1 Асимптотическое время выполнения

1.1. Что такое алгоритм

Предположим, вам нужно научить робота делать бутерброд с арахисовым маслом (рис. 1.1). Ваши инструкции могут быть примерно такими.

1. Открыть верхнюю левую дверцу шкафчика.
2. Взять банку с арахисовым маслом и вынуть ее из шкафчика.
3. Закрыть шкафчик.
4. Держа банку с арахисовым маслом в левой руке, взять крышку в правую руку.
5. Поворачивать правую руку против часовой стрелки, пока крышка не откроется.
6. И так далее...

Это программа: вы описали каждый шаг, который компьютер должен выполнить, и указали всю информацию, которая требуется компьютеру для выполнения каждого шага. А теперь представьте, что вы объясняете

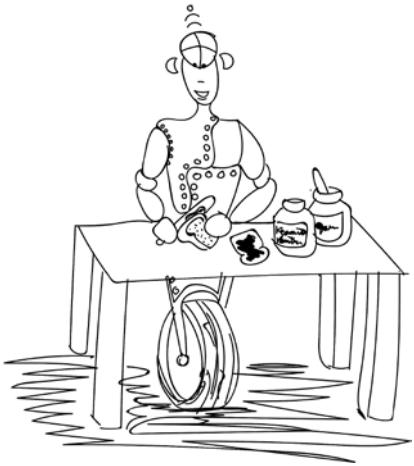

Рис. 1.1. Робот — изготовитель бутербродов

человеку, как сделать бутерброд с арахисовым маслом. Ваши инструкции будут, скорее всего, такими.

1. Достаньте арахисовое масло, джем и хлеб.
2. Намажьте ножом арахисовое масло на один ломтик хлеба.
3. Намажьте ложкой джем на второй ломтик хлеба.
4. Сложите два ломтика вместе. Приятного аппетита!

Это алгоритм: процесс, которому нужно следовать для получения желаемого результата (в данном случае бутерброда с арахисовым маслом и джемом). Обратите внимание, что алгоритм более абстрактный, чем программа. Программа сообщает роботу, откуда именно нужно взять предметы на конкретной кухне, с точным указанием всех необходимых деталей. Это реализация алгоритма, которая предоставляет все важные детали,

но может быть выполнена на любом оборудовании (в данном случае — на кухне), со всеми необходимыми элементами (арахисовое масло, джем, хлеб и столовые приборы).

1.2. Почему скорость имеет значение

Современные компьютеры достаточно быстры, поэтому во многих случаях скорость алгоритма не особенно важна. Когда я нажимаю кнопку и компьютер реагирует за 1/25 секунды, а не за 1/100 секунды, эта разница для меня не имеет значения — с моей точки зрения, компьютер в обоих случаях реагирует мгновенно.

Но во многих приложениях скорость все еще важна, например, при работе с большим количеством объектов. Предположим, что у вас есть список из миллиона элементов, который необходимо отсортировать. Эффективная сортировка занимает одну секунду, а неэффективная может длиться несколько недель. Возможно, пользователь не захочет ждать, пока она закончится.

Мы часто считаем задачу неразрешимой, если не существует известного способа ее решения за разумные сроки, где «разумность» зависит от различных реальных факторов. Например, безопасность шифрования данных часто зависит от сложности разложения на множители (факторизации) больших чисел. Если я отправляю вам зашифрованное сообщение, содержимое которого нужно хранить в секрете в течение недели, то для меня не имеет значения, что злоумышленник перехватит это сообщение и расшифрует его через три года. Задача

не является неразрешимой — просто наш любитель подслушивать не знает, как решить ее достаточно быстро, чтобы решение было полезным.

Важным навыком в программировании является умение оптимизировать только те части программы, которые необходимо оптимизировать. Если пользовательский интерфейс работает на одну тысячную секунды медленнее, чем мог бы, это никого не волнует — в данном случае мы предпочли бы незаметному увеличению скорости удобочитаемую программу. А вот код, расположенный внутри цикла, который может выполняться миллионы раз, должен быть написан максимально эффективно.

1.3. Когда секунды (не) считаются

Рассмотрим алгоритм приготовления бутербродов из раздела 1.1. Поскольку мы хотим, чтобы этот алгоритм можно было использовать для любого количества различных роботов — изготовителей бутербродов, мы не хотим измерять количество секунд, затрачиваемое на выполнение алгоритма, поскольку для разных роботов оно будет различаться. Один робот может дольше открывать шкафчик, но быстрее вскрывать банку с арахисовым маслом, а другой — наоборот.

Вместо того чтобы измерять фактическую скорость выполнения каждого шага, которая будет различаться для разных роботов и кухонь, лучше подсчитать (и минимизировать) количество шагов. Например, алгоритм, который требует, чтобы робот сразу брал и нож и ложку, эффективнее, чем алгоритм, в котором робот открывает

ящик со столовыми приборами, берет нож, закрывает ящик, кладет нож на стол, снова открывает ящик, достает ложку и т. д.

Измерение времени: алгоритм или программа?

Напомню, что алгоритмы являются более обобщенными, чем программы. Для нашего алгоритма приготовления бутербродов мы хотим посчитать число шагов. Если бы мы действительно использовали конкретного робота — изготовителя бутербродов, то нас бы больше интересовало точное время, которое требуется для изготовления каждого бутерброда.

Следует признать, что не все шаги потребуют одинакового времени выполнения; скорее всего, взять нож — быстрее, чем намазать арахисовое масло на хлеб. Поэтому мы хотим не столько узнать точное количество шагов, сколько иметь представление о том, сколько шагов потребуется в зависимости от размера входных данных. В случае с нашим роботом время, необходимое для приготовления бутерброда, при увеличении количества бутербродов не увеличивается (при условии, что нам хватит джема).

Два компьютера могут выполнять алгоритм с разной скоростью. Это зависит от их тактовой частоты, объема доступной памяти, количества тактовых циклов, требуемого для выполнения каждой инструкции, и т. д. Однако обоим компьютерам, как правило, требуется приблизительно одинаковое число инструкций, и мы можем измерить скорость, с которой количество требуемых инструкций увеличивается в зависимости от размера задачи (рис. 1.2). Например, при сортировке

массива чисел, если увеличить его размер в тысячу раз, один алгоритм может потребовать в тысячу раз больше команд, а второй — в миллион раз больше[1].

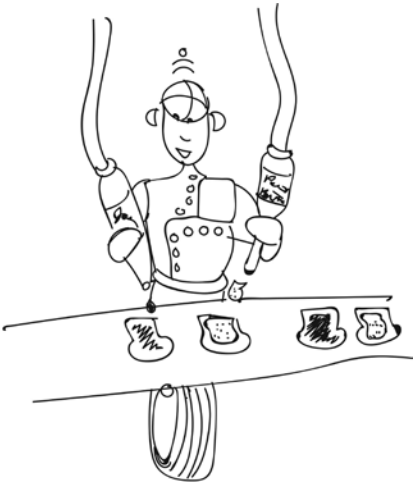

Рис. 1.2. Более эффективный робот — изготовитель бутербродов

Часто мы хотим измерить скорость алгоритма несколькими способами. В жизненно важных ситуациях — например, при запуске двигателей на зонде, который приземляется на Марсе, — мы хотим знать время выполнения при самом неблагоприятном раскладе. Мы можем выбрать алгоритм, который в среднем работает немного медленнее, но зато гарантирует, что его выполнение никогда не займет больше времени, чем мы считаем приемлемым (и наш зонд не разобьется вместо того, чтобы приземлиться). Для более повседневных сценариев мы

[1] Конкретные примеры см. в главе 8.

можем смириться со случайными всплесками времени выполнения, если при этом среднее время не растет; например, в видеоигре мы бы предпочли генерировать результаты в среднем быстрее, смирившись с необходимостью время от времени прерывать длительные вычисления. А бывают случаи, когда мы хотели бы знать наилучшую производительность. Однако в большинстве ситуаций мы просто рассчитываем наихудшее время выполнения, которое все равно часто совпадает со средним временем выполнения.

1.4. Как мы описываем скорость

Предположим, у нас есть два списка целых чисел, которые мы хотим отсортировать: {1, 2, 3, 4, 5, 6, 7, 8} и {3, 5, 4, 1, 2}. Сортировка какого списка займет меньше времени?

Ответ на вопрос: неизвестно. Для одного алгоритма сортировки тот факт, что первый список уже отсортирован, позволит почти сразу завершить работу. Для другого алгоритма решающим фактором может быть то, что второй список короче. Но обратите внимание, что оба свойства входных данных — размер списка и последовательность расположения чисел — влияют на количество шагов, необходимых для сортировки.

Если некоторое свойство, например «отсортированность», изменяет время выполнения конкретного алгоритма только на постоянную величину, мы можем просто его игнорировать, поскольку его влияние незаметно по сравнению с влиянием размера задачи. Например, робот может делать бутерброды с виноградным или

малиновым джемом, но банка с малиновым джемом открывается немного дольше. Если робот делает миллион бутербродов, то влияние выбора джема на время приготовления бутерброда незаметно на фоне количества бутербродов. Поэтому мы можем его игнорировать и просто сказать, что приготовление миллиона бутербродов занимает примерно в миллион раз больше времени, чем приготовление одного бутерброда.

С точки зрения математики мы вычисляем *асимптотическое* время выполнения алгоритма, то есть скорость увеличения времени выполнения в зависимости от размера входных данных. Нашему роботу, который делает бутерброды, требуется некоторое постоянное время c для приготовления одного бутерброда и в n раз больше времени, то есть cn, чтобы сделать n бутербродов. Мы отбрасываем константу и говорим, что наш алгоритм приготовления бутербродов занимает время $O(n)$ (произносится как «O большое от n» или просто «O от n»). Это означает, что время выполнения в худшем случае пропорционально количеству бутербродов, которое будет сделано. Нас интересует не точное количество необходимых шагов, а скорость, с которой это число увеличивается по мере роста размера задачи (в данном случае — количества бутербродов).

1.5. Скорость типичных алгоритмов

Сколько бы бутербродов ни делал наш робот, это не увеличивает время, необходимое для изготовления одного бутерброда. Это *линейный* алгоритм — общее время выполнения пропорционально количеству обрабатываемых элементов. Для большинства задач это лучшее, чего

можно добиться[1]. Типичный пример линейного алгоритма в программировании — чтение списка элементов и выполнение некоторой задачи для каждого элемента списка: время, затрачиваемое на обработку каждого элемента, не зависит от других элементов. Есть цикл, который выполняет постоянный объем работы и осуществляется один раз для каждого из n элементов, поэтому все вместе занимает $O(n)$ времени.

Но чаще количество элементов списка влияет на объем работы, которую необходимо выполнить для отдельного элемента. Алгоритм сортировки может обрабатывать каждый элемент списка, разделяя список на два меньших списка, и повторять это до тех пор, пока все элементы не окажутся в своем собственном списке. На каждой итерации алгоритм выполняет $O(n)$ операций и требует $O(\lg n)$ итераций, что в общей сложности составляет $O(n) \times O(\lg n) = O(n \lg n)$ времени.

Математическое предупреждение

Логарифм описывает, в какую степень необходимо возвести число, указанное в основании, чтобы получить желаемое значение. Например, $\log_{10} 1000 = 3$, так как $10^3 = 1000$.

В компьютерах мы часто делим на 2, поэтому обычно используются логарифмы по основанию 2. Сокращенно $\log_2 n$ обозначается как $\lg n$.[2] Таким образом, $\lg 1 = 0$, $\lg 2 = 1$, $\lg 4 = 2$, $\lg 8 = 3$ и т. д.

---

[1] Поскольку n — это размер входных данных, то в общем случае только для их чтения требуется время $O(n)$.

[2] Строго говоря, lg — это логарифм по основанию 10, но часто именно в Computer Science принимают, что это логарифм по основанию 2. — *Примеч. науч. ред.*