ДИНАМИЧЕСКОЕ ПРОГРАММИРОВАНИЕ

Основные определения (подробно)

Многие задачи практического программирования являются задачами на перебор вариантов и выбор среди этих вариантов допустимого или наилучшего по тому или иному критерию. Однако рассмотреть все варианты, в силу чрезвычайно большого их количества, зачастую не представляется возможным .

Для ряда задач, сходных по формулировке с проблемами, действительно требующими полного перебора вариантов, можно найти гораздо более эффективное решение.

Чаще всего в таких случаях решение сводится к нахождению решений подзадач меньшей размерности, которые запоминаются в таблице и никогда более не пересчитываются, а подзадачи большей размерности используют эти уже найденные решения.

Такой метод называется ***динамическим программированием***, еще его называют табличным методом.

В общей же форме под динамическим программированием понимают процесс пошагового решения задачи оптимизации, при котором на каждом шаге из множества допустимых решений выбирается одно, которое оптимизирует заданную целевую или критериальную функцию.

Как правило, связь задач и подзадач формулируется в виде некоторого "принципа оптимальности" и выражается системой уравнений (рекуррентных соотношений).

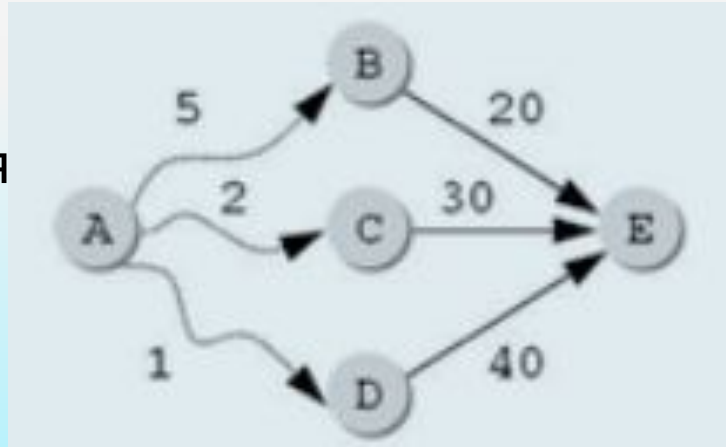
Основы теории динамического программирования были заложены Р. Беллманом (*Беллман Ричард.* Динамическое программирование).

Заметим, что слово «программирование» в приведенном названии (dynamic programming), также как и в "линейном программировании" (linear programming) **не** означает составление программ для компьютера.

Основные определения (коротко)

Динамическое программирование — это способ решения сложных задач путем сведения их к более простым задачам того же типа.

Такой подход впервые систематически применил американский математик Ричард Беллман при решении сложных многошаговых задач оптимизации. Его идея состояла в том, что оптимальная последовательность шагов оптимальна на любом участке. Например, пусть нужно перейти из пункта A в пункт E через один из пунктов B, C или D (числами обозначена "стоимость" маршрута): Пусть уже известны оптимальные маршруты из пунктов B, C и D в пункт E (они обозначены сплошными линиями) и их "стоимость". Тогда для нахождения маршрута из A в E нужно выбрать вариант, который даст минимальную стоимость по сумме двух шагов. В данном случае это маршрут A-B-E, стоимость которого равна 25. Как видим, такие задачи решаются "с конца».



Задача. Найти количество K_N цепочек, состоящих из N нулей и единиц, в которых нет двух стоящих подряд единиц.

При больших N решение задачи методом перебора потребует огромного времени вычисления.

Для того, чтобы использовать метод динамического программирования, нужно:

1) выразить K_N через предыдущие значения последовательности $K_1, K_2, \ldots, K_{N-1}, K_N$;
2) выделить массив для хранения всех предыдущих значений K_i *(i= 1,…, N-1).*

Самое главное — вывести рекуррентную формулу, выражающую K_N через решения аналогичных задач меньшей размерности. Рассмотрим цепочку из N бит, первый элемент которой — 0.

1	2	3			N-1	N
0			…			

Продолжение решения:

Поскольку дополнительный 0 не может привести к появлению двух соседних единиц, подходящих последовательностей длиной *N* с нулем в начале существует столько, сколько подходящих последовательностей длины N-1, то есть K_{N-1}.

Если же первый символ — 1, то вторым обязательно должен быть 0, а остальная цепочка из N-2 битов должна быть любой "правильной". Поэтому подходящих последовательностей длиной N с единицей в начале существует столько, сколько подходящих последовательностей длины N-2, то есть K_{N-2}.

1	2	3			N-1	N
1	0		…			

В результате получаем $K_N = K_{N-1} + K_{N-2}$. Значит, для вычисления очередного числа нам нужно знать два предыдущих.

Рассмотрим простые случаи.

Очевидно, что есть две последовательности длиной 1 (0 и 1), то есть K_1 — 2. *Далее, есть три подходящих последовательности длины 2 (00, 01 и 10), поэтому K = 3.*

Задача. Найти количество K_N цепочек, состоящих из 10 нулей и единиц, в которых нет двух стоящих подряд единиц.

Ответ:144

Задача С3 ЕГЭ

Что нужно знать:

- динамическое программирование – это способ решения сложных задач путем сведения их к более простым задачам того же типа
- с помощью динамического программирования решаются задачи, которые требуют полного перебор вариантов:
  - «подсчитайте количество вариантов…»
  - «как оптимально распределить…»
  - «найдите оптимальный маршрут…»
- динамическое программирование позволяет ускорить выполнение программы за счет использования дополнительной памяти; полный перебор не требуется, поскольку запоминаются решения всех задач с меньшими значениями параметров

Пример задания (ЕГЭ 2012)

У исполнителя Утроитель две команды, которым присвоены номера:

1. прибавь 1

2. умножь на 3

Первая из них увеличивает число на экране на 1, вторая – утраивает его.

Программа для Утроителя – это последовательность команд.

Сколько есть программ, которые число 1 преобразуют в число 20?

Ответ обоснуйте.

Решение

1. заметим, что при выполнении любой из команд число увеличивается (не может уменьшаться)
2. начнем с простых случаев, с которых будем начинать вычисления: для чисел 1 и 2, меньших, чем 3, существует только одна программа, состоящая только из команд сложения; если через K_N обозначить количество разных программ для получения числа *N* из 1, то $K_1 = K_2 = 1$.
3. теперь рассмотрим общий случай, чтобы построить рекуррентную формулу, связывающую K_N с предыдущими элементами последовательности $K_1, K_2, \ldots, K_N$, то есть с решениями таких же задач для меньших N
4. если число N не делится на 3, то оно могло быть получено только последней операцией сложения, поэтому $K_N = K_{N-1}$
5. если N делится на 3, то последней командой может быть как сложение, так и умножение, поэтому для получения K_N нужно сложить K_{N-1} (количество программ с последней командой сложения) и $K_{N/3}$ (количество программ с последней командой умножения). В итоге получаем:

   если N не делится на 3: $K_N = K_{N-1}$

   если N делится на 3: $K_N = K_{N-1} + K_{N/3}$
6. остается заполнить таблицу для всех значений от 1 до N:

N	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
K_N	1	1	2	2	2	3	3	3	5	5	5	7	7	7	9	9	9	12	12	12

Заметим, что количество вариантов меняется только в тех столбцах, где N делится на 3, поэтому из всей таблицы можно оставить только эти столбцы:

N	1	3	6	9	12	15	18	21
K_N	1	2	3	5	7	9	12	15

заданное число 20 попадает в последний интервал (от 18 до 21), поэтому …

ответ – 12.

Задача №2963. Калькулятор
Сайт дистанционной подготовки – динамическое программирование – одномерная динамика

Ограничение по времени: 3 сек

Имеется калькулятор, который выполняет три операции:

Прибавить к числу X единицу.

Умножить число X на 2.

Умножить число X на 3.

Определите, какое наименьшее число операций необходимо для того, чтобы получить из числа 1 заданное число N.

Формат входных данных

Программа получает на вход одно число, не превосходящее 10^6.

Формат выходных данных

Одно число: наименьшее количество искомых операций.

Пример

Ввод	Вывод
1	0
5	3
562340	19

Классические задачи динамического программирования

Задача 1. «МАРШРУТ» В таблице *N*x*N*, клетки заполнены случайным образом цифрами от 0 до 9. Найти маршрут из клетки *A*(1,1) в клетку *A*(*N*, *N*) такой, что:

1) он будет состоять из отрезков, соединяющих центры клеток, имеющих общую сторону
2) длина маршрута минимально возможная
3) из всех маршрутов, удовлетворяющих условиям (1) и (2), искомый маршрут тот, сумма цифр в клетках которого максимальна.

Указания к решению:

Пусть клетка **(1, 1)** это левый верхний угол таблицы, а **(*N*, *N*),** соответственно, правый нижний угол. Из условия (2) задачи следует, что за каждый шаг мы будем продвигаться по таблице либо на шаг вправо, либо на шаг вниз, что сразу нам гарантирует минимальность в длине пути и избавляет от анализа вариантов по данному критерию.

Рассмотрим произвольную клетку таблицы (*i*, *j*). В нее мы можем прийти или из клетки **(*i*-1, *j*)**, или из **(*i*, *j*-1).** Тогда, если мы уже знаем оптимальные маршруты из клетки (1, 1) в каждую из этих двух клеток, то оптимальным маршрутом в клетку (*i*, *j*) будет подмаршрут с максимальной из двух сумм суммой плюс отрезок, соединяющий (*i*, *j*) с концом выбранного подмаршрута. Оптимальные маршруты из (1, 1) в (1, 2) и (2, 1) определены однозначно. Зная их, по указанному выше способу мы найдем оптимальные маршруты в (1, 3), (2, 2), (3, 1) и запишем их в соответствующих клетках таблицы (записывать нужно только сумму цифр маршрута и направление его последнего отрезка). Этот процесс можно продолжить пока вся таблица не будет заполнена, причем заполнять ее можно по строчкам слева направо. В клетке (*N*, *N*) мы в итоге получим значение суммы цифр искомого маршрута и последний его отрезок.

По такой таблице легко восстановить и весь маршрут, начиная с клетки (*N*, *N*).

Рассмотрим любую часть оптимального маршрута, например, между клетками (*i*1, *j*1) и (*i*2, *j*2). Докажем, что эта часть маршрута является решением исходной задачи для указанных клеток. Пусть это не так и существует маршрут с большей суммой, соединяющий эти клетки и имеющий такую же длину. Тогда и для клеток (1, 1) и (*N*, *N*) мы можем построить лучший маршрут, используя отрезки, соединяющие (1, 1) и (*i*1, *j*1), а также (*i*2, *j*2) и (*N*, *N*) из старого маршрута плюс улучшенный маршрут из (*i*1, *j*1) в (*i*2, *j*2), а это противоречит тому, что мы изначально рассматривали часть из уже оптимального маршрута. Значит, любая часть оптимального маршрута в свою очередь является оптимальной.

Поиск оптимального решения

Задача. В цистерне N литров молока. Есть бидоны объемом 1, 5 и 6 литров. Нужно разлить молоко в бидоны так, чтобы все бидоны были заполнены и количество используемых бидонов было минимальным.

Решение: Задача решается перебором вариантов для любого введенного числа N. Самый простой подход — заполнять сначала бидоны самого большого размера (6 л), затем — меньшие и т.д. Алгоритм, который мы применили, называется "жадным". Он состоит в том, чтобы на каждом шаге многоходового процесса выбирать наилучший в данный момент вариант, не думая о том, что впоследствии этот выбор может привести к худшему решению. «Жадный» алгоритм не всегда приводит к оптимальному решению. Например, для *N — 10 «жадный»* алгоритм дает решение 6+1 + 1 + 1 + 1 — всего 5 бидонов, в то время как можно обойтись двумя (5 + 5).

Как и в любом решении, использующем динамическое программирование, главная проблема – составить рекуррентную формулу. Сначала определим оптимальное число бидонов *K*, а потом подумаем, как определить, какие именно бидоны нужно использовать.

Выбираем бидоны постепенно. Тогда последний выбранный бидон может иметь объем 1 л, 5 л, 6 л.

Если 1 л, то $K_N = 1 + K_{N-1}$. *Если последний бидон имеет объем 5 л, то* $K_N = 1 + K_{N-5}$, *а если 6 л —* $K_N = 1 + K_{N-6}$.

Так как нам нужно выбрать минимальное значение, то

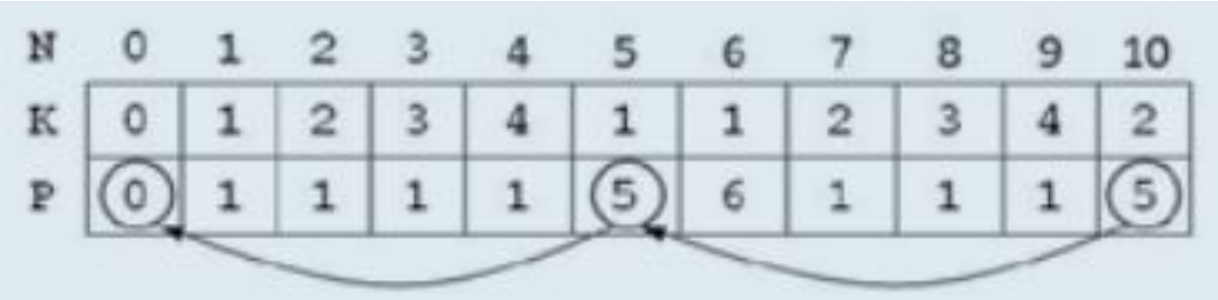
$$K_N = 1 + \min(K_{N-1}, K_{N-5}, K_{N-6}).$$

Вариант, выбранный при поиске минимума, определяет последний добавленный бидон, его нужно сохранить в отдельном массиве P. Этот массив будет использован для определения количества выбранных бидонов каждого типа. В качестве начального значения берем $K_0 = 0$.

Полученная формула применима при *N* > =6. *Например,*

$K_3 = 1+K2=3$, $K_5 = 1 + \min(K_4, K_0) = 1$.

массивы для *N – 10*.



N	0	1	2	3	4	5	6	7	8	9	10
K	0	1	2	3	4	1	1	2	3	4	2
P	0	1	1	1	1	5	6	1	1	1	5