Недостаток: распределение кодов неравномерно! Поэтому и сумма будет распределена неравномерно, и существенная часть элементов таблицы не будет использоваться, а строки, совпадающие с точностью до порядка символов, получат одинаковые значения. Плохо.

Полиномиальное хэширование: берётся некоторое основание степени p. Пусть $w = a_1 \ldots a_\ell$ — строка. Тогда используется сумма $\sum_{i=1}^{\ell} a_i \cdot p^{\ell-i}$, взятая по модулю m.

2 Структура данных для хранения множества строк

Как вообще можно хранить множество строк?

- В виде списка.
- В виде двоичного дерева поиска, используя отношение порядка на множестве строк. Лексикографический порядок.
- Используя хэш-таблицу.
- Особая структура данных: префиксное дерево.

Префиксное дерево (prefix tree; более распространенное английское название — trie, от reTRIEval). Корневое дерево с дугами, помеченными символами алфавита. Дуги, исходящие из всякой вершины, должны быть помечены различными символами алфавита. Каждая вершина соответствует некоторой строке (которая нигде не хранится) и хранит один бит информации, определяющий, принадлежит ли эта строка хранимому множеству (вместе с битом можно хранить любое значение, сопоставленное этой строке). Корень соответствует пустой строке. Если вершина соответствует строке w, и исходящая из неё дуга помечена символом $a \in \Sigma$, то эта дуга идёт в вершину, соответствующую строке wa.

Все операции с любой данной строкой длины ℓ (поиск, вставка, удаление) выполняются за время $O(\ell)$, не зависящее от числа элементов в хранимом множестве.

Компактное префиксное дерево (compact prefix tree, radix tree) — то же самое, но дуги помечены не отдельными символами, а непустыми строками. Эти непустые строки задаются парами указателей.

3 Алгоритмы поиска в строке

Строки над алфавитом Σ, подстроки, префиксы, суффиксы.

Задача поиска в строке: дана длинная строка («текст») $w = a_1 \dots a_n$, и короткая искомая строка $x = b_1 \dots b_m$. Требуется найти все вхождения x в w в качестве подстроки, то есть, все смещения s, для которых подстрока $w_s = a_{s+1} \dots a_{s+m}$ совпадает с $b_1 \dots b_m$.

«Наивный» алгоритм: сравнивать все m символов для каждого s, время $O(mn)$. В худшем случае достигается, пример: $x = 0^{m-1}1$ и $w = 0^{2m-1}1$, всего m^2 сравнений.

Но можно искать быстрее.

3.1 Алгоритм Рабина–Карпа

Алгоритм основан на полиномиальном хэшировании степени $m-1$: сперва вычисляется значение хэш-функции для искомой строки, а затем для всех m-символьных подстрок данного текста последовательно вычисляется значение их хэш-функции. Когда значение для подстроки совпадает со значением для искомой строки, алгоритм проводит прямое сравнение символов, как в «наивном» алгоритме.

Значение хэш-функции для искомой строки: $X = \sum_{i=1}^{m} b_i \cdot p^{m-i}$ по модулю q.

Значение хэш-функции для подстроки со смещением s: $W_s = \sum_{i=1}^{m} a_{s+i} \cdot p^{m-i}$ по модулю q.

Алгоритм сперва вычисляет X, а затем последовательно вычисляет W_s для s от 0 до $n-m+1$, по следующей формуле (вся арифметика — по модулю q)

$$W_{s+1} = \sum_{i=1}^{m} a_{s+i+1} \cdot p^{m-i} = a_{s+m+1} - a_{s+1} \cdot p^{m-1} + p \cdot \sum_{i=1}^{m} a_{s+i} \cdot p^{m-i} = p \cdot W_s + a_{s+m+1} - a_{s+1} \cdot p^{m-1}$$

Рис. 1: Майкл Рабин (род. 1931) и Ричард Карп (род. 1935).

Сложность: $\Theta(m)$ на подготовку, и затем в худшем случае $O(mn)$, если хэш-функция выдаст одинаковые значения для всех подстрок. Это неизбежно, если, например, все подстроки одинаковы, то есть, $w = a^n$ и $x = a^m$.

Но в среднем случае получается время работы $\Theta(n)$.