Algorytmy i struktury danych - Funkcje, Rekurencja

Marcin Żurowski

12 października 2024

Plan zajęć

1. Funkcja
2. Rekurencja
3. Zadania

Funkcja - definicja

Funkcja (procedura, procedura funkcyjna, metoda) - wydzielona część algorytmu wykonująca pewne operacje, ewentualnie zwracająca ich wyniki i posiadająca nazwę.
do funkcji można przekazać pewne parametry (argumenty):

- **Parametry formalne** - obiekt występujący w treści funkcji, nie odpowiada mu bezpośrednio żaden obiekt głównego algorytmu.
- **Parametry aktualne** - wartość, która w momencie użycia funkcji w głównym algorytmie przyporządkowana do parametru formalnego.

Definicja funkcji:

```
procedure NAZWA(ParametryFormalne)
  SpecyfikacjaParametrów
  Instrukcje
```

Funkcja - definicja

Funkcja (procedura, procedura funkcyjna, metoda) - wydzielona część algorytmu wykonująca pewne operacje, ewentualnie zwracająca ich wyniki i posiadająca nazwę.
do funkcji można przekazać pewne parametry (argumenty):

- **Parametry formalne** - obiekt występujący w treści funkcji, nie odpowiada mu bezpośrednio żaden obiekt głównego algorytmu.
- **Parametry aktualne** - wartość, która w momencie użycia funkcji w głównym algorytmie przyporządkowana do parametru formalnego.

Definicja funkcji:

```
procedure NAZWA(ParametryFormalne)
  SpecyfikacjaParametrów
 Instrukcje
```

Funkcja - definicja

Funkcja (procedura, procedura funkcyjna, metoda) - wydzielona część algorytmu wykonująca pewne operacje, ewentualnie zwracająca ich wyniki i posiadająca nazwę.
do funkcji można przekazać pewne parametry (argumenty):

- **Parametry formalne** - obiekt występujący w treści funkcji, nie odpowiada mu bezpośrednio żaden obiekt głównego algorytmu.
- **Parametry aktualne** - wartość, która w momencie użycia funkcji w głównym algorytmie przyporządkowana do parametru formalnego.

Definicja funkcji:

```
procedure NAZWA(ParametryFormalne)
  SpecyfikacjaParametrów
 Instrukcje
```

Funkcja - definicja

Funkcja (procedura, procedura funkcyjna, metoda) - wydzielona część algorytmu wykonująca pewne operacje, ewentualnie zwracająca ich wyniki i posiadająca nazwę.
do funkcji można przekazać pewne parametry (argumenty):

- **Parametry formalne** - obiekt występujący w treści funkcji, nie odpowiada mu bezpośrednio żaden obiekt głównego algorytmu.
- **Parametry aktualne** - wartość, która w momencie użycia funkcji w głównym algorytmie przyporządkowana do parametru formalnego.

Definicja funkcji:

```
procedure NAZWA(ParametryFormalne)
  SpecyfikacjaParametrów
 Instrukcje
```

Funkcja - definicja

Funkcja (procedura, procedura funkcyjna, metoda) - wydzielona część algorytmu wykonująca pewne operacje, ewentualnie zwracająca ich wyniki i posiadająca nazwę.
do funkcji można przekazać pewne parametry (argumenty):

- **Parametry formalne** - obiekt występujący w treści funkcji, nie odpowiada mu bezpośrednio żaden obiekt głównego algorytmu.
- **Parametry aktualne** - wartość, która w momencie użycia funkcji w głównym algorytmie przyporządkowana do parametru formalnego.

Definicja funkcji:

```
procedure NAZWA(ParametryFormalne)
  SpecyfikacjaParametrów
 Instrukcje
```

Funkcja - przykład

```
procedure ODWRACANIE_TABLICY(A,n)
  integer n
  integer array A[1..n]
 for i = 1 to n DIV 2
  A[i] <-> A[n + 1 - i]
```

Zdefiniowane funkcje można wielokrotnie wywoływać w głównym algorytmie.

Funkcja - przykład

```
procedure ODWRACANIE_TABLICY(A,n)
  integer n
  integer array A[1..n]
 for i = 1 to n DIV 2
  A[i] <-> A[n + 1 - i]
```

Zdefiniowane funkcje można wielokrotnie wywoływać w głównym algorytmie.

Funkcja - wywołanie

Wywołanie funkcji:

```
NAZWA(ParametryAktualne)
```

Funkcja - wywołanie

Wywołanie funkcji:

```
NAZWA(ParametryAktualne)
```

Funkcja - przykład

```
for i = 1 to 5
 T[i] = 2 * i
ODWRACANIE_TABLICY(T,5)
```

Rekurencja

Rekurencja - odwoływanie się definicji lub funkcji do samej siebie

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \in \mathbb{N}_+ \end{cases}$$

Każda definicja rekurencyjna musi zawierać co najmniej jeden przypadek bazowy (nierekurencyjny).

Rekurencja

Rekurencja - odwoływanie się definicji lub funkcji do samej siebie

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \in \mathbb{N}_+ \end{cases}$$

Każda definicja rekurencyjna musi zawierać co najmniej jeden przypadek bazowy (nierekurencyjny).

Rekurencja

Rekurencja - odwoływanie się definicji lub funkcji do samej siebie

$$n! = \begin{cases} 1 & \text{dla } n = 0 \\ n(n-1)! & \text{dla } n \in \mathbb{N}_+ \end{cases}$$

Każda definicja rekurencyjna musi zawierać co najmniej jeden przypadek bazowy (nierekurencyjny).

Rekurencja - przykład

Algorytm rekurencyjny:

```
procedure SILNIA(n)
  integer n
 if n=0
  return 1
 else
  return n * SILNIA(n - 1)
```

Algorytm iteracyjny:

```
procedure SILNIA(n)
  integer n, s
 s = 1
 for i = 1 to n
  s = s * i
 return s
```

Rekurencja - przykład

Algorytm rekurencyjny:

```
procedure SILNIA(n)
  integer n
 if n=0
  return 1
 else
  return n * SILNIA(n - 1)
```

Algorytm iteracyjny:

```
procedure SILNIA(n)
  integer n, s
 s = 1
 for i = 1 to n
  s = s * i
 return s
```

Rekurencja - przykład

Algorytm rekurencyjny:

```
procedure SILNIA(n)
  integer n
 if n=0
  return 1
 else
  return n * SILNIA(n - 1)
```

Algorytm iteracyjny:

```
procedure SILNIA(n)
  integer n, s
 s = 1
 for i = 1 to n
  s = s * i
 return s
```

Rekurencja - przykłady

Rekurencyjna wersja algorytmu odwracania zapisu dziesiętnego liczby całkowitej dodatniej n

```
procedure ODWRACANIE(n)
  integer n
 if n > 0
  READ(n MOD 10)
  ODWRACANIE(n DIV 10)
```

Rekurencja - przykłady

Rekurencyjna wersja algorytmu odwracania zapisu dziesiętnego liczby całkowitej dodatniej n

```
procedure ODWRACANIE(n)
  integer n
 if n > 0
  READ(n MOD 10)
  ODWRACANIE(n DIV 10)
```

Rekurencja - przykłady

Rekurencyjna wersja algorytmu odwracającego kolejność elementów w tablicy `A[1..n]`

Funkcja pomocnicza:

```
procedure ODWRACANIE_PODTABLICY(A,i,j)
  integer i, j
  integer array A[1..n]
 if i < j
  A[i] <-> A[j]
  ODWRACANIE_PODTABLICY(A,i + 1,j - 1)
```

Główny algorytm:

```
procedure ODWRACANIE_TABLICY(A,n)
  integer n
  integer array A[1..n]
 ODWRACANIE_PODTABLICY(A,1,n)
```

Rekurencja - przykłady

Rekurencyjna wersja algorytmu odwracającego kolejność elementów w tablicy `A[1..n]`

Funkcja pomocnicza:

```
procedure ODWRACANIE_PODTABLICY(A,i,j)
  integer i, j
  integer array A[1..n]
 if i < j
  A[i] <-> A[j]
  ODWRACANIE_PODTABLICY(A,i + 1,j - 1)
```

Główny algorytm:

```
procedure ODWRACANIE_TABLICY(A,n)
  integer n
  integer array A[1..n]
 ODWRACANIE_PODTABLICY(A,1,n)
```

Rekurencja - przykłady

Rekurencyjna wersja algorytmu odwracającego kolejność elementów w tablicy `A[1..n]`

Funkcja pomocnicza:

```
procedure ODWRACANIE_PODTABLICY(A,i,j)
  integer i, j
  integer array A[1..n]
 if i < j
  A[i] <-> A[j]
  ODWRACANIE_PODTABLICY(A,i + 1,j - 1)
```

Główny algorytm:

```
procedure ODWRACANIE_TABLICY(A,n)
  integer n
  integer array A[1..n]
 ODWRACANIE_PODTABLICY(A,1,n)
```

Ciąg Fibonacciego

$$F_n = \begin{cases} 0 & \text{dla } n = 0 \\ 1 & \text{dla } n = 1 \\ F_{n-1} + F_{n-2} & \text{dla } n \geq 2 \end{cases}$$

Ciąg Fibonacciego - rekurencyjnie

```
procedure FIB(n)
  integer n
 if n = 0
  return 0
 else
  if n = 1
   return 1
  else
   return FIB(n - 1) + FIB(n - 2)
```

Dziel i zwyciężaj

Dziel i zwyciężaj (divide and conquer, D & C) - metoda algorytmiczna polegająca na rekurencyjnym dzieleniu problemu na dwa lub więcej podobnych podproblemów tak długo, aż podproblemy te staną się trywialne.

Dziel i zwyciężaj

Dziel i zwyciężaj (divide and conquer, D & C) - metoda algorytmiczna polegająca na rekurencyjnym dzieleniu problemu na dwa lub więcej podobnych podproblemów tak długo, aż podproblemy te staną się trywialne.

Ciąg Fibonacciego - iteracyjnie z wykorzystaniem tablicy

```
procedure FIB(n)
  integer n, k
  integer array F[1..n]
 F[0] = 0
 F[1] = 1
 for k = 0 to n
  F[k] = F[k - 1] + F[k - 2]
 return F[n]
```

Programowanie dynamiczne

Programowanie dynamiczne (dynamic programming, DP) - metoda algorytmiczna polegająca na rozwiązywaniu podproblemów w kolejności najmniejszych do największych i zapisywaniu otrzymanych rozwiązań do późniejszego wykorzystania.

Programowanie dynamiczne

Programowanie dynamiczne (dynamic programming, DP) - metoda algorytmiczna polegająca na rozwiązywaniu podproblemów w kolejności najmniejszych do największych i zapisywaniu otrzymanych rozwiązań do późniejszego wykorzystania.

Ciąg Fibonacciego - iteracyjnie bez tablicy

```
procedure FIB(n)
  integer n, f0, f1, fk
 if n = 0
  return 0
 else
  if n = 1
   return 1
 f0 = 0
 f1 = 1
 for k = 2 to n
  fk = f0 + f1
  f0 = f1
  f1 = fk
 return fk
```

Współczynnik dwumianowy

$$\binom{n}{k} = \begin{cases} 1 & \text{dla } k = 0 \text{ lub } k = n \\ \binom{n-1}{k-1} + \binom{n-1}{k} & \text{dla } 0 < k < n \end{cases}$$

Współczynnik dwumianowy - rekurencyjnie

```
procedure BINOM(n,k)
  integer n,k
 if k = 0 or k = n
  return 1
 else
  return BINOM(n - 1, k - 1) + BINOM(n - 1, k)
```

Współczynnik dwumianowy - programowanie dynamiczne

```
procedure BINOM(n,k)
  integer n, k
  integer array B[1..n, 1..k]
 for i = 0 to n
  for j = 0 to MIN(i,k)
   if j = 0 OR j = i
    B[i,j] = 1
   else
    B[i,j] = B[i - 1,j - 1] + B[i - 1,j]
 return B[n,k]
```

1

Wyznaczyć zależność rekurencyjną określającą liczbę spójnych obszarów, na które dzieli płaszczyznę n prostych, z których żadne dwie nie są równoległe i żadne trzy nie przecinają się w jednym punkcie.

2

Znaleźć zależność rekurencyjną określającą liczbę różnych sposobów wejścia po schodach zbudowanych z n stopni, jeśli w każdym kroku można pokonać jeden lub dwa stopnie.