Algorytmy i Struktury Danych

Prof. dr hab. Stanisław Gawiejnowicz
Wydział Biologii UAM
Semestr letni 2022/2023

Wykład nr 1: Podstawowe pojęcia i definicje

- Trzy główne pojęcia wykładu
- Problemy algorytmiczne
- Pojęcie algorytmu
- Sposoby zapisu algorytmów
- Przykłady algorytmów
- Krótka historia teorii algorytmów
- „Łatwe” i „trudne” problemy algorytmiczne
- Pojęcie programu
- Pojęcie struktury danych
- Elementy składowe algorytmów
- „Dobre” i „złe” algorytmy

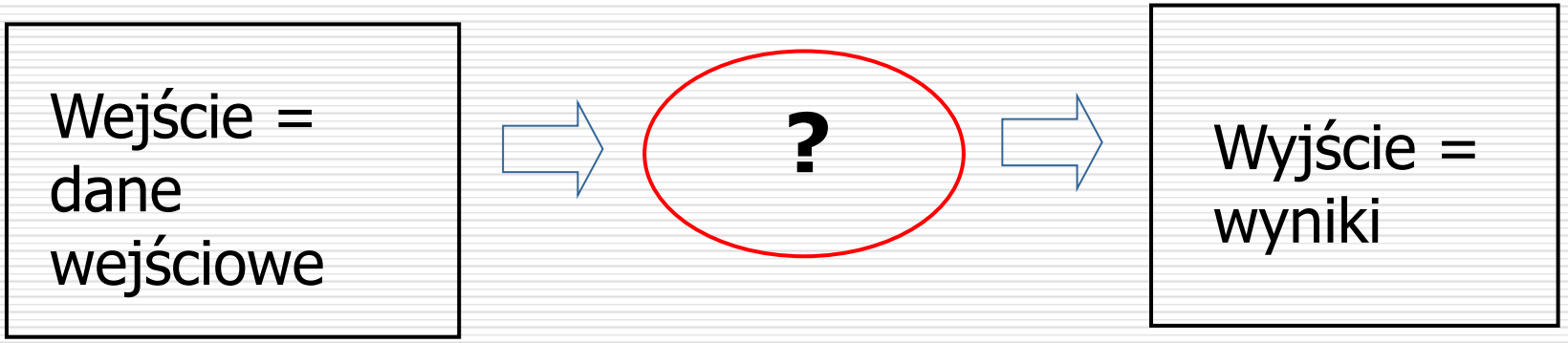
Trzy główne pojęcia wykładu

- W toku wykładu będą często występować trzy, wzajemnie powiązane pojęcia:
  - **algorytm**
  - **program**
  - **struktura danych**
- Pojęcia te będą przedstawiane w rożnych kontekstach, pozwalających na przedstawienie pewnej liczby pojęć pokrewnych

Problemy algorytmiczne

- Pod pojęciem **problem** będziemy rozumieć zagadnienie do rozwiązania podane w sformalizowanej postaci
- Opis problemu jest wyrażony za pomocą pewnych **parametrów**
- Parametry określają **dane wejściowe** problemu oraz warunki jakie muszą spełniać jego **wyniki**

Problemy algorytmiczne

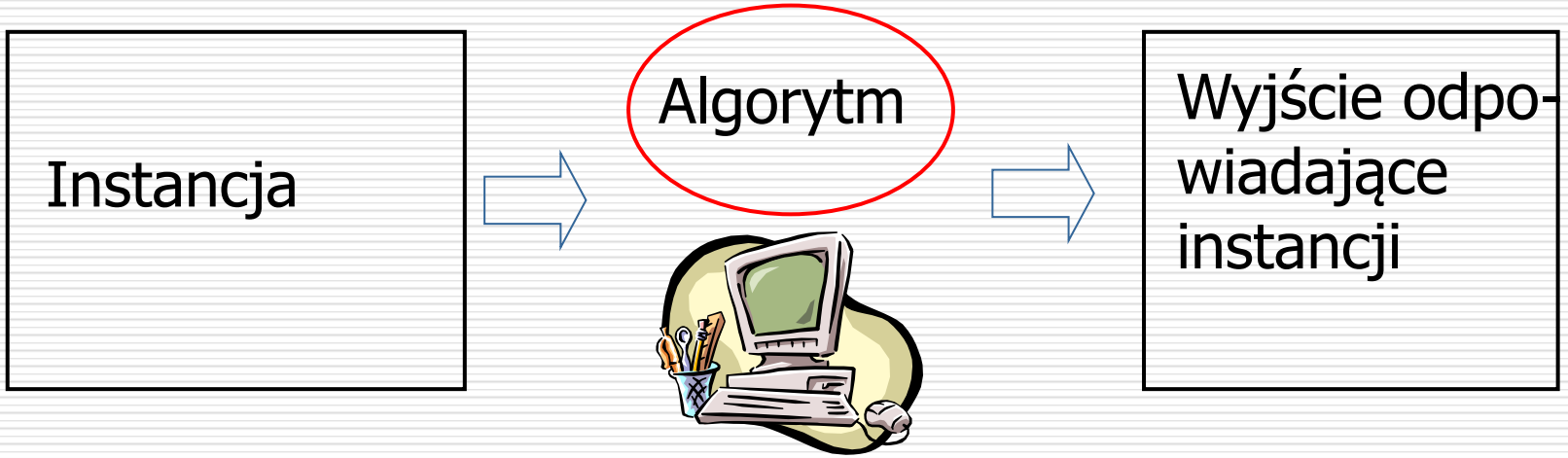


- Algorytm można zinterpretować jako przekształcenie transformujące dane wejściowe w wyniki
- Konkretne wartości przypisane parametrom opisującym dane wejściowe nazywamy **instancją** problemu

Problemy algorytmiczne

- Problem dla którego istnieje algorytm to **problem algorytmiczny**
- W dalszym toku rozważamy głównie problemy algorytmiczne
- Stąd też, o ile nie powiedziano inaczej, zamiast terminu **problem algorytmiczny** używamy terminu **problem**

Pojęcie algorytmu



- Algorytm opisuje jakie **elementarne operacje** są dokonywane na instancji danego problemu
- Dla problemu może istnieć wiele algorytmów

Pojęcie algorytmu

- Algorytm to ogólna procedura służąca do rozwiązania problemu algorytmicznego w skończonym czasie za pomocą sekwencji elementarnych operacji
- Własności rozważanych dalej algorytmów:
  - Precyzyjne sformułowanie
  - Determinizm
  - Skończoność
  - Poprawność
  - Ogólność

Pojęcie algorytmu

- Do formalizmów równoważnych pojęciu „algorytm” należą:
  - **rachunek λ** (Church:1939)
  - **funkcja obliczalna** (Rosser:1939, Kleeny:1943)
  - **maszyna Turinga** (Turing:1936-37)
  - **system produkcji dla symboli** (Post:1936)
  - **automat normalny** (Markow:1952)
- Wszystkie te formalizmy są równoważne w tym sensie, że to co się da obliczyć w skończonym czasie w jednym z nich, da się także obliczyć w skończonym czasie w pozostałych

Pojęcie programu

- Algorytm zapisany w **języku programowania** to **program**
- Innymi słowy, program to **implementacja** algorytmu w danym języku programowania
- Proces tworzenia programu obejmuje
  - wybór algorytmu
  - implementację algorytmu
  - **testowanie** napisanego programu
  - **uruchomienie** programu

Sposoby zapisu algorytmów (1/3)

- Algorytm można zapisać
  - słownie
  - graficznie
  - w postaci pseudokodu
  - w języku programowania

Sposoby zapisu algorytmów (2/3)

- słownie

Dane wejściowe: A[1..n], *q*
Wynik: *j takie, że* A[*j*] = *q lub NIL*, jeżeli $\forall j$ ($1 \leq j \leq n$): A[*j*] $\neq q$

```
Podstaw za j wartość 1
Tak długo jak j ≤ n oraz A[j] ≠ q
   wykonuj podstaw za j wartość o 1 większą
jeżeli j ≤ n to zwróć j
w przeciwnym przypadku zwróć NIL
```

Sposoby zapisu algorytmów (3/3)

- w postaci pseudokodu

INPUT: A[1..n], *q*
OUTPUT: *j* : A[*j*] = *q* or *NIL*, if $\forall j$ ($1 \leq j \leq n$): A[*j*] $\neq q$

```
j←1
while j ≤ n and A[j] ≠ q
   do j++
if j ≤ n then return j
else return NIL
```

Przykład 1: Wyszukiwanie

WEJŚCIE

- ciąg *n* liczb
- pojedyncza liczba

WYJŚCIE

- indeks znalezionej liczby lub *NIL*

$a_1, a_2, a_3, \ldots, a_n;\ q$ → j

2 5 7 13 17; 5 → 2

2 5 7 13 17; 9 → *NIL*

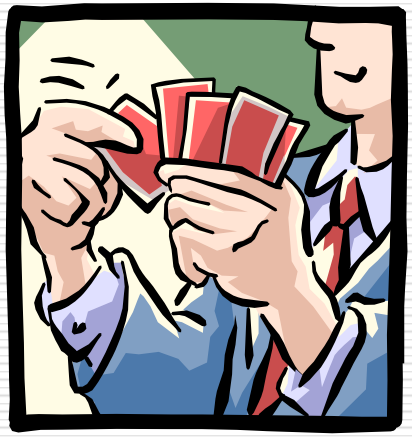
Przykład 2: Sortowanie

WEJŚCIE

ciąg *n* liczb

$a_1, a_2, a_3, \ldots, a_n$

2 5 4 10 7



WYJŚCIE

permutacja wejściowego ciągu liczb

$b_1, b_2, b_3, \ldots, b_n$

2 4 5 7 10

Poprawność algorytmu (wymagania dotyczące wyjścia)

Dla dowolnego danego wejścia algorytm kończy się generując wyjście:

- $b_1 \leq b_2 \leq b_3 \leq \ldots \leq b_n$
- ciąg $b_1, b_2, b_3, \ldots, b_n$ jest permutacją ciągu $a_1, a_2, a_3, \ldots, a_n$

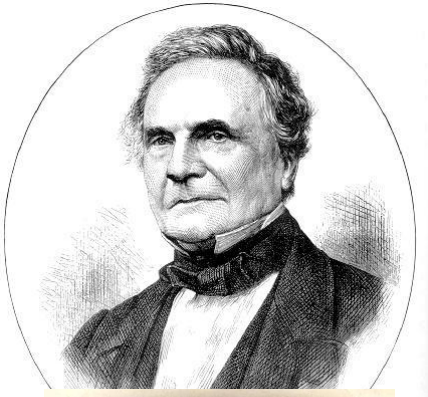
Krótka historia algorytmów (1/3)

- Nazwa „algorytm” pochodzi od nazwiska Mohammed ibn-Musa al-Chwarizmi (Mohammad, syn Musy z Chorezmu, 780-850)

- Pierwszy (?) algorytm: Euklides, algorytm znajdowania NWD dwu liczb, ok. 400-300 p.n.e.

Krótka historia algorytmów (2/3)

- Wkład w rozwój teorii algorytmów wniosło wielu matematyków
- Wiek XIX
  - Charles Babbage (1791-1871)
  - Ada Lovelace (1815-1852)



- Charles Babbage i Ada Byron pisali pierwsze programy!

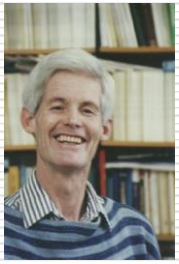
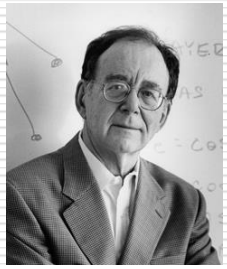
Krótka historia algorytmów (3/3)

- 1. poł. XX w.: Alan Turing, Emil Post, Alonzo Church, John von Neumann




- 2. poł. XX w.: Jack Edmonds, Leonid Levin, Stephen Cook, Richard Karp



„Łatwe” i „trudne” problemy

- Problemy dla których znane są algorytmy to problemy „łatwe”, gdyż można znaleźć rozwiązanie dla dowolnych instancji tych problemów
- Innymi słowy, problem jest „łatwy”, jeśli istnieje dla niego jakikolwiek algorytm
- Tak rozumiane „łatwe” problemy można podzielić na „bardziej łatwe” i „mniej łatwe”
- Stosowanym kryterium podziału jest „szybkość” algorytmów znanych dla danego problemu – problem A jest „bardziej łatwy” niż problem B, jeśli dla A istnieje „szybszy” algorytm niż dla B

„Łatwe” i „trudne” problemy

- **Nie wszystkie problemy można rozwiązać za pomocą algorytmów**
- Alan Turing i Emil Post udowodnili w 1936 roku istnienie **problemów niealgorytmicznych (nierozstrzygalnych)**
- Przykładami takich problemów są:
  - problem stopu maszyny Turinga
  - dziesiąty problem Hilberta

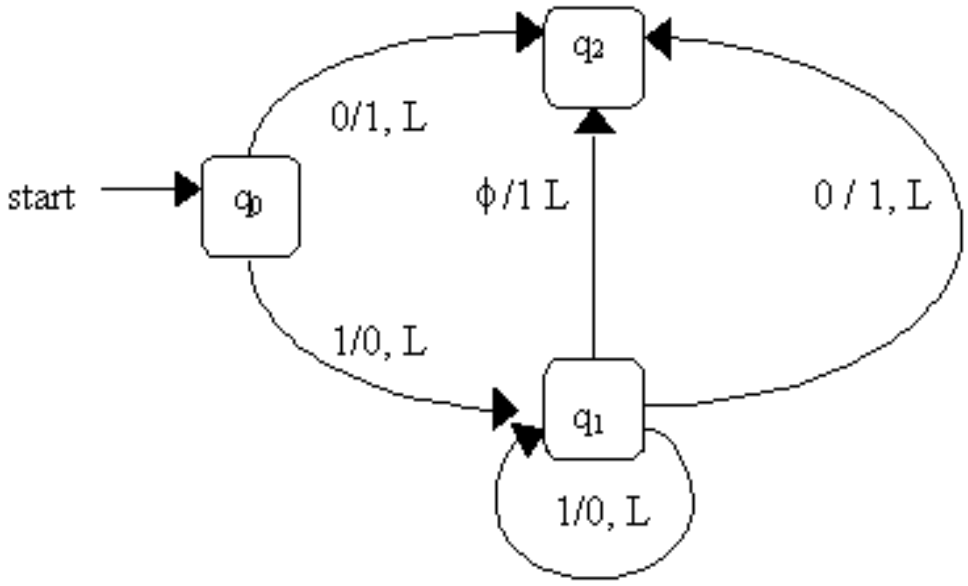
Maszyna Turinga

- Maszyna Turinga MT = $< Q, \Sigma, \delta, \Gamma, q_0, \varphi, F >$:
  - Q - skończony **zbiór stanów**
  - q_0 - **stan początkowy**, $q_0 \in Q$,
  - F - zbiór **stanów końcowych**
  - Γ - skończony **zbiór dopuszczalnych symboli**
  - φ - **symbol pusty**, $\varphi \in \Gamma$,
  - Σ - zbiór **symboli wejściowych**, podzbiór zbioru Γ,
  - δ - **funkcja przejścia** pobierająca aktualny stan maszyny oraz symbol wejściowy, dająca w wyniku symbol jaki ma się pojawić na taśmie, kolejny stan maszyny oraz przesunięcie głowicy maszyny (w lewo, w prawo lub bez przesunięcia)

Maszyna Turinga

- **Funkcja przejścia** (program) oraz **automat** odpowiadający pewnej MT

	q_0	q_1	q_2
ϕ	q_0 ϕ, L	q_2 1, L	K
0	q_2 1, L	q_2 1, L	K
1	q_1 0, L	q_1 0, L	K



Dziesiąty problem Hilberta

- Problem sformułowany przez Dawida Hilberta w 1900 roku
- Dotyczy istnienia procedury rozwiązywania dowolnego **równania diofantycznego**
- Problem ten rozwiązał w 1970 roku Yuri Matiyasevich

Elementy składowe algorytmu

- Algorytm zawiera opisy
  - danych wejściowych (**wejścia** algorytmu),
  - operacji dokonywanych na danych wejściowych (**ciała** algorytmu),
  - wyników (**wyjścia** algorytmu).
- Dane wejściowe algorytmu są opisywane za pomocą **deklaracji**
- Ciało algorytmu zawiera opis tego, co wykonuje algorytm za pomocą **operacji elementarnych**

Podstawowe operacje elementarne

- Do podstawowych operacji elementarnych należą:
- **operacje arytmetyczne**:
  - dodawanie, odejmowanie, mnożenie, dzielenie
- **operacja podstawienia**
- **operacje porównania**
- **operacje logiczne**
  - koniunkcja, alternatywa, negacja

Podstawowe operacje złożone

- Zbiór operacji elementarnych można powiększać o nowe operacje (złożone)
- **Operacje złożone** można definiować tylko za pomocą już istniejących operacji elementarnych

Podstawowe operacje złożone

- Przykłady złożonych operacji:
  - operacja wywołania funkcji/procedury (**call**)
  - operacja wczytania danych (**read**)
  - operacja wypisania wyników (**write**)
  - operacja wypisania komunikatu (**write**)
  - operacja przekazania wyniku (**return**)

Operatory

- **Operator** to znak lub symbol, który określa, jakiego typu obliczenia mają być wykonane w **wyrażeniu**
- Przykłady operatorów:
  - operatory elementarnych operacji arytmetycznych **+**, **-**, *****, **/**
  - operatory porównania **<**, **<=**, **>**, **>=**, **==**, **<>**
  - operatory logiczne **and**, **or**, **not**

Operatory

- Przykłady operatorów cd.:
  - operatory nieelementarnych operacji arytmetycznych (**mod**, **div**, **xchg**)
  - **operatory specjalne** (zależne od języka programowania): operator pobierania adresu **&**, operator konkatenacji **+**, **|** itp.

Pojęcie struktury danych

- Operacje elementarne algorytmu są wykonywane na danych
- Do przechowywania tych danych wykorzystywane są różnego rodzaju **struktury danych**
- Struktura danych określa sposób organizacji danych oraz zbiór dopuszczalnych operacji dokonywanych na tej strukturze
- Istnieje wiele rodzajów struktur danych

Pojęcie zmiennej oraz stałej

- Najprostsze struktury danych to
  - **stałe** oraz
  - **zmienne**
- **Stała** przechowuje niezmienną wartość wykorzystywaną przez algorytm
- **Zmienna** przechowuje zmieniającą się w czasie wartość obliczoną w trakcie działania algorytmu

Pojęcie zmiennej oraz stałej

- **Wartość stałej** nie zmienia się w trakcie wykonywania algorytmu
- **Wartość zmiennej** zmienia się w trakcie wykonywania algorytmu
- Do stałej (zmiennej) zwykle odwołujemy się za pomocą **nazwy** tej stałej (zmiennej)

Zmienne statyczne i dynamiczne

- Zasady tworzenia nazw stałych i zmiennych zależą od zasad przedstawiania algorytmu
- Zmienne do których odwołujemy się za pomocą nazwy to **zmienne statyczne**
- Istnieją **zmienne dynamiczne**, do których odwołujemy się za pomocą adresu ich położenia w pamięci operacyjnej

Pojęcie typu danych

- Zmienne bądź inne obiekty przechowujące dane mogą zawierać wartości różnych **typów danych**
- **Typem danych** nazywamy zbiór wartości wraz ze zbiorem operacji, których można dokonać na elementach tego zbioru

Pojęcie typu danych

- **Przykłady typów danych:**
  - **byte** – liczby nieujemne nie większe niż 255, operacje dodawania, odejmowania, mnożenia
  - **integer** - liczby całkowite z przedziału (-32768, 32767), operacje jak wyżej
  - **char** – zbiór znaków kodu ASCII (rozszerzonego), operacja konkatenacji

Pojęcie typu danych

- Operacje elementarne mogą realizować te same operacje arytmetyczne w odniesieniu do różnych typów danych
- **Przykład**
  - dodawanie (mnożenie) stałoprzecinkowe
  - dodawanie (mnożenie) zmiennoprzecinkowe
- Pojęcie typu danych jest właściwe dla algorytmów implementowanych w językach programowania, ma ono jednak także istotne znaczenie dla algorytmów w ogóle

Operacja podstawienia

- Operacja podstawienia postaci

  zmienna = wyrażenie

  powoduje nadanie zmiennej **zmienna** wartości wyrażenia **wyrażenie**
- Wartość wyrażenia **wyrażenie** oraz **zmienna** muszą być tego samego typu
- **Przykład:** instrukcja podstawienia

  x = 3.0

  jest niepoprawna, gdy x jest typu całkowitoliczbowego (np. typu **integer**)

Instrukcja podstawienia

- Po prawej stronie znaku = **zawsze** występuje wyrażenie
- **Przykład:**

y = y + 2

jest poprawną instrukcja podstawienia rozumianą następująco: „za y podstaw aktualną wartość y powiększoną o 2”

Podstawowe instrukcje sterujące

- O kolejności wykonywania operacji algorytmu decydują **instrukcje sterujące**
- **Podstawowe instrukcje sterujące to:**
  - sekwencja operacji
  - zdanie warunkowe
  - pętla sterowana licznikiem
  - pętla sterowana warunkiem

Sekwencja operacji

- Sekwencja operacji określa bezpośrednie następstwo operacji A i B, co zapisujemy

  A

  B

Przykład

x = 5*5-a

y = x/(x-1)

- **Uwaga:** W dalszym toku zakładamy, że A oraz B nie muszą być proste - mogą być złożone!

Zdanie warunkowe

- Zdanie warunkowe określa, która operacja zostanie wykonana w zależności od wartości wyrażenia logicznego „war”
- W najprostszym przypadku ma postać

  if war

  then A

  B

Zdanie warunkowe

- W pseudokodzie opisującym zdanie warunkowe występują dwa **słowa kluczowe**: **if** oraz **then**
- Słowo kluczowe **if** tłumaczymy jako „jeżeli”, słowo kluczowe **then** – „to”
- Zdanie warunkowe w podanej wyżej postaci powoduje wykonanie operacji A, gdy warunek „war” jest spełniony; wykonanie B ma miejsce niezależnie od tego czy jest on spełniony

Zdanie warunkowe

- **Przykład**

  if (X mod 2 == 0)

  then write „x jest parzyste”
- Symbol == oznacza **porównanie**
- Nawiasy po **if** nie są konieczne, zwiększają jednak czytelność pseudokodu
- Słowo kluczowe **write** tłumaczymy jako „pisz”

Zdanie warunkowe

- Zdanie warunkowe może też jawnie określać operację do wykonania w przypadku, gdy warunek „war” nie jest spełniony; jest wtedy postaci

  if war

  then A

  else B

- Słowo kluczowe **else** tłumaczymy „w przeciwnym przypadku”

Zdanie warunkowe

- **Przykład**

  if (n mod 2==0)

  then n = n div 2

  else n = 3*n+1

- **Uwaga:** powyższe zdanie warunkowe wykonywane tak długo, jak długo n jest różne od 1, wiąże się z tzw. **problemem Collatza**

Pętla sterowana licznikiem

- **Pętlą** nazywamy sekwencję operacji wykonywanych wielokrotnie
- Każde pojedyncze wykonanie pętli jest nazywane **iteracją**
- Wyróżniamy dwa rodzaje pętli:
  - sterowane licznikiem
  - sterowane warunkiem
- W przypadku pętli sterowanej licznikiem znamy liczbę wykonań pętli
- Liczbą wykonań w/w pętli steruje zmienna zwana **licznikiem pętli**

Pętla sterowana licznikiem

- Pętla sterowana licznikiem jest też nazywana pętlą **for**
- Pętla sterowana licznikiem ma postać

  for zm=wp **step** krok **to** wk **do** A

  gdzie
  - „zm” to licznik pętli,
  - „wp” to **wartość początkowa licznika**,
  - „krok” to **wartość kroku pętli** – wartość o jaką jest zwiększany licznik „zm” w każdej iteracji,
  - „wk” **wartość końcowa** licznika „zm”

Pętla sterowana licznikiem

- Słowo kluczowe **for** tłumaczymy „dla”
- Słowo kluczowe **to** tłumaczymy „do”
- Słowo kluczowe **step** tłumaczymy „z krokiem”
- Słowo kluczowe **do** tłumaczymy „wykonuj”

Pętla sterowana licznikiem

- Wartość początkowa licznika, rozmiar kroku oraz wartość końcowa są wartościami zmiennych lub wyrażeń
- Jeśli rozmiar kroku jest dodatni (ujemny), to pętla **for** wykonuje się tak długo, jak długo wartość początkowa jest nie większa (nie mniejsza) niż wartość końcowa

Pętla sterowana licznikiem

- Jeśli rozmiar kroku jest równy 1, to pętla sterowana licznikiem jest postaci

  for zm=wp **to** wk **do** A

- Jeśli rozmiar kroku jest równy -1, to pętla sterowana licznikiem jest postaci

  for zm=wp **downto** wk **do** A

Pętla sterowana licznikiem

- **Przykład**

1) **for** k = m **to** n **do** // m <= n
   A

2) k = m
et: **if** k <= n // etykieta
   then A
      k = k+1
      goto et // skok bezwarunkowy
...

Pętla sterowana licznikiem

- **Przykład**

1) **for** k = n **downto** m **do** // n >= m
    A

2) k = n
et: **if** k >= m // etykieta
    then A
        k = k-1
        goto et // skok bezwarunkowy
...

Pętla sterowana licznikiem

- **Przykłady**

1) s = 0

```
for i = 1 to n do
        s = s + i
```

2) s = 0

```
for i = n downto 1 do
        s = s + i
```

Pętla sterowana licznikiem

- **Przykład** obliczanie a^n, a≠0, n>=0

```
POTEGA1(a,n) // nagłówek algorytmu
b = a
for k=1 to n-1 do
    b=b*a
return b // zwrócenie wyniku
```

Pętla sterowana licznikiem

- **Przykład** obliczanie a^n, a≠0, n>=0

```
POTEGA2(a,n)
b = 1
for k=1 to n do
    b=b*a
return b
```

Pętla sterowana licznikiem

- **Przykład** obliczanie a^n, a≠0, n>=0

```
POTEGA3(a,n)
b = 1
for k=n downto 1 do
    b=b*a
return b
```

Pętle sterowane warunkiem

- Niekiedy nie wiemy, ile razy pętla się wykona, znamy jednak warunek, który musi być spełniony by mogła się wykonywać bądź warunek, który musi być spełniony by mogła się zakończyć
- Pętle tego typu są nazywane **pętlami sterowanymi warunkiem**

Pętle sterowane warunkiem

- W pierwszym przypadku pętla jest wykonywana tak długo, jak długo zachodzi warunek „war”
- Pętla tego typu (nazywana pętlą **while**) ma postać

  while war **do**

  A

Pętle sterowane warunkiem

- Słowo kluczowe **while** tlumaczymy „tak długo, jak”
- **Przykład**

```
read a
read b
while (not a == b) do
   if (a >= b)
   then a = a-b
   else b = b-a
return a
```

Pętle sterowane warunkiem

- **Przykład**

```
POTEGA4(a,n)
b=1
k=1
while k<>0 do
  b = b*a
  k = k-1
return b
```

Pętle sterowane warunkiem

- **Przykład** (wypisywanie cyfr liczby w odwrotnym porządku)

```
ODWROC1(n)
while n<>0 do
  cyfra = n mod 10
  write cyfra
  n = n div 10
return // koniec bez zwracania wyniku
```

Pętle sterowane warunkiem

- Jeśli znamy warunek, który musi być spełniony by pętla przestała się wykonywać, to mamy do czynienia z drugim rodzajem pętli sterowanej warunkiem – pętlą **repeat .. until**
- Pętla tego typu jest postaci

  repeat

  A

  until war
- Słowa kluczowe **repeat .. until** tłumaczymy „powtarzaj tak długo aż”

Pętle sterowane warunkiem

- **Przykład**

```
s = 0
i = 1
repeat
  s = s + i
  i = i +1
until (i>n)
```

Pętle sterowane warunkiem

- **Przykład**

```
POTEGA5(a,n)
b = 1
k = n
repeat
  b = b*a
  k = k-1
until k==0
return b
```

Pętle sterowane warunkiem

- **Przykład**

```
ODWROC2(n)
repeat
  cyfra = n mod 10
  write cyfra
  n = n div 10
until n==0
return
```

Zagnieżdżanie instrukcji sterujących

- Instrukcje sterujące jednego typu mogą być elementami instrukcji sterujących innego typu
- Jeśli wewnątrz instrukcji sterującej jednego typu występuje instrukcja sterująca innego typu, to mamy do czynienia z **zagnieżdżonymi instrukcjami sterującymi**

Zagnieżdżanie instrukcji sterujących

- **Przykłady**

1) if a == b
then if b == c
then write „Liczby sa rowne”
else write „Liczby nie sa rowne”

2) if x > 0
then write „x jest dodatni”
else if x < 0
then write „x jest ujemny”
else write „x jest rowny 0”

Zagnieżdżanie instrukcji sterujących

- **Przykłady**

3) **for** i = 1 **to** n **do**
    while war **do**
        A

4) **for** i = n **downto** 1 **do**
    for j = m **downto** 1 **do**
        if war
        then A
        else B

„Dobre” i „złe” algorytmy

- W informatyce „szybkie” algorytmy (formalnie: „działające w wielomianowym czasie”) uważa się za „dobre”, natomiast „wolne” algorytmy (formalnie: „działające w wykładniczym czasie”) – za „złe”
- „Dobre” algorytmy działają szybko nawet dla dużych rozmiarów danych wejściowych, „złe” – wręcz przeciwnie
- Nie dla wszystkich problemów znane są „dobre” algorytmy

Przykłady „trudnych” problemów (1/3)

- Dla wielu problemów ważnych praktycznie nie są znane „dobre” algorytmy
- **Problem podziału zbioru**: dany jest zbiór $A=\{a_1,a_2,\ldots,a_n\}$ liczb całkowitych nieujemnych. Czy istnieje podzbiór A’ zbioru A taki, że suma elementów A’ jest równa sumie elementów A-A’?
- Wszystkie znane algorytmy dla problemu podziału zbioru są „wolne” (formalnie: „działają w wykładniczym czasie”)

Przykłady „trudnych” problemów (2/3)

- Innym przykładem problemu „trudnego” jest następujący problem
- **Problem pakowania:** dany jest zbiór $X=\{x_1,x_2,\ldots,x_n\}$, $0 \le x_i \le C$, oraz liczba całkowita dodatnia K. Czy można zapakować elementy zbioru X do co najwyżej K pudełek o pojemności C?
- Nikt nie podał dotąd „szybkiego” (formalnie: „wielomianowego”) algorytmu dla problemu pakowania

Przykłady „trudnych” problemów (3/3)