Algorytmy rekurencyjne

dr Marcin Ziółkowski

Instytut Matematyki i Informatyki
Akademia im. Jana Długosza w Częstochowie

23 maja 2017 r.

ALGORYTMY REKURENCYJNE

Algorytmy rekurencyjne są bardzo ważnymi algorytmami z punktu widzenia informatyki. Wiele problemów rozwiązujemy w oparciu o algorytmy takiego typu. W ogólnym przypadku algorytmy rekurencyjne są algorytmami, które wywołują same siebie. Jest to sposób oparty na matematycznym pojęciu rekurencji. Należy jednak podkreślić, że algorytmów tych nie należy nadużywać. Jeżeli istnieje inny nierekurencyjny sposób rozwiązania danego problemu, będzie on z reguły skuteczniejszy. Spowodowane jest to faktem, że algorytmy rekurencyjne używają stosu do zapisywania wywołań i przy dużej liczbie wywołań (dużych argumentach) stos może szybko się przepełniać i program będzie działał bardzo wolno. Przedstawimy kilka przykładów algorytmów rekurencyjnych oraz programów napisanych na ich podstawie.

REKURENCYJNA DEFINICJA SILNII

W matematycznym sensie ciąg liczbowy jest zdefiniowany rekurencyjnie, gdy jest podanych kilka początkowych wyrazów tego ciągu oraz zasada (wzór), na podstawie którego możemy z poprzednich wyrazów uzyskiwać następne. Dla przykładu rozważmy matematyczne pojęcie silnii. Możemy ją zdefiniować rekurencyjnie w sposób następujący:

$$0! = 1$$

$$n! = (n-1)! \cdot n, \;\; n \geqslant 1$$

Jest jasne, że na podstawie wzorów rekurencyjnych bardzo łatwo pisze się funkcje wykorzystywane potem w programach. **Nie musimy bowiem wykorzystywać pętli iteracyjnych, stosujemy tylko odpowiednie wzory w definicjach funkcji i program jest gotowy.**

REKURENCYJNE OBLICZANIE SILNII - PROGRAM

OBLICZANIE SILNII - PROGRAM REKURENCYJNY

```
def silnia(n):
    if n==0:
        return 1
    else:
        return silnia(n-1)*n
def main():
    print("n=")
    n=int(input())
    print(n,"!=",silnia(n))
    input()
main()
```

REKURENCYJNA DEFINICJA POTĘGI O WYKŁADNIKU NATURALNYM

Niech $a \in R \setminus \{0\}$ oraz $n \in N$. Wówczas możemy zdefiniować potęgę a^n w sposób następujący:

$$a^0 = 1$$

$$a^n = a^{n-1} \cdot a, \ \ n \geqslant 1$$

Na podstawie tej definicji łatwo napisać program obliczający potęgi.

OBLICZANIE POTĘG - PROGRAM REKURENCYJNY

```
def potega(a,n):
    if n==0:
        return 1
    else:
        return potega(a,n-1)*a
def main():
    print("Podaj podstawę a:")
    a=float(input())
    print("Podaj wykładnik n:")
    n=int(input())
    print(a,"^",n,"=",potega(a,n))
    input()
main()
```

CIĄG FIBONACCIEGO

Ciąg Fibonacciego jest bardzo znanym matematycznym ciągiem mającym wiele zastosowań i odniesień. W ogólnym przypadku jego definicja rekurencyjna jest następująca:

$$F_0 = 1$$

$$F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2}, \ n \geqslant 2$$

Zatem ciąg rozpoczynają dwie jedynki, a kolejne wyrazy powstają z dodania dwóch bezpośrednio znajdujących się przed nimi wyrazów. Taki rodzaj definicji jest bardzo wygodny w porównaniu z definicją jawną:

$$F_n = \frac{1}{\sqrt{5}} \left[\left(\frac{1+\sqrt{5}}{2} \right)^{n+1} - \left(\frac{1-\sqrt{5}}{2} \right)^{n+1} \right]$$

Oto kilka początkowych wyrazów ciągu Fibbonacciego:

$$1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, \ldots$$

REKURENCYJNE OBLICZANIE WYRAZÓW CIĄGU FIBONACCIEGO - PROGRAM

OBLICZANIE WYRAZÓW CIĄGU FIBONACCIEGO - PROGRAM REKURENCYJNY

```
def fibonaci(n):
    if n==0 or n==1:
        return 1
    else:
        return fibonaci(n-1)+fibonaci(n-2)
def main():
    print("Podaj numer wyrazu n:")
    n=int(input())
    print("f_",n,"=",fibonaci(n))
    input()
main()
```

OBLICZANIE WYRAZÓW CIĄGU FIBONACCIEGO - WERSJA ITERACYJNA

Poprzedni program działa efektywnie dla niedużych wartości n. Już dla $n > 30$ program zaczyna działać bardzo wolno i w zasadzie staje się nieużyteczny. Ilość odwołań do poprzednich obliczeń rośnie wykładniczo. W tej sytuacji więc wykorzystanie rekurencji okazało się nieskuteczne. Można jednak rozwiązać ten problem nieco inaczej, używając pętli iteracyjnych.

OBLICZANIE WYRAZÓW CIĄGU FIBONACCIEGO - ITERACJA

```
def itfibonaci(n):
    if n==0 or n==1:
        return 1
    else:
        a,b=1,1
        for i in range(2,n+1):
            fib=a+b
            a=b
            b=fib
        return fib
def main():
    print("Podaj numer wyrazu n:")
    n=int(input())
    print("f_",n,"=",itfibonaci(n))
    input()
main()
```

OBLICZANIE NWD - WERSJA REKURENCYJNA

Innym przykładem wykorzystania rekurencji jest obliczanie największego wspólnego dzielnika. Zauważmy najpierw, że NWD może być obliczony w oparciu o następujący wzór:

$$NWD(0, b) = b$$

$$NWD(a, b) = NWD(b \ mod \ a, a), \ a > 0$$

Na przykład:
$NWD(24, 60) = NWD(12, 24) = NWD(0, 12) = 12$

OBLICZANIE NWD - PROGRAM

NWD - PROGRAM W WERSJI REKURENCYJNEJ

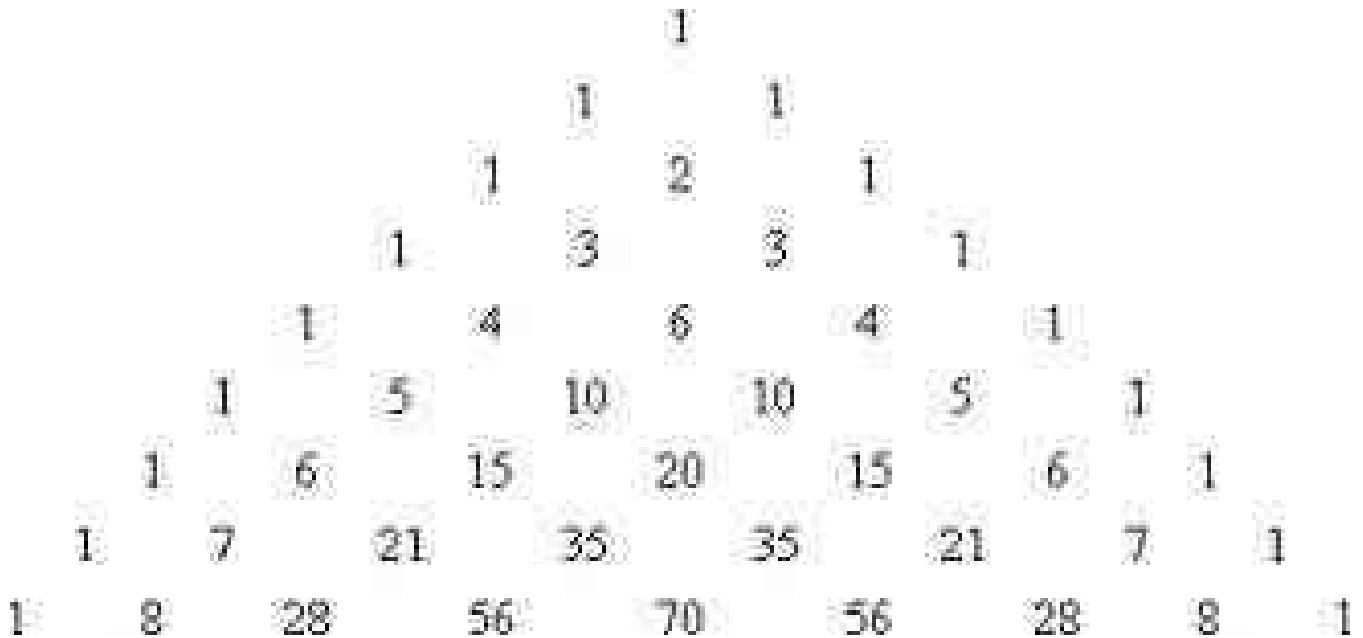
```
def NWD(a,b):
    if a==0:
        return b
    else:
        return NWD(b%a,a)
def main():
    a=int(input("Podaj pierwszą liczbę:"))
    b=int(input("Podaj drugą liczbę:"))
    print("NWD(",a,",",b,") wynosi", NWD(a,b))
    input()
main()
```

SYMBOL NEWTONA

Symbol Newtona $\binom{n}{k}$ jest obliczany zwykle w oparciu o definicję:

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

Jednak dla dużych wartości n oraz k jego obliczanie staje się niewygodne. Można jednak łatwo zauważyć, że można obliczać go również w oparciu o trójkąt Pascala.



SYMBOL NEWTONA

Wtedy możemy go zdefiniować w sposób rekurencyjny:

$$\binom{n}{0} = 1$$

$$\binom{n}{n} = 1$$

$$\binom{n}{k} = \binom{n-1}{k-1} + \binom{n-1}{k}, \ k \in (0, n)$$

Na kolejnym slajdzie przedstawiony jest program obliczający wartość symbolu Newtona dla podanych wartości n oraz k.

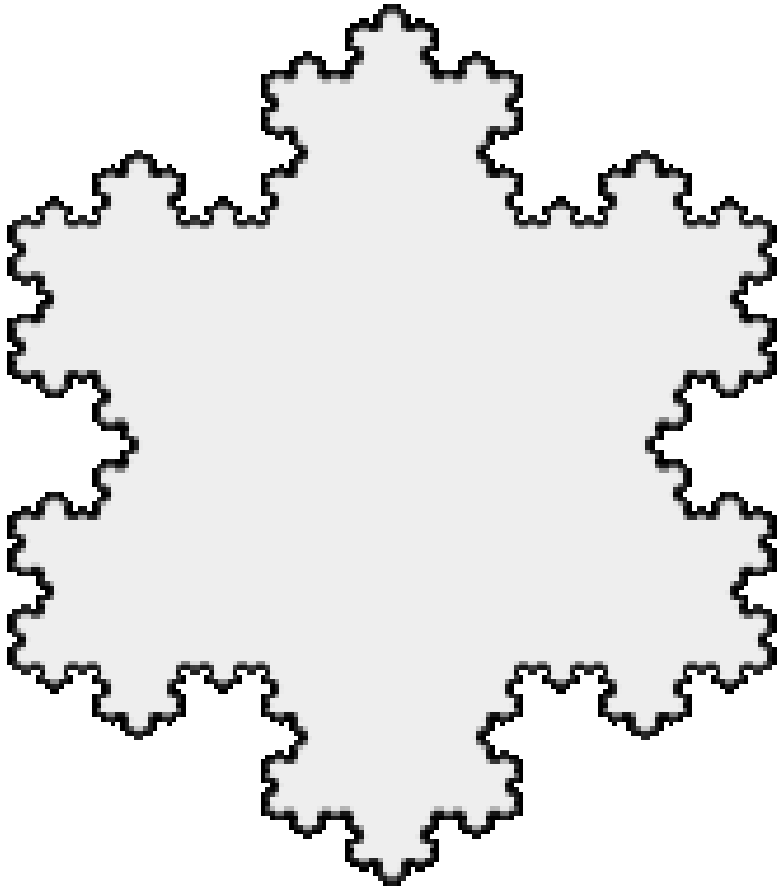
SYMBOL NEWTONA - PROGRAM W WERSJI REKURENCYJNEJ

```
def newton(n,k):
    if k==0 or k==n:
        return 1
    else:
        return newton(n-1,k-1)+newton(n-1,k)
def main():
    n=int(input("Podaj pierwszą (górną) liczbę:"))
    k=int(input("Podaj drugą (dolną) liczbę:"))
    print("[",n,",",k,"] wynosi", newton(n,k))
    input()
main()
```

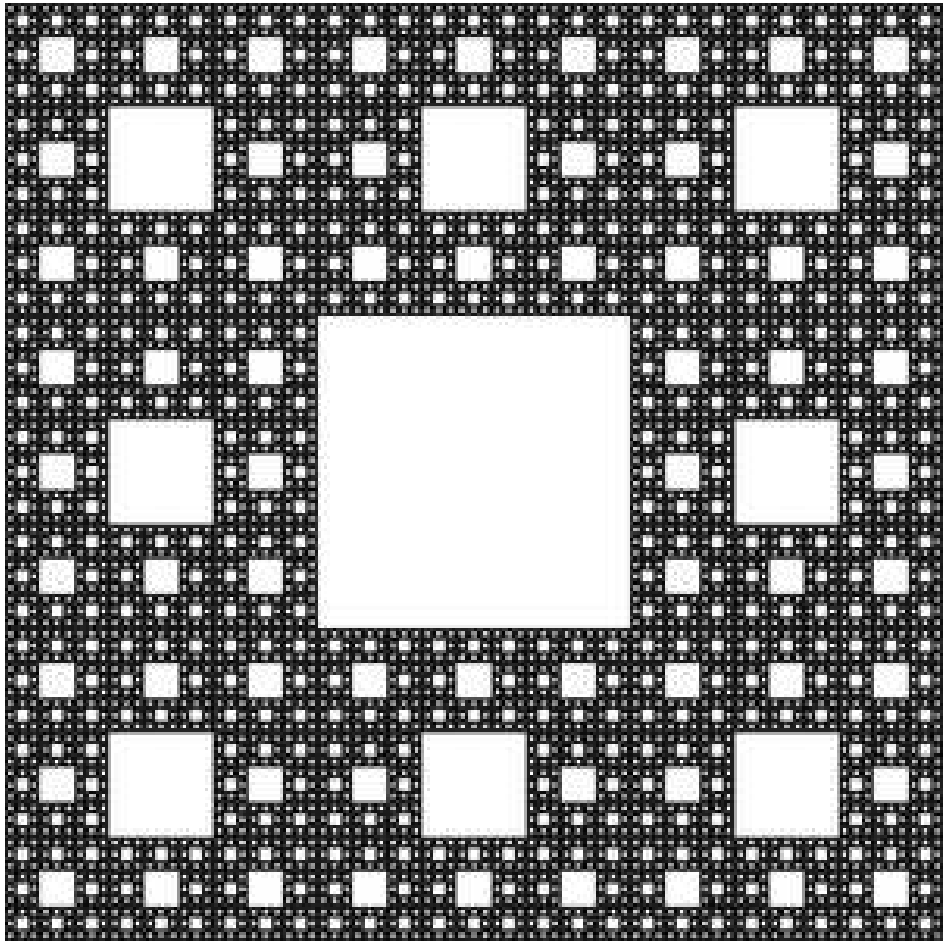
FRAKTALE

Fraktale są bardzo ciekawymi obiektami geometrycznymi, które cechuje samopodobieństwo (fragment fraktala jest podobny do całości). Mają one ciekawe własności np. fraktale dwuwymiarowe mają skończone pole i nieskończony obwód. Mają one również ułamkowy wymiar. Teoria fraktali mocno się rozwija, a dodatkowo frakale występują dość często w przyrodzie. Uważa się, że świat jest zbudowany z fraktali (jeśli się bliżej przyjrzeć). Fraktale są bardzo często wyrażane przez odpowiednie wzory rekurencyjne. Przedstawimy kilka ciekawych fraktali.

ŚNIEŻYNKA KOCHA



DYWAN SIERPIŃSKIEGO



ZBIORY JULII