Wykład I

- **Pojęcie algorytmu,**
- **Schematy blokowe,**
- **Etapy rozwiązywania zadań,**
- **Algorytm NWD,**
- **Złożoność obliczeniowa.**

Pojęcia pierwotne:

Akcja → głównym pojęciem występującym wszędzie jest pojęcie akcji. W programowaniu zakładamy, że akcja jest pewnym zdarzeniem o skończonym czasie trwania i o zamierzonym dobrze określonym skutku. Skutek będzie nazywany efektem (akcji).

Obiekt → Każda akcja musi być wykonywana na pewnym obiekcie, wynik działania akcji rozpoznawany jest po zmianach stanu tego obiektu.

Instrukcja → Jeżeli chcemy opisać akcję w jakimś języku formalnym to używamy instrukcji.

Proces, Obliczenie → Składa się z kilku części składowych (akcji). W procesie poszczególne akcje najczęściej muszą następować w ściśle określonej kolejności.

Program → Instrukcja lub ciąg instrukcji opisujący proces będzie nazywany programem.

Algorytm → Dokładny schemat postępowania prowadzący do rozwiązania określonego zadania. Pochodzi od przekształconego arabskiego przydomka Alchwarizini.

- Algorytm opracowuje się w celu rozwiązywania problemów, dla których metoda z algorytmu jest powtarzalna.

- Algorytm projektuje się dla zadań, dla których istnieje rozwiązanie. Gdy nie wiadomo czy istnieje rozwiązania, należy określić moment przerwania wykonywania algorytmu.

- Dla danego problemu może być wiele algorytmów, które je rozwiązują.

Wybrane zdarzenia z rozwoju algorytmów:

- Algorytm Euklidesa – 400-300 p.n.e (wyznaczanie NWD dla dwóch liczb, np. NWD(33,22) = 11,

- Muhammed Alchwarizini (IX w.) – matematyk, autor reguł podstawowych operacji arytmetycznych dla liczb dziesiętnych (łac Algorismus). Od jego nazwiska pochodzi nazwa algorytm,

- Charles Babbage (1833) – matematyk, wynalazca maszyny różnicowej i autor projektu „maszyny analitycznej" sterowanej algorytmami kodowanymi na dziurkowanych kartach,

- Herman Hollerith (1890) – wynalazca maszyny wspomagającej spis powszechny w USA, skrócił czas z blisko ½ roku do kilku dni,

- Alan Turing, John von Neumann i inni – pierwsze komputery elektroniczne,

- Połowa lat 60 – Informatyka staje się dyscypliną akademicką.

Procesor, komputer → Algorytm może być wykonywany zarówno przez człowieka (jest to raczej praca nudna i niewdzięczna) jak również przez automaty/maszyny.

Pamięć → Obiekty muszą być gdzieś umieszczane. W tym celu powstała pamięć.

Języki programowania → Samo ułożenie algorytmu jest niewystarczające. Trzeba go zaimplementować.

Algorytm układa człowiek natomiast wykonuje komputer/maszyna (najczęściej).

Zatem algorytm musi być zapisany za pomocą języka programowania.

Język programowania: składnia i semantyka.

Etapy rozwiązywania zadań

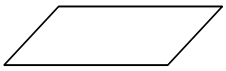
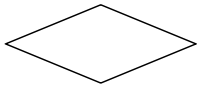
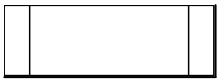
1. Sformułowanie zadania z wyróżnieniem informacji wejściowych i wyjściowych.

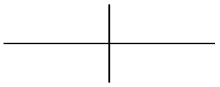
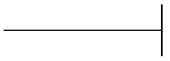
2. Opracowanie algorytmu. Algorytm jest często przedstawiany za pomocą graficznej reprezentacji z wykorzystaniem odpowiednio zdefiniowanych symboli graficznych, nazywamy to schematem blokowym lub siecią działań.

3. Opracowanie kodu programu według zasad i składni zadanej przez wybrany język programowania.

4. Sprawdzenie poprawności działania programu, tzn. skompilowanie programu i przetestowanie. W przypadku niewłaściwych wyników należy powtórzyć wszystkie etapy w celu znalezienia błędu.

Tablica 1. Symbole stosowane w schematach blokowych

Nazwa operacji	Wyjaśnienie	Symbol
Przetwarzanie	Operacja lub grupa operacji, w wyniku których ulega zmianie wartość, postać lub miejsce zapisu danych	
Wprowadzanie lub wyprowadzanie	Wprowadzanie lub wyprowadzanie	
Decyzja	Operacja określająca wybór jednej z alternatywnych dróg działania	
Proces zdefiniowany (podprogram)	Proces (ciąg instrukcji) zdefiniowany poza programem	
Początek lub koniec	Oznaczenie miejsca rozpoczęcia lub zakończenia działania schematu blokowego	
Droga przepływu danych	Więź operacyjna między poszczególnymi operacjami procesu przetwarzania	

Droga przepływu danych o wskazanym kierunku	**Więź operacyjna między poszczególnymi operacjami procesu przetwarzania, gdy ten kierunek nie jest jednoznacznie określony**	
Skrzyżowanie dróg przepływu danych bez powiązania	**Przecięcie więzi operacyjnych nie związanych ze sobą**	
Łączenie dróg	**Łączenie dróg przepływu danych**	
Łącznik stronicowy	**Wejście lub wyjście z wyodrębnionych fragmentów schematu znajdujących się na różnych stronach**	
Komentarz	**Oznaczenie miejsca na komentarz**	

Przykład algorytmizacji zadania (metoda intuicyjna - nieformalna) – przykład literaturowy

Zadanie: Dane są liczby całkowite a i b. Znajdź ich największy wspólny podzielnik.

Problem I (trywialny):

Co jest dane?
Dane są dwie liczby.
W razie potrzeby możemy je oznaczyć jako a i b.

Problem II (trochę trudniejszy):

Co to jest największy wspólny podzielnik?
Jest to największa liczba całkowita, przez którą dzielą się bez reszty obydwie liczby.
NWD zawsze istnieje.

Problem III (najtrudniejszy):

Jak obliczyć NWD?

NWD(-a,b) = NWD(a,b), dla a,b>0

NWD(a,0) = a,

NWD(a,b) = NWD(b,a).

Gdyby udało się nam zmniejszać liczby dla których szukamy NWD tak, by nie zmieniać NWD, to w końcu doprowadzilibyśmy do sytuacji, że szukalibyśmy NWD(a,0) lub NWD(0,b).

Intuicyjny algorytm można zapisać:

(1) Pod x podstaw a, pod y podstaw b,

(2) Jeśli y < > 0 to

(a) zmniejsz y i zmień x w ten sposób, by obie liczby pozostały >= 0 i by wartość NWD(x,y) nie zmieniła się,
(b) powtórz krok (2),

(3) jeśli y = 0, to NWD(x,0) = x.

(*) x = (x div y)*y + x mod y,

x div y oznacza wynik dzielenia całkowitego,
x mod y oznacza resztę z dzielenia (x >= y).

dla x=30 y=7 mamy
30 = (30 div 7)*7+30 mod 7=4*7+2=30

dla x=7 i y=30 mamy
7=(7 div 30)*30+7 mod 30=0*30+7=7.

Zatem
(**) x mod y = x - (x div y)*y.

Jeśli NWD dzieli x i y to NWD dzieli również
x - (x div y)*y, bo
(x-(x div y)*y)/NWD=x/NWD-(x div y)*(y/NWD)
jest liczbą całkowitą.

Zachodzi więc (na podstawie **):

NWD(x,y) = NWD(y,x mod y)

Otrzymaliśmy sposób na wykonanie kroku (a).

(a1) Pod r podstaw x mod y,
(a2) Pod x podstaw y,
(a3) Pod y podstaw r.

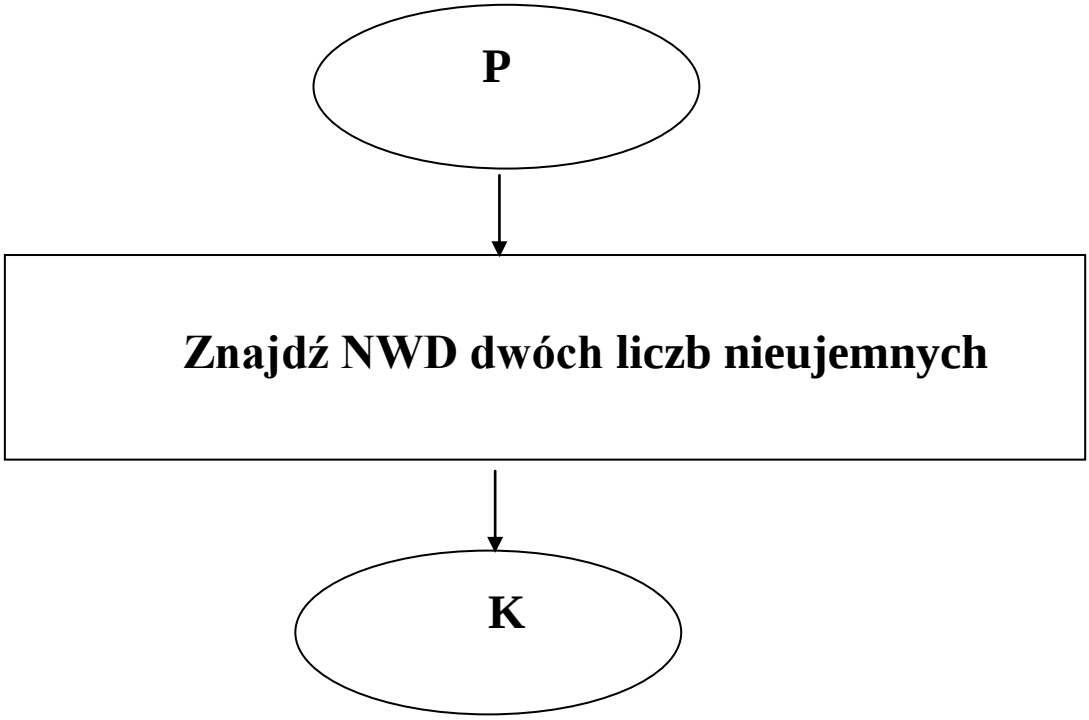
Jak obliczyć x mod y?

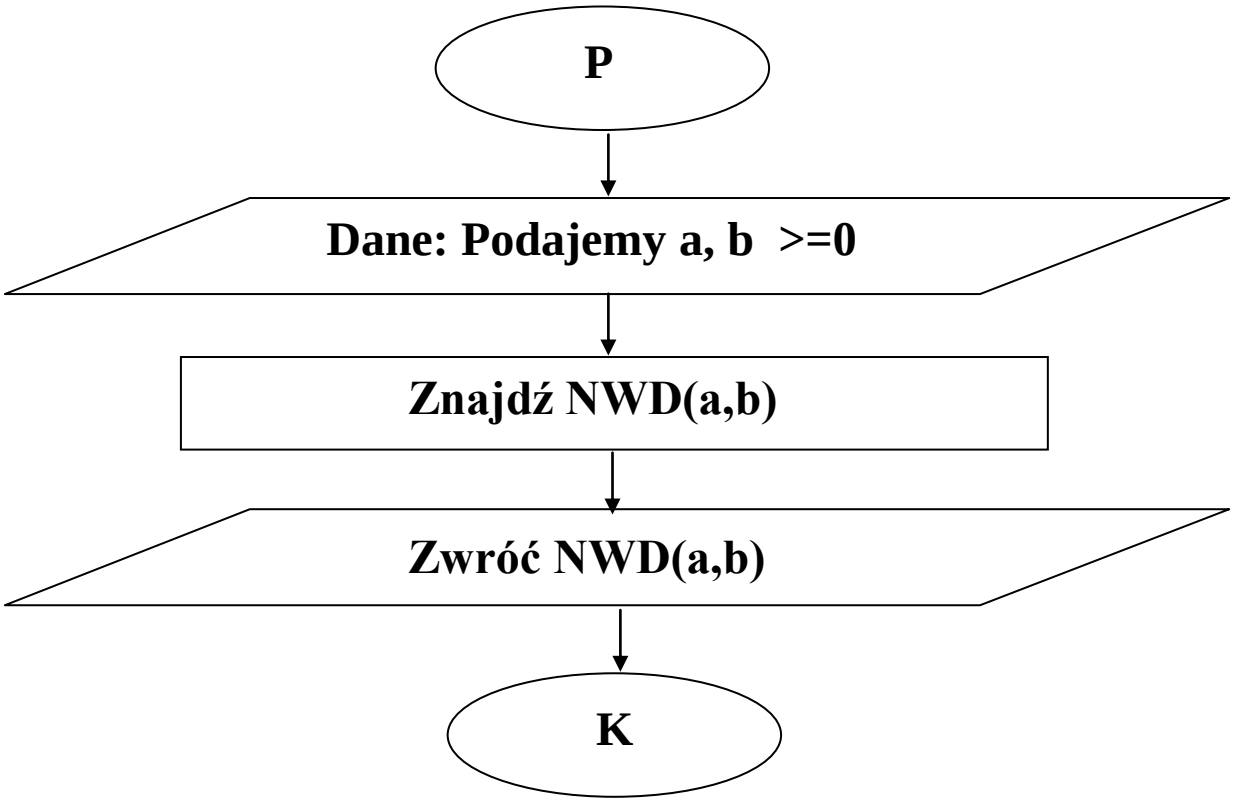
Najlepiej zrobić to zauważając, że x= q*y+r, gdzie 0 <= r <y. Stąd mamy, że r = x - q*y.

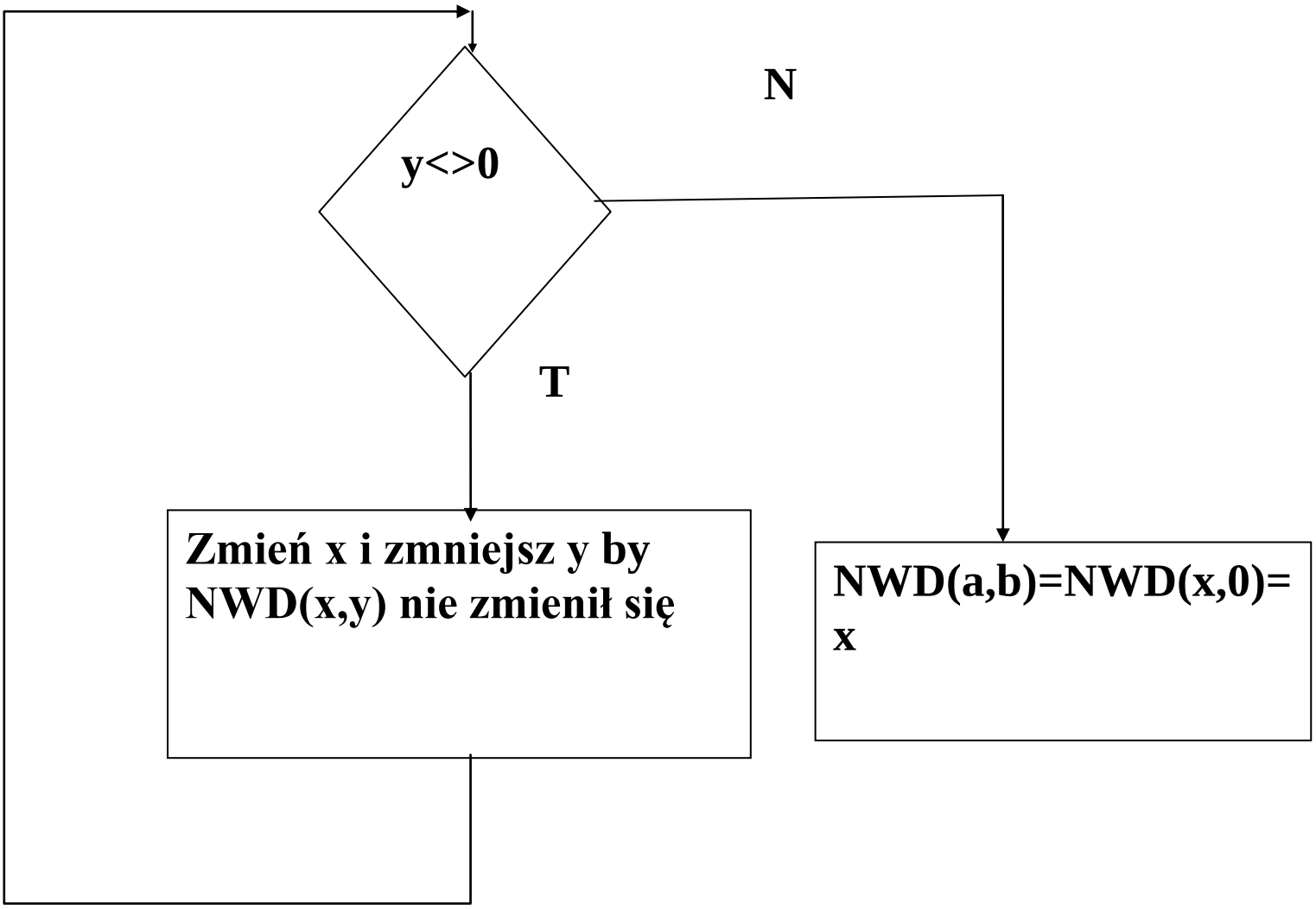
Należy od x odjąć y i sprawdzić, czy wynik r spełnia warunek 0 <= r <y, jeśli tak, to mamy resztę, jeśli nie to powtarzamy odejmowanie.

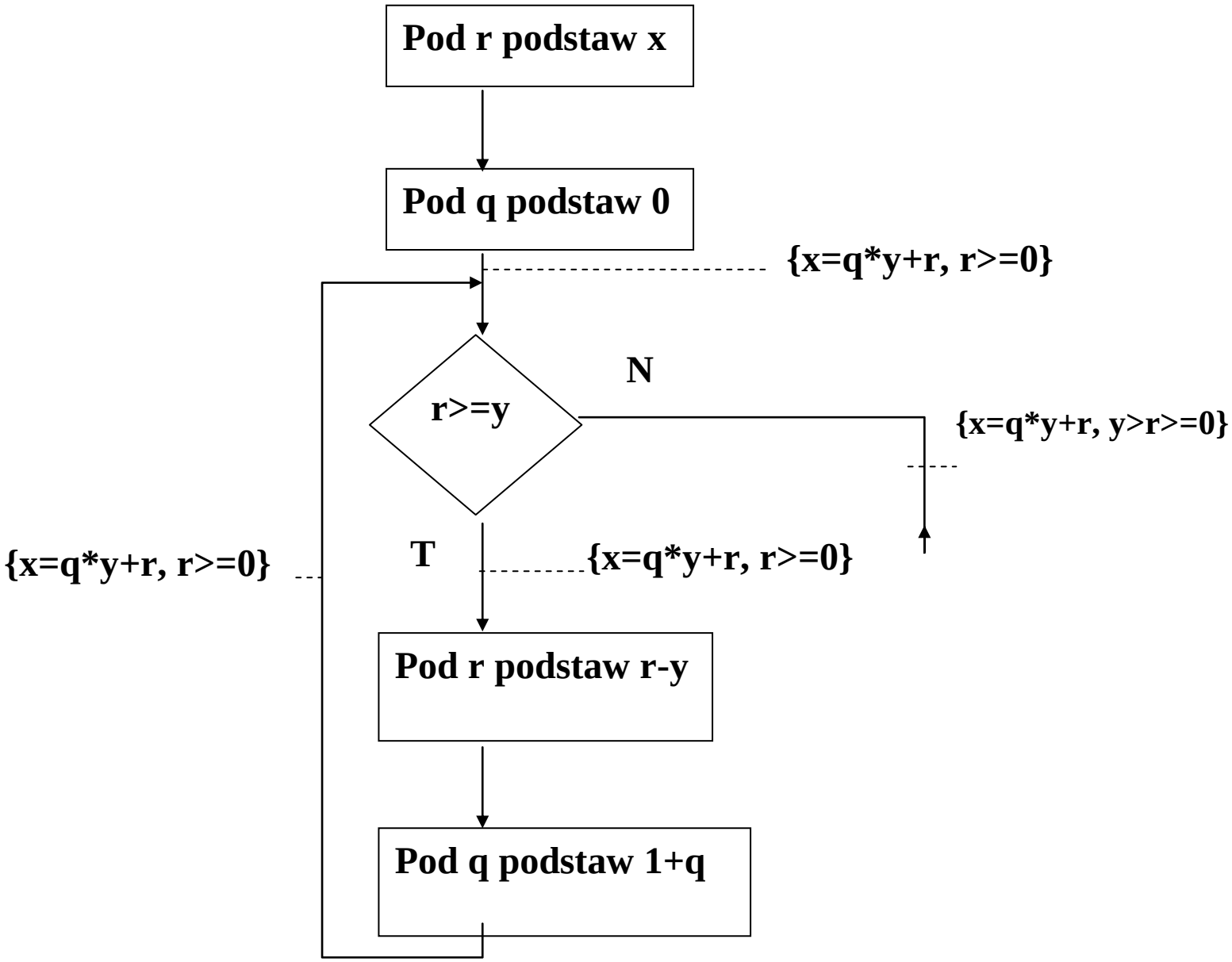
Krok (a1) można rozpisać jako:

(a11) Pod q podstaw 0,
(a12) Pod r podstaw x,
(a13) Sprawdź, czy 0 <= r <y
(a14) jeśli tak, to została znaleziona reszta,
(a15) jeśli nie, to
    (a151) Pod r podstaw r - y,
    (a152) Pod q podstaw 1+q
    (a153) Wykonaj algorytm od kroku (a13).


P
Znajdź NWD dwóch liczb nieujemnych
K


P
Dane: Podajemy a, b >=0
Znajdź NWD(a,b)
Zwróć NWD(a,b)
K


y<>0
N
T
Zmień x i zmniejsz y by NWD(x,y) nie zmienił się
NWD(a,b)=NWD(x,0)= x


Pod r podstaw x
Pod q podstaw 0
{x=q*y+r, r>=0}
r>=y
N
{x=q*y+r, y>r>=0}
T
{x=q*y+r, r>=0}
{x=q*y+r, r>=0}
Pod r podstaw r-y
Pod q podstaw 1+q

```
begin {x>=0, y>0}
q:=0; r:=x;
      while r >= y do {x=q*y+r, r>=y}
      begin
            r:=r-y;
            q:=1+q
      end;
      {x=q*y+r, 0<=r<y}
end
```

```
10 let q=0
20 let r=x
30 if r<y goto 100
40 let r=r-y
50 let q=1+q
60 goto 30
100 Rem DALEJ
```

- FORTRAN
  - Do obliczeń naukowo-technicznych,
  - Opracowany w 1957r,
  - Popularny w zastosowaniach numerycznych,
  - Wykorzystywana instrukcja skoku goto, brak rekurencji.

COBOL
  - Popularny w środowiskach biznesu,
  - Powstał w 1959 r.,
  - Zawiera mechanizmy definiowania struktury pliku z danymi (sekcja data division),
- PL/I
  - Zaproponowany przez użytkowników IBM i sponsorowany przez IBM,
  - Przeznaczony do obliczeń inżynierskich, do zastosowań w biznesie, zawiera dużo instrukcji, rozbudowane struktury danych, b. złożony.
- C (C++)
  - Opracowany w 1972 r.,
  - Język wysokiego poziomu, zawiera instrukcje charakterystyczne dla języków niskiego poziomu,
  - Pozwala tworzyć programy łatwe do przenoszenia na inne platformy sprzętowe,
  - Opracowuje się w nim systemy operacyjne, np. UNIX,

- LISP
  - Opracowany w latach 50 i 60,
  - Oparty na listach i rekurencji,
  - Zwykle interpretowany (dużo pamięci),
  - Sprawdza się w obliczeniach symbolicznych (gry komputerowe, przetwarzanie języka naturalnego, podejmowanie decyzji),
- Prolog
  - Powstał w latach 70,
  - Opiera się na klauzulach, stwierdzających, że pewne fakty logicznie wynikają z innych faktów,
  - Wykorzystuje rekurencję,
  - Program jest zbiorem faktów, klauzul i dyrektyw polecających sprawdzenie prawdziwości pewnego faktu,
- JAVA
  - Powstała w 1995 r.,
  - Obiektowy, strukturalny, imperatywny,
  - Kompilowany do kodu bajtowego, czyli postaci wykonywalnej przez maszynę wirtualną, później interpretowany
  - Silnie typowany,
  - Brak wielo-dziedziczenia.

Aktualnie: Java Script**,** PHP, Scala, SQL, Python, Swift, C#, ….

Obszary badań nad językami programowania:

- Podstawy matematyczne interpretacji i kompilacji,
- Definiowanie semantyki metodą operacyjną i denotacyjną,
- Tworzenie języków zapytań i manipulowania danymi,
- Opracowywanie środowisk programowania,
- Wizualne języki programowania,
- Rozbudowa struktur sterujących,
- Definiowanie własnych struktur.

Metoda **TOP-DOWN**

W rozwiązywaniu problemów najczęściej wykorzystywana jest metoda zstępująca (czasem spotyka się określenie zastępująca), ang. Top-down. Polega ona tym, by

Rozłożyć cały problem na ściśle określone podproblemy i udowodnić, że jeśli każdy podproblem jest rozwiązany poprawnie, a te rozwiązania są połączone w określony sposób, to pierwotny problem też jest rozwiązany poprawnie. Proces dzielenia na podproblemy należy prowadzić tak długo, aż ich rozwiązanie można zapisać za pomocą kilku wierszy w wybranym języku programowania.

Każdy algorytm ma strukturę statyczną i strukturę dynamiczną.

Strukturę statyczną reprezentuje tekst programu (w postaci schematu blokowego lub programu w języku np. Pascal).

Strukturą dynamiczną jest proces obliczeń.

Poważnym problemem jest, by udowodnić formalnie, że instrukcje zmieniają stany obliczeń w określony dający się badać sposób. W tym celu należy określić instrukcje, jakie mogą wystąpić w języku programowania i podać dowody ich poprawności.

W przypadku badania poprawności algorytmu należy dążyć do usunięcia błędów (poprawność programu niewątpliwie tego wymaga) a następnie udowodnić formalnie poprawność samego algorytmu.

Ogólne uwagi dotyczące poprawności programu

- **Błędy językowe** – wynikają ze złego wykorzystania składni języka lub jej nieznajomości. Skutkiem ich jest zatrzymanie kompilacji lub interpretacji (może, ale nie musi być wyświetlany komunikat), są to łatwe do usunięcia błędy,
- **Błędy semantyczne** – wynikają z nieznajomości semantyki języka programowania. Skutkiem jest to, że program nie realizuje prawidłowo algorytmu, trudne do wyszukania i przewidzenia, możliwe do usunięcia przy odpowiedniej trosce o używane instrukcje,
- **Błędy logiczne** – wynikają z niewłaściwego rozwiązania zadania, w pewnych sytuacjach programy mogą działać poprawnie, trudne do usunięcia i dlatego są b. groźne, mogą długo pozostawać w ukryciu,
- **Błędy algorytmiczne** – występują wtedy, gdy algorytm dla pewnych dopuszczalnych danych daje niepoprawny wynik, program nie kończy swojego działania.

Aby tych błędów uniknąć należy przeprowadzić testowanie programu (algorytmu).

Algorytm abstrakcyjny

W momencie rozwiązywania złożonego problemu należy skupić się na wyrażeniu rozwiązania w języku naturalnym. Takie rozwiązanie będzie nazywane algorytmem abstrakcyjnym. Wyraża on tylko ogólną strategię rozwiązania problemu wraz z ogólną strukturą rozwiązania, które chcemy otrzymać.

Metoda zstępująca programowania

Budowanie programu polega na określaniu kolejnych uściśleń tak, aby w kolejnych krokach instrukcje i dane abstrakcyjne wyrażać w kategoriach wybranego języka programowania.

Sprawdzanie poprawności

Ważną cechą stopniowego uściślania programu jest to, że równolegle można sprawdzać poprawność budowanego programu. Należy dowodzić poprawności każdej kolejnej wersji programu (oczywiście na pewnym poziomie abstrakcji).

Ponieważ programy abstrakcyjne są znacznie prostsze i krótsze, więc dowody poprawności są również krótkie i mniej uciążliwe. W kolejnym etapie stosujemy to samo podejście do poszczególnych instrukcji abstrakcyjnych kończąc w chwili, gdy każdą instrukcję zapiszemy w wybranym języku programowania.

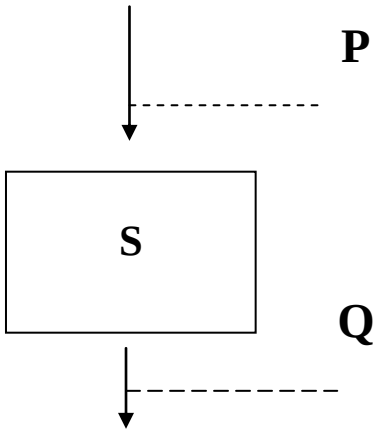
Formalne dowodzenie poprawności

Ogólny schemat instrukcji (programu) jest postaci:

{P} S {Q}

Specyfikacja programu:

Jeśli relacja P zachodzi przed wykonaniem programu S to po jego wykonaniu zachodzi relacja Q.



{P} S {Q}

Definicja częściowej poprawności programu względem specyfikacji:

Jeśli spełniona jest relacja P i program S się zakończy, to będzie spełniona relacja Q.

Definicja pełnej poprawności programu względem specyfikacji:

Jeśli spełniona jest relacja P, to program S zakończy się i będzie spełniona relacja Q.

Podejście do programowania można sformułować następująco:

- projektowanie powinno rozpocząć się od pełnej specyfikacji $\{P\}S\{Q\}$, którą program powinien spełniać. Oznacza to, że należy podać, dla jakich danych możemy program stosować oraz jakie wyniki powinny być otrzymane (jest to sformułowanie zadania).

- Proces projektowania zstępującego pozwala dzielić problem na specyfikacje $\{P_i\}S_i\{Q_i\}$ fragmentów pod-programów S_i, z których program jest zbudowany.

- Projektowaniu towarzyszy dowodzenie poprawności wszystkich tych specyfikacji, przy czym przyjmujemy, że $\{P_{i+1}\}=\{Q_i\}$.

TYPY DANYCH

1. Typ danych pozwala osiągnąć logiczną jasność, wskazując czym jest dana zmienna (np. liczbą rzeczywistą) i jakie operacje można na niej wykonywać.

2. Podczas wykonywania programu bieżąca wartość zmiennej może być zapamiętywana w kilku komórkach. Typ danych pozwala translatorowi (kompilatorowi, interpretatorowi) na zarezerwowanie niezbędnej liczby komórek z przeznaczeniem na przechowywanie wartości zmiennych oraz określenie jakie procedury mogą służyć do kodowania lub dekodowania wartości.

3. Każdy język programowania ma swój repertuar podstawowych instrukcji testujących wartości pewnych zmiennych lub przekształcających wartości zmiennych, aby otrzymać nowe wartości. Wymaga to jasnej specyfikacji typu danych dla zmiennych uczestniczących w tych operacjach.

Złożoność obliczeniowa

To jeden z najważniejszych parametrów charakteryzujących algorytm.

Decyduje o efektywności całego programu.

Podstawowymi zasobami systemowymi uwzględnianymi w analizie algorytmów są czas działania oraz obszar zajmowanej pamięci.

Na złożoność czasową składają się dwie wartości: **pesymistyczna**, czyli taka, która charakteryzuje najgorszy przypadek działania oraz **oczekiwana**.

Najczęściej algorytmy mają złożoność czasową proporcjonalną do funkcji:

- log(n)- złożoność logarytmiczna
- n - złożoność liniowa
- nlog(n) - złożoność liniowo-logarytmiczna
- n^2 - złożoność kwadratowa
- n^k - złożoność wielomianowa
- 2^n - złożoność wykładnicza
- n! - złożoność wykładnicza,

  ponieważ $n!>2^n$ już od n=4

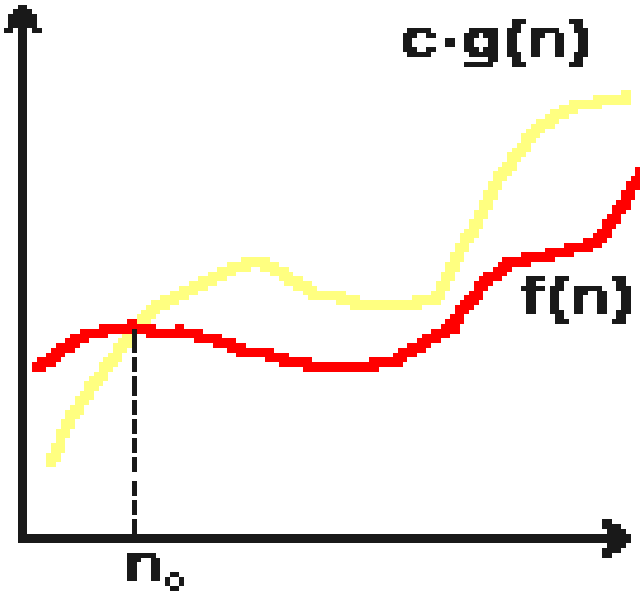
Rząd wielkości

służy do opisu czasu działania algorytmu.

Istnieją 3 notacje służące do tego celu.

Notacja **O (omikron)** - ograniczenie funkcji od góry.

Gdy mówimy, że pewna f(n) funkcja jest rzędu g(n), co zapisujemy f(n)=O(g(n)), to znaczy, że istnieje taki argument n_0, którego począwszy dla każdego niemniejszego od n_0 wartości funkcji f(n) są niewiększe od wartości funkcji g(n) z dokładnością do stałej c.



Jest to asymptotyczna granica górna. Służy do

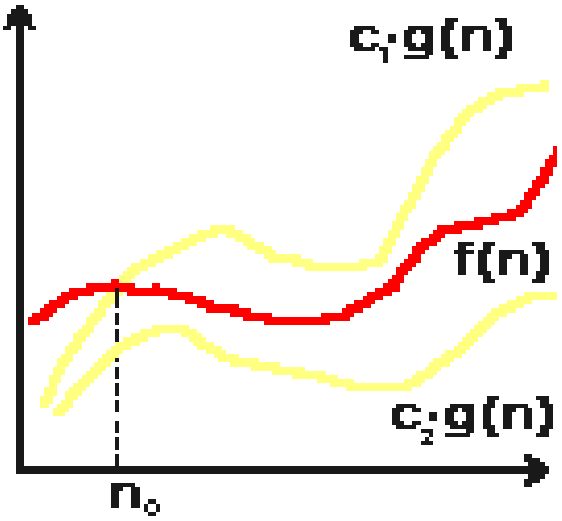
szacowania czasu działania algorytmu w przypadku pesymistycznym.

Notacja Θ (theta)

Notacja **Θ** ogranicza funkcję f(n) od góry, tak jak notacja O oraz dodatkowo od dołu.

Czyli, jeżeli f(n)=Θ(g(n)) to istnieje taki argument n_0, od którego począwszy dla każdego argumentu od niego niemniejszego:

1. wartości funkcji f(n) są niewiększe od wartości funkcji g(n) z dokładnością do stałej c1
2. wartości funkcji f(n) są niemniejsze od wartości funkcji g(n) z dokładnością do stałej c2

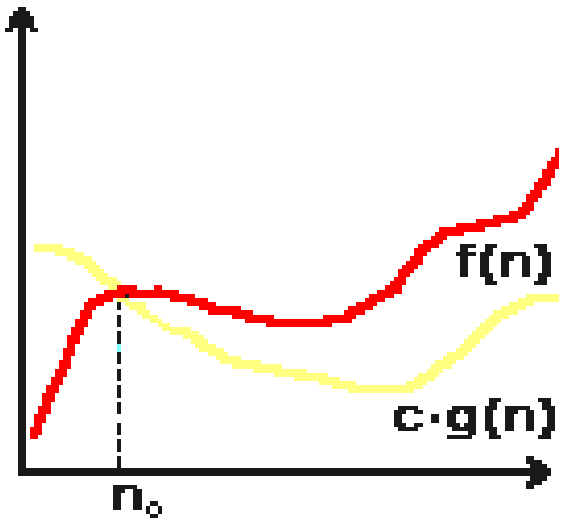


Jest to asymptotyczne oszacowanie dokładne.

Notacja **Ω (omega).**

Notacja ta ogranicza funkcję f(n) od dołu.

Czyli, jeśli f(n)=Ω(g(n)) to istnieje taki argument n_0, od którego począwszy dla każdego argumentu od niego niemniejszego funkcja f(n) jest niemniejsza niż g(n) z dokładnością do stałej c:



Jest to **asymptotyczna granica dolna.**

Służy do oszacowania działania algorytmu w najlepszym przypadku.