Algorytmy

Przepis

Warzenie piwa Brunświckiego

DOKŁADNA NAUKA

WARZENIA PIWA.

WEDŁUG METODY ŁATWEY, STWIERDZONEY OSMIOLETNIEM DOSWIADCZENIEM.

DO WYNALAZKÓW NAYNOWSZYCH ZASTOSOWANA,

Z PRZYDANEM OPISANIEM APPARATU DO STUDZENIA, ZASTĘPUIĄCEGO ZWYCZAYNE KILSZTOKI, ZA POMOCĄ KTÓREGO PIWO WRZĄCE, W PRZECIĄGU IEDNEY MINUTY DO TEMPERATURY WODY STUDZIENNEY OCHŁODZONE BYDŹ MOŻE;

PRZEDSTAWIONA W SPOSOBIE PRAKTYCZNYM,

PRZEZ KAROLA WILHELMA SCHMIDT,
AUTORA ROZMAITYCH DZIEŁ TECHNOLOGICZNYCH.

PRZEŁOŻONA Z JĘZYKA NIEMIECKIEGO,
DLA UŻYTKU ZIEMIAN POLSKICH.

WARSZAWA.

NAKŁAD I DRUK N. GLÜCKSBERGA,
KSIĘGARZA I TYPOGRAFA KRÓL: WARSZAW: UNIWERSYT.

1830.

Chambers udziela w swych pismach odmienny sposób robienia Piwa Brunświckiego, odpisany iak twierdzi, z pierwotnéy Recepty, zachowywanéy na Ratuszu mieyskim w Brunświku.

Według Recepty téy wziąść należy 200 miarek zwanych *Kanne* wody, i te tak długo gotować, pokąd 1/3 części iéy nie ubędzie. Sypie się potém do téy wody siedm Szeflów słodu Pszennego i jeden Szefel drobnéy Fasoli; gdy odleie się do beczki dla wyrobienia, nie należy zapełniać iéy całkowicie, bo skoro fermentacyia nastąpi, kładzie się do beczki wewnętrznéy kory z Sośniny funtów trzy, pączków Brzozowych i kotków (pączków) Sosnowych po funcie jednym, trzy garści ziela Cardi-Benedicti, garść jedną lub dwie kwiatu zwanego *Sonnenthaublüthe*, Pimpinelli, Betoniki, Majeranu, ziela zwanego Poley i dzikiego Tymianku, każdego pół garści, albo garść całą, kwiatu Bzowego garści dwie, owocu róży polnéy uncyi 30, utłuczonych.— Po nakładzeniu tych zapraw do beczki, dolewa się ta do pełności, zaczém skutkiem fermentacyi Piwo naciągnie nieco zapachu i smaku, z zapraw przydanych. — Przy ukończeniu fermentacyi wybiją się do beczki 10 jay świeżo zniesionych i ta szpuntuie się. We dwa roki dopiero po zaszpuntowaniu ściąga się Piwo do picia.

Przepis

Warzenie piwa Brunświckiego

- składniki (dane wejściowe): woda, słód, itd.
- wynik: beczka piwa
- sprzęt: beczka, piwowar (mielcarz)
- przepis: oprogramowanie, algorytm
- instrukcje: dodawanie składników, gotowanie, odlewanie, dolewanie, fermentacja, szpuntowanie

Chambers udziela w swych pismach odmienny sposób robienia Piwa Brunświckiego, odpisany iak twierdzi, z pierwotnéy Recepty, zachowywanéy na Ratuszu mieyskim w Brunświku.

Według Recepty téy wziąść należy 200 miarek zwanych *Kanne* wody, i te tak długo gotować, pokąd 1/3 części iéy nie ubędzie. Sypie się potém do téy wody siedm Szeflów slodu Pszennego i jeden Szefel drobnéy Fasoli; gdy odleie się do beczki dla wyrobienia, nie należy zapełniać iéy całkowicie, bo skoro fermentacyia nastąpi, kładzie się do beczki wewnętrznéy kory z Sośniny funtów trzy, pączków Brzozowych i kotków (pączków) Sosnowych po funcie jednym, trzy garści ziela Cārdi-Benedicti, garść jedną lub dwie kwiatu zwanego *Sonnenthaublüthe*, Pimpinelli, Betoniki, Majeranu, ziela zwanego Poley i dzikiego Tymianku, każdego pół garści, albo garść całą, kwiatu Bzowego garści dwie, owocu róży polnéy uncyi 30, utłuczonych.— Po nakładzeniu tych zapraw do beczki, dolewa się ta do pełności, zaczém skutkiem fermentacyi Piwo naciągnie nieco zapachu i smaku, z zapraw przydanych. — Przy ukończeniu fermentacyi wybiją się do beczki 10 jay świeżo zniesionych i ta szpuntuie się. We dwa roki dopiero po zaszpuntowaniu ściąga się Piwo do picia.

Algorytm

jednoznacznie zdefiniowany ciąg operacji prowadzący w skończonej liczbie kroków do rozwiązania zadania.

Algorytm Euklidesa, ok. 300 p.n.e.

algorytm znajdowania największego wspólnego dzielnika (NWD) dwóch liczb całkowitych dodatnich. Uznawany za pierwszy kiedykolwiek wymyślony niebanalny algorytm.

Zadanie obliczeniowe

Algorytmy to rozwiązania pewnych zadań - **zadań algorytmicznych**.

Specyfikacja zadania algorytmicznego

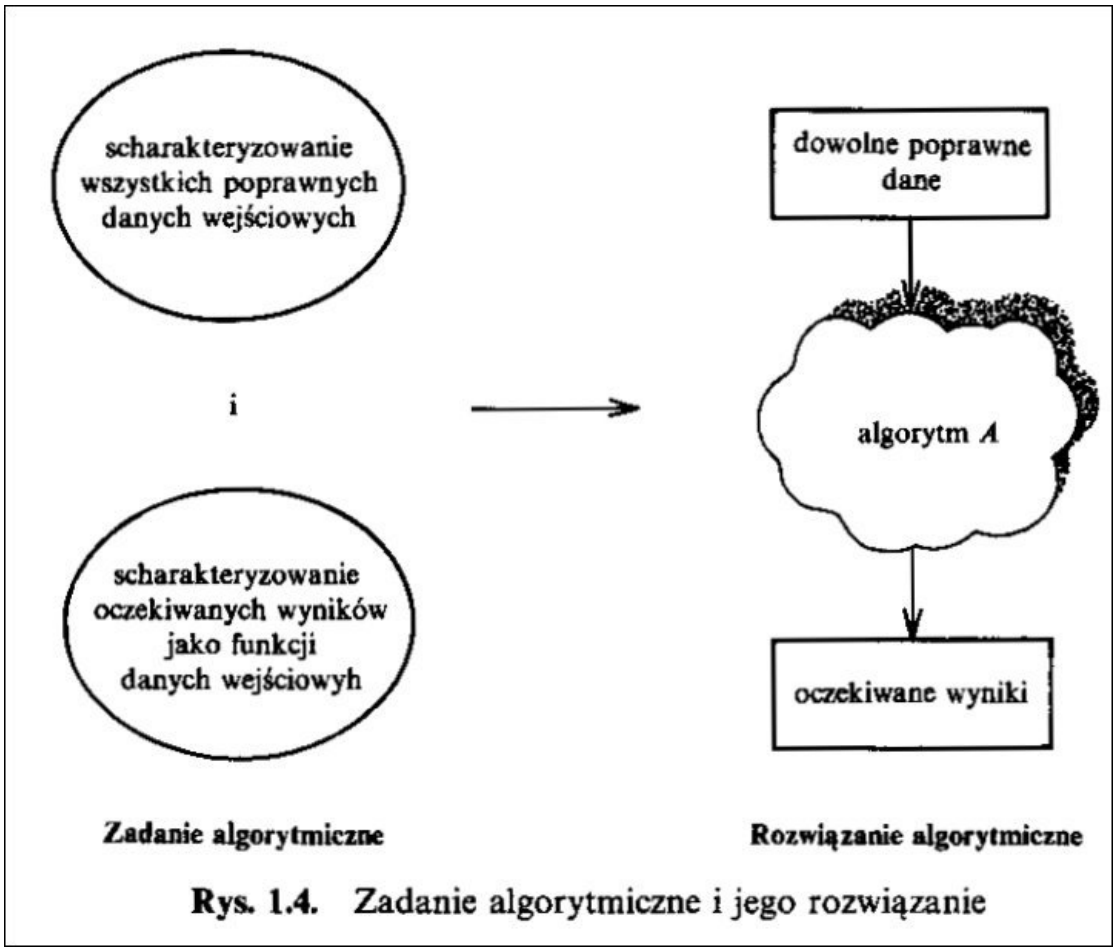
- warunki jakie muszą spełniać dane wejściowe
- określenie oczekiwanych wyników jako funkcji danych wejściowych

Dodatkowe ograniczenia algorytmu, np. dotyczące ilości operacji.

Przykład: Algorytm Euklidesa

Dane wejściowe: liczby całkowite $x, y > 0$
Wynik: $NWD(x, y)$

Zadanie obliczeniowe



Rys. 1.4. Zadanie algorytmiczne i jego rozwiązanie

D. Harel, [2]

Cechy algorytmu

- **skończoność** - ograniczona liczna kroków
- **poprawność** - zgodny ze specyfikacją
- **uniwersalność** - poprawny dla klasy problemów
- **efektywność** - niska złożoność to gwarancja użyteczności
- **określoność** - zrozumiałe polecenia, możliwe do wykonania w jednoznacznej kolejności
- określony stan początkowy i wyróżniony koniec

Struktury sterujące

Ciąg struktur sterujących definiuje kolejność wykonywanych operacji.

- bezpośrednie następstwo: *wykonaj* A, *potem* B
- wybór warunkowy (rozgałęzienie): *jeśli* Q, *to wykonaj* A, *w przeciwnym razie wykonaj* B
- iteracja ograniczona: *wykonaj* A *dokładnie* N *razy*
- iteracja warunkowa: *dopóki* Q, *wykonuj* A
- instrukcja skoku: *skocz do G*
- podprogram - wyodrębniony fragment programu, funkcja

Struktury sterujące można dowolnie składać: np. pętle zagnieżdżone. „Nie ma granic zawiłości algorytmów".

Reprezentacja algorytmu

- Opis językiem naturalnym
- Lista kroków
- Schematy blokowe
- Pseudo-języki
- Języki wysokiego poziomu

Problem:

znalezienie największej liczby całkowitej dzielącej bez reszty liczby całkowite dodatnie a i b

Pomysł:

Zauważmy, że

$$NWD(a,b) = k \implies a = nk, b = mk \implies a - b = (m-n)k$$

Stąd dla $a > b$ zachodzi

$$NWD(a,b) = NWD(a-b,b) = NWD(a-b,a)$$

.

Algorytm Euklidesa

Dane są dwie liczby całkowite. Odejmij od większej liczby mniejszą liczbę a większą liczbę zastąp uzyskaną różnicą. Powtarzaj tą czynność tak długo aż obie liczby będą równe. Otrzymana liczba jest największym wspólnym dzielnikiem liczb wejściowych.

Specyfikacja

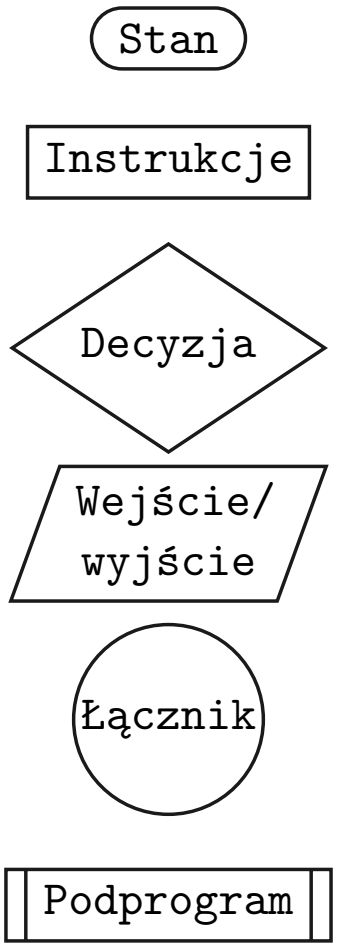
Dane wejściowe: liczby całkowite $a, b > 0$
Wynik: liczba całkowita a stanowiąca największy wspólny dzielnik

Lista kroków

1. jeśli a jest równe b to jest to największy dzielnik
2. jeśli $a > b$ to zastąp a wartością $a - b$ i wróć do punktu 1
3. jeśli $a < b$ to zastąp b wartością $b - a$ i wróć do punktu 1

Algorytm zakłada istnienie operacji $-$, $=$ (porównanie) oraz $>$.

Schematy blokowe



Blok graniczny: start, stop, przerwanie, opóźnienie.

Blok operacyjny: zmiana wartości, postaci lub miejsca zapisu danych.

Blok decyzyjny, rozgałęzienie.

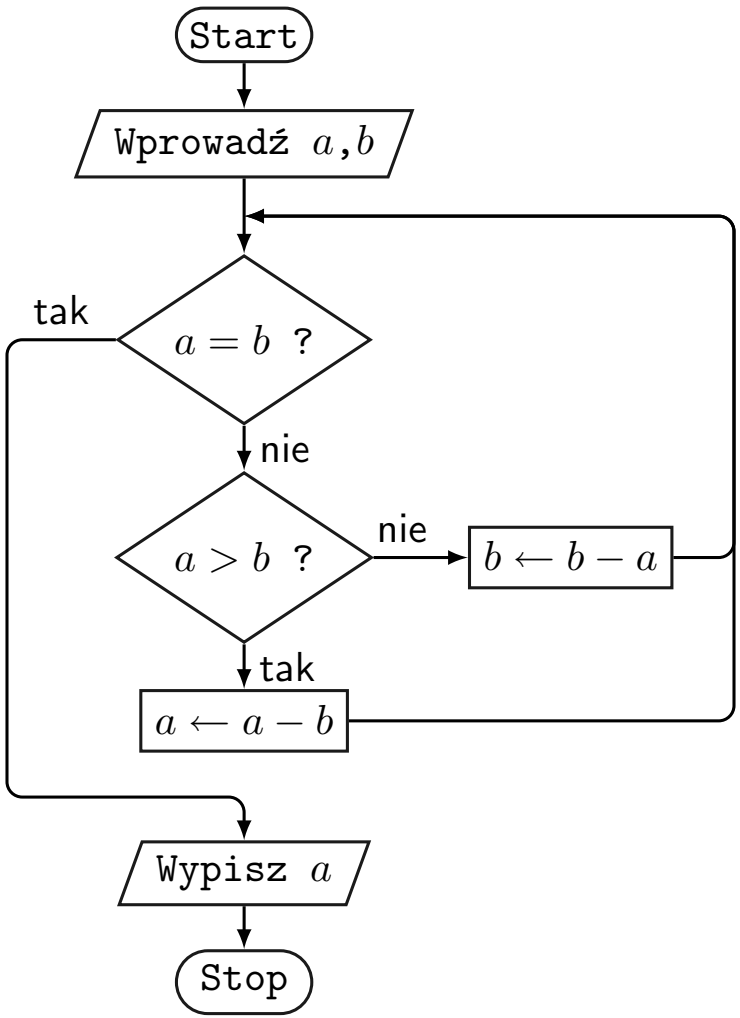
Wprowadzanie danych i wyprowadzenia wyników.

Połączenie z innym fragmentem diagramu.

Wywołanie podprogramu.

Przykład: Schemat blokowy

Algorytm Euklidesa z odejmowaniem



Algorytm 1 Algorytm Euklidesa

dopóki $a \neq b$ **wykonuj**
  jeżeli $a > b$ **wykonaj**
    $a \leftarrow a - b$
  w przeciwnym wypadku
    $b \leftarrow b - a$

- struktura kodu języka wysokiego poziomu (często Pascal)
- uproszczona składnia na rzecz prostoty i czytelności
- formuły matematyczne, język naturalny, podprogramy
- nie zawiera szczegółów implementacji
  „Dla ludzi, nie dla maszyn”.

Program w języku C

```c
/*  Algorytm Euklidesa. */

#include <stdio.h>

int main()
{
    int a, b;

    printf("Podaj dwie liczby calkowite: ");
    scanf("%d %d", &a, &b);

    while (a != b)
        if (a > b) a = a - b;
        else       b = b - a;

    printf("NWD = %d\n", a);

    return 0;
}
```

☞ euclid1.c

Program w języku Pascal

```
1  program NWD(input,output);
2  { Algorytm Euklidesa. }
3  var
4     A, B : Integer;
5  begin
6     Writeln('Podaj dwie liczby calkowite: ');
7     Readln(a,b);
8
9     while a <> b do
10    begin
11       if a > b then a := a - b
12       else b := b - a;
13    end;
14    Writeln('NWD = ', a);
15 end.
```

☞ euclid1.pas

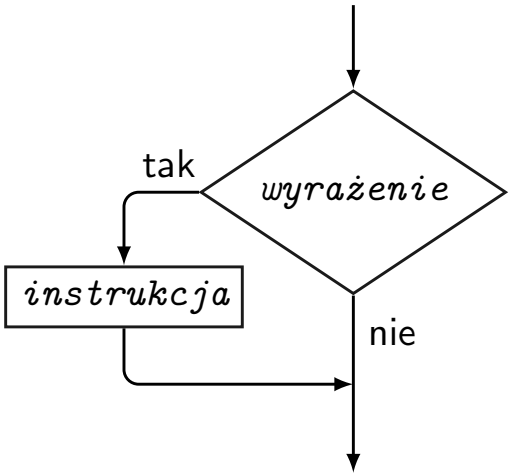
Program w języku Fortran 77

```
1         PROGRAM EUCLID1
2   c     Algorytm Euklidesa w jezyku Fortran 77
3
4         WRITE (*,*) 'Podaj dwie liczby calkowite: '
5         READ (*,*) N, M
6
7         DO WHILE ( N .NE. M )
8            IF ( N .GT. M ) THEN
9               N = N - M
10           ELSE
11              M = M - N
12           ENDIF
13        ENDDO
14        WRITE (*,*) 'NWD=', N
15        END
```

☞ euclid1.f

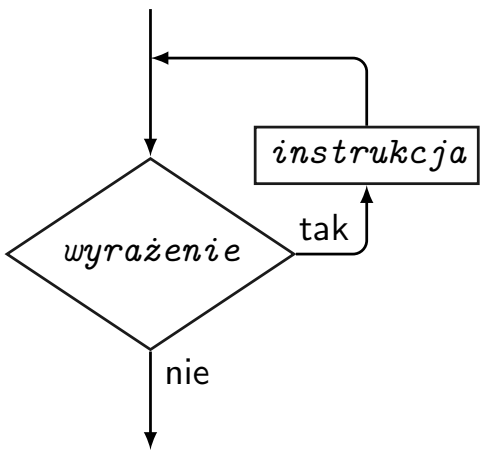
Warunek if (jeżeli)

```
if ( wyrażenie )
    instrukcja
```



Pętla while (dopóki)

```
while ( wyrażenie )
    instrukcja
```



Algorytm Euklidesa cd.

- Czy algorytm jest poprawny? Dla jakich danych?
- Problem stopu.
- Efektywność algorytmu. Ile iteracji należy się spodziewać dla różnych danych?

Algorytm Euklidesa z operacją modulo

Algorytm 2 Algorytm Euklidesa

1: **dopóki** $b \neq 0$ **wykonuj**
2: $\quad c \leftarrow a \mod b$
3: $\quad a \leftarrow b$
4: $\quad b \leftarrow c$

- wymaga operacji dzielenie modulo oraz $\neq$
- złożoność: dla $a > b \geqslant 0$ co najwyżej $2\log_2(a+b)$ iteracji

```c
#include <stdio.h>

int main()
{
    int a, b, c;

    printf("Podaj dwie liczby calkowite: ");
    scanf("%d %d", &a, &b);
    printf("NWD(%d,%d) = ", a, b);

    while (b != 0)
    {
        c = a % b;
        a = b;
        b = c;
    }
    printf("%d\n", a);

    return 0;
}
```

☞ euclid2.c

Esencja programowania proceduralnego

- **Podprogram** - funkcja lub procedura, wydzielona część programu, która może być wielokrotnie użyta
- ekonomiczność - ujednolica powtarzające się bloki programu - mniej kodowania
- przejrzystość, nawet przy złożonych i obszernych algorytmach
- podprogram staje się nową instrukcją elementarną

  w języku C brak wbudowanych funkcji, tylko `main()`
- uproszczenie problemu poprzez rozbicie na mniejsze pod-problemy
  - programowanie zstępujące (*top-down*)
  - programowanie wstępujące (*bottom-up*)
- podprogram uruchamiający sam siebie - rekurencja

```c
int nwd(int a, int b)
{
    int c;
    while (b != 0)
    {
        c = a % b;
        a = b;
        b = c;
    }
    return a;
}
```

☞ euclid3.c

```c
int nwd(int a, int b)
{
    if (b == 0) return  a;
    return nwd(b, a % b );
}
```

☞ euclid3rekurencja.c

Algorytm 3 Sortowanie przez wybieranie (selection sort)

Dane wejściowe: ciąg n liczb $A = \{a_1, a_2, \ldots, a_n\}$
Wynik: uporządkowany ciąg $a_1 \leqslant a_2 \leqslant, \ldots, \leqslant a_n$

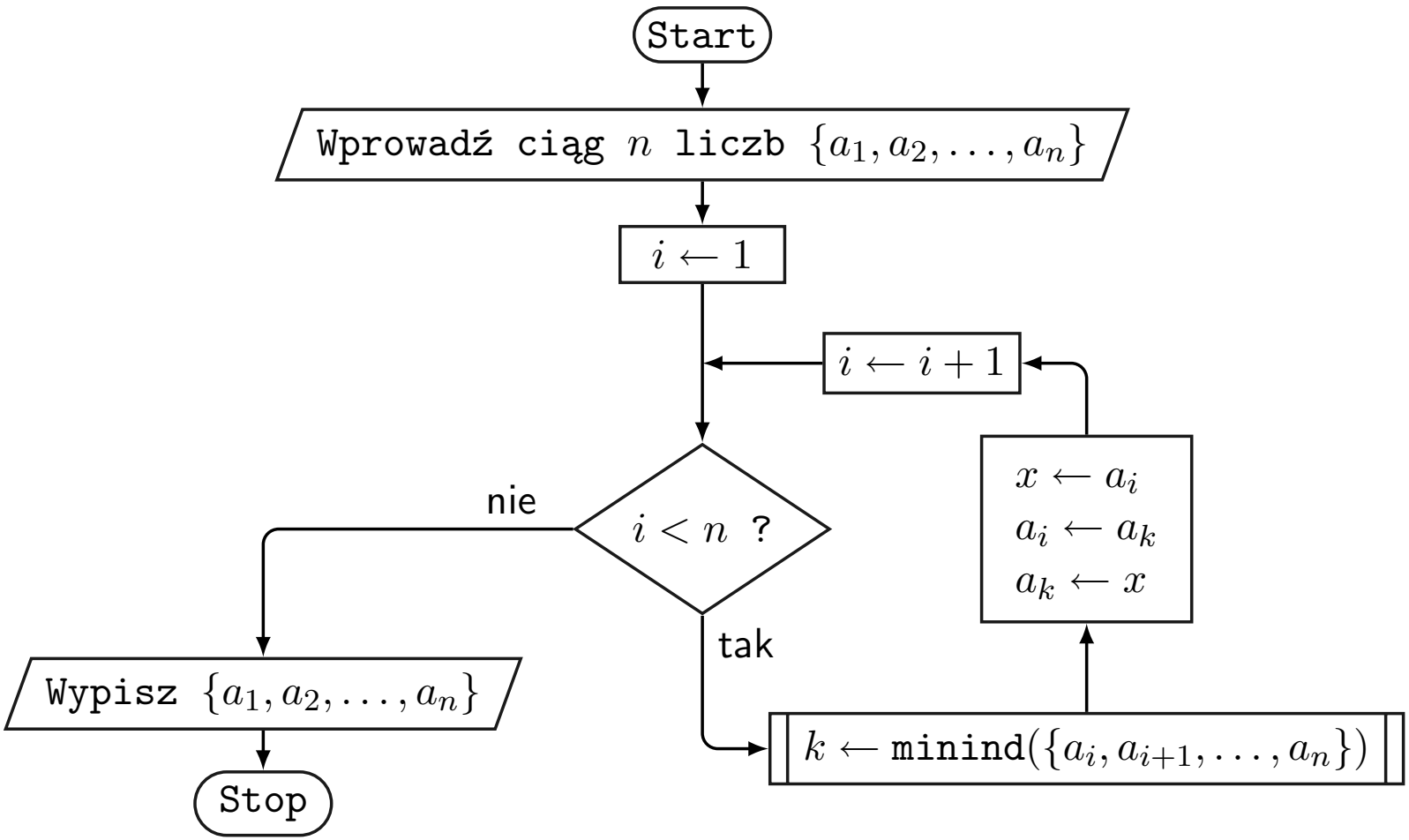
1: $i \leftarrow 1$
2: **dopóki** $i < n$ **wykonuj**
3: $\quad k \leftarrow \texttt{minind}(\{a_i, a_{i+1}, \ldots, a_n\})$ ▷ Podprogram
4: $\quad a_i \longleftrightarrow a_k$
5: $\quad i \leftarrow i + 1$

`minind()` w podanym ciągu wyszukuje indeks elementu o najmniejszej wartości.

Wizualizacja algorytmów sortowania: ☞ AlgoRythmics

Przykład: Schemat blokowy

Sortowanie przez wybieranie



Złożoność obliczeniowa algorytmu

Złożoność obliczeniowa

liczba operacji wykonywanych przez algorytm. Zazwyczaj wyznaczana względem ilości danych lub ich rozmiaru.

Ile operacji wymaga ...

- porównanie dwóch liczb całkowitych?
- obliczenie $NWD(a, b)$?
- znalezienie pewnego elementu wśród N elementów?
- ile operacji wymaga posortowanie listy N elementów?
- Problem komiwojażera: znalezienie najkrótszej drogi łączącej wszystkie miasta?

Problem TSP

Travelling salesman problem

Rysunek 1.1. Najkrótsza trasa premiera przebiegająca przez wszystkie miasta wojewódzkie w podziale administracyjnym z 1975 roku (A. Adrabiński)

Tabela 1.1. Czasy znalezienia przez komputer, wykonujący 100 miliardów operacji na sekundę, najkrótszej trasy podróży premiera (zob. ćwiczenie 1.3)

Liczba województw	Czas znajdowania najkrótszej trasy
17	3.5 minuty
25	$2 \cdot 10^5$ lat
49	$4 \cdot 10^{42}$ lat

[1] M.M. Sysło, „Algorytmy”

Problemy o „rozsądnych” rozwiązaniach

Problemy P i NP

Funkcja \ N	10	50	100	300	1000
$5N$	50	250	500	1500	5000
$N \times \log_2 N$	33	282	665	2469	9966
N^2	100	2500	10 000	90 000	1 milion (7 cyfr)
N^3	1000	125 000	1 milion (7 cyfr)	27 milionów (8 cyfr)	1 miliard (10 cyfr)
2^N	1024	liczba 16-cyfrowa	liczba 31-cyfrowa	liczba 91-cyfrowa	liczba 302-cyfrowa
$N!$	3,6 miliona (7 cyfr)	liczba 65-cyfrowa	liczba 161-cyfrowa	liczba 623-cyfrowa	niewyobrażalnie duża
N^N	10 miliardów (11 cyfr)	liczba 85-cyfrowa	liczba 201-cyfrowa	liczba 744-cyfrowa	niewyobrażalnie duża

Dla porównania: liczba protonów w znanym wszechświecie ma 126 cyfr,
liczba mikrosekund od „wielkiego wybuchu” ma 24 cyfry.

Rys. 7.3. Niektóre wartości pewnych funkcji

[2] D. Harel, Rzecz o istocie informatyki. Algorytmika.

Problemy o „rozsądnych” rozwiązaniach

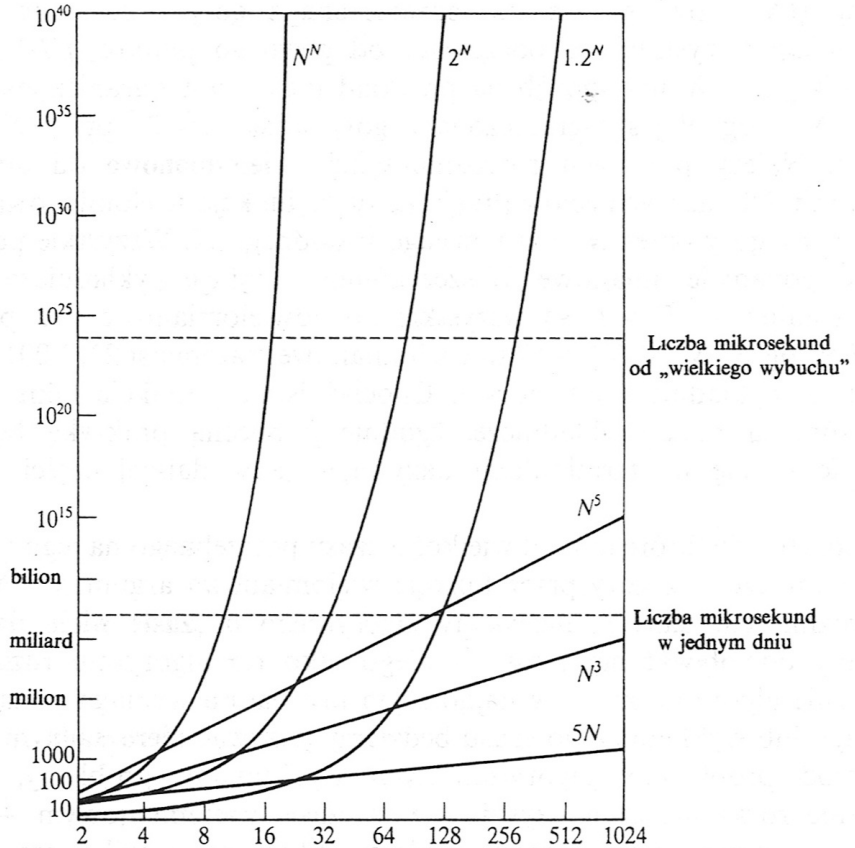
Problemy P i NP

- Rozsądne rozwiązania, wykonywalne w czasie wielomianowym

  $\log N,\ N, N \log N, N^7 + N^3 + 2$

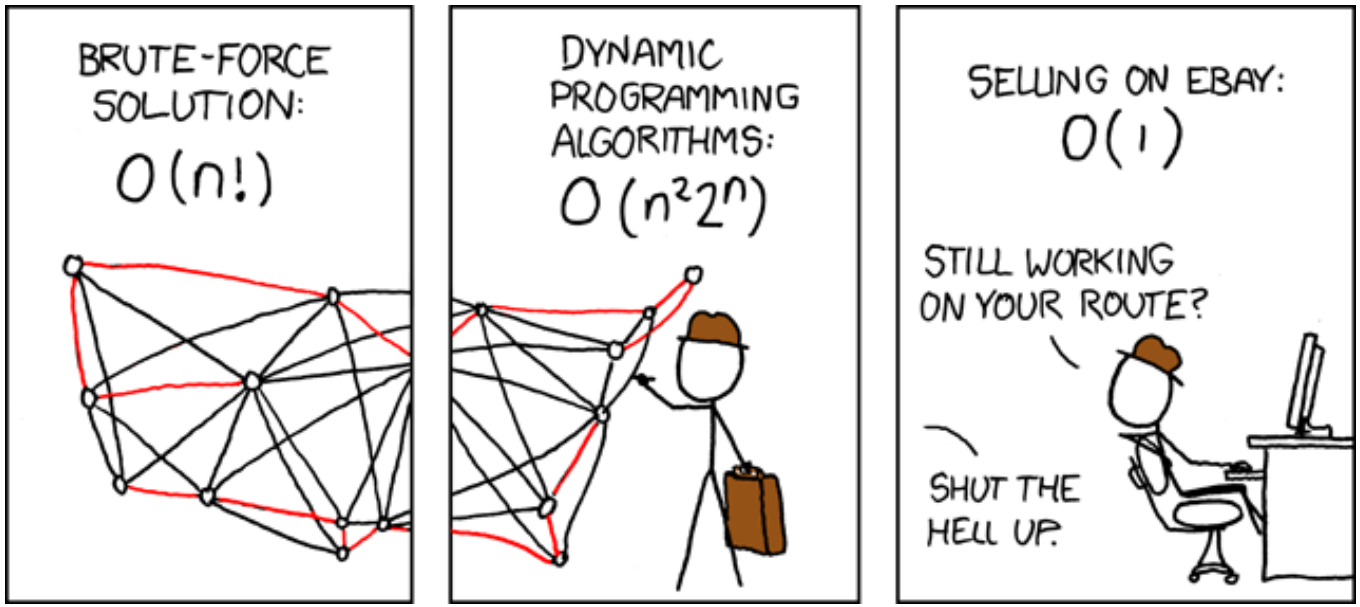
- Nierozsądne, niepraktyczne, czas ponad-wielomianowy

  $2^N, 1.001^N + N^7, N^N, N!$



Rys. 7.4. Tempo wzrostu pewnych funkcji

[2] D. Harel, Rzecz o istocie informatyki. Algorytmika.



http://xkcd.com

Algorytmy manipulują danymi

dane wejściowe ⇒ Algorytm ⇒ Dane wyjściowe

Reprezentacja danych kształtuje algorytm!

- zmienne: liczby, znaki, `a = 3`
- tablice, wektory, listy, `a[3]=3`
- tablice wielowymiarowe, tablice tablic, `a[3,3]=3` `a[3][3]=3`
- rekordy, struktury, `a.b=3` `a.c='a'`
- kolejki, stosy, drzewa
- pliki, bazy danych

Złożoność pamięciowa algorytmu - miara ilości wykorzystanej pamięci

Od problemu do rozwiązania

- analiza problemu
- specyfikacja zadania:
  - dopuszczalny zestaw danych wejściowych
  - pożądane wyniki jako funkcja danych wejściowych
- algorytm - rozwiązanie zadania
- poprawność algorytmu:
  - względem specyfikacji
  - problem stopu
  - efektywność (złożoność): „*Każda akcja zajmuje czas!*”
- implementacja algorytmu $\Rightarrow$ program

Literatura dodatkowa

- Maciej M. Sysło, *„Algorytmy”*, WSiP, Warszawa, 2002.
- D. Harel, *Rzecz o istocie informatyki. Algorytmika.*, WNT, Warszawa, 1992.
- Fulmański Piotr, Sobieski Ścibór, *„Wstęp do informatyki”*, Uniwersytet Łódzki, 2004