Wstęp do informatyki

Algorytmy i struktury danych

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

30 października 2009

Spis treści

Nazwa

Termin **algorytm** pochodzi od nazwiska perskiego astronoma i matematyka żyjącego na przełomie VIII i IX w n.e. W 825 roku Muhammad ibn Musa al-Chorezmi (al-Khawarizmy) napisał traktat, w którym podał wiele precyzyjnych opisów dotyczących różnych matematycznych reguł (np. dodawania czy mnożenia liczb dziesiętnych). W XII wieku dzieło to zostało przetłumaczone na łacinę jako *Algoritmi de numero Indorum*, co należało rozumieć następująco: *Algoritmi o liczbach Indyjskich*. Pojawiające się tutaj po raz pierwszy słowo *Algoritmi* było oczywiście inaczej zapisanym nazwiskiem matematyka. Większość ludzi rozumiała jednak tytuł bardziej jako *Algorytmy o liczbach Indyjskich* a stąd już blisko do *Algorytmy na liczbach indyjskich (arabskich)*. W ten oto sposób precyzyjnie opisaną metodę obliczeniową zaczęto nazywć algorytmem (łac. *algorismus*).

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Algorytm

Nieformalna definicja

Nie ma jednej uniwersalnej definicji algorytmu. W potocznym tego słowa znaczeniu, algorytm oznacza sposób postępowania, przepis na coś, schemat działania.

Bardziej formalnie

Algorytm - w matematyce oraz informatyce to skończony, uporządkowany ciąg jasno zdefiniowanych czynności, koniecznych do wykonania pewnego zadania.

- wyróżniony początek i koniec
- warunek jednoznaczności
- warunek dyskretności
- warunek uniwersalności
- warunek efektywności

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Miejsce

Miejsce algorytmu na przykładzie implementacji oprogramowania rozwiązującego postawiony problem.

- problem
- komputer (czas, wewnętrzna reprezentacja danych, oprogramowanie)
- język (dostępne konstrukcje i typy danych)
- **algorytm**
- program

Przetwarzane informacje

Ograniczona informacja

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

Ostatni punkt prowadzi nas do pojęcia typu danej (struktury danej).

Przetwarzane informacje

Ograniczona informacja

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

Ostatni punkt prowadzi nas do pojęcia typu danej (struktury danej).

Przetwarzane informacje

Ograniczona informacja

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

Ostatni punkt prowadzi nas do pojęcia typu danej (struktury danej).

Przetwarzane informacje

Ograniczona informacja

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

Ostatni punkt prowadzi nas do pojęcia typu danej (struktury danej).

Przetwarzane informacje

Ograniczona informacja

- Informacja przechowywan w komputerze i przez niego przetwarzana stanowi pewien niewielki wycinek rzeczywistości zawierający dane niezbędne do rozwiązania postawionego problemu.
- Musimy się zastanowić jakie informacje są nam niezbędne, jakie mogą pomóc a jakie są całkiem niepotrzebne.
- Musimy się zastanowić jak będziemy reprezentować wybrane przez nas informacje.

Ostatni punkt prowadzi nas do pojęcia typu danej (struktury danej).

Struktury danych

Struktura danych jest takim sposobem przechowywania danych w komputerze aby ułatwiać ich wykorzystanie. Często rozsądny wybór struktury danych pozwala na wykorzystanie efektywniejszych algorytmów.

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Typ danej

Najpopularniejszy podział rozróżnia typy proste, nazywany też typami wbudowanymi i typy złożone — typy złożone z typów prostych.
Do typów prostych zaliczamy zwykle:

- typy liczbowe (np. całkowity, zmiennoprzecinkowy, stałoprzecinkowy)
- typ znakowy (znaki alfanumeryczne)
- typ logiczny

Do typów złożonych (nazywanych też strukturami) zaliczamy zwykle:

- tablicę/listę
- słownik
- zbiór
- rekord
- plik
- kolejkę
- stos
- drzewo

Przykłady zapisu tablic w różnych językach

Ada:

```
-- definicja typu tablicowego
type TableType is array(1 .. 100) of Integer;
-- definicja zmiennej określonego typu tablicowego
MyTable : TableType;
```

Visual Basic:

```
Dim a(1 to 5,1 to 5) As Double
Dim MyIntArray(10) As Integer
Dim MySingleArray(3 to 5) As Single
```

Przykłady zapisu tablic w różnych językach

C:

```
char my_string[40];
int my_array[] = {1,23,17,4,-5,100};
```

Java:

```
int [] counts;
counts = new int[5];
```

PHP:

```
$pierwszy_kwartal = array(1 =>'Styczeń','Luty','Marzec');
```

Python:

```
mylist = ["List item 1", 2, 3.14]
```

Przykłady zapisu słownika

Python:

```
d = {"key1":"val1", "key2":"val2"}
x = d["key2"]
d["key3"] = 122
d[42] = "val4"
```

Metody opisu algorytmów

- język naturalny
  - teoretycznie łatwy do napisania (wypunktowanie czynności)
  - często mała precyzja
  - kłopoty implementacyjne
- schemat blokowy
  - duża przejrzystość i czytelność
  - odzwierciedla strukturę algorytmu wyraźnie zaznaczając występowanie rozgałęzień (punktów decyzyjnych)
  - kłopoty implementacyjne
- pseudojęzyk
  - ułatwia implementację
  - mniejsza przejrzystość

Metody opisu algorytmów

- język naturalny
  - teoretycznie łatwy do napisania (wypunktowanie czynności)
  - często mała precyzja
  - kłopoty implementacyjne
- schemat blokowy
  - duża przejrzystość i czytelność
  - odzwierciedla strukturę algorytmu wyraźnie zaznaczając występowanie rozgałęzień (punktów decyzyjnych)
  - kłopoty implementacyjne
- pseudojęzyk
  - ułatwia implementację
  - mniejsza przejrzystość

Metody opisu algorytmów

- język naturalny
  - teoretycznie łatwy do napisania (wypunktowanie czynności)
  - często mała precyzja
  - kłopoty implementacyjne
- schemat blokowy
  - duża przejrzystość i czytelność
  - odzwierciedla strukturę algorytmu wyraźnie zaznaczając występowanie rozgałęzień (punktów decyzyjnych)
  - kłopoty implementacyjne
- pseudojęzyk
  - ułatwia implementację
  - mniejsza przejrzystość

Metody opisu algorytmów

- język naturalny
  - teoretycznie łatwy do napisania (wypunktowanie czynności)
  - często mała precyzja
  - kłopoty implementacyjne
- schemat blokowy
  - duża przejrzystość i czytelność
  - odzwierciedla strukturę algorytmu wyraźnie zaznaczając występowanie rozgałęzień (punktów decyzyjnych)
  - kłopoty implementacyjne
- pseudojęzyk
  - ułatwia implementację
  - mniejsza przejrzystość

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Algorytm Euklidesa

Przyjrzyjmy się dobrze znanemu przykładowi — algorytmowi Euklidesa.

- **1.** Weźmy dwie liczby całkowite dodatnie: a i b.
- **2.** Jeśli $b = 0$ idź do **3.**, w przeciwnym razie wykonaj:
  - **2.1.** Jeśli $a > b$ to $a := a - b$.
  - **2.2.** W przeciwnym razie $b := b - a$.
  - **2.3.** Przejdź do **2.**
- **3.** a jest szukanym największym dzielnikiem.
- **4.** Koniec

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Symbole

- początek i koniec
- blok instrukcji
- decyzja/warunek
- łącznik
- zapis/odczyt

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Reguły

1. bloki połączone są zorientowanymi liniami
2. wykonywane są wszystkie instrukcje w bloku albo żadna
3. dalsze operacje nie zależą od poprzednich, chyba że zależności zostały przekazane za pomocą danych
4. kolejność wykonania operacji jest ściśle określona przez zorientowane linie łączące poszczególne bloki
5. do każdego bloku prowadzi dokładnie jedna linia
6. linie mogą się łączyć, a punkt połączenia nazywa się punktem zbiegu

Schemat blokowy algorytmu Euklidesa.

Instrukcje

Pseudojęzyk nie ma ścisłej definicji. Zwykle formą zapisu przypomina jeden z wielu występujących dziś języków proceduralnych, stanowiąc mieszaninę konstukcji zapożyczonych z wielu z nich, jak na przykład C, Java, PHP czy Python. Szczegóły nie związane z algorytmem (np. zarządzanie pamięcią) zwykle są pomijane. Często bloki kodu, np. występującego w pętli, zastępowane są krótkim wyrażeniem języka naturalnego.
Na potrzeby zajęć przyjmujemy następujące formy zapisu.

- instrukcja podstawienia

```
x:=y;
wiek:=12.6;
imie:="Piotr";
```

Instrukcje

Pseudojęzyk nie ma ścisłej definicji. Zwykle formą zapisu przypomina jeden z wielu występujących dziś języków proceduralnych, stanowiąc mieszaninę konstukcji zapożyczonych z wielu z nich, jak na przykład C, Java, PHP czy Python. Szczegóły nie związane z algorytmem (np. zarządzanie pamięcią) zwykle są pomijane. Często bloki kodu, np. występującego w pętli, zastępowane są krótkim wyrażeniem języka naturalnego.
Na potrzeby zajęć przyjmujemy następujące formy zapisu.

- instrukcja podstawienia

```
x:=y;
wiek:=12.6;
imie:="Piotr";
```

Instrukcje

- blok

```
begin
  instrukcje zawarte
  w bloku
end
```

Instrukcje

- instrukcja warunkowa

```
if (WARUNEK) then
begin
  TRUE
end
```

```
if (WARUNEK) then
begin
  TRUE
end
else
begin
  FALSE
end
```

- `WARUNEK` — wyrażenie zwracające wartość logiczną, np

```
x=7
x>12
x>12 and y<3
x=5 and (y=1 or z=2)
```

- `TRUE` (`FALSE`) — instrukcje wykonywane, gdy warunek jest prawdziwy (fałszywy)

Instrukcje

- instrukcja pętli do-while i while

```
do                          while (WARUNEK)
begin                       begin
  instrukcje                  instrukcje
end                         end
while (WARUNEK);
```

Instrukcje

- instrukcja pętli for

```
for i:=1 to 10 step 1 do
begin
  instrukcje
end

for i in NAZWA do
begin
  instrukcje
end
```

- `NAZWA` — nazwa zmiennej reprezentującej listę, słownik, kolejkę, zbiór itp.

Funkcja

Funkcja jako czarna skrzynka wykonująca określone zadanie.

- wywołanie funkcji:

```
NazwaFunkcji(argumenty);
x:=Funkcja(arg1,arg2,arg3);
```

- definicja funkcji (ciało funkcji):

```
function NazwaFunkcji(argumenty)
begin
  instrukcje
  return zwracanaWartosc;
end
```

Funkcja

Funkcja jako czarna skrzynka wykonująca określone zadanie.

- wywołanie funkcji:

```
NazwaFunkcji(argumenty);
x:=Funkcja(arg1,arg2,arg3);
```

- definicja funkcji (ciało funkcji):

```
function NazwaFunkcji(argumenty)
begin
  instrukcje
  return zwracanaWartosc;
end
```

Funkcja

Funkcja jako czarna skrzynka wykonująca określone zadanie.

- wywołanie funkcji:

```
NazwaFunkcji(argumenty);
x:=Funkcja(arg1,arg2,arg3);
```

- definicja funkcji (ciało funkcji):

```
function NazwaFunkcji(argumenty)
begin
  instrukcje
  return zwracanaWartosc;
end
```

Iteracja i rekurencja

Iteracja

Iteracja (łac. *iteratio*) to czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli.

Rekurencja

Rekurencja (ang. *recursion*, z łac. *recurrere*, przybiec z powrotem) to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

Iteracja i rekurencja

Iteracja

Iteracja (łac. *iteratio*) to czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli.

Rekurencja

Rekurencja (ang. *recursion*, z łac. *recurrere*, przybiec z powrotem) to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

Silnia iteracyjnie

```
n! = 1 * 2 * 3 * ... * n
```

Silnia rekurencyjnie

```
n! = n * (n-1)!
```

Silnia

```
function SilniaI(n)
begin
  i:=0;
  s:=1;
  while (i<n) do
  begin
    i:=i+1;
    s:=s*i;
  end
  return s;
end
```

```
function SilniaR(n)
begin
  if (n=0) then
  begin
    return 1;
  end
  else
  begin
    return n*SilniaR(n-1);
  end
end
```

Silnia iteracyjnie

```
n! = 1 * 2 * 3 * ... * n
```

Silnia rekurencyjnie

```
n! = n * (n-1)!
```

Silnia

```
function SilniaI(n)
begin
  i:=0;
  s:=1;
  while (i<n) do
  begin
    i:=i+1;
    s:=s*i;
  end
  return s;
end
```

```
function SilniaR(n)
begin
  if (n=0) then
  begin
    return 1;
  end
  else
  begin
    return n*SilniaR(n-1);
  end
end
```

Silnia iteracyjnie

```
n! = 1 * 2 * 3 * ... * n
```

Silnia rekurencyjnie

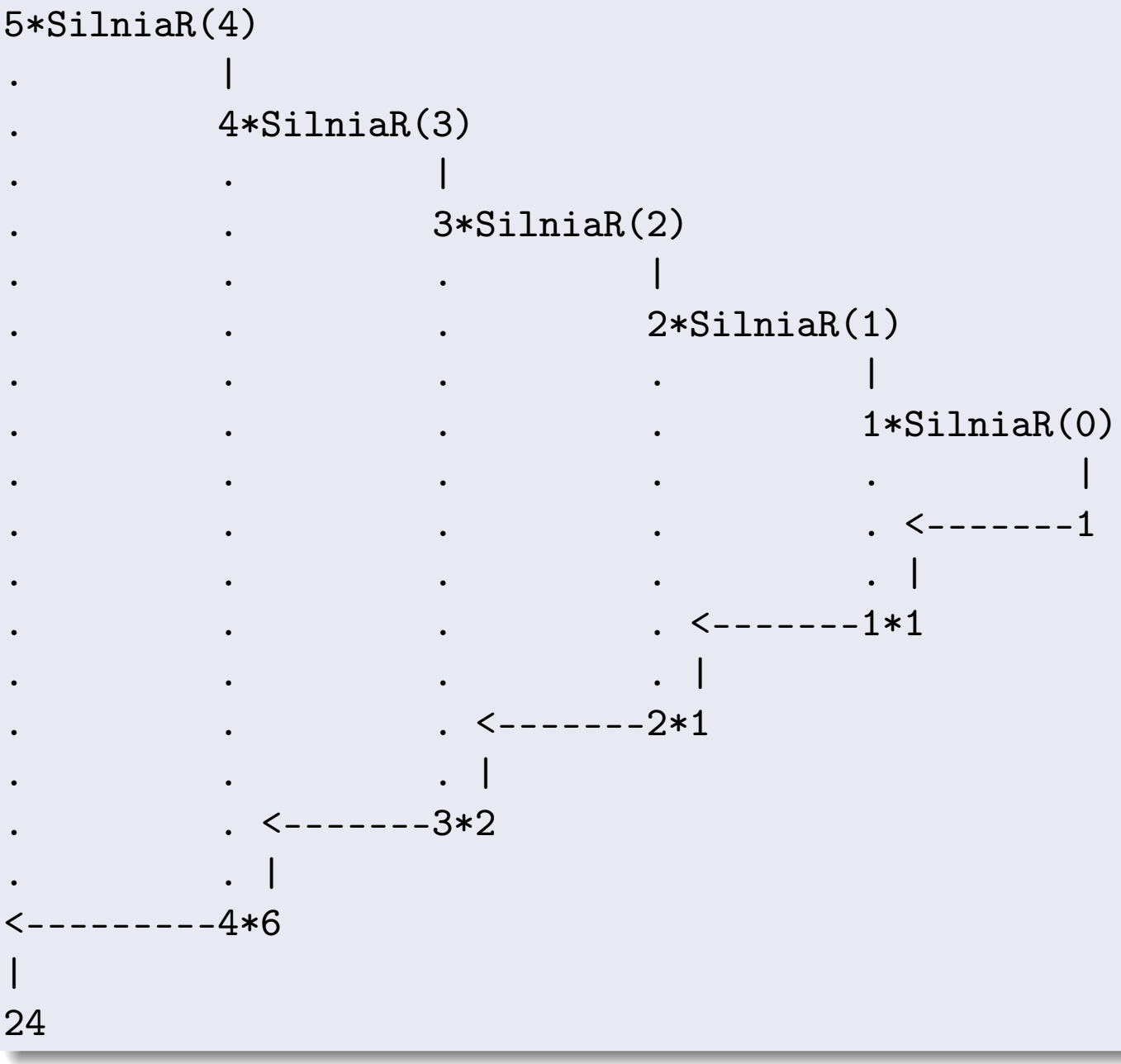
```
n! = n * (n-1)!
```

Silnia

```
function SilniaI(n)
begin
  i:=0;
  s:=1;
  while (i<n) do
  begin
    i:=i+1;
    s:=s*i;
  end
  return s;
end
```

```
function SilniaR(n)
begin
  if (n=0) then
  begin
    return 1;
  end
  else
  begin
    return n*SilniaR(n-1);
  end
end
```

Drzewo wywołań rekurencyjnych podczas obliczania wartości 4!



Definicja ciągu Fibonacciego

Dla $n > 1$ mamy

$$fib_n = fib_{n-1} + fib_{n-2},$$

natomiast wyrazy 1. i 0. przyjmują wartość 1.

Fibonacci rekurencyjnie

```
function FibR(n)
begin
  if ( n=0 or n=1) then
  begin
    return 1;
  end

  return FibR(n-1)+FibR(n-2);
end
```

Definicja ciągu Fibonacciego

Dla $n > 1$ mamy

$$fib_n = fib_{n-1} + fib_{n-2},$$

natomiast wyrazy 1. i 0. przyjmują wartość 1.

Fibonacci rekurencyjnie

```
function FibR(n)
begin
  if ( n=0 or n=1) then
  begin
    return 1;
  end

  return FibR(n-1)+FibR(n-2);
end
```

Czas

Drzewo wywołań rekurencyjnych podczas obliczania 5. wyrazu ciągu Fibonacciego

```
FibR(5)
      |
      +--FibR(4)
      |        |
      |        +--FibR(3)
      |        |        |
      |        |        +--FibR(2)
      |        |        |        |
      |        |        |        +--FibR(1)
      |        |        |        +--FibR(0)
      |        |        |
      |        |        +--Fib(1)
      |        |
      |        +--FibR(2)
      |                 |
      |                 +--FibR(1)
      |                 +--FibR(0)
      +--FibR(3)
               |
               +--FibR(2)
                  ...
```

Liczba wywołań

0	1
1	1
2	3
3	5
4	9
5	15
6	25
7	41
8	67
9	109
10	177
15	1973
20	21891
25	242785
30	2692537

Fibonacci iteracyjnie

```
function FibI(n)
begin
  i:=1;
  x:=1;
  y:=1;

  while (i<n)
  begin
    z:=x;
    i:=i+1;
    x:=x+y;
    y:=z;
  end

  return x;
end
```