POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI
INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

WPROWADZENIE DO INFORMATYKI

Analiza algorytmów
Maszyna Turinga
Automaty i języki formalne

Dr hab. Małgorzata Charytanowicz

Fundusze Europejskie Wiedza Edukacja Rozwój
Rzeczpospolita Polska
Unia Europejska Europejski Fundusz Społeczny

Analiza algorytmów

Analiza algorytmów

Analiza algorytmu polega na **określeniu zasobów, jakie są potrzebne do jego wykonania**, takich jak:

- czas obliczeń,
- pamięć.

Algorytm jako pewna określona procedura obliczeniowa posiada cechy, które decydują o jej użyteczności. Są to:

- poprawność,
- złożoność obliczeniowa ,
- złożoność czasowa.

Złożoność najgorszego przypadku (pesymistyczna) – złożoność maksymalna dla wszystkich danych pewnego rozmiaru.

Złożoność oczekiwana – złożoność średnia dla wszystkich danych pewnego rozmiaru.

Analiza algorytmów

Algorytm poprawny to algorytm, który dla każdego egzemplarza problemu **zatrzymuje się** i **daje dobry wynik**, tym samym algorytm ten rozwiązuje zadany problem obliczeniowy.

- Poprawność algorytmu jest warunkiem koniecznym jego użyteczności.
- Metoda wnioskowania dowodzenia warunku stopu została opracowana przez Roberta Floyda w latach sześćdziesiątych, korzysta z zasady indukcji matematycznej.

Jeżeli testowanie programu nie jest wystarczające, jedynym sposobem określenia poprawności algorytmu jest **formalne dowodzenie** poprawności poprzez ustalanie relacji między trzema obiektami:

- stanem początkowym,
- instrukcją,
- stanem końcowym.

Poprawność algorytmu

Relacja między tymi trzema obiektami ma oznaczać, że jeśli w pierwszym stanie, zwanym początkowym, wykonana zostanie badana instrukcja, to przejdziemy do stanu drugiego, który nazwiemy końcowym.

Definicja 1.

Trójka $\{\{\varphi_1\}, P, \{\varphi_2\}\}$ to formuła logiczna, która jest prawdziwa wtedy i tylko wtedy, gdy zachodzi następujący warunek: dla każdego stanu spełniającego formułę φ_1, jeżeli program P zakończy swoje działanie, to stan końcowy będzie spełniał formułę φ_2 .

Takie formuły to tzw. **trójki Hoare'a** albo **formuły częściowej poprawności programu.** Zostały wprowadzone przez Anthony'ego Hoare'a w połowie lat 60.

Poprawność algorytmu

Definicja 2.

Asercją nazywamy warunek logiczny wyrażający zależność między zmiennymi algorytmu, a ich aktualnym stanem.

- **Asercją początkową** nazywamy asercję określającą dane wejściowe algorytmu.
- **Asercją końcową** nazywamy asercję określającą wyniki algorytmu.
- Asercje początkowa i końcowa są znane w momencie definiowania problemu, stanowiąc **specyfikację algorytmu.**

Definicja 3.

Algorytm *A* ma własność **określoności obliczeń** względem warunku α, co zapisujemy w postaci *obl* (α, *A*) , jeśli dla każdych danych spełniających warunek α działanie algorytmu *A* nie zostanie zerwane.

Poprawność algorytmu

Definicja 4.

Algorytm *A* ma **własność stopu** względem warunku α, co zapisujemy w postaci *stop*(α, *A*), jeśli dla każdych danych spełniających warunek α działanie algorytmu *A* nie ciągnie się w nieskończoność.

Definicja 5.

Algorytm *A* jest **częściowo poprawny** względem warunku początkowego α oraz warunku końcowego β, co zapisujemy w postaci *cp*(α,*A*, β), gdy dla każdych danych spełniających warunek początkowy α, jeżeli działanie algorytmu dochodzi do końca, wyniki spełniają warunek końcowy β.

Definicja 6.

Algorytm *A* jest **całkowicie poprawny** (semantycznie poprawny) względem warunku początkowego α oraz warunku końcowego β, co zapisujemy w postaci *sp*(α, *A*, β), jeśli ma własność określoności obliczeń względem warunku α, ma własność stopu względem warunku α oraz jest częściowo poprawny względem warunku α i względem warunku β.

Poprawność algorytmu

Przykład 1.

Rozważmy algorytm, który dla danych $x \in R$ oraz $n \in Z$ oblicza wartość $y = 1/x^n$.

```
y = 1
i = n
while i!=0
   y = y/x
   i = i-1
```

- Asercja początkowa: $\alpha_1 = \{ x \in R, n \in Z\}$
- Własność obliczalności zapewnimy przez: $\alpha_2 = \{ x \in R, x \neq 0, n \in Z\}$
- Własność stopu zapewnimy przez : $\alpha_3 = \{ x \in R, x \neq 0, n \in N_0\}$
- Asercja końcowa: $\beta = \{ y = 1/x^n \}$

Częściowa poprawność algorytmu

Do dowodzenia częściowej poprawności algorytmów wykorzystuje się zapis strukturalny algorytmu. Dowodzenie polega na rozbiciu dowodu częściowej poprawności całego algorytmu na ciąg dowodów częściowej poprawności składowych, stanowiących osobne algorytmy.

- **Instrukcja pusta**

  Częściową poprawność sekwencji opisuje równoważność:

  $cp(\alpha, , \beta) \Leftrightarrow \alpha => \beta$

  Jeżeli asercja kończąca składowej poprzedzającej jest inna od asercji początkowej składowej następującej, to sytuacja ta odpowiada instrukcji pustej umieszczonej między dwiema kolejnymi asercjami. Dlatego dla dowodu poprawności konieczne jest określenie poprawności sekwencji warunków sprowadzające się do wynikania z warunku poprzedzającego warunku następującego.

Częściowa poprawność algorytmu

- **Instrukcje przypisania** są elementarne z punktu widzenia algorytmu, stanowią składową niepodzielną. Formalnie częściową poprawność instrukcji podstawienia opisuje równoważność:

$$cp(\alpha, z \leftarrow w, \beta) \Leftrightarrow \alpha \Rightarrow \beta|_{z \leftarrow w}$$

Przy asercji początkowej α oraz asercji końcowej β podstawienie $z \leftarrow w$ jest częściowo poprawne wtedy i tylko wtedy, gdy dowiedziemy implikacji od asercji α do asercji β, z zastąpieniem każdego wystąpienia zmiennej z podstawianym wyrażeniem w.

- **Sekwencja instrukcji**

Częściową poprawność sekwencji obejmującej instrukcje I_1 oraz I_2 przy warunku początkowym α i warunku końcowym β opisuje warunek:

$$cp(\alpha, I_1; I_2, \beta) \Leftrightarrow \exists\gamma: cp(\alpha, I_1, \gamma) \wedge cp(\gamma, I_2, \beta)$$

Konieczne jest znalezienie pośredniego warunku γ, pozwalającego osobno określić częściowe poprawności instrukcji sekwencji.

Częściowa poprawność algorytmu

- **Krótka instrukcja warunkowa**

  Częściową poprawność krótkiej instrukcji warunkowej wykonującej działanie I w zależności od spełniania warunku γ przy warunku początkowym α i warunku końcowym β opisuje równoważność:

  $$cp(\alpha, \text{ if } \gamma \text{ then } I, \beta) \Leftrightarrow ((\alpha \wedge \sim \gamma) \Rightarrow \beta) \wedge cp(\alpha \wedge \gamma, I, \beta)$$

  Instrukcja nie wykona się w przypadku fałszywości warunku γ, mimo to algorytm powinien działać poprawnie. Stąd konieczność wynikania warunku końcowego β z koniunkcji warunku początkowego α i zaprzeczenia warunku γ.

Częściowa poprawność algorytmu

- **Długa instrukcja warunkowa**

  Częściową poprawność długiej instrukcji warunkowej wykonującej działania I_{prawda} oraz $I_{\text{fałsz}}$ w zależności od wartości logicznej warunku γ określa równoważność:

  $cp(\alpha, \text{if } \gamma \text{ then } I_{\text{prawda}} \text{ else } I_{\text{fałsz}}, \beta) \Leftrightarrow cp(\alpha \wedge \gamma, I_{\text{prawda}}, \beta) \wedge cp(\alpha \wedge \sim\gamma, I_{\text{fałsz}}, \beta)$

  Każdy możliwy przebieg działania algorytmu powinien mieć osobno udowodnioną częściową poprawność. Przebiegi zależą od wartości warunku γ, skąd bierze się wzmocnienie warunku początkowego α odpowiednim wartościowaniem.

Częściowa poprawność algorytmu

- **Pętla *while***

  Częściową poprawność pętli *dopóki* określa równoważność:

$$cp(\alpha, \text{while } \gamma \text{ do } I, \beta) \Leftrightarrow$$
$$\exists\ \delta_1, \delta_2: cp(\delta_1, I, \delta_2) \wedge (\alpha \wedge \sim\gamma \Rightarrow \beta) \wedge (\alpha \wedge \gamma \Rightarrow \delta_1) \wedge$$
$$(\delta_2 \wedge \sim\gamma \Rightarrow \beta) \wedge (\delta_2 \wedge \gamma \Rightarrow \delta_1)$$

  Konieczne jest wymyślenie warunków związanych z częściową poprawnością ciała pętli, poprawnością całości algorytmu w przypadku ominięcia wykonania ciała pętli, poprawnością wejścia do wnętrza pętli, poprawności przerwania trwania pętli oraz poprawnością trwania pętli.

Analiza algorytmów

- **Złożoność czasowa algorytmu** – wymagany przez algorytm czas wyrażony jako funkcja rozmiaru zadania.
- **Asymptotyczną złożonością czasową** – zachowanie się tej złożoności w granicy, gdy rozmiar zadania wzrasta.
- Asymptotyczna złożoność algorytmu rozstrzyga o rozmiarze zadania, które może być rozwiązane przez ten algorytm.
- **Rozmiar danych wejściowych** może być wyrażony:
  - liczbą elementów w ciągu wejściowym (sortowanie),
  - liczbą bitów (reprezentacji binarnej liczb),
  - rozmiarem macierzy (mnożenie macierzy),
  - liczbą wierzchołków i krawędzi (grafy).
- **Czas działania** algorytmu dla konkretnych danych może być wyrażony liczbą elementarnych operacji lub kroków.

Złożoność algorytmów

- **Złożoność obliczeniowa algorytmu** jest określana ilością zasobów systemu liczącego potrzebnych do jego wykonania.
- Podstawowymi zasobami są:
  - czas procesora (czas działania algorytmu),
  - obszar zajętej pamięci.
- Wyróżniamy:
  - złożoność czasowa,
  - złożoność pamięciową.
- Złożoność czasowa zależy od wszystkich jednostkowych czynności wykonywanych podczas realizacji algorytmu.
- Złożoność pamięciowa określa zapotrzebowanie algorytmu na pamięć.
- Za **jednostkę złożoności pamięciowej** przyjmuje się zwykle słowo pamięci komputera.

Przykład

Czynności jednostkowe obliczania wartości wyrażenia `w(x)`:

$$w(x) = 3x^6 + 2x^4 + 5x^2 + 10$$

dla danej wartości `x`.

Algorytm 1. `w = 3*x*x*x*x*x*x + 2*x*x*x*x + 5*x*x + 10`

Liczba operacji jednostkowych: 16

- 1 przypisanie,
- 3 dodawania,
- 12 mnożeń.

Algorytm 2 . `Niech t = x*x`

`w = ((3*t + 2)*t +5)*t + 10`

Liczba operacji jednostkowych: 8

- 2 przypisania
- 3 dodawania
- 3 mnożenia.

Czynności jednostkowe

Za **czynności jednostkowe** przyjmuje się:

- wykonanie operacji arytmetycznej (dodawanie, odejmowanie, mnożenie, dzielenie, itp.),
- nadanie wartości zmiennej,
- zbadanie relacji lub wykonanie operacji logicznej,
- wprowadzenie danej.

- Często dokładne zliczanie operacji jednostkowych jest bardzo trudne lub niemożliwe i w związku z tym wyróżnia się tzw. **operacje dominujące.**
- Za operacje dominujące w danym algorytmie uważa się te, których liczba jest proporcjonalna do liczby wykonań wszystkich operacji jednostkowych w dowolnej komputerowej realizacji tego algorytmu.
- Za **jednostkę złożoności czasowej** przyjmuje się wykonanie jednej operacji dominującej.

Pesymistyczna złożoność czasowa

Pesymistyczna złożoność czasowa algorytmu określana jest funkcją $w(n)$:

$$w(n) = \sup\{t(d) : d \in D_n\}$$

gdzie:

n – rozmiar danych,

$t(d)$ – liczba operacji dominujących dla zestawu danych wejściowych d,

D_n – zbiór zestawów danych wejściowych o rozmiarze n.

Oczekiwana złożoność czasowa

Oczekiwana złożoność czasowa algorytmu określana jest funkcją $A(n)$:

$$A(n) = \sum_{k \geq 1}^{n} k \cdot p_{nk}$$

gdzie:

$A(n)$ – wartość oczekiwana zmiennej losowej X_n,

X_n – zmienna losowa, której wartością jest liczba operacji dominujących dla zestawu danych wejściowych $d \in D_n$,

p_{nk} – prawdopodobieństwo, że dla danych o rozmiarze n algorytm wykona k operacji dominujących.

Przyrost funkcji

Czas działania algorytmów charakteryzowany jest zwykle przez parametr *n*, który może być:

- stopniem wielomianu,
- rozmiarem pliku lub liczbą elementów do posortowania lub przeszukania,
- liczbą znaków przetwarzanego napisu, itp.

Czas działania algorytmów jest zwykle proporcjonalny do jednej z funkcji:

- 1 – program lub algorytm wykonuje się raz lub kilka razy; jeżeli wszystkie instrukcje programu mają tę własność, mówimy że czas działania jest stały,
- log *n* – program spowalnia wraz ze wzrostem *n*; złożoność logarytmiczna występuje dla algorytmów typu: zadanie rozmiaru *n* zostaje sprowadzone do zadania rozmiaru *n*/2 plus pewna stała liczba działań, np. algorytm *wyszukiwania binarnego*; jeżeli podwoimy wartość *n* wartość log *n* zwiększy się o stałą, wartość log *n* można podwoić dopiero poprzez wzrost od *n* do n^2,

Przyrost funkcji

- n – złożoność liniowa występuje dla algorytmów, w których jest wykonywana pewna stała liczba działań dla każdego z n elementów danych wejściowych, np. *schemat Hornera*, taka sytuacja jest optymalna, jeżeli algorytm musi przetworzyć n obiektów wejściowych,
- $n\log n$ – złożoność liniowo-logarytmiczna występuje, gdy zadanie rozmiaru n zostaje sprowadzone do zadania rozmiaru $n/2$ plus pewna stała liczba działań liniowa względem rozmiaru n, potrzebna najpierw do rozbicia zadania, a potem do scalenia rozwiązań, np. algorytm *QuickSort*,
- n^2 – złożoność kwadratowa, występuje dla algorytmów z zagnieżdżoną instrukcją iteracyjną, np. *sortowanie bąbelkowe;* zwykle algorytm musi przejrzeć wszystkie pary n elementów wejściowych,
- 2^n – złożoność wykładnicza, tylko nieliczne algorytmy o takiej złożoności są akceptowalne w rozwiązaniach praktycznych.

Przyrost funkcji – podsumowanie

Funkcje złożoności	Uwagi
1	Bez względu na rozmiar danych czas działania jest taki sam, występuje raczej jako wielokrotne wykorzystanie fragmentu bardziej złożonych algorytmów
log *n*	Bardzo szybki algorytm, stosowany wielokrotnie
n	Dla każdej danej wykonujemy stałą liczbę operacji
*n*log*n*	Bardzo wysoka efektywność
n^2, n^3	Dobra efektywność, ale czasem poszukuję się algorytmów o lepszej złożoności
n^p	Złożoność wielomianowa, dla p > 3 niska efektywność, pożądane jest obniżenie złożoności
2^n	Fatalna efektywność, zawsze szuka się lepszego algorytmu
n!	Fatalna efektywność, zawsze szuka się lepszego algorytmu.

Podstawy matematyczne
Zależności rekurencyjne

Standardowe notacje

Części całkowite liczb

- Dla każdej liczby rzeczywistej *x* zapis $\lfloor x \rfloor$ oznacza największą liczbę całkowitą nie większa niż *x*, natomiast zapis $\lceil x \rceil$ oznacza najmniejszą liczbę całkowitą, nie mniejszą niż *x*

$$x - 1 < \lfloor x \rfloor \leq x \leq \lceil x \rceil < x + 1$$

- Dla każdej liczby naturalnej *n* jest

$$\lceil n/2 \rceil + \lfloor n/2 \rfloor = n$$

Wzór Stirlinga

- Wzór Stirlinga stosowany jest do aproksymacji *n*!

$$\log n! \approx n\log n - n\log e + \log\sqrt{2\pi n}$$

Szacowanie sum

Jeżeli suma da się wyrazić w postaci $\sum_{k=m}^{n} f(k)$, gdzie $f(k)$ jest funkcją monotonicznie rosnącą, możemy oszacować ją przez całki

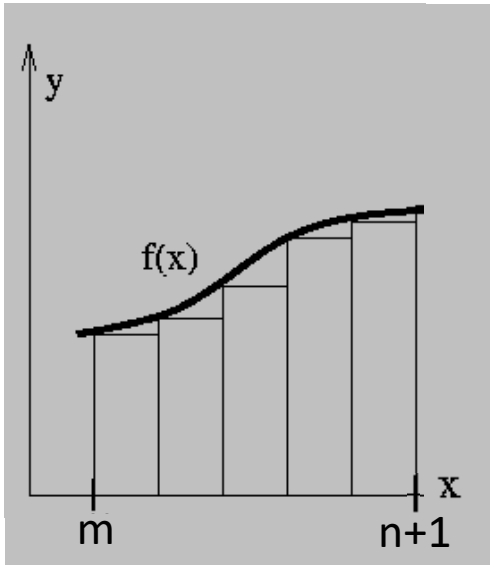
$$\int_{m-1}^{n} f(x)dx \leq \sum_{k=m}^{n} f(k) \leq \int_{m}^{n+1} f(x)dx$$

Przykład 3.

$$\sum_{k=2}^{n} \frac{1}{k} \leq \int_{1}^{n} \frac{1}{x} dx$$

Stąd

$$\sum_{k=1}^{n} \frac{1}{k} \leq \ln n + 1$$



Analiza algorytmów

Tab. 1. Wartości funkcji najczęściej występujących w analizie złożoności algorytmów

$\log n$	$\sqrt{n}$	n	$n \log n$	$n (\log n)^2$	$n^{3/2}$
3	3	10	33	110	32
7	10	100	664	4414	1000
10	32	1000	9966	99317	31623
13	100	10000	132877	1765633	1000000
17	316	100000	1660964	27588016	31622777
20	1000	1000000	19931569	397267426	1000000000

- Szczególną rolę w projektowaniu i analizie algorytmów odgrywa funkcja logarytmiczna.
- W informatyce najczęściej stosuje się logarytm o podstawie 2.

Analiza algorytmów

Tab. 2. Czasy rozwiązywania dużych problemów

Liczba operacji na sekundę	Zadanie o rozmiarze 1 miliona		
	n	$n \log n$	n^2
10^6	sekundy	sekundy	tygodnie
10^9	natychmiast	natychmiast	godziny
10^{12}	natychmiast	natychmiast	sekundy

Analiza algorytmów

Tab. 3. Czasy rozwiązywania dużych problemów

Liczba operacji na sekundę	Zadanie o rozmiarze 1 miliarda		
	n	$n \log n$	n^2
10^6	godziny	godziny	nigdy
10^9	sekundy	sekundy	lata
10^{12}	natychmiast	natychmiast	tygodnie

Zależności rekurencyjne

- Znaczna część algorytmów oparta jest na rekurencyjnej dekompozycji dużego problemu na mniejsze zdania cząstkowe.
- Czas działania algorytmów rekurencyjnych można określić na podstawie:
  - liczby zadań cząstkowych,
  - czasu potrzebnego na przeprowadzenie dekompozycji.
- Aby wyznaczyć czas działania w zależności od *n* należy przekształcić wór rekurencyjny na postać nierekurencyjną, to znaczy rozwikłać zależność rekurencyjną.

Zależności rekurencyjne

Przykład 1.

Rozwikłać zależność rekurencyjną postaci

$$C_n = \begin{cases} C_{n-1} + n & \text{dla} \quad n \geq 2 \\ 1 & \text{dla} \quad n = 1 \end{cases}$$

występującą w programach rekurencyjnych, które przeglądają w pętli dane wejściowe w celu wyeliminowania jednego elementu.

Rozwiązanie:

C_n jest bliskie $n^2/2$:

$$C_n = C_{n-1} + n = C_{n-2} + (n-1) + n = \cdots =$$

$$1 + 2 + \cdots + n = \frac{n(n+1)}{2}$$

Zależności rekurencyjne

Przykład 2.

Rozwikłać zależność rekurencyjną postaci

$$C_n = \begin{cases} C_{n/2} + 1 & \text{dla} \quad n \geq 2 \\ 1 & \text{dla} \quad n = 1 \end{cases}$$

występującą w programach rekurencyjnych, które przy każdym wykonaniu dzielą dane wejściowe na dwie części.

Rozwiązanie:

Niech najpierw $n = 2^m$. Wtedy rekurencja jest dobrze zdefiniowana i C_n jest bliskie log$n$:

$$C_n = C_{2^m} = C_{2^{m-1}} + 1 = C_{2^{m-2}} + 1 + 1 = \cdots =$$

$$C_{2^0} + m = m + 1 = \log n + 1$$

Zależności rekurencyjne

- Rozwiązanie dla dowolnego *n* zależy od interpretacji *n*/2.
- Zauważmy, że usuwając z reprezentacji binarnej liczby *n* skrajny prawy bit , otrzymujemy liczbę o wartości $\lfloor n/2 \rfloor$.
- Oznacza to, że liczba bitów reprezentacji binarnej liczby *n* jest o 1 większa od liczby bitów reprezentacji binarnej liczby $\lfloor n/2 \rfloor$.
- Jeżeli *n*/2 zinterpretujemy jako $\lfloor n/2 \rfloor$ to rozwikłaniem wzoru rekurencyjnego będzie liczba $\lfloor \log_2 n \rfloor + 1$, oznaczająca liczbę bitów reprezentacji binarnej liczby *n*.

Zależności rekurencyjne

Przykład 3.

Rozwikłać zależność rekurencyjną postaci

$$C_n = \begin{cases} 2C_{n/2} + n & \text{dla} \quad n \geq 2 \\ 0 & \text{dla} \quad n = 1 \end{cases}$$

występującą w programach przeglądających liniowo dane wejściowe przed, podczas albo po podziale danych na dwie połowy. Taka rekurencja występuje w algorytmach typu „dziel i rządź”.

Rozwiązanie:

C_n jest bliskie *n*log*n* ($n = 2^m$):

$$C_n = C_{2^m} = 2C_{n/2} + n = 2C_{2^{m-1}} + 2^m$$

$$\frac{C_{2^m}}{2^m} = \frac{C_{2^{m-1}}}{2^{m-1}} + 1 = \frac{C_{2^{m-2}}}{2^{m-2}} + 1 + 1 = \cdots = m$$

Notacja asymptotyczna

Definicja 1.

Mówimy, że funkcja $f(n)$ jest $O(g(n))$, jeżeli istnieje dodatnia stała c taka, że

$$f(n) \leq c\, g(n)$$

dla wszystkich nieujemnych wartości n prócz pewnego skończonego (być może pustego) zbioru tych wartości.

Notacja $\Theta(g(n))$ oznacza, że funkcja f jest ograniczona od dołu i od góry:

$$0 \leq c_1\, g(n) \leq f(n) \leq c_2\, g(n)$$

Przykład 4.

Jeżeli algorytm przetwarza dane o rozmiarze n w czasie cn^2 dla pewnej stałej dodatniej c, to mówimy że czasowa złożoność tego algorytmu jest rzędu n^2, czyli jest $O(n^2)$.

Notacja asymptotyczna

Przykład 5.

Wyraź funkcję $0.5n^2 - 3n$ używając notacji dużego Θ.

$$c_1 n^2 \leq 0.5n^2 - 3n \leq c_2 n^2$$

$$c_1 \leq \frac{1}{2} - \frac{3}{n} \leq c_2$$

$$c_1 \leq \frac{n-6}{2n} \leq c_2$$

Lewa strona jest prawdziwa dla $n > 6$ i $c_1 = \frac{1}{14}$

Prawa strona jest prawdziwa dla $n > 6$ i $c_2 = 0.5$

Zatem $0.5n^2 - 3n = \Theta(n^2)$

Notacja asymptotyczna

Składniki niższego rzędu mogą być pominięte, gdyż są mało istotne dla dużych *n*.

- Dla funkcji kwadratowej:

$$f(n) = an^2 + bn + c, \text{dla } a > 0 \qquad f(n) = \Theta(n^2)$$

- Dla wielomianu :

$$p(n) = \sum_{i=0}^{m} a_i n^i, \text{dla } a_m > 0 \qquad p(n) = \Theta(n^m)$$

- Jeżeli notacja asymptotyczna pojawia się we wzorze, interpretujemy ją jako anonimową funkcję o pomijalnej nazwie, na przykład:

$$T(n) = 2T(n/2) + \Theta(n)$$

Obliczanie sumy

Przykład 6.

Zaproponuj algorytm obliczający wartość sumy liczb wpisanych do tablicy:

1					
1	2				
1	2	3			
…	…	…	…		
1	2	3	…	n-1	
1	2	3	…	n-1	n

Suma(n) = 1 + (1+2) + (1+2+3) + … + (1+2+…+n-1) + (1+2+…+n)

Przykład

Algorytm: obliczanie sumy

```
def  suma(n):
    s = 0
    for  i  in range(1,n+1):
        for j in range(1, i+1):
            s = s + j
    return s
```

Liczba dodawań:

$1 + 2 + 3 + \ldots + n = n(1 + n)/2 = \Theta(n^2)$

Obliczanie tablicy różnic

Przykład 7.

Dany jest (n+1)- elementowy ciąg liczb $a_0, a_1, \ldots, a_n$. Skonstruuj ciąg różnic odejmując od elementu a_{k+1} element a_k. Operację należy powtarzać aż do uzyskania jednego elementu.

a_0	$b_0 = a_1 - a_0$	$c_0 = b_1 - b_0$	…	…	…
a_1	$b_1 = a_2 - a_1$	$c_1 = b_2 - b_1$		…	
a_2	$b_2 = a_3 - a_2$	…	…		
…	…	$c_{n-2} = b_{n-1} - b_{n-2}$			
a_{n-1}	$b_{n-1} = a_n - a_{n-1}$				
a_n					

Liczba odejmowań: $S(n) = 1 + 2 + 3 + \ldots + n = n(1+n)/2$

Wyszukiwanie sekwencyjne

Zadanie 1.

Określ złożoność czasową algorytmu wyszukiwania elementu ***a*** w *n*-elementowym ciągu ***T*** **.**

Dane początkowe:

n – liczba naturalna,

a – poszukiwany element,

T – nieposortowany ciąg elementów $\{ t_j \}$, $j = 0, \ldots, n-1$.

Algorytm: wyszukiwanie sekwencyjne

```
j = 0
while l[j] < n:
    if a == t[j] return j
    j = j + 1
return -1
```

Wyszukiwanie sekwencyjne

Wyszukiwanie sekwencyjne wymaga *n* sprawdzeń, jeśli kończy się niepowodzeniem oraz przeciętnie *n*/2 sprawdzeń, jeśli kończy się sukcesem:

- Pesymistyczna złożoność czasowa: *n*
- Oczekiwana złożoność czasowa:

$$A(n) = \frac{1 + 2 + \cdots + n}{n} = \frac{n + 1}{2}$$

Wyszukiwanie binarne

Algorytm: wyszukiwanie w tablicy uporządkowanej

```
def binarne(T, left, right, a):
    while left <= right:
        k = (left+right)//2
        if a == T[k]:
            return k
        if a < T[k]:
            right = k-1
        else:
            left = k+1
    return None
```

Dla tablicy `n`-elementowej indeksowanej od 0 wywołanie funkcji:

```
left = 0, right = n-1
```

Wyszukiwanie binarne

Wyszukiwanie binarne nie wymaga nigdy więcej niż $\lfloor \log_2 n \rfloor + 1$ sprawdzeń:

- Niech $T(n)$ reprezentuje liczbę porównań wymaganych w najgorszym przypadku. Wówczas z faktu, iż algorytm redukuje wielkość przeglądanej tablicy o połowę, wynika że:

$$T_n = \begin{cases} T_{\lfloor n/2 \rfloor} & \text{dla} \quad n \geq 2 \\ 1 & \text{dla} \quad n = 1 \end{cases}$$

- Zatem na podstawie rozważań z Przykładu 2 jest T(n) ≤ $\lfloor \log_2 n \rfloor + 1$
- Problem wyszukiwania dla danych o wielkości do jednego miliona wykorzystuje maksymalnie 20 porównań.

Analiza algorytmów

Tab. 4. Ograniczenia rozmiaru zadania wynikające ze złożoności czasowej algorytmu (jednostka czasu jest równa 1 milisekundzie)

Algorytm	Złożoność czasowa	Maksymalny rozmiar zadania n	
		1 sekunda	1 minuta
A1	n	1000	6×10^4
A2	$n \log_2 n$	140	4893
A3	n^2	31	244
A4	n^3	10	39
A5	2^n	9	15

Klasy złożoności obliczeniowej

Klasy algorytmów pod względem złożoności obliczeniowej:

- łatwe (P –polynomial), rozwiązywalne co najwyżej w czasie wielomianowym
- trudne (NP –non-polynomial), wymagające czasu dłuższego niż wielomianowy,
- NP zupełne (NPC –NP complete) – nie udowodniono, iż nie posiadają rozwiązania wielomianowego.

Donald Knuth

Donald Ervin Knuth – amerykański matematyk, informatyk, profesor na katedrze informatyki Uniwersytetu Stanforda.

- Jeden z pionierów informatyki, znany z dzieła *Sztuka programowania*, uznawanego za najbardziej zaawansowane opracowanie na temat analizy algorytmów.
- Jest autorem systemu składu drukarskiego TeX i języka opisu fontów METAFONT oraz twórcą i propagatorem techniki *literate programming*.
- Znacząco rozwinął algorytmikę, opracował teoretycznie wiele zagadnień z zakresu matematyki i informatyki.
- Ważniejsze nagrody: Nagroda Grace Murray Hopper (1971), Nagroda Turinga (1974), Narodowy Medal Nauki (1979), Medal Johna von Neumanna (1995), Nagroda Harveya (1995), Nagroda Kioto (1996).
- Zajmował się analizą starobabilońskich procedur obliczeniowych, uznając je za prawdziwe algorytmy.

Maszyna Turinga

Wprowadzenie

Alan Turing – żył w latach 1912-1954, jedna z najważniejszych postaci informatyki, brytyjski matematyk, kryptolog, uważany za ojca sztucznej inteligencji.

Alan Turing

- twórca podstaw obliczalności, zajmującej się określeniem tego, co jest możliwe do automatycznego wyliczenia,
- twórca teoretycznego modelu współczesnego komputera zwanego maszyną Turinga, uważanego za jedno z **najpiękniejszych i najbardziej intrygujących odkryć intelektualnych** XX wieku,
- jedna z głównych postaci brytyjskiego ośrodka łamania kodu niemieckiej maszyny szyfrującej Enigma,
- konstruktor jednego z pierwszych elektronicznych komputerów,
- twórca tak zwanego testu Turinga, sprawdzającego zdolność maszyn do rozumienia języka naturalnego.

Maszyna Turinga

W koncepcyjnym opisie część sprzętowa urządzenia obejmuje:

- **obustronnie nieskończoną taśmę** podzieloną na klatki przechowujące pojedyncze znaki,
- **głowicę** poruszającą się wzdłuż taśmy.

Idea pracy głowicy obejmuje w jednym cyklu:

- **odczyt znaku** z aktualnej klatki,
- **zapis znaku** w aktualnej klatce oraz przesunięcie głowicy o jedną klatkę w lewo (L) lub w prawo (R), z ewentualnością pozostania na miejscu (S).

Oprócz powyższego sprzętu (*hardware*) maszyna Turinga:

- posiada **sterowanie** (*software*), obejmujące reguły zachowania głowicy w zależności od odczytanego znaku;
  różnorodne zestawy reguł sterowania maszyną Turinga mają na celu realizację obliczeń rozwiązań zadanych problemów.

Maszyna Turing stanowi teoretyczny komputer. Wszystkie realnie działające komputery można zastąpić odpowiednią maszyną Turinga.

Maszyna Turinga

Taśma

- Nieskończona taśma jest odpowiednikiem współczesnej **pamięci komputera**.
- Taśma dzieli się na komórki, w których umieszczone zostały symbole, czyli znaki przetwarzane przez maszynę Turinga. Symbole te stanowią odpowiednik **danych wejściowych**.
- Maszyna Turinga odczytuje te dane z kolejnych komórek i przetwarza na inne symbole, czyli dane wyjściowe.
- Wyniki obliczeń również są zapisywane w komórkach taśmy.
- Można definiować różne symbole dla maszyny Turinga. Najczęściej rozważa się jedynie **symbole 0, 1** (alfabet) oraz tzw. znak pusty – czyli zawartość komórki, która nie zawiera żadnej danej do przetworzenia.

Maszyna Turinga

Głowica

- Aby przetwarzać dane, maszyna Turinga musi je odczytywać i zapisywać na taśmę. Do tego celu przeznaczona jest głowica zapisująco-odczytująca, która odpowiada funkcjonalnie **urządzeniom wejścia/wyjścia** lub układom odczytu i zapisu pamięci.
- Głowica zawsze znajduje się nad jedną z komórek taśmy. Może ona odczytywać zawartość tej komórki oraz zapisywać do niej inny symbol.
- Oprócz odczytywania i zapisywania symboli w komórkach głowica wykonuje **ruchy w prawo lub w lewo** (R lub L) do sąsiednich komórek na taśmie, brak ruchu głowicy oznaczany jest literą S (the same) lub N (none). W języku polskim ruch głowicy oznaczamy symbolami P, L N, brak ruchu głowicy oznaczamy również poziomą kreską; by uniezależnić opis od wersji językowej stosuje się symbole strzałek →, ←, ↓.
- Przed rozpoczęciem pracy maszyny Turinga głowica jest ustawiana nad komórką taśmy zawierającą pierwszy symbol do przetworzenia.

Maszyna Turinga

Układ sterowania

- Sterowanie opisane jest skończonym zbiorem reguł δ odpowiadających pojedynczym możliwym cyklom pracy. Operacje wykonywane przez układ sterowania zależą od:
  - symbolu odczytanego z komórki na taśmie,
  - bieżącego stanu układu sterującego.
- Układ sterowania odczytuje za pomocą głowicy symbole z komórek taśmy i przesyła do głowicy symbole do zapisu oraz określa ruch głowicy:
  - **Niedeterministyczna maszyna Turinga** umożliwia przyporządkowanie jednej parze złożonej ze stanu i czytanego symbolu wielu trójek złożonych z nowego symbolu, stanu i ruchu głowicy. Jest rzadziej wykorzystywana.
  - **Deterministyczna maszyna Turinga** na relację opisującą sterowanie nakłada warunek bycia funkcją, gwarantujący jedno możliwe działanie dla każdego układu stanu i czytanego symbolu.
- Stany oznaczamy symbolami q_0, q_1, q_2, ... ,q_n, gdzie q_0 jest stanem początkowym przed rozpoczęciem przetwarzania symboli na taśmie.

Maszyna Turinga

Maszyna Turinga jest definiowana przez siódemkę:

$M = (Q, \Gamma, \Sigma, \delta, q_0, B, F)$,

gdzie:

- Q jest niepustym, skończonym i uporządkowanym zbiorem stanów,
- Γ jest niepustym, skończonym i uporządkowanym zbiorem symboli taśmy,
- $\Sigma \in \Gamma$ jest zbiorem symboli wejściowych, jest podzbiorem zbioru symboli taśmy z odrzuceniem symbolu pustego,
- $B \in \Gamma - \Sigma$ oznacza symbol pusty,
- q_0 jest stanem początkowym,
- F jest stanem końcowym,
- δ: $Q \times \Gamma \rightarrow \Gamma \times Q \times \{L,R\}$ to funkcja przejścia odwzorowująca dwuelementowy układ maszyny złożony ze stanu i symbolu w układ trójelementowy złożony z nowego symbolu i stanu oraz określenia ruchu głowicy (L lub R) .

Maszyna Turinga

Zbiór stanów maszyny Turinga:

- niepusty, skończony, uporządkowany **zbiór stanów** *Q*; maszyna Turinga zawsze znajduje się jednym z możliwych stanów, w jednym cyklu może zmienić stan na inny, również ze zbioru stanów *Q*,
- **stan początkowy** q_0 to szczególny wyróżniony stan ze zbioru stanów *Q*, w którym maszyna Turinga znajduje się na początku pracy,
- **zbiór stanów końcowych** F to podzbiór zbioru stanów *Q* powodujących zatrzymanie maszyny Turinga.

Maszyna Turinga

Zbiór symboli taśmy maszyny Turinga:

- zbiór symboli taśmy Γ jest dowolnym niepustym, skończonym i uporządkowanym zbiorem, zawierającym:
  - **symbol pusty *B***: taśma maszyny Turinga bez żadnej ingerencji jest zapełniona wyłącznie symbolami pustymi, naturalnym symbolem pustym jest znak spacji, z powodu braku widoczności w praktyce zastępuje się go innymi znakami, np. znakiem #,
  - zbiór **symboli wejściowych** Σ jest podzbiorem zbioru symboli taśmy Γ z odrzuceniem symbolu pustego; w praktyce oznacza użyteczne symbole wprowadzone dla bezpośredniego działania maszyny Turinga.

Maszyna Turinga

Instrukcją dla maszyny Turinga jest piątka:

$(S_o, q_i, S_z, q_j, L/R)$

gdzie:

S_o – symbol odczytany przez głowicę z bieżącej komórki na taśmie,

q_i – bieżący stan układu sterowania,

S_z – symbol, jaki zostanie zapisany w bieżącej komórce na taśmie,

q_j – nowy stan, w który przejdzie układ sterowania po wykonaniu tej operacji.

- Dziedzina sterowania deterministycznej maszyny Turinga jest dwuwymiarowa, dlatego funkcja przejścia dla maszyny Turinga jest zapisywany jako tablica instrukcji.
- Dla argumentów, dla których tablica nie mam określonej wartości maszyna zatrzymuje się.

Przykład 1.

Maszyna Turinga jest zdefiniowana przez:

- zbiór stanów $Q = \{q_0, q_1, q_2, q_3\}$
- zbiór symboli taśmy $\Gamma = \{0, 1\}$
- stan początkowy q_0
- symbol pusty $B = \{0\}$
- stan końcowy $F = \{q_3\}$
- funkcję przejścia określoną za pomocą tablicy sterującej:

δ	0	1
q_0	$1, q_1, \rightarrow$	$1, q_2, \leftarrow$
q_1	$1, q_0, \leftarrow$	$1, q_1, \rightarrow$
q_2	$1, q_1, \leftarrow$	$1, q_3, \rightarrow$
q_3		

Przykład 1.

Funkcja przejścia może być określona również za pomocą **diagramu stanów**.

Diagram stanów jest grafem, którego:

- wierzchołki są etykietowane stanami,
- krawędzie odpowiadają niepustym elementom tablicy sterującej.

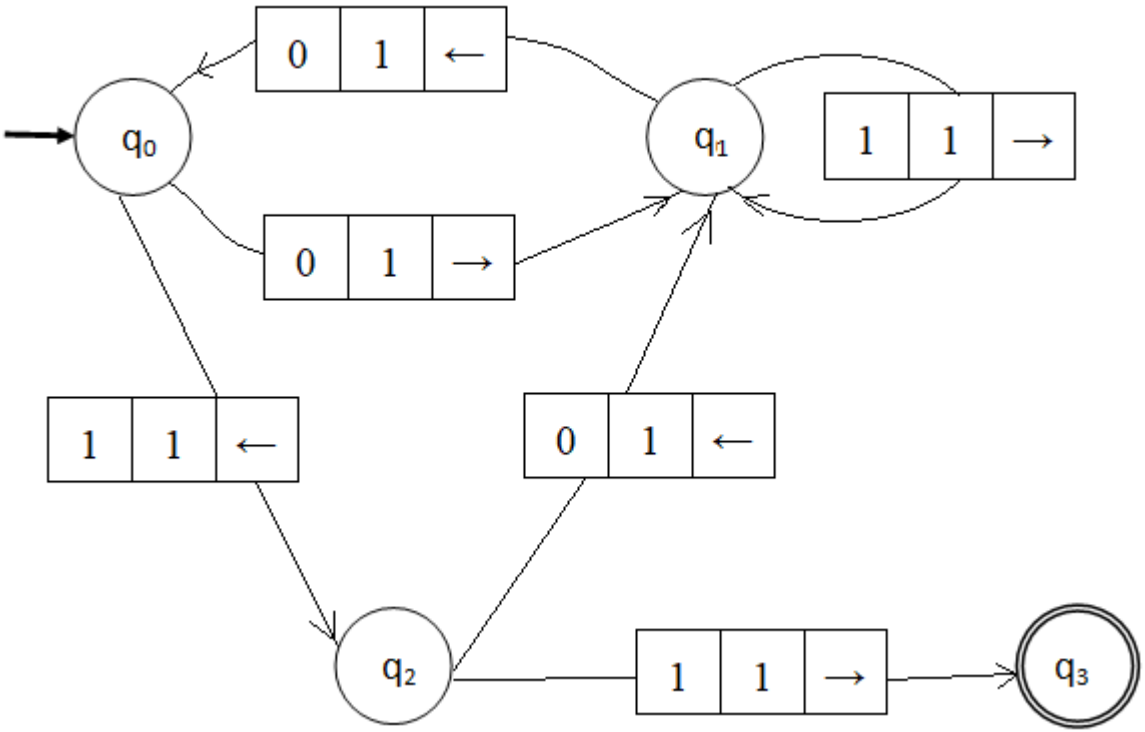
Etykietami krawędzi są trójki obejmujące:

- symbol odczytany,
- symbol pisany,
- określenie ruchu głowicy.

Stan początkowy wyróżniamy strzałką.

Stan końcowy wyróżniamy podwójną linią.

Diagram stanów



Opis sterowania

- Zadanie polega na opracowaniu maszyny Turinga z dwuznakowym zbiorem symboli taśmy i zadaną liczbą stanów nie końcowych (trzech), działającej możliwie jak najdłużej, ale nie w nieskończoność.
- Na początku taśma zawiera wyłącznie symbole puste.
- W pierwszym cyklu pracy, maszyna będąc w stanie początkowym q_0, odczyta znak 0, wydrukuje znak 1, przejdzie do stanu q_1, przesuwając głowicę w prawo na kolejny znak 0.
- Będąc w stanie q_1, odczyta znak 0, wydrukuje znak 1, przejdzie do stanu q_0, przesuwając głowicę w lewo na znak 1, itd.
- Działanie maszyny Turinga można prześledzić na diagramie postępu obliczeń, obrazującym w kolejnych wierszach aktualną zawartość taśmy z wyróżnioną aktualną pozycją i aktualnym stanem.
- Stan początkowy przedstawia wiersz 1, stan po pierwszym cyklu wiersz 2, itd. Kontynuując analizę działania, w stanie końcowym q_3 otrzymujemy ciąg sześciu znaków 1, przedstawiony w wierszu 14.

Diagram postępu obliczeń

1	q_0	0	0	0	0	**0**	0	0	0
2	q_1	0	0	0	0	1	**0**	0	0
3	q_0	0	0	0	0	**1**	1	0	0
4	q_2	0	0	0	**0**	1	1	0	0
5	q_1	0	0	**0**	1	1	1	0	0
6	q_0	0	**0**	1	1	1	1	0	0
7	q_1	0	1	**1**	1	1	1	0	0
8	q_1	0	1	1	**1**	1	1	0	0
9	q_1	0	1	1	1	**1**	1	0	0
10	q_1	0	1	1	1	1	**1**	0	0
11	q_1	0	1	1	1	1	1	**0**	0
12	q_0	0	1	1	1	1	**1**	1	0
13	q_2	0	1	1	1	**1**	1	1	0
14	q_3	0	1	1	1	1	**1**	1	0

Przykład 2.

Maszyna Turinga jest zdefiniowana przez:

- zbiór stanów $Q = \{q_0\}$
- zbiór symboli wejściowych $\Sigma = \{0, 1\}$
- stan początkowy q_0
- symbol pusty $B = \{\#\}$
- funkcję przejścia:

  $q_0, 0, q_0, 1, \leftarrow$ 0 zamień na 1

  $q_0, 1, q_0, 0, \leftarrow$ 1 zamień na 0
- tablica sterująca

δ	0	1	#
q_0	$q_0, 1, \leftarrow$	$q_0, 0, \leftarrow$	Stan nieokreślony, koniec programu

- taśma zawiera ciąg symboli: ... # 1 0 1 1 **0** # ...
- głowica ustawiona jest na pierwszym od prawej symbolu 0

Przykład 2.

Taśma z głowicą	Odczytany znak	Stan bieżący	Wykonywana operacja
# 1 0 1 1 0 #	0	q_0	Odczytany znak i stan q_0 wyznacza instrukcję $q_0,0,q_0, 1, \leftarrow$. Za bieżący znak maszyna Turinga umieści symbol 1, stanu nie zmieni (pozostanie w q_0) i przemieści głowicę do komórki po lewej stronie.
# 1 0 1 1 1 **#**	1	q_0	Instrukcja $q_0,1,q_0, 0, \leftarrow$
# 1 0 1 0 1 **#**	1	q_0	Instrukcja $q_0,1,q_0, 0, \leftarrow$
# 1 0 0 0 1 **#**	0	q_0	Instrukcja $q_0,0, q_0, 1, \leftarrow$
# 1 1 0 0 1 **#**	1	q_0	Instrukcja $q_0,1,q_0, 0, \leftarrow$
# 0 1 0 0 1 **#**	#	q_0	Instrukcja niezdefiniowana, program się zakończy

Przykład 2.

Stan taśmy przed i po wykonaniu programu

...	#	1	0	1	1	0	#	...
...	#	0	1	0	0	1	#	...

Program dokonał binarnej negacji bitów wejściowych.

Przykład 3.

Maszyna Turinga jest zdefiniowana przez:

- zbiór stanów $Q = \{q_0, q_1, q_2\}$
- zbiór symboli wejściowych $\Sigma = \{0, 1\}$
- stan początkowy q_0
- stan końcowy $F = \{q_2\}$
- symbol pusty #
- funkcję przejścia:

$q_0, 0, q_0, 0, \leftarrow$
$q_0, 1, q_1, 0, \leftarrow$
$q_1, 0, q_0, 1, \leftarrow$
$q_1, 1, q_1, 1, \leftarrow$
$q_0, \#, q_2, 0, \leftarrow$
$q_1, \#, q_2, 1, \leftarrow$

δ	0	1	#
q_0	$q_0, 0, \leftarrow$	$q_1, 0, \leftarrow$	$q_2, 0, \leftarrow$
q_1	$q_0, 1, \leftarrow$	$q_1, 1, \leftarrow$	$q_2, 1, \leftarrow$

Taśma zawiera następujący ciąg symboli … # 1 1 0 1 0 1 # …

Przykład 3.

Taśma z głowicą	Odczytany znak	Stan bieżący	Wykonywana operacja	Zapisany znak	Stan nowy
# # 1 1 0 1 0 **1** #	1	q_0	q_0, 1, q_1, 0, ←	0	q_1
# # 1 1 0 1 **0** 0 #	0	q_1	q_1, 0, q_0, 1, ←	1	q_0
# # 1 1 0 **1** 1 0 #	1	q_0	q_0, 1, q_1, 0, ←	0	q_1
# # 1 1 **0** 0 1 0 #	0	q_1	q_1, 0, q_0, 1, ←	1	q_0
# # 1 **1** 1 0 1 0 #	1	q_0	q_0, 1, q_1, 0, ←	0	q_1
# # **1** 0 1 0 1 0 #	1	q_1	q_1, 1, q_1, 1, ←	1	q_1
# **#** 1 0 1 0 1 0 #	#	q_1	q_1, #, q_2, 1, ←	1	q_2
# 1 1 0 1 0 1 0 #	#	q_2	Instrukcja niezdefiniowana, program się zakończy		

Przykład 3.

Stan taśmy przed i po wykonaniu programu

...	...	#	1	1	0	1	0	1	#	...
...	#	1	1	0	1	0	1	0	#	...

Program dokonał przesunięcia o 1 pozycję w lewo.

Przykłady zastosowań:

- inkrementacja liczby binarnej,
- dekrementacja liczby binarnej,
- sumowanie liczb binarnych.

Architektura von Neumanna

Rok

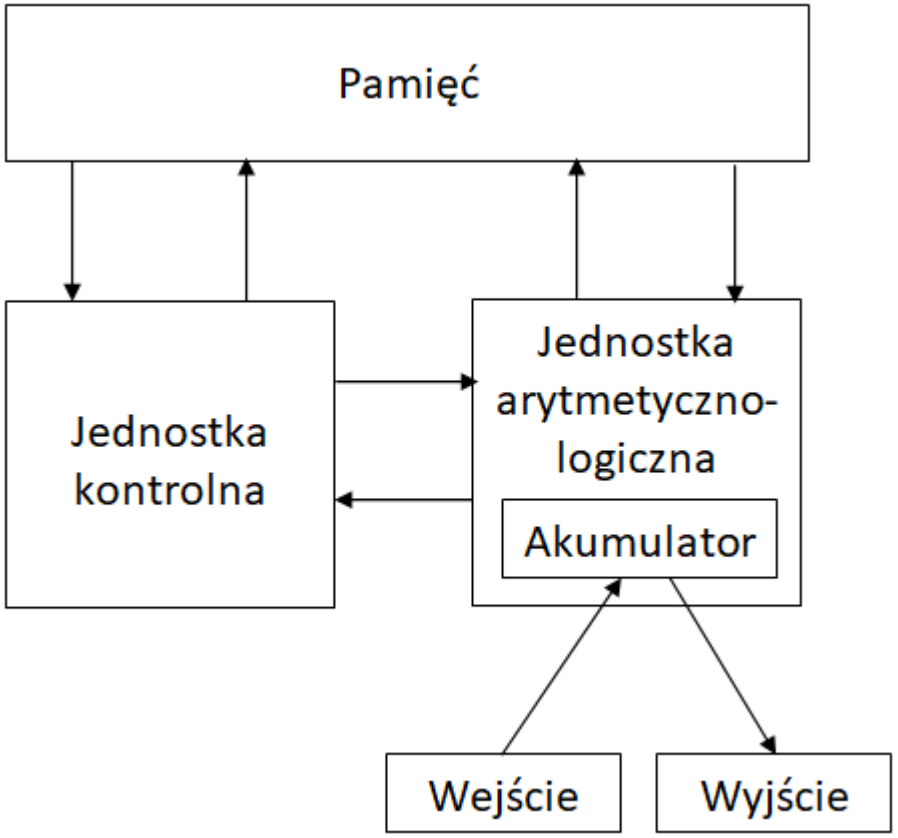
- 1945

Autorzy

- John von Neumann
- John Wiliam Mauchly
- John Prosper Eckert

Architektura von Neumanna

- opracowanie pierwszej powszechnie wykorzystywanej i zrealizowanej architektury komputera,
- forma sposobu organizacji elementów składowych komputera.



Rys. 1. Architektura von Neumanna

Architektura von Neumanna

Zasadnicze składowe komputera

- jednostka arytmetyczno-logiczna, wykonująca podstawowe operacje przetwarzania informacji,
- pamięć komputera, przechowująca dane i instrukcje do wykonania,
- jednostka kontrolna, odpowiadająca za przesyłanie informacji między pamięcią, a jednostką arytmetyczno-logiczną i zapewniająca jej poprawne działanie,
- urządzenia wejściowe i wyjściowe.

Założenia

- każdy element pamięci ma unikalny identyfikator nazywany adresem, pamięć jest jednolita, implikuje przechowywanie danych wraz z instrukcjami (w odróżnieniu od architektury harwardzkiej, gdzie występuje osobna pamięć dla danych i dla instrukcji),
- jednostka kontrolna wraz z jednostką arytmetyczno-logiczną tworzą moduł zwany procesorem,
- najważniejszą częścią jednostki arytmetyczno-logicznej jest akumulator.

Architektura von Neumanna

Wykonywanie programów opiera się na:

- skończonej i kompletnej liście rozkazów,
- reprezentacji danych i rozkazów na skończonej liczbie bitów, ich rozróżnienie wynika jedynie z odmiennej reprezentacji,
- bezpośredniej i jednakowej dostępności całej pamięci dla procesora przed wykonaniem i w trakcie wykonywania programu,
- przetwarzaniu informacji przez sekwencyjny odczyt kolejnych instrukcji z pamięci i ich wykonywaniu przez procesor,
- działaniu polegającym na pobraniu z pamięci pojedynczego rozkazu, dekodowaniu rozkazu, obliczeniu ostatecznej wartości operanda i wykonaniu rozkazu.

Architektura von Neumanna

Architektura von Neumanna

- niemal każdy komputer jest komputerem o architekturze von Neumanna,
- sama architektura to ogólna idea , która po określeniu liczby bitów dla komórki pamięci i procesora oraz listy rozkazów pozwala na zdefiniowanie wirtualnego lub rzeczywistego modelu maszyny liczącej,
- poszczególne modele maszyny cyfrowej różnią się liczbą i nazewnictwem oraz strukturą rejestrów procesora.

Budowa maszyny cyfrowej

Przykładowa maszyna cyfrowa

- pamięć MEM – 512 elementów 16-bitowych, adresowanych liczbami od 0 do 511,
- procesor złożony z jednostki arytmetyczno-logicznej ALU i 16-bitowych rejestrów:
  - akumulatora AC (*acumulator*) – współdziała z ALU, przechowuje argumenty i odbiera wyniki,
  - rejestru operanda OR (*operand register*) – służy do wyliczania i przechowania ostatecznego argumentu dla ALU,
  - licznika rozkazów PC (*program counter*) – służy do wskazywania w pamięci miejsca, gdzie znajduje się instrukcja wykonania, zwiększa się automatycznie o 1 w każdym cyklu pracy procesora,
  - rejestru instrukcji IR (*instruction register*) – przechowuje aktualnie wykonywaną instrukcję po skopiowaniu jej z pamięci.

Budowa maszyny cyfrowej

Interpretacja 16-bitowego słowa maszynowego

- **dane** – z przewidzianymi trzema typami: całkowitoliczbowym, (kodowanie uzupełnieniowe), znakowym (kodowanie ASCII), logicznym (0 – fałsz, nie zero – prawda),
- **rozkazy** – bit pierwszy od lewej nieużywany, następne 4 bity to kod rozkazu (*code*), następne 2 bity to sposób wyliczania operanda *mode* (tryb adresowania), pozostałe 9 bitów to zasadniczy argument rozkazu *arg* (najczęściej adres).

Działanie maszyny cyfrowej

Rozkazy

- nadzorujące przebieg pracy: STOP (zatrzymanie działania),
- przesyłania: LOAD (AC ← OR), STORE (AC → MEM[OR]),
- sterujące kolejnością wykonywania rozkazów: skok JUMP (PC ← OR), JNEG (jeśli AC < 0), JZERO (jeśli AC = 0),
- arytmetyczne: AC ← AC □ OR, gdzie symbol □ oznacza jedną z operacji ADD (+), SUB (–), MULT (*), DIV (dzielenie całkowite /),
- logiczne: AC ← AC □ OR, gdzie symbol □ oznacza AND (&) lub OR (|); NOT (AC ← ~OR), tzn. odwrócenie wartości każdego bitu,
- bitowych porównań: CMP, wstawia -1 do AC, jeżeli AC = OR i zeruje AC w przeciwnym razie,
- przesunięć bitów akumulatora: SHZ – przesunięcie o liczbę miejsc równą wartości bezwzględnej rejestru operanda, w prawo dla wartości ujemnej, w lewo dla dodatniej; zwalniane miejsca uzupełnia zerami;
  SHC – przesunięcie cykliczne zawartości akumulatora.

Działanie maszyny cyfrowej

Tryby adresowania

- natychmiastowy (bezpośrednie obliczenia) – za operand przyjmuje pole arg rejestru instrukcji,
- bezpośredni (operacje na zmiennych) – za operand przyjmuje zawartość komórki pamięci, której adres pamiętany jest w polu arg rejestru instrukcji,
- pośredni – za operand przyjmuje zawartość komórki pamięci, której adres pamiętany jest w komórce o adresie przechowywanym w polu arg rejestru instrukcji ,
- indeksowy – za operand przyjmuje zawartość komórki pamięci, której adres pamiętany jest w akumulatorze, z przesunięciem o wartość pola arg rejestru instrukcji.

Automaty i języki formalne

Podstawowe definicje

- **Alfabet** – oznaczamy symbolem *V* lub Σ, jest to skończony niepusty zbiór rozróżnialnych symboli (liter) z ustalonym porządkiem.
- **Słowo** (napis) nad danym alfabetem *V* – każdy skończony ciąg złożony z elementów alfabetu *V* oraz słowo puste. Napis pusty to napis, który nie zawiera symboli.
  - Ciąg należy rozumieć w jego formalnym matematycznym znaczeniu, jako funkcję ze zbioru liczb naturalnych w zbiór *V.*
  - Dopuszczamy istnienie słowa, które nie zawiera żadnego elementu alfabetu, nazywanego **słowem pustym,** oznaczanego najczęściej przez ε lub λ.
- **Słownik** *V** – zbiór wszystkich słów nad alfabetem *V* wraz ze słowem pustym. Zbiór wszystkich słów nad alfabetem *V* jest nieskończony i przeliczalny. Przez V^+ oznaczać będziemy zbiór *V** – $\{\varepsilon\}$.
- **Język *L*** (nad danym alfabetem *V*) – dowolny podzbiór słownika *V**. Język jest językiem skończonym wtedy i tylko wtedy gdy istnieje ograniczenie długości słów języka.

Przykłady alfabetów

Przykłady alfabetów:

$$V_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, w, x, y, z\}$$

$$V_2 = \{a, ą, b, c, ć, d, e, ę, f, g, h, i, j, k, l, ł, m, n, o, ó, p, q, r, s, ś, t, u, w, x, y, z, ź, ż\}$$

$$V_3 = \{\alpha, \beta, \gamma, \delta, \varepsilon, \epsilon, \zeta, \eta, \theta, \vartheta, \iota, \kappa, \lambda, \mu, \nu, \xi, o, \pi, \rho, \varrho, \tau, \upsilon, \varphi, \chi, \psi, \omega\}$$

$$V_4 = \{0,1\}$$

$$V_5 = \{a, \ldots, z, A, \ldots, Z, 0, \ldots, 9\}$$

Słowem (napisem) nad danym alfabetem V nazywamy funkcję u: {1,2,…,n} → V .

Przykłady słów nad alfabetem V_1: *aab*, *aababbcca*, *alakotama*

Dla pierwszego słowa otrzymujemy:
$u(1) = a$, $u(2) = a$, $u(3) = b$.

Podstawowe operacje

Konkatenacja (sklejenie) słów – działanie na słowniku polegające na łączeniu słów będących argumentami w jedno słowo.

Definicja 1.
Konkatenacją słów u: {1,2,…,n} → V , v: {1,2,…,m} → V nazywamy funkcję f: {1,2,…,$n+m$} → V określoną jako:

$$f(i) = \begin{cases} u(i) & \text{dla} \quad i > 0 \ \wedge \ i \leq n \\ v(i-n) & \text{dla} \quad i > n \ \wedge \ i \leq n+m \end{cases}$$

- Konkatenacja jest łączna i nie jest przemienna, elementem neutralnym jest słowo puste.
- Długość słowa $v = (v_1, \ldots, v_n) = v_1 \ldots v_n \in V^*$ jest oznaczana $|v|$ i wynosi $|v| = n$. Długość słowa pustego jest równa 0.
- Jeżeli u i v są napisami, to konkatenacją u i v jest napis uv, ponadto $|uv| = |u| + |v|$. Jeżeli uwv jest napisem, to u jest przedrostkiem, w podnapisem, natomiast v jest końcówką.

Podstawowe operacje

Długość słowa określamy indukcyjnie:

(1) $|\varepsilon| = 0$,

(2) $|v_1 v_2 \ldots v_{n+1}| = |v_1 v_2 \ldots v_n| + 1$, gdzie $v_i \in V$, $i = 1, 2, \ldots, n+1$.

Opierając się na operacji konkatenacji określamy operację n-tej potęgi słowa dla $i = 0, 1, \ldots$. Niech $v \in V^*$. Wówczas:

$v^0 = \varepsilon$

$v^n = vv^{n-1}$

Jak stąd widać $v^n = v \ldots v$ (n razy).

Przykład 1. $u = aab$, $u^3 = aabaabaab$

Własności:

$|u^n| = n|u|$

$u^{n+m} = u^n u^m$

Podstawowe operacje

Odbicie lustrzane (rewers R) słowa u: $\{1,2,\ldots,n\} \rightarrow V$ definiowane jest jako funkcja g: $\{1,2,\ldots,n\} \rightarrow V$ taka, że

$g(i) = v(n - i)$

Przykład 2. $u = abc$, $g(u) = cba$

Własności:

$R(uv) = R(v)R(u)$

$R(u^n) = (R(u))^n$

Możemy teraz podać formalną definicję zbioru V^*.
Zbiór V^* określamy indukcyjnie:

(1) ε jest elementem zbioru V^*,

(2) jeżeli $v \in V^*$ oraz $v_i \in V$, to $vv_i \in V^*$ i $v_i v \in V^*$,

(3) żadne inne elementy do zbioru V^* nie należą.

Przykłady języków

Język *L* nad alfabetem *V* jest zbiorem napisów nad *V*:

- zbiór palindromów nad alfabetem małych liter łacińskich,
- zbiór liczb binarnych bez nieznaczących zer nad alfabetem *V* = {0,1},
- zbiór liczb pierwszych zapisanych w układzie pozycyjnym dziesiętnym nad alfabetem cyfr arabskich *V* ={0,1,2,3,4,5,6,7,8,9},
- język obejmujący wszystkie niepuste napisy nad alfabetem $V_3 = \{a, b, c\}$, znak … informuje o nieskończonej kontynuacji,
  $L_3 = V_3^+ = \{ a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, \dots \}$
- zbiór napisów obejmujących dwa fragmenty z równą liczbą znaków *a* oraz *c*, z umieszczonym między nimi pojedynczym znakiem *b*:
  $L = \{a^n b c^n : n = 1,2,3, \dots \}$

Operacje na językach

Niech L_1 i L_2 będą dwoma językami. Język L_1L_2 nazywany konkatenacją L_1 i L_2 to zbiór postaci $\{uv: u \in L_1 \text{ i } v \in L_2\}$.
Niech teraz L będzie językiem. Definiujemy:

$$L^0 = \{\varepsilon\}$$

$$L^i = LL^{i-1} \text{ dla } i \geq 1$$

$$L^* = \bigcup_{i=0}^{\infty} L^i$$

$$L^+ = \bigcup_{i=1}^{\infty} L^i$$

Język L^* nazywamy domknięciem Kleene'go L, język L^+ nazywamy domknięciem dodatnim L.

Operacje na językach

(1) Suma języków $L_1, L_2 \in V^*$

$L = L_1 \cup L_2 = \{u \in V^* : u \in L_1 \text{ lub } u \in L_2\}$

(2) Część wspólna $L_1, L_2 \in V^*$

$L = L_1 \cap L_2 = \{u \in V^* : u \in L_1 \text{ i } u \in L_2\}$

(3) Dopełnienie L

$\overline{L} = \{u \in V^* : u \notin L\}$

Przykład 3.

$V = \{a,b\}$

$L = \{a,b,aa,ab,ba,bb\}$

$V^* = \{\varepsilon, a, b, aa, bb, ab, ba, aaa, bbb, \ldots\}$

$\overline{L} = V^* - L = \{\varepsilon, aaa, bbb, \ldots\}$

$L_1 = \{u \in \{a\}^+ : |u| \text{ jest parzysta}\}$

$L_2 = \{u \in \{a\}^+ : |u| \text{ jest podzielna przez } 3\}$

$L_1 \cap L_2 = \{a^{6k} \text{ dla } k > 0\}$

Wyrażenia i języki regularne

Definicja 2.

Wyrażenie regularne nad alfabetem *V* definiujemy rekurencyjnie jako wyrażenie otrzymane przez zastosowanie skończonej liczby następujących reguł:

(1) $\emptyset$ jest wyrażeniem regularnym i opisuje zbiór pusty,

(2) ε jest wyrażeniem regularnym i opisuje zbiór $\{\varepsilon\}$,

(3) dla każdego $a \in V$, a jest wyrażeniem regularnym i opisuje zbiór $\{a\}$,

(4) jeżeli *r* i *s* są wyrażeniami regularnymi, to wyrażenia otrzymane przez zastosowanie następujących operacji do wyrażeń regularnych są wyrażeniami regularnymi:

(r+s) – operacja sumy,

(rs) – operacja konkatenacji (łączenia),

(r*) – operacja Kleene'ego.

Wyrażenia i języki regularne

Wyrażenia regularne definiują odpowiadające im języki:

(1) $\emptyset$ wyrażenie regularne definiuje język pusty $\emptyset$,

(2) wyrażenie regularne ε definiuje język złożony ze słowa pustego $\{\varepsilon\}$,

(3) dla każdego $a \in V$, wyrażenie regularne a definiuje język złożony z jednoliterowego słowa a: $\{a\}$,

(4) jeżeli wyrażenia regularne r i s definiują odpowiednio języki R i S, to wyrażenia regularne (r+s), (rs), (r*) definiują odpowiednio języki $(R \cup S)$, (RS), (R^*) .

Przykład 4.

$(\{\varepsilon\} \cup \{a\}) = \{\varepsilon, a\}$

$((\{a\}\ \{a\})(\{b\}\ \{a\})) = \{aaba\}$

$(((\{a\}\ \{a\})\{b\})(\{b\} \cup \{a\})) = \{aabb, aaba\}$

$(\{a\} \cup (\{a\}((\{b\}\{b\})^*))) = \{a, abb, abbbb, abbbbbb, \ldots\}$

W celu uproszczenia zapisu poszczególny operacjom na wyrażeniach regularnych nadawane są priorytety, wg kolejności: operacja sumy, konkatenacji, Kleene'ego. Wtedy można pominąć część nawiasów.

Wyrażenia i języki regularne

Definicja 3.
Językami regularnymi nazywamy języki generowane przez wyrażenia regularne.

Przykład 5.
Przykładem języka regularnego jest zbiór L liczb dziesiętnych bez nieznaczących zer: L = {0,1,23,4,5,6,7,8,9,10,11,…}

Wyrażeniem regularnym definiującym ten język jest:
0+(1+2+3+4+5+6+7+8+9)(0+1+2+3+4+5+6+7+8+9)*

Lemat Myhilla-Nerode'a
Język L jest językiem regularnym wtedy i tylko wtedy, gdy relacja R_L indukowana przez język L jest relacją o indeksie skończonym (tzn. ma skończoną klasę abstrakcji).

Lemat o pompowaniu
Jeżeli L jest językiem regularnym, to istnieje taka stała n_L, że dla dowolnego słowa $z \in L$ spełniony jest warunek:

$$(|z| \geq n_L) \Longrightarrow [(\exists_{u,v,w}\ z = uvw \wedge |uv| \leq n_L \wedge |v| \geq 1) \forall_{i=0,1,2,\ldots}\ uv^i w \in L]$$

Wyrażenia i języki regularne

Wnioski:

- Języki regularne są ze swej natury językami skończonymi.
- Długość słów języka regularnego jest ograniczona przez stała n_L.
- Jeżeli język regularny zawiera słowa dłuższe niż stała n_L, to będzie językiem nieskończonym i tym samym długość słów tego języka nie będzie ograniczona.
- Lemat o pompowaniu pozwala jedynie na próbę odgadnięcia, że język jest językiem regularnym.
- W zastosowaniach praktycznych zaprzeczenie lematu o pompowaniu będzie wykorzystywane do wskazania, że dany język nie jest językiem regularnym.

Lemat 1. Zaprzeczenie lematu o pompowaniu

Jeżeli dla dowolnej stałej $n \in N$ istnieje $z \in L$ takie, że:

$$(|z| \geq n) \wedge [(\forall_{u,v,w}\, z = uvw \wedge |uv| \leq n \wedge |v| \geq 1) \exists_{i=0,1,2,\ldots}\, uv^i w \notin L]$$

to język L nie jest regularny.

Automaty skończone

Definicja 4.

Deterministycznym automatem skończonym nazywamy uporządkowaną piątkę

$M = (Q, \Sigma, \delta, q_0, F)$,

gdzie:

- Q jest skończonym zbiorem stanów,
- Σ jest skończonym alfabetem wejściowym,
- q_0 jest wyróżnionym stanem w Q, zwanym stanem początkowym,
- F jest wyróżnionym podzbiorem Q, zwanym zbiorem stanów akceptujących (końcowych),
- $\delta: Q \times \Sigma \rightarrow Q$ jest funkcją przejścia.

Wartościami funkcji przejścia automatu są pojedyncze stany ze zbioru stanów Q.

Automaty skończone

Automat skończony deterministyczny przed rozpoczęciem obliczeń znajduje się w konfiguracji początkowej:

- sterowanie jest w stanie początkowym q_0,
- głowica jest ustawiona na pierwszą literę słowa wejściowego.

Obliczenie automatu polega na wykonaniu kolejnych kroków określonych wartościami funkcji przejścia na podstawie symbolu czytanego przez głowicę i stanu, w którym jest sterowanie następuje:

- przejście sterowania do pewnego stanu q z Q,
- przesunięcie głowicy o jedną komórkę w prawo.

Automat będzie wykonywał kolejne kroki aż do wczytania wszystkich symboli wejściowych. Wtedy nastąpi zatrzymanie automatu. Automat zaakceptuje dane wejściowe wtedy i tylko wtedy, gdy po zakończeniu obliczenia automat wejdzie w stan akceptujący.

Automaty skończone

Definicja 5.

Językiem akceptowalnym przez automat skończony deterministyczny nazywamy zbiór słów nad alfabetem Σ, dla których automat kończy obliczenie w stanie akceptującym.

Definicja 6.

Opisem chwilowym automatu skończonego deterministycznego nazywamy ciąg symboli $q\alpha$, gdzie q to bieżący stan automatu należący do Q, natomiast α to ciąg symboli nie wczytanych przez automat.

Ponieważ w każdym kroku automatu głowica przesuwa się o jeden symbol wejściowy w prawo, dla słowa wejściowego $w = a_1a_2...a_n$ będziemy stosować uproszczony zapis obliczenia automatu w postaci ciągu:

$$q_0\, a_1\, q_1\, a_2\, q_2 \ldots a_n\, q_n\,,$$

gdzie q_0 jest stanem początkowym, $q_1, q_2, \ldots, q_n$ są stanami do których przechodzi sterowanie automatu po wczytaniu kolejnych symboli wejściowych $a_1, a_2, \ldots, a_n$.

Funkcja przejścia automatu skończonego jest przedstawiana w postaci tabeli albo w postaci grafu. Prezentuje ona pełny opis automatu.

Automaty skończone

Przykład 6.

Skonstruować automat skończony deterministyczny akceptujący liczby binarne bez nieznaczących zer.

Liczby binarne bez nieznaczących zer to ciągi zaczynające się od jedynki i liczba zero. Obliczenie automatu polega na sprawdzeniu pierwszego symbolu wejściowego:

- jeżeli pierwszym symbolem było 0, to musi to być jedyny symbol wejściowy,
- jeżeli pierwszym symbolem byłą 1, to niezależnie od pozostałych symboli słowo zostanie zaakceptowane.

Automaty skończone

Funkcję przejścia określa tabela:

δ	0	1
$\rightarrow q_0$	q_1	q_3
$q_1 \rightarrow$	q_2	q_2
q_2	q_2	q_2
$q_3 \rightarrow$	q_3	q_3

Przykład:
Ciąg wejściowy 10010
$q_0\ 1\ q_3\ 0\ q_3\ 0\ q_3\ 1\ q_3\ 0\ \boldsymbol{q_3}$
akceptacja

Ciąg wejściowy 00101
$q_0\ 0\ q_1\ 0\ q_2\ 1\ q_2\ 0\ q_2\ 1\ \boldsymbol{q_2}$
brak akceptacji

Symbol strzałki → umieszczony przed stanem q_0 oznacza stan wejściowy.

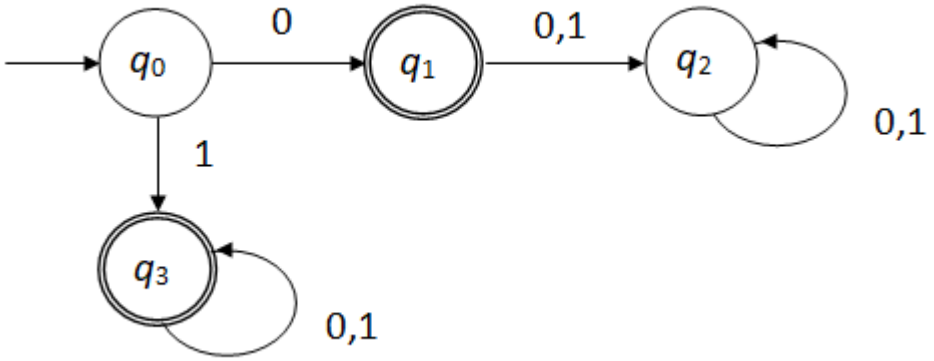
Symbol strzałki → umieszczony po stanie q_i dla i = 1, 3, oznacza stan akceptujący.

Podana funkcja przejścia opisuje automat:

$M = (\{q_0, q_1, q_2, q_3\}, \{0,1\}, \delta, q_0, \{q_1, q_3\})$

Automaty skończone

Alternatywnym sposobem przedstawienia automatu skończonego deterministycznego jest graf automatu:



Graf konstruujemy według następujących reguł:

- wierzchołki reprezentują możliwe stany automatu,
- wychodzące z nich krawędzie są etykietowane symbolami wejścia, tzn. krawędź etykietowana symbolem x od wierzchołka q_i do wierzchołka q_j istnieje wtedy i tylko wtedy, gdy $\delta(q_i,x) = q_j$,
- wierzchołki akceptujące są wyróżnione podwójną linią.

Automaty skończone

Definicja 7.

Domknięciem funkcji przejścia automatu skończonego deterministycznego nazywamy funkcję:

$\hat{\delta}: Q \times \Sigma^* \rightarrow Q$

taką, że:

(1) $(\forall q \in Q)\ \hat{\delta}(q, \varepsilon) = q,$

(2) $(\forall q \in Q)(\forall a \in \Sigma)(\forall w \in \Sigma^*)\ \hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a).$

Definicja ta określa rozszerzenie dziedziny funkcji przejścia z $Q \times \Sigma$ na $Q \times \Sigma^*$. Funkcja przejścia δ określa ruch automatu dla symbolu wejściowego, podczas gdy uogólnienie funkcji przejścia $\hat{\delta}$ określa zachowanie automatu dla słowa nad alfabetem wejściowym Σ.

Automaty skończone

Definicja 8.

Automatem skończonym niedeterministycznym nazywamy uporządkowaną piątkę

$M = (Q, \Sigma, \delta, q_0, F)$,

gdzie:

- Q jest skończonym zbiorem stanów,
- Σ jest skończonym alfabetem wejściowym,
- q_0 jest wyróżnionym stanem w Q, zwanym stanem początkowym,
- F jest wyróżnionym podzbiorem Q, zwanym zbiorem stanów akceptujących (końcowych),
- $\delta: Q \times \Sigma \rightarrow 2^Q$ jest funkcją przejścia.

Automaty skończone

Funkcja przejścia automatu skończonego niedeterministycznego jest określona dla wszystkich argumentów. Jej wartościami są podzbiory zbioru stanów Q wraz ze zbiorem pustym.

Automat skończony niedeterministyczny przed rozpoczęciem obliczeń znajduje się w następującej konfiguracji początkowej:

- sterowanie jest w stanie początkowym q_0,
- głowica jest ustawiona na pierwszą literę słowa wejściowego.

Obliczenie automatu polega na wykonaniu kolejnych kroków określonych wartościami funkcji przejścia δ na podstawie symbolu czytanego przez głowicę i stanu, w którym jest sterowanie. Następuje:

- przejście sterowania do pewnego stanu q z Q należącego do podzbioru stanów będącego wartością funkcji przejścia; wybór stanu dokonywany jest niedeterministycznie, dla zbioru pustego obliczanie zostaje zakończone bez wczytania całego słowa, co oznacza brak akceptacji wejścia,
- przesunięcie głowicy o jedną komórkę w prawo.

Automaty skończone

Definicja 9.

Obliczeniem niedeterministycznego automatu skończonego dla słowa wejściowego $w = a_1 a_2 \ldots a_n$ jest drzewo, którego:

- wierzchołki są etykietowane opisami chwilowymi automatu,
- korzeń jest etykietowany opisem chwilowym konfiguracji początkowej,
- potomkami dowolnego wierzchołka na wysokości k-1 są takie wierzchołki na wysokości k, że etykiety wierzchołka i jego potomka pozostają w relacji ruchu automatu dla symbolu wejściowego a_k ,
- liść drzewa obliczenia nazwiemy akceptującym, gdy jest on wierzchołkiem na wysokości n, a stan opisu chwilowego automatu jest stanem akceptującym.

Definicja 10.

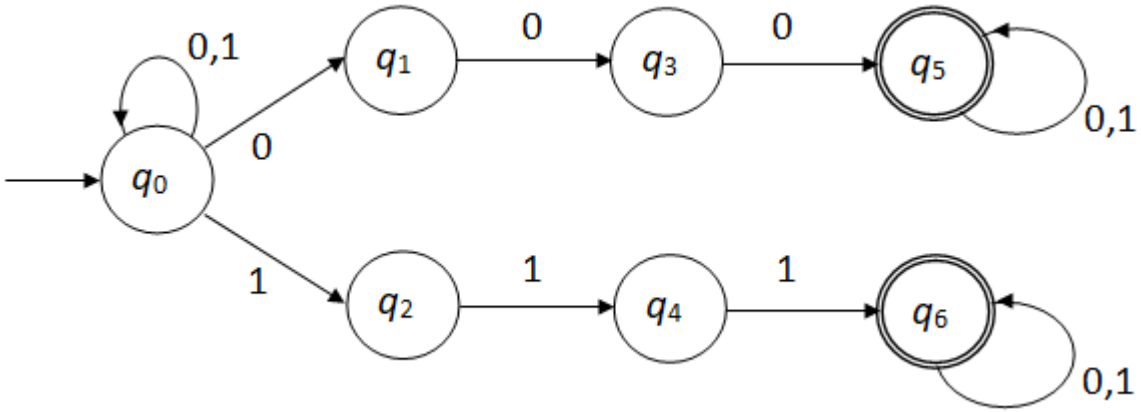
Językiem akceptowalnym przez automat niedeterministyczny nazywamy zbiór słów nad alfabetem Σ, dla których obliczenie automatu zawiera ścieżkę prowadzącą od korzenia do liścia akceptującego.

Automaty skończone

Przykład 7.

Skonstruować automat skończony deterministyczny akceptujący język złożony ze słów binarnych zawierających ciąg trzech 0 lub z trzech 1.

Graf automatu:



W stanie q_0 automat wczytuje symbole wejściowe aż do napotkania początku sekwencji trzech zer lub trzech jedynek, po osiągnięciu stanu q_5 lub q_6 automat pozostaje w tym stanie aż do zakończenia obliczenia.

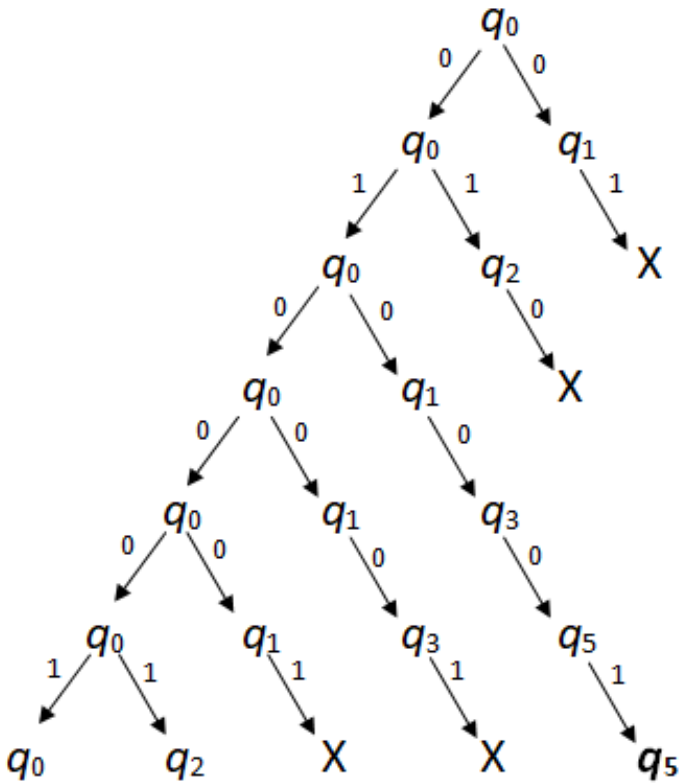
Automaty skończone

Tabela funkcji przejścia:

δ	0	1
$\rightarrow q_0$	$\{q_0, q_1\}$	$\{q_0, q_2\}$
q_1	$\{q_3\}$	0
q_2	0	$\{q_4\}$
q_3	$\{q_5\}$	0
q_4	0	$\{q_6\}$
$q_5 \rightarrow$	$\{q_5\}$	$\{q_5\}$
$q_6 \rightarrow$	$\{q_6\}$	$\{q_6\}$

Automaty skończone

Drzewo obliczania automatu dla słowa 010001:



Ścieżka

$q_0\ 0\ q_0\ 1\ q_0\ 0\ q_1\ 0\ q_3\ 0\ q_5\ 1\ \boldsymbol{q_5}$

prowadzi od korzenia do stanu akceptującego.

Tak skonstruowany automat nie jest automatem z minimalną liczbą stanów.

Można przeprowadzić operację minimalizacji stanów.

Automaty skończone

Zadanie 1.

Skonstruować automaty deterministyczny i niedeterministyczny akceptujące ciągi binarne kończące się trzema jedynkami.

Okazuje się, że klasy automatów deterministycznych i niedeterministycznych są równoważne ze względu na akceptowalne języki.

Twierdzenie 1.

Język *L* jest akceptowalny przez pewien automat niedeterministyczny wtedy i tylko wtedy, gdy jest akceptowalny przez pewien automat deterministyczny.

Gramatyka formalna

Formalną definicję gramatyki zawdzięczamy urodzonemu w 1928 roku Noamowi Chomsky'emu, wieloletniemu profesorowi MIT. Teorie Chomsky'ego okazały się fundamentalne w dziedzinie języków programowania.

Gramatykę formalną możemy zdefiniować przez czwórkę

$G = (N, T, P, S)$,

gdzie:

- *N to n*iepusty, skończony, uporządkowany zbiór elementów terminalnych,
- *T* to niepusty, skończony, uporządkowany zbiór elementów terminalnych, dla uniknięcia niejednoznaczności zakładamy rozłączność zbioru nieterminali i terminali; zbiór $V = N \cup T$ nazywamy alfabetem ogólnym gramatyki.
- *P* to skończony, niepusty zbiór produkcji – reguł zastępowania jednych napisów innymi; każda produkcja to para złożona w pierwszym elemencie z napisu zastanego przed pojedynczym krokiem generacji, a w drugim z napisu wynikowego pojedynczego kroku generacji:

$$P \subset V^* \times V^* = \{(\alpha, \beta) : \alpha, \beta \in V^*\}, \qquad \alpha \mapsto \beta \in P$$

- *S* to nieterminal startowy odpowiedzialny za najbardziej ogólną kategorię generowanych napisów i za inicjowanie procesu generacji.

Gramatyka formalna

Zastępowanie jednych napisów innymi zgodnie z produkcjami nosi nazwę wywodu:

Przy danej gramatyce $G = (N, T, P, S)$ mówimy, że z napisu $\alpha \in V^*$ **wywodzi się bezpośrednio** napis $\beta \in V^*$, co oznaczamy:

$$\alpha \Rightarrow \beta$$

wtedy i tylko wtedy, gdy:

$$\exists \lambda, \rho \in V^*, \exists \gamma \mapsto \eta \in P : \alpha = \lambda\gamma\rho, \beta = \lambda\eta\rho$$

Wywód złożony z kolejnych kroków nazywany jest wywodem pośrednim:

Przy danej gramatyce $G = (N, T, P, S)$ mówimy, że z napisu $\alpha \in V^*$ **wywodzi się pośrednio** napis $\beta \in V^*$, co oznaczamy:

$$\alpha \overset{*}{\Rightarrow} \beta$$

wtedy i tylko wtedy, gdy

$$\exists \gamma_0, \ldots, \gamma_n \in V^* : \alpha \Rightarrow \gamma_0 \Rightarrow \gamma_1 \Rightarrow \gamma_n = \beta$$

Gramatyka formalna

Definicja 11.

Przy danej gramatyce $G = (N, T, P, S)$ zbiór:

$$L(G) = \left\{ w \in T^* : S \overset{*}{\Rightarrow} w \right\}$$

nazywamy **językiem generowanym przez gramatykę** G.

- Wywód bezpośredni oznacza równość napisów początkowego i końcowego, z wyjątkiem odpowiednio zawartych w nich podciągów stanowiących poprzednik i następnik produkcji.
- Interesuje nas proces generacji od nieterminalnego symbolu startowego do napisu złożonego wyłącznie z terminali.
- Gdyby za alfabet uznać alfabet terminalny gramatyki i rozważyć wszystkie napisy możliwe do wywiedzenia z nieterminala startowego, to otrzymany zbiór napisów jest językiem nad alfabetem terminalnym, a ponadto wygenerowane napisy podlegają regułom gramatycznym ujętym w produkcje.
- Dochodzimy więc do pojęcia **gramatyka formalna.** Ten sam język może być generowany przez wiele gramatyk, są to tzw. gramatyki równoważne.

Hierarchia Chomsky'ego

Hierarchia Chomskie'go obejmuje:

1. **Gramatyki klasy 0** (tzw. **gramatyki rekurencyjnie przeliczalne** lub **kombinatoryczne)** są najogólniejszą klasą gramatyk. Produkcje mają najbardziej ogólną postać:

   $P \subset V^* \times V^*$

   i nie podlegają żadnym dodatkowym warunkom w stosunku do wyjściowej definicji gramatyki.

2. **Gramatyki klasy 1 (gramatyki kontekstowe)** zawierają się w gramatykach klasy 0, w literaturze przedmiotu można znaleźć dwie równoważne definicje:

   - gramatyka kontekstowa to gramatyka której produkcje spełniają warunek:

     $P \subset V^* N V^* \times V^+$

     przy czym $\alpha, \beta \in V^*, A \in N, \gamma \in V^*$

   - gramatyka kontekstowa to gramatyka o produkcjach postaci:

     $\alpha \mapsto \beta$

     ale tylko wtedy i tylko wtedy, gdy napis α zawiera nie mniej niż jeden symbol nieterminalny, a ponadto $|\alpha| \leq |\beta|$.

Hierarchia Chomsky'ego

3. **Gramatyki klasy 2** (**gramatyki bezkontekstowe)** zawierają się w gramatykach kontekstowych, warunkiem narzuconym na każdą produkcję jest to, by poprzednik był pojedynczym symbolem nieterminalnym :

   $$P \subset N \times V^{+}$$

   Warunek oznacza, że każda produkcja musi być w postaci:

   $$N \ni \mathrm{N} \mapsto \alpha \in V^{*}$$

4. **Gramatyki klasy 3 (gramatyki regularne)** zawierają się w gramatykach bezkontekstowych. Poprzednikiem produkcji jest pojedynczy symbol nieterminalny, ale w następniku może występować nie więcej niż jeden nieterminal. wyłącznie na początku lub końcu następnika we wszystkich produkcjach. Jeżeli możliwe symbole nieterminalne są ostatnimi symbolami następników mówimy o gramatyce prawostronnie regularnej; w przypadku terminali będących pierwszymi symbolami następników mówimy o gramatyce lewostronnie regularnej. Formalnie warunki możemy zapisać w postaci
   - dla gramatyki prawostronnie regularnej:
     $P \subset N \times T^{*}(V \cup \{\varepsilon\})$
   - dla gramatyki lewostronnie regularnej:
     $P \subset N \times (V \cup \{\varepsilon\})T^{*}$

POLITECHNIKA LUBELSKA
WYDZIAŁ ELEKTROTECHNIKI I INFORMATYKI

INFORMATYKA

Zintegrowany Program Rozwoju Politechniki Lubelskiej - część druga

Materiały zostały opracowane w ramach projektu
„Zintegrowany Program Rozwoju Politechniki Lubelskiej – część druga”,
umowa nr **POWR.03.05.00-00-Z060/18-00**
w ramach Programu Operacyjnego Wiedza Edukacja Rozwój 2014-2020
współfinansowanego ze środków Europejskiego Funduszu Społecznego

Fundusze Europejskie Wiedza Edukacja Rozwój

Rzeczpospolita Polska

Unia Europejska Europejski Fundusz Społeczny