Algorytmy wyznaczania wartości symbolu Newtona

Piotr Beling

Uniwersytet Łódzki

2020 (aktualizacja: 2021)

http://pbeling.w8.pl

Symbol Newtona (współczynnik dwumianowy Newtona)
dla liczb naturalnych $0 \leq k \leq n$, definiujemy

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

i czytamy: ***n po k***, *n nad k*, lub *k z n*.
(gdzie $x! = 1 \cdot 2 \cdot \ldots \cdot x$, zaś $0! = 1$; czytamy *x silnia*)

Wartość $\binom{n}{k}$ równa jest **liczbie k-elementowych podzbiorów zbioru n-elementowego**.

W szczególności:

- $\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{n!} = 1$ (jest 1 podzbiór pusty);
- $\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n!} = 1$ (1 podzbiór będący całym zbiorem).

Symbol Newtona (współczynnik dwumianowy Newtona)
dla liczb naturalnych $0 \leq k \leq n$, definiujemy

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

i czytamy: ***n po k***, *n nad k*, lub *k z n*.
(gdzie $x! = 1 \cdot 2 \cdot \ldots \cdot x$, zaś $0! = 1$; czytamy *x silnia*)

Wartość $\binom{n}{k}$ równa jest **liczbie *k*-elementowych podzbiorów zbioru *n*-elementowego**.

W szczególności:

- $\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{n!} = 1$ (jest 1 podzbiór pusty);
- $\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n!} = 1$ (1 podzbiór będący całym zbiorem).

Symbol Newtona (współczynnik dwumianowy Newtona)
dla liczb naturalnych $0 \leq k \leq n$, definiujemy

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

i czytamy: ***n po k***, *n nad k*, lub *k z n*.
(gdzie $x! = 1 \cdot 2 \cdot \ldots \cdot x$, zaś $0! = 1$; czytamy *x silnia*)

Wartość $\binom{n}{k}$ równa jest **liczbie k-elementowych podzbiorów zbioru n-elementowego**.

W szczególności:

- $\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{n!} = 1$ (jest 1 podzbiór pusty);
- $\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n!} = 1$ (1 podzbiór będący całym zbiorem).

Symbol Newtona (współczynnik dwumianowy Newtona)
dla liczb naturalnych $0 \leq k \leq n$, definiujemy

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} = \frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!}$$

i czytamy: ***n po k***, *n nad k*, lub *k z n*.
(gdzie $x! = 1 \cdot 2 \cdot \ldots \cdot x$, zaś $0! = 1$; czytamy *x silnia*)

Wartość $\binom{n}{k}$ równa jest **liczbie *k*-elementowych podzbiorów zbioru *n*-elementowego**.

W szczególności:

- $\binom{n}{0} = \frac{n!}{0!(n-0)!} = \frac{n!}{n!} = 1$ (jest 1 podzbiór pusty);
- $\binom{n}{n} = \frac{n!}{n!(n-n)!} = \frac{n!}{n!} = 1$ (1 podzbiór będący całym zbiorem).

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Obliczanie symbolu Newtona wprost z definicji

- Wartość $\binom{n}{k}$ można wyznaczyć wprost z definicji

$$\frac{n \cdot (n-1) \cdot \ldots \cdot (n-k+1)}{k!},$$

- wykonując $2k$ mnożeń i 1 dzielenie
- i używając stałej ilości pamięci.
- Jednakże wyniki pośrednie mogą nie zmieścić się w zakresach używanych typów (słów maszynowych),
- nawet gdy zmieściłyby się w nich sam wynik.
- Licznik w powyższym wzorze jest $k!$ razy większy od wyniku.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k<k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k < k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k<k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k < k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k < k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Redukcja wartości k

Zachodzi:

$$\binom{n}{n-k} = \frac{n!}{(n-k)!(n-(n-k))!} = \frac{n!}{(n-k)!k!} = \binom{n}{k}$$

Co jest zgodne z intuicją, bo:

- z dowolnym, k-elementowym podzbiorem P,
- dowolnego, n-elementowego zbioru U,
- związany jest zbiór $U \setminus P$, mający $n-k$ elementów.

Zamiast więc obliczać $\binom{n}{k}$, można obliczyć $\binom{n}{n-k}$, co warto zrobić gdy $n-k < k$; tj. zawsze liczmy $\binom{n}{\min(k,n-k)}$.

Obliczanie symbolu Newtona z zależności $\binom{n}{k} = n\binom{n-1}{k-1}/k$

Problem przekraczania zakresów można zredukować, korzystając z jednej z (rekurencyjnych) zależności:

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases} = \begin{cases} 1 & \text{dla } k = 0 \\ (n-k+1)\binom{n}{k-1}/k & \text{dla } k > 0 \end{cases}$$

Które dla $k > 0$ zachodzą, bo: $\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot\ldots\cdot 1} =$

- $\frac{n}{k} \cdot \frac{(n-1)\cdot\ldots\cdot(n-k+1)}{(k-1)\cdot\ldots\cdot 1} = \frac{n}{k} \cdot \binom{n-1}{k-1}$
- $\frac{n-k+1}{k} \cdot \frac{n\cdot\ldots\cdot(n-k+2)}{(k-1)\cdot\ldots\cdot 1} = \frac{n-k+1}{k} \cdot \binom{n}{k-1}$

Tym razem:

- największy wynik pośredni jest k razy większy od końcowego,
- zaś algorytm wykonuje k mnożeń i k dzieleń.

Obliczanie symbolu Newtona z zależności $\binom{n}{k} = n\binom{n-1}{k-1}/k$

Problem przekraczania zakresów można zredukować, korzystając z jednej z (rekurencyjnych) zależności:

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases} = \begin{cases} 1 & \text{dla } k = 0 \\ (n-k+1)\binom{n}{k-1}/k & \text{dla } k > 0 \end{cases}$$

Które dla $k > 0$ zachodzą, bo: $\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot\ldots\cdot 1} =$

- $\frac{n}{k} \cdot \frac{(n-1)\cdot\ldots\cdot(n-k+1)}{(k-1)\cdot\ldots\cdot 1} = \frac{n}{k} \cdot \binom{n-1}{k-1}$
- $\frac{n-k+1}{k} \cdot \frac{n\cdot\ldots\cdot(n-k+2)}{(k-1)\cdot\ldots\cdot 1} = \frac{n-k+1}{k} \cdot \binom{n}{k-1}$

Tym razem:

- największy wynik pośredni jest k razy większy od końcowego,
- zaś algorytm wykonuje k mnożeń i k dzieleń.

Obliczanie symbolu Newtona z zależności $\binom{n}{k} = n\binom{n-1}{k-1}/k$

Problem przekraczania zakresów można zredukować, korzystając z jednej z (rekurencyjnych) zależności:

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases} = \begin{cases} 1 & \text{dla } k = 0 \\ (n-k+1)\binom{n}{k-1}/k & \text{dla } k > 0 \end{cases}$$

Które dla $k > 0$ zachodzą, bo: $\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot\ldots\cdot 1} =$

- $\frac{n}{k} \cdot \frac{(n-1)\cdot\ldots\cdot(n-k+1)}{(k-1)\cdot\ldots\cdot 1} = \frac{n}{k} \cdot \binom{n-1}{k-1}$
- $\frac{n-k+1}{k} \cdot \frac{n\cdot\ldots\cdot(n-k+2)}{(k-1)\cdot\ldots\cdot 1} = \frac{n-k+1}{k} \cdot \binom{n}{k-1}$

Tym razem:

- największy wynik pośredni jest k razy większy od końcowego,
- zaś algorytm wykonuje k mnożeń i k dzieleń.

Obliczanie symbolu Newtona z zależności $\binom{n}{k} = n\binom{n-1}{k-1}/k$

Problem przekraczania zakresów można zredukować, korzystając z jednej z (rekurencyjnych) zależności:

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases} = \begin{cases} 1 & \text{dla } k = 0 \\ (n-k+1)\binom{n}{k-1}/k & \text{dla } k > 0 \end{cases}$$

Które dla $k > 0$ zachodzą, bo: $\binom{n}{k} = \frac{n\cdot(n-1)\cdot\ldots\cdot(n-k+1)}{k\cdot(k-1)\cdot\ldots\cdot 1} =$

- $\frac{n}{k} \cdot \frac{(n-1)\cdot\ldots\cdot(n-k+1)}{(k-1)\cdot\ldots\cdot 1} = \frac{n}{k} \cdot \binom{n-1}{k-1}$
- $\frac{n-k+1}{k} \cdot \frac{n\cdot\ldots\cdot(n-k+2)}{(k-1)\cdot\ldots\cdot 1} = \frac{n-k+1}{k} \cdot \binom{n}{k-1}$

Tym razem:

- największy wynik pośredni jest k razy większy od końcowego,
- zaś algorytm wykonuje k mnożeń i k dzieleń.

Pseudokod algorytmu opartego o zależności: $\binom{n}{k} = \binom{n}{\min(k,n-k)}$

oraz $\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases}$:

```
fun newton_rek(n, k):
    if k = 0: return 1
    return n * newton_rek(n-1, k-1) / k

fun newton(n, k):
    return newton_rek(n, min(k, n-k))
```

- Złożoność pamięciowa `newton`: $\Theta(\min(k, n-k))$; ze względu na $\min(k, n-k)$ wywołań rekurencyjnych `newton_rek`;
- Można zaimplementować iteracyjny odpowiednik funkcji `newton_rek` o złożoności pamięciowej $\Theta(1)$, która w i-tej iteracji $(i = 1, \ldots, k)$ wyznacza:

$$W_i = (n - k + i)W_{i-1}/i,$$

gdzie $W_0 = 1$ i $\binom{n}{k} = W_k$.

Pseudokod algorytmu opartego o zależności: $\binom{n}{k} = \binom{n}{\min(k,n-k)}$

oraz $\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases}$:

```
fun newton_rek(n, k):
    if k = 0: return 1
    return n * newton_rek(n-1, k-1) / k

fun newton(n, k):
    return newton_rek(n, min(k, n-k))
```

- Złożoność pamięciowa `newton`: $\Theta(\min(k, n-k))$; ze względu na $\min(k, n-k)$ wywołań rekurencyjnych `newton_rek`;
- Można zaimplementować iteracyjny odpowiednik funkcji `newton_rek` o złożoności pamięciowej $\Theta(1)$, która w i-tej iteracji $(i = 1, \ldots, k)$ wyznacza:

$$W_i = (n - k + i)W_{i-1}/i,$$

gdzie $W_0 = 1$ i $\binom{n}{k} = W_k$.

Pseudokod algorytmu opartego o zależności: $\binom{n}{k} = \binom{n}{\min(k,n-k)}$

oraz $\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \\ n\binom{n-1}{k-1}/k & \text{dla } k > 0 \end{cases}$:

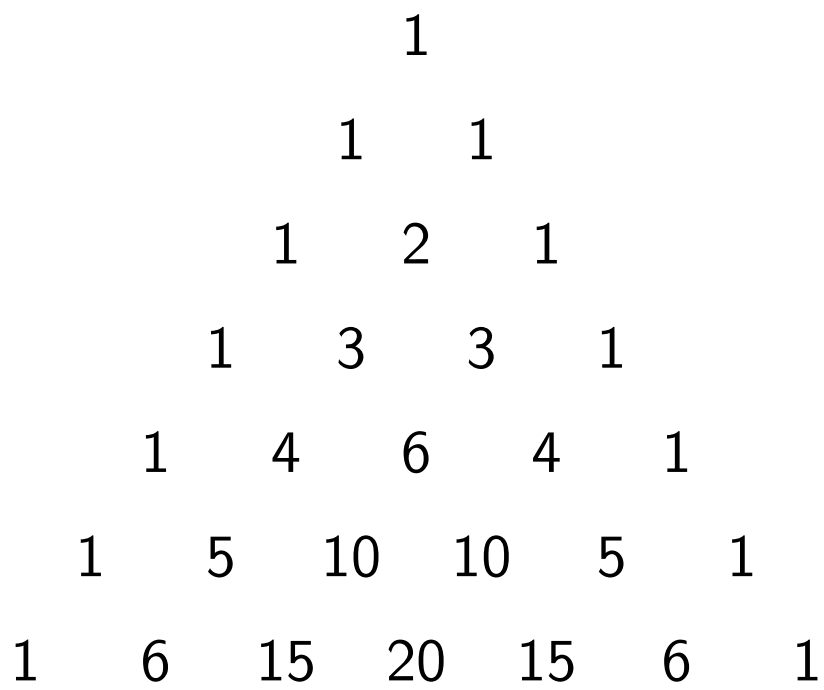
```
fun newton_rek(n, k):
    if k = 0: return 1
    return n * newton_rek(n-1, k-1) / k

fun newton(n, k):
    return newton_rek(n, min(k, n-k))
```

- Złożoność pamięciowa `newton`: $\Theta(\min(k, n-k))$; ze względu na $\min(k, n-k)$ wywołań rekurencyjnych `newton_rek`;
- Można zaimplementować iteracyjny odpowiednik funkcji `newton_rek` o złożoności pamięciowej $\Theta(1)$, która w i-tej iteracji $(i = 1, \ldots, k)$ wyznacza:

$$W_i = (n - k + i)W_{i-1}/i,$$

gdzie $W_0 = 1$ i $\binom{n}{k} = W_k$.

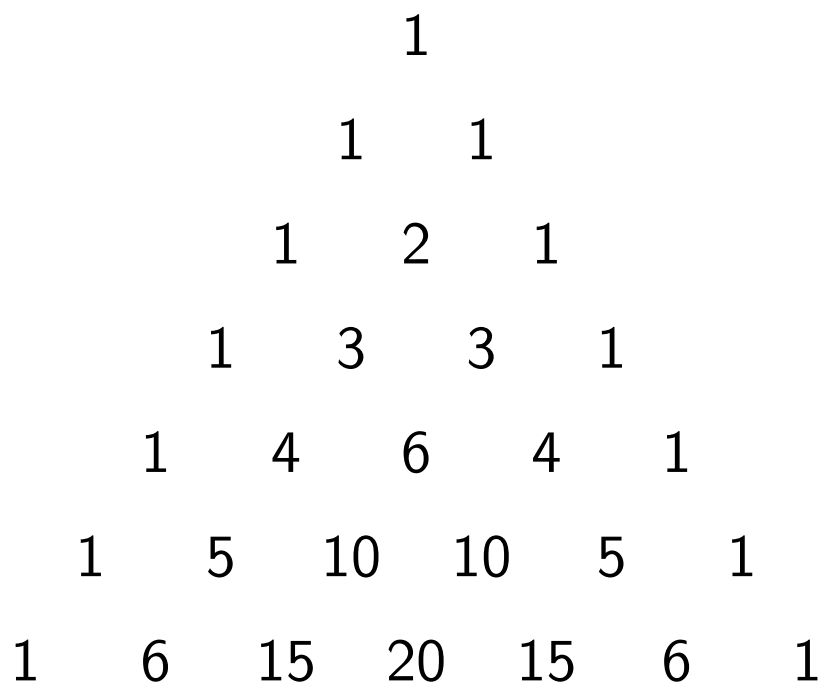
Trójkąt Pascala to popularna i efektywna metoda wyznaczania współczynników dwumianowych (czyli symboli) Newtona:



- na bokach trójkąta są liczby 1,
- w pozostałych miejscach sumy par liczb stojących bezpośrednio powyżej

- trójkąt zawiera współczynniki dwumianowe Newtona

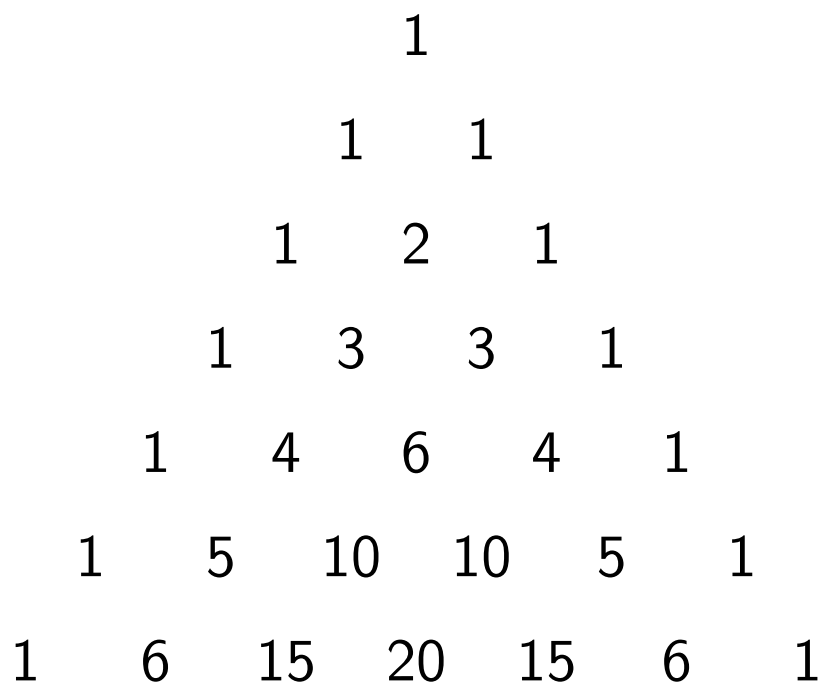
Trójkąt Pascala to popularna i efektywna metoda wyznaczania współczynników dwumianowych (czyli symboli) Newtona:



- na bokach trójkąta są liczby 1,
- w pozostałych miejscach sumy par liczb stojących bezpośrednio powyżej

- trójkąt zawiera współczynniki dwumianowe Newtona

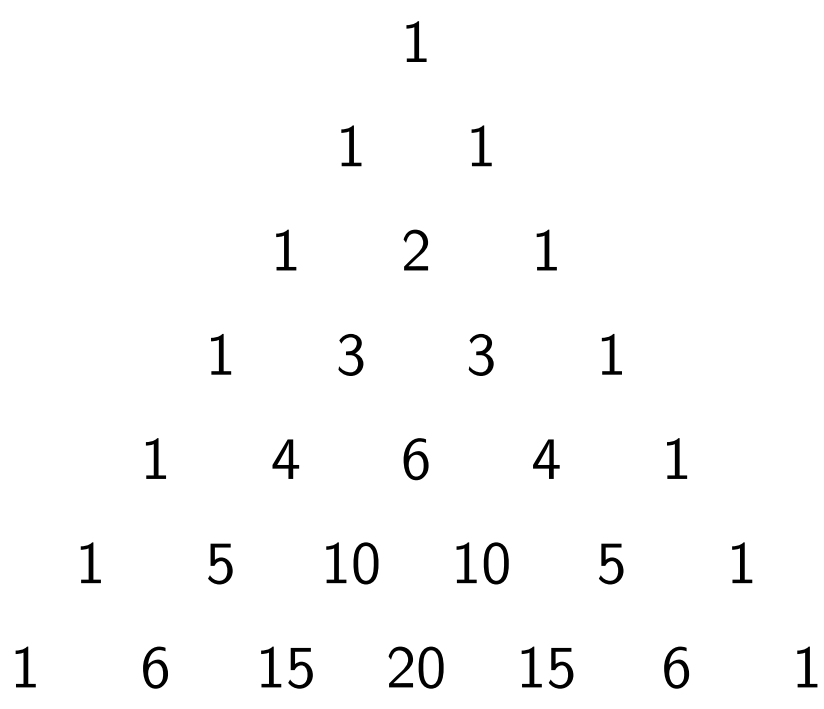
Trójkąt Pascala to popularna i efektywna metoda wyznaczania współczynników dwumianowych (czyli symboli) Newtona:

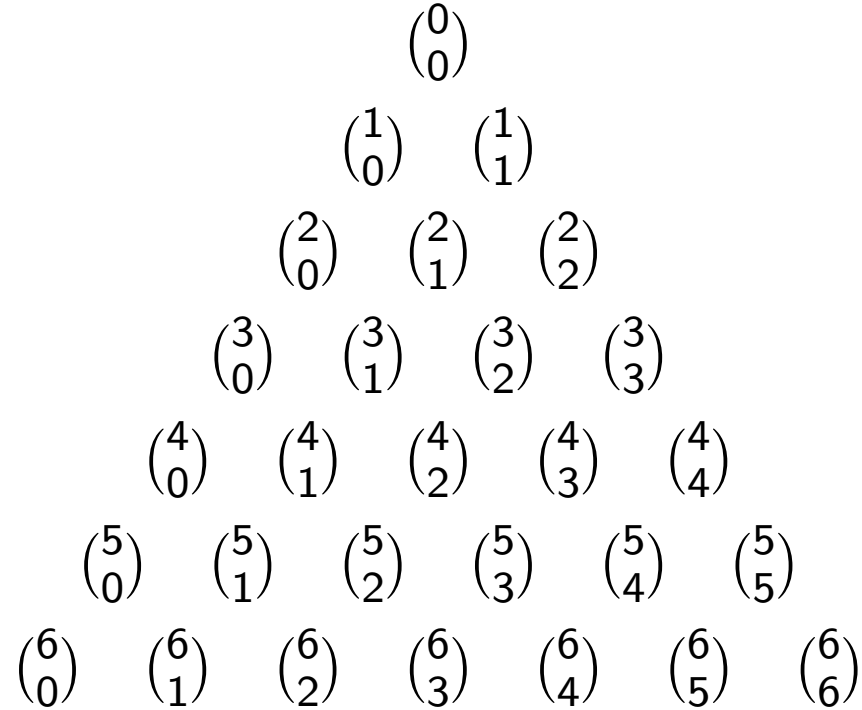


- na bokach trójkąta są liczby 1,
- w pozostałych miejscach sumy par liczb stojących bezpośrednio powyżej

- trójkąt zawiera współczynniki dwumianowe Newtona

Trójkąt Pascala to popularna i efektywna metoda wyznaczania współczynników dwumianowych (czyli symboli) Newtona:





- na bokach trójkąta są liczby 1,
- w pozostałych miejscach sumy par liczb stojących bezpośrednio powyżej

- trójkąt zawiera współczynniki dwumianowe Newtona

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
- każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
- podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
- każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
- podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
  - każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
  - podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
- każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
- podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
- każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
- podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Trójkąt Pascala – wzór

Trójkąt Pascala, dla $0 < k < n$, bazuje na następującej zależności:

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

Dowód poprawności:

- Niech U będzie dowolnym, niepustym, n-elementowym zbiorem, $e \in U$ dowolnym jego elementem, zaś $R = U \setminus \{e\}$ podzbiorem U składającym się z pozostałych $n-1$ elementów.
- Wszystkich k-elementowych podzbiorów U jest $\binom{n}{k}$; część z tych podzbiorów zawiera e, pozostałe nie:
- każdy podzbiór zawierający e składa się też z $k-1$ elementów tworzących podzbiór R; jest $\binom{n-1}{k-1}$ takich podzbiorów;
- podzbiory niezawierające e są k-elementowymi podzbiorami R, jest więc ich $\binom{n-1}{k}$.

Dla $k = 0$ lub $k = n$, trójkąt Pascala zawiera 1, bo $\binom{n}{0} = \binom{n}{n} = 1$.

Pseudokod algorytmu implementującego wprost zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(n, k):
    if k=0 or k=n: return 1
    return newton(n-1, k-1) + newton(n-1, k)
```

Zalety:

- wyniki pośrednie nie przekraczają wyniku ostatecznego;
- algorytm używa jedynie dodawania;
- złożoność pamięciowa $O(n)$ (dowód: każde kolejne wywołanie rekurencyjne następuje z n zmniejszonym o 1, co ogranicza głębokość rekurencji do n; gdy $n = 0$ to $k = 0$, bo $k \leq n$).

Wada: złożoność czasowa $\Omega(2^{\min(k,n-k)})$.

Pseudokod algorytmu implementującego wprost zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(n, k):
    if k=0 or k=n: return 1
    return newton(n-1, k-1) + newton(n-1, k)
```

Zalety:

- wyniki pośrednie nie przekraczają wyniku ostatecznego;
- algorytm używa jedynie dodawania;
- złożoność pamięciowa $O(n)$ (dowód: każde kolejne wywołanie rekurencyjne następuje z n zmniejszonym o 1, co ogranicza głębokość rekurencji do n; gdy $n = 0$ to $k = 0$, bo $k \leq n$).

Wada: złożoność czasowa $\Omega(2^{\min(k,n-k)})$.

Pseudokod algorytmu implementującego wprost zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(n, k):
    if k=0 or k=n: return 1
    return newton(n-1, k-1) + newton(n-1, k)
```

Zalety:

- wyniki pośrednie nie przekraczają wyniku ostatecznego;
- algorytm używa jedynie dodawania;
- złożoność pamięciowa $O(n)$ (dowód: każde kolejne wywołanie rekurencyjne następuje z n zmniejszonym o 1, co ogranicza głębokość rekurencji do n; gdy $n = 0$ to $k = 0$, bo $k \leq n$).

Wada: złożoność czasowa $\Omega(2^{\min(k,n-k)})$.

Pseudokod algorytmu implementującego wprost zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(n, k):
    if k=0 or k=n: return 1
    return newton(n-1, k-1) + newton(n-1, k)
```

Zalety:

- wyniki pośrednie nie przekraczają wyniku ostatecznego;
- algorytm używa jedynie dodawania;
- złożoność pamięciowa $O(n)$ (dowód: każde kolejne wywołanie rekurencyjne następuje z n zmniejszonym o 1, co ogranicza głębokość rekurencji do n; gdy $n = 0$ to $k = 0$, bo $k \leq n$).

Wada: złożoność czasowa $\Omega(2^{\min(k,n-k)})$.

Pseudokod algorytmu implementującego wprost zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

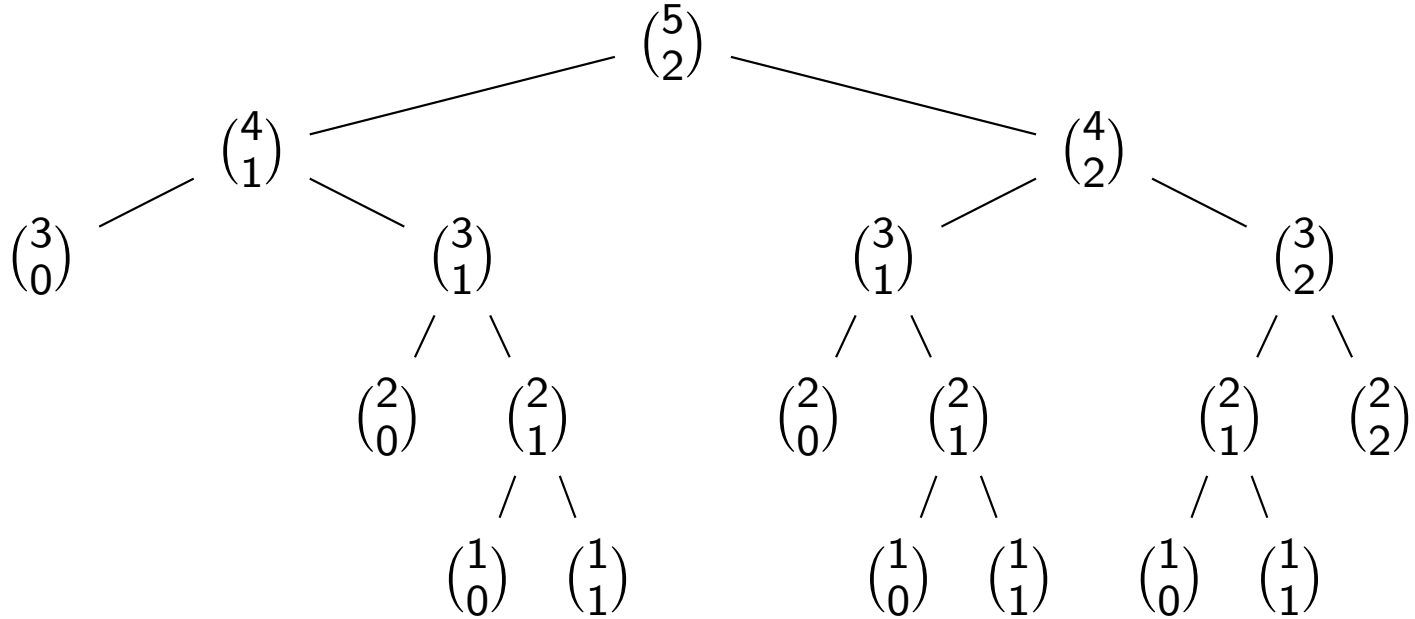
```
fun newton(n, k):
    if k=0 or k=n: return 1
    return newton(n-1, k-1) + newton(n-1, k)
```

Zalety:

- wyniki pośrednie nie przekraczają wyniku ostatecznego;
- algorytm używa jedynie dodawania;
- złożoność pamięciowa $O(n)$ (dowód: każde kolejne wywołanie rekurencyjne następuje z n zmniejszonym o 1, co ogranicza głębokość rekurencji do n; gdy $n = 0$ to $k = 0$, bo $k \leq n$).

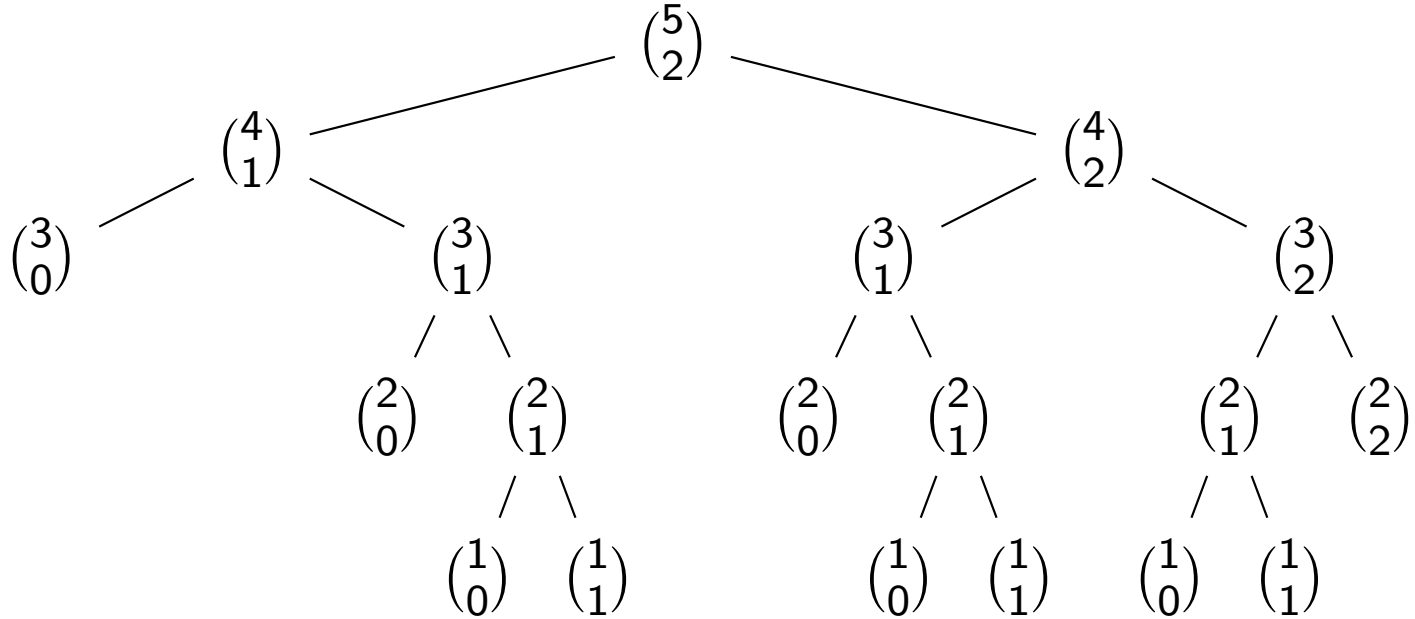
Wada: złożoność czasowa $\Omega(2^{\min(k,n-k)})$.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



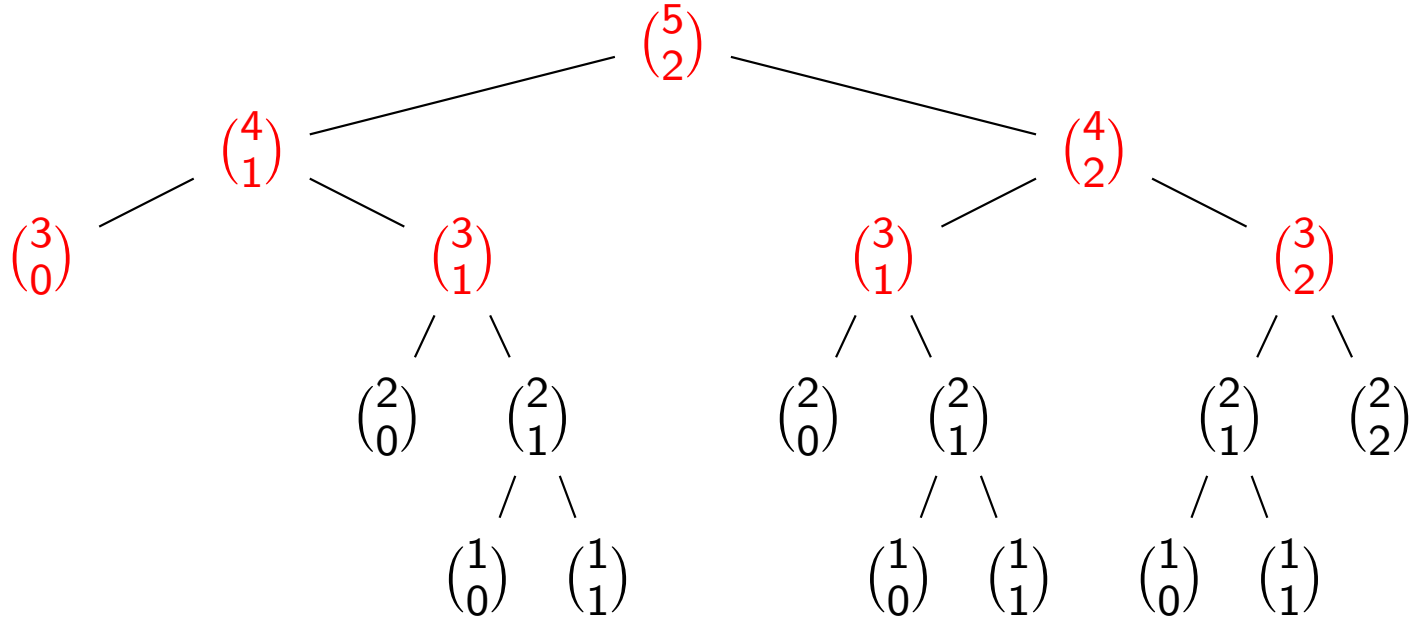
- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



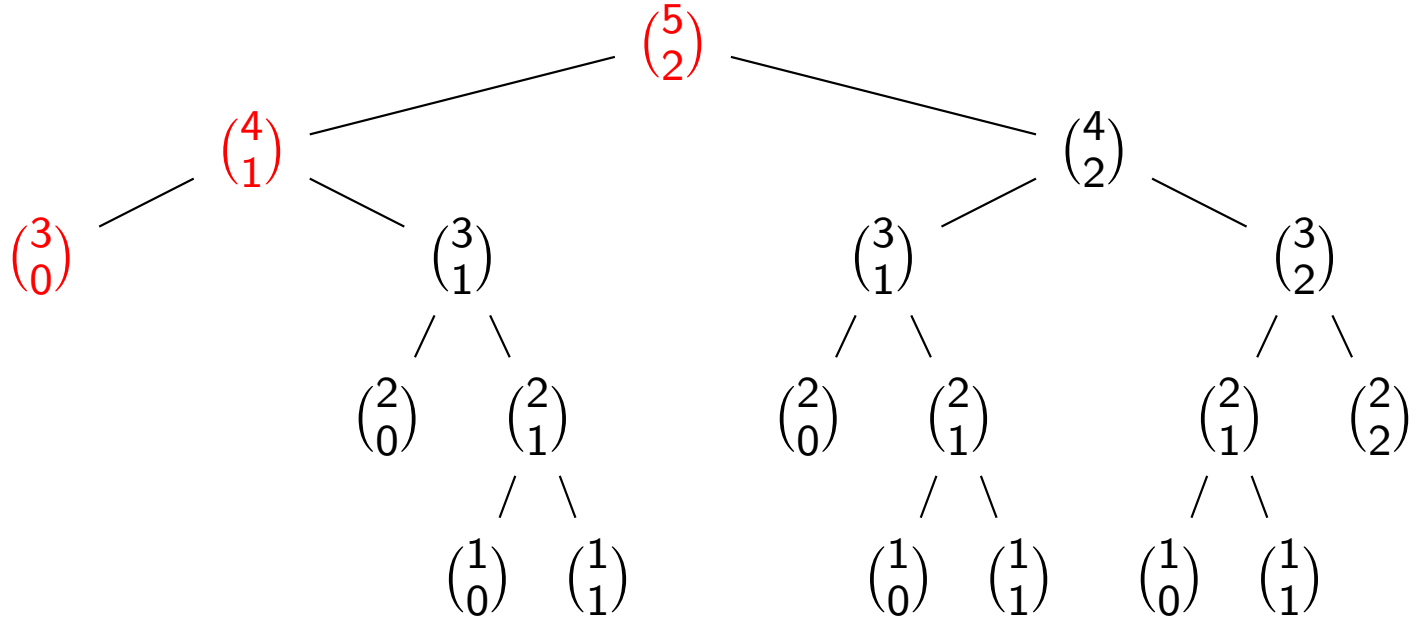
- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



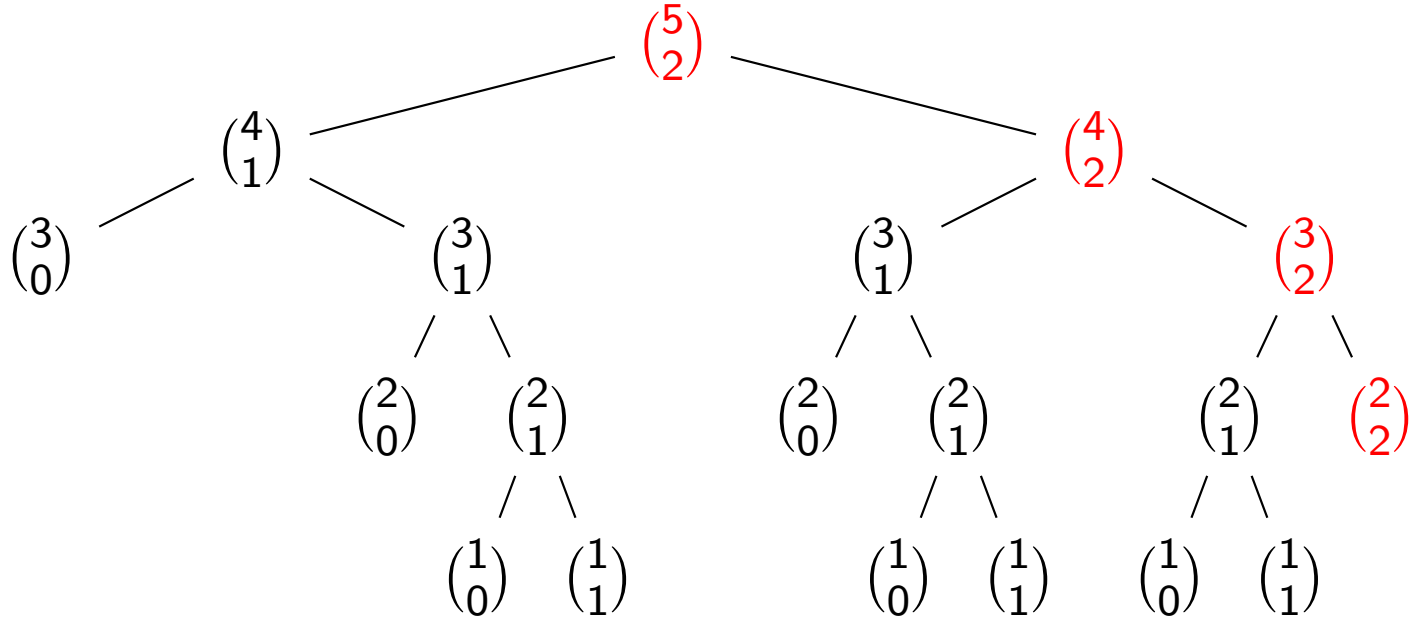
- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



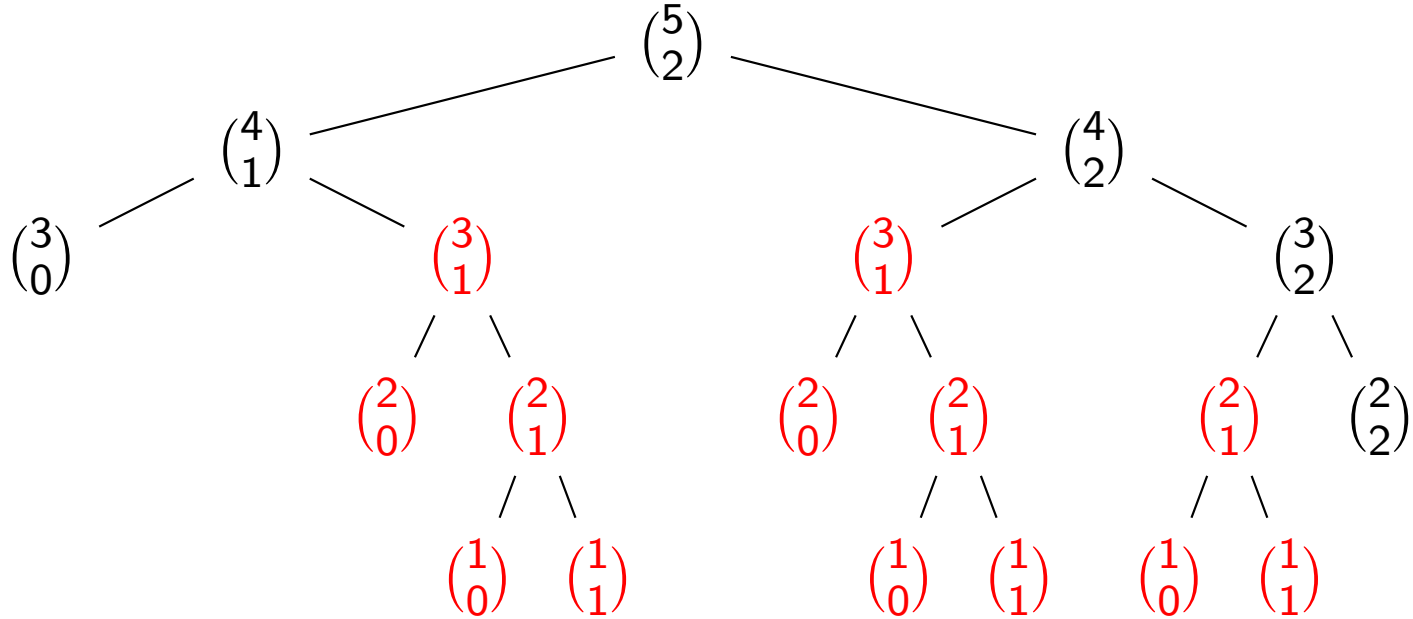
- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Przykładowe drzewo wywołań rekurencyjnych (obliczania $\binom{5}{2}$):



- Drzewo jest binarne, bo każde wywołanie niekończące rekursji (czyli dla $0 < k < n$), pociąga za sobą 2 kolejne.
- Drzewo dla $\binom{n}{k}$ jest pełne do poziomu $\min(k, n-k)$ (stąd złożoność czasowa $\Omega(2^{\min(k,n-k)})$) bo potrzeba najmniej:
- k wywołań by spełnić warunek $k = 0$; $\binom{n}{k}, \binom{n-1}{k-1}, \ldots, \binom{n-k}{0}$;
- $n-k$ wywołań by spełnić $n = k$; $\binom{n}{k}, \binom{n-1}{k}, \ldots, \binom{n-(n-k)}{k}$.
- Algorytm wielokrotne rozwiązuje te same podproblemy.

Programowanie dynamiczne

- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

Programowanie dynamiczne

- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

Programowanie dynamiczne

- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

Programowanie dynamiczne

- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

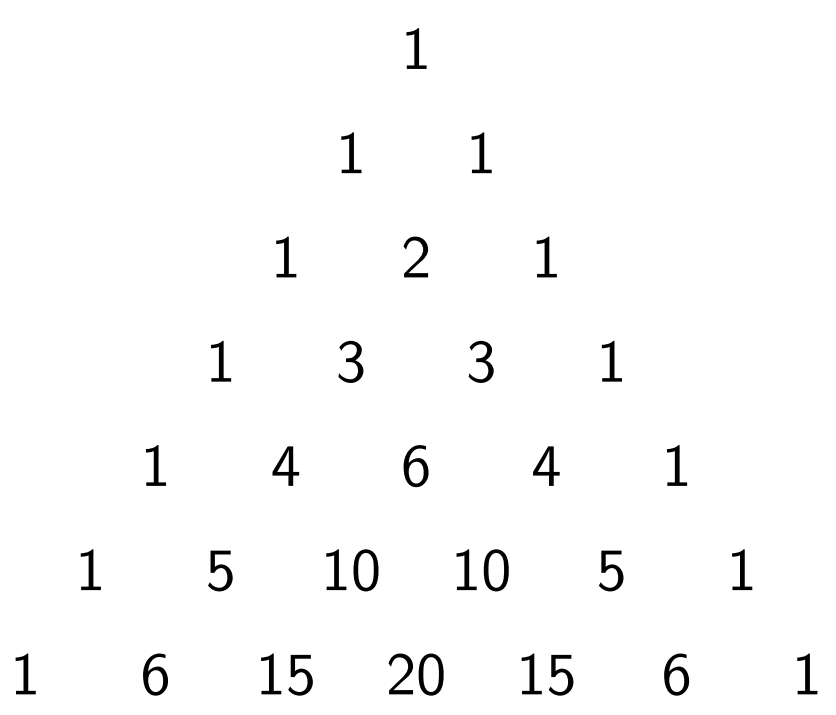
Programowanie dynamiczne

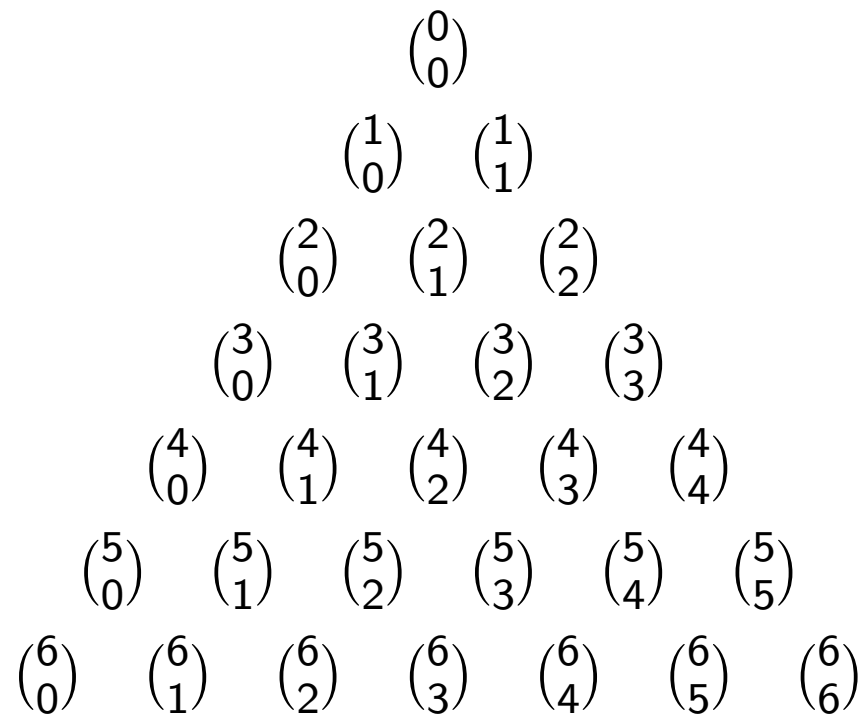
- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

Programowanie dynamiczne

- Do przyspieszania algorytmów, które wielokrotnie rozwiązują małą liczbę różnych podproblemów (i nie zachodzą zależności cykliczne pomiędzy podproblemami) możemy użyć technikę zwaną **programowaniem dynamicznym**.
- Programowanie dynamiczne unika wielokrotnego rozwiązywania podproblemów zapisując ich wyniki w pamięci.
- Podproblemy są rozwiązywane w takiej kolejności, że jeżeli wynik podproblemu A jest potrzebny do rozwiązania podproblemu B, to A jest rozwiązywany przed B
- zaś przy obliczaniu B, wynik A odczytywany jest z pamięci.
- Podproblemy rozwiązywane są więc w kolejności od „najmniejszych" do „największych".
- Przykładowo do policzenia $\binom{3}{2}$ potrzeba $\binom{2}{1}$ i $\binom{2}{2}$, więc należy je policzyć wcześniej. Zaś przed $\binom{2}{1}$ należy policzyć $\binom{1}{0}$ i $\binom{1}{1}$. Dobra kolejność liczenia to: $\binom{1}{0}$, $\binom{1}{1}$, $\binom{2}{1}$, $\binom{2}{2}$, $\binom{3}{2}$.

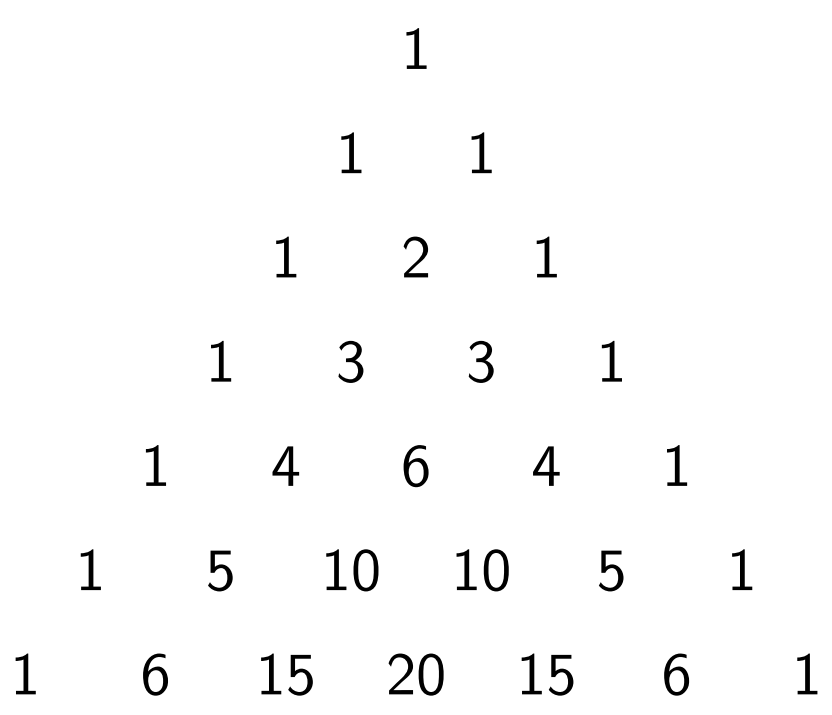
Programowanie dynamiczne a trójkąt Pascala

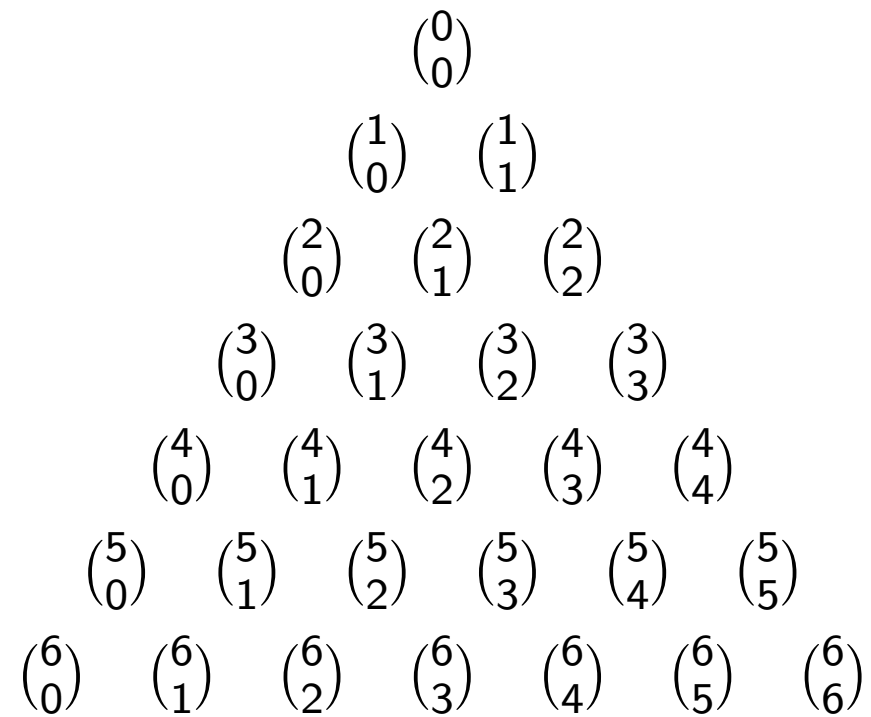




- Ogólnie, symbole postaci $\binom{n-1}{?}$ należy policzyć przed $\binom{n}{?}$.
- Dlatego trójkąt Pascala wypełniamy kolejnymi wierszami,
- które zawierają wyniki dla kolejnych $n = 0, 1, \ldots$.
- Kolejność wypełniania wewnątrz wiersza jest dowolna (k można rozpatrywać w dowolnej kolejności).

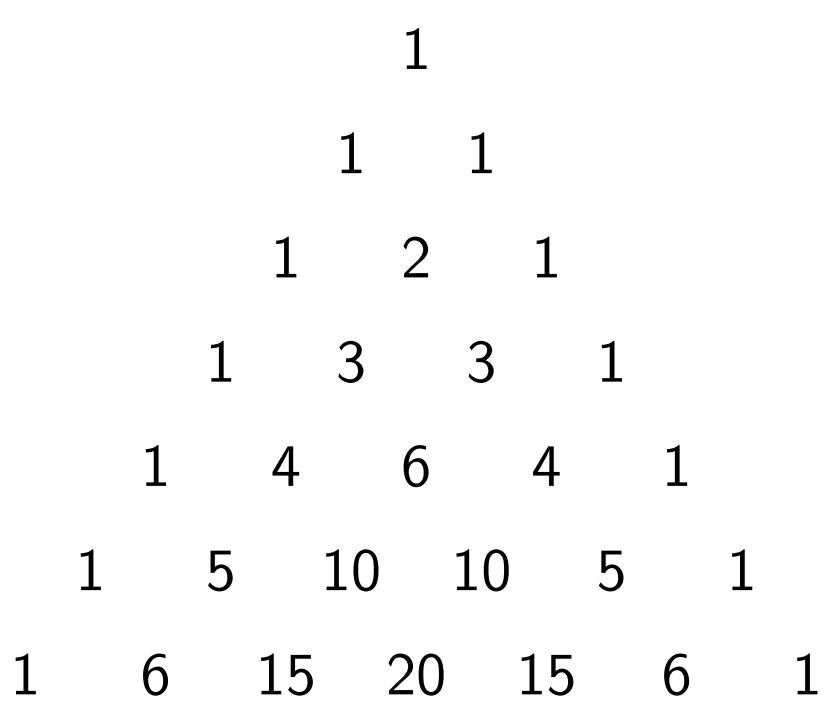
Programowanie dynamiczne a trójkąt Pascala

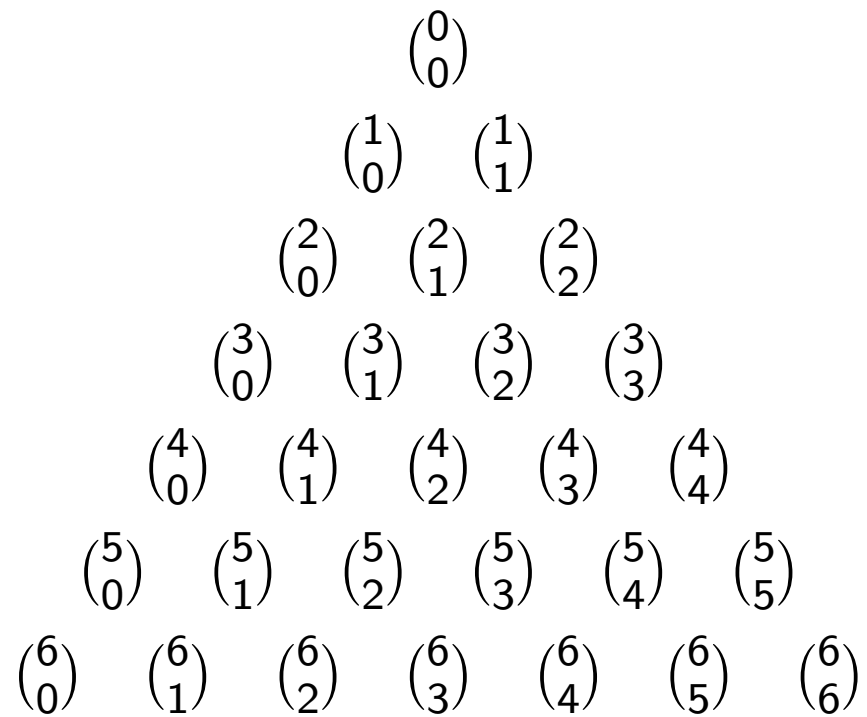




- Ogólnie, symbole postaci $\binom{n-1}{?}$ należy policzyć przed $\binom{n}{?}$.
- Dlatego trójkąt Pascala wypełniamy kolejnymi wierszami,
- które zawierają wyniki dla kolejnych $n = 0, 1, \ldots$.
- Kolejność wypełniania wewnątrz wiersza jest dowolna (k można rozpatrywać w dowolnej kolejności).

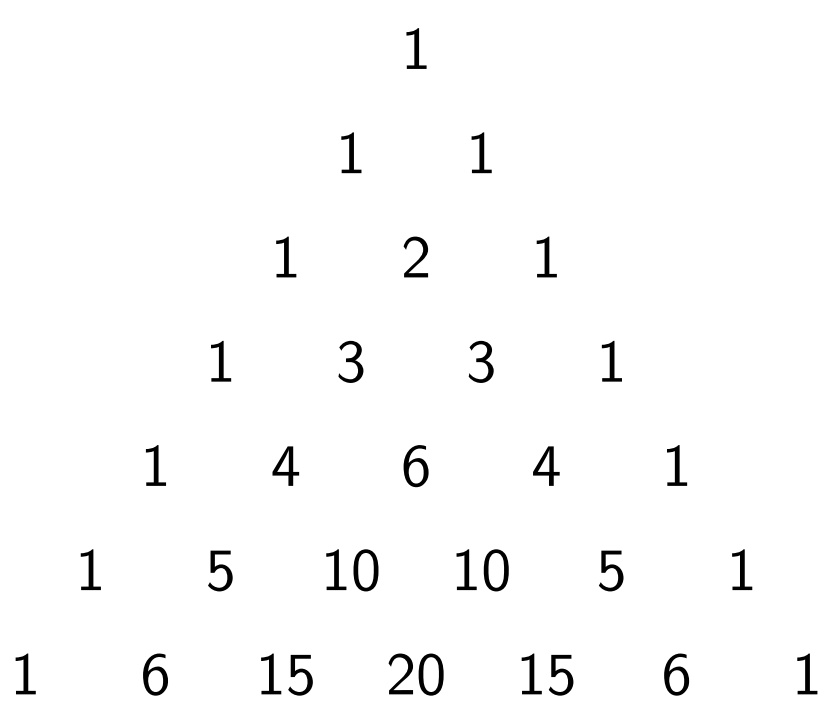
Programowanie dynamiczne a trójkąt Pascala

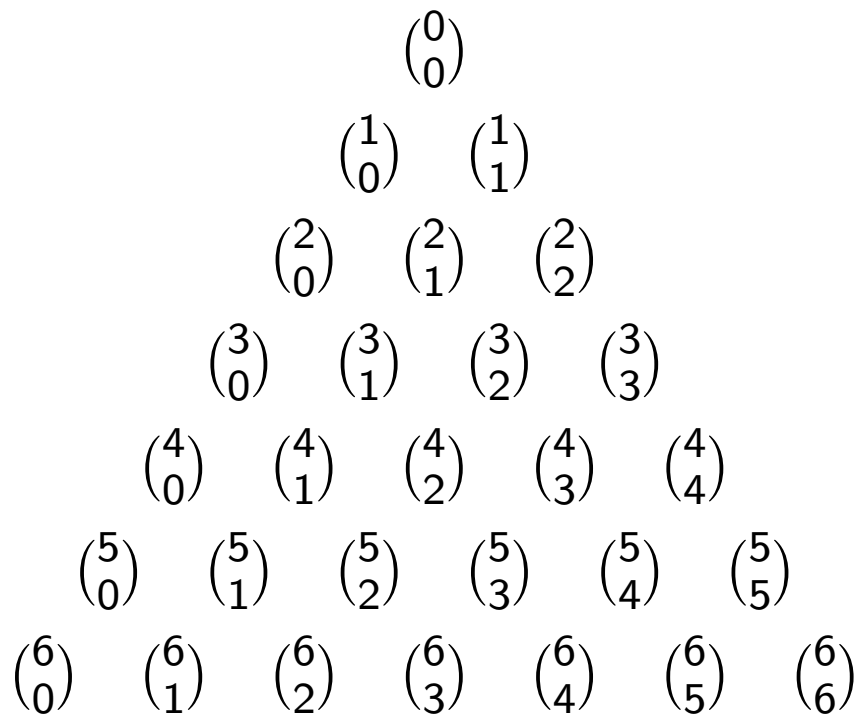




- Ogólnie, symbole postaci $\binom{n-1}{?}$ należy policzyć przed $\binom{n}{?}$.
- Dlatego trójkąt Pascala wypełniamy kolejnymi wierszami,
- które zawierają wyniki dla kolejnych $n = 0, 1, \ldots$.
- Kolejność wypełniania wewnątrz wiersza jest dowolna (k można rozpatrywać w dowolnej kolejności).

Programowanie dynamiczne a trójkąt Pascala





- Ogólnie, symbole postaci $\binom{n-1}{?}$ należy policzyć przed $\binom{n}{?}$.
- Dlatego trójkąt Pascala wypełniamy kolejnymi wierszami,
- które zawierają wyniki dla kolejnych $n = 0, 1, \ldots$.
- Kolejność wypełniania wewnątrz wiersza jest dowolna (k można rozpatrywać w dowolnej kolejności).

Pseudokod dynamicznego algorytmu implementującego zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(N, K):
  // opcjonalnie: K ← min(K, N-K)
  stwórz tablicę T o wymiarach N+1 na K+1
  for n ← 0, 1, ..., N:
    for k ← 0, 1, ..., min(K, n):
      if k = 0 or k = n:
        T[n][k] ← 1
      else:
        T[n][k] ← T[n-1][k-1] + T[n-1][k]
  return T[N][K]
```

- Złożoność czasowa i pamięciowa: $\Theta(NK)$ – taki rozmiar ma tablica T, zaś liczba operacji $(+)$ jest do niego proporcjonalna.
- Opcjonalna linia redukuje złożoności do $\Theta(N \min(K, N-K))$ wykorzystując zależność $\binom{N}{K} = \binom{N}{\min(K, N-K)}$.

Pseudokod dynamicznego algorytmu implementującego zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(N, K):
  // opcjonalnie: K ← min(K, N-K)
  stwórz tablicę T o wymiarach N+1 na K+1
  for n ← 0, 1, ..., N:
    for k ← 0, 1, ..., min(K, n):
      if k = 0 or k = n:
        T[n][k] ← 1
      else:
        T[n][k] ← T[n-1][k-1] + T[n-1][k]
  return T[N][K]
```

- Złożoność czasowa i pamięciowa: $\Theta(NK)$ – taki rozmiar ma tablica T, zaś liczba operacji (+) jest do niego proporcjonalna.
- Opcjonalna linia redukuje złożoności do $\Theta(N \min(K, N-K))$ wykorzystując zależność $\binom{N}{K} = \binom{N}{\min(K, N-K)}$.

Pseudokod dynamicznego algorytmu implementującego zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}:$$

```
fun newton(N, K):
  // opcjonalnie: K ← min(K, N-K)
  stwórz tablicę T o wymiarach N+1 na K+1
  for n ← 0, 1, ..., N:
    for k ← 0, 1, ..., min(K, n):
      if k = 0 or k = n:
        T[n][k] ← 1
      else:
        T[n][k] ← T[n-1][k-1] + T[n-1][k]
  return T[N][K]
```

- Złożoność czasowa i pamięciowa: $\Theta(NK)$ – taki rozmiar ma tablica T, zaś liczba operacji $(+)$ jest do niego proporcjonalna.
- Opcjonalna linia redukuje złożoności do $\Theta(N \min(K, N-K))$ wykorzystując zależność $\binom{N}{K} = \binom{N}{\min(K, N-K)}$.

Pseudokod dynamicznego algorytmu implementującego zależność

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases} :$$

```
fun newton(N, K):
  // opcjonalnie: K ← min(K, N-K)
  stwórz tablicę T o wymiarach N+1 na K+1
  for n ← 0, 1, ..., K: //dla wszystkich n
    T[n][n] ← 1  //(n choose n) = 1; obsługa warunku k=n
  for n ← 1, 2, ..., N:
    T[n][0] ← 1
    for k ← 1, 2, ..., min(K, n-1):
      T[n][k] ← T[n-1][k-1] + T[n-1][k]
  return T[N][K]
```

Usprawnienie programistyczne: zamiast sprawdzać warunek „$k = 0$ lub $k = n$” w wewnętrznej pętli, przypadki te obsłużono przed nią, zaś sama pętla rozpatruje $k \in [1, n-1]$.

T zawiera fragment trójkąta Pascala

Tablica T uzyskana przy obliczaniu $\binom{5}{2}$ i trójką Pascala:

n \ k	0	1	2
0	$\binom{0}{0}$		
1	$\binom{1}{0}$	$\binom{1}{1}$	
2	$\binom{2}{0}$	$\binom{2}{1}$	$\binom{2}{2}$
3	$\binom{3}{0}$	$\binom{3}{1}$	$\binom{3}{2}$
4	$\binom{4}{0}$	$\binom{4}{1}$	$\binom{4}{2}$
5	$\binom{5}{0}$	$\binom{5}{1}$	$\binom{5}{2}$

n \ k	0	1	2
0	1		
1	1	1	
2	1	2	1
3	1	3	3
4	1	4	6
5	1	5	10

1

1 1

1 2 1

1 3 3 1

1 4 6 4 1

1 5 10 10 5 1

(komórki niewypełnione przez algorytm pozostawiono puste)

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy T wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- T można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w T mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy `T` wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- `T` można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w `T` mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy `T` wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- `T` można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w `T` mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy `T` wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- `T` można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w `T` mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy `T` wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- `T` można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w `T` mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm używający mniej pamięci

- Dynamiczny algorytm korzystający z zależności

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{w pozotałych przypadkach} \end{cases}$$

  do obliczania wartości $\binom{n}{\dots}$ używa jedynie $\binom{n-1}{\dots}$.
- Inaczej mówiąc, każdy wiersz tablicy `T` wyznaczany jest jedynie na podstawie wiersza poprzedniego.
- `T` można więc zredukować (N krotnie) do pojedynczego, ostatnio policzonego wiersza,
- redukując tym samym złożoność pamięciową do $\Theta(K)$ albo, dzięki $\binom{N}{K} = \binom{N}{N-K}$, do $\Theta(\min(K, N-K))$.
- Co więcej, by policzyć $\binom{n}{k}$ potrzebne są jedynie $\binom{n-1}{k-1}$ i $\binom{n-1}{k}$, zaś nigdy nie są potrzebne żadne wartości $\binom{n-1}{b}$ dla $b > k$.
- Jeśli więc w `T` mamy wartości $\binom{n-1}{\dots}$, to można je bezpiecznie, **w kolejności od końca do początku tablicy**, nadpisać wartościami $\binom{n}{\dots}$, nie tracąc liczb potrzebnych później.

Dynamiczny algorytm o złożoności pamięciowej $\Theta(\min(K, N-K))$:

```
fun newton(N, K):
  K ← min(K, N-K)  // opcjonalnie
  stwórz tablicę T o długości K+1
  // 1-ki obsługują przypadki n=0 lub k=n:
  for i ← 0, 1, ..., K: // dla wszystkich n
    T[i] ← 1 // (n 0) = (n n) = 1
  for n ← 2, 3, ..., N:
    k_max ← min(K, n-1)
    for k ← k_max, k_max-1, ..., 1:
      // tu T[k-1]=(n-1 k-1) i T[k]=(n-1 k)
      T[k] ← T[k-1] + T[k]
      // tu T[k-1]=(n-1 k-1) i T[k]=(n k)
  return T[K]
```

W wewnętrznej pętli, tablica `T` jest nadpisywana od końca.
W każdym kroku, jej postać jest przekształcana z:
$\binom{n-1}{0}, \binom{n-1}{1}, \ldots, \binom{n-1}{k-1}, \binom{n-1}{k}, \binom{n}{k+1}, \binom{n}{k+2}, \ldots, \binom{n}{\min(K,n)}$ do
$\binom{n-1}{0}, \binom{n-1}{1}, \ldots, \binom{n-1}{k-1}, \binom{n}{k}, \binom{n}{k+1}, \binom{n}{k+2}, \ldots, \binom{n}{\min(K,n)}$.

Bibliografia

- K. Diks, A. Malinowski, W. Rytter, T. Waleń *Algorytmy i struktury danych/Wstęp: elementarne techniki algorytmiczne i struktury danych*, WWW: http://wazniak.mimuw.edu.pl/index.php?title=Algorytmy_i_struktury_danych/Wst%C4%99p:_elementarne_techniki_algorytmiczne_i_struktury_danych
- Thomas H. Cormen, Charles E. Leiserson, Roland L. Rivest, Clifford Stein *Wprowadzenie do algorytmów*
- P. Idziak, B. Bosek, P. Micek *Matematyka dyskretna 1/Wykład 5: Współczynniki dwumianowe*, WWW: http://wazniak.mimuw.edu.pl/index.php?title=Matematyka_dyskretna_1/Wyk%C5%82ad_5:_Wsp%C3%B3%C5%82czynniki_dwumianowe