

Одномерное динамическое программирование: количество способов

Информатика. Язык программирования Python в курсе информатики. Все классы.

Задача о кузнечике

Рассмотрим следующую задачу. На числовой прямой сидит кузнечик, который может прыгать вправо на одну или на две единицы. Первоначально кузнечик находится в точке с координатой 0. Определите количество различных маршрутов кузнечика, приводящих его в точку с координатой n.

Обозначим количество маршрутов кузнечика, ведущих в точку с координатой n, как F(n). Теперь научимся вычислять функцию F(n). Прежде всего заметим, что F(0)=1 (это вырожденный случай, существует ровно один маршрут из точки 0 в точку 0 — он не содержит ни одного прыжка), F(1)=1, F(2)=2. Как вычислить F(n)? В точку n кузнечик может попасть двумя способами — из точки n−2 при помощи прыжка длиной 2 и из точки n−1 прыжком длины 1. То есть число способов попасть в точку nравно сумме числа способов попасть в точку n−1 и n−2, что позволяет выписать рекуррентное соотношение: F(n)=F(n−2)+F(n−1), верное для всех n⩾2.

Рекурсивное решение

Теперь мы можем оформить решение этой задачи в виде рекурсивной функции:

ПРИМЕР НА ЯЗЫКЕ PYTHON

# неэффективное решение  
def F(n):  
   if n < 2:  
       return 1  
   else:  
       return F(n - 1) + F(n - 2)

ПРИМЕР НА ЯЗЫКЕ PASCAL

function f(n: longint): longint;  
begin  
 if n < 2 then  
   f := 1  
 else  
   f := f(n - 1) + f(n - 2);  
end;

 Но при попытке вычислить решение этой функции для уже не очень больших n, например, для n=40, окажется, что эта функция работает крайне медленно. И при этом время работы функции с увеличением n растет экспоненциально, то есть такое решение неприемлемо по сложности. Причина этого заключается в том, что при вычислении рекурсивной функции подзадачи, для которых вычисляется решение, «перекрываются». То есть для того, чтобы вычислить F(n) нам нужно вызвать F(n−1) и F(n−2). В свою очередь F(n−1) вызовет F(n−2) и F(n−3). То есть функция F(n−2) будет вызвана два раза — один раз это будет сделано при вычислении F(n−1) и один раз — при вычислении F(n−2). Значение F(n−3) будет вычислено уже три раза, а значение F(n−4) будет вычисляться уже пять раз. При увеличении глубины рекурсии количество «перекрывающихся» вызовов функций будет расти экспоненциально. То есть одна из причин неэффективности рекурсивного решения — одно и то же значение функции вычисляется несколько раз, так как оно используется для вычисления нескольких других значений функции.

Нерекурсивное решение

На самом деле несложно видеть, что значения рекурсивной функции в данном случае будут совпадать с числами Фибоначчи, так как вычисляются по тем же рекуррентным соотношениям. А для вычисления чисел Фибоначчи можно использовать цикл, а не рекурсию — следующее число Фибоначчи определяется, как сумма двух предыдущих.

ПРИМЕР НА ЯЗЫКЕ PYTHON

ПРИМЕР НА ЯЗЫКЕ PASCAL

Сложность такого решения будет O(n). Сложность вычисления уменьшается за счет того, что для каждого промежуточного iзначение F(i) вычисляется один раз и сохраняется в списке, чтобы впоследствии использовать это значение несколько раз для вычисления F(i+1)  и F(i+2).

Такой прием называется**динамическим программированием.** Динамическое программирование использует те же рекуррентные соотношения, что и рекурсивное решение, но в отличии от рекурсии в динамическом программировании значения вычисляются в цикле и сохраняются в списке. При этом заполнение списка идет от меньших значений к большим, в то время как в рекурсии — наоборот, рекурсивная функция вызывается для больших значений, а затем вызывает сама себя для меньших значений.

Модификации задачи о кузнечике

Модифицируем задачу. Пусть кузнечик прыгает на одну, две или три единицы, необходимо также вычислить количество способов попасть в точку n. В рекуррентном соотношении добавится еще одно слагаемое: F(n)=F(n−1)+F(n−2)+F(n−3). И начальные значения для вычисления теперь должны состоять из трех чисел: F(0), F(1), F(2). Решение изменится не сильно:

ПРИМЕР НА ЯЗЫКЕ PYTHON

F = [0] \* (n + 1)  
F[0] = 1  
F[1] = F[0]  
F[2] = F[1] + F[0]  
for i in range(3, n + 1):  
   F[i] = F[i - 3] + F[i — 2] + F[i — 1]

ПРИМЕР НА ЯЗЫКЕ PASCAL

var  
 F: array[0..100] of longint;  
 i, n: longint;  
begin  
 readln(n);  
 for i := 1 to n do  
   F[i] := 0;  
 F[0] := 1;  
 F[1] := F[0];  
 F[2] := F[1] + F[0];  
 for i := 3 to n do  
   F[i] := F[i - 1] + F[i - 2] + F[i - 3];  
 writeln(F[n]);    
end.

Еще раз модифицируем задачу. Пусть некоторые точки являются «запретными» для кузнечика, он не может прыгать в эти точки. «Карта» запрещенных точек задается при помощи списка Map: если Map[i] == 0 (для языка Pascal — массива Map), то в точку номер i кузнечик не может прыгать, а если Map[i] == 1, то данная точка является разрешенной для кузнечика. Как и в предыдущей задаче, необходимо найти количество маршрутов в точку n.

В данном случае также придется модифицировать вид рекуррентного соотношения: если Map[i] == 0, то F[i] = 0, то есть если точка — «запрещенная», то количество способов попасть в эту точку равно 0, так как нет ни одного допустимого маршрута, заканчивающегося в этой точке. Если же Map[i] == 1, то значение F[i] вычисляется по тем же рекуррентным соотношениям, что и ранее. Получаем следующее решение:

ПРИМЕР НА ЯЗЫКЕ PYTHON

F = [0] \* (n + 1)  
F[0] = 1  
for i in range(1, n + 1):  
   if Map[i] == 0:  
       F[i] = 0  
   else:  
       F[i] = sum(F[max(0, i — 3): i])

ПРИМЕР НА ЯЗЫКЕ PASCAL

var  
 F, Map: array[0..100] of longint;  
 i, n: longint;  
begin  
 readln(n);  
 for i := 1 to n do  
   read(Map[i]);  
 F[0] := 1;  
 F[1] := Map[1]\*F[0];  
 F[2] := Map[2]\*(F[1] + F[0]);  
 for i := 1 to n do  
     F[i] := Map[i]\*(F[i - 1] + F[i - 2] + F[i - 3]);  
 writeln(F[n]);    
end.

Здесь используется немного другой код для вычисления суммы F[i - 3] + F[i - 2] + F[i - 1] для того, чтобы крайние значения F[1] и F[2] также можно было вычислить при помощи этого кода в основном цикле, а не перед ним.



PyTurtle: Фракталы

Информатика. Язык программирования Python в курсе информатики. Все классы.

Снежинка Коха

Код функции рисования кривой Коха очень короток и прост:

def forward\_snowy(length, n):  
    if n == 0:  
        t.forward(length)  
    else:  
        forward\_snowy(length/3, n-1)  
        t.left(60)  
        forward\_snowy(length/3, n-1)  
        t.right(120)  
        forward\_snowy(length/3, n-1)  
        t.left(60)  
        forward\_snowy(length/3, n-1)

Для рисования снежинки Коха нужно нарисовать три таких кривых.

Например, таким кодом:

def snow\_ball(length):  
    global t  
    t = Turtle()  
    t.penup()  
    t.backward(300)  
    t.pendown()  
    t.speed(100)  
    t.color('blue')  
    for i in range(3):  
        forward\_snowy(length, 4)  
        t.right(120)

Кривая Леви

Код рисования кривой Леви:

def forward\_levy(length, n):  
    if n == 0:  
        t.forward(length)  
    else:  
        t.left(45)  
        forward\_levy(length/2\*\*0.5, n-1)  
        t.right(90)  
        forward\_levy(length/2\*\*0.5, n-1)  
        t.left(45)

Кривая дракона

Функция рисования кривой дракона очень похожа на функцию рисования кривую Леви, только изломы теперь чередуются — то влево, то вправо:

def forward\_dragon(length, n, kink = +1):  
    """kink должен быть или 1 или -1  
        +1 — излом влево, -1 — излом вправо"""  
    if n == 0:  
        t.forward(length)  
    else:  
        t.left(45\*kink)  
        forward\_dragon(length/2\*\*0.5, n-1, +1)  
        t.right(90\*kink)  
        forward\_dragon(length/2\*\*0.5, n-1, -1)  
        t.left(45\*kink)

Обе эти функции запускать можно примерно таким кодом:

def simple\_example(forward\_function):  
    global t  
    t = Turtle()  
    t.penup()  
    t.backward(100)  
    t.pendown()  
    t.speed(200)  
    t.color('blue')  
    forward\_function(200, 10)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
    simple\_example(forward\_dragon)

одсчет числа комбинаторных последовательностей

Информатика. Язык программирования Python в курсе информатики. Все классы.

Подсчет числа двоичных последовательностей

Рассмотрим задачи подсчета количества различных комбинаторных объектов, аналогичные рассмотренные ранее задачам о переборе всех подобных объектов.

Легко решается задача подсчета всех двоичных последовательностей длины n. Если обозначить количество таких последовательностей через F(n), то очевидное рекуррентное соотношение - F(n+1)=2F(n), так как любую последовательность длины n можно продолжить до последовательности длины n+1 двумя способами. Учитывая начальные значение F(0)=1, F(1)=2 можно вывести и общую формулу: F(n)=2n.

Теперь рассмотрим более сложную задачу - подсчет числа двоичных последовательностей, в которых нет двух единиц подряд. Задачу можно решить полным перебором, но в этом случае решение будет иметь экспоненциальную сложность. Попробуем вывести рекуррентные соотношения для количества таких последовательностей.

Проблема возникнет в том, что не всякую двоичную последовательность длины n можно продолжить двумя способами, соблюдая условие.  Все зависит от того, какой цифрой заканчивается последовательность. Если она заканчивается на 0, то ее можно продолжить как числом 0, так и числом 1, а если она заканчивается на 1, то можно продолжить только числом  0.

Будем отдельно считать F0(n) - число двоичных последовательностей не содержащих двух единиц подряд, заканчивающихся числом 0 и F1(n) - число двоичных последовательностей не содержащих двух единиц подряд, заканчивающихся числом 1.

Последовательность длины n+1, зананчивающуюся числом 0, можно получить из последовательностей длины n, заканчивающихся и на 0, и на 1. Поэтому F0(n+1)=F0(n)+F1(n). Последовательность длины n+1, заканчивающуюся числом 1, можно получить только из последовательности, заканчивающейся числом 0, поэтому F1(n+1)=F1(n).

Теперь зададим начальные значения: F0(1)=1, F1(1)=1. Для вычисления значений F для больших n воспользуемся динамическим программированием.

F = [[0] \* (n + 1) for i in range(2)]  
F[0][1] = 1  
F[1][1] = 1  
for i in range(2, n + 1):  
    F[0][i] = F[0][i - 1] + F[1][i - 1]  
    F[1][i] = F[0][i - 1]

Ответом является сумма двух значений:

print(F[0][n]+F[1][n])

Теперь подсчитаем число двоичных последовательностей длины n, не содержащих трех одинаковых символов подряд. В данном случае понадобится хранить информацию о двух последних символах последовательности. Пусть:

F0(n) - количество последовательностей длины n, заканчивающихся на один символ 0 (но не на два символа 0).

F00(n) - количество последовательностей длины n, заканчивающихся на два символа 0.

F1(n) - количество последовательностей длины n, заканчивающихся на один символ 1.

F11(n) - количество последовательностей длины n, заканчивающихся на два символа 1.

Выведем рекуррентные соотношения. Последовательности, заканчивающиеся ровно на один символ 0 (или 1) можно получить только из последовательностей, заканчивающихся на один или два символа 1 (или 0 соответственно), поэтому

F0(n+1)=F1(n)+F11(n+1), F1(n+1)=F0(n)+F0(n+1).

А последовательность, заканчивающуюся на два нуля, можно получить только из последовательностей, заканчивающихся на один нуль, поэтому  F00(n+1)=F0(n), F11(n+1)=F1(n).

Начальные значения: F0(1)=F1(1)=1, F00(1)=F11(1)=0.

Программа для вычисления значений F:

F0 = [0] \* (n + 1)  
F00 = [0] \* (n + 1)  
F1 = [0] \* (n + 1)  
F11 = [0] \* (n + 1)  
F0[1] = 1  
F1[1] = 1  
for i in range(2, n + 1):  
    F0[i] = F1[i - 1] + F11[i - 1]  
    F1[i] = F0[i - 1] + F00[i - 1]  
    F00[i] = F0[i - 1]  
    F11[i] = F1[i - 1]

Подсчет числа правильных скобочных последовательностей

Подсчитаем количество правильных скобочных последовательностей из n открывающихся и n закрывающихся скобок одного вида (эти значения называются «Числами Каталана»). Рассмотрим два способа вычисления чисел Каталана.

Рассмотрим префикс последовательности какой-то длины k (0⩽k⩽2n). Возможность продолжение этого префикса зависит от разности числа открывающихся и закрывающихся скобок на этом префиксе. Обозначим это значение («баланс последовательности») через b.

Пусть F(k,b)  - это число префиксов правильной скобочной последовательности длины k и баланса b. Тогда k-я поставленная скобка либо увеличивает баланс на 1, если это открывающаяся скобка, либо уменьшает. Итак, рекуррентное соотношение выглядит так:

F(k,b)=F(k−1,b−1)+F(k−1,b+1).

Теперь зададим граничные значения. Удобно ввести «каемку», считая, что F(k,−1)=0 и F(k,n+1)=0. Эти условия следуют из того, что баланс не может быть отрицательным, а также не может превосходить значение n (не может быть больше nоткрывающихся скобок). Поэтому мы считаем, что массив значений F окружен каемкой для b=−1 и b=n+1, на которой всегда записаны нулевые значения.

Теперь рассмотрим значения функции F при k=0. В этом случае F(0,0)=1, F(0,b)=0 при b≠0.

F = [[0] \* (n + 2) for i in range(2 \* n + 1)]  
F[0][0] = 1  
for k in range(1, 2 \* n + 1):  
     for b in range(0, n + 1):  
         F[k][b] = F[k - 1][b - 1] + F[k - 1][b + 1]

В данном случае допустимые значения для индекса k от 0 до 2n, а для индекса b - от 0 до n+1, при этом внутри цикла обходятся только значения b от 0 до n. Также мы пользуемся отрицательными индексами в языке Питон, то есть допускается обращение к элементу массива при b=−1, в этом случае также происходит обращение к каемке массива, соответствующей b=n+1, то есть одно и то же значение b служит каемкой как для b=−1, так и для b=n+1.

Ответом (n-м числом Каталана) является значение F(2n,0), так как общий баланс у последовательности должен быть равен 0.

Второй способ вычисления чисел Каталана основан непосредственно на определении правильной скобочной последовательности.

Пусть C(k) - число правильных скобочных последовательностей длины 2k. Выведем рекуррентное соотношение. Рассмотрим первую скобку в последовательности и парную ей скобку. Тогда последовательность представляется в виде «(A)B», где A и B - тоже правильные скобочные последовательности, возможно, пустые. Если длина последовательности A равна 2i, то последовательность A можно записать C(i) способами. Тогда длина последовательности B равно k−1−i и последовательность B можно записать C(k−1−i) способами.

Поскольку выбор последовательностей A и B независим, то выбрать и последовательность A, и последовательность B можно C(i)×C(k−1−i).

Значение i может быть произвольным от 0 до k−1, поэтому

C(k)=C(0)×C(k−1)+C(1)×C(k−2)+…+C(k−1)×C(0)=∑i=0k−1C(i)×C(k−1−i).

Начальное значение - C(0)=1.

*Пример программы*:

C = [0] \* (n + 1)  
C[0] = 1  
for k in range(1, n + 1):  
    for i in range(0, k):  
        C[k] += C[i] \* C[k - 1 - i]

Подсчет числа разбиений на слагаемые

Рассмотрим задачу - дано натуральное число n, необходимо подсчитать (или вывести) все возможные представления числа nна слагаемые. Разбиения, различающиеся перестановкой слагаемых, будем считать одинаковыми.

Например, число 4 можно представить в виде нескольких слагаемых так 4 = 3 + 1 = 2 + 2 = 2 + 1 + 1 = 1 + 1 + 1 + 1 (всего 5 представлений).

Для начала решим задачу перебора всех разбиений. Чтобы избежать повторов будем считать, что слагаемые в разбиении записаны в порядке неубывания (как в примере). Будем считать, что уже построена некоторая начальная часть разбиения, которая передается в функцию в списке prefix, а также передается значение n, которое необходимо представить в виде суммы слагаемых. При этом значение n будет уменьшаться, то есть n - это оставшееся число, которое нужно записать в виде суммы слагаемых.

Переберем возможный следующее слагаемое разбиения. Оно не может превышать значения последнего элемента в списке prefix и также оно не может превышать значения n, поэтому максимальное значение для этого числа есть min(n,prefix[−1]). При этом если prefix - пустой список, то максимальное значение следующего слагаемого равно n.

Минимальное же значение следующего слагаемого может быть равно 1. Переберем все возможные значения для следующего слагаемого и рекурсивно вызовем функцию:

def generate(n, prefix):  
    if n == 0:  
        print(prefix)  
    else:  
        if len(prefix) > 0:  
            m = min(prefix[-1], n)  
        else:  
            m = n  
        for i in range(m, 0, -1):  
            generate(n - i, prefix + [i])

Теперь подсчитаем количество таких разбиений. При подсчете числа разбиений перебирается следующее слагаемое, а все оставшиеся слагаемые не превосходят выбранного слагаемого. Поэтому динамическое программирование будет двумерным: F(k,i) - число разбиения k на слагаемые, не превосходящие i.

Для определения F(k,i) переберем максимальное слагаемое в разбиении. Оно может принимать любые значения от 1 до i. В примере ниже для этого слагаемого используется переменная j. Тогда число k можно представить в виде суммы j и k−j. При этом число k−j необходимо представить в виде слагаемых, не превосходящих j. Поэтому

F(k,i)=∑j=1iF(k−j,j).

Зададим граничные значения. F(0,i)=1 при всех i  - число 0 представимо в виде пустой последовательности слагаемых. Возможна также ситуация, когда необходимо вычислить F(k,i) при i>k, в этом случае F(k,i)=F(k,k).

В следующей программе используется динамическое программирование по параметрам k и i. Сначала инициализируются значения F(0,i) для всех i. Затем во внешнем цикле перебирается значение k - число, которое необходимо представить. Во вложенном цикле перебирается максимальное значение слагаемого i и вычисляется значение F(k,i). Для этого в третьем цикле перебирается значение j - максимального слагаемого в представлении от 1 до i, после чего обновляется значение  F(k,i)- к нему добавляется число разбиений F(k−j,j). Наконец, для i>k значение F(k,i) устанавливается равным F(k,k).

F = [[0] \* (n + 1) for i in range(n + 1)]  
for i in range(0, n + 1):  
    F[0][i] = 1  
for k in range(1, n + 1):  
    for i in range(1, k + 1):  
        for j in range(1, i + 1):  
            F[k][i] += F[k - j][j]  
    for i in range(k + 1, n + 1):  
        F[k][i] = F[k][k]

Ответом на задачу (количеством разбиения числа n) является значение F(n,n).