Курс лекций по олимпиадной информатике (C)

Лекция 8. Динамическое программирование с двумя и более параметрами

Михаил Густокашин, 2009

Конспекты лекций подготовлены для системы дистанционной подготовки, действующей на сайте **informatics.mccme.ru**.

При нахождении ошибок или опечаток просьба сообщать по адресу

gustokashin@gmail.com

Версия C от 3.12.2009

1 Введение

Динамическое программирование с двумя или более параметрами почти не отличается от динамического программирования с одним параметром: в нем также встречаются задачи, в которых требуется подсчёт количества решений и нахождение оптимального решения. Также задачи можно классифицировать по количеству используемых решений меньших подзадач: задачи, использующие одну или несколько меньших подзадач; задачи, использующие все задачи меньшего размера (динамика по подмножествам) и частный случай — LR-динамика (динамика по подстрокам).

Приёмы динамического программирования будут рассматриваться на примерах реальных олимпиадных и классических задачах.

2 Использование нескольких подзадач

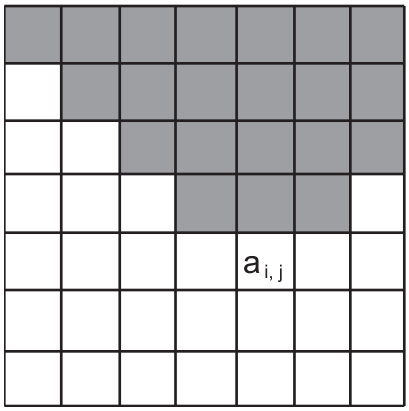
Аналогично одномерной динамике, существуют задачи использующие лишь несколько подзадач меньшего размера. Рассмотрим такие задачи на примерах.

Таблица

4-й этап Всероссийской олимпиады 2008, запад

Рассмотрим прямоугольную таблицу размером $n \times m$. Занумеруем строки таблицы числами от 1 до n, а столбцы — числами от 1 до m. Будем такую таблицу последовательно заполнять числами следующим образом.

Обозначим через a_{ij} число, стоящее на пересечении i-ой строки и j-ого столбца. Первая строка таблицы заполняется заданными числами — a_{11}, a_{12}, ..., a_{1m}. Затем заполняются строки с номерами от 2 до n. Число a_{ij} вычисляется как сумма всех чисел таблицы, находящихся в «треугольнике» над элементом a_{ij}. Все вычисления при этом выполняются по модулю r.



Более точно, значение a_{ij} вычисляется по следующей формуле:

$$a_{ij} = \left(\sum_{k=1}^{i-1} \sum_{\substack{i-k \\ 1 \leqslant t \leqslant m}}^{i+k} a_{i-k,t} \right) mod \ \ r$$

Например, если таблица состоит из трёх строк и четырёх столбцов, и первая строка состоит из чисел $2, 3, 4, 5$, а $r = 40$ то для этих исходных данных таблица будет выглядеть следующим образом (взятие по модулю показано только там, где оно приводит к изменению числа):

2	3	4	5
$5 = 2 + 3$	$9 = 2 + 3 + 4$	$12 = 3 + 4 + 5$	$9 = 4 + 5$
$23 = 2 + 3 + 4 + 5 + 9$	$0 = (2 + 3 + 4 + 5 + 5 + 9 + 12) \bmod 40 = 40 \bmod 40$	$4 = (2 + 3 + 4 + 5 + 9 + 12 + 9) \bmod 40 = 44 \bmod 40$	$33 = 3 + 4 + 5 + 12 + 9$

Требуется написать программу, которая по заданной первой строке таблицы (a_{11}, a_{12}, …, a_{1m}), вычисляет последнюю строку, как описано выше.

Формат входных данных

Первая строка входного файла содержит числа n, m и r ($2 \leqslant n, m \leqslant 2000, 2 \leqslant r \leqslant 10^9$) — число строк и столбцов таблицы соответственно, а так же число, по модулю которого надо посчитать ответ. Следующая строка содержит m целых чисел — первую строку таблицы: a_{11}, a_{12}, …, a_{1m}. Все a_{1i} неотрицательны и не превосходят 10^9.

Формат выходных данных

В первой строке выходного файла необходимо вывести m чисел — последнюю строку таблицы: a_{n1}, a_{n2}, ..., a_{nm}.

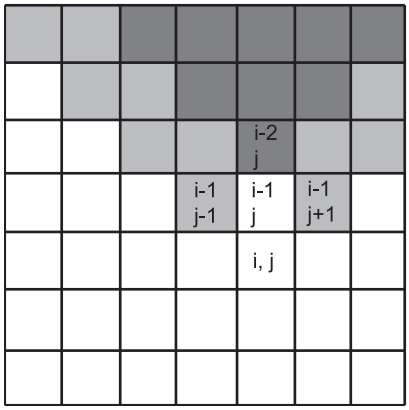
Примеры

Входные данные	Выходные данные
2 3 10 1 2 3	3 6 5
3 3 10 1 1 1	8 0 8
3 4 40 2 3 4 5	23 0 4 33

Разбор

Из условия задачи следует, что таблицу необходимо анализировать, двигаясь сверху-вниз. При этом очерёдность анализа элементов в строке не важна, т.к. никакого влияния на вычисляемую ячейку другие ячейки той же строки не оказывают.

База динамики (первая строка) нам явно задана. Рассмотрим два варианта вычисления значения в ячейке с использованием уже вычисленных данных.



В первом варианте нам достаточно завести массив B в котором мы будем накапливать ответы для ячеек. Очевидно, первая строка массива B совпадает с первой строкой массива A. Рассмотрим, как можно вычислить значение ячейки $B[i, j]$. Треугольник с вершиной в точке (i, j) состоит из треугольников с вершинами в точках $(i-1, j-1)$, $(i-1, j+1)$ а также из точек $(i-1, j)$ и (i, j). В то же время, треугольник с вершиной в точке $(i-2, j)$ перекрывается дважды, что приводит к двукратному суммированию значений в этом треугольнике. Избежать этого очень просто — достаточно один раз вычесть уже подсчитанную для этого треугольника сумму. Таким образом, в итоге получим следующую формулу $B[i][j] = B[i-1][j-1] + B[i-1][j+1] - B[i-2][j] + A[i-1][j] + A[i][j]$. Пользуясь этой формулой легко подсчитать массив, содержащий ответ.

Такой вариант также можно автоматизировать по используемой памяти: достаточно закольцевать по первому измерению массив A (при этом нам достаточно хранить всего две строки: текущую и предыдущую), а также закольцевать массив B (здесь понадобится три строки: текущая и две предыдущих). Пример закольцовывания массива приведён в разделе, посвящённом реализации очередей.

Для удобства программирования можно создать виртуальную «нулевую» строку в массиве B и заполнить ее нулями (создать барьер) — это избавит от необходимости рассматривать вторую строку как частный случай. Также разумно использовать барьеры из нулей слева и справа от таблицы.

Второй способ решения этой задачи также опирается на идеи динамического программирования, но использует решения подзадач, отличных от исходных. А именно, можно создать два дополнительных массива L и R, таких, что ячейка $L[i][j]$ будет содержать сумму всех элементов на диагонали, проведённой от точки (i, j) вверх и влево (для массива R диагональ проводится вверх и вправо). Первая строка массивов L и R будет совпадать с первой строкой массива A. Таким образом, формулы пересчёта будут

выглядеть следующим образом:

- $L[i][j] = L[i-1][j-1] + A[i][j]$
- $R[i][j] = R[i-1][j+1] + A[i][j]$
- $B[i][j] = B[i-1][j] + L[i-1][j-1] + R[i-1][j+1]$

При реализации этого метода барьеры слева и справа также будут полезны, а все массивы можно закольцевать по первому измерению всего в две строки.

Оба способа будут иметь сложность $O(n \times m)$ т.к. каждый элемент таблицы просматривается один раз.

Рассмотрим ещё одну задачу, при решении которой используется несколько подзадач меньшей размерности. Это задача о наибольшей общей подпоследовательности (НОП) двух последовательностей.

Если из некоторой последовательности $\{a\}$ вычеркнуть часть ее элементов, то получившаяся последовательность будет называться подпоследовательностью последовательности $\{a\}$. Пусть заданы две последовательности $\{a\}$ и $\{b\}$. Их наибольшей общей подпоследовательностью $\{c\}$ называется самая длинная подпоследовательность $\{a\}$, которая также является подпоследовательностью $\{b\}$.

	1	*2*	*3*	*4*	*5*
2	0	**1**	1	1	1
3	0	1	**2**	2	2
2	0	**1**	2	2	2
4	0	1	2	**3**	3
5	0	1	3	3	**4**

Рис. 1: Пример таблицы для последовательностей $\{1, 2, 3, 4, 5\}$ и $\{2, 3, 2, 4, 5\}$

Базой динамики в этом случае будет выступать тот факт, что НОП двух пустых последовательностей является также пустая последовательность. Теперь необходимо найти способ получать решение задачи, через меньшие подзадачи. Рассмотрим первые n элементов последовательности $\{a\}$ и первые m элементов последовательности $\{b\}$ и попытаемся найти НОП этих последовательностей. Допустим, что для всех $p < n$ и $q < m$ мы научились находить НОП. Тогда переход можно сформулировать так:

- Если $a_n = b_m$, то НОП(a, b) = НОП$(a-1, b-1)$ + 1
- Если $a_n \neq b_m$, то НОП(a, b) = max(НОП$(a-1, b,\ a, b-1)$)

Результаты вычислений можно хранить в двумерной таблице (первый индекс — параметр n, второй — m) и заполнять ее двигаясь сначала по строкам, а затем по столбцам. В случае, если не ставится задача восстановления ответа, можно закольцевать эту таблицу и использовать лишь две строки. В случае, если необходимо восстановление ответа, необходимо хранить всю таблицу и перемещаться из текущей ячейки (n, m) либо в ячейку $(n-1, m-1)$ если $a_n = b_m$, либо в ячейку $(n-1, m)$ или $(n, m-1)$ в зависимости от того, в какой из ячеек значение совпадает со значением в (n, m). Ответ будет восстановлен, как обычно, в обратном порядке.

3 Использование предыдущего столбца

Если в случае одномерной динамики при мы могли использовать только один или несколько предыдущих элементов (или все предыдущие элементы), то в двух и более

мерной динамике часто возникают задачи, в которых используется только предыдущий столбец (т.е. для подсчёта решения задачи необходимо знать решения всех подзадач, у которых один из параметров на единицу меньше, чем у данной). Рассмотрим этот случай также на примере реальной задачи:

Маскарад

Московская олимпиада 10-11 2005

По случаю введения больших новогодних каникул устраивается великий праздничный бал-маскарад. До праздника остались считанные дни, поэтому срочно нужны костюмы для участников. Для пошивки костюмов требуется L метров ткани. Ткань продаётся в N магазинах, в которых предоставляются скидки оптовым покупателям. В магазинах можно купить только целое число метров ткани. Реклама магазина номер i гласит «Мы с радостью продадим Вам метр ткани за P_i бурлей, однако если Вы купите не менее R_i метров, то получите прекрасную скидку — каждый купленный метр обойдётся Вам всего в Q_i бурлей». Чтобы воплотить в жизнь лозунг «экономика страны должна быть экономной», правительство решило потратить на закупку ткани для костюмов минимальное количество бурлей из государственной казны. При этом ткани можно купить больше, чем нужно, если так окажется дешевле. Ответственный за покупку ткани позвонил в каждый магазин и узнал, что:

1. реклама каждого магазина содержит правдивую информацию о ценах и скидках;
2. магазин номер i готов продать ему не более F_i метров ткани.

Ответственный за покупку очень устал от проделанной работы и поэтому поставленную перед ним задачу «закупить ткань за минимальные деньги» переложил на своих помощников. Напишите программу, которая определит, сколько ткани нужно купить в каждом из магазинов так, чтобы суммарные затраты были минимальны.

Формат входных данных

В первой строке входного файла записаны два целых числа N и L ($1 \leqslant N \leqslant 100, 0 \leqslant L \leqslant 100$). В каждой из последующих N строк находится описание магазина номер i — 4 целых числа P_i, R_i, Q_i, F_i ($1 \leqslant Q_i \leqslant P_i \leqslant 1000, 1 \leqslant R_i \leqslant 100, 0 \leqslant F_i \leqslant 100$).

Формат выходных данных

Первая строка выходного файла должна содержать единственное число — минимальное необходимое количество бурлей.

Во второй строке выведите N чисел, разделённых пробелами, где i-ое число определяет количество метров ткани, которое нужно купить в i-ом магазине. Если в i-ом магазине ткань покупаться не будет, то на i-ом месте должно стоять число 0. Если вариантов покупки несколько, выведите любой из них.

Если ткани в магазинах недостаточно для пошивки костюмов, выходной файл должен содержать единственное число -1.

Примеры

Входные данные	Выходные данные
2 14 7 9 6 10 7 8 6 10	88 10 4
1 20 1 1 1 1	-1

Разбор

Будем решать эту задачу «динамикой по магазинам». В качестве базы динамики можно взять простой и понятный факт: покупка любого количества ткани в пустом наборе магазинов обойдётся в бесконечное число бурлей (в качестве бесконечности можно взять любое число, большее 100 000 (максимальное количество метров умноженное на максимальную цену), но такое, чтобы сумма двух бесконечностей не приводила к переполнению типа.

При реализации решения полезно написать функцию, которая по номеру магазина и количеству метров, которые необходимо купить в этом магазине, будет определять стоимость покупки. Обозначим эту функцию за $f(s, m)$, где s — номер магазина, а m — количество метров.

Теперь необходимо находить решение с помощью уже решённых задач. Пусть мы научились покупать любое количество метров ткани в первых $i-1$ магазине наилучшим образом. К нашему набору добавляется i-ый магазин и необходимо для каждого количества метров определить, сколько будет стоить их покупка в i магазинах (т.е. для каждого из i первых магазинов необходимо определить, сколько метров ткани в каком магазине покупать). Будем перебирать всевозможные количества метров ткани, которые надо купить. Пусть на данный момент нам необходимо научится покупать j метров ткани в i первых магазинах, при этом цены покупки k метров ткани в наборе из $i-1$ магазине известны для всех k и составляют $C[i-1][k]$.

Купить j метров ткани в i магазинах можно $j+1$ способом: можно купить 0 метров ткани в наборе из $i-1$ первого магазина, а все j метров — в магазине с номером i; можно купить один метр ткани в наборе из $i-1$ магазина, а $j-1$ — в магазине с номером i и т.д. Поскольку нас интересует минимальная цена покупки, то формула будет следующей:

$$C[i][j] = \min_{k=0\ldots j}(f(i,k) + C[i-1][j-k])$$

В этой задаче требуется восстановление решение. Для этого можно завести ещё один массив такого же как C размера и в каждой его ячейке хранить количество метров, купленное в этом магазине для заданных i и j.

Казалось бы, задача решена. Но в процессе разбора мы оставили некоторую неопределённость в вопросе того, какое максимальное количество метров необходимо купить. Первое приходящее в голову решение: не покупать больше чем L метров неправильно. Ведь может оказаться, что нам нужно купить 10 метров по 20 бурлей (итого 200), но если оптом купить 100 по оптовой цене в 1 бурль, то цена составит всего 100 бурлей (оставшиеся метры можно выкинуть). Поскольку каждый магазин продаёт не более 100 метров ткани, то в качестве максимального количества покупаемой ткани можно взять $L+100$. В таком случае для окончательного решения необходимо выбрать минимальное стоимость покупки K метров ткани, где $L \leqslant K \leqslant L+100$. В остальном решение не изменится. Его сложность будет составлять $O(N \times (L+100)^2)$.

Рассмотрим ещё один, более изощрённый пример использования подзадач из «предыдущего столбца».

Еловая аллея

Московская заочная олимпиада 2003

Мэр города Урюпинска решил посадить на главной аллее города, которая проходит с запада на восток, голубые ели. Причём сажать ели можно не во всех местах, а только на специально оставленных при асфальтировании аллеи клумбах.

Как оказалось, голубые ели бывают M различных сортов. Для ели каждого сорта известна максимальная длина ее тени в течение дня в западном и в восточном направлении (W_i и E_i соответственно). При этом известно, что ели растут гораздо лучше, если в течение дня они не оказываются в тени других елей.

Координатная ось направлена вдоль аллеи с запада на восток.

По заданным координатам клумб вычислите максимальное число елей, которое можно посадить, соблюдая условие о том, что никакая ель не должна попадать в тень от другой ели.

Формат входных данных

Во входном файле записано сначала натуральное число M — количество сортов елей ($1 \leqslant M \leqslant 100$). Затем идёт M пар чисел W_i, E_i, описывающих максимальную длину тени в западном и восточном направлении в течение дня для каждого сорта ели (числа W_i, E_i — целые, из диапазона от 0 до 30000). Далее идёт натуральное число N — количество клумб, в которых можно сажать ели ($1 \leqslant N \leqslant 100$). Далее идёт N чисел, задающих координаты клумб (координаты — целые числа, по модулю не превышающие 30000). Клумбы перечислены с запада на восток (в порядке возрастания их координат).

Если на клумбе с координатой X мы посадили ель, максимальная тень которой в восточном направлении равна E, то все клумбы с координатами от $X + 1$ до $X + E\check{}1$ попадают в тень от этой ели, а клумба с координатами $X + E$ — уже нет. Аналогично для тени в западном направлении.

Формат выходных данных

В выходной файл выведите сначала число A — максимальное количество елей, которые удастся посадить, а затем A пар чисел, описывающих ели. Первое число каждой пары задаёт номер клумбы, в которую садится ель. Второе число определяет номер сорта этой ели.

Пример

Входные данные	Выходные данные
3 1000 3 1 200 128 256 3 1 2 4	2 1 1 3 2

Разбор

В этой задаче потребуется динамика с двумя параметрами. Одним из них будет количество высаженных ёлок, а вторым — последняя занятая клумба. В самой таблице будем хранить сорт ели, посаженной в эту последнюю занятую клумбу.

Подумаем, какая база динамики здесь может пригодится. Путём несложных рассуждений можно понять, что первая клумба всегда должна быть занята, её пропуск может только ухудшить решение. При этом, поскольку слева от самой левой клумбы ничего не растёт, то никаких ограничений на тень, отбрасываемую на запад ёлкой, посаженной в первую клумбу, не накладывается. Логично, что необходимо минимизировать тень, отбрасываемую этой ёлкой на восток. Таким образом, базой динамики будет следующее: для количества ёлок 1 и последней занятой клумбой 1 в клумбу номер 1 необходимо

сажать ёлку, которая отбрасывает наименьшую тень на восток:

$$D[1][1] = K, \text{где } E_k \leqslant \underset{i=0\ldots M-1}{E_i}$$

Все остальные первого столбца (где количество высаженных ёлок превышает 1, номер последней занятой клумбы) можно заполнить признаком невозможности такой конфигурации — числом -1, например.

База динамики известна, теперь необходимо сформулировать общее правило перехода к большему решению. Для каждого количества занятых клумб, для каждого номера последней занятой клумбы будем выбирать сорт ели, который можно в неё посадить и который отбрасывает наименьшую тень на восток (сорт, отбрасывающий большую тень на восток может только ухудшить наше решение, накрыв лишние клумбы, но не улучшить его). Будем последовательно рассматривать соотношения, которые должны выполняться. В данный момент мы сажаем ёлку в клумбу номер i и общее количество ёлок должно стать j (логично, что $j \leqslant i$). Будем перебирать все предыдущие клумбы $prev = 1 \ldots i-1$. Для каждой из предыдущих клумб необходимо выполнить некоторые проверки и действия. Естественно, что $D[j-1][prev] \neq -1$, иначе предыдущей конфигурации не существовало. Далее необходимо проверить, что предыдущая ёлка не накрывает своей тенью текущую клумбу, т.е. $X[prev] + E[D[j-1][prev]] \leqslant X[i]$. Если эти условия выполнены, то в клумбу с номером i при условии, что последняя предыдущая занятая клумба имеет номер $prev$ теоретически можно посадить ель (i-я клумба не накрывается тенью от предыдущей ели). Будем перебирать все сорта елей $p = 0 \ldots M$. Для каждого из сортов необходимо проверить, что он, будучи высаженным в клумбу номер i не накрывает своей тенью предыдущую клумбу (с номером $prev$). Эта проверка осуществляется следующим образом: $X[prev] \leqslant X[i] - W[p]$. Если условие выполнено, то такой сорт ели в эту клумбу посадить можно. Среди всех подходящих сортов выберем и запишем в ячейку $D[j][i]$ номер этого сорта. Для восстановления решения удобно содержать массив такого же размера, в соответствующую ячейку которого записывать значение $A[j][i] = prev$, при котором был достигнут наилучший результат.

Чтобы найти ответ, необходимо пройти по таблице и найти такое $D[j][i] \neq -1$, в котором j (количество высаженных ёлок) максимально. Выведем это число. Запомним, какой сорт ели был высажен в клумбу с номером i и восстановим полное решение. Поскольку одна ель уже была высажена, то оставшееся количество уменьшилось на 1. Номер предыдущей занятой клумбы нам известен — это $A[j][i]$, сорт ели для предыдущей клумбы легко определить: $D[j-1][A[j][i]]$. Затем проделаем те же действия для ячейки $D[j-1][A[j][i]]$. Уменьшая количество высаженных ёлок на 1, мы восстановим полное решение.

Сложность решения этой задачи составит $O(N^3 \times M)$ (необходим проход по всем ячейкам таблицы за N^2, для каждой из ячеек мы перебираем всевозможные предыдущие клумбы за N и всевозможные сорта за M).

4 LR-динамика

В предыдущих разделах мы рассматривали динамику, где либо использовалось некоторое фиксированное количество предыдущих подзадач, либо все задачи размерностью от 1 до $N-1$. В двумерном случае также возможен вариант, когда задача задаётся не

одним параметром N, в двумя параметрами L и R — левой и правой границей подряд идущих значений, для которых мы ищем решение. Также такой метод называется «динамика по подстрокам». Обычно, в таких задачах за базу динамики берётся диагональ матрицы, т.е. подстроки, состоящие из одного значения (при $L = R$). Принципы динамического программирования остаются те же самые, тем не менее, рассмотрим на примерах.

Скобки

Московская командная олимпиада 2004

Назовём строку S правильной скобочной последовательностью, если она состоит только из символов '{', '}', '[', ']', '(', ')' и выполнено хотя бы одно из следующих трёх условий:

1. S — пустая строка;
2. S можно представить в виде $S = S_1 + S_2 + S_3 + ... + S_N$ ($N > 1$), где S_i — непустые правильные скобочные последовательности, а знак "+"обозначает конкатенацию (приписывание) строк;
3. S можно представить в виде $S = '\{' + C + '\}'$ или $S = '[' + C + ']'$ или $S = '(' + C + ')'$, где C является правильной скобочной последовательностью.

Дана строка, состоящая только из символов '{', '}', '[', ']', '(', ')'. Требуется определить, какое минимальное количество символов надо вставить в эту строку для того, чтобы она стала правильной скобочной последовательностью.

Формат входных данных

В первой строке входного файла записана строка, состоящая только из символов '{','}', '[',']', '(',')'. Длина строки не превосходит 100 символов.

Формат выходных данных

Вывести в первую строку выходного файла единственное неотрицательное целое число — ответ на поставленную задачу.

Примеры

Входные данные	Выходные данные
{(})	2
([{}])	0

Разбор

Задача решается методом LR-динамики. А именно, в ячейке таблицы $D[L, R]$ будем хранить сколько скобок необходимо добавить, чтобы последовательность символов с L по R стала правильной скобочной последовательностью.

Для облегчения программирования введём базу динамики следующим образом:

1. Все последовательности, состоящие из одной скобки, требуют добавления одной скобки (т.е. главную диагональ матрицы заполняем 1).
2. Последовательности, у которых правая граница меньше левой, требуют добавления 0 скобок.

Теперь необходимо научится решать задачу для параметров L и R считая, что все меньшие подзадачи уже решены. В LR-динамике размер задачи определяется как разность между правой и левой границей (в нашем случае это количество символов).

Задача решается очень легко. Если на месте L стоит открывающая скобка, а на месте R — соответствующая ей закрывающая, то $D[L,R] = D[L+1, R-1]$. Это соотношение объясняется тем, что на интервале $L+1, R-1$ у нас уже получена правильная скобочная последовательность, а окружённая парой скобок она также остаётся правильной и добавлений не требует.

В случае же если на местах L и R не стоит пара соответствующих скобок, то решение определяется следующим образом:

$$D[L,R] = \min_{K=L\ldots R-1}(D[L,K] + D[K+1,R])$$

Эта формула обозначает, что мы разбиваем последовательность от L до R на две части всеми возможными способами и выбираем среди всех таких разбиений наилучшее. В частности, последовательность '[]()' будет разбита на две последовательности из 2 символов, каждая из которых требует 0 добавленных скобок, и результат для неё также будет равен 0.

Метод разбиения на 2 части применяется практически во всех задачах на LR-динамику, а отличаются они лишь начальной инициализацией и специфичными условиями, при которых можно находить лучшее решение.

Также в этой задаче удобно использовать программистский приём, называемый «ленивой динамикой» или «рекурсией с меморизацией». Действительно, если в предыдущих задачах порядок решения подзадач определялся и реализовывался легко, то в LR-динамике необходимо решать подзадачи, двигаясь по диагоналям матрицы, что уже неприятно. А встречаются задачи, в которых порядок обхода ещё более запутанный. В таких ситуациях удобно пользоваться рекурсивной функцией, которая будет похожа на перебор всех вариантов, но будет запоминать однажды посчитанное решение. В неё же можно загнать и ограничения. Идея ее следующая: сначала заполним всю матрицу, которая необходима для подсчёта, признаком отсутствия решения (например, числом -1). Обращаться к этой матрице при ее подсчёте можно только с помощью рекурсивной функции, т.е. чтобы обратится к элементу $D[L][R]$ необходимо каждый раз вызывать функцию $f(L,R)$. Функция, в свою очередь, будет проверять, подсчитано ли это значение и если да, то возвращать его. Иначе необходимо подсчитать значение, сохранить его в матрице и опять же вернуть. Для это задачи функция может выглядеть так (мы рассмотрим пример только для круглых скобок, остальные несложно добавить):

```
int f(int L, int R)
{
  int K, now;
  if (D[L][R] == -1) // значение еще не подсчитано
    if (R < L) D[L][R] = 0; // база динамики №1
    else if (R == L) D[L][R] = 1; // база динамики №2
    else
      if ((S[L] == '(') && (S[R] == ')')) D[L][R] = f(L+1, R-1);
      else { // первая и последняя скобки не образуют пары
        D[L][R] = R-L+1; // в худшем случае к каждой добавим
```

```
        for (K = L; K < R; K++) { // разбиваем на две части
          now = f(L, K) + f(K+1, R); // ищем стоимость
          if (now < D[L][R]) D[L][R] = now; // если надо меняем
        }
      }
  return D[L][R]; // возвращаем последнее значение
}
```

Теперь для того чтобы найти решение задачи достаточно считать входную строку S, заполнить матрицу D признаками неподсчитанности решения и вывести значение функции $f(0, N-1)$, где N — длина входной строки S. Все остальное сделается «само» и о порядке вычисления подзадач беспокоится не следует.

Реализовывать ленивую динамику надо всегда исходя из приведённой выше схемы: это позволит писать ее коротко и правильно. Ленивая динамика применима во всех задачах динамического программирования и, местами, даже более понятна, но злоупотреблять ей не стоит, т.к. она вызывает накладные расходы как по памяти (забивается стек), так и по времени (запуск функции — не самая быстрая операция).

Рассмотрим ещё одну, немного нестандартную, простую задачу на LR-динамику:

Максимальный подпалиндром

Командный чемпионат школьников Санкт-Петербурга 1999

Палиндромом называется строка, которая одинаково читается как слева направо, так и справа налево. Подпалиндромом данной строки называется последовательность символов из данной строки, не обязательно идущих подряд, являющаяся палиндромом. Например, `HELOLEH` является подпалиндромом строки `HTEOLFEOLEH`. Напишите программу, находящую в данной строке подпалиндром максимальной длины.

Формат входных данных

Во входном файле находится строка длиной не более 100 символов, состоящая из заглавных букв латинского алфавита.

Формат выходных данных

Выведите на первой строке выходного файла длину максимального подпалиндрома, а на второй строке сам максимальный подпалиндром. Если таких подпалиндромов несколько, то ваша программа должна вывести любой из них.

Пример

Входные данные	Выходные данные
`HTEOLFEOLEH`	`7` `HELOLEH`

Разбор

Отличие этой задачи от стандартной LR-динамики состоит в том, что разбиения на две части не предусматривается. При решении задачи будем пользоваться стандартными средствами: заведём матрицу D где первый индекс будет означать левую границу подстроки, для которой мы ищем максимальный подпалиндром, а второй — правую.

База динамики будет следующей: последовательность из 0 или отрицательного количества символов (когда L больше чем R) имеет максимальный подпалиндром длины 0. Последовательность из 1 символа даёт подпалиндром длины 1 (поскольку строка длины 1 всегда является палиндромом).

Переход к большей задаче осуществляется следующим образом:

- Если $S[L] = S[R]$, то $D[L, R] = D[L + 1, R - 1] + 2$
- Если $S[L] \neq S[R]$, то $D[L, R] = max(D[L, R - 1], D[L + 1, R])$

При реализации удобно использовать ленивую динамику.

5 Динамика по профилю

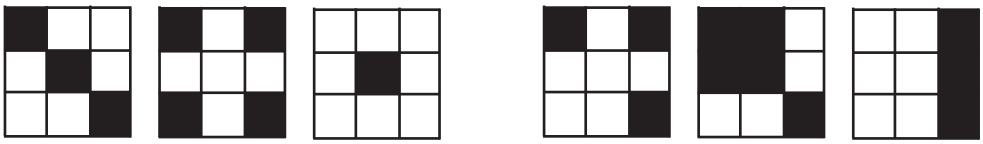
Динамическим программированием по профилю называется такой вид динамического программирования, когда подзадача меньшего размера оканчивается некоторой конфигурацией (профилем), которую можно закодировать числом. К подзадаче добавляется ещё один элемент с некоторой конфигурацией и он становится профилем новой подзадачи. Кроме того, можно составить таблицу допустимых переходов от одного профиля к другому; составить набор допустимых начальных профилей и решать задачу, увеличивая количество элементов в подзадаче. Обычно задачи на динамику по профилю требуют подсчёта числа решений. Обратимся, как обычно при изучении динамического программирования, к примерам.

Симпатичные узоры

Всероссийская командная олимпиада школьников по программированию 2000

Компания *B*rokenTiles планирует заняться выкладыванием во дворах у состоятельных клиентов узор из чёрных и белых плиток, каждая из которых имеет размер 1×1 метр. Известно, что дворы всех состоятельных людей имеют наиболее модную на сегодня форму прямоугольника $M \times N$ метров.

Однако при составлении финансового плана у директора этой организации появилось целых две серьёзных проблемы: во первых, каждый новый клиент очевидно захочет, чтобы узор, выложенный у него во дворе, отличался от узоров всех остальных клиентов этой фирмы, а во вторых, этот узор должен быть симпатичным. Как показало исследование, узор является симпатичным, если в нем нигде не встречается квадрата 2×2 метра, полностью покрытого плитками одного цвета. На рисунке 1 показаны примеры различных симпатичных узоров, а на рисунке 2 — несимпатичных.



Для составления финансового плана директору необходимо узнать, сколько клиентов он сможет обслужить, прежде чем симпатичные узоры данного размера закончатся. Помогите ему!

Формат входных данных

В первой строке входных данных содержатся два положительных целых числа, разделённых пробелом: M и N ($1 \leqslant M \times N \leqslant 30$).

Формат выходных данных

Выведите единственное число — количество различных симпатичных узоров, которые можно выложить во дворе размера $M \times N$. Узоры, получающиеся друг из друга сдвигом, поворотом или отражением, считаются различными.

Примеры

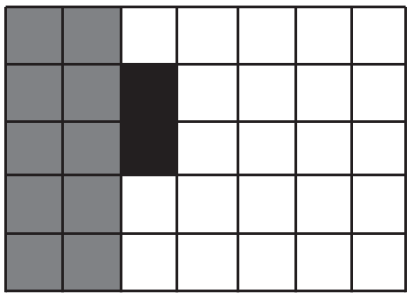
Входные данные	Выходные данные
2 2	14
3 3	322

Разбор

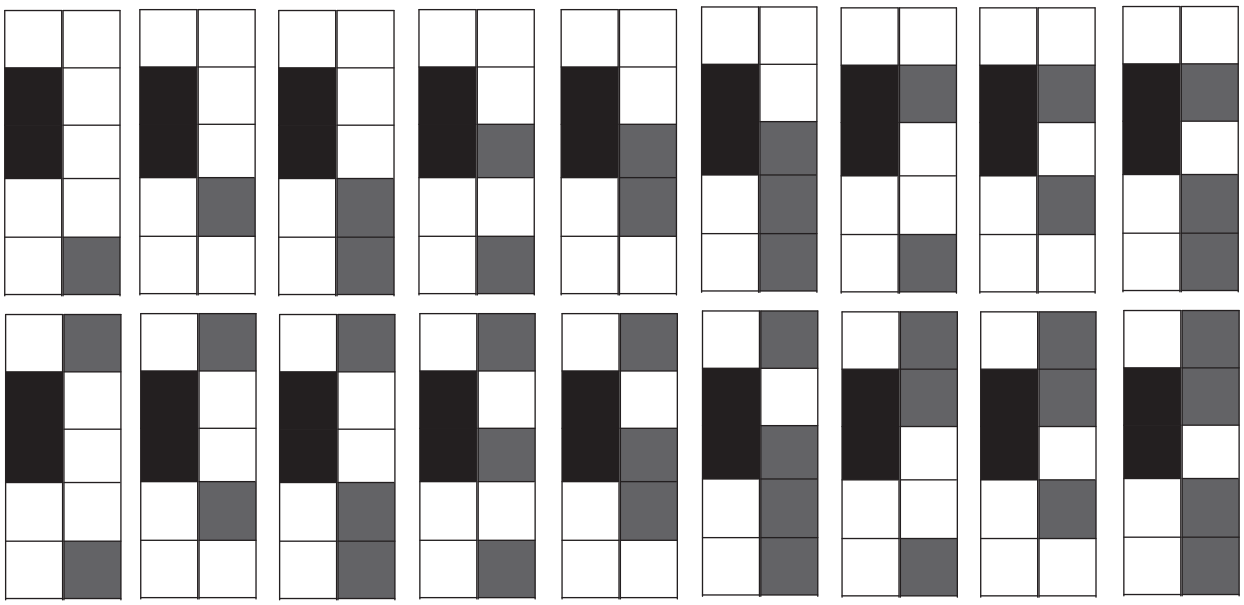
Заметим, что каждая клетка таблицы может иметь два состояния: быть белой (0) или чёрной (1). Из ограничений к задаче можно понять, что длина минимальной из сторон не превысит 5, будем называть минимальную из сторон столбцом.

Если в таблице всего один столбец, то он может быть заполнен произвольным образом, т.е. ответом будет являться число M^2.

Каждый столбец кодируется 5 битами и может иметь 2^5 состояний. Будем называть число, которым кодируется столбец *профилем*. Будем считать, что все столбцы с номерами от 1 до $K-1$ уже полностью корректно заполнены и столбец с номером $K-1$ имеет профиль равный числу M.



На рисунке последний столбец имеет профиль 01100_2 или 12_{10}. Покажем совместимые с ним профили, которые при расположении столбцов рядом не образуют квадрат 2×2 одного цвета. Такими профилями будут профили 00001_2, 00010_2, 00011_2, 00101_2, 00110_2, 00111_2, 01001_2, 01010_2, 01011_2, 10001_2, 10010_2, 10011_2, 10101_2, 10110_2, 10111_2, 11001_2, 11010_2 и 11011_2.



Таким образом, если имеется корректная таблица из $K-1$ столбца, оканчивающаяся профилем 01100_2, то из неё можно получить таблицы длиной K и имеющие в качестве профиля одно из 18 совместимых с 01100_2 чисел. При этом для каждой длины таблицы и её профиля можно хранить количество вариантов составить такую корректную таблицу, а при каждом переходе суммировать для таблицы длины K с профилем P все значения для таблиц длины $K-1$ и профилем, совместимым с P. В виде формулы это можно записать так:

$$D[K][P] = \sum_{L=0}^{2^M} D[K-1][L] \times g(P, L)$$

Где K — длина таблицы, P — интересующий нас профиль, M — высота столбца, L — профиль предыдущего столбца, а функция $g(P, L)$ возвращает 1 в случае, если столбцы P и L совместимы и 0 в противном случае. База динамики, как уже говорилось выше, состоит в следующем: при $K = 1$ для любого профиля количество вариантов раскраски клеток равно единице. Получать решение следует двигаясь сначала по количеству столбцов, для каждого из столбцов перебирать профиль и для каждого профиля, в свою очередь, перебирать все предыдущие профили.

Функцию $g(P, L)$ можно заменить предподсчитанной матрицей, чтобы не вызывать проверку на совместимость профилей много раз. При этом считать совместимость профилей удобно с использованием битовых операций.

Сложность такого решения составит $O(N \times (2^M)^2)$. Ответ гарантированно не превысит 2^30 степени, т.к. всего, даже без ограничений, всю таблицу можно раскрасить 2^30 способами.

Рассмотрим ещё один метод решения задач на динамику по профилю.

Симпатичные узоры возвращаются

Зимние сборы к международной олимпиаде 2003

Со времён написания условия предыдущей задачи многое изменилось. Симпатичные узоры стали очень популярны по всему миру, поэтому люди готовы содержать очень большой участок земли, лишь бы иметь на ней узор, не встречающийся больше нигде.

Теперь компания BrokenTiles является ведущим производителем симпатичных узоров в мире! Для составления плана исполнительному директору Васе по-прежнему необходимо знать, сколько клиентов могут рассчитывать на узор данных размеров.

Так как масштабы буквально мировые, $N \leqslant 10^{100}$. Однако Вася не любит большие числа, поэтому просит выдать ответ по модулю P.

Формат входных данных

В первой строке входных данных содержатся три положительных целых числа, разделённых пробелом: N, M и P ($1N \leqslant 10^{100}, 1 \leqslant M \leqslant 5, 1 \leqslant P \leqslant 10\,000$).

Формат выходных данных

Выведите единственное число — количество различных симпатичных узоров, которые можно выложить во дворе размера $M \times N$ по модулю P. Узоры, получающиеся друг из друга сдвигом, поворотом или отражением, считаются различными.

Примеры

Входные данные	Выходные данные
2 2 5	4
3 3 23	0

Разбор

Эта задача очень похожа на предыдущую, но та решалась за линейное от N время, что наверняка вызовет превышение ограничения на время работы программы, при $N = 10^{100}$.

Тем не менее, задача решаемая. Достаточно вспомнить, что мы умеем считать числа Фибоначчи за $O(\log N)$ и воспользоваться похожим методом. Пусть у нас есть таблица совместимости профилей G размером $2^M \times 2^M$. Приведём пример такой таблицы для $M = 2$:

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 0 \end{pmatrix}$$

Единица на пересечении i-ой строки и j-го столбца обозначает, что переход от профиля i к профилю j возможен. Если мы просуммируем значения всех ячеек этой матрицы, то получим количество вариантов раскраски прямоугольника 2×2. Вспомним, как делался переход от одного профиля к другому:

$$D[K][P] = \sum_{L=0}^{2^M} D[K-1][L] \times G[P][L]$$

Заметим, что для некоторого профиля мы перебираем все другие профили и суммируем количество вариантов в случае, если они совместимы (если они несовместимы, то тоже суммируем, но нули). Это чрезвычайно похоже на умножение матриц. Действительно, возьмём матрицу G и возведём ее в квадрат. При этом идейно произойдёт следующее: мы будем идти от профиля i в профиль j через всевозможные профили, т.е. мы переберём все k и, в случае, если возможны переходы из i в k и из k в j, то увеличим количество вариантов попасть из i в j за два шага на 1. Таким образом, сумма элементов матрицы G^2 будет равна ответу для задачи при $N = 3$. Аналогично можно делать переход не через один, а через несколько шагов. Т.е. сумма элементов матрицы G^{N-1} будет равна ответу на задачу длины N. Для ускорения можно воспользоваться быстрым возведением в степень (оно применимо для любых объектов, допускающих умножение), оно и другие необходимые операции описаны в лекции, посвящённой арифметике и теории чисел.