Wstęp do programowania

Rekursja

O co chodzi w rekursji?

- Bardzo często właściwym podejściem do rozwiązywania problemów jest odpowiedzenie sobie na 2 pytania:
  - Czy potrafimy problem rozwiązać dla małych danych?
  - Czy jeśli mielibyśmy problem rozwiązany dla wszystkich danych mniejszego rozmiaru od aktualnie rozważanej danej d, to poradzilibyśmy sobie też z daną d?
- Jeśli na oba pytania odpowiemy twierdząco, to mamy rozwiązanie dla dowolnych danych
- Takie dwa założenia są podstawą myślenia rekurencyjnego o rozwiązywaniu problemów
- Rekursja jest informatycznym odpowiednikiem indukcji matematycznej.

Silnia

```
Function silnia(n:Integer): Integer;
{n>=0}
begin
 if n<=1 then silnia := 1
 else silnia := n * silnia(n-1)
end;
```

Wyszukiwanie liniowe

```
Function Jest(x: Integer;
   const A:tablica; n:Integer): Boolean;
{Jest przyjmie wartość true wttw gdy
  istnieje 1<=i<=n: A[i]=x}
begin
 if n<1 then Jest := false
 else if A[n]=x then Jest:=true
 else Jest:=Jest(x,A,n-1)
end;
```

Wyszukiwanie binarne w tablicy posortowanej

```
Function Jest(x: Integer;
 const A:tablica; l,p:Integer): Boolean;
{Jest przyjmie wartość true wttw gdy istnieje
  l<=i<=p: A[i]=x,          A[1}≤...≤A[n]}
var s:Integer; begin
 if l>p then Jest := false
 else if l=p then Jest:= A[l]=x
 else begin s:=(l+p) div 2;
      if x<=A[s] then Jest:=Jest(x,A,l,s)
                 else Jest:=Jest(x,A,s+1,p)
      end
end;
```

Wyznaczanie największej wartości w tablicy

```
Function Maks(const A:tab;n:Integer):Integer;
{Maks przyjmie wartość maksymalnej wartości
  wśród A[1..n], n>=1}
var x:Integer;
begin
 if n=1 then Maks :=A[1]
 else begin
  x:=Maks(A,n-1);
  if x>A[n] then Maks:=x
            else Maks:=A[n]
 end
end;
```

Wyznaczanie największej wartości w tablicy

```
Function Maks(const A:tab;n:Integer):Integer;
{Maks przyjmie wartość maksymalnej wartości
  wśród A[1..n]}
var x:Integer;
begin
 Maks :=A[n];
 if n>1 then begin
  x:=Maks(A,n-1);
  if x>A[n] then Maks:=x
 end
end;
```

Liczby Fibonacciego

```
Function Fibo(n: Integer):Integer;
{Oblicza n-tą liczbę Fibonacciego}
begin
 if n<=1 then Fibo:=n
 else Fibo:=Fibo(n-1)+Fibo(n-2)
end;
```

Liczby Fibonacciego (2)

```
Var F:array[0..n] of Integer;
Function FiboM(n: Integer):Integer;
{tym razem ze spamiętywaniem;
 F[i]<0 wttw gdy jeszcze nieznane}
begin
 if n<=1 then F[n]:=n
 else if F[n] < 0 then
      F[n]:=FiboM(n-1)+FiboM(n-2);
 FiboM:=F[n]
end;
```

Złożoność Fibo i FiboM

- Funkcja Fibo ma złożoność wykładniczą ze względu na n. Aby bowiem obliczyć Fibo(n) trzeba za każdym razem od nowa (!) obliczać Fibo(n-1) i Fibo(n-2). Jeśli przez T(n) oznaczymy liczbę wywołań rekurencyjnych, to dostaniemy równanie rekurencyjne:

  $T(0)=1;\ T(1)=1;\ T(n)=T(n-1)+T(n-2)+1$ dla $n>1$.

- Widać gołym okiem, że $T(n)\geq F(n)$.
- Natomiast funkcja FiboM ma złożoność liniową ze względu na n – w każdym wywołaniu odwołujemy się do 2 wywołań dla mniejszych wartości, co ma koszt stały.

Spamiętywanie

- Pamiętajmy: programując rekurencję nie możemy pozwolić sobie na to, żeby wielokrotnie wywoływać ją dla tych samych danych!
- Jeśli z jakichś powodów musimy tak robić, to należy spamiętywać uzyskane poprzednio wyniki i do nich zaglądać, zanim porwiemy się na wywołanie rekursji.
- Ta technika nazywa się spamiętywaniem (memoization) i jest powszechnie stosowana, np. w przypadku algorytmów z nawrotami.

Wieże Hanoi

- Wieże Hanoi wymyślił matematyk francuski Edouard Lucas w 1883, prowadząc rubrykę matematyczną w popularnej gazecie.
- Według jego zmyślonej opowieści, w Hanoi w jednej ze świątyń mnisi przekładają krążki między trzema prętami: za każdym razem jeden krążek i zawsze mniejszy krążek kładą na większym.
- Na początku wszystkie 64 krążki były na jednym z prętów uporządkowane malejąco do góry.
- Celem jest przeniesienie całej wieży na trzeci pręt za pomocą drugiego.

Wieże Hanoi

```
procedure przenies(co,s,d:Integer);
 begin Writeln(co,' z',s, ' na ',d) end;
procedure Hanoi(n,s,d:Integer);
 begin
  if n>0 then begin
   Hanoi(n-1,s,6-s-d);
   przenies(n,s,d);
   Hanoi(n-1,6-s-d,d)
  end
end;
```