12.2.3. Применение: поиск в огромных пространствах состояний

Хеш-таблицы всецело предназначены для ускорения поиска. Одна из прикладных областей, в которой поиск встречается повсеместно, — это компьютерные игры, и, если говорить более обобщенно, задачи планирования. Подумайте, например, о шахматной программе, разведывающей последствия различных ходов. Их последовательность можно рассматривать как пути в огромном ориентированном графе, где вершины соответствуют состояниям игры (позициям всех фигур и тому, чей сейчас ход), а ребра — ходам (переходам из одного состояния в другое). Размер этого графа — астрономический (более 10^{100} вершин), поэтому нет никакой надежды записать его явно и применить любой из наших алгоритмов графового поиска из главы 8. Более сговорчивая альтернатива — выполнить алгоритм графового поиска, такой как поиск в ширину, начиная с текущего состояния, и разведать краткосрочные последствия разных ходов до достижения ограничения по времени. Для того чтобы узнать как можно больше, важно избегать разведывания вершины более одного раза, поэтому алгоритм поиска должен отслеживать уже посещенные им вершины. Как и в нашем приложении для устранения дублирования, эта задача практически создана для хеш-таблицы. Когда алгоритм поиска достигает вершины, он отыскивает ее в хеш-таблице. Если вершина уже существует, то алгоритм ее пропускает и возвращается назад; в противном случае он вставляет вершину в хеш-таблицу и продолжает ее разведывание[1, 2].

---

[1] В игровых приложениях самый популярный алгоритм графового поиска называется поиском *A** («*A* звезда»). Алгоритм поиска *A** является целеориентированным обобщением алгоритма Дейкстры (глава 9), который добавляет в дейкстрову оценку (уравнение 9.1) ребра (*v*, *w*) эвристическую оценку стоимости, необходимой для перемещения из *w* в «целевую вершину». Например, если вы вычисляете маршруты движения из заданного исходного пункта до заданного пункта назначения *t*, то эвристической оценкой может быть расстояние по прямой из *w* в *t*.

[2] Задумайтесь о современных технологиях и поразмышляйте, где еще используются хеш-таблицы. На то, чтобы у вас появилось несколько хороших догадок, не уйдет много времени!

12.2.4. Решение тестового задания 12.2

Правильный ответ: (б). Первый шаг может быть реализован за время $O(n \log n)$ с помощью алгоритма `MergeSort` (описанного в *первой части*) либо алгоритма `HeapSort` (раздел 10.3.1)[1]. Каждая из n итераций цикла может быть реализована со временем $O(\log n)$ посредством двоичного поиска. Сложив все вместе, получим конечную границу времени выполнения $O(n \log n)$.

*12.3. Реализация: высокоуровневая идея

В этом разделе рассматриваются наиболее важные высокоуровневые идеи в реализации хеш-таблиц: хеш-функции (которые отображают ключи в позиции в массиве), коллизии (разные ключи, которые отображаются в одну и ту же позицию) и наиболее распространенные стратегии разрешения коллизий. Раздел 12.4 предлагает более подробные рекомендации по реализации хеш-таблицы.

12.3.1. Два простых решения

Хеш-таблица хранит множество S ключей (и ассоциированных с ними данных), взятых из универсума U всех возможных ключей. Например, U может быть всеми 2^{32} возможными IPv4-адресами, всеми возможными строковыми значениями длиной не более 25 символов, всеми возможными состояниями шахматной доски и так далее. Множество S может быть IP-адресами, фактически посетившими веб-страницу в течение последних 24 часов, фактическими именами ваших друзей либо состояниями шахматной доски, которые ваша программа разведала за последние пять секунд. В большинстве применений хеш-таблиц размер универсума U является астрономическим, но размер подмножества S поддается управлению.

---

[1] Более быстрая реализация невозможна. По крайней мере, с помощью управляемого сравнением алгоритма сортировки (см. сноску 2 на с. 141 в главе 10).

Одним концептуально простым способом реализовать операции Просмотреть, Вставить и Удалить является отслеживание объектов в большом массиве, с одной записью в массиве для каждого возможного ключа в U. Если U является небольшим множеством наподобие всех трехсимвольных строковых значений (скажем, для того чтобы отслеживать аэропорты по их трехбуквенным кодам), то это решение на основе массива является хорошим, при этом все операции в нем выполняются за постоянное время. Во многих приложениях, в которых универсум U чрезвычайно велик, это решение будет абсурдным и невыполнимым; мы можем реалистично рассматривать только те структуры данных, которые требуют пространства, пропорционального $|S|$ (а не $|U|$).

Вторым простым решением является хранение объектов в связном списке. Хорошей новостью является то, что пространство, используемое этим решением, пропорционально $|S|$. Плохая новость заключается в том, что время выполнения операций Просмотреть и Удалить также масштабируется линейно вместе с $|S|$ — что гораздо хуже, чем постоянно-временные операции, поддерживаемые решением на основе массива. Смысл хеш-таблицы состоит в том, чтобы достичь наилучшего из обоих миров — пространства, пропорционального $|S|$, и постоянно-временных операций (табл. 12.2).

Таблица 12.2. Хеш-таблицы сочетают в себе лучшие возможности массивов и связных списков с пространством, линейным по числу хранимых объектов, и постоянно-временными операциями. Звездочка (*) указывает на то, что граница времени выполнения соблюдается тогда и только тогда, когда хеш-таблица реализована правильно и данные не являются патологическими

Структура данных	Пространство	Типичное время выполнения операции Просмотреть
Массив	$\Theta\|U\|$	$\Theta(1)$
Связный список	$\Theta\|S\|$	$\Theta\|S\|$
Хеш-таблица	$\Theta\|S\|$	$\Theta(1)$*

12.3.2. Хеш-функции

Для достижения наилучших результатов в обоих мирах хеш-таблица имитирует простое решение на основе массива, но с длиной массива n, про-

порциональной $|S|$, а не $|U|$[1]. На данный момент вы можете думать об n как примерно равном $2|S|$.

Хеш-функция выполняет трансляцию из того, что нас действительно волнует — имен наших друзей, состояний шахматной доски и так далее — в позиции в хеш-таблице. Формально хеш-функция — это отображение из множества U всех возможных ключей в множество позиций в массиве (рис. 12.2). Позиции в хеш-таблице обычно нумеруются с 0, поэтому множество позиций в массиве равно $\{0, 1, 2, ..., n - 1\}$.

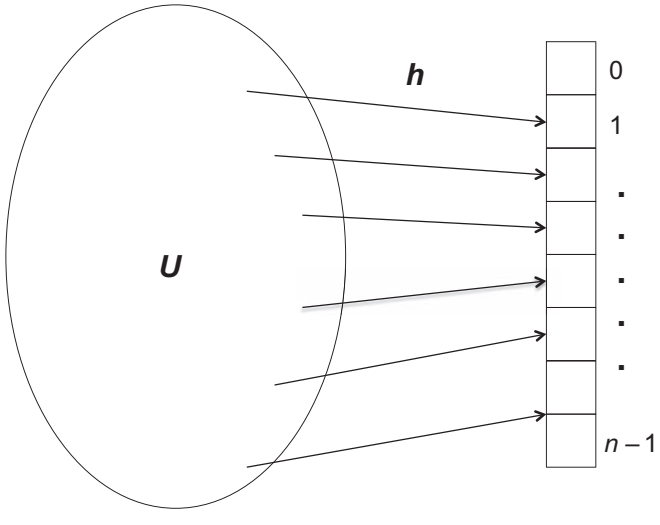

Рис. 12.2. Хеш-функция отображает каждый возможный ключ из универсума U в позицию в $\{0, 1, 2, ..., n - 1\}$. При $|U| > n$ два разных ключа будут отображаться в одинаковую позицию

ХЕШ-ФУНКЦИИ

Хеш-функция $h : U \rightarrow \{0, 1, 2, ..., n - 1\}$ присваивает каждый ключ из универсума U позиции в массиве длины n.

[1] Подождите! Разве множество S не меняется со временем? Да, это так, но совсем нетрудно периодически изменять размер массива так, чтобы его длина оставалась пропорциональной текущему размеру S; см. также раздел 12.4.2.

Хеш-функция сообщает вам, с чего начинать поиск объекта. Если вы выбираете хеш-функцию h, где h («Алиса») = 17 — в каковом случае мы говорим, что строковое значение «Алиса» хешируется в 17, — то позиция 17 массива является местом, с которого следует начинать искать номер телефона Алисы. Аналогичным образом позиция 17 является первым местом, где следует попытаться вставить номер телефона Алисы в хеш-таблицу.

12.3.3. Коллизии неизбежны

Возможно, вы заметили серьезную проблему: а что, если два разных ключа (например, «Алиса» и «Боб») хешируются в одну и ту же позицию (например, 23)? Если вы ищете номер телефона Алисы, но находите номер Боба в позиции 23 массива, как вы узнаете, имеется ли номер Алисы в хеш-таблице? Если вы пытаетесь вставить номер телефона Алисы в позицию 23, но эта позиция уже занята, куда вы ее поместите?

Когда хеш-функция h отображает два разных ключа k_1 и k_2 в одну и ту же позицию (то есть когда $h(k_1) = h(k_2)$), такая ситуация называется *коллизией*, и говорят, что ключи конфликтуют.

КОЛЛИЗИИ

Два ключа k_1 и k_2 из U конфликтуют в рамках хеш-функции h, если $h(k_1) = h(k_2)$.

Коллизии вызывают путаницу относительно того, где объект находится в хеш-таблице, и мы хотели бы, насколько это возможно, свести их к минимуму. Почему нельзя разработать сверхумную хеш-функцию без каких-либо коллизий? Потому что *коллизии неизбежны*. Причиной тому является так называемый *принцип голубей и ящиков* (или принцип Дирихле), интуитивно очевидный факт, что для каждого натурального числа n, независимо от того, как вы засовываете $n + 1$ голубей в n ячеек, всегда будет существовать ячейка по крайней мере с двумя голубями. Таким образом, всякий раз, когда число n позиций массива (ячеек) меньше размера универсума U (голубей), каждая хеш-функция (закрепление голубей за ячейками) — независимо от

того, насколько умной она является — страдает минимум от одной коллизии (см. рис. 12.2). В большинстве применений хеш-таблиц, в том числе в разделе 12.2, $|U|$ намного больше, чем n.

Коллизии еще неизбежнее, чем предполагает аргумент принципа голубей и ящиков. Причиной тому *парадокс дня рождения*, предмет следующего тестового задания.

ТЕСТОВОЕ ЗАДАНИЕ 12.3

Рассмотрим n человек со случайными днями рождения, причем каждый из 366 дней года равновероятен. (Предположим, что все n человек родились в високосный год.) Насколько большим должно быть n, прежде чем будет существовать 50 %-ный шанс, что у двух человек их день рождения будет совпадать?

а) 23

б) 57

в) 184

г) 367

(Решение и пояснение см. в разделе 12.3.7.)

Какое отношение парадокс дня рождения имеет к хешированию? Представьте себе хеш-функцию, которая независимо, равномерно и случайным образом закрепляет за каждым ключом позиции в $\{0, 1, 2, ..., n - 1\}$. Эта хеш-функция не является практически жизнеспособной (см. тестовое задание 12.5), но такие случайные функции являются золотым стандартом, с которым мы сравниваем практические хеш-функции (см. раздел 12.3.6). Из парадокса дня рождения вытекает, что даже в случае золотого стандарта мы, вероятно, начнем встречать коллизии в хеш-таблице размера n, как только будут вставлены объекты в количестве произведения малой константы на $\sqrt{n}$. Например, при n = 10 000 вставка 200 объектов может вызвать по крайней мере одну коллизию — даже если не менее 98 % позиций массива совершенно не используется!