Динамическое программирование

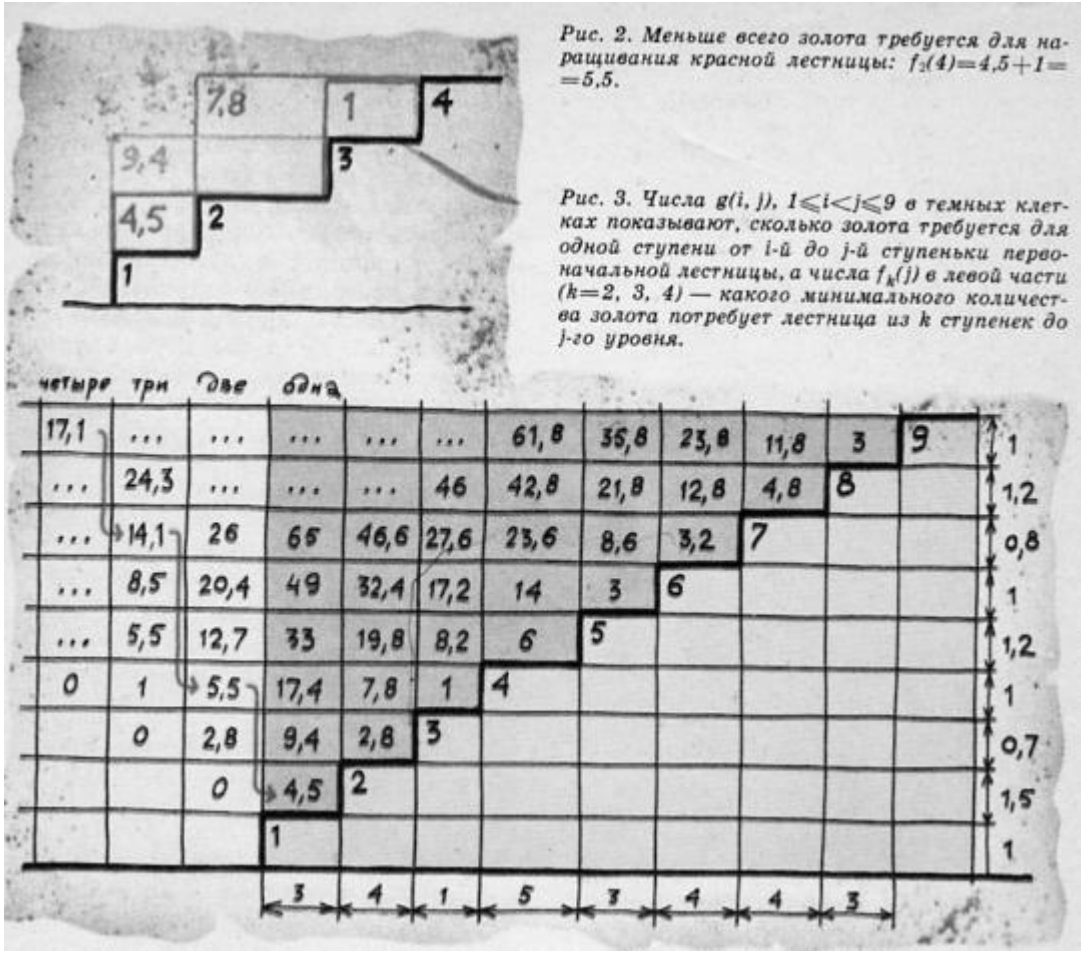

Рис. 2. Меньше всего золота требуется для наращивания красной лестницы: $f_2(4)=4,5+1=5,5$.

Рис. 3. Числа $g(i, j)$, $1 \leqslant i < j \leqslant 9$ в темных клетках показывают, сколько золота требуется для одной ступени от i-й до j-й ступеньки первоначальной лестницы, а числа $f_k(j)$ в левой части ($k=2, 3, 4$) — какого минимального количества золота потребует лестница из k ступенек до j-го уровня.

X

- Сведение задачи к подзадачам
- Понятие рекуррентного соотношения
- Использование таблиц для запоминания решений подзадач
- Алгоритм динамического программирования
- Условия применения динамического программирования

При формулировке любой задачи необходимо определить исходные данные, которые мы будем называть параметрами задачи.

Например, если мы решаем задачу нахождения корней квадратного уравнения $ax2 + bx + c = 0$, то эта задача определяется тремя параметрами – коэффициентами a, b и c.

Если же мы хотим решить задачу нахождения среднего арифметического некоторого набора чисел, то параметрами задачи будут количество чисел и их значения.

Мы хотим научиться решать задачу, сводя ее к решению подзадач.

При таком подходе любая задача может быть формализована в виде некоторой функции, аргументами которой могут являться такие величины, как:

- количество параметров;
- значения параметров.

После того, как задача представлена в виде функции с некоторыми аргументами, определим понятие ***подзадачи*** . Под ***подзадачей*** мы будем понимать ту же задачу, но с меньшим числом параметров или задачу с тем же числом параметров, но при этом хотя бы один из параметров имеет меньшее значение.

При этом для решения исходной задачи может потребоваться решение одной или нескольких подзадач.

Рассмотрим пример: Найти наибольший общий делитель (НОД) двух натуральных чисел N и M.

Если числа равны, то их НОД равен одному из чисел, т. е.

$$НОД(N, M) = N.$$

Рассмотрим случай, когда числа не равны. Известно, что

$$НОД(N, M) = НОД(N, M + N) = НОД(N + M, M).$$

Кроме того, при $N > M$ $НОД(N, M) = НОД(N – M, M)$,

а при $M > N$ $НОД(N, M) = НОД(N, M – N)$.

Последние соотношения и обеспечивают основной принцип сведения решения задачи к подзадачам: значение одного из параметров стало меньше, хотя их количество и осталось прежним.

Таким образом, решение задачи нахождения НОД(N, M) при различных значениях N и M сводится к двум подзадачам:

НОД($N – M, M$), если $N > M$;

НОД($N, M – N$), если $M > N$.

***Рассмотрим задачу* нахождения суммы *N* элементов таблицы *A*.**

Пусть функция *S*(*N*) соответствует решению нашей исходной задачи. Эта функция имеет один аргумент *N* – количество суммируемых элементов таблицы *A*. Понятно, что для поиска суммы *N* элементов достаточно знать сумму первых *N* – 1 элементов и значение *N*-го элемента. Поэтому решение исходной задачи можно записать в виде соотношения

$$S(N) = S(N - 1) + a[N].$$

Следует отметить, что это соотношение справедливо для любого количества элементов $N \geq 1$.

Это соотношение можно переписать в виде

$$S(i) = S(i - 1) + a[i], \text{ при } i > 1, S(0) = 0.$$

Последовательное применение первого соотношения при *i* = 1, 2, ..., *N* и используется при вычислении суммы *N* элементов, при этом вычисление функции производится от меньших значений аргументов к большим.

```
S[0]: = 0;
for i:= 1 to N do                                    (1)
 S[i]: = S[i – 1] + a[i];
```

В *S*[*i*] хранится значение функции *S*(i).

В круглых скобках записываются аргументы функции, а в квадратных – индексы элементов массива. При этом имя функции и имя массива, в котором хранится значение этой функции, могут совпадать.

Индекс у *S* может быть опущен, но смысл соотношения при этом остается прежним. Это связано с тем, что для вычисления следующего элемента таблицы *S* необходимо знать только предыдущий.

Задания для самостоятельного решения:

Написать соотношение для:

- нахождения произведения N элементов таблицы A;
- нахождения максимума N элементов таблицы A.

решение

Понятие рекуррентного соотношения

Найденный способ сведения решения исходной задачи к решению некоторых подзадач может быть записан в виде соотношений, в которых значение функции, соответствующей исходной задаче, выражается через значения функций, соответствующих подзадачам.

Рассмотрим следующий пример:

Вычислить сумму $S = 1 + 1/x + 1/x^2 + ... + 1/x^N$ при $x \neq 0$.

Как и в предыдущем примере можно записать следующее соотношение:

$S(i) = S(i - 1) + a(i)$, $i \geq 1$, где

$a(i) = 1/x^i$, $S(0) = 1$.

Конечно, можно и эти соотношения использовать для написания программы. При этом у нас возникла новая задача – найти способ вычисления $a(i)$. Для этого можно воспользоваться тем же приемом – попытаться вычислить $a(i)$ через значение $a(i - 1)$. Соотношение между значениями $a(i)$ и $a(i - 1)$ имеет следующий вид:

$a(i) = a(i - 1)/x$, $i > 1$, $a(0) = 1$

Поэтому поставленную задачу можно решить следующим образом.

```
S[0]: = 1; a[0]: = 1;
for i:=1 to N do                                    (2)
begin
 a[i]: = a[i – 1]/x;
 S[i]: = S[i – 1] + a[i]
end;
```

Задачи для самостоятельного решения:

Написать и реализовать рекуррентные соотношения для вычисления следующих задач:

- вычислить $\sum_{i=1}^{n} \frac{(-1)^i x^i}{i!}$

для i=1,…,n;

решение

$$C_n^m = \frac{n!}{((n-m)!m!)}$$

где n!=1*2*…*n, 0!=1;

Соотношения, связывающие одни и те же функции, но с различными аргументами, называются ***рекуррентными соотношениями*** или ***рекуррентными уравнениями***.

Использование таблиц при решении подзадач. Метод динамического программирования.

Важнейшим моментом при решении задачи является способ сведения задачи к подзадачам. Но не менее важным вопросом является и способ построения решения исходной задачи из решений подзадач. Одним из наиболее эффективных способов построения решения исходной задачи является использование таблиц для запоминания решений подзадач. Такой метод решения задач называется ***методом динамического программирования***.

Задача может быть формализована в виде функции, которая зависит от одного или нескольких аргументов. Если взять таблицу, у которой количество элементов равно количеству всех возможных различных наборов аргументов функции, то каждому набору аргументов может быть поставлен в соответствие элемент таблицы. Вычислив элементы таблицы (решения подзадач), можно найти и решение исходной задачи.

Одним из способов организации таблиц является такой подход, когда размерность таблицы определяется количеством аргументов у функции, соответствующей подзадаче.

Рассмотрим примеры:

1. В заданной числовой последовательности A[1..N] определить максимальную длину последовательности подряд идущих одинаковых элементов.

Пусть L(i) обозначает максимальную длину последователь-ности, последним элементом которой является элемент с номером i. Тогда значение L(i+1) может быть либо на 1 больше L(i), если элементы A(i+1) и A(i) равны, либо L(i+1) будет равно1, так как перед элементом с номером i+1 стоит отличный от него элемент. Максимальное значение $L(i)_{i=1,\ldots,n}$ и соответствует решению задачи.

```
L[1]: = 1;
For  i:=2 to N do
      if A[i-1]: = A[i]  then
       L[i]:=L[i-1]+1
     else
       L[i]:=1;
   IndL:=1;
For  i:=2 to N do
      if L[i]> L[IndL]  then
       IndL:=i;

 Max_I:=L[IndL];
```

A(n): 1,0,0,2,2,2,1,1,1,0,2,2,2,2,1,1

L(n):

1	1	2	1	2	3	1	2	3	1	1	2	3	4	1	2

2. Определить, сколькими различными способами можно подняться на 10-ю ступеньку лестницы, если за один шаг можно подниматься на следующую ступеньку или через одну.

Решение. Пусть $K(10)$ –количество способов подъема на 10 ступеньку. Определим подзадачу $K(i)$ нашей задачи как количество способов подъема на i-ю ступеньку.

Исходя из условия задачи, на 10 ступеньку можно подняться непосредственно с 8-й и 9-й ступенек. Поэтому, если мы знаем количество способов подъема $K(8)$ и $K(9)$ на 8 и 9 ступеньки соответственно, то количество способов подъема на 10 ступеньку может быть определено как $K(10) = K(8) + K(9)$.

Такое соотношение получается потому, что любой способ подъема на 8-ю ступеньку превращается в способ подъема на 10-ю ступеньку добавлением перешагивания через 9-ю ступеньку, а любой способ подъема на 9-ю ступеньку превращается в способ подъема на 10-ю ступеньку добавлением подъема с 9 на 10-ю ступеньку. Все эти способы различны. Аналогичное соотношение справедливо для любой ступеньки i, начиная с третьей, т.е.

$$K(i) = K(i - 2) + K(i - 1).$$

Осталось определить значения $K(1)$ и $K(2)$,

которые равны: $K(1) = 1$, $K(2) = 2$.

Следовательно, для решения задачи достаточно одномерной таблицы с 10 – ю элементами, для которой необходимо последовательно вычислить значения элементов таблицы согласно приведенным выше рекуррентным соотношениям. Для одномерной таблицы таким способом обычно является последовательное вычисление элементов, начиная с первого.

```
K[1]: = 1;
K[2]: = 2;
For  i:=3 to 10   do
 K[i]: = K[i – 1] + K[i – 2]
```

Таким образом, ***размерность таблицы, достаточная для реализации рекуррентных соотношений, определяется количеством аргументов у функций, соответствующих подзадачам. Количество же элементов по каждой размерности (количество элементов в строках, столбцах) определяется количеством возможных значений соответствующего аргумента.***

Алгоритм динамического программирования

Динамическое программирование – ***метод оптимизации, приспособленный к задачам, в которых требуется построить решение с максимизацией или минимизацией, т.е. оптимальным значением некоторого параметра.***

Его алгоритм можно сформулировать так:

1. Выделить и описать подзадачи, через решение которых будет выражаться искомое решение;
2. Выписать рекуррентные соотношения (уравнения), связывающие оптимальные значения параметра для всех подзадач;
3. Вычислить оптимальное значение параметра для всех подзадач;
4. Построить само оптимальное решение.

В задачах на подсчет количеств допустимых вариантов (задачи рассмотрены выше) пункт 4 не нужен

Принцип оптимальности (принцип Беллмана):

Каково бы ни было начальное состояние на любом шаге и решение, выбранное на этом шаге, последующие решения должны выбираться оптимальными относительно состояния, к которому придет система в конце данного шага.

Использование этого принципа гарантирует, что решение, выбранное на любом шаге, является не локально лучшим, а лучшим с точки зрения задачи в целом.

Данный метод усовершенствует решение задач, решаемых, например, с помощью рекурсий или перебора вариантов.

Условия применения динамического программирования

1. Оптимальное решение задачи выражается через оптимальное решение задач меньшей размерности, представимых в виде подзадач (подпрограмм). Улучшая решение подзадач, тем самым, улучшается решение общей задачи.

2. Небольшое число подзадач, что позволяет хранить решения каждой подзадачи в таблице. Уменьшение числа подзадач происходит из-за многократного их повторения(т.н. перекрывающиеся подзадачи).

3. Дискретность (неделимость) величин, рассматриваемых в задаче.

4. Один из главных критериев, когда принцип ДП дает эффект уменьшения временной сложности: если в процессе решения необходимо многократно перебирать одни и те же варианты (за счет увеличения емкостной сложности уменьшается временная сложность).

Вычислить сумму:

$$\sum_{i=1}^{n} \frac{(-1^i)x^i}{i!}$$

Решение:

Функция S(N) соответствует решению исходной задачи.

Воспользовавшись вышеописанным приемом получим соотношения:

S(0)=0; S(i)=S(i-1)+a(i)/b(i), где
a(0)=1; a(i)=(-1)*a(i-1)*x,
b(0)=1; b(i)=b(i-1)*I, при i>=1
;

```
Begin
. . .
a[0]:=1; b[0]:=1; S[0]:=0;
for i:=1 to N do begin
a[i]:=(-1)*a[i-1]*x;
b[i]:=b[i-1]*i;
S[i]:=S[i-1]+a[i]/b[i];

…

End;
```

назад

Найти значение:

$$C_n^m = \frac{n!}{(n-m)!m!}$$

Решение:

Функция C(N,M) соответствует решению исходной задачи.

Запишем соотношение:

C(i,j)=C(i-1,j-1)*(i/k*j);
i=1..n, j=1..m, k=1…(n-m)
C(0,0)=1;

C(i,j)=NF(i)/(NMF(i,j)*MF(j))
NF(0)=1; MF(0,0)=1;MF(0)=1;
NF(i)=NF(i-1)*I, i>=1
MF(j)=MF(j-1)*j; j>=1
NMF(i,j)=NMF(i-1,j-1)*(i-j)

```
Begin …
NF[0]:=1;MF[0]:=1;NMF[0]=1;
for i:=1 to N do NF[i]:=NF[i-1]*i;
for j:=1 to M do MF[j]:=MF[j-1]*j;
for i:=1 to N-M do NMF[i]:=NMF[i-1]*i;
C[N,M]=NF[n]/NMF[N-M]*MF[M]
… end.
```

Решение:

- нахождения произведения *N* элементов таблицы *A*;
- нахождения максимума *N* элементов таблицы *A*.

Пусть функция P(*N*) соответствует решению нашей исходной задачи. Эта функция имеет один аргумент *N* – количество умножаемых элементов таблицы *A*

Решение исходной задачи можно записать в виде соотношения

P(*N*) = P(*N* – 1) * *a[N]*, N>=1

Это соотношение можно переписать в виде
P(*i*) = *P*(*i* – 1)* *a[i]*, при *i*> 1, P(0) = 1.

Вычисление функции производится от меньших значений аргументов к большим.

```
P[0]: = 1;
for i:= 1 to N do P[i]: = P[i – 1] * a[i];
```

В P[*i*] хранится значение функции *P*(i).

Пусть Max_l(N) соответствует решению исходной задачи.

Решение исходной задачи можно записать в виде соотношения:

Indl=1;

Maxl(N)=max(a[i],a[indl]), i>=2

Вычисление функции:

```
Indl:=1;
For i:=2 to N do
If a[i]>a[Indl] Then
Indl:=I;
Maxl:=a[Indl];
```

Литература:

1. Котов В.М., Уроки по динамическому программированию, «Информатика и образование», №8-2001

2. Котов В.М. Динамическое программирование