Matematyka Dyskretna

Wykład 4
Indukcja i rekurencja

Wojciech Kordecki
wojciech.kordecki@collegiumwitelona.pl

Collegium Witelona
Wydział Nauk Technicznych i Ekonomicznych
Zakład Informatyki

Semestr letni 2023/24

Przypomnienie

Tautologia:

$$(p \wedge (p \implies q)) \implies q.$$

Przypomnienie

Tautologia:

$$(p \wedge (p \implies q)) \implies q.$$

Oznacza to, że jeśli

Przypomnienie

Tautologia:

$$(p \wedge (p \implies q)) \implies q.$$

Oznacza to, że jeśli

- zdanie p jest prawdziwe oraz

Przypomnienie

Tautologia:

$$(p \wedge (p \implies q)) \implies q.$$

Oznacza to, że jeśli

- zdanie p jest prawdziwe oraz
- prawdziwe jest wynikanie $p \implies q$,

Przypomnienie

Tautologia:

$$(p \wedge (p \implies q)) \implies q.$$

Oznacza to, że jeśli

- zdanie p jest prawdziwe oraz
- prawdziwe jest wynikanie $p \implies q$,
- to prawdziwe jest zdanie q.

Definicje (1)

Zasada indukcji matematycznej: ZIM.

Załóżmy, że dana jest pewna własność P liczb naturalnych.

$P(n)$ oznacza, że liczba n ma własność P. Każda liczba naturalna $n \geq n_0$ ma własność P, o ile spełnione są następujące dwa warunki:

1. podstawa indukcji: zachodzi $P(n_0)$,
2. krok indukcyjny: z tego, że zachodzi $P(n)$, wynika, że zachodzi $P(n+1)$.

Definicje (2)

Zasadę indukcji matematycznej można także sformułować inaczej. Niech $p(m), p(m+1), \ldots$ będzie ciągiem zdań. Jeśli:

(P) zdanie $p(m)$ jest prawdziwe oraz

(I) zdanie $p(k+1)$ jest prawdziwe, tylko jeśli zdanie $p(k)$ jest prawdziwe i $m \leq k$,

to wszystkie te zdania są prawdziwe.
Warunek (P) nazywamy *warunkiem początkowym*, natomiast (I) nazywamy *krokiem indukcyjnym*.

Iteracja

Procedura iteracyjna obliczania kolejnych wartości ciągu a_n, $n = N_0, \ldots, N$.
Potrzebna jest znajomość a_{N_0} i funkcji $a_{n+1} = f(a_n)$.

1. Obliczamy a_{N_0}.
2. Dla kolejnych $n \in (N_0 + 1, \ldots, N)$ obliczany $a_{n+1} = f(a_n)$ – pętla `for`.

Suma sześcianów liczb naturalnych (1)

Wzór na sumę sześcianów n pierwszych liczb naturalnych, $n \geq 1$:

$$1^3 + 2^3 + \cdots + n^3 = \left(\frac{n(n+1)}{2}\right)^2.$$

$n = 1$:

$$1^3 = 1 = \left(\frac{1 \cdot (1+1)}{2}\right)^2.$$

Zakładamy, że wzór zachodzi dla liczby naturalnej n. Pokażemy, że wzór jest prawdziwy dla liczby $n+1$, to znaczy:

$$1^3 + 2^3 + \cdots + n^3 + (n+1)^3 = \left(\frac{(n+1)(n+2)}{2}\right)^2.$$

Suma sześcianów liczb naturalnych (2)

$$1^3 + 2^3 + \cdots + n^3 + (n+1)^3 = \left(\frac{n(n+1)}{2}\right)^2 + (n+1)^3$$
$$= \left(\frac{n+1}{2}\right)^2 n^2 + \left(\frac{n+1}{2}\right)^2 4(n+1)$$
$$= \left(\frac{n+1}{2}\right)^2 (n^2 + 4n + 4) = \left(\frac{n+1}{2}\right)^2 (n+2)^2$$
$$= \left(\frac{(n+1)(n+2)}{2}\right)^2.$$

Na podstawie zasady indukcji matematycznej wzór jest prawdziwy dla każdej liczby naturalnej $n \geq 1$.

Liczby harmoniczne (1)

Liczby harmoniczne definiuje się jako sumę odwrotności kolejnych liczb naturalnych, to znaczy:

$$H_0 = 1$$

oraz dla $n > 0$:

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

Liczby harmoniczne (1)

Liczby harmoniczne definiuje się jako sumę odwrotności kolejnych liczb naturalnych, to znaczy:

$$H_0 = 1$$

oraz dla $n > 0$:

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}.$$

$$H_{2^n} \leq n + 1.$$

Liczby harmoniczne (2)

Podstawa indukcji:

$$H_{2^0} = H_1 = 1 \leq 1.$$

Krok indukcyjny:

$$\begin{aligned} H_{2^{n+1}} &= H_{2^n} + \frac{1}{2^n+1} + \frac{1}{2^n+2} + \cdots + \frac{1}{2^{n+1}} \\ &\leq H_{2^n} + \frac{1}{2^n} 2^n = H_{2^n} + 1 \leq (n+1) + 1. \end{aligned}$$

Liczby harmoniczne – iteracja

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$$

oraz

$$H_1 = 1,$$

czyli $N_0 = 1$ i obliczamy H_n dla kolejnych $n = 1, 2, \ldots$.

Liczby harmoniczne – iteracja

$$H_n = 1 + \frac{1}{2} + \cdots + \frac{1}{n}$$

oraz

$$H_1 = 1,$$

czyli $N_0 = 1$ i obliczamy H_n dla kolejnych $n = 1, 2, \ldots$.

$$H_{n+1} = H_n + \frac{1}{n+1}$$

oraz $H_1 = 1$.

Kolejne liczby harmoniczne

n	H_n
1	$H_1 = 1$
2	$H_1 + \frac{1}{2} = 1.500$
3	$H_2 + \frac{1}{3} = 1.833$
4	$H_3 + \frac{1}{4} = 2.083$
5	$H_4 + \frac{1}{5} = 2.283$
6	$H_5 + \frac{1}{6} = 2.450$
7	$H_6 + \frac{1}{7} = 2.593 \ldots$

Własność:

$$\lim_{n\to\infty} (H_n - \ln n) = \gamma = 0.577218.$$

γ – stała Eulera.

Kolejne liczby harmoniczne

n	H_n
1	$H_1 = 1$
2	$H_1 + \frac{1}{2} = 1.500$
3	$H_2 + \frac{1}{3} = 1.833$
4	$H_3 + \frac{1}{4} = 2.083$
5	$H_4 + \frac{1}{5} = 2.283$
6	$H_5 + \frac{1}{6} = 2.450$
7	$H_6 + \frac{1}{7} = 2.593 \ldots$

Własność:

$$\lim_{n\to\infty} (H_n - \ln n) = \gamma = 0.577218.$$

γ – stała Eulera.

$$H_7 - \ln 7 = 0.6469 \approx \gamma.$$

Wieże Hanoi – iteracja

Wieża składa się z trzech słupków: A, B, C. Na słupku A jest n krążków.
Zadanie: przenieść wszystkie krążki ze słupka A na C, posługując się słupkiem B jako buforem i stosując zasady:

- Można układać tylko mniejszy na większy krążek.
- Można przekładać tylko jeden krążek jednocześnie.
- Można brać tylko krążek z samej góry.

Wieże Hanoi – iteracja

Wieża składa się z trzech słupków: A, B, C. Na słupku A jest n krążków.
Zadanie: przenieść wszystkie krążki ze słupka A na C, posługując się słupkiem B jako buforem i stosując zasady:

- Można układać tylko mniejszy na większy krążek.
- Można przekładać tylko jeden krążek jednocześnie.
- Można brać tylko krążek z samej góry.

Algorytm iteracyjny składa się z następujących kroków:

1. przenieś najmniejszy krążek na kolejny słupek,
2. wykonaj jedyny możliwy do wykonania ruch, nie zmieniając położenia krążka najmniejszego,
3. powtarzaj punkty 1 i 2, aż do odpowiedniego ułożenia wszystkich krążków.

Rekurencja – rekursja

Rekurencja, zwana także rekursją (ang. *recursion*), to w logice, programowaniu i matematyce odwoływanie się definicji do samej siebie.

Algorytm rekurencyjny polega na tym, że rozwiązanie badanego problemu wyraża się za pomocą rozwiązań tego samego problemu dla danych o mniejszych rozmiarach.

Drzewo binarne – definicja

Drzewem binarnym T (ang. *binary tree*) o n wierzchołkach nazywa się drzewo puste $T = \emptyset$, gdy $n = 0$, lub trójkę $T = (L, r, P)$, gdzie r jest wierzchołkiem zwanym *korzeniem drzewa* (ang. *root*), L jest drzewem binarnym o l wierzchołkach (zwanym lewym poddrzewem), P jest drzewem binarnym o p wierzchołkach (zwanym prawym poddrzewem) oraz $l + p + 1 = n$.

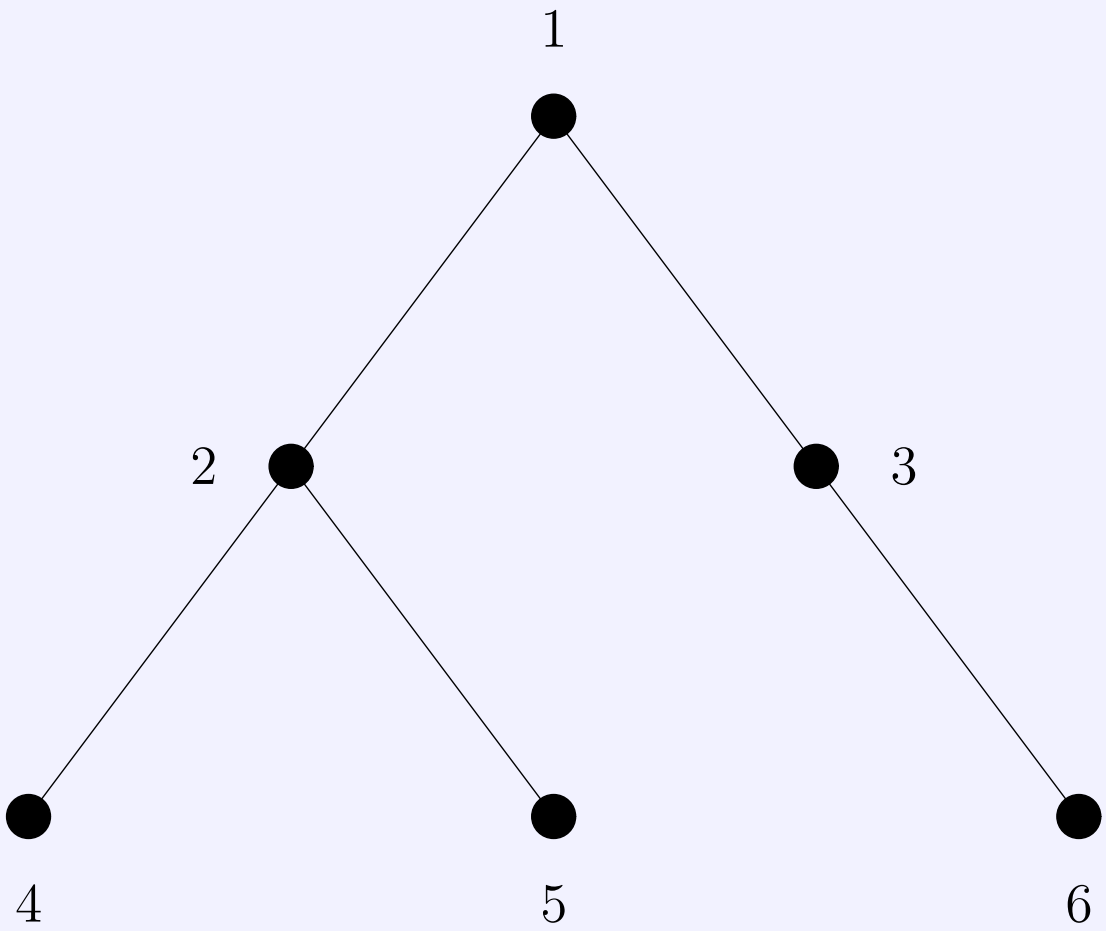
Rekurencja polega tutaj na tym, że każdy element trójki definiującej drzewo odwołuje się do informacji zawartej w wyznaczonych poprzednio mniejszych drzewach.

Drzewo binarne – konstrukcja

Drzewo o sześciu wierzchołkach.

- $T_1 = (T_2, 1, T_3)$,
- $T_2 = (T_4, 2, T_5)$, $T_3 = (\emptyset, 3, T_6)$,
- $T_4 = (\emptyset, 4, \emptyset)$, $T_5 = (\emptyset, 5, \emptyset)$, $T_6 = (\emptyset, 6, \emptyset)$.

Drzewo binarne – rysunek



Rekurencja w programowaniu

W informatyce rekurencja jest sposobem rozwiązania problemu przez zastosowanie algorytmu rekurencyjnego. Jego realizacją są obliczenia, w których wydzielony podprogram wywołuje sam siebie. Rekurencja jest stosowana w efektywnych algorytmach sortowania, między innymi w quicksort.

Rekurencja C++ (1)

Obliczanie $1 + 2 + \cdots + n$.

```
#include <iostream>
using namespace std;

long long suma(int n)
{
   if(n<1)
      return 0;

   return n+suma(n-1);
}
```

Rekurencja C++ (2)

```
int main()
{
   int n;

   cout<<"Podaj liczbę: ";
   cin>>n;

   cout<<"Suma "<<n<<"="
<<suma(n)<<endl;

   return 0;
}
```

Ciągi arytmetyczne i geometryczne

Najprostszymi definicjami rekurencyjnymi są:

- ciąg arytmetyczny – ciąg liczb rzeczywistych określonych wzorami:
  (P) $a_0 = a$,
  (R) $a_n = a_{n-1} + r$;
- ciąg geometryczny – ciąg liczb rzeczywistych określonych wzorami:
  (P) $b_0 = b$,
  (R) $b_n = b_{n-1}q$.

Liczby harmoniczne ponownie

$$H_1 = 1$$

$$H_n = H_{n-1} + \frac{1}{n} \text{ dla } n > 1.$$

Liczby harmoniczne ponownie

$$H_1 = 1$$
$$H_n = H_{n-1} + \frac{1}{n} \text{ dla } n > 1.$$

Znając tylko powyższe równanie, a nie znając wcześniej wzoru n H_n, można starać wyznaczyć H_n z tego równania. Jest to *równanie rekurencyjne*.

Ciąg zdefiniowany rekurencyjnie

Ciąg jest zdefiniowany rekurencyjnie, jeśli:

(P) określony jest pewien skończony zbiór wyrazów ciągu, zazwyczaj kilka pierwszych lub pierwszy wyraz,

(R) pozostałe wyrazy ciągu są zdefiniowane za pomocą poprzednich wyrazów ciągu.

Wzór definiujący ciąg w taki sposób nazywa się wzorem, równaniem lub zależnością rekurencyjną.
Definicja rekurencyjna odwołuje się do samej siebie, do elementu o mniejszym stopniu komplikacji, określa element ciągu poprzez odwołanie się do elementu poprzedzającego (jednego lub wielu).

Przykład rozwiązania rekurencji

Równanie rekurencyjne:

$$P_0 = 0,$$
$$P_{n+1} = P_n + n + 1 \text{ dla } n \geq 1.$$

Jego rozwiązanie to jak najprostsza formuła dająca sposób obliczania kolejnych wyrazów. Metoda, dzięki której możemy łatwo rozwiązać to konkretne równanie, to jego rozwinięcie, czyli cofanie się w rekursji:

$$P_n = P_{n-1} + n = P_{n-2} + (n-1) + n$$
$$= \cdots = P_0 + (1 + 2 + \cdots + n) = \frac{n(n+1)}{2}.$$

Liczby Fibonacciego

$$f_1 = 1,$$
$$f_2 = 1,$$
$$f_n = f_{n-2} + f_{n-1}.$$

Liczby Fibonacciego

$$f_1 = 1,$$
$$f_2 = 1,$$
$$f_n = f_{n-2} + f_{n-1}.$$

Kolejne liczby: 1,1,2,3,5,8,13,21,34,55,89,144,233,...

Liczby Fibonacciego

$$f_1 = 1,$$
$$f_2 = 1,$$
$$f_n = f_{n-2} + f_{n-1}.$$

Kolejne liczby: 1,1,2,3,5,8,13,21,34,55,89,144,233,...
Wzór Bineta

$$f_n = \frac{1}{\sqrt{5}} \left(\left(\frac{1+\sqrt{5}}{2} \right)^n - \left(\frac{1-\sqrt{5}}{2} \right)^n \right).$$

Algorytm w pseudokodzie

```
Fibonacci(n)
  if n=1 then return 1
  if n=2 then return 1
  f1 := 1
  f2 := 1
  for i := 3 to n
    do
      m := f1 + f2
      f1 := f2
      f2 := m
    end
  return f2
```

Algorytm rekurencyjny – problemy (1)

D. Kopec. *Klasyczne problemy informatyki w Pythonie*. PWN, Warszawa, 2020.

$$f_1 = 1,$$
$$f_2 = 1,$$
$$f_n = f_{n-2} + f_{n-1}.$$

```
def fib1(n:int) -> int:
    return fib1(n-2)+fib1(n-1)
```

Uruchamiamy:

```
if __name__=="__main__":
    print(fib1(5))
```

Algorytm rekurencyjny – problemy (2)

```
if __name__=="__main__":
    print(fib1(5))

Traceback (most recent call last):
  File "C:/Dom/Wojtek/moje_programy/Python/MD/fibo1.py", li
    print(fib1(5))
  File "C:/Dom/Wojtek/moje_programy/Python/MD/fibo1.py", li
    return fib1(n-2)+fib1(n-1)
  File "C:/Dom/Wojtek/moje_programy/Python/MD/fibo1.py", li
    return fib1(n-2)+fib1(n-1)
  File "C:/Dom/Wojtek/moje_programy/Python/MD/fibo1.py", li
    return fib1(n-2)+fib1(n-1)
  [Previous line repeated 1022 more times]
RecursionError: maximum recursion depth exceeded
```

Algorytm rekurencyjny – problemy (3)

```
def fib2(n:int) -> int:
    if n<2:
        return n
    return fib2(n-2)+fib2(n-1)
```

Uruchamiamy:

```
if __name__=="__main__":
    print(fib2(5))
    print(fib2(10))
```

```
5
55
```

Algorytm rekurencyjny – problemy (4)

Uruchamiany:

```
if __name__=="__main__":
    print(fib2(20))
    print(fib2(30))
    print(fib2(40))
```

```
6765
832040
102334155
```

Jak będzie dla $n = 50$?

Raczej się nie doczekamy wyniku!

Algorytm rekurencyjny – problemy (5)

Wywołania:

```
fib2(4)  fib(3)   fib(2)   fib2(1)
                           fib2(0)
                  fib2(1)
         fib(2)   fib2(1)
                  fib2(0)
```

Dla (20) mamy 21 891 wywołań.

Algorytm rekurencyjny – problemy (6)

Memoizacja:

```
from typing import Dict
memo: Dict[int,int]={0:0,1:1} # początek

def fib3(n:int) -> int:
    if n not in memo:
        memo[n]=fib3(n-1)+fib3(n-2) # memoizacja
    return memo[n]

if __name__=="__main__":
    print(fib3(50))
```

```
12586269025
```

Algorytm rekurencyjny – problemy (7)

Memoizacja automatyczna:

```
from functools import lru_cache

@lru_cache(maxsize=None)

def fib4(n:int) -> int:
    if n<2:
        return n
    return fib4(n-2)+fib4(n-1)


if __name__=="__main__":
    print(fib4(50))

12586269025
```

Algorytm rekurencyjny – problemy (8)

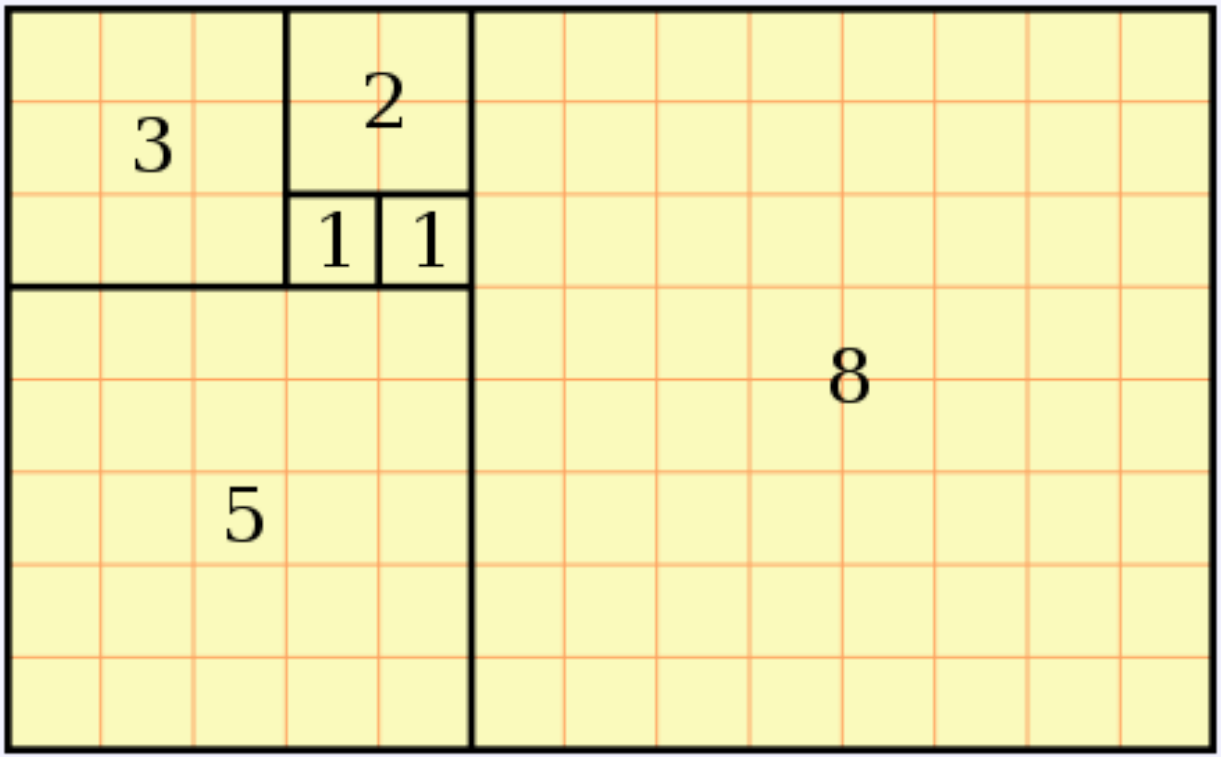
Wracamy do iteracji:

```
def fib5(n:int) -> int:
    if n==0: return n
    last:  int=0 # poczatkowo fib(0)
    next: int=1 # poczatkowo fib(1)
    for _ in range(1,n):
        last, next=next, last+next
    return next

if __name__=="__main__":
    print(fib5(50))
```

```
12586269025
```

Kwadraty z liczb Fibonacciego



Źródło: Wikipedia

Uogólniony ciąg Fibonacciego

Uogólniony ciąg liczb Fibonacciego:

$$f_n = f_{n-r} + f_{n-s} \mod m,$$

gdzie $n \geq r, r \geq s \geq 1$

Ciąg taki został wykorzystany w generatorach Fibonacciego liczb losowych.

Uogólniony ciąg Fibonacciego

Uogólniony ciąg liczb Fibonacciego:

$$f_n = f_{n-r} + f_{n-s} \mod m,$$

gdzie $n \geq r, r \geq s \geq 1$

Ciąg taki został wykorzystany w generatorach Fibonacciego liczb losowych.

Dla kompletnego zdefiniowania uogólnionych liczb Fibonacciego potrzebnych jest r pierwszych wyrazów ciągu.

Generator liczb losowych

Generator liczb losowych postaci:

$$x_n = x_{n-5} + x_{n-17} \mod 2^{32}.$$

Okres generatora liczb losowych x_n to najmniejsza liczba m taka, że $x_{n+m} = x_n$. Generator ten ma okres

$$\left(2^{17} - 1\right) 2^{29} = 70368207306752 \approx 7.03 \cdot 10^{13}.$$

Dla porównania: rok ma $1892160000000 \approx 1.89 \cdot 10^{12}$ milisekund.

Wieże Hanoi –rekurencja

Wieża składa się z trzech słupków: A, B, C. Na słupku A jest n krążków.
Zadanie: przenieść wszystkie krążki ze słupka A na C, posługując się słupkiem B jako buforem i stosując zasady:

- Można układać tylko mniejszy na większy krążek.
- Można przekładać tylko jeden krążek jednocześnie.
- Można brać tylko krążek z samej góry.

Wieże Hanoi –rekurencja

Wieża składa się z trzech słupków: A, B, C. Na słupku A jest n krążków.
Zadanie: przenieść wszystkie krążki ze słupka A na C, posługując się słupkiem B jako buforem i stosując zasady:

- Można układać tylko mniejszy na większy krążek.
- Można przekładać tylko jeden krążek jednocześnie.
- Można brać tylko krążek z samej góry.

Algorytm rekurencyjny składa się z następujących kroków:

1. przenieś (rekurencyjnie) $n-1$ krążków ze słupka A na słupek B posługując się słupkiem C,
2. przenieś jeden krążek ze słupka A na słupek C,
3. przenieś (rekurencyjnie) $n-1$ krążków ze słupka B na słupek C posługując się słupkiem A.

Implementacja: Python

```
def hanoi(n:int, A:list, B:list, C:list):
    """słupki A, B, C są listami"""
    if n > 0:
        hanoi(n-1, A, C, B)
        C.insert(0, A.pop(0))
        hanoi(n-1, B, A, C)
```

Źródło: Wikipedia

Implementacja: C++ (1)

```
#include <iostream>
using namespace std;

void hanoi(int n, char A, char B, char C)
{
  // przekłada n krążków z A korzystając z B na C
  if (n > 0)
  {
    hanoi(n-1, A, C, B);
    cout << A << " -> " << C << endl;
    hanoi(n-1, B, A, C);
  }
}
```

Implementacja: C++ (2)

```
int main(int argc, char *argv[])
{
  hanoi(3, 'A', 'B', 'C');
  return 0;
}
```

Źródło: Wikipedia

Liczba ruchów

Niech $f(n)$ oznacza liczbę ruchów koniecznych do przełożenia krążków.

Wzór rekurencyjny:

$$f(n) = 2f(n-1) + 1.$$

Wzór jawny:

$$f(n) = 2^n - 1.$$

Liczba ruchów

Niech $f(n)$ oznacza liczbę ruchów koniecznych do przełożenia krążków.

Wzór rekurencyjny:

$$f(n) = 2f(n-1) + 1.$$

Wzór jawny:

$$f(n) = 2^n - 1.$$

Obszerne omówienie w książce Rębowskiego, Matematyka Dyskretna oraz na licznych stronach internetowych.