Введение

В узком смысле динамическим программированием называется один из методов решения оптимизационных задач, применяющийся при выполнении принципа оптимальности Беллмана. Оптимизационной называется задача выбора из множества допустимых решений одного, оптимального по некоторому признаку. Формально, это означает, что по каждому допустимому решению вычисляется некоторая целевая функция, и оптимальным считается то решение, значение целевой функции которого является наибольшим (наименьшим).

Пусть нахождение решения задачи разбивается на n шагов, на каждом из которых рассматривается та же задача, но меньшей размерности. Если значение целевой функции оптимального решения на k-том шаге зависит от значения целевой функции оптимального решения на шаге $k-1$ и не зависит от самого решения, то говорят, что имеет место принцип оптимальности Беллмана, и задачу можно решить методами динамического программирования

Рассмотрим пример. Последовательность целых чисел $a_1, a_2, \ldots, a_k$ формируется по шагам, причем на каждом шаге в конец последовательности добавляется одно новое число. Требуется после каждого шага найти суффикс последовательности, сумма которого будет максимальна. Суффиксом последовательности длины r называются числа $a_{k-r+1}, a_{k-r+2}, \ldots, a_k$, то есть r самых правых чисел последовательности. Суффикс может быть пустым, тогда его сумма считается равной нулю.

Обозначим за $R(k)$ длину суффикса с максимальной суммой после шага k, а за $S(k)$ — значение этой суммы. Тогда возможны два варианта. Если $R(k) = 0$, то $S(k) = 0$. Иначе, $S(k) = a_k + T(k-1)$, где $T(k-1)$ сумма чисел некоторого суффикса последовательности $a_1, a_2, \ldots, a_{k-1}$. Сумма двух слагаемых максимальна, если каждое из слагаемых максимально. Следовательно, $T(k-1) = S(k-1)$. Окончательно, $S(k) = \max(0, S(k-1) + a_k)$. Обратим внимание, что $R(k)$ в этой формуле не фигурирует, что является проявлением принципа Беллмана: величина оптимального решения на k-том шаге зависит от величины оптимального решения на $k-1$ шаге и не зависит от того, как именно оно было получено.

Вместе с тем, в задачах динамического программирования само решение тоже может быть легко вычислено. В данном случае: если $S(k) = 0$, то $R(k) = 0$; иначе $R(k) = R(k-1) + 1$.

Рассмотрим еще одну широко известную задачу. Задана прямоугольноя матрица из натуральных чисел a_{ij}. Найти траекторию от левого верхнего до правого нижнего угла матрицы, сумма чисел в которой будет минимальна. Траекторией считаем твкую последовательность клеток, смежных по стороне, что каждая следующая клетка расположена справа или снизу от предыдущей. Например, в следующих матрицах оптимальные траектории выделены жирным шрифтом.

1	**1**	1	1
4	**7**	5	6
6	**2**	5	4
1	**1**	**1**	**1**

1	**3**	2	1	4
4	**1**	**2**	6	5
6	3	**3**	**2**	5
1	5	3	**4**	**1**

Обозначим за $S(i,j)$ длину оптимальной траектории от левого верхнего угла матрицы к ячейке в строке i и столбце j. Пусть $i > 1$ и $j > 1$. Всякая траектория к ячейке (i,j) проходит либо через ячейку $(i-1,j)$, либо через $(i,j-1)$. В первом случае $S(i,j) = a_{ij} + T(i-1,j)$, где $T(i-1,j)$ — длина некоторой траектории к ячейке $(i-1,j)$. Сумма является минимальной, если минимальны оба слагаемых, следвательно $T(i-1,j) = S(i-1,j)$. Тогда $S(i,j) = a_{ij} + S(i-1,j)$. Аналогично, если кратчайший путь проходит через ячейку $(i,j-1)$, то $S(i,j) = a_{ij} + S(i,j-1)$. Для получения окончательного ответа требуется взять наименьшее из этих двух значений, $S(i,j) = a_{ij} + min(S(i-1,j), S(i,j-1))$. Для получения полного решения требуется рассмотреть также случаи $i = 1$ и $j = 1$. Полная

формула будет выглядеть так

$$S(i,j) = \begin{cases} a_{11} & i=1, j=1 \\ S(i-1,1) + a_{i1} & i>1, j=1 \\ S(1,j-1) + a_{1j} & i=1, j>1 \\ min(S(i-1,j), S(i,j-1)) + a_{ij} & i>1, j>1 \end{cases}$$

Наиболее простой способ реализации этого решения заключается в создании двумерного вектора, сохраняющего все возможные значения $S(i,j)$. При необходимости экономии памяти, можно обойтись хранением одной строки или одного столбца.

Функцию $S(i,j)$ часто называют динамикой задачи. Часто бывает, что найти нужную динамику не совсем просто, поэтому при разборе решения в первую очередь предъявляется именно она.

Для сравнения рассмотрим еще одну задачу с динамикой, для которой принцип Беллмана уже не работает. Она называется задачей о рюкзаке. Пусть $\{a_1, a_2, \ldots, a_n\}$ мультимножество натуральных чисел. Требуется выбрать в нем такое подмножество, чтобы сумма чисел выбранного подмножества была максимальной но не превосходила заданного параметра w.

По аналогии с предыдущими задачами можно предложить динамику $D(k)$ — максимальная сумма не превосходящая w, составленная из некоторого подмножества чисел мультимножества $\{a_1, a_2, \ldots, a_k\}$. Но зависит ли как-нибудь $D(k)$ от $D(k-1)$? Можно привести множество примеров с отрицательным ответом на этот вопрос. Пусть мультимножество содержит числа $\{3, 6, 7, 5\}$ и $s = 11$ Для мультимножества $\{3, 6, 7\}$ оптимальным выбором будет $3 + 7 = 10$, но при добавлении нового элемента 5, состав выбранного подмножества полностью изменяется $6 + 5 = 11$. Таким образом, ранее подсчитанное значение $D(k-1)$ никак не помогает упростить вычисления для $D(k)$, что не соответствует принципу динамического программирования.

Важно отметить, что представленные рассуждения касаются только одной предложенной динамики, и отрицательный ответ о невозможности ее использования не означает, что задачу в принципе невозможно решить методами динамического программирования. Вопрос заключается лишь в сложности вычислений. Так для этой задачи можно предложить другой вариант решения методами динамического программирования. Обозначим за $S(k)$ множество всевозможных сумм не превосходящих w, составленных из некоторых элементов $\{a_1, a_2, \ldots, a_k\}$. Например, для мультимножества $\{2, 5, 1, 7\}$ и $w = 13$ $S(k) = \{1, 2, 3, 5, 6, 7, 8, 9, 10, 12, 13\}$. Для вычисления $S(k)$ можно применить следующие формулы $S(0) = \{0\}$, если $S(k-1) = \{b_1, b_2, \ldots, b_r\}$ то $S(k) = \{b_1, b_2, \ldots, b_r\} \cup \{b_1 + a_k, b_2 + a_k, \ldots, b_r + a_k\}$. Эта формула означает, что при добавлении к мультимножеству нового числа a_k его можно либо не использовать в суммах, тогда они не изменятся, либо использовать, тогда они увеличатся на a_k.

Этот метод решения может быть реализован со сложностью $O(nw)$, и, если эта сложность является приемлемой, то можно говорить о том, что задача решена методами динамического программирования.

Динамическое программирование в комбинаторных задачах

В более общем смысле динамическое программирование представляет собой прием решения задач при котором процесс вычислений организован по шагам, причем на каждом шаге запоминается только та информация, которая влияет на последующие вычисления.

Рассмотрим следующий пример. Найти количество последовательностей из нулей и единиц, в которых не встречается трех единиц подряд. Как и в ранее рассмотренных задачах, будем искать решение последовательно, сначала для $n = 1$ и $n = 2$, потом для $n = 3$ и так далее. Но, если мы обозначим за $F(k)$ количество требуемых последовательностей длины k, то сможем ли мы вычислить $F(k)$, используя только $F(k-1)$. К каждой последовательности длины $k-1$ можно приписать справа ноль или единицу, получив $2F(k-1)$

последовательностей длины k. Но, среди них окажутся и последовательности, оканчивающиеся на три единицы, которые мы не должны учитывать. Таким образом, мы должны выделять последовательности длины k, которые оканчиваются на ноль, на одну единицу, на две единицы. Это и есть та информация, которая влияет на последующие вычисления.

Обозначим за $F(k,0)$, $F(k,1)$, $F(k,2)$ количество последовательностей длины k, которые оканчиваются на 0, на 01, на 11 соответственно. Тогда, очевидны следующие формулы $F(2,0) = 2$, $F(2,1) = 1$, $F(2,2) = 1$. Если $k > 2$, то, рассмотрев способы получения разных видов последовательностей, можно записать формулы $F(k,0) = F(k-1,0) + F(k-1,1) + F(k-1,2)$, $F(k,1) = F(k-1,0)$ $F(k,2) = F(k-1,1)$, которые можно использовать при написании программы.

ООбратим внимание, что из указанных формул можно легко вывести, что $F(k,1) = F(k-1,0)$, $F(k,2) = F(k-2,0)$. Тогда, $F(k,0) = F(k-1,0) + F(k-2,0) + F(k-3,0)$. Этот пример показывает, что при вычислении можно использовать все ранее вычисленные значения F, а не только $F(k-1)$.

Рассмотрим еще один пример. Найти количество представлений натурального числа n в виде суммы k различных натуральных слагаемых. Использовать функцию динамики $F(n,k)$ в чистом виде не получится, так как на каждом следующем шаге нам надо знать, какие слагаемые были использованы на предыдущих шагах, чтобы избежать повторов. Решение состоит в том, что мы будем строить суммы из слагаемых упорядоченных по убыванию и запоминать последнее из слагаемых. Тогда динамикой задачи будет $F(n,k,r)$ — количество способов получить число n в виде суммы k различных слагаемых, величина которых не превосходит r. Формулы для вычисления динамики могут быть следующими

$$F(n,k,r) = \begin{cases} 0 & | \; n > r, k = 1 \\ 1 & | \; n \le r, k = 1 \\ 0 & | \; n == 1, k > 1 \\ 0 & | \; r < k, k > 1 \\ F(n,k,r-1) & | \; 1 < n \le r, r \ge k, k > 1 \\ F(n,k,r-1) + F(n-r,k-1,r-1) & | \; n > r, r \ge k, k > 1 \end{cases}$$

Последняя строка формулы получается по следующей логике. Если сумма содержит слагаемое r, то, выбросив это слагаемое, получим представление числа $n-r$ суммой $k-1$ слагаемого, которые не превосходят $r-1$, а $F(n-r,k-1,r-1)$ — количество таких представлений. Если же сумма не содержит слагаемого r, то все слагаемые не превосходят $r-1$, а $F(n,k,r-1)$ — количество таких представлений. Остальные строки формулы очевидны или получаются вследствие подобных рассуждений.

Ответом к задаче является $F(n,k,n)$. Элементарный вариант реализации может выглядеть как заполнение трехмерного массива значениями по приведенной формуле. В случае необходимости экономии памяти можно обойтись и двумерным массивом, так как $F(n,k,n)$ использует только значения F от $k-1$ и нет необходимости хранить все предшествующие значения.

Здесь нам потребовалось ввести параметр, которого не было в условии задачи. Именно поиск наиболее подходящих параметров для динамики является наибольшей сложностью при решении задач методами динамического программирования.

Другие распространенные виды динамик

Динамика по префиксу последовательности

Один из наиболее простых видов динамического программирования. Задана последовательность чисел $a_1, a_2, \ldots, a_n$. Поставленная задача решается для всех префиксов $a_1, a_2, \ldots, a_k$ от меньших к большим. Часто при этом ставятся дополнительные условия на последний элемент префикса. Например, если требуется выбрать подпоследовательность элементов с некоторым свойством, часто ставят дополнительное условие о том, что a_k

обязательно принадлежит искомой подпоследоательности. Например, в задаче о нахождении возрастающей подпоследовательности максимальной длины можно ввести динамику: F(k) — максимальная длина возрастающей подпоследовательности, которая заканчивается на элементе a_k. В этом случае ответ к задаче можно найти как $\max\limits_{1 \le i \le n} F(i)$

Динамика в двумерной таблице

Этот вид динамического программирования предполагает функцию динамики как минимум с двумя переменными. Часто в качестве $F(n, m)$ рассматривается целевая функция определенная на фрагменте таблицы из n строк и m столбцов. Примером задачи этого типа является рассмотренная выше задача о поиске в таблице траектории с мминимальной суммой

Динамика по префиксам двух последовательностей

В задачах этого типа заданы две последовательности элементов $a_1, a_2, \ldots, a_n$ и $b_1, b_2, \ldots, b_m$. Функция динамики имеет вид $F(i, j)$ и предназначена для решения той же задачи на начальных участках последовательностей $a_1, a_2, \ldots, a_i$ и $b_1, b_2, \ldots, b_j$. В качестве примера такой задачи можно рассмотреть задачу о поиске наибольшей общей подпоследовательности.

Последовательность чисел $c_1, c_2, \ldots, c_k$ назвается подпоследоательностью $a_1, a_2, \ldots, a_n$, если из последовательности $\tilde{a}$ можно удалить $n-k$ элементов, не изменяя порядка остальных, так, чтобы оставшиеся элементы образовали последовательность $\tilde{c}$.

Требуется найти последовательность $\tilde{c}$ максимальной длины, которая являлась бы одновременно подпоследовательностями $\tilde{a}$ и $\tilde{b}$.

Функция динамики по префиксам $F(i, j)$, очевидно, задается следующими формулами

$$F(i,0) = F(0,j) = 0 F(i,j) = \left\{ \begin{array}{l} F(i-1, j-1) + 1 \\ \min(F(i-1,j), F(i,j-1)) \end{array} \middle| \begin{array}{l} a_i = b_j \\ a_i \neq b_j \end{array} \right.$$

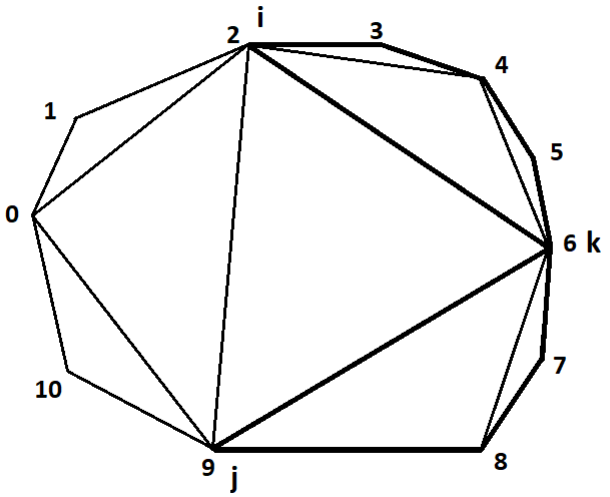
Смысл формул очень простой. Попробуйте разобраться в нем самостоятельно.

Динамика по подмассивам (подстрокам)

В задачах этого типа в задана одна строка или массив некоторых элементов, причем выделение некоторого подмассива имеет смысл с точки зрения условия. Классической задачей с таким свойством является задача о минимальной триангуляции выпуклого многоугольника.

Триангуляцией многоугольника называют его разбиение на непересекающиеся треугольники. Требуется выбрать триангуляцию с минимальной суммой периметров треугольников. Заданы вектора `x` и `y` координат выпуклого многоугольника в порядке их обхода по часовой стрелке. Отметим, что подмассив состоящий из точек с последовательными номерами от i до j также является выпуклым многоугольником, поэтому для него также определена рассматриваемая задача.

Рассмотрим функцию $F(i, j)$, $i+1 < j$ равную сумме периметров минимальной триангуляции многоугольника с точками от i до j включительно. Обозначим за $P(a, b, c)$ периметр треугольника, составленного из точек с номерами a, b и c.



Многоугольник с точками от i до j содержит сторону, соединяющую точки i и j эта сторона должна попасть в один из треугольников. Пусть третья точка треугольника имеет номер k, $i < k < j$. Тогда триангуляция будет содержать треугольник с точками i, j, k и некоторую триангуляцию многоугольников с точками от i до k и от $k+1$ до j. Например, многоугольник с точками от 2 до 9 может быть разбит на треугольник $\{2, 6, 9\}$ и два многоугольника $\{2, 3, 4, 5, 6\}$ и $\{6, 7, 8, 9\}$

Сумма минимальна, если каждое слагаемое минимально. Тогда, $F(i, j) = P(i, j, k)) + F(i, k) + F(k, j)$, при условии, что минимальная триангуляция содержит треугольник i, j, k. Теперь требуется перебрать все варианты для точки k и выбрать минимальный. Окончательно получаем $F(i, j) = \min\limits_{i<k<j}(P(i, j, k) + F(i, k) + F(k, j)$.

Здесь для хранения значений целевой функции обязательно использовать двумерный вектор. Однако, его заполнение двумя циклами

```
for (int i=0;i<n;i++)
  for (int j=i+2;j<n;j++)
```

будет неверным. (Самостоятельно ответьте на вопрос, почему?)

Правильным вариантом будет цикл по возрастанию расстояния от i до j

```
for (int len=2;len<n;len++)
  for (int i=0;i+len<n;i++) {
     int j=i+len;
  }
```

Динамика на деревьях

Задач этого типа в данном курсе пока нет.

Динамика в графах

Примером применения метода динамического программирования в теории графов могут служить рассмотренные ранее алгоритмы Прима и Дейкстры.

Динамика по профилю

Задач этого типа в данном курсе пока нет.

Динамика по подмножествам

Задач этого типа в данном курсе пока нет.

Последовательность вычисления динамики и меморизация

Во всех рассмотренных примерах, кроме задачи о минимальной триангуляции выпуклого многоугольника, динамика зависила от такой же функции с меньшими значениями переменных. Например, $F(x, y)$ зависила от $F(x-1, y)$ и $F(x, y-1)$. В этом случае последовательность вычисления F очевидна — по возрастанию значений переменных, причем порядок самих переменных неважен. В задаче о триангуляции последовательность является несколько менее очевидной, но также не представляет особенных трудностей.

Но что делать, если последовательность вычислений неочевидна или трудна в реализации? В этом случае можно применить прием, называемый меморизацией. Динамика F реализуется как рекурсивная функция. Также заводится структура данных, хранящая все параметры, для которых ранее вызывалась функция F. В простом случае это может быть многомерный вектор, заполненный изначально некоторым недопустимым значением, которое является признаком того, что функция с данным набором параметров ранее не вычислялась. В более сложных случаях это может быть `map` заданных параметров. При входе в рекурсивную функцию выполняется проверка, вызывалась ли ранее функция с этим набором параметров. В случае положительного ответа, результат извлекается из хранилища. Иначе, происходит рекурсивное вычисление по формулам, и результат заносится в хранилище. Приведем пример для динамики с двумя параметрами. В качестве недупустимого значения используется -1.

```
vector<vector<int>> Mem(n, vector<int>(n, -1));
int F(int x, int y) {
  if (Mem[x][y]==-1) {
    // Рекурсивное вычисление по формуле
     Mem[x][y]=res;
  }
  return Mem[x][y];
}
```

В принципе, такой прием можно применять во всех случаях, однако, следут понимать, что рекурсивные программы при прочих равных условиях проигрывают итеративным по скорости, поэтому их использование менее предпочтительно.