Matematyka dyskretna

Andrzej M. Borzyszkowski

Instytut Informatyki
Uniwersytet Gdański

sem. zimowy 2025/26

inf.ug.edu.pl/~amb/MaD

Indukcja matematyczna

Dobry porządek

- tw.: każdy niepusty podzbiór zbioru liczb naturalnych ma element najmniejszy
- oczywiście pusty podzbiór nie ma
- podzbiór zbioru liczb całkowitych nie musi mieć najmniejszego elementu, np. wszystkie liczby całkowite
- podzbiór zbioru wszystkich liczb nieujemnych nie musi mieć najmiejszego elementu, np. wszystkie liczby wymierne dodatnie
- ale jeśli liczby są całkowite i nieujemne, to każdy niepusty podzbiór ma element najmniejszy

Indukcja matematyczna

- chcemy dowieść twierdzenia $\forall n \in \mathbb{N}.\, P(n)$ dla pewnej własności P
- gdyby tak nie było, to zbiór $\{n \in \mathbb{N} \mid \neg P(n)\}$ byłby niepusty
- i miałby najmniejszy element
- jeśli pokażemy, że $P(0)$, to nie jest nim 0
- jeśli pokażemy, że $P(k) \Rightarrow P(k{+}1)$ dla $k \in \mathbb{N}$, to nie jest nim żadne $n > 0$, bo zastosujemy implikację do $k = n{-}1$
- czyli zbiór ew. kontrprzykładów musi być pusty – nie ma kontrprzykładów
- w implikacji można nawet założyć więcej, łatwiej może być dowieść $(\forall k{<}\ell.\, P(k)) \Rightarrow P(\ell)$

Przykłady indukcji

- $\Sigma_{i=0}^{n} i = \frac{n\cdot(n+1)}{2}$: trochę obliczeń
- każda liczba naturalna jest iloczynem liczb pierwszych:
  dw.: niech n będzie najmniejszą liczbą, która nie jest takim iloczynem.
  Nie może być pierwsza (bo to też jest iloczyn).
  Czyli $n = k \cdot \ell$ dla $k, \ell < n$.
  Ale one są iloczynami liczb pierwszych (założenie indukcyjne), więc i n jest.

Niespodzianki w dowodach

- „wszystkie konie są jednej maści"
- $P(n) = \forall n \in \mathbb{N}\setminus\{0\}$ wszystkie konie w zbiorze n koni są jednej maści
- $P(1)$ oczywiście zachodzi
- badamy $P(n+1)$ dla pewnego n, numerujemy konie, pierwszych n: $\{k_0, \ldots, k_{n-1}\}$ jest jednej maści (założenie indukcyjne), podobnie ostatnich n: $\{k_1, \ldots, k_n\}$, tak więc wszystkie są jednej maści
- błędem jest użycie wielokropka i sugerowanie, że zbiory mają niepusty przekrój $\{k_1, \ldots, k_{n-1}\}$
  dla $n = 1$ zbiory $\{k_0\}$ i $\{k_1\}$ są rozłączne i konie mogą być różnej maści
- ale jeśli w pewnym zbiorze koni $P(2)$ zachodzi, to rzeczywiście wszystkie konie w tym zbiorze są jednej maści

Rekursywny typ danych

- dobrze postawione nawiasy, np. (()()(()))
  - pusty napis ε jest DPN
  - jeśli s jest DPN to (s) też jest
  - jeśli s i t są DPN to $s\,t$ też jest
- to nie jest zbyt dobra definicja, mamy dwa różne dowody, że ()()() jest DPN – lewa para z dostawionymi dwiema parami z prawej lub na odwrót, czasami to przeszkadza
- wyrażenia arytmetyczne np. $((5 \cdot (x + y)) + (z \cdot 4))$
  - liczba i zmienna są wyrażeniami
  - $(w_1 \cdot w_2)$ jest wyrażeniem
  - $(w_1 + w_2)$ jest wyrażeniem
  - tak naprawdę praktyczna definicja jest bardziej skomplikowana by uprościć notację i było mniej nawiasów

Przykład twierdzenia o rekursywnych typach danych

- tw.: DPN mają tyle samo nawiasów lewych co prawych
- dw.: ε – zero nawiasów
  jeśli s ma tyle samo nawiasów lewych co prawych, to (s) ma też tyle samo, konkretnie o jeden więcej każdego rodzaju
  jeśli s i t z osobna mają tyle samo nawiasów lewych co prawych, to $s\,t$ też: sumę liczb nawiasów
- tw.: w dowolnym prefiksie DPN liczba nawiasów lewych jest niemniejsza niż prawych
- dw.: ε ma zero nawiasów, jedynym prefiksem jest ε
  załóżmy że s spełnia warunek, prefiksy (s), z wyjątkiem całego napisu, mają o jeden nawias lewy więcej
  prefiksami $s\,t$ są albo prefiksy s, spełniają one warunek (indukcja)
  albo s i dalej prefiks t, też spełniają warunek (z założenia o t i wcześniejszego twierdzenia)

Definicje rekursywne funkcji

- *eval*: wyrażenia arytmetyczne $\times$ wartościowanie zmiennych $\to \mathbb{N}$
  - *eval*(liczba) = ta właśnie liczba
  - *eval*(zmienna) = wartość zmiennej zapisana w wartościowaniu
  - $eval(w_1 \cdot w_2) = eval(w_1) \cdot eval(w_2)$
  - $eval(w_1 + w_2) = eval(w_1) + eval(w_2)$
- znaki $\cdot$ i $+$ po lewej to tylko znaki, po prawej oznaczają operacje w zbiorze $\mathbb{N}$

Liczby naturalne jako typ danych

- $0 \in \mathbb{N}$
- jeśli $n \in \mathbb{N}$ to $n{+}1 \in \mathbb{N}$
- to może być uważane za definicję liczb naturalnych
- indukcja matematyczna jest szczególnym przypadkiem strukturalnej
- $n! = \begin{cases} 1 & n = 0 \\ n \cdot (n-1)! & n > 1 \end{cases}$
- $fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n{-}1) + fib(n{-}2) & n > 1 \end{cases}$

Błędne „definicje”

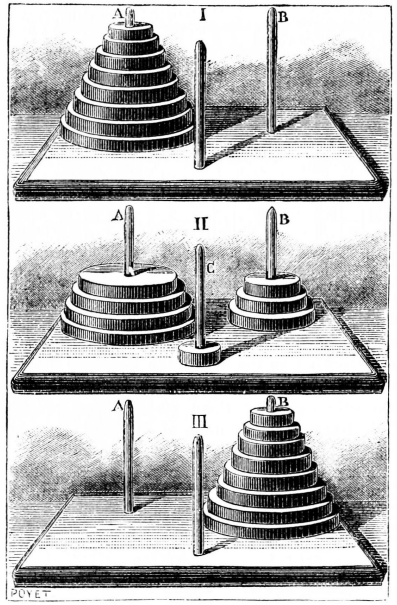
- $f(n) = \{\ f(n-1) + 1$
  brakuje przypadku bazowego, wiele funkcji spełnia takie równanie, program, który to będzie obliczać nie zakończy działania
- $f(n) = \begin{cases} 1 & n = 0 \\ f(n+1) & n > 1 \end{cases}$
  jest przypadek bazowy ale program, który to będzie obliczać nie zakończy działania, nadal wiele funkcji spełnia takie równanie
- $f(n) = \begin{cases} 0 & n \text{ parzyste} \\ 1 & n \text{ podzielne przez } 3 \\ 2 & \text{w p.p.} \end{cases}$
  sprzeczne warunki

Rekurencja

rekurencja

- inaczej rekursja
- *recurrence* – powtórne pojawienie się
- $t_0 = 0$, $t_{n+1} = t_n + 1$ dla $n \in \mathbb{N}$
- obliczamy $t_1 = 1$, $t_2 = 2$, $t_3 = 3$, itd.
- zadanie: dane równanie rekurencyjne, podać wzór na wyrazy definiowanego ciągu
- albo chociaż jakieś własności, np. szybkość wzrostu
- metody:
  - zgadywanie (i sprawdzenie poprawności)
  - podstawianie
  - opracowane rozwiązania dla niektórych klas rekurencji

Wieże z Hanoi



- osiem krążków różniej wielkości ułożonych mniejsze na większych
- wolno przekładać po jednym, zawsze mniejszy na większy
- zadanie: przełożyć całą wieżę ze słupka A na słupek B
- rozwiązanie (rekurencyjne): przenosimy $n-1$ krążków z A do C, potem największy krążek do B w końcu $n-1$ krążków z C do B

Źródło: `commons.wikimedia.org/wiki/File:PSM_V26_D464_The_tower_of_hanoi.jpg`

Wieże z Hanoi – złożoność

- równanie rekurencyjne na funkcję złożoności
- $T_1 = 1$
  $T_n = 2 \cdot T_{n-1} + 1$ dla $n > 1$
- kolejne wartości: $T_2 = 3$, $T_3 = 7$, $T_4 = 15$, ...
- hipoteza: $T_n = 2^n - 1$
- dowód indukcyjny:
  $T_1 = 1 = 2^1 - 1$ OK
  $T_n = 2 \cdot (2^{n-1} - 1) + 1 = 2^n - 2 + 1 = 2^n - 1$
  było wykorzystane założenie indukcyjne
- albo wstawianie
  $T_n = 2 \cdot T_{n-1} + 1 = 2 \cdot (2 \cdot T_{n-2} + 1) + 1 = 4 \cdot T_{n-2} + 2 + 1$
  $= 4 \cdot (2 \cdot T_{n-3} + 1) + 2 + 1 = 8 \cdot T_{n-3} + 4 + 2 + 1$
  $= \ldots = 2^k \cdot T_{n-k} + \Sigma_{i=0}^{k-1} 2^i$
  i dla $k = n - 1$ otrzymujemy rozwiązanie

Rekurencja liniowa

- $fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$
- ogólniej $f(n) = \Sigma_{i=1}^{d} c_i \cdot f(n-i)$ plus warunki początkowe dla $f(i)$, $i = 0, \ldots, d-1$
- rozwiązanie: f ma szansę być funkcją wykładniczą $f(n) = x^n$
- wówczas $x^n = \Sigma_{i=1}^{d} c_i \cdot x^{n-i}$
  musimy to równanie rozwiązać
  nie ma problemu dla $d = 1, 2$, trudniej dla $d = 3$, nie ma szans dla $d > 4$

Rekurencja liniowa, przykład Fibonacci

- $fib(n) = \begin{cases} 0 & n = 0 \\ 1 & n = 1 \\ fib(n-1) + fib(n-2) & n > 1 \end{cases}$
- testujemy rozwiązanie $fib(n) = x^n$
  $x^{n+2} = x^{n+1} + x^n$ dla $n \geqslant 0$
  po podzieleniu przez x^n jest to równanie kwadratowe
  ma dwa pierwiastki $\frac{1 \pm \sqrt{5}}{2}$
- rozwiązanie musi być postaci $fib(n) = A \cdot (\frac{1+\sqrt{5}}{2})^n + B \cdot (\frac{1-\sqrt{5}}{2})^n$
  podstawiając $n = 0$ oraz $n = 1$ otrzymujemy
  $0 = A + B$
  $1 = A \cdot \frac{1+\sqrt{5}}{2} + B \cdot \frac{1-\sqrt{5}}{2}$
- czyli $fib(n) = \frac{1}{\sqrt{5}} \cdot (\frac{1+\sqrt{5}}{2})^n - \frac{1}{\sqrt{5}} \cdot (\frac{1-\sqrt{5}}{2})^n$

Rekurencja liniowa c.d.

- pierwiastków może być więcej, kombinacje liniowe funkcji wykładniczych też będą rozwiązaniami
- tw. jeśli x jest pierwiastkiem wielokrotnym równania, to również funkcje $n \cdot x^n$, $x^2 \cdot x^n$ itd. dla większych krotności będą spełniać równanie rekurencyjne
- na końcu dobieramy współczynniki kombinacji liniowej by spełniać warunki początkowe
- jest d warunków początkowych, równanie stopnia d ma d pierwiastków (licząc krotności), kombinacja liniowa d funkcji ma d współczynników
  czyli rozwiązujemy równanie liniowe z d niewiadomymi

Rekurencja liniowa, przykład Fibonacci c.d.

- ile jest pokryć prostokąta $2 \times n$ kostkami domina 2×1?
  $f(0) = 1$ (pusty prostokąt ma jedno pokrycie – puste),
  $f(1) = 1$,
  $f(n) = f(n-1) + f(n-2)$ (na początku pionowe domino lub dwa poziome)
  czyli $f(n) = fib(n+1)$
- ile jest ciągów n bitów bez dwóch kolejnych zer?
  $f(0) = 1$ (pusty ciąg jest OK),
  $f(1) = 2$,
  $f(n) = f(n-1) + f(n-2)$ (na początku jest jedynka lub zero, jeśli zero to kolejny bit musi być jedynką)
  czyli $f(n) = fib(n+2)$

Sortowanie przez scalanie

- zadanie: dana tablica n liczb ($n > 0$) – ustawić w kolejności rosnącej
- rozwiązanie:
  - jeśli $n = 1$ zadanie już jest wykonane
  - w p.p. dzielimy tablicę na dwie części, sortujemy każdą z osobna, a potem scalamy dwie części do uzyskania jednej tablicy dla wszystkich liczb
- złożoność (oszacowanie z góry):
  $T_1 = 0$
  $T_n = 2 \cdot T_{n/2} + n - 1$ (na razie możemy założyć, że dzielenie nie stanowi problemu, n można „zaokrąglić"do potęgi dwójki)

Sortowanie przez scalanie – oszacowanie

- kilkukrotne podstawianie sugeruje wzór $T_n = 2^k \cdot T_{n/2^k} + \Sigma_{i=0}^{k-1} (n - 2^i)$
- kolejne podstawienie za $T_{n/2^k}$ potwierdza ten wzór dla $k+1$
- czyli jest prawdziwy dla k takiego, że $2^k = n$
- $T_n = n \cdot 0 + k \cdot n - n + 1 = n \cdot (\log n - 1) + 1$
- dla dowolnego n na pewno $T_n \leqslant 2 \cdot n\lceil \log n \rceil$

Dziel i rządź

- zadanie jest dzielone na mniejsze podzadania w pewnych proporcjach
- $T(n) = \Sigma_{i=1}^{k} a_i \cdot T(n/b_i) + g(n)$
  koszt wykonania podzadań plus koszt scalenie w całość
  $a_i \geqslant 0$, $b_i > 1$, $g(n) \geqslant 0$
- prostsza wersja: dzielimy na równe podzadania
  $T(n) = a \cdot T(n/b) + g(n)$
- Twierdzenie:
  - jeśli g rośnie wolniej niż $n^{\log_b(a)}$, to $T(n) \leqslant n^{\log_b(a)}$
  - jeśli g rośnie szybciej niż $n^{\log_b(a)}$, to T jest rzędu g
  - w p.p. T jest rzędu $n^{\log_b(a)} \cdot \log n$

Dziel i rządź – przykład

- szybkie mnożenie:
  $(a \cdot 2^n + b)(c \cdot 2^n + d) = a \cdot c \cdot 2^{2n} + (a \cdot d + b \cdot c) \cdot 2^n + b \cdot d$
  zamiast dwóch środkowych iloczynów można obliczyć jeden
  $(a + b) \cdot (c + d)$ i dalej za pomocą odejmowania
- czyli ogólny wzór na złożoność mnożenia ciągów n bitów jest
  $M(n) = 3 \cdot M(n/2) + O(n)$
- $O(n)$ rośnie wolniej niż $n^{\log_2 3}$ czyli $M(n) \leqslant n^{\log_2 3}$
- opracowano kolejne algorytmy przyspieszające mnożenie, w listopadzie 2020 opublikowano algorytm złożoności $n \log n$