Wykład 14
Programowanie dynamiczne

Programowanie dynamiczne

- Złożoność obliczeniowa funkcji rekurencyjnych
- programowanie dynamiczne
- przykłady programów

Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych

Często funkcję rekurencyjną $f(n)$ zapisać można w postaci wyrażenia:

$$g(n, f(n - i_1), f(n - i_2), \ldots, f(n - i_k)),$$

które należy rozumieć jako wyrażenie, w którym występuje k innych wartości funkcji f.

Example

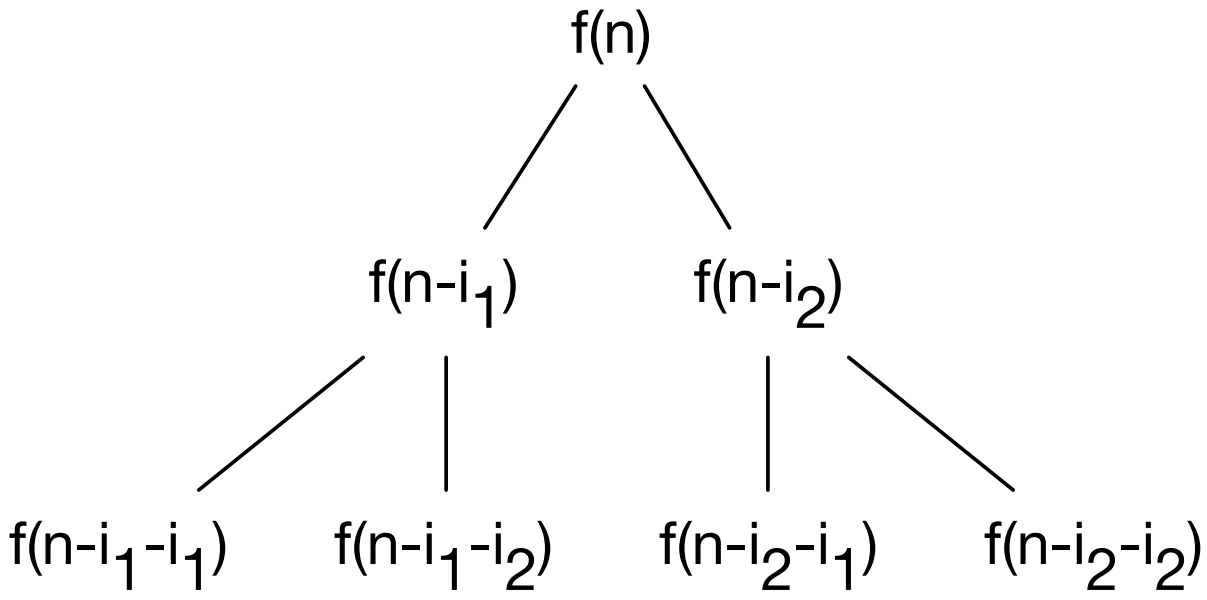
silnia ($k = 1$) $g(n, f(n-1)) = n \cdot f(n-1)$

ciąg Fibonacciego ($k = 2$) $g(n, f(n-1), f(n-2)) = f(n-1) + f(n-2)$

Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych

Gdy $k \geq 2$, liczba obliczanych wyrażeń rośnie wykładniczo.



Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych: ciąg Fibonacciego

Niech s_n oznacza liczbę operacji dodawania wykonywanych podczas obliczania n-tego wyrazu ciągu Fibonacciego zadanego wzorem rekurencyjnym:

$$Fib(n) = \begin{cases} 0 & \text{gdy } n = 0, \\ 1 & \text{gdy } n = 1, \\ Fib(n-1) + Fib(n-2) & \text{gdy } n > 1 \end{cases}$$

Wartość s_n można wyliczyć, z poniższych zależności:

$$s_0 = 0,\ s_1 = 0,\ s_n = 1 + s_{n-1} + s_{n-2} \text{ dla } n > 1$$

Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych: ciąg Fibonacciego

W poniższej tabeli zebrano pierwsze wyrazy ciągu $Fib(n)$ i s_n:

n	0	1	2	3	4	5	6	7
$Fib(n)$	0	1	1	2	3	5	8	13
s_n	0	0	1	2	4	7	12	20

Łatwo zauważyć w niej następujące dwie prawidłowości:

$$\begin{aligned} s_n &= \sum_{i=0}^{n-1} Fib(i) \\ s_n &= Fib(n+1) - 1 \end{aligned}$$

Ćwiczenie

Udowodnij indukcyjnie oba powyższe równania.

Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych: ciąg Fibonacciego

Stosując funkcje tworzące – wykraczające poza tematykę naszego wykładu – można wyprowadzić wzór analityczny na n-ty wyraz ciągu Fibonacciego:

$$Fib(n) = \frac{\varphi^n - \psi^n}{\sqrt{5}} = \left\lfloor \frac{\varphi^n}{\sqrt{5}} + \frac{1}{2} \right\rfloor$$

gdzie $\varphi = \frac{1+\sqrt{5}}{2} \approx 1.618$ i $\psi = \frac{1-\sqrt{5}}{2} \approx -0.618$.

Ćwiczenie

Udowodnij poprawność powyższej postaci analitycznej.

Granica $\lim_{n\to\infty} \frac{Fib(n)}{\varphi^n} = \frac{1}{\sqrt{5}}$, zatem

$$Fib(n) = \Theta(\varphi^n).$$

Programowanie dynamiczne

Złożoność obliczeniowa funkcji rekurencyjnych: ciąg Fibonacciego

- Z tego, że $Fib(n+2) = \Theta(\varphi^n)$, można wywnioskować, że liczba s_n wykonywanych operacji dodawania podczas obliczania $Fib(n)$ jest rzędu $\Theta(\varphi^n)$, zatem rośnie wykładniczo wraz ze wzrostem n.
- Powrócimy jeszcze do ciągu Fibonacciego pokazując jak policzyć jego wartości wykonując $\Theta(n)$ operacji dodawania.
- Dzięki programowaniu dynamicznemu, można w prosty sposób przeformułować funkcję rekurencyjną tak aby była efektywniej obliczana (np. w czasie wielomianowym albo nawet liniowym) z użyciem iteracji i tablic do przechowywania pośrednich wyników.

Programowanie dynamiczne

Idea

- Aby uniknąć wielokrotnego obliczania wartości tych samych wyrażeń tablicuje się je.
- Programowanie dynamiczne nazywamy d-wymiarowym, jeśli rekurencyjna funkcja $f : N^d \mapsto R$ jest d-argumentowa.
- Wyniki pośrednie w d-wymiarowym programowaniu dynamicznym tablicuje się w d-wymiarowej tablicy $t[n_1, n_2, \ldots, n_d]$, gdzie $n_1, n_2, \ldots, n_d$ są rozmiarami tablicy w kolejnych wymiarach.
- Cała trudność programowania dynamicznego tkwi w dobraniu odpowiedniej kolejności wyliczania elementów tablicy t.

Programowanie dynamiczne

Idea

Example ($d = 1$)

W tym przypadku najczęściej wystarcza obliczanie wartości $t[i]$ dla kolejnych wartości $i = 1, 2, 3, \ldots$

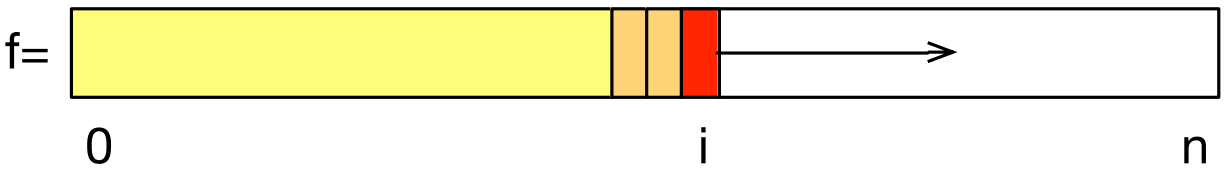
Example ($d = 2$)

W tym przypadku najczęściej oblicza się wartości $t[i, j]$ kolejnymi wierszami albo kolumnami. Czasami trzeba jednak bardziej wyrafinowanej kolejności jak np. kolejnymi coraz dłuższymi przekątnymi:

```
1: for k ← 1, n do                          ▷ przetwarzamy k-tą przekątną
2:     for i ← 1, k do                      ▷ przetwarzamy i-ty element k-tej przekątnej
3:         j ← k + 1 − i;                   ▷ na k-tej przekątnej zachodzi warunek i + j = k + 1
4:         oblicz t[i, j]
5:     end for
6: end for
```

Programowanie dynamiczne

Przykłady programów: ciąg Fibonacciego

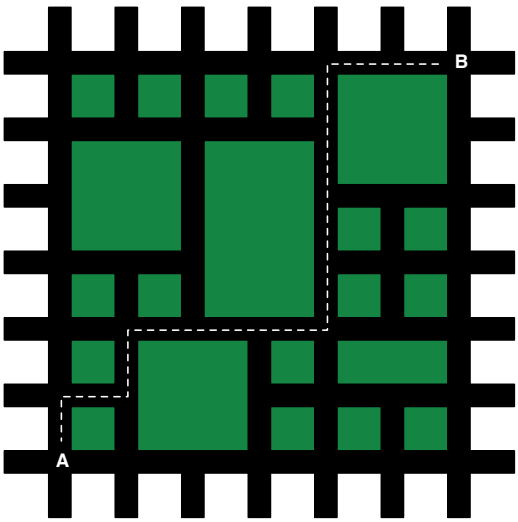



```
1: function Fib(n)
2:     f[0] ← 0;
3:     f[1] ← 1;
4:     for i ← 2, n do
5:         f[i] ← f[i − 1] + f[i − 2]
6:     end for
7:     return f[n]
8: end function
```


Programowanie dynamiczne

Przykłady programów: zliczanie dróg

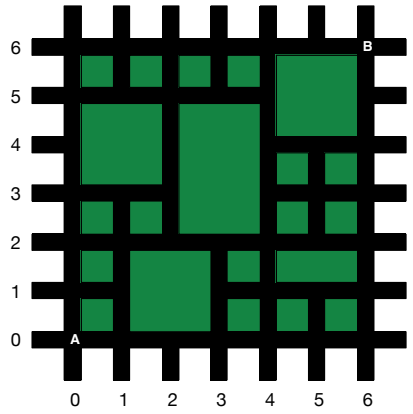
Rozpatrzmy problem zliczania najkrótszych dróg między zadanymi dwoma skrzyżowaniami A i B:



Programowanie dynamiczne

Przykłady programów: zliczanie dróg

- Ponumerujmy ulice od lewej do prawej kolejnymi liczbami naturalnymi $0, 1, 2, \ldots$
- Podobnie ponumerujmy ulice od dołu do góry kolejnymi liczbami naturalnymi $0, 1, 2, \ldots$
- Przy takim ponumerowaniu skrzyżowanie A ma współrzędne $(0, 0)$ a skrzyżowanie B współrzędne $(6, 6)$.



Programowanie dynamiczne

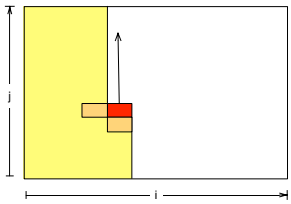
Przykłady programów: zliczanie dróg

- Symbolem $N[i, j]$ oznaczmy liczbę najkrótszych dróg od skrzyżowania A do skrzyżowania o współrzędnych (i, j), tj. przecięcie i-tej ulicy pionowej z j-tą ulicą poziomą.
- Przyjmujemy, że $N[i, j] = 0$, gdy skrzyżowanie (i, j) jest zajęte przez budynek.
- Pozostałe wartość $N[i, j]$ można wyliczyć ze wzoru:

$$N[i,j] = \begin{cases} 1 & \text{gdy } i = 0, j = 0, \\ N[i-1,j] + N[i,j-1] & \text{gdy połączone z } (i-1,j) \text{ oraz } (i,j-1), \\ N[i-1,j] & \text{gdy połączone z } (i-1,j) \text{ ale nie z } (i,j-1), \\ N[i,j-1] & \text{gdy połączone z } (i,j-1) \text{ ale nie z } (i-1,j). \end{cases}$$

Programowanie dynamiczne

Przykłady programów: zliczanie dróg

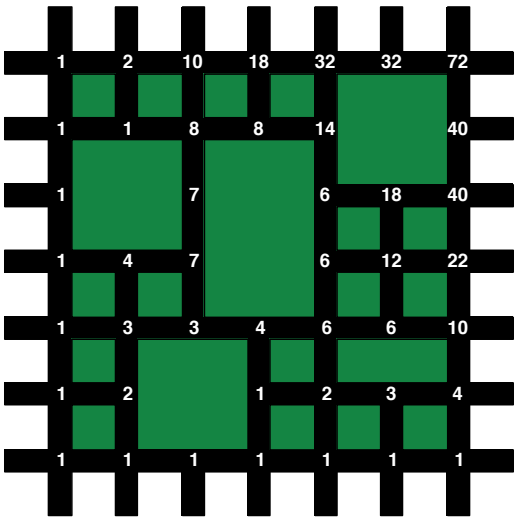


1: **for** $i \leftarrow 0, m$ **do**
2: **for** $j \leftarrow 0, n$ **do**
3: **if** (i, j) jest zajęte przez budynek **then**
4: $N[i, j] \leftarrow 0$
5: **else**
6: **if** $i = 0 \wedge j = 0$ **then**
7: $N[i, j] \leftarrow 1$
8: **else**
9: $N[i, j] \leftarrow N[i-1, j]$ gdy połączone z $(i-1, j) + N[i, j-1]$ gdy połączone z $(i, j-1)$
10: **end if**
11: **end if**
12: **end for**
13: **end for**

Programowanie dynamiczne

Przykłady programów: zliczanie dróg

Na poniższym rysunku zaznaczono dla każdego skrzyżowania (i, j) liczbę $N[i, j]$ najkrótszych dróg prowadzących do niego ze skrzyżowania $A(0, 0)$:



Programowanie dynamiczne

Przykłady programów: zliczanie dróg

- Liczba najkrótszych dróg rośnie wykładniczo szybko wraz ze wzrostem i oraz j.
- Liczbę najkrótszych dróg można policzyć bez potrzeby ich generowania.
- Dzięki wzorowi rekurencyjnemu i programowaniu dynamicznemu możemy policzyć ile jest najkrótszych dróg w czasie rzędu $O(i \cdot j)$.
- Zauważ, że gdyby nie było budynków zajmujących część połączeń wówczas $N[i,j] = \binom{i+j}{i}$.
- Kilka pierwszych wartości $N[n,n] = \binom{2n}{n}$:

n	1	2	3	4	5	6	7	8
$N[n,n]$	2	6	20	70	252	924	3432	12870

Programowanie dynamiczne

Przykłady programów: problem plecakowy

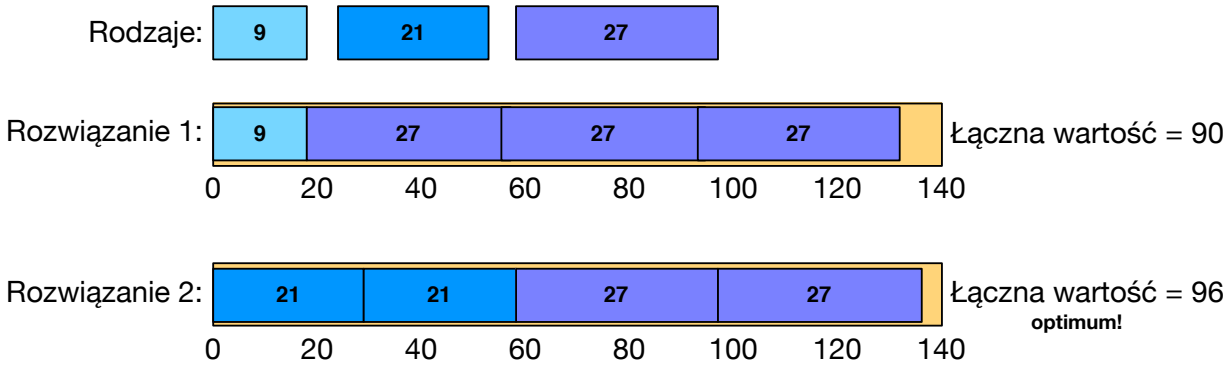
- Danych jest n rodzajów przedmiotów, przy czym zakładamy, że jest dowolnie dużo przedmiotów każdego z tych rodzajów.
- Każdy przedmiot i-tego rodzaju ma rozmiar $size[i]$ i wartość $value[i]$.
- Należy zapakować plecak o pojemności k (suma rozmiarów nie może przekroczyć pojemności), tak aby łączna wartość spakowanych przedmiotów była jak największa.

Programowanie dynamiczne

Przykłady programów: problem plecakowy

Example

$n = 3, size[1] = 19, size[2] = 29, size[3] = 39, value[1] = 9, value[2] = 21, value[3] = 27$



Na powyższym rysunku przedstawiono przykładowe dwa rozwiązania dla pojemności $k = 140$. Drugie rozwiązanie jest optymalne.

Programowanie dynamiczne

Przykłady programów: problem plecakowy

- Niech funkcja $maxknapsack(k)$ będzie równa maksymalnej łącznej wartości przedmiotów jakie można spakować do plecaka o pojemności k.
- Załóżmy, że do plecaka o pojemności k zdecydujemy się włożyć przedmiot i-tego rodzaju.
- Jeśli przedmiot i-tego rodzaju mieści się w plecaku ($size[i] \leq k$), to po włożeniu go do plecaka możemy osiągnąć łączną wartość:

$$W_i = maxknapsack(k - size[i]) + value[i].$$

- Wybór rodzaju wkładanego przedmiotu dokonamy wybierając największe W_i, dla $i = 1, 2, \ldots, n$.

Programowanie dynamiczne

Przykłady programów: problem plecakowy

```
1:  function MaxKnapsack(k, size, value, n)
2:      knapsack[0] ← 0;
3:      for i ← 1, k do        ▷ rozpatrywanie plecaków o coraz większej pojemności
4:          max ← 0;
5:          for t ← 1, n do          ▷ rozpatrywanie przedmiotu każdego rodzaju
6:              if size[t] ≤ i then                          ▷ przedmiot mieści się
7:                  total ← knapsack[i − size[t]] + value[t];      ▷ łączna wartość
8:                  if total > max then
9:                      max ← total
10:                 end if
11:             end if
12:         end for
13:         knapsack[i] ← max          ▷ największa wartość przy pojemności i
14:     end for
15:     return knapsack[k]
16: end function
```


Programowanie dynamiczne

Przykłady programów: najdłuższy wspólny podciąg

- Dane są dwa napisy: pierwszy długości m znaków $(a_1, a_2, \ldots, a_m)$ i drugi długości n znaków $(b_1, b_2, \ldots, b_n)$.
- Ciąg znaków $(c_1, c_2, \ldots, c_k)$ nazywamy wspólnym podciągiem a i b jeśli

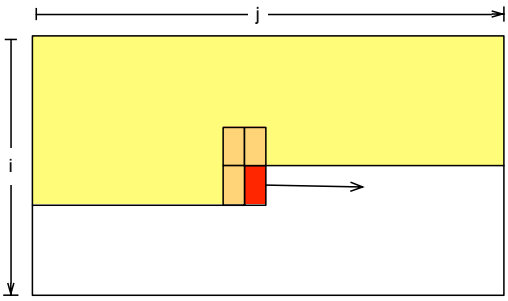
$$\exists_{1 \le i_1 < i_2 < \ldots < i_k \le m} \forall_{l=1,2,\ldots,k} c_l = a_{i_l}$$

$$\exists_{1 \le j_1 < j_2 < \ldots < j_k \le n} \forall_{l=1,2,\ldots,k} c_l = b_{j_l}$$

- Interesuje nas znalezienie jak najdłuższego wspólnego podciągu.
- W tablicy $T[i,j]$ będziemy wyliczać długość najdłuższego wspólnego podciągu dla fragmentów $(a_1, a_2, \ldots, a_i)$ oraz $(b_1, b_2, \ldots, b_j)$.
- Jeśli $a_i = b_j$, to $T[i,j] = T[i-1, j-1] + 1$.
- Jeśli $a_i \neq b_j$, to $T[i,j] = \max\{T[i, j-1], T[i-1, j]\}$.

Programowanie dynamiczne

Przykłady programów: najdłuższy wspólny podciąg




```
1: for i ← 1, m do
2:     for j ← 1, n do
3:         if a_i = b_j then
4:             T[i,j] ← T[i − 1, j − 1] + 1
5:         else
6:             T[i,j] ← max{T[i,j − 1], T[i − 1,j]}
7:         end if
8:     end for
9: end for
```


Programowanie dynamiczne

Przykłady programów: najdłuższy wspólny podciąg

Example (Tajemnica programowania)

		p	r	o	g	r	a	m	o	w	a	n	i	a
		0	1	2	3	4	5	6	7	8	9	10	11	12	13
	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
t	1	0	0	0	0	0	**0**	0	0	0	0	0	0	0	0
a	2	0	0	0	0	0	0	**1**	1	1	1	1	1	1	1
j	3	0	0	0	0	0	0	**1**	1	1	1	1	1	1	1
e	4	0	0	0	0	0	0	**1**	1	1	1	1	1	1	1
m	5	0	0	0	0	0	0	1	**2**	**2**	**2**	**2**	2	2	2
n	6	0	0	0	0	0	0	1	2	2	2	2	**3**	3	3
i	7	0	0	0	0	0	0	1	2	2	2	2	3	**4**	4
c	8	0	0	0	0	0	0	1	2	2	2	2	3	**4**	4
a	9	0	0	0	0	0	0	1	2	2	2	3	3	4	**5**