

Wstęp do informatyki

Algorytmy i struktury danych

Piotr Fulmański

Wydział Matematyki i Informatyki,
Uniwersytet Łódzki, Polska

23 listopada 2015

- 1 Algorytm
- 2 Przetwarzane informacje
- 3 Struktury danych
- 4 Metody opisu algorytmów
- 5 Struktury danych a algorytm
- 6 Iteracja i rekurencja

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 *Algorithmi de numero Indorum*, co należało rozumieć następująco: *Algorithmi o liczbach Indyjskich*. Pojawiające się tutaj po raz pierwszy słowo *Algorithmi* 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 nazywać algorytmem (łac. *algorismus*).

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

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

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

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

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

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

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

Ograniczona informacja

- Informacja przechowywana 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).

Ograniczona informacja

- Informacja przechowywana 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).

Ograniczona informacja

- Informacja przechowywana 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).

Ograniczona informacja

- Informacja przechowywana 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).

Ograniczona informacja

- Informacja przechowywana 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

```
type tablica is array[1..10] of Integer
```

```
tablica: mojaTablica
```

```
mojaTablica[5] := 12
```

```
wypisz(mojaTablica[9])
```

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"
```

```
type osoba is record
  String: imie
  String: nazwisko
  Integer: wiek
end
```

```
osoba: o
```

```
o.wiek = 12
```

```
wypisz(o.imie)
```

Struktura danych a realizacja algorytmu

Uważny i właściwy dobór struktury danych ma zasadnicze znaczenie zarówno dla realizacji algorytmu jak i rozwiązania postawionego zadania. Dobrze dobrana struktura powinna nam ułatwiać rozwiązanie problemu w maksymalnym możliwym stopniu. Czasem wręcz sama w sobie jest tym rozwiązaniem. Przykłady zobaczymy w dalszej części gdy zapoznamy się już z metodami opisu algorytmów.

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

- 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

Metody opisu algorytmów

Schemat blokowy — algorytm Euklidesa

Schemat blokowy algorytmu Euklidesa.

Instrukcje

Pseudojęzyk nie ma ścisłej definicji. Zwykle formą zapisu przypomina jeden z wielu występujących dziś języków proceduralnych, stanowiąc mieszaninę konstrukcji 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 konstrukcji 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 for

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

```
for i in NAZWA do
begin
  instrukcje
end
```

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

Instrukcje

- instrukcja pętli do-while i while

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

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
```

Funkcja – wartość domyślna argumentu

```
nazwaFunkcji(arg1 = 12, arg2 = "text")
```

Możliwe wywołania

```
nazwaFunkcji(3, "test")
```

```
nazwaFunkcji(5)
```

```
nazwaFunkcji()
```

Funkcja – nazewnictwo

Konwencji związanych ze sposobem zapisywania nazw funkcji (a ogólnie mówiąc wszelkich „obiektów” jak np. zmiennych) jest kilka, ale my będziemy trzymać się chyba najpopularniejszej, tj. tzw. **camelCase**. CamelCase jest systemem zapisu ciągów tekstowych, w którym kolejne wyrazy pisane są łącznie, rozpoczynając każdy następny wielką literą (prócz pierwszego). Na przykład

```
jakasDziwnaNazwaFunkcji  
czasMinuty  
predkoscPojazduWKilometrachNaGodzine
```

Dbamy o to aby nazwy funkcji a także zmiennych i wszelkich innych „obiektów” oddawały to czym dany obiekt się zajmuje lub co reprezentuje.

Funkcja – zwracanie wartości

Zasadniczo przyjmujemy, że funkcja zwraca jedną wartość. Gdy chcemy zwrócić ich kilka, należy „upakować” je w jakiejś strukturze danych (tablica, lista, słownik, krotka (ang. *tuple*) lub struktura)

W Pythonie możliwy jest następujący zapis

```
def f(x):  
    y0 = x + 1  
    y1 = x * 3  
    y2 = y0 ** y3  
  
    return (y0, y1, y2)
```

...

```
y0, y1, y2 = f(2)
```

Funkcja – zwracanie wartości

Używając naszej składni dotyczącej zapisu funkcji napiszemy to samo jako

```
function f(x)
begin
  y0 := x + 1
  y1 := x * 3
  y2 := y0 ** y3

  return (y0, y1, y2)
end
...

y0, y1, y2 := f(2)
```

Funkcja – zwracanie wartości

```
type wynikT is record
```

```
  y0
```

```
  y1
```

```
  y2
```

```
end
```

```
function f(x)
```

```
begin
```

```
  wynikT: wynik
```

```
  wynik.y0 := x + 1
```

```
  wynik.y1 := x * 3
```

```
  wynik.y2 := y0 ** y3
```

```
  return wynik
```

```
end
```

Metody opisu algorytmów

Pseudojęzyk — argument czy parametr?

The terms parameter and argument are sometimes used interchangeably, and the context is used to distinguish the meaning. The term **parameter** (sometimes called formal parameter) is often used to refer to the variable as found in the function definition, while **argument** (sometimes called actual parameter) refers to the actual input passed. For example, in the function definition $f(x) = 2 \cdot x$ the variable x is a parameter; in the function call $f(2)$ the value 2 is the argument of the function.

Metody opisu algorytmów

Pseudojęzyk — argument czy parametr?

Please keep in mind that for example in the classical *The C Programming Language* by Brian W. Kernighan and Dennis M. Ritchie in section 5.10 *Command-line Arguments* we can find code like this

```
#include <stdio.h>

main(int argc, char *argv[])
{
    ...
}
```

where we can find not paramc but argc and not paramv but argv.

Metody opisu algorytmów

Pseudojęzyk — argument czy parametr?

Naming the first parameter `argc` is correct, rather than `paramc`. At run time the value we use is an argument. We reserve the term parameter for when discussing subroutine definitions.

According to authors explanation of this part of code: *When `main` is called, it is called with two arguments. The first (conventionally called `argc`, for argument count) is the number of command-line arguments the program was invoked with; the second (`argv`, for argument vector) is a pointer to an array of character strings that contain the arguments, one per string.*

Struktury danych a algorytm

Sortowanie drzewem

XXX

XXX

Struktury danych a algorytm

Wpływ użytej struktury na realizację algorytmu

Opis zadania

Dany jest graf G . Wszystkie węzły grafu są ponumerowane, dzięki czemu możemy mówić np. o wierzchołku 5. Możemy też powiedzieć jakie wierzchołki sąsiadują z zadany wierzchołkiem podając posortowaną od najmniejszej wartości tablicę (a wygodniej kolejkę) wierzchołków. W związku z tym na grafie G możemy wykonać następujące operacje

- 1 pobrać tablicę (kolejkę) wierzchołków sąsiednich dla danego wierzchołka n

`G.getNeighbours(n)`

Zadanie polega na sprawdzeniu, czy istnieje możliwość przejścia od danego wierzchołka n_{Start} do danego wierzchołka końcowego n_{Stop} .

Struktury danych a algorytm

Wpływ użytej struktury na realizację algorytmu

Algorytm

Dane wejściowe:

- graf G
- wierzchołek początkowy $nStart$
- wierzchołek końcowy $nStop$

Wynik działania:

- zwrócenie wartości logicznej `true` gdy istnieje w grafie G możliwość przejścia od zadanego wierzchołka $nStart$ do zadanego wierzchołka końcowego $nStop$; gdy taka ścieżka nie istnieje, zwrócenie wartości `false`

Struktury danych a algorytm

Wpływ użytej struktury na realizację algorytmu

Algorytm

Zmienne użyte w programie:

- `dataStructure` – pewna struktura danych, na której możemy wykonać następujące operacje
 - 1 pobrać wierzchołek ze struktury
`dataStructure.pop()`
 - 2 dodać wierzchołek `n` do struktury
`dataStructure.add(n)`
 - 3 sprawdzić czy wierzchołek `n` jest już w strukturze
`dataStructure.exist(n)`
- `explored` – będzie wyjaśnione na wykładzie

Struktury danych a algorytm

Wpływ użytej struktury na realizację algorytmu

```
function graphSearch(nStart, nStop, G) return true or false
{
  node := nStart
  if (node = nStop) then
    return TRUE
  dataStructure := ???
  explored := an empty set
  loop
  {
    if (the dataStructure is empty) then return FALSE
    node := dataStructure.pop()

    explored.add(node)

    for each child in G.getNeighbours(node)
    {
      if (child is not in explored or dataStructure) then
      {
        if (child = nStop)) then
          return TRUE
        dataStructure.add(child)
      }
    }
  }
}
```

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

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

$$n! = 1 * 2 * 3 * \dots * n$$

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

$$n! = 1 * 2 * 3 * \dots * n$$

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

$$n! = 1 * 2 * 3 * \dots * n$$

Silnia rekurencyjnie

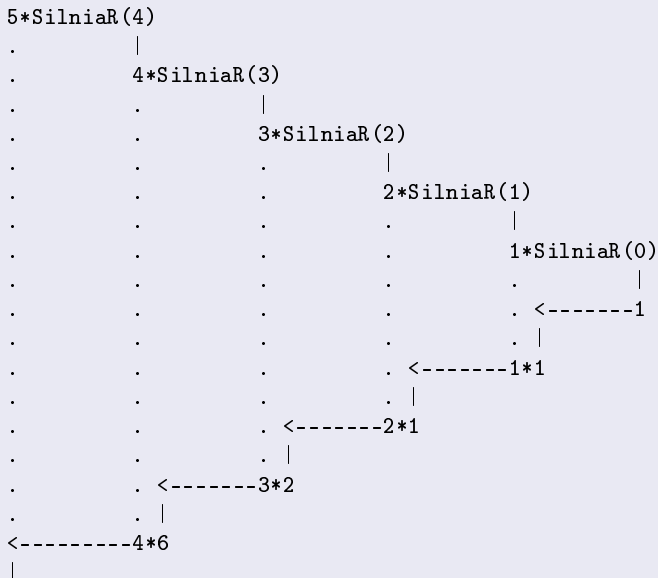
$$n! = n * (n-1)!$$

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.

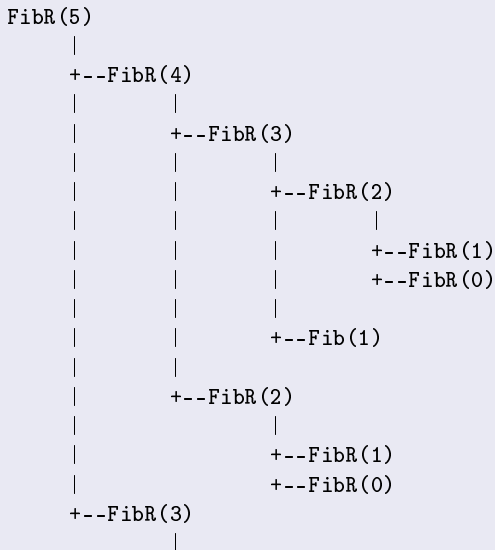
Fibonacci rekurencyjnie

```
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



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
  tmp :=0 // zmienna tymczasowa (pomocnicza)
  x:=1; // wyraz n-1
  y:=1; // wyraz n-2

  for i:=1 to n-1 step 1
  begin
    tmp:=y; // zapamiętaj wyraz n-2
    y:=y+x; // przesun wyraz n-2 na kolejną wartość ciągu
    x:=tmp; // przesun wyraz n-1 na kolejną wartość ciągu
              // czyli na wartość wyrazu n-1 przed jego
              // przesunięciem
  end

  return x;
end
```