



Шилов Николай Вячеславович

Кандидат физико-математических наук, старший научный сотрудник Института систем информатики им. А.П. Ершова СО РАН, специалист в области программных логик, семантики и верификации программ. Автор более 100 публикаций, среди которых 8 научно-популярных статей.



Шилова Светлана Олеговна

Инженер Института систем информатики им. А.П. Ершова СО РАН. Имеет 12 научных публикаций по математическому моделированию верификации, а также 4 научно-популярные статьи.

Кирпичи и динамическое программирование

Говорят, что Галилео Галилей открыл закон равноускоренного движения при свободном падении тел опытным путём, сбрасывая разные предметы со знаменитой Падающей башни в итальянском городе Пиза. Но вот недавно наш коллега по институту за чашкой чая предложил следующую головоломку о бросании кирпичей с высокой башни.

Просто так кирпич никому на голову не упадёт.

1. Пусть марка кирпичей по прочности – это максимальная высота h (в метрах), при падении с которой кирпич не разбивается (но при падении с высоты $(h+1)$ метров разбивается). Надо определить марку кирпичей опытным путём, сбрасывая их с разных уровней башни высотой в сто метров. Считается, что: уровни в башне следуют через каждый метр; если кирпич не разбивается и при падении с 100-го



уровня, марку кирпича принимают за 100; прочность кирпича не изменится, если он не разбился при падении. Сколько бросаний вам для этого достаточно, если у вас только два кирпича? Какое число бросаний является минимальным для определения марки кирпича?

Для решения головоломки нужно придумать алгоритм (т. е. определить, с каких уровней и в какой последовательности бросать) и доказать, что этот алгоритм оптимален (меньшим числом бросаний обойтись нельзя). При высоте башни до 10 метров задачу можно решить подбором, но нас интересует более общий случай. Представим головоломку в следующем обобщённом варианте.

2. Найти минимальное чис-

ло бросаний N_H , достаточное для определения марки кирпичей в высокой башне, если у вас есть только два кирпича и бросать можно только с H уровней.



Оптимальный метод и рекурсия

Возьмёмся за задачу 2. Оптимальный по количеству бросаний метод её решения начинается с некоторого шага: «сбросить первый кирпич с уровня $h...$ ». Предположим, мы как-то выбрали h ; тогда верно равенство

$$N_H = 1 + \max\{(h - 1), N_{(H-h)}\},$$

правая часть которого имеет следующий смысл:

1) «плюс 1» соответствует одному (первому) бросанию;

2) $(h - 1)$ соответствует случаю, когда после первого бросания кирпич разбился и у нас остался единственный кирпич, который надо последовательно бросать с уровней 1, 2, ... $(h - 1)$;

3) $N_{(H-h)}$ соответствует случаю, когда после первого бросания кирпич остался цел, и мы вернулись к исходной задаче для $H - h$ уровней;

4) «max» выбирает худший из случаев, описанных в 2 и 3 – мало



ли какой кирпич нам попадётся!

Как оптимальным образом выбрать h , мы пока не знаем. Но нас интересует *оптимальный* метод, дающий *минимальное* число бросаний, и поэтому мы можем выписать следующее равенство:

$$N_H = \min_{1 \leq h \leq H} (1 + \max\{(h - 1), N_{(H-h)}\}) = 1 + \min_{1 \leq h \leq H} \max\{(h - 1), N_{(H-h)}\}.$$

Кроме того, можно выписать ещё одно очевидное равенство: $N_0 = 0$.

Заметим, что последовательность натуральных чисел $N_0, N_1, \dots, N_H, \dots$, которые удовлетворяют этим двум равенствам, единственна. Действительно, N_0 определено, N_1 определяется только через N_0 , N_2 , — только

$$\begin{cases} N(0) = 0, \\ N(H) = 1 + \min_{1 \leq h \leq H} \max\{(h-1) N(H-h)\}. \end{cases}$$

Аргументы этой функции — натуральные числа, значения — члены последовательности $N_0, N_1, \dots, N_{(H-1)}, \dots$, причём $N(i) = N_i$. Как следует из рассуждения о единственности последовательности чисел $N_0, N_1, \dots, N_H, \dots$, данная система уравнений имеет единственное решение. Решая нашу задачу, мы оптимизировали значения функции N ; эту функцию называют *целевой* для нашей задачи.

Заметим, что эта система уравнений является определением рекурсивной функции, т. е. рекурсивным алгоритмом. И получили мы его для решения нашей задачи, пользуясь лишь рассуждениями об оптимальности.

Для тех, кому термин «рекурсивная функция» не совсем понятен, поясним. Если в определении функции встречается обращение этой функции к самой себе, то такая

через N_0 и N_1 , N_H — только через $N_0, N_1, \dots, N_{(H-1)}$. Такие последовательности называются *рекурсивными* (рекурсивно вычислимыми).

Очевидно, что мы имеем систему уравнений, определяющих функцию¹, по которой можно вычислить число бросаний.

функция называется рекурсивной. Другими словами, рекурсивная функция — это функция, значения которой в одних точках определяются через значения в других точках. Например, функция факториал $\text{Fact}(n) = n!$ может быть определена рекурсивно следующим образом: $\text{Fact}(n) = \text{if } (n=0) \text{ then } 1 \text{ else } n \cdot \text{Fact}(n-1)$. А функция $\text{Fib}(n)$, вычисляющая n -ое число Фибоначчи, определяется рекурсивно следующим образом:

$\text{Fib}(n) = \text{if } (n=0 \text{ or } n=1) \\ \text{then } 1 \text{ else } (\text{Fib}(n-2) + \text{Fib}(n-1)).$

Кстати, само слово «рекурсия» — латинского происхождения, оно происходит от *recursio* и означает «возвращение». Как видно из приведённых примеров, такое исходное значение термина вполне согласуется с его использованием в математике и программировании.

Причём тут динамическое программирование?

Задачи, в которых нужно найти оптимальное решение, называются *оптимизационными*. Метод «*динамического программирования*» был разработан как раз для решения оптимизационных задач. Поиск оптимального решения рекурсивно сводится к поэтапному поиску оптимальных решений; план или про-

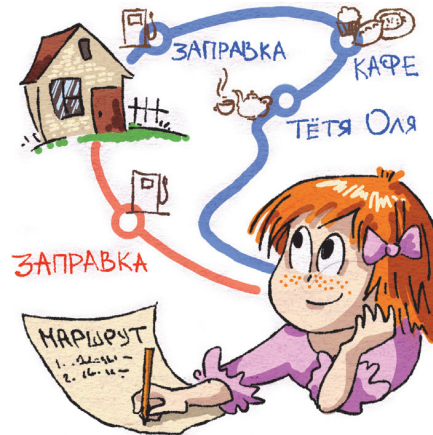
грамма действий, приводящая к оптимальному результату и состоящая из нескольких шагов, остаётся оптимальной на любом своём шаге. Именно так мы и решали нашу задачу. Рекурсивный метод решения оптимизационных задач и есть (исторически) первый лик динамического программирования. В такой

¹Данная система называется системой функциональных уравнений, то есть уравнений, которые выражают связь между значением функции (или функций) в одной точке с её значениями в других точках.

форме динамическое программирование было введено Ричардом Беллманом в 1950-х годах. Он же предложил сам термин «динамическое программирование». Но этот термин имеет мало общего с программированием для ЭВМ, которое только зарождалось 1950-е годы. Существительное «программирование» тогда имело значение «планирование» (например, в названии «линейное программирование»). А прилагательное «динамическое» указывает на смену состояний, – например, в названиях «динамические системы» или «динамическая логика».

Имя Р. Беллмана сохранено в названиях «принцип оптимальности Беллмана» и «уравнение Беллмана». Этот принцип мы уже сформулировали: план, программа действий, приводящая к оптимальному результату и состоящая из нескольких шагов, остаётся оптимальной на любом своём шаге (в частности, после первого шага). Уравнение Беллмана –

это рекурсивное уравнение для целевой функции, выражающее её значение после исполнения очередного шага через её значения перед исполнением этого шага. Например, в случае задачи об оптимальном числе бросаний двух кирпичей это уже знакомое нам выражение $N(H) = 1 + \min_{1 \leq h \leq H} \max\{(h - 1), N(H - h)\}$.



От рекурсии к итерации

Давайте попробуем вычислить одно значение функции N , например $N(4)$, в полном соответствии с её рекурсивным

определением в леворекурсивном порядке (первым всегда вычисляется самый левый самый внутренний вызов):

$$\begin{aligned} N(4) &= 1 + \min_{1 \leq h \leq 4} \max\{(h - 1), N(4 - h)\} = \\ &= 1 + \min\{\max\{0, N(3)\}, \max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} = \\ &= 1 + \min\{\max\{0, 1 + \min\{\max\{0, N(2)\}, \max\{1, N(1)\}, \max\{2, N(0)\}\}, \\ &\quad \max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} = \\ &= 1 + \min\{\max\{0, 1 + \min\{\max\{0, 1 + \min\{\max\{0, N(1)\}, \max\{1, N(0)\}\}, \\ &\quad \max\{1, N(1)\}, \max\{2, N(0)\}\}, \\ &\quad \max\{1, N(2)\}, \max\{2, N(1)\}, \max\{3, N(0)\}\} = \dots = 3. \end{aligned}$$

Уже на представленной части вычислений можно заметить, что многократно приходится вычислять значения функции N при одних и тех же значениях аргументов; в частности, $N(2)$ и $N(1)$ вычислялись многократно.

Возможный вариант оптимизации состоит в том, чтобы вычислять

только новые¹ вызовы функции N , сохранять их значения в памяти и значения повторных² вызовов функции N брать из памяти. Эта техника хорошо известна в функциональном программировании как *мемоизация* (memoization).

¹ Для тех значений аргументов, для которых ранее функция не вычислялась.

² Для тех значений аргументов, для которых функция уже была вычислена ранее.

Однако часто бывает так, что можно «предсказать» конечное множество всех нужных точек, которые могут возникнуть во время вычислений целевой функции в интересующей нас точке. Тогда можно применить метод «*восходящего динамического программирования*», состоящий в следующем.

1. Вычислить и сохранить множество значений функции во всех нужных «тривиальных» точках. Например, в случае функции N интересующее нас значение – это $N(H)$, а значение в тривиальной точке – это $N(0)$, его можно сохранить в ячейке $N[0]$ массива `int array N[0..H]`.

2. Пополнить множество уже вычисленных значений функции значениями в новых «промежуточных» точках, которые можно вычислить непосредственно по уже сохранённым

значениям функции. Например, в случае функции N , если множество уже вычисленных значений N в точках $0, \dots, K$ ($0 \leq K < H$) записано в элементах массива $N[0], \dots, N[K]$, то множество вычисленных значений функции N можно пополнить её значением в точке $(K+1)$ посредством присваивания $N[K+1] := 1 + \min_{1 \leq k \leq K} \max\{(k-1), N[K-k]\}$.

3. Повторять шаг 2, пока не будет вычислено значение целевой функции в нужной «целевой» точке.

Заметим, что метод восходящего динамического программирования свёл рекурсию к итерации – мы вычисляем нужную нам функцию, двигаясь «вперёд» до необходимой точки, а не «назад», используя значения функции в предыдущих точках (как принято в рекурсивных формулах).

Вернёмся к нашим кирпичам

Давайте всё-таки вернёмся к нашим кирпичам и решим вторую задачу, пользуясь лишь знаниями из школьной программы. Для начала

вычислим несколько первых членов последовательности

$$N_H = 1 + \min_{1 \leq h \leq H} \max\{(h-1), N_{(H-h)}\}:$$

$$N_0 = 0$$

$$N_1 = 1 + N_0 = 1$$

$$N_2 = 1 + \min_{1 \leq h \leq 2} \max\{(h-1), N_{(2-h)}\} = 1 + \min\{\max\{0, N_1\}, \max\{1, N_0\}\} = 2$$

$$N_3 = 1 + \min_{1 \leq h \leq 3} \max\{(h-1), N_{(3-h)}\} = \\ = 1 + \min\{\max\{0, N_2\}, \max\{1, N_1\}, \max\{2, N_0\}\} = 2$$

$$N_4 = 1 + \min_{1 \leq h \leq 4} \max\{(h-1), N_{(4-h)}\} = \\ = 1 + \min\{\max\{0, N_3\}, \max\{1, N_2\}, \max\{2, N_1\}, \max\{3, N_0\}\} = 3$$

$$N_5 = 1 + \min_{1 \leq h \leq 5} \max\{(h-1), N_{(5-h)}\} = \\ = 1 + \min\{\max\{0, N_4\}, \max\{1, N_3\}, \max\{2, N_2\}, \max\{3, N_1\}, \max\{4, N_0\}\} = 3$$

$$N_6 = 1 + \min_{1 \leq h \leq 6} \max\{(h-1), N_{(6-h)}\} = \\ = 1 + \min\{\max\{0, N_5\}, \max\{1, N_4\}, \max\{2, N_3\}, \max\{3, N_2\}, \max\{4, N_1\}, \max\{5, N_0\}\} = 3$$

$$N_7 = 1 + \min_{1 \leq h \leq 7} \max\{(h-1), N_{(7-h)}\} = 1 + \\ + \min\{\max\{0, N_6\}, \max\{1, N_5\}, \max\{2, N_4\}, \max\{3, N_3\}, \max\{4, N_2\}, \max\{5, N_1\}, \max\{6, N_0\}\} = 4$$

...

Как видим, последовательность N_i состоит из нуля, одной единицы,

двух двоек, трёх троек... Проверим¹, что эта симпатичная закономерность

¹ Фактически здесь приведён набросок доказательства методом математической индукции.

является законом для этой последовательности. Выберем номер k такой, что для $i = 1, \dots, k$ последовательность N_i состоит из последовательных натуральных чисел $1, \dots, n$, причём

$$\begin{aligned} N_{k-1} &= 1 + \min_{1 \leq h \leq k-1} \max\{(h-1), N_{(k-1-h)}\} = \\ &= 1 + \min\{\max\{0, N_{k-2}\}, \max\{1, N_{k-3}\}, \dots, \max\{n-2, N_{k-n}\}, \\ &\quad \max\{n-1, N_{k-(n+1)}\}, \dots, \max\{k-3, N_1\}, \max\{k-2, N_0\}\} = \\ &= 1 + \min\{N_{k-2}, N_{k-3}, \dots, n-2, \dots, k-3, k-2\} = 1 + \min\{n-1, \dots, n-1, n-2, \\ &\quad n-1, n, n+1, \dots, k-2\} = n-1. \end{aligned}$$

У функции $\min$ в этом выражении $(k-1)$ аргумент, и среди этих аргументов самый маленький —

$$\begin{aligned} N_k &= 1 + \min_{1 \leq h \leq k} \max\{(h-1), N_{(k-h)}\} = \\ &= 1 + \min\{\max\{0, N_{k-1}\}, \max\{1, N_{k-2}\}, \dots, \max\{n-2, N_{k-(n-1)}\}, \max\{n-1, N_{k-n}\}, \\ &\quad \max\{n, N_{k-(n+1)}\}, \dots, \max\{k-2, N_1\}, \max\{k-1, N_0\}\} = n. \end{aligned}$$

Здесь у функции $\min$ уже k аргументов, и n из них (выделенные жирным шрифтом) равны $(n-1)$; остальные больше, чем $(n-1)$.

Переходя к вычислению N_{k+1} и далее до $N_{k+(n-1)}$, заметим, что число аргументов функции $\min$ с каждым шагом возрастает на 1, а число аргументов, на которых достигается минимум (равный $(n-1)$) — на 1 сокращается. Через n шагов, при вычислении N_{k+n} , значением функции $\min$ станет n . Таким образом, n членов последовательности N с номерами $k, \dots, k+(n-1)$ равны n , а $N_{k+n} = n+1$.

Мы убедились, что последовательность N_H состоит из нуля, одной единицы, 2-х двоек, ... n экземпляров числа n и т. д.

Но как по номеру H определить значение N_H ?

Давайте сначала возьмём произвольную «граничную» точку H и пусть $N_H = n$, $N_{H+1} = n+1$. По свойству этой последовательности получаем $H = 1 + 2 + 3 + \dots + n = (1+n) \cdot n / 2$.

каждое значение i повторяется i раз.

Начнём со случая, когда $N_k = n$, а

$$N_{k-1} = n-1.$$

Имеем:

$$\max\{n-2, N_{k-n}\} = \max\{n-2, n-2\}.$$

Заметим, кроме того, что лишь один аргумент $\min$ равен $n-2$. Далее,

Преобразуя уравнение, получим:

$$n^2 + n - 2 \times H = 0.$$



Его целое положительное решение:

$$n = (\sqrt{D} - 1) / 2, \text{ где } D = 1 + 8 \times H.$$

(Понятно, что не для любого натурального H дискриминант нашего уравнения будет полным квадратом, но для «граничных» точек нашей последовательности будет обязательно.)

Для произвольного номера H значение N_H — это минимальное натуральное

ральное число n такое, что $H \leq 1 + 2 + 3 + \dots + n = (1 + n) \times n/2$. Таким образом, мы приходим к квадратному неравенству

$$n^2 + n - 2 \times H \geq 0,$$

из которого следует (учитывая монотонность левой части при $n > 0$), что N_H – минимальное целое число, которое не меньше $(\sqrt{D} - 1)/2$, где $D = 1 + 8 \times H$.

Таким образом, вторая задача полностью решена, причём так, что значения функции $N(H)$ вычислимы по формуле, соответствующий алгоритм вычисления значений этой функции не содержит ни циклов, ни рекурсии, а динамическое программирование (в рекурсивной форме) было использовано только для обоснования этой формулы. Для решения первой задачи о башне высотой 100 метров остаётся только подставить $H=100$ в полученную формулу.

Благодарности. В основу работы положен мастер-класс «Заметки о парадигмах программирования», проведённый одним из авторов 5 ноября 2010 г. в рамках школы-семинара «Искусство программирования» на очном туре XI Открытой Всесибирской олимпиады по программированию им. И.В. Поттосина, и

одноимённый цикл лекций на Зимней школе НГУ-Parallels «Теория и практика программирования», которая прошла с 31 января по 5 февраля 2011 г. в Новосибирском государственном университете. Несколько позже Теодор Ясонович Заркуа, профессор Грузинского университета им. Святого Андрея Первозванного Патриаршества Грузии, сообщил нам, что аналогичная задача используется в олимпиадном программировании (см., например, <http://acm.timus.ru/problem.aspx?space=1&num=1223>). Авторы выражают ему свою признательность, а также благодарность Евгению Викторовичу Бодину, регулярно снабжающему его новыми головоломками за чаепитием.



Юмор Юмор Юмор Юмор Юмор Юмор

- Почему ваши дети всё время ссорятся?
- Конфликт версий, – отвечает программист

Первый урок английского языка во 2 классе. Учительница:

- Кто из вас, дети, знает английский алфавит?
- Антон (сын программиста):
- Я!
- Тогда назови мне его по порядку.
- Q, W, E, R, T, Y...

Получив двойку по информатике, ученик не растерялся – удалил страницу и переформатировал дневник.