

Programowanie dynamiczne

sobota, 1 czerwca 2024 13:04

Dane: $n, k \in \mathbb{N}$ $n \geq k$

Zadanie oblicz $\binom{n}{k}$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}$$

wzrost
danych

$n+k$

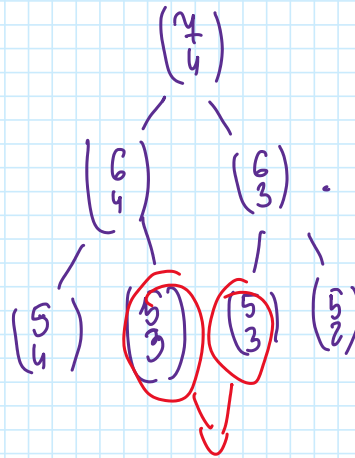
$n+k-2$

$n+k-1$

rekurencja

```

n_pok(n, k)
if (n == k) v (k == 0)
    return 1
return n_pok(n-1, k) + n_pok(n-1, k-1)
    
```



dużo wywołania w
dwóch ścieżkach
rekurencyjnych dla tych
samych argumentów \rightarrow tego
nie chcemy

jak zrobić lepiej?

tablice na przechowywanie wyników - wtedy nie będziemy
wielokrotnie wywoływać tej samej funkcji

```

for i=1 to n do
    for j=0 to k do tab[i,j] ← "?"
    .....
function nPOK(n, k)
    if k = n or k = 0 then tab[k,n] ← 1; return 1;
    if tab[n-1,k-1] = "?" then tab[n-1,k-1] ← nPOK(n-1, k-1)
    if tab[n-1,k] = "?" then tab[n-1,k] ← nPOK(n-1, k)
    tab[n,k] = tab[n-1,k-1] + tab[n-1,k]
    return tab[n,k]
    
```

Rekurencyjne obliczanie $\binom{n}{k}$ z użyciem spamiętywania

ale to nadal
jest rekurencyjne
metoda "top-down"

można zrobić jeszcze lepiej - iteracyjnie: $O(n^2)$

będziemy jednak od problemów najprostszych do najtrudniejszych nie
zastanawiając się czy się nam przyda, czy nie.

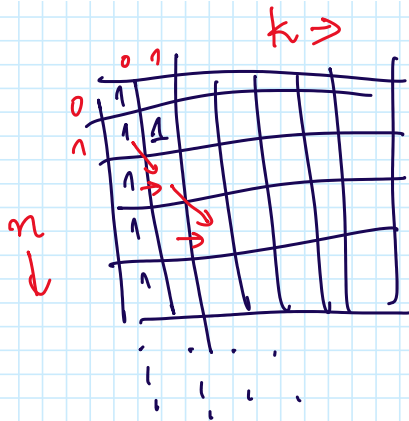
```

for i = 1 to n do tabi,0 ← 1
.....
function nPOk(n, k)
  for j = 1 to k do
    tabj,j ← 1
    for i = j + 1 to n do tabi,j ← tabi-1,j-1 + tabi-1,j
  return tabn,k

```

Obliczanie $\binom{n}{k}$ metodą programowania dynamicznego

myślący kolumnami



kompilujemy z dwóch
przebiegów policzenia
wartości

i musimy to zrobić tyłem
we dwóch ostatnich
kolumnach

jak naprawdę sobie interesuję was

dużo ostatnie wiem, a nie całą tablicę
↓
mnie się zapamięci przedostatnie wiem po obliczeniu
ostatniego i wziąć się za kolejny

— — — — —

7	2	3	2	10	21
4	4	8	4	11	22
9	5	9	7	7	20
6	1	20	6	2	19
6	6	15	4	8	23



$i-1, k-1$ $i, k-1$ $i+1, k-1$
kost: suma liczb połączonych przedostatnich

liczba możliwych $\Omega(2^n)$ (bardziej 3^n)

Om. dla - minimalny koszt osiąga do

Om. d_{ik} - minimalny koszt drogi do pola (i, j)

d_{i1} to pierwszy element
kolumny w kolumnie numer 1

$$\begin{cases} d_{i1} = a_{i1} \\ d_{ij} = \min \{ d_{i-1, k-1}, d_{i, k-1}, d_{i+1, k-1} \} + a_{ij} \end{cases}$$

W kolumnach od 2 do m wybieramy jedno z trzech sąsiadujących pole w poprzedniej kolumnie i dodajemy do komórki

W pierwszej kolumnie min wartość to to co jest w tej komórce

```
for j = 1 to m do  $d_{0,j} \leftarrow d_{n+1,j} \leftarrow \infty$ 
for i = 1 to n do  $d_{i,1} \leftarrow a_{i,1}$ 
for j = 2 to m do
  for i = 1 to n do  $d_{i,j} \leftarrow a_{i,j} + \min \{ d_{i-1,j-1}, d_{i,j-1}, d_{i+1,j-1} \}$ 
return  $\min \{ d_{i,m} \mid i = 1, \dots, n \}$ 
```

Pozostaje wyjaśnić, w jaki sposób można odtworzyć optymalną trasę. Niech i_0 będzie wartością i , dla której osiągane jest $\min \{ d_{i,m} \mid i = 1, \dots, n \}$, a więc $a_{i_0,m}$ jest ostatnim polem optymalnej trasy. Aby wyznaczyć przedostatnie pole wystarczy sprawdzić, która z trzech wartości $d_{j,m-1}$ (dla $j \in \{i_0 - 1, i_0, i_0 + 1\}$) jest minimalna. Postępując dalej rekurencyjnie wyznaczymy całą trasę.

column

```
procedure trasa( $i, j$ )
{
  if  $j = 1$  then return  $i$ 
  if  $d_{i-1,j-1} < d_{i,j-1}$  then  $k \leftarrow i - 1$  else  $k \leftarrow i$ 
  if  $d_{i+1,j-1} < d_{i,j-1}$  then  $k \leftarrow i + 1$ 
  return concat(trasa( $k, j - 1$ ),  $i$ )
}
.....
write(trasa( $i_0, m$ ))
```

podciąg ciągu $A \Rightarrow$ wybieramy jakiś literę i to co otrzymało możemy do niego dodać
 czyli sam ciąg $A = a_1 a_2 \dots a_n$ jest swoim podciągiem
 ale również samo $a_1 a_n$ to jego podciąg

Dane: $X = \langle x_1, \dots, x_n \rangle$
 $Y = \langle y_1, \dots, y_m \rangle$

Wynik: $Z = \langle z_1, \dots, z_k \rangle$ *taki i.e*

Z jest podciągiem X


Z jest podciągiem Y

\hookrightarrow maksymalizuj długość

X -owy podciąg

x_i - i -literowy prefix podciągu

x_i - i -ty element należący do podciągu

X  $X = x_1 \dots x_n$

Y  $Y = y_1 \dots y_m$

2 przypadki rozważamy:

1° $x_n = y_m$

kiedy LCS kończy się
literą x_n

możemy to zredukować do
znalezienia LCS dla x_{n-1} i y_{m-1}

2° $x_n \neq y_m$

$X = x_1 \dots x_{n-1}$

$Y = y_1 \dots y_{m-1}$

Możemy mieć, to redukuje

*nie musimy
realizować go
ponownie, to
zależy do
rozpatrywania
zobacz*

możemy to zredukować do
znalezienia LCSa dla X_{m-1} i Y_{m-1}

Możemy mieć ty redukcje?

$d_{i,j}$ - długość elementów z LCS (X_i, Y_j)

$$d_{i,j} = \begin{cases} 0, & i=0 \vee j=0 \\ 1 + d_{i-1,j-1}, & i,j > 0, x_i = y_j \\ \max(d_{i,j-1}, d_{i-1,j}), & i,j > 0, x_i \neq y_j \end{cases}$$

jeżeli? długość pustego

czyli realnie dwie
i sprawdzamy które dla
danego LCSa

złożoność: $O(n \cdot m)$

↓
kwalifikacja

Przykład:

$X = \text{BAZAR}$

$Y = \text{BURXA}$

$d_{i,j}$ dla $0 \leq i \leq 5$
 $0 \leq j \leq 5$

		B	A	Z	A	R	
		0	1	2	3	4	5
	0	0	0	0	0	0	0
B	1	0	1	1	1	1	1
V	2	0	1	1	1	1	1
R	3	0	1	1	1	1	2
Z	4	0	1	1	2	2	2
A	5	0	1	2	2	3	3

B A Z A R
 B
 V
 R
 Z
 A

B A Z A R

B Z A

ciągło: $O(n \cdot m)$
Bazie: $O(n \cdot m)$

LCS - pseudokod

for $i \in 0$ to m $d_{i,0} \leftarrow 0$

for $j \in 0$ to m $d_{0,j} \leftarrow 0$

for $i \in 1$ to m

for $j \in 1$ to m

for $j = 1$ to n

if $x_i = y_j$

$$d_{ij} \leftarrow 1 + d_{i-1, j-1}$$

else

$$d_{ij} \leftarrow \max \{ d_{i-1, j}, d_{i, j-1} \}$$