

Zagadnienia na licencjat

1. Ciągi liczb rzeczywistych. Zbieżność ciągu, warunek Cauchy'ego.

Definicja

Ciągiem nazywamy funkcję, której dziedziną jest zbiór liczb naturalnych \mathbb{N} lub jego skończony odcinek początkowy $\{1, 2, 3, \dots, m\}$. Ciągiem liczbowym nazywamy ciąg, którego wyrazy są liczbami.

Liczبę g nazywamy **granicą ciągu nieskończonego** ((a_n)) , jeśli dla każdej liczby dodatniej istnieje taka liczba k , że dla $n > k$ zachodzi nierówność: $|a_n - g| < \epsilon$

Ciągiem **zbieżnym (rozbieżnym)** nazywamy ciąg, który **posiada granicę (nie posiada granicy)**.

Własności:

1. Jeżeli ciąg posiada granicę to tylko jedną.
2. Każdy ciąg zbieżny jest ograniczony.
3. Przy założeniu, że ciągi (a_n) i (b_n) są zbieżne, zachodzą następujące wzory:
 - $\lim_{n \rightarrow \infty} (a_n \pm b_n) = \lim_{n \rightarrow \infty} a_n \pm \lim_{n \rightarrow \infty} b_n$ (dla iloczynu i ilorazu też zachodzi)
 - $\lim_{n \rightarrow \infty} -(a_n) = -\lim_{n \rightarrow \infty} a_n$
4. Jeżeli ciąg (b_n) jest zbieżny i $\lim_{n \rightarrow \infty} b_n \neq 0$, to $\lim_{n \rightarrow \infty} \frac{1}{b_n} = \frac{1}{\lim_{n \rightarrow \infty} b_n}$.
5. Jeżeli ciąg (a_n) jest zbieżny, to $\lim_{n \rightarrow \infty} |a_n| = |\lim_{n \rightarrow \infty} a_n|$.
6. (**Tw. o trzech ciągach**): Jeżeli $a_n \leq c_n \leq b_n$ i $\lim_{n \rightarrow \infty} a_n = g = \lim_{n \rightarrow \infty} b_n$, to ciąg (c_n) jest zbieżny, przy czym $\lim_{n \rightarrow \infty} c_n = \lim_{n \rightarrow \infty} b_n = \lim_{n \rightarrow \infty} a_n$.
7. Zmiana skończonej ilości wyrazów ciągu nie wpływa na jego zbieżność/granicę.
8. Podciąg ciągu zbieżnego jest zbieżny do tej samej granicy, co ciąg dany.

Warunek Cauchy'ego:

$$\forall \epsilon > 0 \quad \exists n_0 \in \mathbb{N} \quad \forall_{n, m > n_0} \quad (|a_n - a_m| < \epsilon)$$

Po ludzku: Ciągi Cauchy'ego to takie ciągi, dla których odległości między wyrazami zmierzają do zera. Oznacza to, że wybierając dowolnie małą dodatnią liczbę rzeczywistą ϵ , można ustalić odpowiednio duży wskaźnik \mathbb{N} taki, że dowolne dwa wyrazy o wyższych wskaźnikach są odległe od siebie o mniej niż ϵ .

Własności ciągu Cauchy'ego:

1. Ciąg $(a_n)_{n=1}^{\infty}$ jest zbieżny \iff gdy jest ciągiem Cauchy'ego. (**ale niekoniecznie odwrotnie**).
2. Każdy ciąg Cauchy'ego jest ograniczony.

2. Macierze. Podstawowe operacje na macierzach. Rząd i wyznacznik macierzy.

Macierzą (rzeczywistą) wymiaru $m \times n$, gdzie $m, n \in \mathbb{N}$, nazywamy prostokątną tablicę złożoną z $m \times n$ liczb rzeczywistych ustawionych w m wierszach i n kolumnach

$$\begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

Działania na macierzach

Suma/różnica

Niech macierz $A = [a_{ij}]$, $B = [b_{ij}]$. Sumą/różnicą nazywamy macierz $C = [c_{ij}]_{m \times n}$, której elementy określone są wzorami

$$c_{ij} = a_{ij} \pm b_{ij} \text{ dla } i \in \{1, 2, \dots, m\}, j \in \{1, 2, \dots, n\}.$$

Piszemy wtedy

$$C = A \pm B$$

Zatem,

$$\begin{bmatrix} c_{11} & c_{12} & \dots & c_{1n} \\ c_{21} & c_{22} & \dots & c_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \dots & c_{mn} \end{bmatrix} \begin{bmatrix} a_{11} \pm b_{11} & a_{12} \pm b_{12} & \dots & a_{1n} \pm b_{1n} \\ a_{21} \pm b_{21} & a_{22} \pm b_{22} & \dots & a_{2n} \pm b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & a_{m2} \pm b_{m2} & \dots & a_{mn} \pm b_{mn} \end{bmatrix}$$

Iloczyn macierzy przez liczbę

Niech $A = [a_{ij}]_{m \times n}$, α niech będzie dowolną liczbą rzeczywistą. Iloczynem macierzy A przez liczbę α nazywamy macierz $B = [b_{ij}]_{m \times n}$ której elementy określone są następująco:

$$b_{ij} = \alpha \cdot a_{ij}$$

dla $i \in \{1, 2, \dots, m\}, j \in \{1, 2, \dots, n\}$. Piszemy wtedy

$$B = \alpha \cdot A$$

Zatem

$$\begin{bmatrix} b_{11} & b_{12} & \dots & b_{1n} \\ b_{21} & b_{22} & \dots & b_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ b_{m1} & b_{m2} & \dots & b_{mn} \end{bmatrix} \begin{bmatrix} \alpha \cdot a_{11} & \alpha \cdot a_{12} & \dots & \alpha \cdot a_{1n} \\ \alpha \cdot a_{21} & \alpha \cdot a_{22} & \dots & \alpha \cdot a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ \alpha \cdot a_{m1} & \alpha \cdot a_{m2} & \dots & \alpha \cdot a_{mn} \end{bmatrix}$$

Iloczyn macierzy

Iloczyn macierzy AB jest możliwy jeśli macierz A ma tyle samo kolumn co macierz B ma wierszy. Iloczynem macierzy $A = [a_{ij}]_{n \times p}$ przez macierz $B = [b_{ij}]_{p \times m}$ nazywamy macierz $C = [c_{ij}]_{n \times m}$ taką, że:

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{ip}b_{pj} = \sum_{k=1}^p a_{ik}b_{kj}$$

Własności iloczynu macierzy:

- mnożenie macierzy jest łączne $A(BC) = (AB)C$ dlatego zapis ABC jest jednoznaczny
- mnożenie macierzy jest rodzielne względem dodawania $A(B+C) = AB + BC$ i $(B+C)A = BA + CA$
- mnożenie macierzy nie jest przemienne $AB \neq BA$

Macierz transponowana

Niech

$$A = \begin{bmatrix} -5 & 2 & 1 \\ -1 & 6 & 2 \\ 2 & 0 & 1 \\ 2 & 64 & 5 \end{bmatrix}$$

Wówczas

$$A^T = \begin{bmatrix} -5 & -1 & 2 & 2 \\ 2 & 6 & 0 & 64 \\ 1 & 2 & 1 & 5 \end{bmatrix}$$

Rząd macierzy

Chcąc obliczyć rząd macierzy musimy znaleźć największą macierz, której wyznacznik jest różny od zera, wielkość tej niezerowej macierzy będzie szukaną wartością, czyli jeśli największą macierzą, której wyznacznik jest różny od zera jest macierz 3×3 to rząd macierzy jest równy 3 .

Przykład

$$A = \begin{bmatrix} 2 & 8 & 3 & -4 \\ 1 & 4 & 1 & -2 \\ 5 & 20 & 0 & -10 \\ -3 & -12 & -2 & 6 \end{bmatrix}$$

Aby obliczyć rzad macierzy zaczynamy od obliczenia wyznacznika największej macierzy czyli w tym przypadku 4×4

$$\det(A) = \begin{vmatrix} 2 & 8 & 3 & -4 \\ 1 & 4 & 1 & -2 \\ 5 & 20 & 0 & -10 \\ -3 & -12 & -2 & 6 \end{vmatrix} = 0$$

Następnie obliczamy macierze 3×3 do momentu aż wyjdzie nam liczba różna od zera. Z macierzy 4×4 można utworzyć 16 macierzy 3×3 w następujący sposób:

1.

$$\det(A) = \begin{vmatrix} - & - & - & - \\ | & 4 & 1 & -2 \\ | & 20 & 0 & -10 \\ | & -12 & -2 & 6 \end{vmatrix}$$

2.

$$\det(A) = \begin{vmatrix} - & - & - & - \\ 1 & | & 1 & -2 \\ 5 & | & 0 & -10 \\ -3 & | & -2 & 6 \end{vmatrix}$$

3....

4....

5....

6....

7....

8....

9....

10....

11.

$$\det(A) = \begin{vmatrix} 2 & 8 & | & -4 \\ 1 & 4 & | & -2 \\ - & - & | & - \\ -3 & -12 & | & 6 \end{vmatrix}$$

12....

13....

14....

15....

16.

$$\det(A) = \begin{vmatrix} 2 & 8 & 3 & | \\ 1 & 4 & 1 & | \\ 5 & 20 & 0 & | \\ - & - & - & - \end{vmatrix}$$

Jeżeli któraś z nich wyjdzie różna od zera to koniec obliczeń. Wystarczy że obliczając wyznacznik macierzy pierwszej 3×3 wyjdzie nam liczba różna od zera - wtedy z automatu możemy powiedzieć, że rząd macierzy wynosi 3. Analogicznie jeżeli wszystkie wzory wyjdą na 0, to wtedy szukamy macierzy 2×2 poprzez wykreślenie wiersza i kolumny analogicznie jak do kroków powyżej tak aby otrzymać macierz 2×2 (czyli wykreślając po dwa wiersze i dwie kolumny).

Dla macierzy

$$\begin{bmatrix} 2 & 8 & 3 & -4 \\ 1 & 4 & 1 & -2 \\ 5 & 20 & 0 & -10 \\ -3 & -12 & -2 & 6 \end{bmatrix}$$

operacja ta wygadła następująco:

$$\begin{bmatrix} - & - & - & - \\ - & - & - & - \\ | & | & 0 & -10 \\ | & | & -2 & 6 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & -10 \\ -2 & 6 \end{bmatrix}$$

Wyznacznik macierzy

$$\begin{aligned} 1 \times 1 : \det(A) &= \begin{vmatrix} 5 \end{vmatrix} = 5 \\ 2 \times 2 : \det(A) &= \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} = a_{11} \cdot a_{22} - a_{21} \cdot a_{12} \\ 3 \times 3 : \text{np. Metoda Sarrusa} \end{aligned}$$

1. Dopusujemy z prawej kolumny 1 oraz 2

$$\det(A) = \left| \begin{array}{ccc|cc} a_{11} & a_{12} & a_{13} & a_{11} & a_{12} \\ a_{21} & a_{22} & a_{23} & a_{21} & a_{22} \\ a_{31} & a_{32} & a_{33} & a_{31} & a_{32} \end{array} \right|$$

2. Następnie wymnażamy wyrazy zaznaczone kolorem

$$\det(A) = \left| \begin{array}{ccc|cc} \color{blue}{a_{11}} & \color{blue}{a_{12}} & \color{red}{a_{13}} & a_{11} & a_{12} \\ a_{21} & \color{blue}{a_{22}} & \color{green}{a_{23}} & \color{red}{a_{21}} & a_{22} \\ a_{31} & a_{32} & \color{blue}{a_{33}} & \color{green}{a_{31}} & \color{red}{a_{32}} \end{array} \right|$$

czyli

$$a_{11} \cdot a_{22} \cdot a_{33} + a_{12} \cdot a_{23} \cdot a_{31} + a_{13} \cdot a_{21} \cdot a_{32}$$

3. Analogicznie w drugą stronę

$$\det(A) = \left| \begin{array}{ccc|cc} a_{11} & a_{12} & \color{blue}{a_{13}} & \color{green}{a_{11}} & \color{red}{a_{12}} \\ a_{21} & \color{blue}{a_{22}} & a_{23} & \color{red}{a_{21}} & a_{22} \\ \color{blue}{a_{31}} & a_{32} & \color{red}{a_{33}} & a_{31} & a_{32} \end{array} \right|$$

czyli

$$a_{13} \cdot a_{22} \cdot a_{31} + a_{11} \cdot a_{23} \cdot a_{32} + a_{12} \cdot a_{21} \cdot a_{33}$$

4. Następnie od wielomianu z kroku 2 odejmujemy wielomian z kroku 3:

$$(a_{11} \cdot a_{22} \cdot a_{33} + a_{12} \cdot a_{23} \cdot a_{31} + a_{13} \cdot a_{21} \cdot a_{32}) - (a_{13} \cdot a_{22} \cdot a_{31} + a_{11} \cdot a_{23} \cdot a_{32} + a_{12} \cdot a_{21} \cdot a_{33})$$

4×4 : np. Rozwinięcie Laplace'a

Szukamy w macierzy wiersza lub kolumny która ma najwięcej zer (dla łatwiejszego obliczania). Na przykład:

Dla macierzy

$$\begin{bmatrix} 1 & -1 & 2 & 4 \\ 0 & 1 & 0 & 3 \\ 5 & 7 & -2 & 0 \\ 2 & 0 & -1 & 4 \end{bmatrix}$$

Sytuacja wygląda tak:

$$\begin{vmatrix} 1 & -1 & 2 & 4 \\ 0 & 1 & 0 & 3 \\ 5 & 7 & -2 & 0 \\ 2 & 0 & -1 & 4 \end{vmatrix} = 0 \cdot (-1)^{2+1} \cdot \begin{vmatrix} -1 & 2 & 4 \\ 7 & -2 & 0 \\ 0 & -1 & 4 \end{vmatrix} + 1 \cdot (-1)^{2+2} \cdot \begin{vmatrix} 1 & 2 & 4 \\ 5 & -2 & 0 \\ 2 & -1 & 4 \end{vmatrix} + 0 \cdot (-1)^{2+3} \cdot \begin{vmatrix} 1 & -1 & 4 \\ 5 & 7 & 0 \\ 2 & 0 & 4 \end{vmatrix} + 3 \cdot (-1)^{2+4} \cdot \begin{vmatrix} 1 & -1 & 2 \\ 5 & 7 & -2 \\ 2 & 0 & -1 \end{vmatrix} =$$

$$((1 \cdot (-2) \cdot 4 + 2 \cdot 0 \cdot 2 + 4 \cdot 5 \cdot (-1)) - (2 \cdot (-2) \cdot 4 + 1 \cdot 0 \cdot (-1) + 4 \cdot 5 \cdot 2)) + 3 \cdot ((1 \cdot 7 \cdot (-1) + (-1) \cdot (-2) \cdot 2 + 2 \cdot 5 \cdot 0) - (2 \cdot 7 \cdot 2 + 1 \cdot (-2) \cdot 0 + (-1) \cdot (-1) \cdot 5)) =$$

$$((-8 + 0 + (-20)) - ((-16) + 0 + 40)) + 3 \cdot (((-7) + 4 + 0) - (28 + 0 + 5)) =$$

$$((-28) - (24)) + 3 \cdot ((-3) - (33)) =$$

$$(-52) + 3 \cdot (-36) =$$

$$-52 + -108 = -160$$

Gdzie po równaniu pierwsza liczba to liczba z wiersza m i kolumny n pomnożone przez (-1) do potegi $(m+n)$ razy macierz po wykreśleniu wiersza m i kolumny n . Następne dodajemy analogicznie.

Macierz odwrotna (A^{-1})

Wzór:

$$A^{-1} = (A^D)^T \cdot \frac{1}{\det(A)}$$

gdzie:

- A^{-1} - macierz odwrotna

- A^D - macierz dopełnień algebraicznych
- $(A^D)^T$ - macierz dołączona - czyli transponowana z macierzy dopełnień algebraicznych
- $\det(A)$ - wyznacznik macierzy

Nie można obliczyć macierzy odwrotnej z macierzy osobliwej, czyli takiej której wyznacznik jest równy 0. Więc jeśli liczymy macierz odwrotną zawsze zaczynamy od obliczania wyznacznika macierzy, jeśli wyjdzie on 0 to znaczy, że z danej macierzy nie można obliczyć macierzy odwrotnej.

Macierz odwrotna jest określona tylko dla macierzy kwadratowych, których wyznacznik W jest $W \neq 0$.

Macierz odwrotna A^{-1} do macierzy kwadratowej A to macierz spełniająca równanie $A^{-1} \cdot A = A \cdot A^{-1} = I$, gdzie I to macierz jednostkowa.

Jeśli macierz A^{-1} istnieje to macierz A nazywamy **odwracalną**, a jeśli macierz A^{-1} nie istnieje to macierz A nazywamy **nieodwracalną**.

Jeśli macierz A jest odwracalna to istnieje tylko jedna macierz odwrotna A^{-1}

Własności macierzy odwrotnej

- macierzą odwrotną od macierzy jednostkowej jest ta sama macierz tzn. $I^{-1} = I$
- $(\text{diag}(a_{11}, a_{22}, \dots, a_{nn}))^{-1} = \text{diag}((a_{11})^{-1}, (a_{22})^{-1}, \dots, (a_{nn})^{-1})$
- $(A^{-1})^{-1} = A$
- $(A^{-1})^T = (A^T)^{-1}$
- $(cA)^{-1} = c^{-1}A^{-1}$, gdzie c to stała
- $(AB)^{-1} = B^{-1}A^{-1}$
- $\det(A^{-1}) = (\det(A))^{-1}$
- macierz odwrotna do nieosobliwej macierzy symetrycznej jest symetryczna
- macierz odwrotna do nieosobliwej macierzy trójkątnej jest trójkątna

Macierz nieosobliwa - macierz kwadratowa o wyznaczniku różnym od zera

Macierz symetryczna - macierz kwadratowa której wyrazy położone symetrycznie względem przekątnej głównej są równe, przykład:

$$\begin{bmatrix} a_{11} & a_{21} & \dots & a_{n1} \\ a_{21} & a_{22} & \dots & a_{n2} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{bmatrix}$$

Proces obliczania macierzy odwrotnej dla macierzy 3×3

$$B = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 3 & 4 \\ 5 & -2 & -3 \end{bmatrix}$$

Podobnie jak wcześniej najpierw obliczamy wyznacznik macierzy 3×3 :

$$\det(B) = \begin{vmatrix} 2 & 5 & 7 \\ 6 & 3 & 4 \\ 5 & -2 & -3 \end{vmatrix} = -1$$

Następnie obliczamy macierz dopełnień algebraicznych:

$$B^D = \left[\begin{array}{c|cc|c|cc|c|cc} + & \left| \begin{array}{cc} a_{22} & a_{23} \\ a_{32} & a_{33} \end{array} \right| & - & \left| \begin{array}{cc} a_{21} & a_{23} \\ a_{31} & a_{33} \end{array} \right| & + & \left| \begin{array}{cc} a_{21} & a_{22} \\ a_{31} & a_{32} \end{array} \right| \\ - & \left| \begin{array}{cc} a_{12} & a_{13} \\ a_{32} & a_{33} \end{array} \right| & + & \left| \begin{array}{cc} a_{11} & a_{13} \\ a_{31} & a_{33} \end{array} \right| & - & \left| \begin{array}{cc} a_{11} & a_{12} \\ a_{31} & a_{33} \end{array} \right| \\ + & \left| \begin{array}{cc} a_{12} & a_{13} \\ a_{22} & a_{23} \end{array} \right| & - & \left| \begin{array}{cc} a_{11} & a_{13} \\ a_{21} & a_{23} \end{array} \right| & + & \left| \begin{array}{cc} a_{11} & a_{12} \\ a_{21} & a_{22} \end{array} \right| \end{array} \right]$$

czyli

$$B^D = \left[\begin{array}{c|cc|c|cc|c|cc} + & \left| \begin{array}{cc} 3 & 4 \\ -2 & -3 \end{array} \right| & - & \left| \begin{array}{cc} 6 & 4 \\ 5 & -3 \end{array} \right| & + & \left| \begin{array}{cc} 6 & 3 \\ 5 & -2 \end{array} \right| \\ - & \left| \begin{array}{cc} 5 & 7 \\ -2 & -3 \end{array} \right| & + & \left| \begin{array}{cc} 2 & 7 \\ 5 & -3 \end{array} \right| & - & \left| \begin{array}{cc} 2 & 5 \\ 5 & -2 \end{array} \right| \\ + & \left| \begin{array}{cc} 5 & 7 \\ 3 & 4 \end{array} \right| & - & \left| \begin{array}{cc} 2 & 7 \\ 6 & 4 \end{array} \right| & + & \left| \begin{array}{cc} 2 & 5 \\ 6 & 3 \end{array} \right| \end{array} \right]$$

Więc:

$$B^D = \begin{bmatrix} -1 & 38 & -27 \\ 1 & -41 & 29 \\ -1 & 34 & -24 \end{bmatrix}$$

Następnie obliczamy macierz transponowaną:

$$(B^D)^T = \begin{bmatrix} -1 & 1 & -1 \\ 38 & -41 & 34 \\ -27 & 29 & -24 \end{bmatrix}$$

więc macierz odwrotna będzie miała postać:

$$\frac{(B^D)^T}{\det(B)} = \begin{bmatrix} 1 & -1 & 1 \\ -38 & 41 & -34 \\ 27 & -29 & 24 \end{bmatrix}$$

Ślad macierzy

Ślad macierzy jest to suma elementów leżących na przekątnej danej macierzy. Ślad macierzy definujemy tylko dla macierzy kwadratowej. Ślad macierzy kwadratowej $A = [a_{ij}]$ stopnia n jest sumą elementów leżących na głównej przekątnej (diagonali). Ślad macierzy oznaczamy $\text{Tr}(A)$, $\text{Tr} A$ lub $\text{trace}(A)$.

$$Tr(A) = \sum_{i=1}^n a_{ii} = a_{11} + a_{22} + \dots + a_{nn}$$

Mając macierz

$$A = \begin{bmatrix} 1 & 2 & 4 & -4 \\ -3 & 4 & -6 & 12 \\ -9 & 2 & -1 & 3 \\ 5 & 0 & 2 & 1 \end{bmatrix}$$

obliczamy ślad macierzy w następujący sposób:

$$tr(A) = tr \begin{bmatrix} 1 & 2 & 4 & -4 \\ -3 & 4 & -6 & 12 \\ -9 & 2 & -1 & 3 \\ 5 & 0 & 2 & 1 \end{bmatrix} = 1 + 4 + (-1) + 1 = 5$$

Własności śladu macierzy

- jeśli macierze $A = a_{ij}$ i $B = b_{ij}$ są macierzami kwadratowymi tego samego stopnia to: $Tr(A + B) = Tr(A) + Tr(B)$
- jeśli macierz $A = a_{ij}$ jest macierzą kwadratową, a α jest liczbą rzeczywistą to: $Tr(\alpha A) = \alpha Tr(A)$
- jeśli $A \in M_n$ a $B \in M_n$ to: $Tr(AB) = Tr(BA)$
- jeśli $A \in M_n$, $B \in M_n$ i $C \in M_n$ (cykliczna przemienność śladu) to: $Tr(ABC) = Tr(CAB) = Tr(BCA)$
- przekątna głównej macierzy nie ulegnie zmianie przy transpozycji: $Tr(A) = Tr(A^T)$

3. Rozwiązywanie układów równań liniowych.

Rozważmy układ m równań liniowych o n niewiadomych x_1, x_2, \dots, x_n :

$$(*) \left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \dots + a_{mn}x_n = b_m \end{array} \right\}$$

oraz macierz $A \in M_{n \times m}$:

$$A = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

nazywać będziemy **macierzą układu** (*).

Macierz

$$B = \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_m \end{bmatrix}$$

nazywać będziemy **macierzą wynikową**.

Macierz

$$A_r = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n}b_1 \\ a_{21} & a_{22} & \dots & a_{2n}b_2 \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn}b_m \end{bmatrix}$$

nazywać będziemy **macierzą rozszerzoną układu (*)**.

Rozwiązaniem układu (*) nazywać będziemy każdy n -elementowy ciąg y_1, y_2, \dots, y_n taki, że po podstawieniu $x_i = y_i$ do układu (*) otrzymujemy n równości:

$$\left\{ \begin{array}{l} a_{11}y_1 + a_{12}y_2 + \dots + a_{1n}y_n = b_1 \\ a_{21}y_1 + a_{22}y_2 + \dots + a_{2n}y_n = b_2 \\ \vdots \\ a_{m1}y_1 + a_{m2}y_2 + \dots + a_{mn}y_n = b_m \end{array} \right\}$$

Operacje elementarne

Za pomocą operacji elementarnych możemy przekształcać wiersze macierzy.

Mögliwe operacje elementarne:

- Dodanie do dowolnego wiersza innego wiersza pomnożonego lub nie przez liczbę
- Odjęcie od dowolnego wiersza innego wiersza pomnożonego lub nie przez liczbę
- Pomnożenie dowolnego wiersza przez liczbę różną od zera
- Zamiana miejscami dwóch wierszy

Metoda eliminacji Gaussa

Metoda eliminacji Gaussa polega na stosowaniu operacji elementarnych do macierzy rozszerzonej układu równań liniowych tak, aby doprowadzić macierz rozszerzoną do postaci schodkowej.

Postać schodkowa:

$$\left[\begin{array}{cccccc} 1 & 0 & 0 & \dots & 0 & a \\ 0 & 1 & 0 & \dots & 0 & b \\ 0 & 0 & 1 & \dots & 0 & c \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & 1 & z \end{array} \right] \text{ gdzie } \{a, b, c, \dots, z\} \in \mathbb{R}$$

Przykład:

$$\left\{ \begin{array}{l} x_1 + 2x_2 + 3x_3 = 14 \\ x_1 + 3x_2 + x_3 = 10 \\ 2x_1 + 5x_2 + 5x_3 = 27 \end{array} \right\} \Leftrightarrow \left[\begin{array}{cccc} 1 & 2 & 3 & 14 \\ 1 & 3 & 1 & 10 \\ 2 & 3 & 1 & 27 \end{array} \right] \xrightarrow[w_3 - 2w_1]{w_2 - w_1} \left[\begin{array}{cccc} 1 & 2 & 3 & 14 \\ 0 & 1 & -2 & -4 \\ 0 & 1 & -1 & -1 \end{array} \right] \xrightarrow[w_3 - w_2]{w_2 - w_1} \left[\begin{array}{cccc} 1 & 2 & 3 & 14 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & 1 & 3 \end{array} \right] \xrightarrow[w_2 + 2w_3]{w_3 - 2w_1} \left[\begin{array}{cccc} 1 & 2 & 3 & 14 \\ 0 & 1 & -2 & -4 \\ 0 & 0 & 1 & 3 \end{array} \right]$$

$$\left[\begin{array}{cccc} 1 & 2 & 3 & 14 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array} \right] \xrightarrow[w_1 - 3w_3]{w_1 - 3w_3} \left[\begin{array}{cccc} 1 & 2 & 0 & 5 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array} \right] \xrightarrow[w_1 - 2w_2]{w_1 - 3w_3} \left[\begin{array}{cccc} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \end{array} \right] \Leftrightarrow \left\{ \begin{array}{l} x_1 = 1 \\ x_2 = 2 \\ x_3 = 3 \end{array} \right\}$$

Twierdzenie Cramera

Rozważamy układ n równań liniowych o n niewiadomych x_1, x_2, \dots, x_n :

$$\left\{ \begin{array}{l} a_{11}x_1 + a_{12}x_2 + \dots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \dots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \dots + a_{nn}x_n = b_n \end{array} \right\}$$

Jeżeli wyznacznik macierzy tego układu jest różny od zero ($\det(A) \neq 0$) to ten układ ma dokładnie jedno rozwiązanie postaci:

$$x_i = \frac{\det(A_i)}{\det(A)}, i = 1, 2, \dots, n$$

Natomiast macierz A_i powstaje z macierzy A poprzez zastąpienie i -tej kolumny macierzy A przez kolumnę macierzy wynikowej.

Przykład

$$\left\{ \begin{array}{l} 8x_1 + x_2 + 2x_3 = 16 \\ 5x_1 - 3x_2 - 7x_3 = -22 \\ -5x_2 + 7x_3 = 11 \end{array} \right\}$$

Najpierw sprawdźmy czy wyznacznik macierzy układu jest różny od zera

$$A = \left[\begin{array}{ccc} 8 & 1 & 2 \\ 5 & -3 & -7 \\ 0 & -5 & 7 \end{array} \right]$$

$$\det(A) = \begin{vmatrix} 8 & 1 & 2 \\ 5 & -3 & -7 \\ 0 & -5 & 7 \end{vmatrix} = -553$$

Oznacza to, że układ ten ma dokładnie jedno rozwiązanie.

Musimy znaleźć wyznacznik odpowiadający kolejnym niewiadomym, czyli A_1, A_2, A_3 .

\$A_1\$

Wykreslamy pierwszą kolumnę macierzy

$$A = \begin{bmatrix} 8 & 1 & 2 \\ 5 & -3 & -7 \\ 0 & -5 & 7 \end{bmatrix}$$

W wolne miejsce wpisujemy kolumnę macierzy wynikowej

$$A_1 = \begin{bmatrix} 16 & 1 & 2 \\ -22 & -3 & -7 \\ 11 & -5 & 7 \end{bmatrix}$$

Obliczamy wartość $\det(A_1)$

$$\det(A_1) = \begin{vmatrix} 16 & 1 & 2 \\ -22 & -3 & -7 \\ 11 & -5 & 7 \end{vmatrix} = -533$$

Teraz możemy wyznaczyć niewiadomą x_1

$$x_1 = \frac{\det(A_1)}{\det(A)} = \frac{-533}{-553} = 1$$

\$A_2\$

$$A = \begin{bmatrix} 8 & 1 & 2 \\ 5 & -3 & -7 \\ 0 & -5 & 7 \end{bmatrix}$$

$$A_2 = \begin{bmatrix} 8 & 16 & 2 \\ 5 & -22 & -7 \\ 0 & 11 & 7 \end{bmatrix}$$

$$\det(A_2) = \begin{vmatrix} 8 & 16 & 2 \\ 5 & -22 & -7 \\ 0 & 11 & 7 \end{vmatrix} = -1066$$

$$x_2 = \frac{\det(A_2)}{\det(A)} = \frac{-1066}{-533} = 2$$

\$A_3\$

$$A = \begin{bmatrix} 8 & 1 & 2 \\ 5 & -3 & -7 \\ 0 & -5 & 7 \end{bmatrix}$$

$$A_3 = \begin{bmatrix} 8 & 1 & 16 \\ 5 & -3 & -22 \\ 0 & -5 & 11 \end{bmatrix}$$

$$\det(A_3) = \begin{vmatrix} 8 & 1 & 16 \\ 5 & -3 & -22 \\ 0 & -5 & 11 \end{vmatrix} = -1599$$

$$x_3 = \frac{\det(A_3)}{\det(A)} = \frac{-1599}{-533} = 3$$

Podsumowując otrzymalismy nastepujace rzwiazanie ukldau rownan:

$$x_1 = 1$$

$$x_2 = 2$$

$$x_3 = 3$$

Wniosek z twierdzenia Cramera

- $\det(A) = 0$ i $\det(A_i) = 0$ (dla kazdego i) - uklad rownan liniowych ma nisekonczenie wiele rzwiazan
- $\det(A) = 0$ i istnieje $\det(A_i) \neq 0$ - uklad rownan liniowych jest sprzeczny

4. Rachunek zdań. Tautologie.

Prawem rachunku zdan lub **tautologia** nazywamy wyrażenie zbudowane ze zdań prostych i spójników, które zawsze jest zdaniem prawdziwym (niezależnie od wartości logicznych zdań prostych).

TAUTOLOGIA

W logice wartość logiczną zdania definiujemy jako **0**, gdy zdanie to jest fałszywe, zaś jako **1**, gdy zdanie to jest prawdziwe. Symbolu **0** używamy również do oznaczenia dowolnego zdania fałszywego, zaś symbolu **1** do oznaczenia dowolnego zdania prawdziwego.

Koniunkcja - to dwa zdania połączone spójnikiem logicznym *i*.

Koniunkcja dwóch zdań $p \wedge q$ jest prawdziwa jedynie wtedy, gdy oba zdania p oraz q są prawdziwe.

Alternatywa - to dwa zdania połączone spójnikiem logicznym *lub*.

Alternatywa dwóch zdań $p \vee q$ jest prawdziwa wtedy, gdy przynajmniej jedno ze zdań p lub q jest prawdziwe.

Koniunkcja \wedge

p	q	$p \wedge q$
1	1	1
1	0	0
0	1	0
0	0	0

Alternatywa \vee

p	q	$p \vee q$
1	1	1
1	0	1
0	1	1
0	0	0

Implikacja \implies - możemy odczytywać na wiele równoważnych sposobów:

- "p pod warunkiem, że q"
- "p wtedy, gdy q"
- "p tylko wtedy, gdy q"
- "p jest warunkiem dostatecznym do tego, że q"
- "p jest warunkiem koniecznym do tego, że q"

Równoważność \iff - możemy odczytywać na wiele równoważnych sposobów:

- "p wtedy i tylko wtedy, gdy q"
- "p dokładnie wtedy, gdy q"
- "p jest warunkiem koniecznym i dostatecznym do tego, że q"
- "p jest równoważne temu, że q"

Implikacja \implies

p	q	$p \implies q$
1	1	1
1	0	0
0	1	1
0	0	1

Równoważność \iff

<i>p</i>	<i>q</i>	<i>p</i> \iff <i>q</i>
1	1	1
1	0	0
0	1	0
0	0	1

Negacja \neg - oznacza jednoargumentowy spójnik negacji, $\neg p$ oznacza zdanie:

- "nie *p*"
- "nieprawda, że *p*"

Negacja \neg

<i>p</i>	$\neg p$
1	0
0	1

Najważniejsze tautologie

Tautologia (z greki) - to wyrażenie, zdanie logiczne, które zawsze jest logiczne

Nazwa tautologii

- prawo wyłączonego środka
- prawo sprzeczności
- prawo podwójnej negacji
- I prawo de Morgana
- II prawo de Morgana
- prawo odrywania
- prawo negacji implikacji
- rozdzielnosc koniunkcji względem alternatywy
- rozdzielnosc alternatywy względem koniunkcji

Tautologia

- $p \vee (\neg p)$
- $\neg(p \wedge (\neg p))$
- $p \Leftrightarrow \neg(\neg p)$
- $(\neg(p \wedge q)) \Leftrightarrow ((\neg p) \vee (\neg q))$
- $(\neg(p \vee q)) \Leftrightarrow ((\neg p) \wedge (\neg q))$
- $(p \wedge (p \Rightarrow q)) \Rightarrow q$
- $(\neg(p \Rightarrow q)) \Leftrightarrow (p \wedge (\neg q))$
- $(p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r))$
- $(p \vee (q \wedge r)) \Leftrightarrow ((p \vee q) \wedge (p \vee r))$

Prawo wyłączonego środka

<i>p</i>	$p \vee (\neg p)$
1	1
0	1

Prawo przemienności koniunkcji

<i>p</i>	<i>q</i>	$(p \wedge q) \iff (q \wedge p)$
1	1	1
1	0	0
0	1	0
0	0	1

Prawo przemienności alternatywy

p	q	$(p \vee q)$	\iff	$(q \vee p)$
1	1	1		
1	0	1		
0	1	1		
0	0	1		

Prawo podwójnej negacji

p	$\neg p$	$\neg(\neg p)$	p	\iff	$\neg(\neg p)$
1	0	1	1		
0	1	0	1		

Prawo sprzeczności

p	$\neg p$	$p \wedge (\neg p)$	$\neg(p \wedge (\neg p))$
1	0	0	1
0	1	0	1

I Prawo de Morgana

p	q	$\neg(p \wedge q)$	$\neg p$	$\neg q$	$(\neg p) \vee (\neg q)$	$(\neg(p \wedge q))$	\iff	$((\neg p) \vee (\neg q))$
1	1	0	0	0	0	1		
1	0	1	0	1	1	1		
0	1	1	1	0	1	1		
0	0	1	1	1	1	1		

Zaprzeczenie koniunkcji dwóch zdań $\neg(p \wedge q)$ jest równoważne alternatywie zaprzeczeń tych zdań $(\neg p) \vee (\neg q)$

II Prawo de Morgana

p	q	$\neg(p \vee q)$	$\neg p$	$\neg q$	$(\neg p) \wedge (\neg q)$	$(\neg(p \vee q))$	\iff	$((\neg p) \wedge (\neg q))$
1	1	0	0	0	0	1		
1	0	0	0	1	0	1		
0	1	0	1	0	0	1		
0	0	1	1	1	1	1		

Zaprzeczenie alternatywy dwóch zdań $\neg(p \vee q)$ jest równoważne koniunkcją zaprzeczeń tych zdań $(\neg p) \wedge (\neg q)$

Prawo negacji implikacji

p	q	$p \implies q$	$\neg(p \implies q)$	$\neg q$	$p \wedge (\neg q)$	$\neg(p \implies q)$	\iff	$(p \wedge (\neg q))$
1	1	1	0	0	0	1		
1	0	0	1	1	1	1		
0	1	1	0	0	0	1		
0	0	1	0	1	0	1		

Zaprzeczenie implikacji dwóch zdań $\neg(p \implies q)$ jest równoważne koniunkcji $p \wedge (\neg q)$

Prawo odrywania

p	q	$p \Rightarrow q$	$p \wedge (p \Rightarrow q)$	$(p \wedge (p \Rightarrow q)) \Rightarrow q$
1	1	1	1	1
1	0	0	0	1
0	1	1	0	1
0	0	1	0	1

Jeżeli prawdziwe są implikacja $p \Rightarrow q$ oraz jej poprzednik p , to również jej następnik q jest zdaniem prawdziwym

Prawo rozdzielności koniunkcji względem alternatywy

p	q	r	$q \vee r$	$p \wedge (q \vee r)$	$p \wedge q$	$p \wedge r$	$(p \wedge q) \vee (p \wedge r)$	$(p \wedge (q \vee r)) \Leftrightarrow ((p \wedge q) \vee (p \wedge r))$
1	1	1	1	1	1	1	1	1
1	1	0	1	1	1	0	1	1
1	0	1	1	1	0	1	1	1
1	0	0	0	0	0	0	0	1
0	1	1	1	0	0	0	0	1
0	1	0	1	0	0	0	0	1
0	0	1	1	0	0	0	0	1
0	0	0	0	0	0	0	0	1

Prawo rozdzielności alternatywy względem koniunkcji

p	q	r	$q \wedge r$	$p \vee (q \wedge r)$	$p \vee q$	$p \vee r$	$(p \vee q) \wedge (p \vee r)$	$(p \vee (q \wedge r)) \Leftrightarrow ((p \vee q) \wedge (p \vee r))$
1	1	1	1	1	1	1	1	1
1	1	0	0	1	1	1	1	1
1	0	1	0	1	1	1	1	1
1	0	0	0	1	1	1	1	1
0	1	1	1	1	1	1	1	1
0	1	0	0	0	1	0	0	1
0	0	1	0	0	0	1	0	1
0	0	0	0	0	0	0	0	1

5. Indukcja matematyczna.

Indukcja matematyczna – metoda dowodzenia twierdzeń o prawdziwości nieskończonej liczby stwierdzeń oraz definiowania rekurencyjnego. W najbardziej typowych przypadkach dotyczą one liczb naturalnych.

Aksjomat indukcji matematycznej

Jeśli S jest podzbiorem \mathbb{N} , który spełnia:

- $1 \in S$
- dla wszystkich $k \in \mathbb{N}$, jeśli $k \in S$, to $k + 1 \in S$

to S stanowi całość \mathbb{N} , tzn $S = \mathbb{N}$.

Innymi słowy oznacza to że jeżeli dowiedziemy się, że dany zbiór S posiada takie same właściwości jak zbiór \mathbb{N} (tj. jest dyskretny, posiada początkowy element oraz odległość między następnymi dyskretnymi elementami jest zawsze taka sama) to element ten jest podzbiorem \mathbb{N} ale np przesunięty i powiększony o jakiś skalar.

Przykłady

Suma początkowych potęg dwójki

Zobacz też: potęga dwójki.

Należy dowieść

$$2^1 + 2^2 + 2^3 + \cdots + 2^n = 2^{n+1} - 2 \quad \text{dla wszystkich } n \in \mathbb{N}.$$

- Jest $2^1 = 2^{1+1} - 2$, co dowodzi prawdziwości stwierdzenia dla $n = 1$.
- Zakładając $2 + 2^2 + 2^3 + \cdots + 2^k = 2^{k+1} - 2$ należy dowieść $2^1 + 2^2 + 2^3 + \cdots + 2^k + 2^{k+1} = 2^{(k+1)+1} - 2$.

Ponieważ

$$\begin{aligned} 2^1 + 2^2 + 2^3 + \cdots + 2^k + 2^{k+1} &= (2^{k+1} - 2) + 2^{k+1} && (\text{hipoteza ind.}) \\ &= 2(2^{k+1}) - 2 \\ &= 2^{k+2} - 2, \end{aligned}$$

to wzór jest prawdziwy dla $n = k + 1$, jeśli tylko jest prawdziwy dla $n = k$.

Zatem

$$2^1 + 2^2 + 2^3 + \cdots + 2^n = 2^{n+1} - 2 \quad \text{dla wszystkich } n \in \mathbb{N}.$$

Nierówność Bernoulliego

Zobacz też: nierówność Bernoulliego.

Niech $h \geq -1$ będzie ustaloną liczbą rzeczywistą. Należy udowodnić, że $(1 + h)^n \geq 1 + nh$ dla wszystkich $n \in \mathbb{N}$.

- Skoro $(1 + h)^1 \geq 1 + 1h$, to nierówność jest prawdziwa dla $n = 1$.
- Przyjmując $(1 + h)^k \geq 1 + kh$ wykazana zostanie $(1 + h)^{k+1} \geq 1 + (k + 1)h$.

Zachodzi

$$\begin{aligned} (1 + h)^{k+1} &= (1 + h)^k(1 + h) \\ &\geq (1 + kh)(1 + h) && (\text{hip. ind. i } 1 + h \geq 0) \\ &= 1 + h + kh + kh^2 \\ &\geq 1 + h + kh + 0 \\ &= 1 + (k + 1)h, \end{aligned}$$

więc nierówność jest prawdziwa dla $n = k + 1$, o ile jest prawdziwa dla $n = k$.

Stąd

$$(1 + h)^n \geq 1 + nh \quad \text{dla wszystkich } n \in \mathbb{N} \text{ (oraz } h \geq -1\text{)}.$$

6. Permutacje, wariacje i kombinacje.

Permutacja

Permutacja zbioru n -elementowego - to dowolny n -wyrazowy ciąg utworzony ze wszystkich elementów tego zbioru.

Liczę permuatacji zbioru n -elementowego możemy obliczyć ze wzoru:

$$P_n = n!$$

Przykłady

1. Na ile sposobów można ustawić 5 osób w kolejce?

$$P_5 = 5! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 = 120$$

2. Ile liczb można utworzyć z cyfr: 1, 2, 3, 4, 5, 6, 7?

$$P_7 = 7! = 1 \cdot 2 \cdot 3 \cdot 4 \cdot 5 \cdot 6 \cdot 7 = 5040$$

Kombinacja

Kombinacja pozwala policzyć na ile sposobów można wybrać k elementów z n -elementowego zbioru.

Wzór na kombinację jest następujący:

$$C_n^k = \frac{n!}{k!(n-k)!}$$

Kombinację zapisujemy krótko za pomocą Symbolu Newtona:

$$\binom{n}{k} = \frac{n!}{k!(n-k)!}$$

Przykłady

1. Na ile sposobów można wybrać 2 osoby w klasie 30 osobowej?

$$\binom{30}{2} = \frac{30!}{2!(30-2)!} = \frac{30!}{2 \cdot 28!} = \frac{29 \cdot 30}{2} = 15 \cdot 29 = 435$$

2. Na ile sposobów można wybrać 3 zawodników w dzudnie 12 osobowej?

$$\binom{12}{3} = \frac{12!}{3!(12-3)!} = \frac{12!}{6 \cdot 9!} = \frac{10 \cdot 11 \cdot 12}{6} = 220$$

Wariacja z powtóżeniami

Przyjmijmy, że mamy dany zbiór elementów (np. zbiór liter). Wariacja z powtórzeniami pozwala na utworzenie ciągu z elementów tego zbioru, z tym, że dopuszcza powtarzanie elementów. Wzór na wariację z powtórzeniami jest następujący:

$$W_n^k = n^k$$

Przykłady

1. Ile słów pięcioliterowych (nawet tych bezsensownych) można utworzyć z liter $\{A, B, C\}$?

Przykładami taki słów są: *AAAAA, AABCA, CBCB*.

Na każde z 5 miejsc możemy wybrać jedną z 3 liter, zatem wszystkich możliwości mamy:

$$3^5 = 243$$

2. Ile słów dwuliterowych (nawet tych bezsensownych) można utworzyć z liter $\{A, B, C, D\}$?

Przykładami taki słów są: *AA, DC, CD*.

Na każde z 2 miejsc możemy wybrać jedną z 4 liter, zatem wszystkich możliwości mamy:

$$4^2 = 16$$

Wariacja bez powtórzeń

Przyjmijmy, że mamy dany zbiór elementów (np. zbiór liter). Wariacja bez powtórzeń pozwala na utworzenie ciągu z elementów tego zbioru, z tym, że nie dopuszcza powtarzania elementów. Wzór na wariację bez powtórzeń jest następujący:

$$V_n^k = \frac{n!}{(n-k)!}$$

Przykłady

1. Ile istnieje czterocyfrowych PIN-kodów składających się z różnych cyfr?

Mamy do dyspozycji 10 cyfr: $\{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$.

Przykładowymi kodami o różnych cyfrach są: *1234, 0189, 9734*. Wszystkich takich wariacji bez powtórzeń jest:

$$\frac{10!}{6!} = 7 \cdot 8 \cdot 9 \cdot 10 = 5040$$

7. Klasyczna definicja prawdopodobieństwa. Prawdopodobieństwo geometryczne.

Definicja

Zakładamy, że przestrzeń zdarzeń elementarnych Ω ma skończoną liczbę zdarzeń elementarnych i każde z nich jest jednakowo prawdopodobne. Wtedy prawdopodobieństwo definiujemy następująco:

Dla dowolnego $A \subset \Omega$: $P(A) = \frac{|A|}{|\Omega|}$

Kilka wyjaśnień:

- $P(A)$ to prawdopodobieństwo zajścia zdarzenia A
- $|A|$ to liczbe zdarzenia A ,
- $|\Omega|$ to liczbe przestrzeni zdarzeń elementarnych Ω (czyli liczba wszystkich zdarzeń elementarnych w doświadczeniu losowym).

Krótko: prawdopodobieństwo uzyskania jakiegoś wyniku to liczba sprzyjających wyników przez liczbę wszystkich możliwych wyników. Zatem chcąc obliczyć prawdopodobieństwo zajścia pewnego zdarzenia:

1. Opisujemy przestrzeń zdarzeń elementarnych Ω .
2. Upewniamy się, że każde zdarzenie elementarne jest jednakowo prawdopodobne.
3. Zliczamy liczbę elementów Ω , czyli obliczamy $|\Omega|$.
4. Opisujemy zdarzenie A , którego prawdopodobieństwo zajścia chcemy obliczyć.
5. Zliczamy liczbę elementów zdarzenia A , czyli obliczamy $|A|$.
6. Obliczamy prawdopodobieństwo zajścia zdarzenia A ze wzoru: $P(A) = \frac{|A|}{|\Omega|}$.

Własności prawdopodobieństwa

Prawdopodobieństwo dowolnego zdarzenia losowego A jest zawsze liczbą z przedziału $(0;1)$.

$$0 \leq P(A) \leq 1$$

Prawdopodobieństwo zdarzenia pewnego jest równe 1.

$$P(\Omega) = 1$$

Prawdopodobieństwo zdarzenia niemożliwego jest równe 0.

$$P(\emptyset) = 0$$

Przydatne wzory

Prawdopodobieństwo zdarzenia przeciwnego:

$$P(A') = 1 - P(A)$$

Prawdopodobieństwo sumy zdarzeń

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

Prawdopodobieństwo warunkowe

Prawdopodobieństwo warunkowe zajścia zdarzenia A pod warunkiem zajścia zdarzenia B liczymy ze wzoru:

$$P(A|B) = P(A \cap B)P(B)$$

gdzie $P(B) > 0$

Prawdopodobieństwo całkowite

Jeżeli zdarzenia B_1, B_2, \dots, B_n są parami rozłączne oraz mają prawdopodobieństwa dodatnie, które sumują się do jedynki, to dla dowolnego zdarzenia A zachodzi wzór:

$$P(A) = P(A|B_1) \cdot P(B_1) + P(A|B_2) \cdot P(B_2) + \dots + P(A|B_n) \cdot P(B_n)$$

Wzór Bayesa

Jeżeli zdarzenia B_1, B_2, \dots, B_n są parami rozłączne oraz mają prawdopodobieństwa dodatnie, które sumują się do jedynki, to dla dowolnego zdarzenia A zachodzi wzór:

$$P(B_k|A) = P(A|B_k) \cdot P(B_k)P(A)$$

Schemat Bernoulliego

W schemacie Bernoulliego prawdopodobieństwo uzyskania k sukcesów w n próbach można obliczyć ze wzoru:

$$P_n(k) = \binom{n}{k} p^k (1-p)^{n-k}$$

gdzie p - to prawdopodobieństwo sukcesu w jednej próbie

8. Struktura logiczna i funkcjonalna klasycznego komputera.

Klasyczny komputer o architekturze podanej przez von Neumana składa się z trzech podstawowych bloków:

- procesora
- pamięci operacyjnej
- urządzeń wejścia/wyjścia.

Struktura logiczna komputera

- W pamięci przechowywane są przetwarzane dane oraz program dla procesora.
- Urządzenia wejścia/wyjścia umożliwiają wymianę informacji pomiędzy komputerem a otoczeniem.
- Procesor umożliwia przetwarzanie danych.

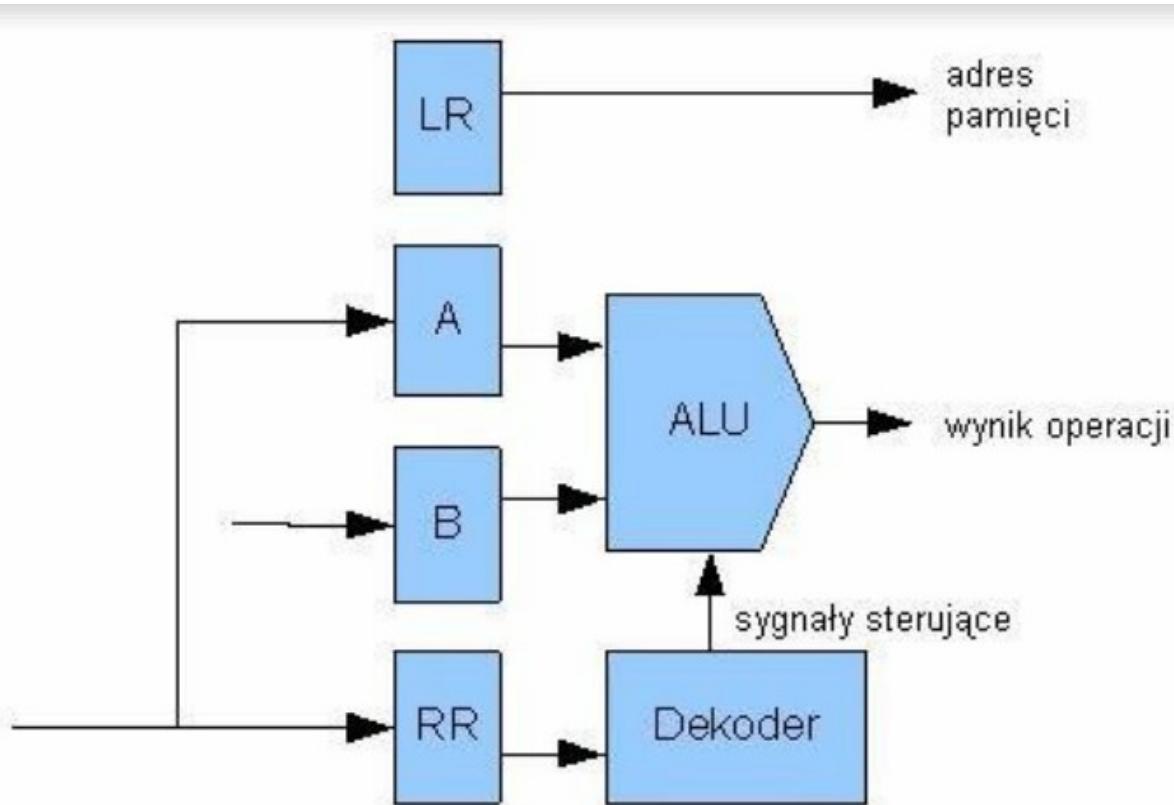
Po załadowaniu programu do pamięci komputera może on zostać w dowolnej chwili wywołany przez operatora. W tym celu musi on wydać polecenie rozpoczęcia wykonywania tego programu przez wymuszenie odczytania pierwszego polecenia tego programu. W tym celu należy spowodować, aby procesor wysłał do pamięci odpowiedni adres. Dalsze polecenia są umieszczone w pamięci kolejno, więc będą odczytywane przez procesor automatycznie. Wykonywanie programu polega, więc na pobieraniu z pamięci kolejnych polecień i odpowiednich dla tych polecień argumentów. Argumenty rozkazu mogą być:

- w pamięci i wówczas rozkaz musi zawierać adres miejsca w pamięci, gdzie one się znajduje,
- w rejestrach procesora i wówczas rozkaz musi wskazywać adres odpowiedniego rejestru,

- w samym rozkazie i wówczas programista umieszcza je w odpowiednio w kodzie programu.

W czasie wykonywania programu procesor odczytuje kolejne rozkazy, które następnie musi rozpoznać (dekodować). Po zdekodowaniu rozkazu, w zależności od treści tego rozkazu, procesor podejmuje odpowiednią akcję. Akcja ta polega na wykonaniu odpowiedniej operacji. Między innymi, z treści rozkazu, może wynikać konieczność odczytania argumentów dla niego.

Jeżeli argument znajduje się, w pamięci, to dalsza akcja polega na odczytaniu adresu tego argumentu. Jeżeli adres ten programista umieścił w kodzie programu, to odczytane będzie następne słowo(a) z kodu programu stanowiące ten adres. Jeżeli argument znajduje się, w rejestrze procesora, to rozkaz musi wskazać, w którym z rejestrów procesora znajduje się adres. Po skompletowaniu całej instrukcji procesor wykonuje ją, a dalej pobiera następny rozkaz i cała akcja się powtarza.



(schemat czytamy od gory - strzalka znika z prawej strony i pojawia sie po lewej dolnej stronie)

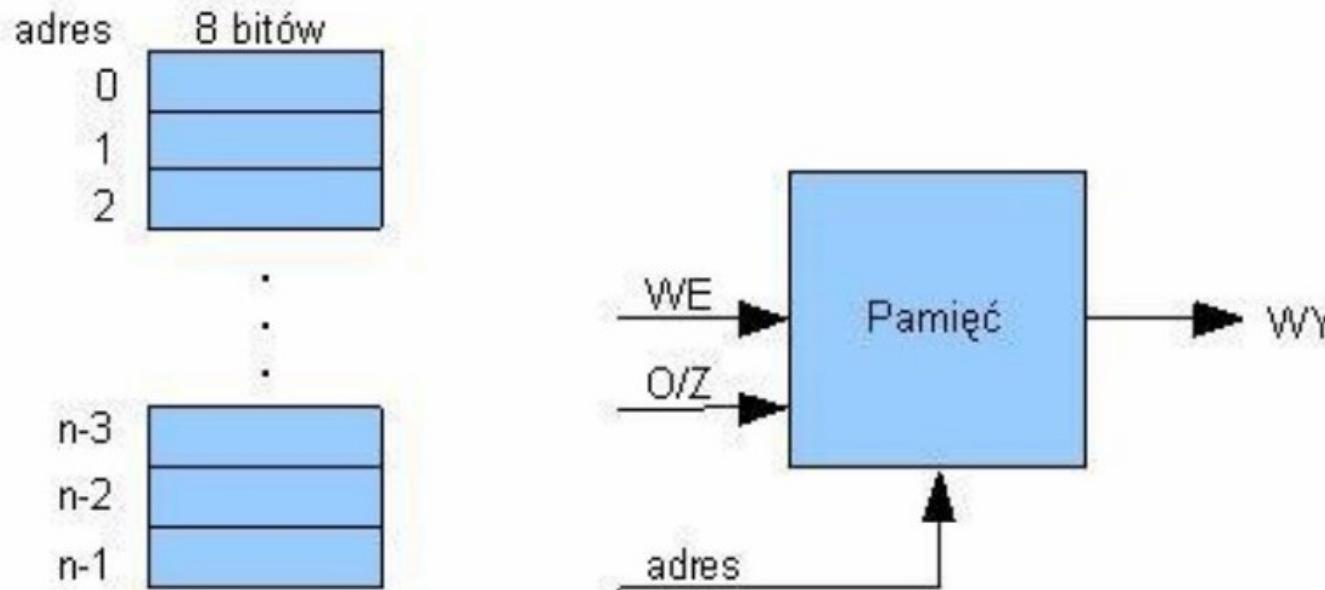
Na schemacie:

- **ALU** - jednostka arytmetyczno-logiczna - wykonuje obliczenia
- **RR** - rejestr rozkazów - posiada tablice dostępnych rozkazów
- **dekoder** - dekoduje rozkazy na sygnały sterujące **ALU**
- **LR** - licznik rozkazów - zlicza rozkazy i uklada je w pamięci aby następnie odczytane przez **RR**

- A oraz B - rejstry argumentów pobranych z pamięci

Typowa organizacja procesora to blok rejestrów, blok ALU i dekoder kodu rozkazowego. Najważniejszym układem procesora jest blok arytmetyczno logiczny ALU wykonujący operacje na argumentach z dwóch rejestrów A i B. Cykl pracy procesora rozpoczyna się od wysłania do pamięci adresu rozkazu. Adres ten znajduje się, w rejestrze LR zwanym licznikiem rozkazów.

Odczytywany z pamięci rozkaz zostaje przesłany do rejestru rozkazów RR. Zawartość tego rejestru jest dekodowana i blok ALU zostaje odpowiednio wysterowany do wykonania danej operacji. Zarówno rozkazy procesora jak i argumenty tych rozkazów są przedstawiane w komputerze w postaci słów binarnych, tj. kodowane w zapisie dwójkowym. (długość słowa zawsze jest taka sama i odpowiada bitowości komputera tj. 8-bitow, 16-bitow, 32-bity itd)



Pamięć jest podzielona na komórki, w których są przechowywane pojedyncze słowa (abajty). Każda komórka ma swój adres i podanie tego adresu na wejście adresowe pamięci umożliwia dostęp do danej komórki, czyli odczyt lub zapis. W zależności od sygnału O (odczyt) / Z (zapis) pamięć jest odczytywana lub zapisywana.

Wielkość takiej pamięci nazywana jest pojemnością pamięci i jest oznaczana przez $n \times 8$ (liczba pamiętanych słów przez długość słowa). W jednym cyku pracy takiej pamięci można odczytać lub zapisać tylko słowo 8-bitowe. W przypadku, gdy długość rozkazu lub argumentu jest większa, to jest on zapisywany w dwóch (lub więcej) komórkach pamięci. Cykl instrukcyjny składa się z 4 faz:

- fazy pobrania rozkazu
- dwóch faz pobrania argumentów rozkazu

- fazy zapisu wyniku do pamięci.

9. Reprezentacja liczb w pozycyjnym systemie liczbowym. Systemy dwójkowy i szesnastkowy oraz ich zastosowania.

Pozycyjny system liczbowy - to system liczbowy który opiera się o pewien skonczony zestaw cyfr tego systemu. Każda liczba w systemie pozycyjnym zostaje przedstawiona jako ciąg cyfr tego systemu gdzie każda następna cyfra k na indeksie n ($n \in \{0, 1, 2, \dots\}$) gdzie m to ilość cyfr tego systemu, przedstawia wartość $k \times m^n$.

Wzór:

$$k_n \cdot m^n + k_{n-1} \cdot m^{n-1} + \dots + k_1 \cdot m^1 + k_0 \cdot m^0$$

Przykłady

Zamiana z (2) , (16) na (10) \$

- $2137_{(10)} = 2 \cdot 10^3 + 1 \cdot 10^2 + \dots + 3 \cdot 10^1 + 7 \cdot 10^0 = 2137_{(10)}$
- $11001000_{(2)} = 1 \cdot 2^7 + 1 \cdot 2^6 + 0 \cdot 2^5 + 0 \cdot 2^4 + 1 \cdot 2^3 + 0 \cdot 2^2 + 0 \cdot 2^1 + 0 \cdot 2^0 = 200_{(10)}$
- $1A4_{(16)} = 1 \cdot 16^2 + A \cdot 16^1 + 4 \cdot 16^0 = 420_{(16)}$

Zamiana z (10) na (2) , (16) \$

- Zamiana z (10) na (16) \$

Divide by the base 16 to get the digits from the remainders:			
Division by 16	Quotient	Remainder (Digit)	Digit #
$(420)/16$	26	4	0
$(26)/16$	1	10	1
$(1)/16$	0	1	2
$= (1A4)_{16}$			

- **Zamiana z (10) na \$(2)\$**

Divide by the base 2 to get the digits from the remainders:

Division by 2	Quotient	Remainder (Digit)	Bit #
$(200)/2$	100	0	0
$(100)/2$	50	0	1
$(50)/2$	25	0	2
$(25)/2$	12	1	3
$(12)/2$	6	0	4
$(6)/2$	3	0	5
$(3)/2$	1	1	6
$(1)/2$	0	1	7
$= (11001000)_2$			

Zastosowanie systemu dwójkowego

System dwójkowy oznaczany $XX.X_{(2)}$ wykorzystywany jest jako główny podłożo wszystkich obliczeń komputerów elektronicznych. Został wybrany na system pozycyjny komputerów ponieważ 0 może przedstawiać brak napięcia a 1 napięcie na danej sciezce, w danej komorce pamięci lub nosziku danych co sprawia że jest to system bardzo prosty ponieważ nie wymaga pomiaru poziomu napięcia prądu przepływającego przez komponent w celu uzyskania danych z tego komponentu.

- używany w matematyce, informatyce i elektronice cyfrowej, gdzie minimalizacja liczby stanów (do dwóch) pozwala na prostą implementację sprzętową odpowiadającą zazwyczaj stanom wyłączony i włączony oraz zminimalizowanie przekłamań danych

Zastosowanie systemu szesnastkowego

System szesnastkowy oznaczany $XX.X_{(16)}$ wykorzystywany jest jako rozszerzenie systemu dwójkowego. Dzięki większej podstawie tego systemu, liczby zapisywane w nim są bardziej kompaktowe i dzięki temu bardziej czytelne.

- adresy sprzętowe
- kody kolorów RGB

- adresy IPv6

10. Arytmetyka stałopozycyjna i zmiennopozycyjna. Reprezentacja liczb w komputerze.

Liczba stałopozycyjna (stałoprzecinkowa)

Komputerowa reprezentacja liczb całkowitych z przedziału od -2^n do 2^{n-1} (przedział zależy od standardu), gdzie n jest liczbą bitów w słowie maszynowym, zapisywanych w kodzie uzupełnien do dwóch. Zakres liczb 16-bitowych w a.s. (komputery PC) mieści się w przedziale $[-32768, +32767]$. Przekroczenie zakresu liczb powoduje nadmiar. W arytmetyce stałopozycyjnej są wykonywane cztery podstawowe działania (+, -, * i /), przy czym stosuje się dzielenie całkowite.

Liczba zmiennopozycyjna (zmiennoprzecinkowa)

Reprezentacja liczby rzeczywistej zapisanej za pomocą notacji naukowej. Ze względu na wygodę operowania na takich liczbach, przyjmuje się ograniczony zakres na mantysę i cechę – nazwy te mają w matematyce znaczenie podane w artykule podloga i sufit, a w niniejszym artykule inne, powszechnie w informatyce. Powoduje to, że reprezentacja liczby rzeczywistej jest tylko przybliżona, a jedna liczba zmiennoprzecinkowa może reprezentować różne liczby rzeczywiste z pewnego zakresu.

Stalopozycyjne (całkowite)

Stalopozycyjne (całkowite)

• kod ZM (znak-moduł)

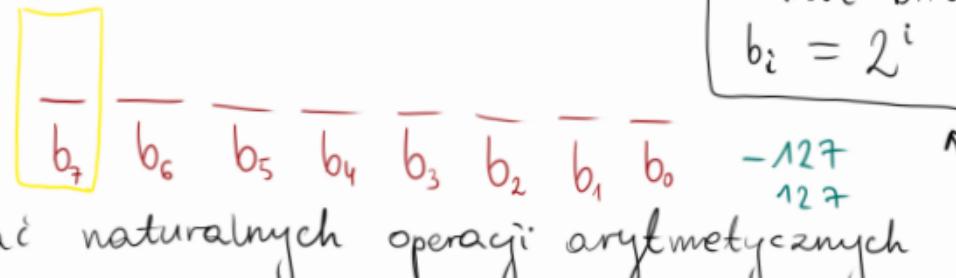
WADA: +0 i -0,

nie można stosować naturalnych operacji arytmetycznych

najstarszy
bit (znak)

$$(-1)^0 \quad (-1)^1$$

WARTOŚĆ BITU
 $b_i = 2^i$



• kod U2 (uzupełnieni do dwóch)

najstarszy bit ma wagę -2^{n-1} pozostałe

ZALETĄ: naturalne dodawanie i odejmowanie

? jak zapisać liczbę ujemną?

① zapisz liczbę dodatnią ② dokonaj inwersji bitów ($0 \rightarrow 1$) ③ dodaj "1"

$$\begin{array}{r} 0 & 0 & 0 & 1 & 1 \\ \underline{1} & \underline{1} & \underline{1} & 0 & 0 \\ 1 & 1 & 1 & 0 & 1 \end{array}$$

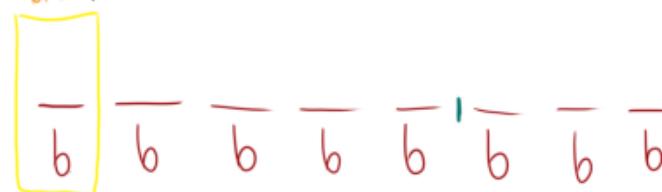
? jak odczytać liczbę ujemną?

$$\begin{array}{r} 1 & 0 & 1 & 0 & 1 \\ \underline{2^4} & \underline{2^3} & \underline{2^2} & \underline{2^1} & \underline{2^0} \end{array}_{U2} = -2^4 + (2^2 + 2^0) = -16 + 5 = -11_{10}$$

Stałopozycyjne (rzeczywiste)

• kod U2

WADY: zwiększenie

najstarszy
bit (znak)

WARTOŚĆ BITU
 $b_i = 2^i$

dokładności części całkowitej reprezentowanej liczby
 odbywa się ZAWSZE kosztem dokładności części ułamkowej
 i ODWROTNIE.

Dodanie liczby ujemnej, to odjęcie liczby dodatniej.

Odejmowanie liczby ujemnej, to dodanie liczby dodatniej.

DODAWANIE

$$7,75_{10} = 00111,110$$

$$0,25_{10} = 00000,010$$

$$\begin{array}{r} 00111,110 \\ + 00000,010 \\ \hline 01000,000_2 = 8_{10} \end{array}$$

ODEJMOWANIE

$$6,75_{10} = 00110110$$

$$1,25_{10} = 00001010$$

$$\begin{array}{r} 00110110 \\ - 00001010 \\ \hline 00101100 \\ 5,5_{10} \end{array}$$

[WAŻNE:
 POZYCZKA]

Stalopozycyjne (rzeczywiste cd.)

Stalopozycyjne (rzeczywiste cd.)

! MNOżENIE w U2:

→ rozszerzamy znakowo obie liczby (zwiększymy dwukrotnie lub względem dłuższej liczby)
POWIELAMY BIT ZNAKU

$$-2 \text{ (4 bity)} \rightarrow \boxed{1}110 \rightarrow \underline{1111}1110$$

$$3 \text{ (4 bity)} \rightarrow \boxed{0}011 \rightarrow \underline{0000}0011$$

$$\begin{array}{r} 11111110 \\ \times 00000011 \\ \hline 11111110 \\ + 11111110 \\ \hline \cancel{10}11111010 \end{array}$$

poza reprezentacją

! DZIelenie w U2:

① ZAPAMIĘTAJ ZNAKI DZIELONYCH LICZB ② ZAMIEŃ UJEMNE NA DODATNIE

③ CYKLICZNE ODEJMOWANIE PRZESUNIĘTEGO DZIELNIKA OD DZIELNEJ
(TAM GDE SIE DA)

④ JEŚLI ZNAK BYŁ ZMIENIONY RAZ → ZMIENIAMY ZNAK WYNIKU

$$13_{10} = 001101_{U2}$$

$$[-2]_{10} = 1110_{U2}$$

$$\hookrightarrow 2_{10} = 0010_{U2}$$

$$\begin{array}{r} 110 \\ \hline 001101 : 0010 \\ - 0010 \\ \hline 000101 \\ - 0010 \\ \hline 000001 \end{array}$$

RESZTA

④ TYLKO ZNAK

WYNIK: 1110

Zmiennopozycyjne:

- UŻYWANE DO REPREZENTACJI DUZYCH LUB BARDZO MAŁYCH LICZB

W POSTACI WYKŁADNICZEJ O PODSTAWIE $\underline{\underline{2}}$

* IEEE754 - standard zapisu zmiennopozycyjnego powszechny w procesorach i oprogramowaniu



PRZYKŁADOWA NOTACJA: $z \cdot M \cdot 2^{C-S}$ $S \rightarrow \text{stała (BIAS)}$ $1 \leq M < 2$

$C, S, C-S \rightarrow \text{liczby całkowite}$

np: $S=6$

$$0,625_{10} = 0,101_2 = 1,01_2 \cdot 2^{-1} = +1,0\underline{01}_2 \cdot 2^{5-6}$$

$$C = 5_{10} = 101_2$$

1 bit 8 bitów 23 bitów

$$1,6M,9C$$

$S=9$

$$0,0224609375_{10} = 0,0000010111_2 = 1,0111 \cdot 2^{-6} = 1,0111 \cdot 2^{3-9} = +1,0\underline{11}_2 \cdot 2^{3-5}$$

$$3_{10} = 11$$

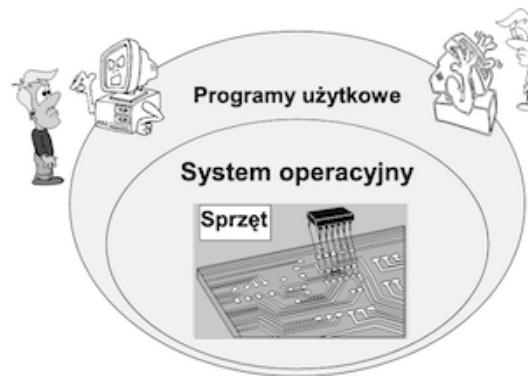
$$0 00000110111000 \dots 0$$

11. System operacyjny. Postrzeganie systemu operacyjnego przez warstwę oprogramowania użytkowego.

Źródło: wykład "Wprowadzenie do systemów operacyjnych"

Definicja

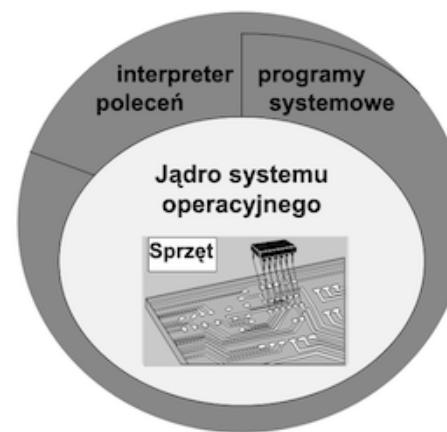
System operacyjny jest warstwą oprogramowania operującą bezpośrednio na sprzęcie, której celem jest zarządzanie zasobami systemu komputerowego i stworzenie użytkownikowi środowiska łatwiejszego do zrozumienia i wykorzystania.



System operacyjny *pośredniczy* pomiędzy użytkownikiem a sprzętem, dostarczając wygodnego środowiska do wykonywania programów. Użytkownik końcowy korzysta z programów (aplikacji), na potrzeby których przydzielane są zasoby systemu komputerowego. Przydziałem tym zarządza system operacyjny, dzięki czemu można uzyskać stosunkowo duży stopień niezależności programów od konkretnego sprzętu oraz odpowiedni poziom bezpieczeństwa i sprawności działania.

Ogólna struktura systemu operacyjnego

Nie ma precyjnego określenia, które składniki wchodzą w skład systemu operacyjnego jako jego części.



W ogólnym przypadku w strukturze systemu operacyjnego wyróżnia się jądro oraz programy systemowe, które dostarczane są razem z systemem operacyjnym, ale nie stanowią integralnej części jądra. **Jądro** jest zbiorem modułów, które ukrywają szczegóły sprzętowej realizacji systemu komputerowego, udostępniając pewien zestaw usług, wykorzystywanych między innymi do implementacji programów systemowych.

Z punktu widzenia kontaktu z użytkownikiem istotny jest **interpreter poleceń**, który może być częścią jądra lub programem systemowym (np. w systemie UNIX). Interpreter wykonuje pewne polecenia wewnętrznie, tzn. moduł lub program interpretera dostarcza implementacji tych poleceń. Jeśli interpreter nie może wykonać wewnętrznie jakiegoś polecenia, uruchamia odpowiedni program (tzw. polecenie zewnętrzne), jako odrębny proces.

Programy systemowe (programy użytkowe systemu):

- programy do obsługi plików, w tym pakujące i archiwizujące
- programy do komunikacji w sieci
- proste edytory tekstów i grafiki
- programy diagnozujące pracę procesora, pamięci, sieci, dysków twardych itp
- kompilatory

Zadania systemu operacyjnego:

- Definicja interfejsu użytkownika
- Udostępnianie systemu plików
- Udostępnianie środowiska do wykonywania programów użytkownika
 - mechanizm ładowania i uruchamiania programów
 - mechanizmy synchronizacji i komunikacji procesów
- Sterowanie urządzeniami wejścia-wyjścia
- Obsługa podstawowej klasy błędów

Podział systemów operacyjnych

- Ze względu na **sposób przetwarzania**
 - systemy przetwarzania bezpośredniego (online processing systems) - bezpośrednią interakcję użytkownik-<->system
 - systemy przetwarzania pośredniego (offline processing systems) - zwłoka czasowa; brak możliwości ingerencji w wykonywanie zadania
- Ze względu na **liczbę wykonywanych programów**
 - jednozadaniowe - tylko jedno zadanie na raz
 - wielozadaniowe - wiele zadań jednocześnie
- Ze względu na **liczbę użytkowników**
 - dla 1 użytkownika - tylko jedno zadanie na raz
 - wielozadaniowe - wielu użytkowników może korzystać ze zasobów systemu komputerowego, a system operacyjny gwarantuje ich ochronę przed nieupoważnioną ingerencją

Procesy

Proces - uruchomiony program. Jeden program to może być wiele procesów, bo np. uruchomimy wiele razy ten jeden program. Każdy proces jest identyfikowany przez numer PID.

W systemie operacyjnym każdy proces posiada proces nadrzędny (rodzica), z kolei każdy proces może, poprzez wywołanie funkcji systemu operacyjnego, utworzyć swoje procesy potomne. W ten sposób tworzy się swego rodzaju drzewo procesów.

W skład procesu wchodzi:

- kod programu
- licznik rozkazów
- stos
- sekcja danych

W trakcie ładowania procesu do pamięci system operacyjny tworzy *stos* (stack) i *sterę* (heap).

Stos – do przechowywania zmiennych, parametrów funkcji, adresów powrotu. **Sterta** – do przechowywania dynamicznie alokowanych danych, np. listy

Stany procesu:

- **Początkowy** (initial) – w trakcie uruchamiania
- **Aktywny** (running, executing) – proces działa na procesorze
- **Gotowy** (ready) – proces jest gotowy do uruchomienia, ale w tej chwili jest wstrzymany
- **Oczekujący** (blocked) ; proces wykonał operację w wyniku której nie może zostać ponownie uruchomiony dopóki nie nastąpi jakieś zdarzenie (np. operacja we/wy).
- **Końcowy** (final) – podczas zamykania
- **Zombie** – ukończony proces, który czeka na jakąś akcję, np. odczytanie kodu wyjścia przez proces rodzica
- **Demon** (ang. daemon czyli duszek) - nazywamy proces działający w tle, nie podlegający sterowaniu z żadnego terminala, uruchamiany zwykle podczas startu systemu i działający do jego zamknięcia.

Wątek

Czasami może być konieczne współbieżne wykonywanie pewnych fragmentów programu. Aby to zrealizować, program może zażądać utworzenia określonej liczby wątków, wykonujących wskazane części programu. Ta cecha systemu operacyjnego to wielowątkowość. W jednym procesie może być kilka wątków. Każdy wątek ma swój własny stos (posiada swoje zmienne lokalne)

Wielozadaniowość

Cechą systemu operacyjnego umożliwiającą równoczesne wykonywanie więcej niż jednego procesu (programu).

Planista (dyspozytor)

Jest jak policjant na skrzyżowaniu, który wskazuje, które auta mogą teraz przejechać przez skrzyżowanie. Jest to część systemu operacyjnego przełączająca procesy według polityki szeregowania zadań. Do jego zadań należy [m.in.](#) przełączanie kontekstu.

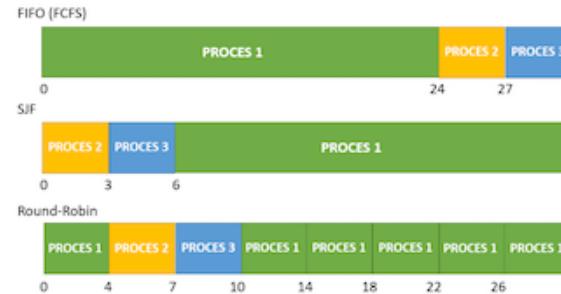
Planista krótkoterminowy ustala wartość priorytetu. Wybiera proces o najwyższym priorytecie do wykonania.

Algorytmy szeregowania

- **FIFO** – (FCFS) - najprostszy, niewywłaszczający, implementowany za pomocą kolejki FIFO;
- **SJF** (Shortest-Job-First) - wiąże z każdym procesem długość jego najbliższej z faz procesora, zapewnia minimalny średni czas oczekiwania; może być wywłaszczający lub nie;

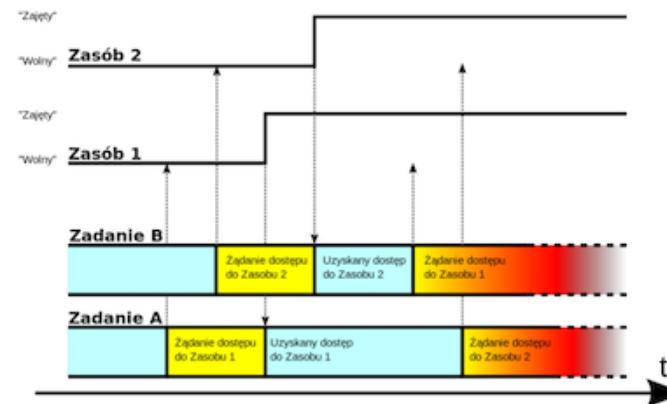
- algorytm **Round-Robin** - czas procesora podzielony na kwanty, kolejka procesów gotowych traktowana jako kolejka cykliczna, algorytm z wywłaszczeniem;

Algorytmy szeregowania - przykład



Możliwe jest **zagłodzenie** procesu, gdy dany proces nie jest w stanie zakończyć działania, ponieważ nie ma dostępu do procesora lub innego współdzielonego zasobu. Występuje najczęściej na skutek niewłaściwej pracy algorytmu szeregowania lub nadmiernego obciążenia systemu.

Zdarza się również tzw. **zakleszczenie**, czyli blokada wzajemna. Powstaje wtedy, gdy wiele zadań w tym samym czasie konkuuuje o wyłączny dostęp do zasobów. Zakleszczenie:



12. Cechy tradycyjnego systemu Unixowego.

Polecanko - slajd 46+

TL;DR: Unix to system:

- wielozadaniowy** - jest to cecha systemu operacyjnego, mówiąca, czy może on wykonywać jednocześnie kilka procesów. Wielozadaniowość otrzymuje się poprzez tzw. scheduler, czyli algorytm kolejkujący i porządkujący procesy, które mają być wykonane. W tym systemie każdy proces jest wykonywany jakiś kwant czasu, a później czeka "w uśpieniu" (oczywiście z uwzględnieniem różnych priorytetów).

- **wielodostępowy** (o ile jego administrator nie zażyczy sobie inaczej) - oznacza możliwość jednoczesnej pracy wielu użytkowników (np. możliwość czytania tego samego pliku przez kilku użytkowników)
- **wielowątkowy** – jest to cecha systemu operacyjnego, dzięki której w ramach jednego procesu można wykonywać kilka wątków lub jednostek wykonawczych. Nowe wątki to kolejne ciągi instrukcji wykonywane oddzielnie. Wszystkie wątki współdzielą między sobą ten sam obszar pamięci. W momencie, gdy system nie wspiera wielowątkowości, pojęcie procesu i wątku utożsamiają się.

Cechy

1. **Jądro** systemu jest, oprócz pewnej części ściśle związanego z obsługiwanyem sprzętem, **napisane w języku wysokiego poziomu (C)**.
2. W Unixie obowiązuje model administracyjny, bazujący na **ograniczonym zaufaniu do użytkowników**. Ujawnia się on między innymi tym, że zwykle użytkownik lokalny ma prawo zapisu jedynie w swoim katalogu domowym, katalogu na pliki tymczasowe oraz w kilku innych, dobrze znanych miejscach. Jednocześnie administratora systemu (użytkownika o numerze identyfikacyjnym 0) nie dotyczą jakiekolwiek ograniczenia.
3. **System praw dostępu do plików** (czyli również do urządzeń czy kanałów komunikacyjnych) jest zbudowany w oparciu o tablicę bitową stałej długości, zapisaną w i-węźle. Zawiera ona zezwolenia na trzy podstawowe operacje – czytanie, zapis i wykonanie dla trzech różnych klas użytkowników: właściciela pliku, członków tzw. grupy pliku oraz innych. Unixowy system praw dostępu jest bardzo efektywny w działaniu, brak dynamicznych list dostępu jest jednakże dość uciążliwy.
4. Unix bezpośrednio po starcie widzi tylko jedno urządzenie pamięci masowej, zawierające tzw. korzeń systemu plików (oznaczany znakiem /). Inne urządzenia są przyłączane do głównego drzewa w procesie tzw. montowania i są widoczne jako fragmenty drzewa plikowego od pewnego katalogu określonego jako punkt montowania.
5. Naturalnym sposobem **organizacji pamięci masowej** jest model indeksowy oparty na tzw. i-węzłach (ang. i-nodes). i-węzeł zawiera w postaci tablicy o stałym rozmiarze wszystkie informacje o pliku poza jego nazwą. Odwzorowaniem i-węzłów na nazwy plików zajmują się pliki specjalne – katalogi.
6. Budowie interfejsu programisty systemu (API) prześwięca minimalizm, ujawniający się choćby tym, że odczyt i zapis informacji w rozmaitych urządzeniach obsługiwanych przez system odbywa się za pomocą tego samego interfejsu jak odczyt i zapis informacji do plików „zwykłych”. Zasadę tę często definiuje się jako: „Dla Unixa wszystko jest plikiem”.
7. Jednostką aktywną w systemie jest **proces**, pracujący w trybie nieuprzywilejowanym procesora, we własnej chronionej przestrzeni adresowej; jedynym elementem aktywnym w trybie uprzywilejowanym jest jądro systemu.
8. Unix wykorzystuje do pracy w środowisku rozproszonym rodzinę protokołów TCP/IP.
9. Plik danych jest ciągiem bajtów.
10. Unix używa pamięci wirtualnej, rozszerzając pamięć operacyjną o tzw. obszary wymiany w pamięci masowej. Niewykorzystaną pamięć operacyjną wypełniają bufora używanych plików.
11. Podstawową metodą tworzenia nowych procesów jest rozwidlanie procesu aktywnego funkcją systemową fork. Po jej wywołaniu system tworzy nowy proces, którego przestrzeń adresowa jest kopią przestrzeni procesu macierzystego. Oba procesy rozpoczynają wspólnie pracę od następnej instrukcji za wywołaniem fork. Często proces potomny wykonuje

niedługo po utworzeniu funkcję systemową execve, która zastępuje kod aktywnego procesu kodem z pliku wykonywalnego.

12. Otwarty plik jest dostępny w procesie poprzez liczbę całkowitą zwaną **deskryptorem pliku**. Predefiniowanymi deskryptorami są tu wartości 0 (standardowe wejście, zwykle związane z klawiaturą terminala), 1 (standardowe wyjście, zwykle związanego z wyjściem terminala) oraz 2 (standardowe wyjście dla błędów).
13. W środowisku tekstowym naturalnym środowiskiem pracy jest tzw. Interpreter poleceń czyli powłoka (ang. shell).
14. Unixowy system plików jest widoczny jako wielopoziomowe drzewo.
15. Procesy korzystają podczas pracy z mechanizmów łączenia dynamicznego, ładując kod wspólnych bibliotek w miarę potrzeb. Podstawową biblioteką uwspólnioną jest standardowa biblioteka języka C (tzw. libc).
16. Komunikacja międzyprocesowa odbywa się przez jądro systemu.

Typy plików Unixowych

- **pliki zwykłe** – (symbol: -) ciągi bajtów, może istnieć w kilku miejscach w systemie plików jednocześnie.
- **katalogi** – (symbol: d) plik binarny zawierający listę plików oraz katalogów, które się w nim znajdują. Typowe operacje dostępu do pliku, np. otwarcie, nie działają dla katalogu. Dowiązania sztywne do katalogu są tworzone jedynie pośrednio przez system. Każdy katalog zawiera dwie specjalne pozycje:
 - . – wskazującą na ten katalog
 - .. – wskazującą na katalog zawierający.
- **dowiązanie symboliczne**, (ang. symbolic link, często skracane jako symlink) wskazuje, odwołując się za pomocą nazwy, na dowolny inny plik lub katalog (który może nawet w danej chwili nie istnieć). Odwołanie jest niewidoczne na poziomie aplikacji tzn. jest traktowane jak zwykły plik lub katalog.

13. Iteracja, rekurencja i ich realizacja.

Źródło

Iteracja - czynność powtarzania (najczęściej wielokrotnego) tej samej instrukcji (albo wielu instrukcji) w pętli.

Rekurencja to w logice, programowaniu i w matematyce odwoływanie się np. funkcji lub definicji do samej siebie.

Najwięcej problemów związanych z rekurencją wiąże się z ograniczeniami stosu wywołań, a właściwie jego pojemności. Na stosie są odkładane kolejne wywołania danej metody i dopiero gdy dojdziemy do ostatniego elementu dane te są zbierane – bardzo łatwo więc o sytuację, gdy po prostu stos przepelnimy.

Klasyczne przykłady

Silnia iteracyjnie: $n! = 1 * 2 * 3...* n$

Silnia rekurencyjnie: $n! = n * (n-1)!$

Ciąg Fibonacciego

Definicja: dla $n > 1$ mamy

$$fib_n = fib_{n-1} + fib_{n-2},$$

natomiast wyrazy 1 i 0 przyjmują wartość 1.

Fibonacci rekurencyjnie:

```
function FibR(n)
begin
    if ( n=0 or n=1) then {
        return 1
    }
    return FibR(n-1) + FibR(n-2)
end
```

Fibonacci iteracyjnie:

```
function FibI(n)
begin
    tmp :=0 // zmienna tymczasowa (pomocnicza)
    x := 1 // wyraz n-1
    y := 1 // wyraz n-2

    for i:=1 to n-1 step 1 {
        tmp := y // zapamiętaj wyraz n-2
        y := y+x // przesuń wyraz n-2 na kolejną wartość ciągu
        x := tmp // przesuń wyraz n-1 na kolejną wartość ciągu
        // czyli na wartość wyrazu n-1 przed jego
        // przesunięciem
    }
    return x
end
```

14. Mechanizmy strukturalizacji programów - instrukcje warunkowe i pętle.

Switch syntax

```
switch(expression) {  
    case x:  
        // code block  
        break;  
    case y:  
        // code block  
        break;  
    default:  
        // code block  
}
```

Note: The default keyword must be used as the **last statement** in the switch, and it **does not need a break**. *default* can be the first statement on the list, but it makes no sense since only this statement will be executed.

break and **default** keywords are *optional*.

Without a break statement, **every statement** from the matched case label to the end of the switch, including the default, is executed.

switch statement behavior

Condition	Action
Converted value matches that of the promoted controlling expression.	Control is transferred to the statement following that label.
None of the constants match the constants in the case labels; a default label is present.	Control is transferred to the default label.
None of the constants match the constants in the case labels; no default label is present.	Control is transferred to the statement after the switch statement.

[Switch documentation](#)

15. Podprogramy. Przekazywanie parametrów podprogramu.

Podprogramy – wydzielona część programu wykonująca określony zbiór instrukcji, posiadająca swoją nazwę i stanowiąca pewną odrębną całość. Ich nazwy powinny informować o ich wyniku działania.

Ogólnie przyjęta konwencja (w przypadku C++) typ_rezultatu nazwa_funkcji(lista parametrów formalnych); na przykład:

```
bool isPrime(int);
```

Podprogramy dzielą się na dwa rodzaje:

- funkcje, które operują na otrzymanych w parametrach danych, zwracają pewną obliczoną wartość, ale nie ingerują w działanie programu.
- procedury mogą przyjmować parametry, a w wyniku ich działania następują zmiany globalne w programie bądź w otrzymanych parametrach; po swoim działaniu nie zwraca niczego.

Innym szczególnym przypadkiem są *metody* – funkcje, które są własnością klasy lub obiektu. Bez ich istnienia nie można się do nich odwołać.

W niektórych językach programowania nie istnieje powyższy podział.

Jeżeli chodzi o C++, formalnie procedury nie istnieją, jednak łatwo się domyślić, że ustawiając jako typ rezultatu void możemy utworzyć coś na jej wzór.

Przekazywanie parametrów do podprogramów odbywa się głównie na dwa sposoby:

- **przez wartość** – wewnątrz bloku podprogramu tworzona jest zmienna lokalna, do której kopiuowana jest wartość przekazanego parametru, a następnie wszystkie operacje wykonywane są na kopii. Po zakończeniu działania podprogramu wszystkie kopie przestają istnieć, zaś oryginalna zmienna pozostaje niezmieniona
- **przez referencję** – do podprogramu przekazywany jest bezpośredni dostęp do zmiennej, a nie jedynie jej wartość, co zmniejsza zużycie pamięci oraz umożliwia modyfikację zmiennej, której efekty pozostaną także po zakończeniu działania podprogramu.

```
int addOne(int number) {
    return number++; //przez wartość
}

int addOne(int &number) {
    return number++; //przez referencję
}
```

16. Porównanie programowania obiektowego i strukturalnego.

Programowanie strukturalne – paradygmat programowania opierający się na podziale kodu źródłowego programu na procedury i hierarchicznie ułożone bloki z wykorzystaniem *struktur kontrolnych* w postaci instrukcji wyboru i pętli. Język programowania zgodny z paradygmatem programowania strukturalnego nazywa się językiem strukturalnym.

Struktury kontrolne:

- **Sekwencja** – wykonanie ciągu kolejnych instrukcji.
- **Wybór** – w zależności od wartości wykonywana jest odpowiednia instrukcja. W większości języków programowania są to instrukcje takie jak: if, else, switch, case.
- **Iteracja** – wykonywanie instrukcji póki spełniony jest jakiś warunek. Reprezentowana w różnych wariantach jako pętle oznaczane między innymi przez: while, for, do (...) while.
- **Podprogramy** - pozwalają na wydzielenie pewnej grupy instrukcji i traktowania ich jako pojedynczej operacji, są dodatkowo mechanizmem abstrakcji.

- **Bloki** - w językach programowania bloki odpowiadają sekwencjom instrukcji, umożliwiając budowanie programu przez komponowanie struktur kontrolnych – w miejscu, w którym umieścimy blok z instrukcjami jest on traktowany jak pojedyncza instrukcja.

Programowanie obiektowe – paradymat programowania, w którym programy definiuje się za pomocą obiektów – elementów łączących stan i zachowanie. Obiektowy program komputerowy wyrażony jest jako zbiór takich obiektów, komunikujących się pomiędzy sobą w celu wykonywania zadań.

	Zalety	Wady
Programowanie strukturalne	<ul style="list-style-type: none"> • szybkie wykonywanie skryptów • mała liczba zmiennych (oszczędność pamięci) • prostota formy kodu 	<ul style="list-style-type: none"> • trudna modyfikacja kodu • rozdzielenie danych i operacji na nich wykonywanych • częsty nieporządek w kodzie, utrudniający jego czytanie
Programowanie obiektowe	<ul style="list-style-type: none"> • wysoka przejrzystość kodu • łatwość modyfikacji, rozbudowy oraz konserwacji kodu • małe ryzyko błędów przy zmianach • ułatwia współpracę wielu programistów 	<ul style="list-style-type: none"> • wolniejszy czas wykonywania • częsta potrzeba wykorzystania nadmiaru kodu do zdefiniowania klas

17. Hermetyzacja danych - cechy klas obiektowych (pola, metody, poziomy prywatności danych).

■ aka enkapsulacja

Polega na **ukrywaniu informacji** - ukrywanie pewnych danych składowych lub metod w obiektach danej klasy tak, aby były one dostępne tylko dla metod wewnętrznych danej klasy lub dla metod z klas z nią zaprzyjaźnionych.

Z pełną enkapsulacją mamy do czynienia wtedy gdy dostęp do wszystkich pól w klasie jest możliwy tylko i wyłącznie poprzez metody, lub inaczej: gdy wszystkie pola w klasie znajdują się w sekcji prywatnej (lub chronionej)

Ukrywanie wewnętrznej struktury obiektu jest bardzo ważne z kilku powodów (Zalety):

- obiekt taki jest odizolowany, a więc nie jest narażony na celowe, bądź niezamierzone działanie ze strony użytkownika
- obiekt ten jest chroniony od niepożądanych referencji ze strony innych obiektów.
- dzięki ukryciu wewnętrznej struktury obiektu, można uzyskać jego przenośność. Innymi słowy, zastosować definiującą go klasę w innym fragmencie kodu, czy też programie.
- Uodparnia tworzony model na błędy polegające na przykład na błędnych przypisywaniach wartości oraz umożliwia wykonanie czynności pomocniczych
- Umożliwia rozbicie modelu na mniejsze elementy.

Klasa to definicja obiektu, zawierająca stan obiektu, określony wartościami pól, oraz możliwe zachowanie, określone dostępnymi metodami.

Obiekt to utworzony egzemplarz (instancja) określonej klasy, który posiada własny, indywidualny stan i zbiór zachowań.

Metoda to funkcja lub procedura, skojarzona z ogółem klasy lub poszczególnymi jej obiektami; określa możliwe zachowania

Pole (Właściwość) to zmienna dowolnego typu, skojarzona z ogółem klasy lub poszczególnymi jej obiektami; określa aktualny stan obiektu

Dziedziczenie to mechanizm definiowania nowej klasy na bazie już istniejącej, wzbogacając ją o nowe pola, metody lub zmieniając zakres ich widoczności.

Zmienna składowa lub metoda może mieć trzy różne poziomy dostępnosci:

- **public** - dostęp publiczny dla innych struktur/klas i funkcji
- **private** - dostęp prywatny, tylko dla metod struktury/klasy
- **protected** - dostęp chroniony, dla metod struktury/klasy i jej struktur/klas pochodnych (dziedziczenie)
- **friend** - deklaracja przyjazni, dostęp dla wyspecyfikowanych struktur/klas lub funkcji

Struct - wszystkie składowe (pola i metody) są domyślnie **publiczne**

Class - wszystkie składowe (pola i metody) są domyślnie **prywatne**

18. Typy metod: konstruktory, destruktory, selektory, zapytania, iteratory.

Metoda

- Funkcja składowa
- Funkcja powiązana z klasą
- Funkcja mająca dostęp do składowych klas
- Tworzy interfejs klas

```
class MyClass {  
    void privateMethod(); //deklaracja w klasie  
  
public:  
    void publicMethod() { //definicja w klasie  
        //do nothing;  
    };  
}
```

Definicja klasy musi zawierać przynajmniej deklarację metody

Definicja metody, często dla czytelności kodu, jest umieszczana poza klasą

```
void MyClass::privateMethod() { //definicja poza klasą  
    //this method is depressed
```

}

Konstruktory

- **Konstruktor** – specjalna metoda do inicjalizacji atrybutów obiektu
- uruchamiany zawsze przy tworzeniu obiektu
- nazwa metody taka jak nazwa klasy
- możliwe wiele przeciążeń z różnymi parametrami, **brak typu wynikowego!**

```
struct Obj {  
    int a, b; //od C++11 możliwa inicjalizacja w klasie, np.  
    //int a = 0;  
  
    Obj(int _a = 0, int _b = 0){ //konstruktor  
        a = _a;  
        b = _b;  
    }  
};  
  
{  
    Obj x, y(1), z(1,2), v = 3, u = {3,4}; //wywołania konstruktora  
    Obj t[5], *s = new Obj, *p = new Obj[3]; //wielokrotne  
}
```

Konstruktor domyślny:

- bez parametrów (lub gdy wszystkie parametry mają wartości domyślne)
- tworzony automatycznie przez kompilator (bez gwarancji inicjalizacji atrybutów obiektu) tylko gdy nie zdefiniowano jawnie żadnego konstruktora

```
struct Obj {  
    int a, b;  
  
    Obj(){ //konstruktor domyślny  
        a = 0;  
        b = 0;  
    }  
  
    Obj()=default; //konstruktor domyślny bez inicjalizacji(C++11)  
};  
  
{
```

```
Obj x, t[5]; //wywołania konstruktora domyślnego  
Obj *s = new Obj, *p = new Obj[3]; //tu też  
}
```

- (!!!) jeżeli jakikolwiek konstruktor został zdefiniowany, kompilator nie tworzy domyślnego
- w celu ułatwienia wykorzystania klasy należy zawsze zdefiniować konstruktor domyślny

```
struct Obj {  
    int a, b;  
  
    Obj(int _a, int _b = 0){ //1- lub 2-parametry konstruktor  
        a = _a;  
        b = _b;  
    }  
};  
  
{  
    Obj y(1), z(1, 2); //wywołania konstruktora  
    Obj x; //brak konstruktora domyślnego - błąd kompilacji!  
}
```

Konstruktor kopiujący:

- przyjmuje jako jedyny parametr referencję do obiektu tej samej klasy
- tworzony automatycznie przez kompilator gdy nie zdefiniowano jawnie takiego konstruktora (kopia atrybutów obiektu)
- wywoływany przy przekazywaniu i zwracaniu obiektów w funkcjach

```
struct Obj {  
    int a, b;  
    ... //inne konstruktory łącznie z domyślnym  
  
    Obj(const Obj &o){ //konstruktor kopiujący  
        a = o.a;  
        b = o.b;  
    }  
};  
  
{  
    Obj x;  
    Obj y(x), z = x; //wywołania konstruktora kopiującego  
}
```

Destruktory

- **destruktor** – specjalna metoda do zwolnienia pamięci zajmowanej przez obiekt
- wywoływany automatycznie gdy zmienna przestaje istnieć (na końcu bloku)
- metoda o nazwie takiej jak nazwa klasy lecz z poprzedzającą tyldą (~)
- **możliwy tylko jeden destruktor, brak typu wynikowego!**

```
struct Obj {  
    int a, b;  
    ... //konstruktory łącznie z domyślnym  
  
    ~Obj(){...} //destruktor  
    ~Obj()=default; //destruktor domyślny (C++11)  
};  
  
{  
    Obj x, *p = new Obj; //wywołania konstruktora  
    delete p; //jawne wywołanie destruktora (obiekt *p)  
} //niejawne wywołanie destruktora (obiekt x)
```

Destruktory obiektów:

- tworzone automatycznie przez kompilator gdy nie zostaną jawnie zdefiniowane (destruktor domyślny nie zwalnia pamięci dla atrybutów klasy alokowanych dynamicznie za pomocą operatora new)
- uruchamiane w kolejności odwrotnej do kolejności tworzenia obiektów (wywołań odpowiednich konstruktorów)

```
{  
    Obj x, *p = new Obj, z;  
  
    { Obj y; } //destruktor dla obiektu y  
  
    delete p; //destruktor dla obiektu *p  
} //destruktor dla obiektu z i dalej dla x
```

Kiedy wywoływany jest destruktor?

- Decyzję o wywołaniu destruktora podejmuje kompilator. Kod nie powinien jawnie wywoływać destruktora.
- Jeżeli obiekt tworzony jest w pamięci statycznej, wówczas jego destruktor wywoływany jest przed zakończeniem programu.
- Jeżeli obiekt tworzony jest w sposób automatyczny wówczas destruktor jest wywoływany kiedy program opuszcza blok kodu w którym został zdefiniowany ten obiekt.

- Jeżeli obiekt utworzono w sposób dynamiczny (tzn. za pomocą operatora new), wówczas destruktor tego obiektu jest wywoływany automatycznie, gdy użyjemy delete do zwolnienia pamięci.

Metody domyślne

Dla każdej klasy kompilator tworzy **automatycznie (o ile nie zdefiniowano ich jawnie)** następujące metody:

- konstruktor domyślny (bezparametryczny, brak inicjalizacji)
- konstruktor kopiący (kopia atrybutów)
- destruktor (brak zwolnienia dynamicznej pamięci)
- operator przypisania (kopia atrybutów)

```
struct Obj {  
    int a, b;  
};  
  
{  
    Obj x; //konstruktor domyślny, atrybuty są przypadkowe  
    Obj y = x; //konstruktor kopiący  
    x = y; //operator przypisania  
} //destruktor domyślny obiektów y i x
```

[^Zródło](#)

Iteratory

W programowaniu obiektowym jest to obiekt pozwalający na sekwencyjny dostęp do wszystkich elementów lub części zawartych w innym obiekcie, zwykle kontenerze lub liście.

Podstawowym celem iteratatora jest pozwolić użytkownikowi przetworzyć każdy element w kolekcji bez konieczności zagłębiania się w jej wewnętrzną strukturę. Np.: przejść do kolejnego elementu, na koniec na początek. Użytkownik nie musi np. zajmować się tym, że odwoła się do nieistniejącego elementu.

W C++ iteratory są szeroko wykorzystywane w bibliotece STL. Iteratory stosuje się zwykle w parach, gdzie jeden jest używany do właściwej iteracji, zaś drugi oznacza koniec kolekcji.

Iteratory tworzone są przez odpowiadający im kontener standardowymi metodami, takimi jak **begin()** i **end()**. Iteratator zwrócony przez *begin()* wskazuje na pierwszy element, podczas gdy iteratator zwrócony przez *end()* wskazuje na pozycję za ostatnim elementem kontenera.

```
int main() {  
    vector<int> ar = { 1, 2, 3, 4, 5 };  
  
    // Declaring iterator to a vector  
    vector<int>::iterator ptr;
```

```
// Displaying vector elements using begin() and end()
cout << "The vector elements are : ";
for (ptr = ar.begin(); ptr < ar.end(); ptr++)
    cout << *ptr << " ";

return 0;
}
```

Output:

The vector elements are : 1 2 3 4 5

Just in case: przeciążenie operatorów [slajd 7+](#) (Cybula) oraz [slajd 8+](#) (Wardowski).

Selektory

- Za jego pomocą możemy wywoływać metody klasy dla obiektu
- Jest on oznaczony za pomocą kropki
- Przykład użycia selektora:

```
x.set(4, 4.5);
```

Zapytania

- Kwerendy utworzone w języku zapytań
- Umożliwiają wyszukanie, tworzenie, usunięcie lub modyfikację danych w bazie danych
- Przykładem języka operującego na zapytaniach w bazach danych jest SQL (ang. Structured Query Language)

```
INSERT INTO TABLE osoby VALUES ('Jan', 'Kowalski');
SELECT imie FROM osoby WHERE nazwisko = 'Kowalski';
UPDATE osoby SET imie = 'Adam';
DELETE FROM osoby WHERE nazwisko = 'Kowalski';
CREATE TABLE osoby (imie VARCHAR(50), nazwisko VARCHAR(50));
```

- Inne mniej popularne to np. QBE (ang. Query By Example) czy XQuery

19. Dziedziczenie i dynamiczny polimorfizm.

Jest to mechanizm umożliwiający tworzenie nowych klas na podstawie klasy już istniejących w ten sposób, że nowa klasa przejmuje (dziedziczy) wszystkie metody drugiej klasy.

Zalety:

- Możliwość tworzenia kodu, który da się wielokrotnie wykorzystać.
- Lepiej jest stosować kod już sprawdzony, niż wymyślać rozwiązań od podstaw
- Korzystanie z istniejącego kodu nie tylko skraca czas programowania, lecz również pozwala uniknąć błędów.
- Im mniej koncentrujemy się na szczegółach tym bardziej zrozumiał i przejrzysty jest cały projekt.
- Wzbogacanie istniejących klas o dodatkową funkcjonalność oraz deklarowanie w nowych klasach dodatkowych danych.
- Modyfikowanie działań metod już istniejących bez modyfikacji starego kodu.

| Klasa oryginalna, na podstawie której tworzymy nową, nazywamy klasą **macierzystą**.

| Klasa, która dziedziczy funkcjonalność innej klasy nazywamy klasą **potomną**.

```
class Pracownik {  
    private:  
        enum {ILE = 20};  
        char imie[ILE];  
        char nazwisko[ILE];  
        char stanowisko[ILE];  
        double pensja;  
    public:  
        Pracownik(const char*, const char*, const char*, double p);  
        void wypiszDane() const;  
        void ustawPensja(double);  
        double getPensja() const;  
};  
  
class Dyrektor : public Pracownik {  
    private:  
        double dodatekFunkcyjny;  
    public:  
        Dyrektor(double, const char* i, const char*, const char*, double p);  
        Dyrektor(double dF, const Pracownik &);  
        double getDodatekFunkcyjny() {return dodatekFunkcyjny;}  
        void setDodatekFunkcyjny(double dF) {dodatekFunkcyjny = dF;}  
};
```

Dwukropka oznacza, że klasa *Dyrektor* powstała z klasy *Pracownik*, która tutaj stanowi publiczną klasę macierzystą (**dziedziczenie publiczne**). Obiekt klasy potomnej zawiera wszystkie pola składowe i metody klasy macierzystej. Gdy **dziedziczenie** jest **publiczne**, to wszystkie składowe publiczne klasy macierzystej stają się składowymi publicznymi klasy potomnej. Dostęp do odziedziczonych prywatnych składowych jest możliwy poprzez odziedziczone publiczne lub chronione metody klasy macierzystej.

Obiekt klasy potomnej

Obiekt klasy *Dyrektor* ma następujące cechy:

- Zawiera w sobie poza swoimi polami, pola klasy macierzystej, czyli: imie, nazwisko, stanowisko, pensja, dodatekFunkcyjny.
- Może korzystać z metod klasy macierzystej: *wypiszDane()*, *ustawPensja()*, *getPensja()*.
- Składowe prywatne imie, nazwisko, stanowisko, pensja są dziedziczone, lecz nie są bezpośrednio dostępne.
- Wartość składowej pensja jest dostępna jedynie poprzez odziedziczoną metodę publiczną *getPensja()*.

W klasie potomnej powinny być zdefiniowane własne konstruktory, które dostarczają danych zarówno dla nowych pól jak i odziedziczonych. Klasa potomna może być uzupełniona o dodatkowe pola składowe i metody.

Konstruktory klasy potomnej

Klasa potomna **nie może korzystać** z prywatnych składowych klasy macierzystej, musi więc odwoływać się do nich za pomocą publicznego interfejsu klasy macierzystej. W konsekwencji konstruktory klasy potomnej mogą wykorzystywać konstruktory klasy macierzystej.

Podczas tworzenia obiektu klasy potomnej tworzony jest najpierw obiekt klasy macierzystej. Aby wywołać odpowiedni konstruktor klasy macierzystej, wykorzystuje się tzw. listę inicjatorów konstruktora (listę inicjalizacyjną).

```
Dyrektor::Dyrektor(double dF, const char* i, const char* n, const char* s, double p) : Pracownik (i,n, s, p) {  
    dodatekFunkcyjny = dF;  
}  
  
//konstruktor bez listy inicjalizacyjnej  
Dyrektor::Dyrektor(double dF, const char* i, const char* n, const char* s, double p = 0) {  
    dodatekFunkcyjny = dF;  
}  
  
//powyższy konstruktor jest równoważny poniższemu:  
Dyrektor::Dyrektor(double dF, const char* i, const char* n, const char* s, double p) : Pracownik() {  
    dodatekFunkcyjny = dF;  
}  
  
//konstruktor powodujący wywołanie konstruktora kopującego:  
Dyrektor::Dyrektor(double dF, const Pracownik & p) : Pracownik(p) {  
    dodatekFunkcyjny = dF;  
}
```

Podczas likwidacji obiektu klasy potomnej w pierwszej kolejności wywoływany jest destruktor tej klasy, a następnie wywoływany jest destruktor klasy macierzystej.

Relacje między klasą macierzystą a potomną

- Obiekt klasy potomnej może korzystać z metod klasy macierzystej

```
Dyrektor anna(500, "Anna", "Lis", "Dyrektor", 3000);  
anna.wypiszDane();
```

- Wskaźnik do klasy macierzystej może wskazywać na obiekt klasy potomnej.

```
Pracownik* p1;  
p1 = &anna;  
p1->wypiszDane();
```

- Referencja do klasy macierzystej może odnosić się do obiektu klasy potomnej.

```
Pracownik& p2 = anna;  
p2.wypiszDane();
```

- Metody zdefiniowane w klasie potomnej mogą być wywołane jedynie przez referencję lub wskaźnika do obiektu klasy potomnej.

```
p1->setDodatekFunkcyjny(500); //błąd!!!, p1 wskazuje na obiekt klasy macierzystej
```

- Nie można przypisywać adresów i obiektów klasy macierzystej do wskaźników i referencji klasy potomnej.

```
Pracownik p3();  
Dyrektor& d1 = p3; //błąd!!!  
Dyrektor* d2 = &p3; //błąd!!!
```

Inicjalizacja obiektu klasy macierzystej za pomocą obiektu klasy potomnej

```
Dyrektor dyr(300, "Jan", "Kowalski", "Dyrektor", 3000);  
Pracownik p(dyr);
```

W powyższej sytuacji działa **konstruktor kopiący** klasy macierzystej:

```
Pracownik(const Pracownik&);
```

Przypisanie obiektu klasy potomnej do obiektu klasy macierzystej

```
Dyrektor dyr(300, "Jan", "Kowalski", "Dyrektor", 3000);
Pracownik p;
p = dyr;
```

W powyższej sytuacji działa w sposób niejawny **operator przypisania**:

```
Pracownik& operator=(const Pracownik&);
```

Rodzaje dziedziczenia

W C++ wyróżniamy trzy rodzaje dziedziczenia:

- **publiczne** - relacja typu "jest". Zgodnie z tą relacją obiekt klasy potomnej jest również obiektem klasy macierzystej. Wszystko co jest możliwe do wykonania z obiektem klasy macierzystej powinno się dać zrobić z obiektem klasy potomnej.
- **prywatne**
- **chronione**

Dziedziczenie - kontrola dostępu:

składowe klasy macierzystej	public	protected	private
składowe publiczne są	składowymi publicznymi klasy potomnej	składowymi chronionymi klasy potomnej	składowymi prywatnymi klasy potomnej
składowe chronione są	składowymi chronionymi klasy potomnej	składowymi chronionymi klasy potomnej	składowymi prywatnymi klasy potomnej
składowe prywatne są	Dostępne poprzez interfejs publiczny klasy macierzystej	Dostępne poprzez interfejs publiczny klasy macierzystej	Dostępne poprzez interfejs publiczny klasy macierzystej

Klasy abstrakcyjne

- Nie można tworzyć obiektów klasy abstrakcyjnej, ale można tworzyć wskaźniki

```
MyAbstractClass abclass; //błąd!
MyAbstractClass * p; //OK
```

- Klasy abstrakcyjne służą jako klasy macierzyste, a więc tworzymy je po to by z nich dziedziczyć.
- Mechanizm abstrakcyjnych klas macierzystych pozwala projektować hierarchię klas w sposób bardziej usystematyzowany i zdyscyplinowany.

Dziedziczenie wielokrotne

Dziedziczenie wielokrotne zazwyczaj prowadzi do niejednoznaczności wywołań funkcji. Najlepszym rozwiązaniem jest przeddefiniowanie wszystkich metod w klasie, która dziedziczy z wielu klas macierzystych. W przeddefiniowanych metodach przeważnie wskazujemy w sposób jawnym, które wersje metod chcemy wywołać.

Polimorfizm

Mechanizm polegający na tym, że jedna metoda może występować w wielu różnych postaciach w zależności od kontekstu jej wywołania nazywamy polimorfizmem.

W celu *wdrożenia* polimorficznego działania dziedziczenia publicznego stosujemy:

- redefinicje metod klasy macierzystej w klasie potomnej

- metody wirtualne

Aby odpowiednia wersja metody została wywołana należy przed deklaracją metody umieścić słowo **virtual**. W definicji słowo **virtual** pomijamy.

```
virtual double getPensja(); //deklaracja metody dla klasy Dyrektor  
...  
double getPensja() { //definicja metody  
    return Pracownik::getPensja() + dodatekFunkcyjny;  
}
```

W przypadku poprzedzenia deklaracji słowem **virtual**, odpowiednia wersja metody zostanie wywołana w oparciu o typ obiektu, do którego odwołuje się **referencja** lub **wskaźnik**.

```
Dyrektor anna;  
Pracownik janek;  
Pracownik& p1 = anna;  
Pracownik& p2 = janek;  
p1.getPensja(); //wywołana metoda Dyrektor::getPensja();  
p2.getPensja(); //wywołana metoda Pracownik::getPensja();
```

W przypadku, gdy metoda przeddefiniowana nie jest poprzedzona w części deklaracyjnej słowem **virtual**, wówczas sposób działania metody opiera się na typie referencji, a nie na typie obiektu.

```
Dyrektor anna;  
Pracownik janek;  
Pracownik& p1 = anna;  
Pracownik& p2 = janek;  
p1.getPensja(); //wywołana metoda Pracownik::getPensja();  
p2.getPensja(); //wywołana metoda Pracownik::getPensja();
```

Zazwyczaj dobrą praktyką jest poprzedzanie w klasie macierzystej słowem **virtual** deklaracje tych metod, które są przeddefiniowane w klasie potomnej. Zabieg ten pozwala wybrać odpowiednie wersje metod, na podstawie obiektu, na rzecz którego są one wywoływane, a nie na postawie referencji lub wskaźnika.

Poprzedzenie destruktatorów słowem **virtual** powoduje, że podczas destrukcji obiektu zostanie wywołany odpowiedni kod destruktora.

- Jeżeli w deklaracji klasy daną metodę poprzedzimy słowem kluczowym **virtual**, wówczas metoda będzie metodą wirtualną w klasie macierzystej, potomnej i innych klasach dziedziczących po klasie potomnej.
- Jeżeli metoda wirtualna wywoływana jest na rzecz **referencji lub wskaźnika**, to program użyje tej wersji metody, która odpowiada typowi obiektu na który dana referencja czy wskaźnik wskazuje.
- Na metody wirtualne wybieramy te, które w klasach potomnych będą przedefiniowane.
- Jeżeli w którejś klasie potomnej nie zostanie przedefiniowana metoda wirtualna, wówczas obiekt tej klasy będzie korzystał z funkcji wirtualnej najbliższego „przodka”.
- Konstruktory **nie mogą** być metodami wirtualnymi, gdyż klasa potomna nie dziedziczy konstruktorów klasy macierzystej.
- Jeżeli dana klasa będzie stanowić klasę macierzystą, wówczas jej destruktory **powinny** być wirtualne.
- Funkcje zaprzyjaźnione (**friend**) **nie mogą być wirtualne**, gdyż nie są metodami klasowymi.

Wiązanie statyczne i dynamiczne

Wiązanie nazwy funkcji polega na określeniu odpowiedniego bloku wykonywalnego (w kodzie skompilowanym), który ma zostać użyty.

Wiązanie statyczne to wiązanie, które jest realizowane podczas kompilacji kodu źródłowego.

Wiązanie dynamiczne, to odpowiedni mechanizm, który pozwala wybrać odpowiednią metodę wirtualną podczas działania programu.

Uwaga Wiązanie dynamiczne zachodzi wówczas, gdy odpowiednie metody wywoływanie są przez **wskaźniki lub referencje**.

20. Polimorfizm statyczny – szablony.

Polimorfizm statyczny jest często implementowany za pomocą **szablonów**. Jest on nieograniczony, bo interfejsy typów uczestniczących w polimorfizmie nie są z góry określone.

Szablony służą do tworzenia ogólnych deklaracji klas (lub funkcji). W ten sposób realizowana jest koncepcja tzw. typów sparametryzowanych. Typ jest argumentem przekazywanym do ogólnego wzorca klasy lub funkcji.

Szablony pozwalają na *wielokrotne wykorzystanie istniejącego kodu* źródłowego struktury danych dla wielu wersji tej struktury z tym samym interfejsem, ale różnymi typami dla wewnętrznych komponentów (**programowanie generyczne/uogólnione, struktury parametryzowane**)

```
template <typename T>
class Punkt {

private:
    T x, y;
public:
    Punkt();
    Punkt(T,T);
```

```
T getX();
T getY();

void setXY(T,T);
void wypisz();

};

template <typename T>
Punkt<T>::Punkt() {
    x = y = 0;
}

template <typename T>
T Punkt<T>::getX() {
    return x;
}

...
```

Chcąc wygenerować klasę na podstawie zdefiniowanego szablonu należy jawnie określić typ parametru (tzn. dokonać jawnej konkretyzacji typu).

```
#include <iostream>
#include "Punkt.h"
using namespace std;

int main() {
    Punkt<int> p(3,2);
    cout << p.getX();

    Punkt<double> z(3.2,2.1);
    z.setXY(5.01,3);

    return 0;
}
```

■ Konkretyzując klasę możemy użyć zarówno typu wbudowanego jak i obiektu jakiejś klasy.

■ W języku C++ możemy używać szablony, które posiadają więcej niż jeden argument typu. Korzystając z tej możliwości możemy utworzyć klasę do przechowywania dwóch elementów różnych typów.

```
template <typename T1, typename T2>
class Pair {
```

```

private: T1 x, T2 y;
public:
    T1 & first(const T1 & f) {x = f; return x;};
    T2 & second(const T2 & s) {y = s; return y;};
    T1 first() const {return x;};
    T2 second() const {return y;};

    Pair(const T1 & f, const T2 & s) : x(f), y(s) {}
    Pair(){}
};


```

21. Listy i drzewa oraz ich zastosowania. Stosy i kolejki.

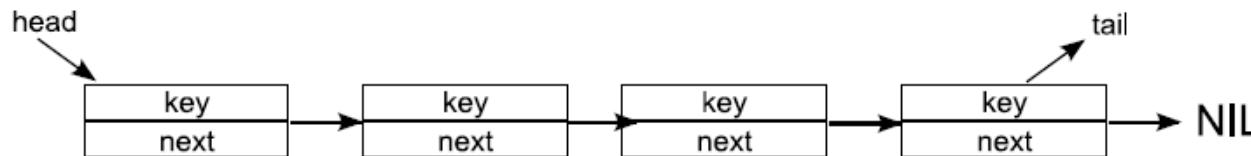
Listy

Lista to ciąg elementów, gdzie każdy zawiera atrybuty: **key**, **next** oraz **previous**. Wartość każdego z tych elementów odczytujemy przez **key[x]**. Listy reprezentujemy poprzez obiekt, zawierający atrybuty **head** oraz **tail**, wskazujące odpowiednio na początek i koniec listy.

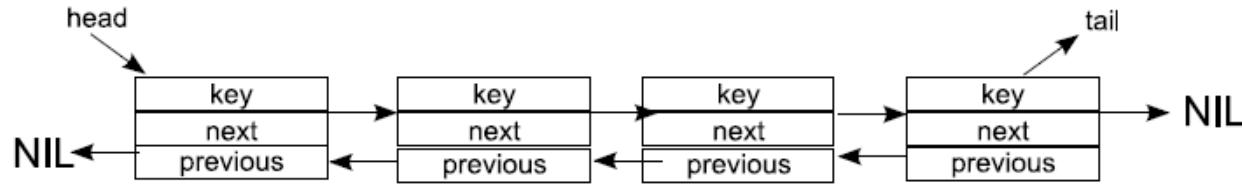
Listy dzielimy na 3 typy:

1. listy jednokierunkowe
2. Listy dwukierunkowe
3. Listy cykliczne

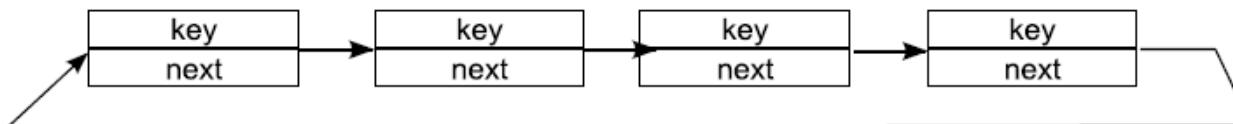
W liście **jednokierunkowej** każdy z elementów wskazuje na element następny, czyli dla każdego x , $\text{next}[x]$ wskazuje na kolejny. Ostatni element listy wskazuje na pusty element **NIL**.



W liście **dwukierunkowej** każdy element zawiera trzy atrybuty. Oprócz atrybutu **next** zawiera także **previous** wskazujący na poprzedni element listy. Na pusty element **NIL** wskazuje zarówno pierwszy jak i ostatni element listy.



Listą **cykliczną** nazywamy listę, w której ostatni element **zamiast wskazywać** na pusty element **NIL**, wskazuje na pierwszy element listy. **Nie występują** tutaj atrybuty **head i tail**.



Stos

Stos to liniowa struktura danych, gdzie elementy przetwarzane są w kolejności od tego, który pojawił się najpóźniej (jest na górze stosu) do tego, który pojawił się na samym początku (na dole stosu).

Poszczególne elementy można przeglądać, lecz by pobrać element znajdujący się poniżej, trzeba pobrać ze stosu wszystkie elementy znajdujące się nad nim.

W algorytmach stos reprezentowany jest przez strukturę **LIFO (Last In First Out)**.

Kolejka

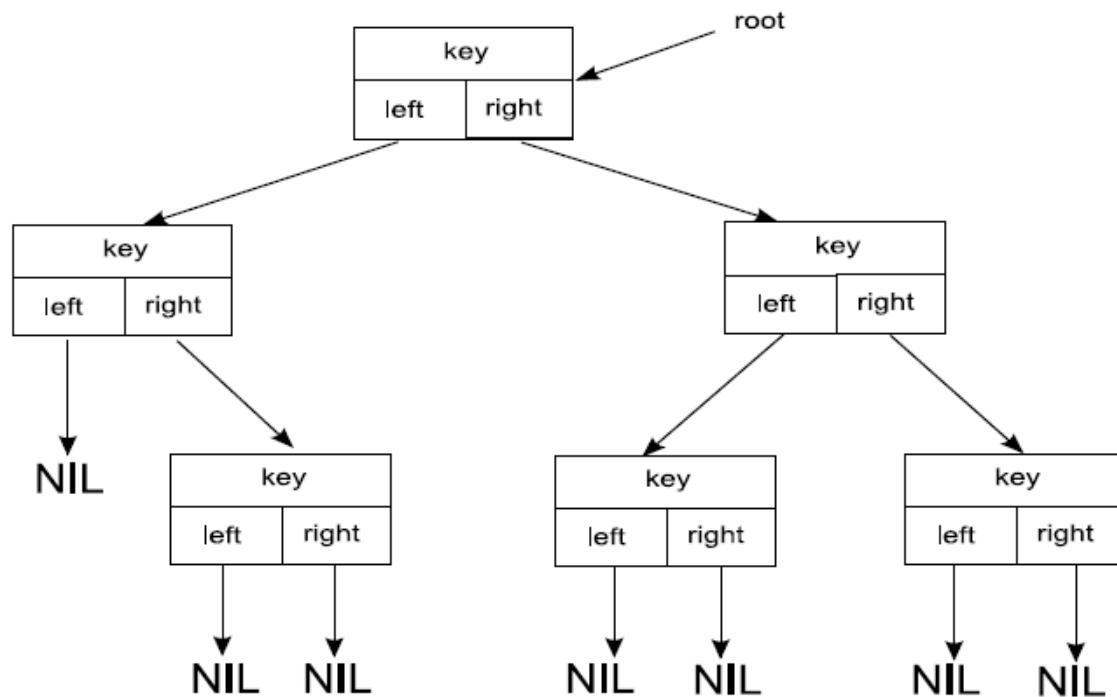
Kolejka jest strukturą działającą przeciwnie do stosu. Dane są przetwarzane w kolejności ich pojawienia się, tzn.: zaczynając od tego, który pojawił się na początku, kończąc na tym, który znalazł się na samym końcu.

W algorytmach kolejka reprezentowana jest poprzez strukturę **FIFO (First In First Out)**.

Drzewa

Drzewo składa się z elementów, które posiadają **trzy** atrybuty: **key, left** oraz **right**.

Drzewo reprezentowane jest przez obiekt z atrybutem **root (korzeń)**. Kolejne elementy są jego potomkami.



W informatyce drzewo wykorzystywane jest do budowy **drzew decyzyjnych**, które są podstawą działania takich algorytmów jak **min-max** (wyznaczanie optymalnych ruchów).

22. Grafy i metody ich przeszukiwania. Zastosowania.

Grafy

Graf To struktura matematyczna służąca do przedstawiania i badania relacji między obiektami. W uproszczeniu **Graf to zbiór wierzchołków, które mogą być połączone krawędziami w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków.**

Grafem nieskierowanym nazywamy parę $\mathbf{G}=(\mathbf{V}, \mathbf{E})$, gdzie \mathbf{V} jest pewnym zbiorem skończonym zwanym **zbiorem wierzchołków grafu \mathbf{G}** .

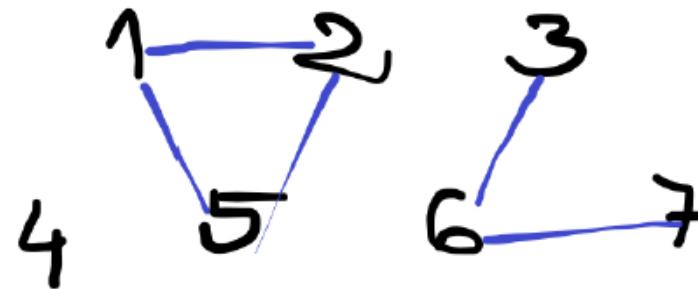
Natomiast \mathbf{E} jest zbiorem nieuporządkowanych par $\{\mathbf{u}, \mathbf{v}\}$ gdzie $\mathbf{u}, \mathbf{v} \in \mathbf{V}$ oraz $\mathbf{u} \neq \mathbf{v}$.

Zbiór \mathbf{E} nazywamy zbiorem krawędzi grafu \mathbf{G} .

PRZYKŁAD Zilustrować graf $G = (V, E)$, gdzie

$V = \{1, 2, 3, 4, 5, 6, 7\}$ oraz

$E = \{\{1, 2\}, \{1, 5\}, \{2, 5\}, \{3, 6\}, \{6, 7\}\}$.



Jeśli $\{u, v\}$ jest krawędzią grafu nieskierowanego G , to mówimy, że $\{u, v\}$ jest **incydentna** z wierzchołkami u i v .

Stopniem wierzchołka w grafie nieskierowanym nazywamy **liczbę incydentnych z nim krawędzi**. Pętlę liczymy za 2.

Grafem skierowanym nazywamy parę $G = (V, E)$, gdzie V jest pewnym zbiorem skończonym zwanym **zbiorem wierzchołków grafu G**.

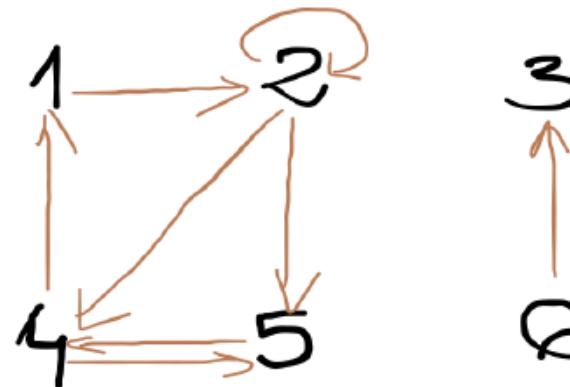
Natomiast E - zbiór krawędzi grafu G .

G - zbiór uporządkowanych par $\{(u, v)\}$ oznaczanych (u, v) , gdzie $u, v \in V$

PRZYKŁAD Zilustrować graficznie graf $G = (V, E)$, gdzie

$V = \{1, 2, 3, 4, 5, 6\}$ oraz

$E = \{(1, 2), (2, 2), (2, 4), (2, 5), (4, 1), (4, 5), (5, 4), (6, 3)\}$.



Stopniem wierzchołka w grafie skierowanym nazywamy **sumę liczby krawędzi wchodzących do wierzchołka i wychodzących z tego wierzchołka**

Rząd grafu - liczba wierzchołków w grafie.

Rozmiar grafu - liczba krawędzi w grafie.

Droga (ścieżka) - drogą w grafie będziemy nazywać ciąg krawędzi taki, że koniec jednej stanowi początek następnej. Drogę nazywamy **prostą**, gdy **wszystkie jej wierzchołki są różne**.

Osiągalność - mówimy, że **v** jest osiągalny z **u**, gdy istnieje droga z **u** do **v**.

Cykl - cyklem nazywamy zamkniętą drogę $x_1x_2x_3\dots x_nx_1$ w grafie skierowanym, gdzie to wierzchołki $x_1x_2\dots x_n$ to wierzchołki drogi, która jest długości co najmniej 1. **Gdy wszystkie wierzchołki są różne to cykl nazywamy prostym. Cykl o długości 1 nazywamy pętlą.**

Mówimy, że ścieżka $\langle v_0, v_1, \dots, v_k \rangle$ tworzy **cykl** w grafie **nieskierowanym**, gdy $v_0 = v_k$, $v_1 \dots v_k$ są różne oraz $k \geq 2$

Graf niezawierający cykli nazywamy grafem **acyklicznym**.

Acykliczny graf nieskierowany nazywamy **lasem**.

Graf prosty to graf bez krawędzi wielokrotnych i bez pętli.

Graf regularny to graf, w którym wszystkie wierzchołki są tego samego stopnia.

Graf pusty to graf, w którym w ogóle nie ma krawędzi (są same wierzchołki izolowane).

Graf pełny to graf prosty, w którym każdy wierzchołek jest połączony krawędzią z każdym.

Graf jest spójny, gdy każda para różnych wierzchołków jest połączona drogą w tym grafie.

Spójny podgraf grafu **G**, który nie jest zawarty w żadnym większym spójnym podgrafie tego grafu, nazywamy **składową grafu G**.

Spójny, acykliczny graf nieskierowany nazywamy **drzewem (wolnym)**.

Cykl Eulera - droga zamknięta przechodząca przez każdą krawędź grafu dokładnie raz.

Droga Eulera - droga przechodząca przez każdą krawędź grafu dokładnie raz.

Graf, który posiada cykl Eulera **Musi mieć wszystkie wierzchołki stopnia parzystego**.

Graf, który posiada drogę Eulera ma **albo dokładnie dwa wierzchołki stopnia nieparzystego, albo nie ma w ogóle takich wierzchołków**.

Graf spójny, mający dokładnie 2 wierzchołki stopnia nieparzystego **posiada drogę Eulera**.

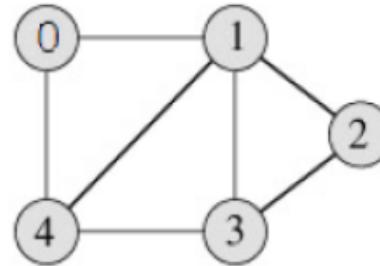
Formy reprezentacji grafów

- **Macierz sąsiedztwa**

Macierzą sąsiedztwa grafu (skierowanego lub nie) nazywamy macierz **M** o wymiarze **VxV**, w której wartości reprezentują wagę połączeń pomiędzy wierzchołkami, 1 gdy połączone, 0 gdy nie ma.

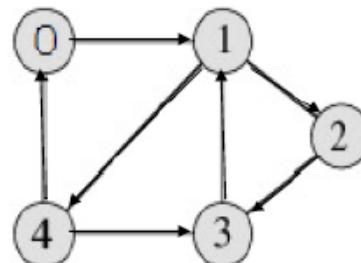
a) Graf nieskierowany

	0	1	2	3	4
0	0	1	0	0	1
1	1	0	1	1	1
2	0	1	0	1	0
3	0	1	1	0	1
4	1	1	0	1	0



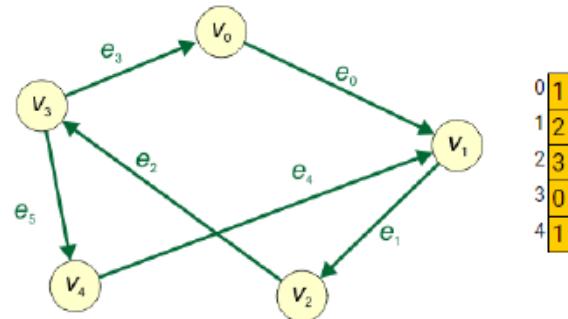
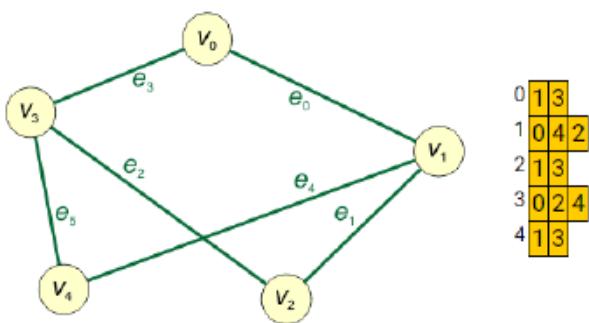
b) graf skierowany

	0	1	2	3	4
0		1			
1			1		1
2				1	
3		1			
4	1			1	



- **Lista sąsiedztwa**

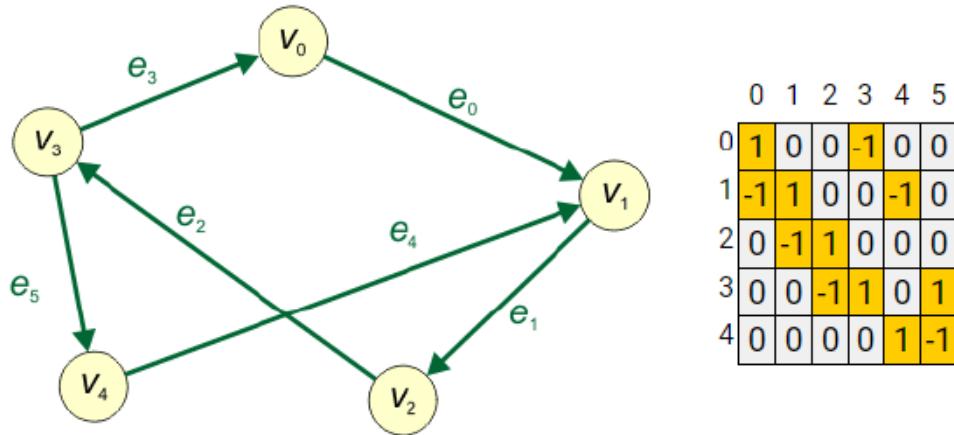
Dane zapisywane są w postaci listy obiektów zawierających wierzchołek grafu, wraz z listą wierzchołków sąsiednich. W przypadku grafu nieskierowanego listy są dłuższe, ponieważ muszą odzwierciedlać krawędzie w obu kierunkach.



- **Macierz incydencji**

Macierz incydencji jest macierzą \mathbf{A} o wymiarze $n \times m$, gdzie n oznacza liczbę wierzchołków grafu, natomiast m liczbę jego krawędzi. Każdy wiersz tej macierzy odwzorowuje jeden wierzchołek grafu. Każda kolumna odwzorowuje jedną krawędź. Zawartość komórki $\mathbf{A}[i, j]$ określa powiązanie (incydencję) wierzchołka v_i z krawędzią e_j w sposób następujący:

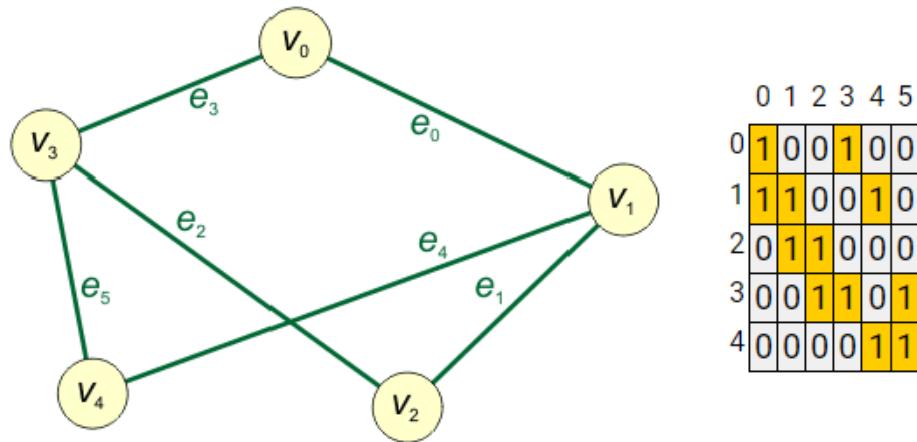
$$A[i, j] = \begin{cases} 0, & \text{jeśli } v_i \text{ nie należy do } e_j \\ 1, & \text{jeśli } v_i \text{ jest początkiem } e_j \\ -1, & \text{jeśli } v_i \text{ jest końcem } e_j \end{cases}$$



	0	1	2	3	4	5
0	1	0	0	-1	0	0
1	-1	1	0	0	-1	0
2	0	-1	1	0	0	0
3	0	0	-1	1	0	1
4	0	0	0	0	1	-1

Jeśli graf jest nieskierowany, to definicję macierzy należy uprościć:

$$A[i, j] = \begin{cases} 0, & \text{jeśli } v_i \text{ nie należy do } e_j \\ 1, & \text{jeśli } v_i \text{ należy do } e_j \end{cases}$$



Zastosowania

Mapy - Aby znaleźć najkrótszą drogę by dostać się z jednego miejsca do drugiego można wykorzystać graf, którego wierzchołki będą odpowiadały miejscowością a krawędzie drogom.

Dokumenty hipertekstowe - Przeszukując internet napotykamy dokumenty, które zawierają odnośniki do innych dokumentów - internet jest grafem, którego wierzchołkami są dokumenty a krawędziami odsyłacze.

Sieci - Sieć komputerowa zbudowana jest z komputerów, które przesyłają między sobą informacje. Komputery w danej sieci reprezentowane są przez wierzchołki grafu, a połączenia między nimi krawędziami.

Struktura programu - Kompilator buduje graf reprezentujący strukturę wywołań podprogramów w komplikowanym programie. Wierzchołkami grafu są różne funkcje, natomiast krawędzie są kojarzone z wywołaniem funkcji./

23. Metody projektowania algorytmów (dziel i rządź, programowanie dynamiczne i algorytmy zachłanne).

- **Dziel i rządź**

Polega na rekurencyjnym dzieleniu problemu na mniejsze podproblemy. Dzielenie trwa dopóki nie uzyskamy problemów, które da się w prosty sposób rozwiązać.

Algorytmy wykorzystujące metodę "dziel i rządź":

- Sortowanie przez wybieranie
- Sortowanie przez wstawianie
- Sortowanie przez scalanie
- Quicksort

- **Programowanie dynamiczne**

Jest stosowane głównie do rozwiązywania problemów optymalizacyjnych. Jest alternatywą dla niektórych zagadnień rozwiązywanych metodami zachłannymi. Wyniki poszczególnych obliczeń są zapamiętywane w pomocniczej tablicy, która jest wykorzystywana w kolejnych krokach. Eliminuje to konieczność wielokrotnego powtarzania tych samych obliczeń.

Algorytmy wykorzystujące programowanie dynamiczne:

- Algorytm Bellmana-Forda
- Algorytm Helda-Karpa
- Algorytm wykorzystujący problem wydawania reszty

- **Algorytm zachłanny**

W każdym kroku dokonuje wyboru będącego na daną chwilę tym najlepszym. Podejmuje decyzje optymalne tylko lokalnie. Kontynuuje działania wynikające z poprzednich decyzji.

Algorytmy wykorzystujące metodę zachłanną:

- Algorytm Dijkstry (poszukiwanie najkrótszych ścieżek)
- Algorytm Kruskala (poszukujący minimalnego drzewa rozpinającego)

24. Elementarne i nieelementarne metody sortowania.

Elementarne metody sortowania:

- **Sortowanie przez selekcję (selection sort)** - Jego czas działania jest określany z góry $O(n^2)$. Sortowanie to jest najlepsze, spośród innych elementarnych do sortowania elementów o małych kluczach i dużych polach, ponieważ wykonuje najmniej wstawień. W pierwszym przebiegu algorytm znajduje najmniejszy element w tablicy i zamienia go z pierwszym. W drugim algorytm znajduje najmniejszy element w podtablicy i zamienia go z drugim. Tak aż do zamiany r-tego elementu z r-1 elementem.
- **Sortowanie bąbelkowe** - Czas działania jest z góry określony przez $O(n^2)$ - algorytm wykonuje w najgorszym i średnim przypadku podobną ilość porównań i zamian. Zadada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełniania kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Nieelementarne metody sortowania:

- **Quicksort (sortowanie szybkie)** - Średnia złożoność obliczeniowa $O(n \log n)$. Bazuje na strategii dziel i rządź. Problem dzielimy rekurencyjnie na podproblemy a następnie łączymy w jedno rozwiązanie. Z tablicy wybiera się element rozdzielający, po czym tablica jest dzielona na dwa fragmenty: do początkowego przenoszone są wszystkie elementy nie większe od rozdzielającego, do końcowego wszystkie większe. Potem sortuje się osobno początkową i końcową część tablicy. Rekurencję kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa tablica nie wymaga sortowania.
- **Sortowanie pozycyjne** - Algorytm porządkujący stabilnie ciągi wartości (liczb, słów) względem konkretnych cyfr, znaków itp, kolejno od najmniej znaczących do najbardziej znaczących pozycji. Złożoność obliczeniowa jest równa $O(d(n+k))$, gdzie k to liczba różnych cyfr, a d liczba cyfr w kluczach. Wymaga $O(n+k)$ dodatkowej pamięci.

25. Elementarne metody wyszukiwania. Haszowanie.

Wyszukiwanie liniowe/sekwencyjne

- Polega na przeglądaniu kolejnych elementów zbioru **Z**
- W przypadku odnalezienia elementu, który posiada odpowiednie własności, zwraca jego pozycje w zbiorze i kończy pracę.
- W przypadku pesymistycznym, gdy poszukiwanego elementu nie ma w zbiorze lub też znajduje się on na samym końcu zbioru, algorytm musi wykonać przynajmniej **n** obiegów pętli sprawdzającej poszczególne elementy
- klasa złożoności obliczeniowej jest równa **O(n)**
- W przypadku poszukiwania wszystkich wystąpień poszukiwanej własności elementu, algorytm po zwrocie pierwszej odnalezionej pozycji kontynuuje pracę od następnego indeksu.

Wyszukiwanie binarne

- Opiera się na metodzie **dziel i rządź** oraz działa na uporządkowanych tablicach
- Polega na sprawdzeniu środkowego elementu zbioru oraz przyjęciu nowego podzbioru, gdy środkowy element nie spełnia kryteriów wyszukiwania
- Wyszukiwanie kontynuowane jest w podzbiorze spełniającym warunek porównania **mniejsze-większe**
- W przypadku odnalezienia elementu, posiadającego odpowiednie własności, zwraca on jego pozycję w tablicy i kończy pracę
- Złożoność obliczeniowa równa **O(log₂n)**
- W przypadku tablicy o milionie elementów, algorytm musi sprawdzić maksymalnie 20 pozycji

Wyszukiwanie max lub min

- Opiera się na metodzie wyszukiwania liniowego
- Przyjmuje pierwszy element zbioru za tymczasowy maksymalny/minimalny a następnie porównuje go z kolejnym elementem i na podstawie wyniku porównania może przyjąć nowy tymczasowy element maksymalny/minimalny
- Po przejściu przez wszystkie elementy zbioru, wartość elementu tymczasowego zostaje przyjęta jako maksimum lub minimum zbioru
- Złożoność obliczeniowa równa **O(n)**

Naiwne wyszukiwanie wzorca w tekście

- Algorytm ustawia okno o długości wzorca **p** na pierwszej pozycji w łańcuchu **s** a następnie sprawdza czy zawartość wzorca **p** i porównywanego fragmentu łańcucha **s** są sobie równe
- W przypadku pozytywnym pozycja okna jest zwracana jako wynik
- Po porównaniu okno przesuwa się o jedną pozycję w prawo i cała procedura powtarza się, dopóki okno nie wyjdzie poza koniec łańcucha **s**
- Posiada złożoność obliczeniową równą **O(n x m)** pesymistycznie, oraz **o(n)** w najlepszym przypadku.
n - długość łańcucha
m - długość wzorca

Haszowanie

- Technika rozwiązywania ogólnego problemu słownika, czyli takiego zorganizowania struktur danych i algorytmów, aby można było w miarę efektywnie przechowywać i wyszukiwać elementy należące do pewnego dużego zbioru danych (uniwersum),
- Przykładem tablicy z haszowaniem jest książka telefoniczna, w której kluczem są imię i nazwisko a wyszukiwaną informacją jest numer telefonu.
- Czas uzyskania informacji jest **niezależny** od rozmiaru tablicy lub położenia elementu
- W najprostszym przypadku wartość **funkcji mieszającej** dla danego **klucza** wyznacza **indeks** szukanej informacji w tablicy (złożoność obliczeniowa wynosi **O(1)**)
- Może wystąpić problem **kolizji**, czyli przypisania przez funkcję mieszającą tej samej wartości dwóm różnym kluczom

Unikanie kolizji

- Zastąpienie istniejącego elementu lub rezygnacja ze wstawienia nowego elementu
- Metoda Łąćuchowa** - przechowywanie elementów o tej samej wartości wewnętrz listy lib drzewa związkanych z danym indeksem (pesymistycznie **O(n)**)
- Adresowanie Otwarte** - nowy element wstawia się w innym miejscu niż wynikłoby to z wartości funkcji mieszającej a nowy indeks określany jest przez dodanie do wartości tzw. funkcji przyrostu **p(i)**, gdzie **i** oznacza numer próby (liczba kolizji)
- Współczynnik wypełnienia** - iloraz liczby elementów zapisanych w tablicy mieszającej (**m**) od fizycznego rozmiaru tablicy (**n**), czyli $\alpha = m/n$. Dzięki temu można przewidzieć prawdopodobieństwo wystąpienia kolizji i odpowiednio skorygować fizyczny rozmiar tablicy
- Haszowanie kukułcze** - stosuje dwie tablice i dwie odpowiadające im funkcje haszujące. Do momentu kolizji elementy wstawiane są do pierwszej tablicy za pomocą pierwszej funkcji haszującej. W wypadku kolizji element wstawiany jest do drugiej tablicy przez drugą funkcję. Jeśli ponownie nastąpi kolizja, to zastępujemy istniejący już obiekt a dla niego zostaje uruchomiona ponowna procedura wstawiania, jednak tym razem na siłę będzie wstawiony do pierwszej tablicy. W przypadku zapętlenia się algorytmu losowane są nowe funkcje haszujące i wszystkie elementy tablicy zostają ponownie przemieszane. Odczyt elementu z tablicy odbywa się zawsze w czasie stałym.

Haszowanie - zastosowania

- kompilatory, interpretery (w dynamicznych językach obiektowych)

- bazy danych - indeksowanie, łączenie, grupowanie
- analiza i agregacja danych
- trasowanie
- systemy pamięci podręcznej
- monitorowanie
- implementacja zbiorów
- mapowanie
- kompresja danych
- wyszukiwanie wzorca w tekście

Haszowanie - wady

- Współczesne procesory wykorzystują pamięć podręczną, która przyspiesza odwołania do komórek pamięci operacyjnej, gdy te są zgrupowane blisko siebie. Zastosowanie tablicy mieszającej (z haszowaniem) dla małej liczby elementów może być wolniejsze niż zastosowanie zwykłej tablicy przeszukującej sekwencyjnie
- Występuje ryzyko dużej złożoności obliczeniowej w pesymistycznym przypadku wyszukiwania wynoszącej $O(n)$
- Obliczenie wartości dobrej funkcji haszującej (eliminującej kolizje) może być kosztowne względem czasu lub zasobów pamięci

26. Złożoność obliczeniowa algorytmu. Przykłady.

Mierzymy liczbę operacji wykonanych na modelu. Następnie próbujemy znaleźć funkcję, która będzie opisywała liczbę operacji w zależności od wejścia algorytmu. Funkcje te możemy porównać ze sobą.

```
int sum(int[] numbers) {
    int sum = 0
    for(int number : numbers) {
        sum += number;
    }
    return sum;
}
```

Ille mamy tu operacji? Przypisanie, pętla for. Jej ciało zawiera jedną operację. Sama pętla wykona się dokładnie tyle razy ile jest elementów tablicy numbers. Liczbę tych elementów określmy jako n . Na końcu mamy instrukcję return sum.

Dodając do siebie otrzymujemy wzór:

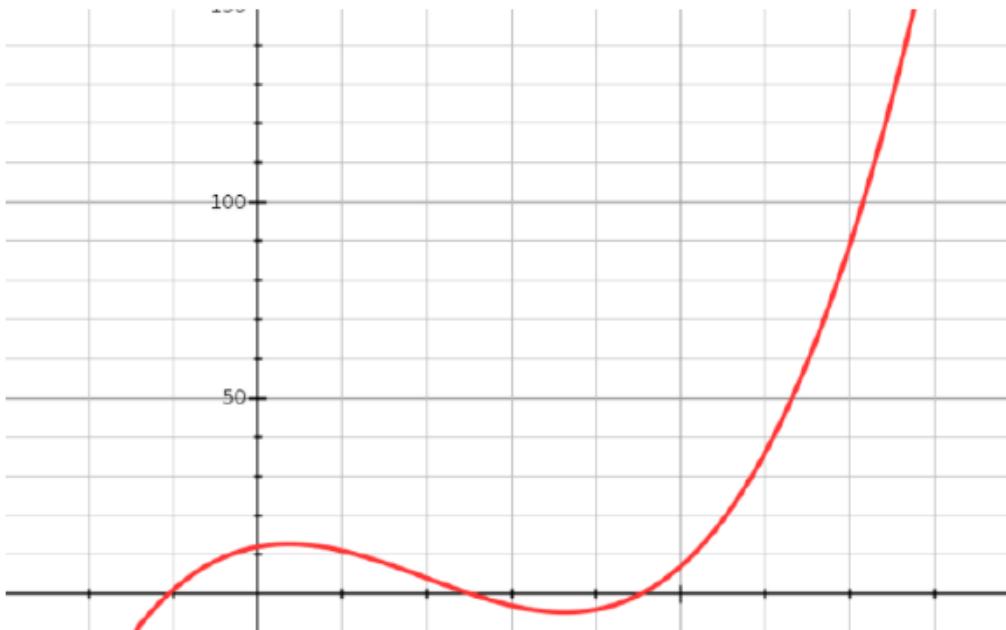
$$f(n) = 1 + n + 1 = n + 2$$

Zatem złożoność naszego algorytmu opisana jest przez funkcję $f(n) = n + 2$. Tak więc wyznacza się ją poprostu licząc operacje.

Jak oszacować rzad złożoności funkcji?

Wyobraźmy sobie pewien algorytm. Funkcja, która go opisuje to np.: $f(n) = n^3 - 6n^2 + 4n + 12$

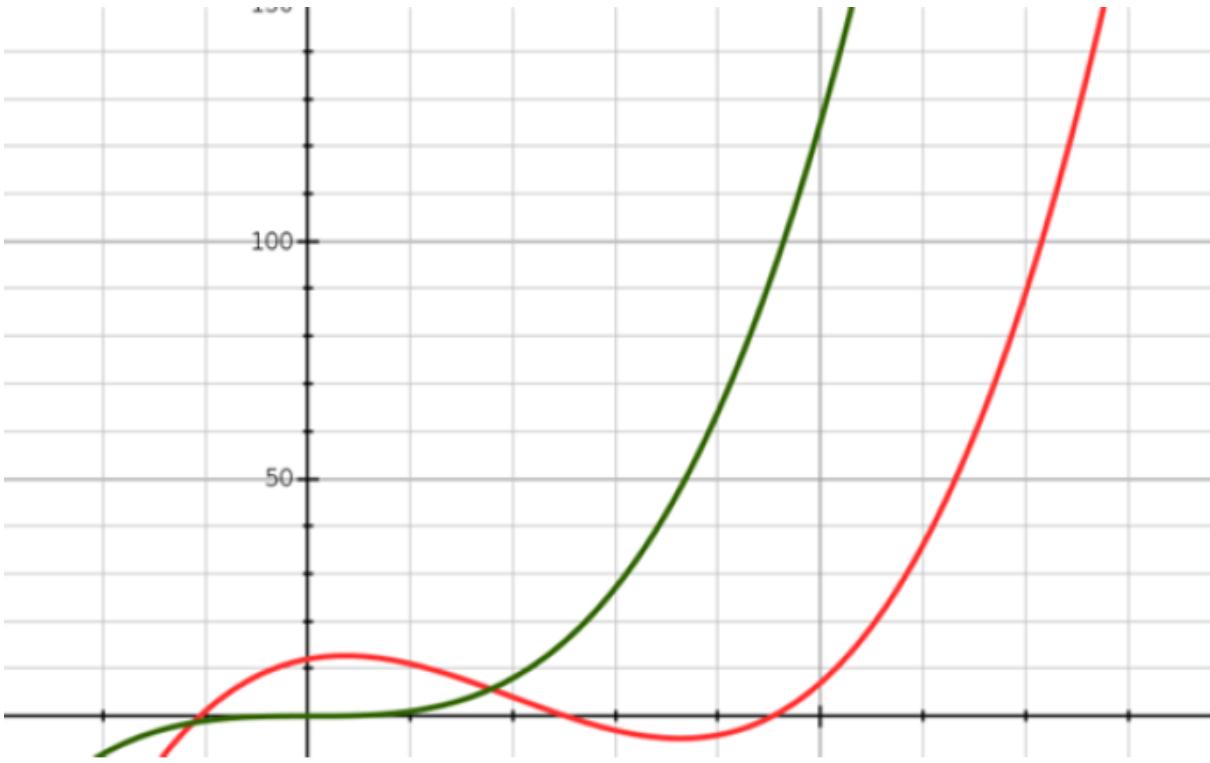
Argument n to rozmiar danych wejściowych do algorytmu. Wykres tej funkcji wygląda następująco:



Notacja Dużego O zakłada, że istnieje funkcja $g(n)$, dla której spełniona jest poniższa wartość:

$$\forall n \geq n_0 : f(n) \leq c \cdot g(n)$$

Oznacza to, że wynik funkcji $g(n)$ pomnożony przez jakąś stałą c , będzie większy bądź równy wynikowi funkcji $f(n)$. Własność ta jest spełniona dla wszystkich n , które będą większe od n_0 . Jeszcze łatwiej wygląda to na wykresie:



Pokazuje on dwie funkcje. Pierwszą z poprzedniego wykresu. Druga to wykres funkcji $g(n) = n^3$. Od pewnego punktu zielona linia jest zawsze ponad czerwoną linią. To nic innego jak oszacowanie z góry. To właśnie robi notacja O. Zatem w naszym przypadku nasza funkcja $f(n)$ ma złożoność $O(n^3)$

27. Pojęcie bazy danych - funkcje i możliwości.

Baza danych jest zbiorem danych oraz narzędzi **Zystemu Zarządzania Bazą Danych (SZBD)** przeznaczonego do zarządzania nią oraz gromadzenia, przekształcania i wyszukiwania danych.

To zbiór usystematyzowanych informacji (danych), który dotyczy rzeczywistości a konkretne określonego jej fragmentu (wycinka), który reprezentuje. Fragment ten określamy mianem obszaru analizy.

Cechy bazy danych

- **Trwałość danych** - oznacza możliwość przechowywania danych w pamięci masowej (trwałej) komputera
- **Niezależność danych** - pozwala osiągnąć większą elastyczność, ponieważ programy wymieniające informacje z bazą danych są niezależne od przechowywania danych na dysku i szczegółów reprezentacji danych na dysku
- **Ochrona danych** - baza danych oferuje mechanizmy kontroli dostępu do danych w sposób umożliwiający użytkowanie danych wyłącznie przez uprawnionych do tego użytkowników
- **Integralność danych** - zgodność z rzeczywistością. Dane w bazie danych są odwzorowaniem rzeczywistości
- **Współdzielenie danych** - poszczególne fragmenty danych mogą być używane przez kilku użytkowników jednocześnie (dostęp współbieżny)
- **Abstrakcja danych** - dane opisują tylko istotne aspekty obiektów świata rzeczywistego

- **Integracja danych** = gwarantująca, że dane i związki między nimi nie powtarzają się jeśli nie jest to konieczne ale wszelkie zmiany w obrębie bazy nie powodują wieloznaczności

System zarządzania bazą danych (SZBD) obsługuje użytkowników bazy danych, umożliwiając im eksploatację oraz tworzenie baz danych. By stworzyć i zaprojektować bazę danych, należy ją zidentyfikować, a do tego konieczne jest określenie typów przechowywanych w niej danych. Istotną rolę odgrywa również wyznaczenie użytkowników oraz ich praw dostępu.

Właściwości bazy danych (funkcje SZBD)

- Tworzenie struktur baz danych
- Wykonywanie operacji CRUD (Create, Read, Update, Delete)
- Obsługa zapytań (selekcyjowanie danych)
- Generowanie raportów i zestawień
- Administracja bazą danych

Podział baz danych

- **Model relacyjno-objektowy** jest mieszanym modelem bazodanowym, pozwala on w relacyjnych tabelach tworzyć kolumny, w których przechowywane są dane typu obiektowego, pozwala na definiowanie zmiennych oraz metod, które będą wykonywane na danych wprowadzanych do obiektu. Podstawa wielodostępu: identyfikatory użytkowników i ich hasła, procedura logowania systemu, praw dostępu i uprawnień, grupy użytkowników
- **Objektowy model danych** opiera się na koncepcji obiektów (podobnie jak w projektowaniu obiektowym – obiekt jest odwzorowaniem rzeczywistości lub abstrakcji). Odwołania do określonego obiektu w tym modelu bazy danych są wykonywane za pomocą interfejsu, dzięki któremu są zachowane integralność i bezpieczeństwo danych.
- **Relacyjny model danych** w najprostszym ujęciu w modelu relacyjnym dane grupowane są w relacje, które reprezentowane są przez tablice. Relacje są pewnym zbiorem rekordów o identycznej strukturze wewnętrznie powiązanych za pomocą związków zachodzących pomiędzy danymi. Relacje zgrupowane są w tzw. schematy bazy danych. Relację może być tabela zawierająca dane teleadresowe pracowników, zaś schemat może zawierać wszystkie dane dotyczące firmy. Takie podejście w porównaniu do innych modeli danych ułatwia wprowadzanie zmian, zmniejsza możliwość pomyłek, ale dzieje się to kosztem wydajności. W 1985 r. Edgar Frank Codd (twórca) przedstawił 12 zasad opisujących model relacyjny baz danych.

Postulaty Codda

- **Postulat informacyjny** – dane są reprezentowane jedynie poprzez wartości atrybutów wierszach tabel.
- **Postulat dostępu** – każda wartość w bazie danych jest dostępna poprzez podanie nazwy tabeli, atrybutu i wartości klucza podstawowego.
- **Postulat dotyczący wartości NULL** – dostępna jest specjalna wartość NULL dla reprezentacji zarówno wartości nieokreślonej, jak i nieadekwatnej.
- **Postulat dotyczący katalogu** – wymaga się, aby system obsługiwał wbudowany katalog relacyjny z bieżącym dostępem dla uprawnionych użytkowników.
- **Postulat języka danych** – system musi dostarczać pełny język przetwarzania danych, który może być używany zarówno w trybie interaktywnym, jak i w obrębie programów aplikacyjnych, obsługuje operacje definiowania danych, operacje manipulowania danymi, ograniczenia związane z bezpieczeństwem i integralnością oraz operacje zarządzania transakcjami.
- **Postulat modyfikalności perspektyw** – system musi umożliwiać modyfikowanie perspektyw, o ile jest ono semantycznie realizowalne.

- **Postulat modyfikalności danych** – system musi umożliwiać operacje modyfikacji danych, musi obsługiwać operatory INSERT, UPDATE oraz DELETE.
- **Postulat fizycznej niezależności danych** – zmiany fizycznej reprezentacji danych i organizacji dostępu nie wpływają na aplikacje.
- **Postulat logicznej niezależności danych** – zmiany wartości w tabelach nie wpływają na aplikacje.
- **Postulat niezależności więzów spójności** – więzy spójności są definiowane w bazie i nie zależą od aplikacji.
- **Postulat niezależności dystrybucyjnej** – działanie aplikacji nie zależy od modyfikacji i dystrybucji bazy.
- **Postulat bezpieczeństwa względem operacji niskiego poziomu** — operacje niskiego poziomu nie mogą naruszać modelu relacyjnego i więzów

Elementy relacyjnej bazy danych

- **Encja** – rodzaj obiektu przechowywanego w bazie. Na przykład towar czy producent. Odpowiednikiem w programowaniu obiektowym jest klasa.
- **Atrybut** – każda encja ma swoje właściwości. Na przykład pracownik ma numer telefonu, imię czy nazwisko. Każdy z tych elementów to atrybut. Podobnie jak w programowaniu obiektowym instancję mają swoje atrybuty. Atrybuty mogą mieć różne typy (np. varchar czyli string).
- **Krotka** – pojedyncza krotka to wiersz w tabeli. Zbierając kilka wierszy tworzy się relacja. Np. relacja „ubrania” będzie zawierała wiersze z typami ubrań oraz ich atrybutami.
- **Klucz główny** – zbiór atrybutów (kolumn w tabeli) tworzy klucz główny. Jest to unikalny identyfikator dla każdego wiersza w tabeli. W większości przypadków tabele zawierają dodatkową kolumnę która zawiera identyfikator. Zazwyczaj jest to liczba odpowiadająca numerowi wiersza.
- **Klucz obcy** – przez to że tabele mogą być ze sobą powiązane musimy mieć również klucz obcy. Jest to dodatkowa kolumna (kolumny), która przekazuje zależność. Np. produkty mogą mieć swój klucz główny, a jako klucz obcy będzie ich producent.

KONIEC SLAJD 186!!!

28. Relacja i jej atrybuty w bazach danych.

Rodzaje powiązań w relacyjnej bazie danych

- **Jeden do jednego** – mając dwie tabele A i B występuje wtedy, gdy każdemu rekordowi z tabeli A jest przyporządkowany jeden rekord z tabeli B i na odwrót. Np. numer rejestracyjny i samochód.
- **Jeden do wielu** – jest najczęściej używanym typem połączeń. Pomiędzy tabelami A i B występuje wtedy, gdy pojedynczemu rekordowi z tabeli A jest podporządkowany jeden lub wiele rekordów z tabeli B, natomiast pojedynczemu rekordowi z tabeli B jest przyporządkowany dokładnie jeden rekord z tabeli A.
- **Wiele do wielu** – pomiędzy tabelami A i B występuje wtedy, gdy pojedynczemu rekordowi z tabeli A jest przyporządkowany jeden lub wiele rekordów z tabeli B i na odwrót. Taka sytuacja będzie np. w relacji nauczycieli do uczniów. Każdy nauczyciel ma wielu uczniów i każdego ucznia uczą różni nauczyciele

Klucz główny

Dość często spotykanym problemem na etapie projektowania bazy danych jest określenie, która kolumna (kolumny) będzie pełnić funkcję klucza głównego. Ponieważ każdy wiersz w tabeli musi jednoznacznie zidentyfikować, zachodzi potrzeba wybrania atrybutów (kolumn), które spełniają to zadanie. Klucz główny odgrywa bardzo ważną rolę w tabeli (relacji), dlatego jego wybór powinien zostać poprzedzony analizą typowanych przez nas kolumn pod kątem wymienionych poniżej własności:

- **trwałość** – wartość kolumny powinna być stale obecna w wierszu, oznacza to, że kolumna taka (należąca do klucza głównego) nie może zawierać wartości NULL.
- **unikatowość** – wartość klucza dla każdego wiersza powinna być unikatowa, ponieważ w niepowtarzalny sposób powinien on identyfikować każdą krotkę (wiersz tabeli). Może się zdarzyć, że taki niepowtarzalny identyfikator otrzymamy, umieszczając w kluczu głównym więcej niż jedną kolumnę. Kombinacja wartości, trzech kolumn, które należą do klucza, będzie unikatowa i jednoznacznie zidentyfikuje każdą krotkę.
- **stabilność** – wartości klucza nie powinny podlegać zmianom. Nie powinno się jako kluczy głównych używać kolumn przechowujących wartości nietrwałe, np. numer telefonu komórkowego, ponieważ mimo jego unikatowości każdy człowiek może go zmienić.

Aby jednoznacznie zidentyfikować wiersz tabeli, stosuje się atrybut (kolumnę), której poszczególne wartości dla kolejnych krotek (wierszy) będą niepowtarzalne

Atrybut będący kluczem głównym możemy stworzyć sztucznie dla przykładu wprowadzając kolejne numerowanie wierszy 1, 2, 3, 4, 5 itd., pod warunkiem że każdy wiersz ma inny numer. Możemy również posłużyć się określona cechą (atrybutem) opisywanej rzeczywistości (encji), np. dokonując spisu ludności, możemy posłużyć się numerem PESEL. Ponieważ każdy człowiek ma inny niepowtarzalny numer PESEL, nie zachodzi obawa, że może on się powtórzyć. Taką kolumnę (atrybut) nazywamy kluczem Głównym (primary key).

Rodzaje kluczy

- **klucz prosty** – to taki, który jest jednoelementowy (składa się z jednej kolumny),
- **klucz złożony** – to taki, który jest kilkuelementowy (składa się z więcej niż jednej kolumny).

Kluczem może być zatem jedna lub kilka kolumn, które wspólnie będą w stanie jednoznacznie zidentyfikować pozostałe dane w tabeli (relacji). Kolumny, które należą do kluczy (używa się ich do jednoznacznej identyfikacji wierszy w tabeli), nazywamy atrybutami podstawowymi. Kolumny nie należące do kluczy (zawierają dane, które w określonej relacji są przedmiotem opisu) nazywamy atrybutami opisowymi.

Do łączenia dwóch tabel (np. A i B) za pomocą związków używa się klucza. Klucz pochodzący z obcej tabeli B (w której jest on kluczem głównym), używany do łączenia tej tabeli z tabelą A, będzie dla tabeli A kluczem obcym.

Superklucz (superkey) – to kolumna lub zestaw kolumn jednoznacznie identyfikujących każdą krotkę tabeli. Super klucz może zawierać kolumny, które samodzielnie mogą nie identyfikować każdej z krotek. Unikatowa identyfikacja każdej z krotek może odbywać się jedynie przez zestaw np. dwóch lub trzech atrybutów. Przedmiotem zainteresowań projektantów baz danych jest taki superklucz, który zawiera minimalny zestaw atrybutów unikatowo identyfikujących krotkę.

Klucz kandydujący (nadklucz, klucz potencjalny) o super klucz zawierający minimalną liczbę kolumn unikatowo identyfikujących krotki relacji. W praktyce to kolumna lub kolumny, których użycie w charakterze klucza głównego jest rozważane przez projektanta baz danych. To właśnie twórca bazy danych decyduje, której kolumnie (kolumnom) nada funkcję klucza głównego.

Klucz główny (primary key) to klucz, który został wybrany, aby unikatowo identyfikować krotki tabeli. Klucz główny jest podyktowany wyborem projektanta bazy danych. h

29. Spójność referencyjna baz danych.

Spójność referencyjna baz danych polega na wprowadzeniu i utrzymaniu powiązań pomiędzy tabelami.

To zespół reguł, które gwarantują logiczną spójność danych wprowadzanych i przechowywanych w bazie. Zadaniem więzów spójności jest zagwarantowanie tego, aby dane w bazie danych wiernie odzwierciedlały świat rzeczywisty, dla opisu którego baza danych została zaprojektowana. Więzy spójności definiowane są na etapie projektowania bazy danych, tworzone wraz z

innymi obiektami, przy tworzeniu bazy.

Wyróżniamy dwa typy więzów spójności:

- **Spójność encji** - ograniczają możliwe wartości, jakie mogą się pojawić w wierszu tabeli:
 - **Więzy klucza głównego PRIMARY KEY** – wartości w określonych kolumnach jednoznacznie identyfikują wiersz tabeli. W kolumnach klucza głównego nie jest dozwolona pseudo-wartość NULL . Automatycznie jest zakładany indeks na kolumnach tworzących klucz główny. Może być określony tylko jeden klucz główny dla jednej tabeli.
 - **Więzy klucza jednoznacznego UNIQUE** – wartości w określonych kolumnach jednoznacznie identyfikują wiersz tabeli. W kolumnach klucza jednoznacznego jest dozwolona pseudo-wartość NULL . Automatycznie jest zakładany indeks na kolumnach tworzących klucz jednoznaczny. Może być określony więcej niż jeden klucz jednoznaczny dla jednej tabeli.
 - **Więzy NOT NULL** – w kolumnie nie jest dozwolona pseudo-wartość NULL.
 - **Więzy CHECK** – warunek, który ma być prawdziwy dla wszystkich wierszy w tabeli. Nie może zawierać podzapytania ani funkcji zmiennych w czasie, jak Sysdate lub User. Może zawierać nazwy jednej lub więcej kolumn.
- **Spójność referencyjna** - zapewniają, że zbiór wartości w kolumnach klucza obcego jest zawsze podzbiorem zbioru wartości odpowiadającego mu klucza głównego lub jednoznacznego. Ponieważ wartości klucza głównego lub jednoznacznego jednoznacznie określają obiekty, więc klucz obcy wskazuje zawsze na istniejący obiekt. Wartością klucza obcego może też być NULL – wówczas klucz obcy nie wskazuje na żaden obiekt. System zapewnia, aby obiekt wskazywany przez wartość klucza obcego zawsze istniał, niezależnie od wszystkich możliwych operacji na tabelach, w których biorą udział klucze główne, jednoznaczne i obce.

30. Normalizacja relacji - postaci normalne.

Normalizacja baz danych - proces w ramach którego doprowadzamy bazę danych do postaci normalnych. W przypadku gdy baza danych nie jest znormalizowana występuje redundancja danych.

Redundancja danych w najprostszym wytłumaczeniu jest sytuacją gdy dane się powtarzają np. są zdublowane.

Pierwsza postać normalna - występuje gdy każda kolumna jest atomowa tzn. nie zawiera list i dane są niepodzielne.

IMIE	NAZWISKO	NAZWA	ROZMIAR	CENA	ADRES	DATA_ZAMOWIENIA
Jan	Kowalski	Margherita	XXL	25	ul. Hery 5, Warszawa	1.01.2019
Piotr	Nowak	Margherita	L	15	ul. Domaniewska 2, Poznań	2.01.2019
Piotr	Nowak	Pepperoni	L	15	ul. Domaniewska 2, Poznań	2.01.2019
Anna	Zaradna	Hawajska	L	15		3.01.2019
Anna	Zaradna	Margherita	XL	25		3.01.2019
Anna	Zaradna	Wiejska	XXL	30		3.01.2019
Anna	Zaradna	Margherita	L	15	ul. JP2, Warszawa	5.01.2019
Kamila	Zaradna	Pepperoni	L	15	ul. JP2, Wrocław	6.01.2019

Przedstawiona tabela nie spełnia pierwszej postaci normalnej (1NF) ponieważ kolumna Adres nie jest atomowa. Możemy ją podzielić na pojedyncze kolumny. Aby więc doprowadzić naszą tabelę do 1NF należy kolumnę adres podzielić na kilka pojedynczych kolumn. Poniższa tabela została tak zmieniona, że spełnia 1NF:

IMIE	NAZWISKO	NAZWA	ROZMIAR	CENA	ULICA	NR BLOKU	NR MIESZKANIA	MASTO	DATA_ZAMOWIENIA
Jan	Kowalski	Margherita	XXL	25	Hery	5		Warszawa	1.01.2019
Piotr	Nowak	Margherita	L	15	Domaniewska	2		Poznań	2.01.2019
Piotr	Nowak	Pepperoni	L	15	Domaniewska	2		Poznań	2.01.2019
Anna	Zaradna	Hawajska	L	15					3.01.2019
Anna	Zaradna	Margherita	XL	20					3.01.2019
Anna	Zaradna	Wiejska	XXL	25					3.01.2019
Anna	Zaradna	Margherita	L	15	JP2			Waszawa	5.01.2019
Kamila	Zaradna	Pepperoni	L	15	JP2			Wrocław	6.01.2019

Druga postać normalna - baza danych jest w drugiej postaci normalnej gdy spełnia pierwszą postać normalną oraz wszystkie kolumny w tabeli zależą tylko od klucza.

Czy powyższa baza jest w 2NF? Nie ponieważ jak się zastanowić to z tabeli możemy wyodrębnić przynajmniej trzy zbiory danych zależne od różnych kluczy: KLIENT, PIZZA, ZAMÓWIENIE.

ID	NAZWA	ROZMIAR	CENA
1	Margherita	XXL	25
2	Margherita	L	15
3	Pepperoni	L	15
4	Hawajska	L	15
5	Margherita	XL	20
6	Wiejska	XXL	25

ID_PIZZA	ID_KLIENT	ID_ADRESU	DATA_ZAMOWIENIA
1	1	1	1.01.2019
2	2	2	2.01.2019
3	2	2	2.01.2019
4	3		3.01.2019
5	3		3.01.2019
6	3		3.01.2019
2	3	3	5.01.2019
3	4	4	6.01.2019

ID_KLIENT	IMIE	NAZWISKO
1	Jan	Kowalski
2	Piotr	Nowak
3	Anna	Zaradna
4	Kamila	Zaradna

Powyższa baza została sprowadzona do drugiej postaci normalnej.

Trzecia postać normalna - Baza jest w trzeciej postaci normalnej wtedy gdy spełnia drugą postać normalną oraz żadna z kolumn nie jest zależna od innej kolumny która nie jest kluczem.

Tabela KLIENT spełnia 3NF natomiast tabela PIZZA jej nie spełnia ponieważ kolumna CENA nie jest zależna od klucza a od wielkość i pizzy. Aby to zmienić należy dane dotyczące cen wyciągnąć do inny tabeli jak na poniższym schemacie:

ID	NAZWA	ROZMIAR
1	Margherita	XXL
2	Margherita	L
3	Pepperoni	L
4	Hawajska	L
5	Margherita	XL
6	Wiejska	XXL

ROZMIAR	CENA
L	15
XL	20
XXL	25

ID	IMIE	NAZWISKO
1	Jan	Kowalski
2	Piotr	Nowak
3	Anna	Zaradna
4	Kamila	Zaradna

ID_PIZZA	ID_KLIENT	ID_ADRESU	DATA_ZAMOWIENIA
1	1	1	1.01.2019
2	2	2	2.01.2019
3	2	2	2.01.2019
4	3		3.01.2019
5	3		3.01.2019
6	3		3.01.2019
2	3	3	5.01.2019
3	4	4	6.01.2019

ID_ADRESU	ULICA	NR_BLOKU	R_MIESZKANIE	MIASTO
1	Hery	5		Warszawa
2	Domaniewska	2		Poznań
3	JP2			Warszawa
4	JP2			Wrocław

Kolejne postacie normalne W bazach danych występują jeszcze inne postaci normalne jak: Byce-Codde, 4NF, 5NF. Kolejne postaci normalne mówią, że naszą bazę można jeszcze bardziej podzielić. Dla przykładu miasto nie jest bezpośrednio związane z adresem a z ulicą na którą zamawiamy dlatego też ulicę można by wyciągnąć do osobnej tabeli.

31. Modelowanie bazy danych - rodzaje połączeń relacyjnych, pojęcie klucza głównego i obcego.

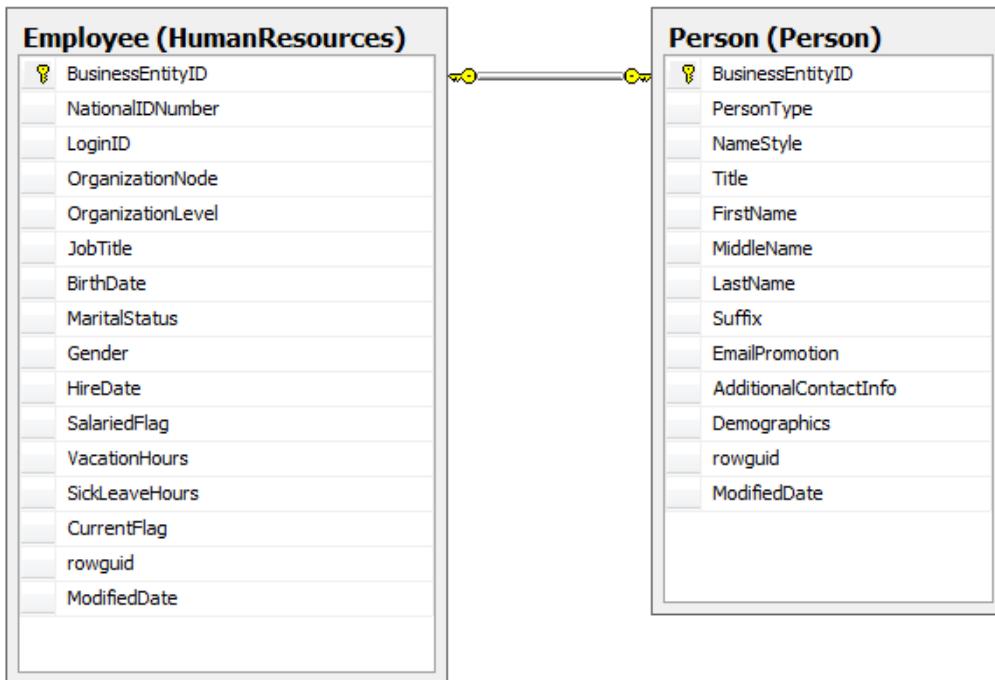
Klucz potencjalny - minimalny zestaw atrybutów relacji, jednoznacznie identyfikujący każdą krotkę tej relacji. W relacji może znajdować się wiele kluczy potencjalnych (zwanych czasem **kandydującymi**). Spośród kluczy potencjalnych wybiera się zazwyczaj jeden klucz, zwany kluczem głównym.

Klucz główny (primary key) – wybrany minimalny zestaw atrybutów relacji, jednoznacznie identyfikujący każdy rekord tej relacji. To oznacza, że taki klucz musi przyjmować **wyłącznie** wartości niepowtarzalne i nie może być wartością pustą (null). Ponadto każda relacja może mieć najwyższej **jeden** klucz główny.

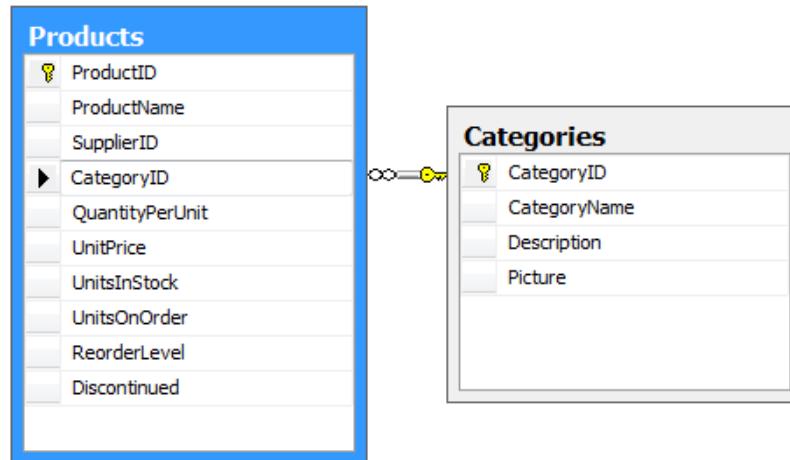
Kluczem głównym może być dowolny klucz potencjalny, ale często stosuje się rozwiązywanie polegające na utworzeniu specjalnego atrybutu, którego wartości domyślne pobierane są z sekwencji (tzw. autonumeracja), tak aby zapewnić unikalność klucza.

Klucz obcy - kombinacja jednego lub wielu atrybutów tabeli, które wyrażają się w dwóch lub większej liczbie relacji (tabel). Wykorzystuje się go do tworzenia powiązania pomiędzy parą tabel, gdzie w jednej tabeli ten zbiór atrybutów jest kluczem obcym, a w drugiej kluczem głównym.

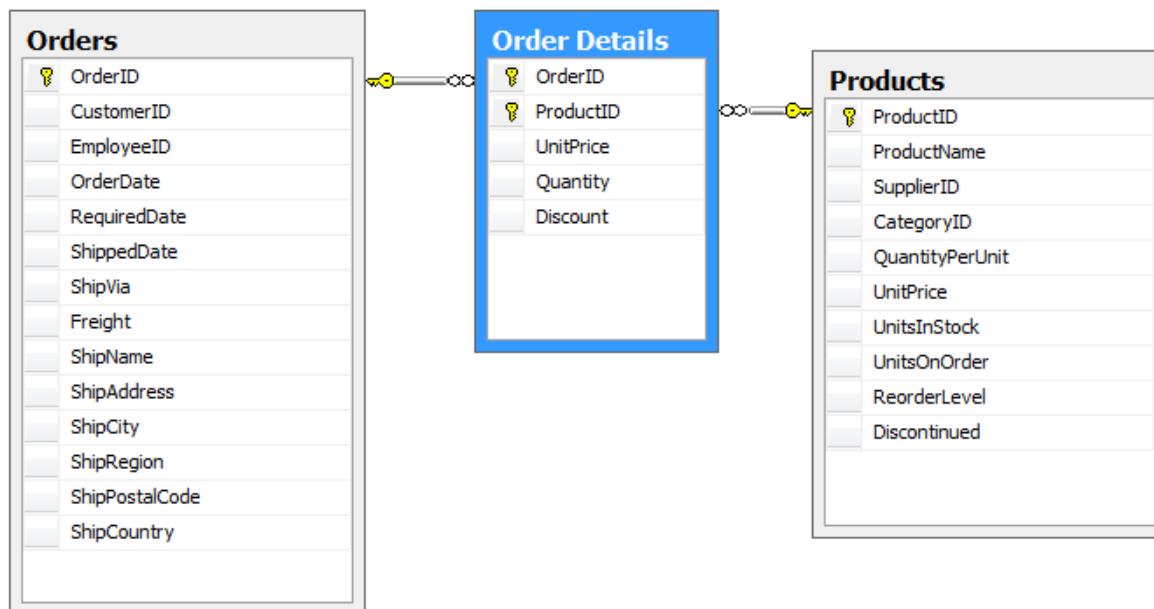
Związek 1:1 (jeden do jednego)



Relacja 1:N (jeden do wielu)



Relacja N:N (wiele do wielu)



32. Pojęcie indeksu - rodzaje i zastosowanie.

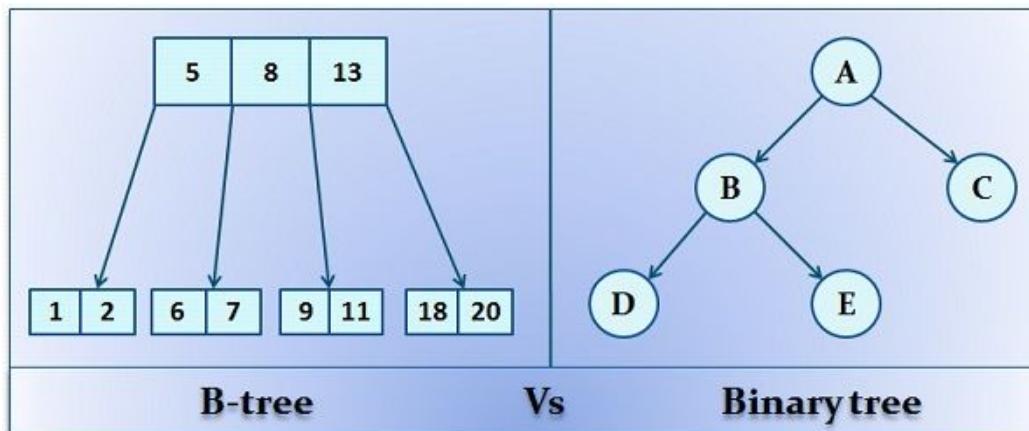
Indeksowanie jest podstawowym mechanizmem wykorzystywanym w celu optymalizacji baz danych SQL. Gdyby porównać bazę danych do książki, indeksy są czymś w rodzaju spisu treści.

Z technicznego punktu widzenia (i mocno uogólniając) indeksy to zbiór wartości typu „klucz – lokalizacja”. Dzięki temu, na podstawie konkretnego klucza czyli parametrów zapytania jest możliwe bardzo szybkie zwrócenie odpowiednich danych.

Klucze w indeksie przechowywane są w strukturze zwanej B-Tree (nie należy mylić tej struktury danych z drzewem binarnym). B-Tree pozwala mechanizmom SQL Server znaleźć pożądany rekord szybciej i wydajniej, ale tylko wtedy, gdy wyszukiwanie w tym drzewie odbywa się za pomocą klucza.

B-drzewo i Drzewo binarne to typy nieliniowej struktury danych. Chociaż warunki wydają się być podobne, ale różnią się pod każdym względem.

Drzewo binarne jest używane, gdy rekordy lub dane są przechowywane w pamięci RAM zamiast na dysku, ponieważ prędkość dostępu do pamięci RAM jest znacznie większa niż na dysku. Z drugiej strony B-tree jest używane, gdy dane są przechowywane na dysku, skracając czas dostępu, zmniejszając wysokość drzewa i zwiększając gałęzie w węźle.



Problem z przeszukiwaniem baz danych polega na tym, że tabele nie są posortowane. Jedyna „kolejność” to często klucz

główny PRIMARY_KEY. Nie jest to przydatna kolejność kiedy szukamy danych nie po kluczu a po jakimś innym polu np: SELECT nazwa_produktu FROM produkty WHERE cena = 128;

W przypadku kiedy dane są posortowane po kluczu głównym trzeba „sprawdzić” wszystkie rekordy i nie mamy możliwości ułatwienia sobie zadania bo produkty mogą mieć przecież różną cenę. Mówiąc wtedy że dokonujemy pełnego skanu tabeli, który działa niekorzystnie na wydajność. Indeksowania polega na unikaniu tego pełnego skanu.

Jak działają indeksy?

Indeks jest uporządkowanym plikiem rekordów indeksu o stałej długości. Rekordy indeksu zawierają dwa pola: klucz reprezentujący jedną z wartości występujących w atrybutach indeksowych relacji oraz wskaźnik do bloku danych zawierający krotkę, której atrybut indeksowy równy jest kluczowi.

Jak np. MySQL używa indeksów

- szybko znajduje wiersze pasujące do klauzuli
- ignoruje pewne wiersze (jeżeli MySQL ma do dyspozycji wiele indeksów używa tego najmniejszego) co przyspiesza skanowanie,
- szybciej zwraca zapytania w przypadku łączenia wielu tabel,
- szybciej zwraca zapytania MIN() i MAX()

Jakie kolumny i tabele należy indeksować

Korzyść wydajnościowa ze stosowania indeksów jest największa w przypadku dużych tabel (zawierających najwięcej rekordów) oraz zapytań, które wykonywane są najczęściej.

W MySQL zaleca się indeksować następujące kolumny: -kolumny najczęściej padające po słowie WHERE,

- kolumny dwóch tabel, które często łączymy,
- kolumny, według których sortujemy dane w raportach (kolumny padające po słowie ORDER BY i GROUP BY),
- kolumny które często zliczamy (SUM(), AVG(), MIN(), MAX(), COUNT())
- klucze obce i kolumny, których będziemy używać tak jak klucz obcych,
- klucze niepowtarzalne UNIQUE_KEY (typu NIP, PESEL itd...),

Nadmiar indeksów

Należy pamiętać, że indeks drastycznie **spowalnia dodawanie, modyfikowanie i usuwanie danych**, ponieważ indeksy muszą być aktualizowane za każdym razem, gdy tabela ulega nawet najmniejszej modyfikacji. Najlepszą praktyką jest dodanie indeksu dla wartości, które są **często używane do wyszukiwania, ale nie ulegają częstym zmianom**.

Rodzaje indeksów

- Indeks pogrupowany (clustered)

W jednej tabeli możemy posiadać tylko jeden index klastrowany dla jednej lub wielu kolumn. Założenie takiego indexu równa się fizycznemu posortowaniu danych na dysku, w związku z tym niemożliwe jest założenie dwóch tego rodzaju indexów.

Odczyt danych z tabeli która nie jest posortowana w przypadku gdy posiada setki tysięcy bądź miliony rekordów jest bardzo czasochłonne. Serwer bazodanowy musi przejść rekord po rekordzie, by zwrócić dane o które prosimy. W związku z tym należy założyć index, na przynajmniej jedną kolumnę aby usprawnić proces „poszukiwania” rekordów.

CREATE CLUSTERED INDEX nazwalndeksu ON nazwaTabeli(nazwaKolumny);

Index klastrowany może być tylko jeden, ale może zostać założony na więcej niż jedną kolumnę

CREATE CLUSTERED INDEX IX_EmployeesAgeFullName ON Employees(Age,FullName);

Nasza tabela została posortowana najpierw po kolumnie Age a następnie po kolumnie FullName

- Indeks niepogrupowany (non clustered)

Są swego rodzaju kopią danych, kopią kolumny na którą został założony taki index. Możemy posiadać więcej niż jeden index non-clustered, jednak warto wiedzieć, że podczas zapisu danych do tabeli, jeśli dane wymagają ponownego posortowania, to operacja zapisu będzie trwać dłużej. Im więcej indexów, tym dłuższy czas oczekiwania na ukończenie operacji. Tego typu index jest również wolniejszy jeżeli odpytujemy o więcej danych, niż zostało to zadeklarowane na początku.

CREATE NONCLUSTERED INDEX nazwalndeksu ON nazwaTablicy(nazwaKolumny);

33. Podstawowe konstrukcje języka SQL.

Cechy języka SQL

- Język wysokiego poziomu
- Jest językiem deklaratywnym, zorientowanym na wynik
- Jest oparty na algebrze relacji
- Zawiera logikę trójwartościową
- Nie posiada instrukcji sterujących wykonywaniem programu
- Nie dopuszcza rekurencji
- Umożliwia definiowanie struktur danych, wyszukiwanie danych oraz operacje na danych
- Działa na zbiorach danych

Struktura i wykorzystanie języka SQL

- Język SQL jest przykładem języka transformacji, co oznacza, że został on tak zaprojektowany, że umożliwia przekształcenie relacji wejściowych na relację wyjściową.
- Jest językiem nieproceduralnym, w którym użytkownik opisuje informację, której potrzebuje, ale nie wskazuje on przy tym, w jaki sposób należy ją odnaleźć.
- Zapytanie języka SQL składa się ze słów zarezerwowanych oraz ze słów zdefiniowanych przez samego użytkownika. Należy je zapisywać w sposobie dokladny, bez jakichkolwiek zmian, tj. dokładnie tak jak wymaga tego składnia języka SQL.
- Każde zapytanie w języku SQL jest kończone średnikiem.

Komponenty języka SQL

- **DDL (Data Definition Language)** – język definiowania struktur danych (CREATE, DROP, ALTER TABLE).
- **DML (Data Manipulation Language)** – język operacji na danych (SELECT, INSERT, UPDATE, DELETE).
- **Instrukcje sterowania danymi** – kontrola uprawnień użytkowników (GRANT, REVOKE).

Tabele w języku SQL

Do manipulowania tabelkami używa się kilku poleceń:

- **CREATE TABLE** – definiuje tabelę i jej kolumny,
- **ALTER TABLE** – zmienia tabelę i kolumny,
- **DROP TABLE** – usuwa tabelę – jej definicję i zawartość,
- **RENAME** – zmienia nazwę tabeli

Polecenia **CREATE TABLE** i **ALTER TABLE** są ponadto używane w celu definiowania ograniczeń kluczowych oraz tzw. parametrów przechowywania, które są rozszerzeniami składni Oracle.

Typy danych // Typy tekstowe

- **VARCHAR2(L)** – oznacza typ danych, za pomocą którego można przechowywać ciągi znaków o zmiennej długości, gdzie L oznacza określoną maksymalną długość zmiennej tego typu. Dopuszczalny rozmiar dla zmiennej VARCHAR2 wynosi 4000 bajtów.

- **VARCHAR(L)** – pozwala przechowywać napisy o zmiennej długości, których długość może wahać się od 1 do 2000 znaków.
- **CHAR(L)** – pozwala przechowywać ciągi znaków o stałej długości wskazanej w parametrze rozmiar. Maksymalna wielkość to 2000 bajtów. Długość domyślna to 1 znak. Ciągi są dopełniane z prawej strony spacjami do pełnej wielkości pola.

Typy numeryczne

- **NUMBER(zakres)** – typ całkowity – pozwala przechowywać liczby całkowite ze znakiem mającą tyle cyfr, ile wynosi parametr zakres (o maksymalnej długości 38 cyfr).
- **NUMBER(zakres, dokładność)** – określa dziesiętną liczbę stałoprzecinkową zapisywaną z dokładnością do maksymalnie 38 cyfr i wykładnikiem pomiędzy -84 a 127.
- **NUMBER** – określa kolumnę zmiennoprzecinkową z 38 cyframi pamiętanymi dokładnie i wykładnikiem pomiędzy 125 a -127.

Typ czasu

- **DATE** – obejmuje okres od pierwszego stycznia 4712 r. p.n.e. do 31 grudnia 4712 r. n.e.
Kolumna typu DATE może przechowywać czas z dokładnością do sekund.

Operacje DDL

Tworzenie nowej tabeli **CREATE TABLE nazwa_tabeli (nazwa_kolumny1 typ_danych1);**

```
nr_towar NUMBER(4),
nazwa VARCHAR2(15),
cena NUMBER(7,2),
kategoria VARCHAR2(30),
stan_magazyn NUMBER,
wycofany CHAR(3) );
```

Wartość NOT NULL i wartość domyślna

- Wartość NOT NULL dla pola By w danym polu tabeli nie mogła wystąpić wartość pusta należy po zdefiniowaniu typu danych pola podać wyrażenie NOT NULL. Wyrażenie NOT NULL wskazuje właśnie, iż kolumna ta nie zaakceptuje wartości pustej NULL.
- W celu ustalenia dla kolumny tabeli wartości domyślnej należy użyć słowa kluczowego DEFAULT, po którym podajemy wartość domyślną. Wartość domyślna może być stałą bądź pseudofunkcją.

```
CREATE TABLE towary (
nr_towar NUMBER(4),
nazwa VARCHAR2(15),
cena NUMBER(7,2),
kategoria VARCHAR2(30),
stan_magazyn NUMBER,
wycofany CHAR(3) );
```

Sposoby usuwania tabel i ich zawartości

Tabele i ich zawartość są usuwane za pomocą polecenia **DROP TABLE nazwa_tabeli**. Dla tabel, do których nie odwołuje się żaden klucz obcy, tabela i jej zawartość zostanie całkowicie usunięta.

- **DROP TABLE** czytelnicy;
- Gdy usuwana tabela jest połączona więzami referencyjnymi (integralności) z innymi tabelami, należy użyć polecenia **DROP TABLE nazwa_tabeli CASCADE CONSTRAINTS**.

Przykład: **DROP TABLE** czytelnicy **CASCADE CONSTRAINTS**:

- W celu natychmiastowego i bezpowrotnego usunięcia zawartości tabeli należy użyć polecenia **TRUNCATE TABLE nazwa_tabeli**. Polecenie **TRUNCATE TABLE** czyści zawartość tabeli bezpowrotnie (ale nie usuwa jej fizycznie), ponieważ mechanizm anulowania jest nieaktywny.

Przykład: **TRUNCATE TABLE** towary;

Zmiana nazwy tabeli i dodawanie kolumny do tabeli

- Zmiana nazwy tabeli:
RENAME stara_nazwa_tabeli TO nowa_nazwa_tabeli;
- Dodawanie kolumny do tabeli Dodawanie kolumn jest wykonywane za pomocą opcji **ADD** polecenia **ALTER TABLE**.
ALTER TABLE nazwa_tabeli ADD (nazwa_kolumny typ_kolumny);

Zmiana definicji kolumny tabeli

Z pewnymi ograniczeniami, przy użyciu opcji **MODIFY** polecenia **ALTER TABLE** można zmienić cztery części definicji kolumny:

- Kolumna może być zmieniona na dowolny prawidłowy typ danych, jeśli nie jest wypełniona w tym znaczeniu, że każdy wiersz musi zawierać wartość **NULL** w tej kolumnie. W innym przypadku kolumna typu **VARCHAR2** może być zmieniona na **CHAR** tego samego rozmiaru i na odwrót.
- Kolumna, która nie jest wypełniona danymi, może być zmieniona na dowolny prawidłowy wymiar. Rozmiar i precyzja kolumny już wypełnionej danymi nie może ulec zmniejszeniu.
- Ograniczenia **NOT NULL** mogą być dodane, jeśli ani jeden wiersz nie posiada ograniczenia **NULL** w tej kolumnie. Ograniczenia **NOT NULL** mogą być również usuwane.
- Można zmieniać wartości domyślne.

Przykład: **ALTER TABLE** towary **MODIFY (kategoria VARCHAR(30) NOT NULL)**;

Usunięcie **kolumny z tabeli**:

Przykład: **ALTER TABLE nazwa_tabeli DROP COLUMN nazwa_kolumny**;

Ograniczenia tabel/kolumn

Ograniczenia, jakie można ustawić dla tabel/kolumn służą następującym celom:

- Ograniczają wartości, które mogłyby zostać wstawione do kolumny lub zestawu kolumn.

- Przyśpieszają lub mogą przyśpieszać pobieranie pojedynczych wierszy lub zestawów wierszy.

Ograniczenia mogą być:

1. **Statyczne**, które limitują wartość lub zakres wartości, które mogą być wstawione (np. **CHECK**, **NOT NULL**),
2. **Dynamiczne**, w relacji ze wszystkimi wartościami kolumny, ograniczając w ten sposób nowe wartości tylko do takich wartości, które nie występują w pewnych kolumnach lub zestawach kolumn (klucz unikalny, klucz podstawowy).
3. **Dynamiczne**, w relacji z inną tabelą, pozwalając w ten sposób na wstawienie jedynie takich wartości, które występują także w innych kolumnach w innej (lub tej samej) tabeli (klucz obcy).
4. **Indeksami** – przyśpieszają one pobieranie danych. Ponadto indeksy tabeli są używane także w celu zapewnienia unikalnej wartości w kolumnie unikalnego lub podstawowego klucza.

Ograniczenia CHECK

Istnieje kilka możliwości wykorzystania ograniczenia **CHECK**:

- **kolumny wymagane** – zapobiega wprowadzaniu wartości niezidentyfikowanych do tych pól,
- **prawidłowe zakresy** – możliwość ograniczenia wprowadzanych wartości od określonych zakresów,
- **zakresy kodów** – możliwość ustawienia odpowiedniej struktury jednoznacznych kodów,
- **reguły biznesowe** – określenie pewnych reguł i wymuszenie pewnych zależności.

Ograniczenie **CHECK** jest definiowane:

- przy użyciu podklauzuli **CHECK**(warunek)
- w poleceniu **CREATE TABLE** lub w poleceniu **ALTER TABLE** nazwa tabeli **ADD**.

Przykład:

```
CREATE TABLE drużyny_pilkarskie (
nr_drużyny NUMBER(4) NOT NULL,
nazwa VARCHAR2(30) NOT NULL,
liczba_goli NUMBER NOT NULL,
liczba_goli_dom NUMBER CHECK(liczba_goli_dom <= liczba_goli), liczba_goli_wyjazd
NUMBER CHECK(liczba_goli_wyjazd <= liczba_goli) );
```

Nazywanie ograniczeń Jeśli w tabeli wprowadzamy ograniczenia bez podawania ich nazw, wówczas system sam nadaje każdemu ograniczeniu unikalną nazwę, która jest przechowywana w przestrzeni nazw.

Dla przykładu utworzymy tabelę towary nazywając ograniczenia:

```
CREATE TABLE towary (
nr_towar NUMBER(4),
```

```
nazwa VARCHAR2(15) CONSTRAINT nazwa_nn NOT NULL,
cena NUMBER(7,2) CONSTRAINT cena_nn NOT NULL,
kategoria VARCHAR2(30) CONSTRAINT kategoria_nn NOT NULL, stan_magazyn
NUMBER DEFAULT 0,
wycofany CHAR(3) DEFAULT 'Nie',
CONSTRAINT towary_pk PRIMARY KEY(nr_towar),
CONSTRAINT wycofany_CH CHECK(wycofany IN ('Tak', 'Nie')));
```

Operacje DML

W języku **SQL** poleceniami służącymi do modyfikowania danych w tabelach są jedynie trzy operacje, które zaliczamy do operacji **DML**:

- wstawianie nowego wiersza do tabeli,
- usuwanie wierszy z tabeli,
- aktualizowanie kolumn w tabeli.

Polecenie **INSERT**

Za pomocą polecenia **INSERT** można:

- utworzyć nowy wiersz w tabeli bazy danych,
- załadować wyszczególnione wartości do wszystkich lub wskazanych kolumn do wskazanej tabeli.
-

Składnia:

- **INSERT INTO** nazwa_tabeli (nazwa_pola1, nazwa_pola2, ..., nazwa_polaN) **VALUES** (wartość1, wartość2, ..., wartośćN);
- **INSERT INTO** nazwa_tabeli **VALUES** (wartość1, wartość2, ..., wartośćN);

Przykład:

```
| INSERT INTO towary (nr_towar, nazwa, cena, nazwa_kategorii, stan magazynu) VALUES (104, 'chleb', 2.20, 'pieczywo',100);
```

Usuwanie wierszy – **DELETE i TRUNCATE**

Składnia **DELETE**:

```
| DELETE FROM nazwa_tabeli WHERE warunek_logiczny;
```

Przykład:

```
DELETE FROM towarywycofane
```

```
WHERE nr_towar IN (SELECT nr_towar  
FROM towary WHERE stan = 0);
```

Polecenie **TRUNCATE TABLE** *nazwa tabeli* przyśpiesza proces usuwania dzięki temu, że nie zapisuje informacji sprzed modyfikacji do przestrzeni wycofania. Ten fakt powoduje jednak, iż nie jest możliwe przywrócenie usuniętych danych. To polecenie nie zawiera także klauzuli **WHERE**, a zatem zawsze usuwa wszystkie wiersze wskazanej tabeli.

Polecenie **UPDATE**

Polecenie UPDATE dotyczące aktualizacji danych różni się od pozostałych dwóch operacji DML dotyczących modyfikacji danych w tabelach, ponieważ nie ma wpływu na liczbę wierszy w tabeli.

Składnia:

```
UPDATE nazwa_tabeli  
SET lista  
WHERE warunek;
```

gdzie:

- **nazwa_tabeli** – wskazuje tabelę, w której przechowywane dane mają zostać zaktualizowane,
- **lista** – wykaz kolumn, które mają zostać zaktualizowane oraz lista wartości, jakie będą przypisane tym kolumnom,
- **warunek** – warunek logiczny, którego spełnienie powoduje, iż dany wiersz dla którego jest spełniony będzie aktualizowany.

Polecenie UPDATE - przykład

```
UPDATE towary SET nazwa = 'czekolada mleczna', cena = 2.50 WHERE nr_towar = 1;
```

Instrukcja **SELECT**

Podstawową, najczęściej używaną instrukcją języka SQL jest instrukcja **SELECT**, która służy do pobierania danych z jednej tabeli lub większej liczby tabel (widoków). Niezależnie od liczby tabel i/lub widoków oraz niezależnie od rodzaju operacji wykonywanych na zbiorach lub pseudozbiorach, zawsze jako wynik otrzymujemy wirtualną pojedynczą tabelę (tzw. dynamiczny zestaw wyników), którą dalej możemy przetwarzać.

Składnia:

```
SELECT lista_kolumn FROM lista_tabel;
```

Pobranie wszystkich wierszy i wszystkich kolumn:

```
SELECT * FROM towary;
```

Pobieranie wszystkich wierszy i wybranych kolumn:

SELECT nazwa, cena **FROM** towary;

Listy przecinkowe

W języku SQL listy przecinkowe są wykorzystywane w różnych celach:

- lista przecinkowa frazy **SELECT** określa, które kolumny mają być wybrane w zapytaniu,
- lista przecinkowa frazy **FROM** podaje tabele, z których dane mają być wybrane,
- lista przecinkowa frazy **GROUP BY** jest używana do agregowania danych w grupach wg podanych kolumn,
- lista przecinkowa frazy **ORDER BY** pozwala ustalić kryteria sortowania danych,
- lista przecinkowa jest wykorzystywana w instrukcji **IN**.

Aliases

Istotna odmiana listy przecinkowej powstaje w wyniku tzw. aliasingu, czyli nadawania **aliasów** („pseudonimów”) dla tych elementów, które bezpośrednio poprzedzają alias. Jeśli alias jest umieszczony za określonym elementem, to do tego elementu można odwołać się zarówno poprzez jego nazwę lub alias (**poza pewnymi przypadkami**, kiedy można odwołać się tylko za pomocą nazwy oraz pewnymi przypadkami, kiedy można odwołać się tylko za pomocą aliasu).

Przykład (nadanie zastępczych aliasów dwóm wyświetlanym kolumnom):

```
SELECT nazwa nazwa_towaru, cena cena_jednostkowa  
FROM towary;
```

Klauzula ORDER BY

Klauzula **ORDER BY** instrukcji **SELECT** służy do sortowania danych w języku SQL. Sortowanie danych można wykonać na dwa sposoby: albo w porządku rosnącym (ustawienie domyślne) – opcja **ASC** lub w porządku malejącym wartości kolumny użytej do sortowania – opcja **DESC**.

Przykład:

```
SELECT nr_towar, nazwa, cena, stan, towary  
FROM towary  
ORDER BY cena DESC;
```

34. Warstwy i funkcje modelu ISO OSI.

PO CO NAM MODELE?

- Zanim dane z urządzenia źródłowego zostaną przesłane do urządzenia końcowego muszą przejść długą drogę, podczas której najpierw są odpowiednio oznaczane, tagowane, opisywane określonymi informacjami pozwalającymi na ich identyfikację, potem przesyłane są pomiędzy wieloma urządzeniami pośredniczącymi, aż trafią do odbiorcy, który dane to potem musi zinterpretować.
- Gdyby nie istniał taki model, który dzieli komunikację na mniejsze, łatwiejsze do zrozumienia i zarządzania etapy oraz określa zadania, jakie muszą być realizowane w poszczególnych warstwach trudno byłoby we właściwy sposób zarządzać komunikacją sieciową ponieważ mnogość rozwiązań i technologii powodowałaby olbrzymi chaos, trudny do opanowania.

Model ISO/OSI składa się z 7 warstw:

- **Warstwa 7: aplikacji** - Udostępnia użytkownikom możliwość korzystania z usług sieciowych, takich jak WWW, poczta elektroniczna, wymiana plików, połączenia terminalowe czy komunikatory.
- **Warstwa 6: prezentacji** - Przekazuje do warstwy aplikacji informacje o zastosowanym formacie danych, np. informuje jakie typy plików będą przesyłane, odpowiada ona również za odpowiednie zakodowanie danych na urządzeniu źródłowym i ich dekodowanie na urządzeniu docelowym.
- **Warstwa 5: sesji** - Warstwa sesji otrzymuje od różnych aplikacji dane, które muszą zostać odpowiednio zsynchronizowane. Synchronizacja występuje między warstwami sesji systemu nadawcy i odbiorcy. Warstwa sesji „wie”, która aplikacja łączy się z którą, dzięki czemu może zapewnić właściwy kierunek przepływu danych –nadzoruje połączenie. Wznawia je po przerwaniu.
- **Warstwa 4: transportu** - Głównym zadaniem jest sprawna obsługa komunikacji pomiędzy urządzeniami. W warstwie tej dane dzielone są na mniejsze części, następnie opatrywane są dodatkowymi informacjami pozwalającymi zarówno przydzielić je do właściwej aplikacji na urządzeniu docelowym, jak i pozwalającymi złożyć je na urządzeniu docelowym w odpowiedniej kolejności.
- **Warstwa 3: sieci** - Warstwa sieciowa jako jedyna dysponuje wiedzą dotyczącą fizycznej topologii sieci. Rozpoznaje, jakie drogi łączą poszczególne komputery (trasowanie) i decyduje, ile informacji należy przesyłać jednym z połączeń, a ile innym. Jeżeli danych do przesłania jest zbyt wiele, to warstwa sieciowa po prostu je ignoruje.
- **Warstwa 2: łącza danych** - Głównym zadaniem jest kontrola dostępu do medium transmisyjnego, a także adresowanie danych, tym razem jednak w celu przesyłania ich pomiędzy hostami w sieci LAN.
- **Warstwa 1: fizyczna** - Koduje dane do postaci czystych bitów (zer i jedynek) i przesyła je poprzez medium transmisyjne do odpowiednich urządzeń.

aplikacji

prezentacji

sesji

transportu

sieci

łącza danych

fizyczna

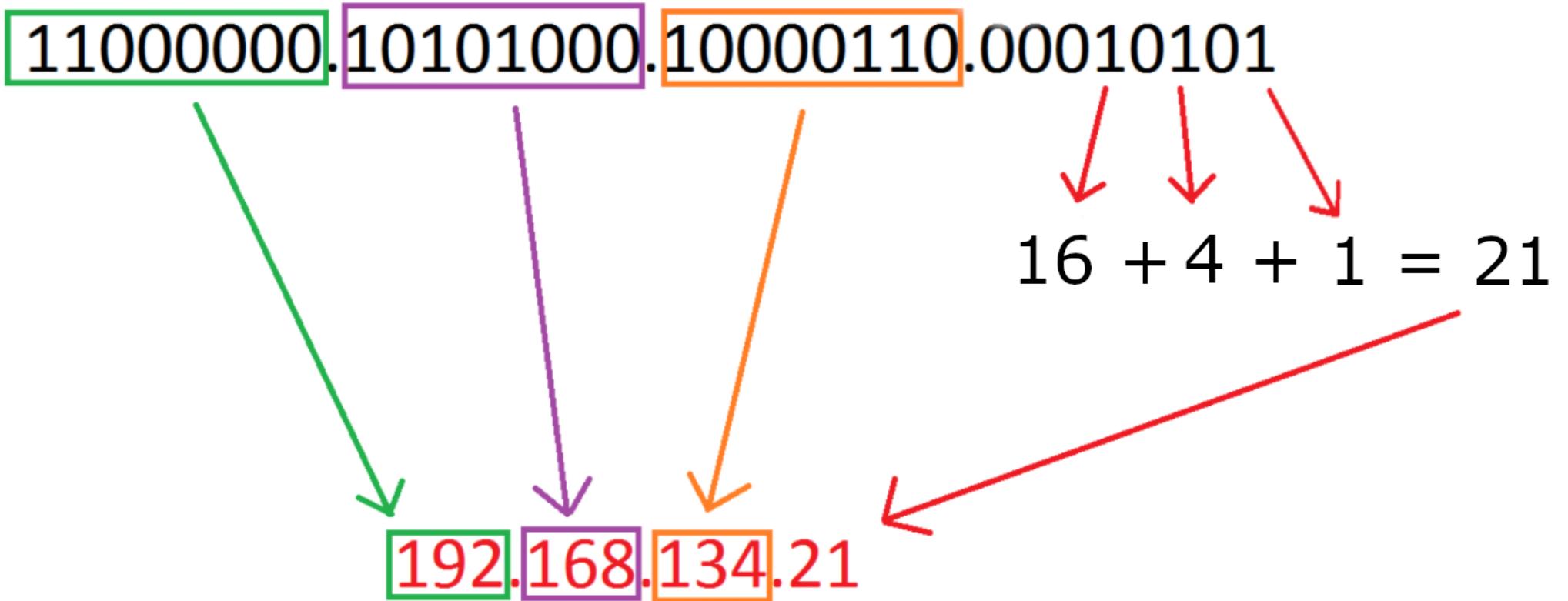
35. Adresowanie logiczne w sieciach komputerowych.

Adresowanie logiczne // Różnice

- Adresowanie **fizyczne** (inaczej sprzętowe) polega na tym że adres fizyczny jest „wypalonym” adresem MAC w układzie ROM karty sieciowej
- Adresowanie **logiczne** z kolei działa w ten sposób, że każdy komputer w sieci ma unikatowy adres IP, którego przydział jest administrowany przez odpowiednie organizacje
- Adresowanie **logiczne** występuje w trzeciej warstwie modelu odniesienia ISO/OSI, czyli w warstwie sieciowej, oraz w warstwie internetowej w modelu TCP/IP.

Adres IP

- Jest unikalny w skali globalnej (poza pewnymi specjalnymi przypadkami)
- Nie musi być jednoznacznie przypisany jednemu urządzeniu (interfejsowi sieciowemu): jeden interfejs może mieć wiele adresów IP, jeden adres IP może przypisany wielu urządzeniom
- Jest 32-bitowy (4 bajty zwane „oktetami”) i zwykle jest zapisywany dziesiętnie
- Jest widoczny „dla użytkownika” w przeciwieństwie do adresu MAC



Podział adresów

- Klasa A: od 0.0.0.0 do 127.255.255.255
- Klasa B: od 128.0.0.0 do 191.255.255.255
- Klasa C: od 192.0.0.0 do 223.255.255.255
- Klasa D: od 224.0.0.0 do 239.255.255.255
- Klasa E: od 240.0.0.0 do 255.255.255.255

Maska sieci

- Jest adresem IP służącym do sprawdzania czy dany adres IP jest adresem z danej sieci, logicznego dzielenia sieci na podsieci (od wyboru maski zależy maksymalna liczba węzłów w danej podsieci) oraz określenia jaka część adresu IP określa adres sieci a jaka adresy węzłów.

Adres rozgłoszeniowy

- Ostatni adres z zakresu adresów IP
- Wszystkie bity służące do numerowania węzłów mają wartość 1
- Jest wykorzystywany przez aplikacje sieciowe zainstalowane na hostach do wysyłania sygnałów do wszystkich użytkowników (węzłów) danej sieci.

Aby skonfigurować protokół IP należy:

- wyznaczyć adres sieci
- wyznaczyć maskę sieci
- wyznaczyć adres rozgłoszeniowy
- określić sposób przydzielania adresów
- określić adres IP domyślnej bramy danej sieci
- określić adresy IP serwerów DNS

36. Najważniejsze protokoły rodziny TCP/IP.

TCP/IP (ang. Transmission Control Protocol/Internet Protocol) to zbiór protokołów służących do transmisji danych przez sieci komputerowe. Model TCP/IP został stworzony w latach 70. XX wieku w [DARPA](#), aby pomóc w tworzeniu odpornych na atak sieci komputerowych. Potem stał się podstawą struktury Internetu. Model TCP/IP implementuje najważniejsze funkcjonalności siedmiu warstw standardowego modelu OSI. Poniższy schemat przedstawia odpowiadające sobie warstwy modeli TCP/IP i OSI.

PORÓWNANIE MODELI

TCP/IP



ISO/OSI

Każda wiadomość wysłana przez aplikację przechodzi przez wszystkie warstwy TCP/IP, od warstwy aplikacji do najniższej warstwy dostępu do sieci. Następnie jest transmitowana przez sieć do drugiego komputera. Na koniec przechodzi przez wszystkie warstwy w przeciwnym kierunku, aż do warstwy aplikacji i docelowego procesu. Podczas przesyłania danych z aplikacji do sieci, każda warstwa dodaje swój własny nagłówek do każdej wiadomości. Każdy z tych nagłówków jest potem odczytywany przez odpowiednią warstwę w komputerze odbierającym wiadomość. Zarówno zawartość jak i wielkość nagłówków zależą od użytych protokołów.

Wysyłanie wiadomości w TCP/IP

Źródłowa Aplikacja

Warstwa Aplikacji

nagłówek
warstwy aplikacji dane
aplikacji

Warstwa Transportowa

nagłówek
warstwy
transportowej nagłówek
warstwy aplikacji dane
aplikacji

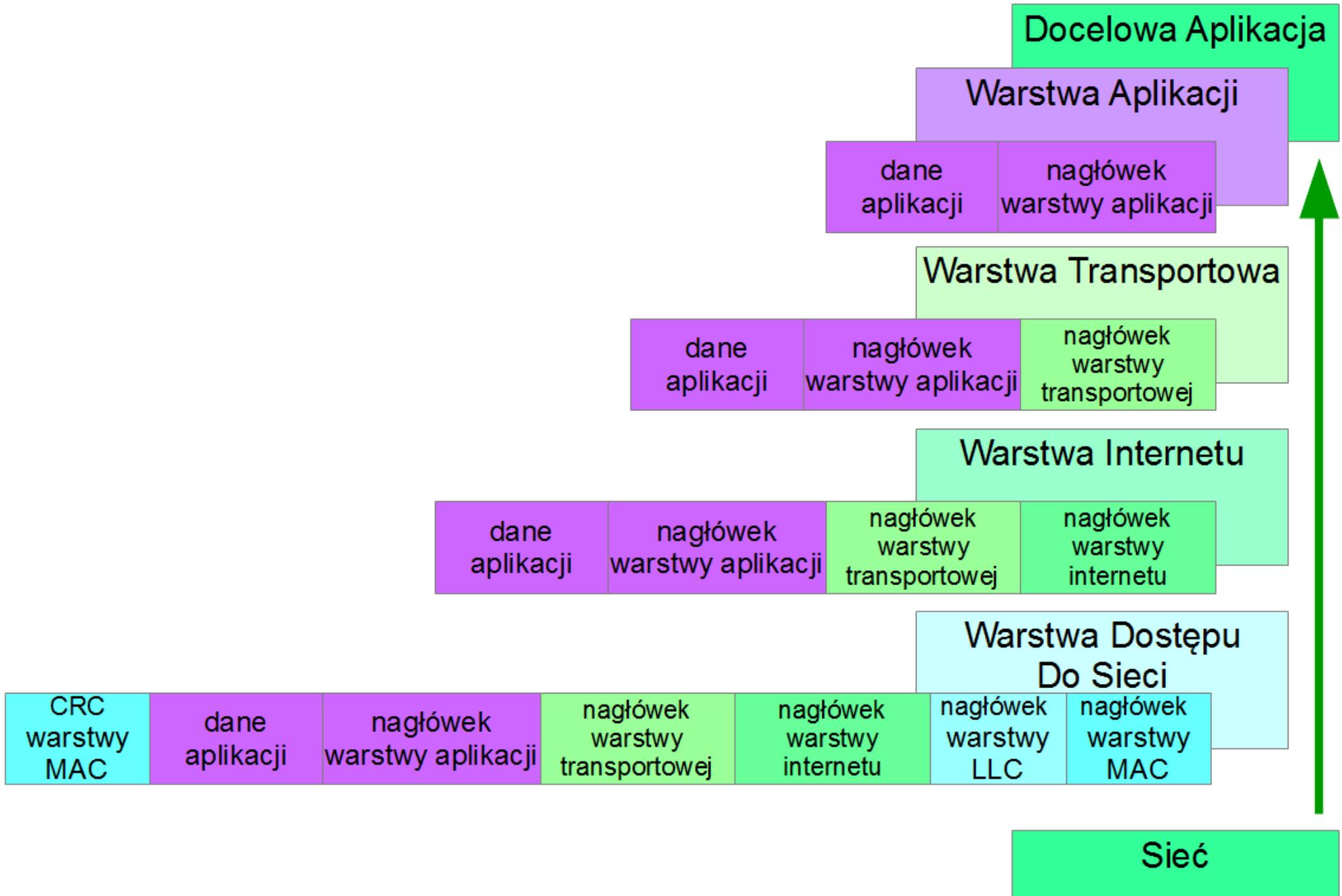
Warstwa Internetu

nagłówek
warstwy
internetu nagłówek
warstwy
transportowej nagłówek
warstwy aplikacji dane
aplikacji

Warstwa Dostępu Do Sieci

nagłówek
warstwy
MAC nagłówek
warstwy
LLC nagłówek
warstwy
internetu nagłówek
warstwy
transportowej nagłówek
warstwy aplikacji dane
aplikacji CRC
warstwy
MAC

Sieć



Pośredniczy w komunikacji pomiędzy programami komputerowymi i protokołami niższych warstw, umożliwiając w ten sposób aplikacjom korzystanie z sieci. Jest to warstwa najbliższa użytkownikowi, ponieważ to właśnie ona pozwala nam w pełni korzystać z usług sieciowych. Kiedy siadamy przed komputerem i uruchamiamy np. przeglądarkę internetową to korzystamy z sieci właśnie na poziomie warstwy aplikacji. Istnieje wiele protokołów warstwy aplikacji, które wykorzystują transmisję TCP/IP.

Jednymi z ważniejszych protokołów warstwy aplikacji są:

- **HTTP, HTTPS** - do przeglądania stron www,
- **FTP, TFTP, NFS** - do transmisji plików,
- **SMTP** - do wysyłania wiadomości email,
- **POP3** - do otrzymywania wiadomości email,
- **IMAP** - do zarządzania wiadomościami email na serwerach,
- **Telnet, rLogin** - do zdalnego logowania się na innych komputerach,
- **SNMP** - do zarządzania sieciami komputerowymi,
- **DNS** - do znajdowania adresów IP przypisanych do adresów WWW,
- **IRC** - do czatów online

Budowa wiadomości warstwy aplikacji różni się w zależności od protokołu, który został użyty. Każdy protokół może wymagać różnych danych wejściowych i produkować różne zapytania, które będą wysłane do warstwy transportowej. Niezależnie od formy wiadomości utworzonej przez warstwę aplikacji, warstwa transportowa traktuje każdą otrzymaną wiadomość jako dane i nie wnika w ich zawartość.

Gniazda sieciowe

Gniazda sieciowe to struktury, które są wykorzystywane podczas komunikacji pomiędzy warstwami aplikacji i transportową. Każdy proces i aplikacja, który próbuje połączyć się z siecią, musi powiązać swoje kanały transmisji danych wejściowych i wyjściowych poprzez utworzenie właściwego obiektu gniazda sieciowego.

Obiekt gniazda sieciowego zawiera informacje o adresie IP, numerze portu i użytym protokole warstwy transportowej. Unikalna kombinacja tych trzech parametrów pozwala na zidentyfikowanie właściwego procesu, do którego wiadomość powinna być dostarczona.

Numer portu może zostać przypisany automatycznie przez system operacyjny, ręcznie przez użytkownika lub może być mu przypisana wartość domyślna, właściwa pewnym popularnym aplikacjom. Numer portu jest 16-bitową liczbą całkowitą (0 - 65535).

Niektóre popularne protokoły warstwy aplikacji używają domyślnych i publicznie znanych numerów porów. Na przykład, HTTP używa portu 80, HTTPS używa portu 443, SMTP portu 25, Telnet portu 23, natomiast FTP używa dwóch portów: 20 do transmisji danych i 21 kontroli transmisji. Lista domyślnych numerów portów jest zarządzana przez organizację Internet Assigned Numbers Authority.

WARSTWA TRANSPORTOWA

Jej głównym zadaniem jest sprawna obsługa komunikacji pomiędzy urządzeniami. W warstwie tej dane dzielone są na mniejsze części, następnie opatrywane są dodatkowymi informacjami (nagłówki) pozwalającymi zarówno przydzielić je do właściwej aplikacji na urządzeniu docelowym, jak i pozwalającymi złożyć je na urządzeniu docelowym w odpowiedniej kolejności. Nagłówek zawiera szereg informacji kontrolnych, w szczególności numery portów nadawcy i odbiorcy.

TCP

Najpopularniejszym protokołem warstwy transportowej jest TCP (ang. Transmission Control Protocol). Podczas transmisji danych, TCP zestawia połączenie pomiędzy komunikującymi się stronami przez zainicjowanie tzw. sesji (ang. session). TCP jest protokołem niezawodnym, w którym odbiorca potwierdza otrzymanie każdej wiadomości. Wszystkie wiadomości dostarczane są w takiej samej kolejności, w jakiej zostały wysłane. Wszystkie cechy wymienione powyżej są zapewniane przez warstwę TCP. Oznacza to, że TCP może współdziałać z innymi, bardziej zawodnymi protokołami niższych warstw i nie powinno to afektować komunikacji z perspektywy warstwy aplikacji.

Niezawodność.

Odbiorca testuje każdy otrzymany pakiet pod kątem błędów transmisji (poprzez wyliczanie sumy kontrolnej danych). Jeśli wiadomość jest poprawna, odbiorca wysyła potwierdzenie do nadawcy. Jeśli nadawca nie otrzyma potwierdzenia w przeciągu określonego (konfigurowalnego) czasu, to ponownie wysyła zagubiony pakiet. Po kilku nieudanych próbach, TCP zakłada, że odbiorca jest nieosiągalny i informuje warstwę aplikacji, że transmisja zakończyła się niepowodzeniem.

Uszeregowanie pakietów w TCP.

Jedno z pól nagłówka TCP zawiera numer sekwencyjny wiadomości. Numer sekwencyjny jest zwiększany o jeden dla każdej wysłanej wiadomości. Podczas odbierania wiadomości, TCP układa pakiety we właściwej kolejności. Dzięki temu, warstwa aplikacji nie musi w ogóle zajmować się kolejnością przychodzących pakietów sieciowych.

Nagłówek TCP

Składa się z 20 lub więcej bajtów. Dokładna wielkość zależy od tego czy opcjonalne pole opcji jest używane. Maksymalna wielkość tego pola to 40 bajtów, więc maksymalna wielkość całego nagłówka to 60 bajtów.

TCP

- **Opis nagłówka TCP**

+	Bit 0 - 3	4 - 9	10 - 15	16 - 31
0	Port nadawcy			Port odbiorcy
32	Numer sekwencyjny			
64	Numer potwierdzenia			
96	Przesunięcie danych	Zarezerwowane	Flagi	Szerokość okna
128	Suma kontrolna			Wskaźnik priorytetu
160	Opcje (opcjonalnie)			
160/192+	Dane			

Sesja **TCP**.

W celu wymiany danych przy pomocy TCP, dwie aplikacje muszą najpierw zainicjować sesję.

TCP wymaga wymiany trzech wiadomości żeby utworzyć sesję:

1. SYN - pierwsza aplikacja (klient) wysyła pakiet synchronize do serwera. Wiadomość zawiera losowy numer sekwencyjny, który został wybrany przez klienta.
2. SYN-ACK - serwer odpowiada do klienta. Otrzymany numer sekwencyjny jest zwiększany o jeden i załączany do odpowiedzi jako numer potwierdzenia. Dodatkowo, odpowiedź zawiera inny numer sekwencyjny, losowo wybrany przez serwer.
3. ACK - klient potwierdza otrzymanie odpowiedzi od serwera. Wiadomość zawiera oba otrzymane numery zwiększone o jeden.

Kiedy transmisja pomiędzy klientem i serwerem zostanie zakończona, sesja powinna zostać zamknięta. Każda strona komunikacji może zakończyć trwającą sesję. Druga strona powinna odpowiedzieć na to, wysyłając odpowiednie potwierdzenie.

Zastosowanie TCP. TCP jest szeroko wykorzystywane w protokołach i aplikacjach, które wymagają wysokiej niezawodności. Można wymienić wiele protokołów warstwy aplikacji, które używane są głównie razem z TCP.

Jednymi z najpopularniejszych są:

- HTTP, HTTPS
- FTP
- SMTP
- Telnet

UDP

Drugim popularnym protokołem używanym w warstwie transportowej jest **UDP** (*ang. User Datagram Protocol lub Universal Datagram Protocol*). Jest to prostszy protokół, w którym komunikacja odbywa się bez nawiązywania żadnego stałego połączenia pomiędzy aplikacjami. Wszystkie pakiety wysyłane są niezależnie od siebie. Dzięki swojej prostocie **UDP** jest szybsze niż **TCP**. Z drugiej jednak strony, nie zapewnia takiej niezawodności działania jak **TCP**, nie gwarantuje, że wiadomości rzeczywiście dotarły do odbiorcy. **UDP** nie dostarcza pakietów w takiej samej kolejności, w jakiej zostały one wysłane. Ciężar uporządkowania otrzymywanych wiadomości i sprawdzenia czy nie nastąpiły błędy transmisji spoczywa na otrzymującą je aplikacji.

Nagłówek **UPD**

Składa się z 8 bajtów, jest więc znacznie krótszy niż odpowiadający mu nagłówek **TCP**.

UDP

- **Struktura nagłówka UDP**

+/-	Bity 0 - 15	16 - 31
0	Port nadawcy	Port odbiorcy
32	Długość	Suma kontrolna
64	Dane	

UDP jest preferowane jeśli przesyłane pakiety danych są nieistotne lub komunikacja musi odbywać się z wyjątkowo dużą prędkością czyli np. podczas transmisji audio i video, gdzie utrata pewnej liczby pakietów nie jest bardzo uciążliwa dla odbiorcy. Istnieje wiele protokołów warstwy aplikacji, które używają UDP, na przykład:

- **DNS**
- **DHCP** - umożliwia połączonym do sieci komputerom pobieranie adresu IP, maski podsieci, adresu bramy i serwera DNS itp.
- **TFTP**
- **SNMP**
- **RIP** - protokół bram wewnętrznych oparty na zestawie algorytmów wektorowych, służących do obliczania najlepszej trasy do celu
- **VOIP** – telefonia internetowa

DCCP

Datagram Congestion Control Protocol jest protokołem, który umożliwia aplikacjom kontrolowanie przepływu danych w celu zapobiegania przeciążeniom sieci i utrzymywania stabilnych połączeń. **DCCP** nie zapewnia niezawodnej komunikacji z zachowaniem kolejności wysyłanych pakietów.

DCCP jest wykorzystywany przez aplikacje, które operują na szybko zmieniających się danych (dane audio i video, gry online, VoIP). W takich sytuacjach często preferuje się użycie nowej porcji dostępnych danych, zamiast proszenia o retransmitowanie starego uszkodzonego pakietu.

RSVP

Resource Reservation Protocol umożliwia zdalne rezerwowanie zasobów przy użyciu sieci komputerowych. Jest używany głównie przez routery i serwery w celu zapewnienia usług o określonej jakości dla klientów.

RSVP jest w stanie rezerwować pasma transmisji dla komunikacji pomiędzy dwoma komputerami oraz pomiędzy jednym serwerem i wieloma klientami. Wymiana wiadomości w ramach **RSVP** jest inicjowana przez klienta (odbiorcę), który prosi router (serwer) o zarezerwowanie zasobów.

SCTP

Stream Control Transmission Protocol umożliwia przesyłanie wielu strumieni danych spakowanych razem w pojedynczym strumieniu. Podobnie jak **TCP**, **SCTP** zapewnia niezawodną transmisję z zachowaniem kolejności pakietów i zapobieganiem przeciążeniom, dodatkowo rozbudowując jego funkcjonalności o umieszczanie pokrewnych strumieni danych w tych samych wiadomościach.

Ogólnie rzecz biorąc **SCTP**, jest bardzo rozbudowanym protokołem zapewniającym dobrą jakość komunikacji. Niestety, z racji braku wspierania tego protokołu przez najpopularniejsze routery i systemy operacyjne, nie jest on popularny i szerzej używany.

WARSTWA INTERNETU

Głównym zadaniem jest odnalezienie najkrótszej i najszybszej drogi do urządzenia docelowego przez sieć rozległą, podobnie jak robią to samochodowe GPS'y, ale także adresowanie danych z wykorzystaniem adresów logicznych (**adresów IP**). Adres IP jest unikalnym wirtualnym numerem, który umożliwia znajdowanie urządzenia w sieci. Istnieje kilka popularnych protokołów, które działają w warstwie internetowej. Najpopularniejszym i najważniejszym z nich jest IP (Internet Protocol), ale warto wymienić też

Inne protokoły warstwy internetowej:

- **ARP** (Address Resolution Protocol)

- **RARP** (Reverse Address Resolution Protocol)
- **ICMP** (Internet Control Message Protocol)

IP

IP służy do przesyłania pakietów danych przez sieć. Obecnie używane są dwie wersje tego protokołu, **IPv4** i **IPv6**.

IP nie zapewnia żadnego systemu potwierdzania dostarczenia wiadomości, co oznacza, że nie jest niezawodnym protokołem. Obowiązek upewniania się, że wszystkie dane zostały dostarczone spoczywa na protokole TCP operującym w warstwie transportowej. Całe połączenie TCP/IP jest więc niezawodne.

Datagramy IP

Pakiety danych otrzymywane z warstwy transportowej są dzielone na mniejsze datagramy. Każdy datagram zawiera nagłówek IP oraz bajty otrzymane z warstwy transportowej. Maksymalna wielkość datagramu zależy od wersji IP: 216–1 bajtów dla **IPv4** oraz 232–1 dla **IPv6**. Jeśli pakiet otrzymany z warstwy transportowej jest zbyt duży, zostanie podzielony na kilka datagramów o odpowiedniej wielkości.

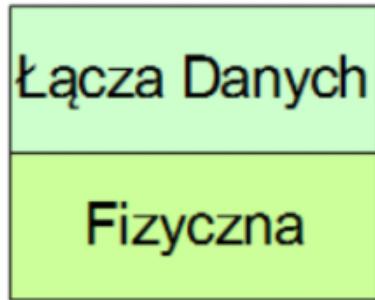
Zwykle dane dzielone są na mniejsze datagramy niż wynikłoby to z ograniczeń protokołu IP. Jest to spowodowane ograniczonymi możliwościami fizycznymi sieci komputerowych. Na przykład, maksymalna wielkość ethernetowych pakietów wynosi 1 500 bajtów, więc zwykle datagramy tworzone w warstwie internetowej operującej na etherencie będą nieco mniejsze niż 1 500 bajtów (aby umożliwić niższym warstwom dodanie swoich nagłówków). Maksymalna wielkość datagramu w sieci jest nazywana MTU (*Maximum Transfer Unit*). IP umożliwia dzielenie datagramów na mniejsze datagramy, jeśli przechodzą one przez sieć z mniejszą wartością MTU. Kiedy mniejsze datagramy docierają znowu do sieci o większej wartości MTU, mogą zostać ponownie zebrane do większego pakietu. W nagłówku IP jest specjalne pole pozwalające na przeprowadzanie takich operacji (nazywające się *Fragment Offset*).

WARSTWA DOSTĘPU DO SIECI

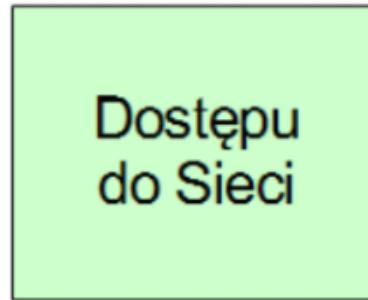
Umożliwia przesłanie datagramów z warstwy internetowej, przez fizyczną sieć do drugiego komputera, gdzie są one przesyłane przez odpowiadającą jej warstwę dostępu do sieci do warstwy internetowej, a następnie poprzez pozostałe warstwy do docelowej aplikacji.

Obecnie, większość komputerów jest podłączona do sieci ethernetowych, które mogą być zarówno przewodowe jak i bezprzewodowe. Wobec tego protokoły TCP/IP wyższych warstw najczęściej są używane razem z zestawem protokołów ethernetowych.

Istnieją **trzy** warstwy ethernetowe. Pierwsze dwie, **Logic Link Control (LLC)** i **Media Access Control (MAC)**, odpowiadają warstwie łącza danych w modelu OSI. Trzecia, najniższa warstwa to warstwa fizyczna, podobnie jak w modelu OSI.



Warstwy Modelu OSI



Warstwa TCP/IP



Warstwy Ethernetu

Warstwa LLC

Jej najważniejszym zadaniem jest przekazanie informacji do docelowej maszyny odnośnie tego jaki protokół powinien być użyty w warstwie transportowej. Umożliwia to poprawne odczytanie przychodzącej wiadomości przez odbiorcę. Warstwa LLC dopisuje informacje o protokole użytym w warstwie internetowej i o protokole, który powinien otrzymać wiadomość. Pozwala to warstwie LLC na docelowym komputerze poprawnie dostarczyć otrzymane datagramy. Zdefiniowana przez standard IEEE 802.2.

Warstwa MAC

jest odpowiedzialna za tworzenie końcowej wiadomości ethernetowej, która będzie wysłana przez sieć komputerową. Podobnie jak inne warstwy, warstwa MAC tworzy swój własny nagłówek i dodaje go do wiadomości. Nagłówek zawiera adresy MAC nadawcy i odbiorcy, czyli fizyczne adresy dwóch komunikujących się maszyn. Jeśli docelowa maszyna znajduje się za routерem, w innej sieci, to pole adresu odbiorcy będzie miało wartość adresu

MAC

routera. Adres MAC odbiorcy będzie zmieniony na inny przez router, kiedy będzie on przetwarzał wiadomość. Warstwa MAC dodaje również 4 kontrolne bajty CRC, które mogą być wykorzystane do naprawienia uszkodzonej wiadomości. Warstwa MAC dla sieci przewodowych jest zdefiniowana przez standard IEEE 802.3. Sieci bezprzewodowe są zdefiniowane przez IEEE 802.11. Warstwa Fizyczna Warstwa fizyczna jest odpowiedzialna za przekształcanie wiadomości w (zależności od typu sieci) impulsy elektryczne lub fale elektromagnetyczne oraz za transmitowanie ich przez sieć fizyczną pomiędzy komunikującymi się maszynami. Jest zdefiniowana przez te same specyfikacje co warstwa MAC, IEEE 802.3 i IEEE 802.11.

37. Cykle życia oprogramowania.

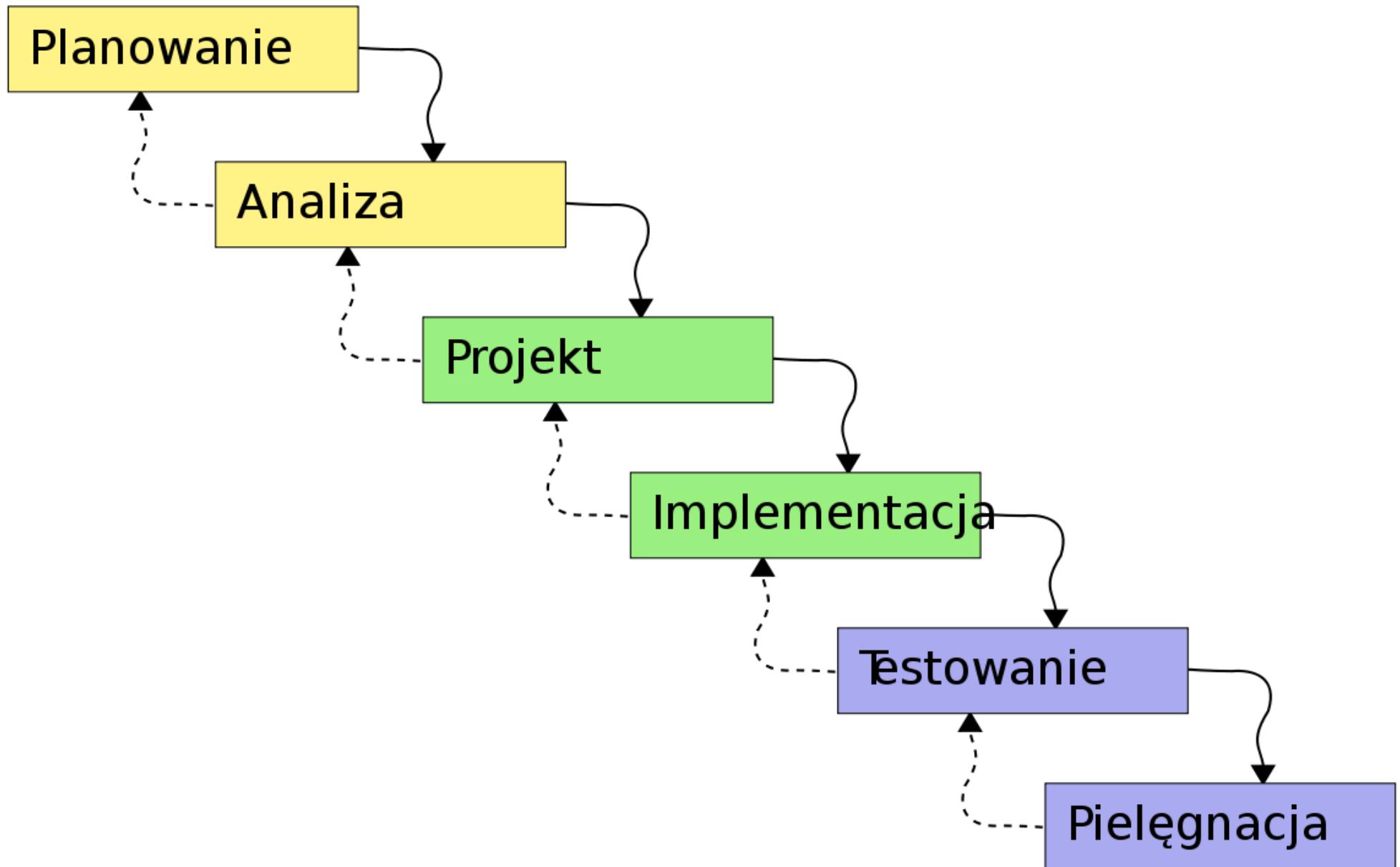
Podstawowe czynności związane z tworzeniem oprogramowania:

- Określanie wymagań i specyfikacji,
- Projektowanie,

- Implementacja,
- Testowanie - walidacja(atestowanie) i weryfikacja
- Konserwacja (pielęgnacja)

Podstawowe modele cyklu życia oprogramowania:

1. Kaskadowy (*waterfall*)



Cechy:

- Nie można przejść do następnej fazy przed zakończeniem poprzedniej
- Błąd popełniony w początkowej fazie ma wpływ na całość

- model ten posiada bardzo nieelastyczny podział na kolejne fazy

- łatwy nadzór, dużo dokumentacji

2. Ewolucyjny (np. Agile, Model Spirralny, Model Przyrostowy)

Cechy:

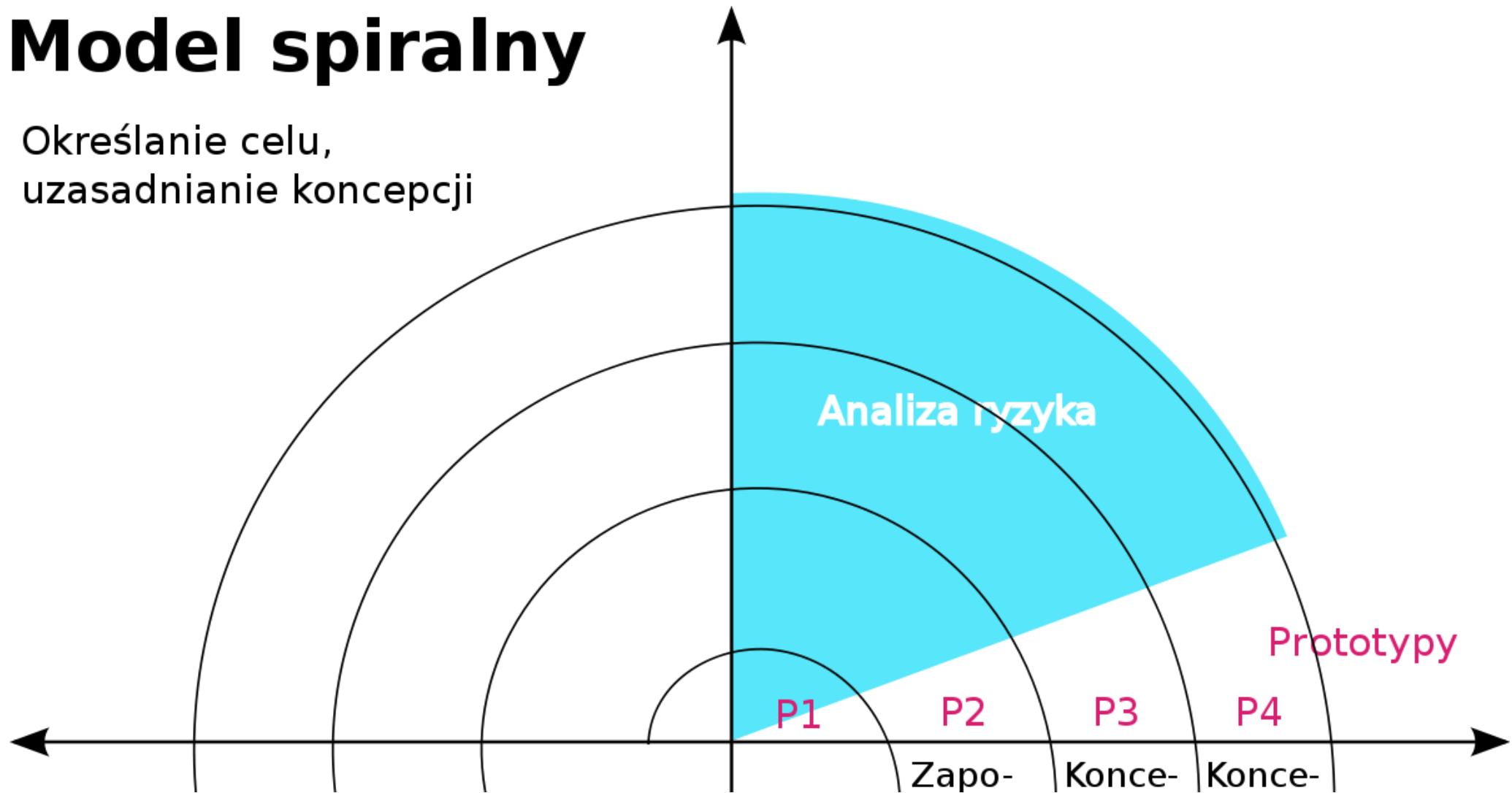
- adaptowanie systemu do zmian w wymaganiach i korygowanie popełnionych błędów
- trudne w nadzorowaniu, wymaga dodatkowych strategii dla uporządkowania procesu tworzenia oprogramowania

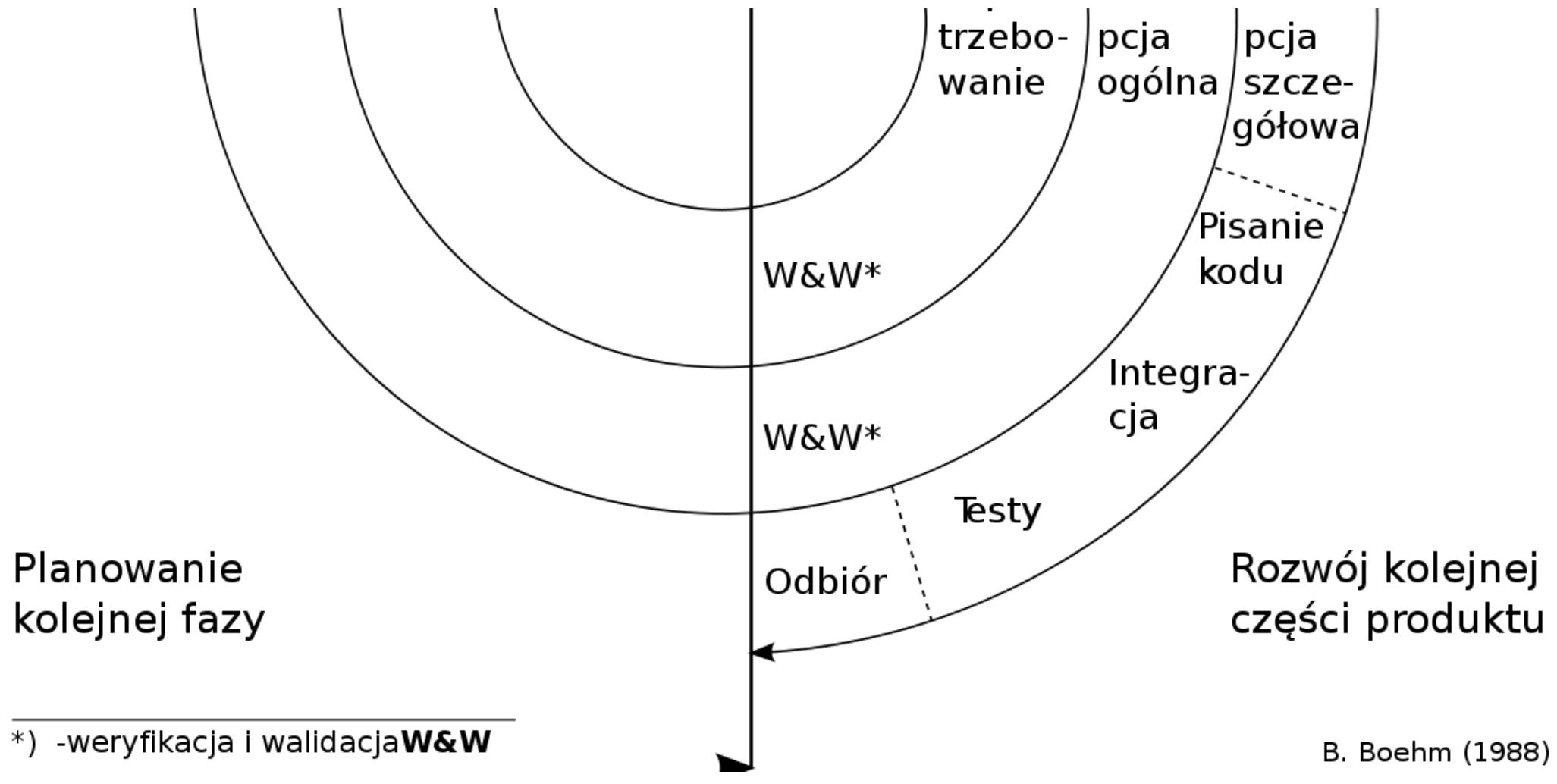
3. Model Komponentowy - Składanie systemu z gotowych komponentów.

4. Model Spirralny

Model spiralny

Określanie celu,
uzasadnianie koncepcji





Cechy:

- Można wykorzystać gotowe komponenty
- Faza oceny w każdym cyklu pozwala uniknąć błędów lub wcześniej je wykryć
- Cały czas istnieje możliwość rozwijania projektu

Cykl życia oprogramowania - Okres czasu rozpoczętyjący się kiedy pojawił się pomysł na oprogramowanie

38. Proces testowania i jego rola w tworzeniu oprogramowania.

Testowanie Oprogramowania - Jest to proces związany z wytwarzaniem oprogramowania. Jest on jednym z procesów kontroli jakości oprogramowania.

Cele testowania:

- Weryfikacja oprogramowania - testowanie zgodności systemu z wymaganiami.
- Walidacja (atestowanie) oprogramowania - ocena systemu lub komponentu podczas lub na końcu procesu jego rozwoju na zgodności z wyspecyfikowanymi wymaganiami.
- Testowanie umożliwia wykrycie błędów we wczesnych stadiach rozwoju oprogramowania co pozwala zmniejszyć koszty usuwania tego błędu.

Podstawowym standardem dla testowania oprogramowania jest IEEE 829-1998 (829 Standard for Software Test Documentation).

Rodzaje testów:

- Testy funkcjonalne. Polegają one na tym, że wcielamy się w rolę użytkownika, traktując oprogramowanie jak, czarną skrzynkę", która wykonuje określone zadania. Nie wnikamy w ogólne techniczne szczegóły działania programu. Testy te często są nazywane **testami czarnej skrzynki**.
- Testy strukturalne. Tym razem tester ma dostęp do kodu źródłowego oprogramowania, może obserwować jak zachowują się różne części aplikacji, jakie moduły i biblioteki są wykorzystywane w trakcie testu. Te testy czasami są nazywane **testami białej skrzynki**

Testy Manualne Testy wykonywane ręcznie przez testera, który przechodzi przez interfejs użytkownika zgodnie z określona sekwencją kroków:

- testy integracyjne,
- testy systemowe dotyczą działania aplikacji jako całość

Testy dopasowane do aktualnego zapotrzebowania/przeznaczenia

- testy funkcjonalne - znane również jako testy czarnej skrzynki,
- testy regresywne - sprawdzają wpływ nowych funkcjonalności na działanie systemu,
- testy akceptacyjne z udziałem klienta,
- testy dokumentacji, których celem jest wykrycie niespójności i niezgodności z dokumentacją,
- testy użyteczności, których celem jest weryfikacja interfejsu użytkownika

Testy automatyczne Testy automatyczne skutecznie przyspieszają proces tworzenia testów systemowych, ich wykonywanie oraz analizę, a tym samym pozwalają na wcześniejsze wykrycie i wyeliminowanie błędów w aplikacjach.

Co podlega testowaniu?

- wydajności systemu,
- interfejsy,
- właściwości operacyjne systemu,
- testy zużycia zasobów,
- zabezpieczenie systemu,
- przenaszalność oprogramowania,
- niezawodność oprogramowania,
- odtwarzalność oprogramowania,
- bezpieczeństwo oprogramowania,

- kompletność i jakość złożonych funkcji systemu

39. UML, jego struktura i przeznaczenie.

Czym jest **UML**

UML - Unified Modeling Language jest to język modelowy używany między innymi w inżynierii oprogramowania jako standardowy sposób wizualizacji projektu systemu.

Historia

Do kreacji **UML** przyczyniła się potrzeba standaryzacji różnych systemów wizualizacji systemów rozwiązywania problemów. Został zaproponowany przez Rational software w połowie lat 90-tych.

W 1997 **UML** został zaimplementowany przez Object Management Group jako główny system organizacji, co było punktem przełomowym w popularyzacji języka.

W 2005 **UML** zostało zaadoptowane do standardu ISO (International Organization for Standardization) i od tamtego czasu przechodzi okresowe rewizje, w 2020 roku wprowadzono specyfikacje wersji 2.5.1

JAK TO DZIAŁA

UML to sposób wizualizacji architektury systemów za pomocą diagramu. UML składa się ze standardowych elementów, które mają na celu przyspieszenie pracy. Diagram posiada sposoby wizualizacji między innymi:

- Wszelkich prac
- Indywidualnych komponentów i jak one wzajemnie na siebie działają
- Jak użytkownik może komunikować się z systemem

UML nie narzuca żadnego systemu pracy, ani żadnego systemu projektowania oprogramowania. Jest jednak narzędziem, która wspomaga projektowanie tych systemów.

W programowaniu, bardzo dobrze sprawdza się jako sposób wizualizacji jak ma działać program opierający się na obiektach.

STRUKTURA DIAGRAMU

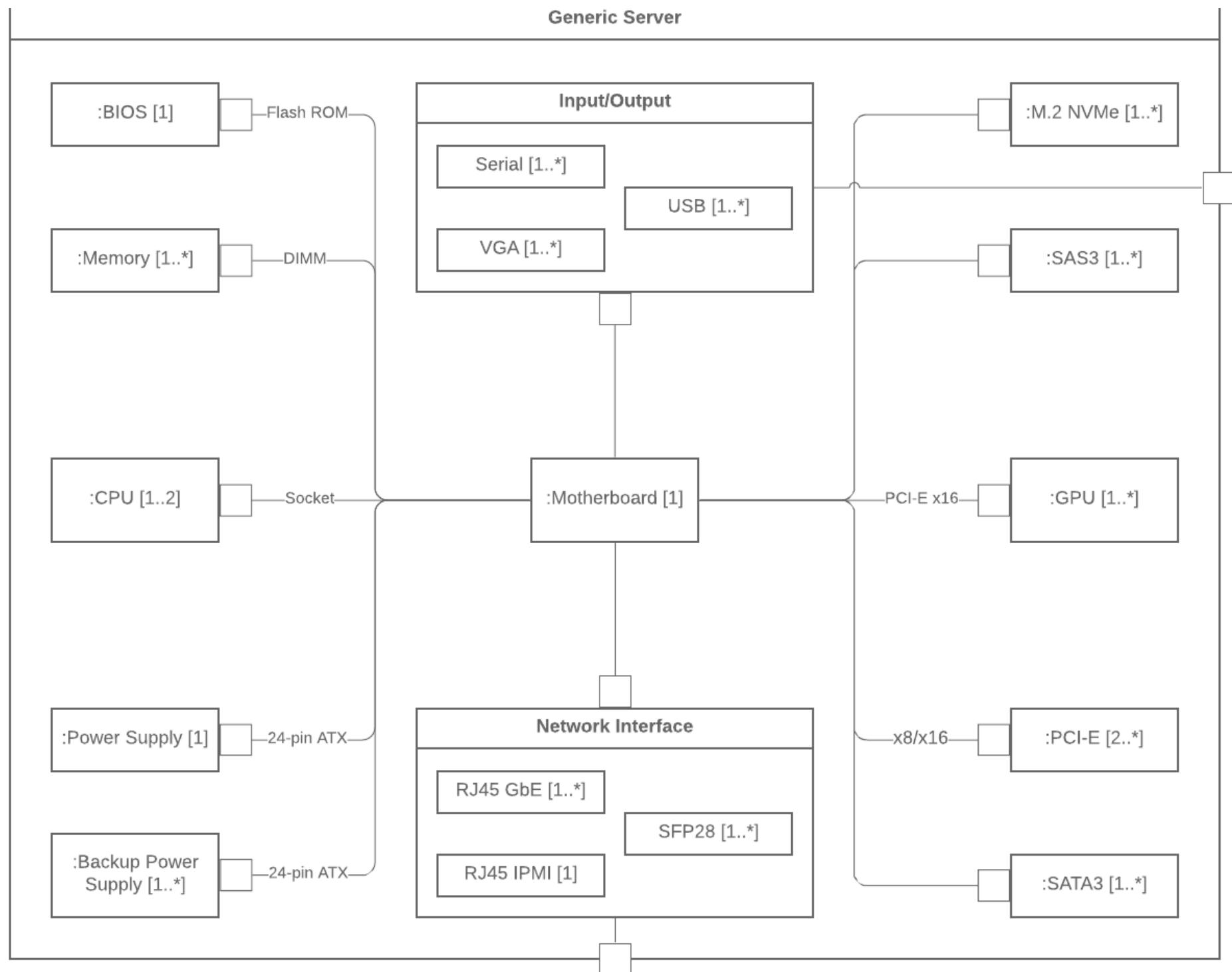
Są różne systemy i **UML** stara się zaproponować wizualizacje każdego z nich za pomocą różnych elementów, wyspecjalizowanych lub nie. Idea natomiast jest taka sama.

Diagram składa się z zawsze z wizualizacji danego obiektu, oraz interakcji między nimi. Z takich standardowych przykładów używania UML możemy wymienić reprezentację systemu za pomocą bramek logicznych.

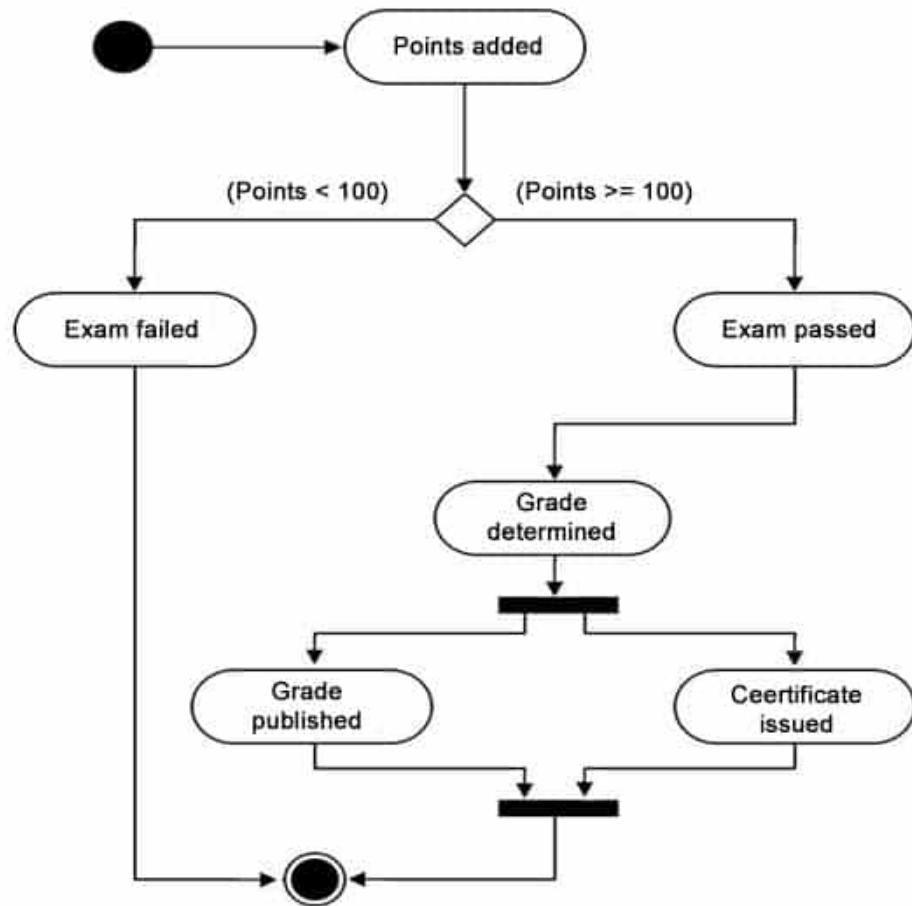
KATEGORIE

UML 2 posiada wiele typów diagramów, które można podzielić na 2 typy :

1. **Strukturalne** pokazują układ obiektów, lub całych systemów i interakcje między innymi.



2. **Behawioralne** pokazują dynamiczne systemy, takie jak np algorytmy.



40. Podstawowe funkcje w zespole projektowym i ich role.

Czym jest zespół projektowy?

Zespół projektowy jest to jednostka organizacyjna, powołana na zasadzie specjalizacji przedmiotowej, realizująca projekt pod bezpośrednim nadzorem kierownika projektu.

Czym jest Agile?

Grupa metod wytwarzania oprogramowania opartego na programowaniu iteracyjno-przyrostowym. Najważniejszym założeniem metodyk zwinnych jest obserwacja, że wymagania odbiorcy (klienta) często ewoluują podczas trwania projektu. Pojęcie zwinnego programowania zostało zaproponowane w 2001 w Manifeście Programowania Zwinnego.

Manifest zwinnego programowania

„(...) Bardziej cenimy:

Ludzi i interakcje od procesów i narzędzi Działające oprogramowanie od szczegółowej dokumentacji Współpracę z klientem od negocjacji umów Reagowanie na zmiany od realizacji założonego planu.

Oznacza to, że elementy wypisane po prawej są wartościowe, ale większą wartość mają dla nas te, które wypisano po lewej."

SCRUM

Są to iteracyjne i przyrostowe ramy postępowania zgodne ze Scrum Guide. Zgodnie z definicją ze Scrum Guide'a w obręb Scruma wchodzą: Zespoły Scrumowe oraz związane z nimi role, wydarzenia, artefakty i reguły.

- ROLE: DEWEOPERZY + PRODUCT OWNER = ZESPÓŁ SCRUMOWY + SCRUM MASTER
- WYDARZENIA: PLANOWANIE SPRINTU + CODZIENNY SCRUM + PRZEGLĄD SPRINTU + RETROSPEKTYWA = SPRINT
- ARTEFAKTY: BACKLOG PRODUKTU + BACKLOG SPRINTU + PRZYROST
- REGUŁY: PRZEJRZYSTOŚĆ + INSPEKCJA + ADAPTACJA

SCRUM - ROLE

ROLE: DEWEOPERZY + PRODUCT OWNER + SCRUM MASTER = ZESPÓŁ SCRUMOWY Deweloperzy, czyli zespół składający się z 3-9 osób z np. testera, analityka, webdewelopera, programisty dowolnego języka. Odpowiadają za sposób wykonania zadań. W zespole wszyscy powinni być równi. Product Owner odpowiada za wybór zadań do wykonania. Product Owner to osoba reprezentująca klienta, ciało jednoosobowe, jedyne, które może zlecać zadania zespołowi, dlatego bardzo ważne jest wsparcie jego roli w organizacji. Scrum Master czuwa nad tym, aby przebieg prac był zgodny z zasadami Scruma i ustalonymi przez zespół. Osoba ta odpowiedzialna jest za usuwanie wszelkich przeszkód uniemożliwiających zespołowi wykonanie zadania.

SCRUM - WYDARZENIA

WYDARZENIA:

PLANOWANIE SPRINTU
CODZIENNY SCRUM +
PRZEGLĄD SPRINTU +
RETROSPEKTYWA =
SPRINT

- **Planowanie Sprintu**, służy ustaleniu na samym początku Sprintu, nad czym Zespół będzie pracował, w jaki sposób i dla jakiego celu. Pieczę nad postępem pracy Zespół zapewnia sobie, organizując
- **Codzienny Scrum**, czyli miniplanowania, które służą codziennej weryfikacji stanu prac i ewentualnym korektom zaplanowanych zadań.
- Na koniec Sprintu Zespół przedstawia swoje dokonania Product Ownerowi i interesariuszom w czasie **Przeglądu Sprintu**
- Zaraz po tym weryfikuje swój sposób pracy i wprowadza ulepszenia w czasie **Retrospektwy**.

SCRUM - ARTEFAKTY

ARTEFAKTY: BACKLOG PRODUKTU + BACKLOG SPRINTU + PRZYROST

Backlog Produktu to uporządkowana lista wszystkich rodzajów zadań potrzebnych do rozwoju, utrzymania i naprawy produktu. Lista ta jest otwarta dla wszystkich w organizacji, natomiast Product Owner ma ostateczne słowo co do treści, wyglądu i zawartości Backlogu. To jest jego narzędzie pracy nad produktem. Backlog Sprintu to analogiczne narzędzie, ale dla Zespołu Deweloperskiego, który dzięki temu artefaktowi w pełni panuje nad pracami zaplanowanymi na Sprint. Przyrost to ukończona przez Zespół zgodnie z Definicją Ukończenia praca na koniec każdego i wszystkich razem Sprintów.

SCRUM - REGUŁY

REGUŁY: PRZEJRZYSTOŚĆ + INSPEKCJA + ADAPTACJA

Przejrzystość zapewnia Zespołowi Scrumowemu i wszystkim w organizacji dostęp do całości prac i takie samo rozumienie każdego elementu Scruma. Dzięki temu nie ma niejasności i nieporozumień. Inspekcja pozwala na bieżące monitorowanie i weryfikowanie przedmiotu pracy i sposobu pracy Zespołu. Adaptacja powinna być wynikiem Inspekcji i prowadzić do niezbędnych zmian naprowadzających Zespół na właściwy tor pracy.

41. Pojęcie Maszyny Turinga - idea pracy automatu, hipoteza Churcha-Turinga

Definicja

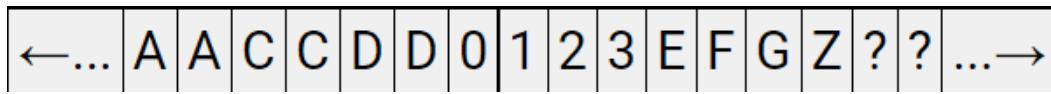
Maszyna Turinga - stworzona przez *Alana Turinga* prosta maszyna logiczna (licząca) służąca do wykonywania algorytmów. Wszystkie współczesne komputery dają się do niej sprowadzić.
Problem jest rozwiążalny na komputerze, jeśli da się zdefiniować rozwiązywaną go maszynę Turinga.

Maszyna Turinga zbudowana jest z trzech głównych elementów:

- Nieskończnej taśmy zawierającej komórki z przetwarzanymi symbolami
- Ruchomej głowicy odczytującej i zapisującej
- Bloku sterowania głowicą.

Taśma

Nieskończona taśma jest odpowiednikiem współczesnej pamięci komputera. Taśma dzieli się na komórki, w których umieszczone zostały 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 oraz tzw. *znak pusty* - czyli zawartość komórki, która nie zawiera żadnej danej do przetworzenia. Wbrew pozorom taki prymitywny zbiór trzech symboli jest równoważny logicznie dowolnemu innemu zbiorowi

Główica

Aby przetwarzać dane, maszyna Turinga musi je odczytywać i zapisywać na taśmie. Do tego celu przeznaczona jest właśnie głowica zapisująco-odczytująca, która odpowiada funkcjonalnie urządzeniom wejścia/wyjścia współczesnych komputerów 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 - na tej zasadzie odbywa się przetwarzanie danych - z jednych symboli otrzymujemy inne. Oprócz odczytywania i zapisywania symboli w komórkach głowica wykonuje ruchy w prawo i w lewo do sąsiednich komórek na taśmie. W ten sposób może się ona przemieścić do dowolnie wybranej komórki taśmy.

Przed rozpoczęciem pracy maszyny Turinga głowica jest zawsze ustawiana nad komórką taśmy zawierającą pierwszy symbol do przetworzenia. W klatce taśmy po lewo jest zapisany specjalny znak, tzw. *lewy ogranicznik*. Jeżeli głowica znajduje się nad lewym ogranicznikiem, to nie może go zamazać ani przesunąć się na lewo od niego. Po zakończeniu danych wejściowych taśma wypełniona jest w nieskończoność specjalnymi pustymi symbolami, tzw. *blank'ami*.

Układ Starowania

Przetwarzaniem informacji zarządza układ sterowania głowicą. Jego współczesnym odpowiednikiem jest procesor komputera. Układ ten odczytuje za pomocą głowicy symbole z komórek taśmy oraz przesyła do głowicy symbole do zapisu w komórkach. Dodatkowo nakazuje on głowicy przemieścić się do sąsiedniej komórki w lewo lub w prawo.

Podstawą działania maszyny Turinga są *stany układu sterowania*. Stan układu sterowania określa jednoznacznie jaką operację wykona, jak zareaguje maszyna Turinga, gdy odczyta z taśmy określony symbol.

Zatem operacje wykonywane przez układ sterowania zależą od dwóch czynników:

- Symbolu odczytanego z komórki na taśmie
- Bieżącego stanu układu sterującego

Stany będziemy określać kolejnymi nazwami: q₀, q₁, q₂, ..., q_n, gdzie q₀ jest stanem początkowym, w którym znajduje się maszyna Turinga przed rozpoczęciem przetwarzania symboli na taśmie.

Instrukcją dla maszyny Turinga jest następująca piątka symboli:

Instrukcja maszyny Turinga	Znaczenia symboli	
$(S_0, q_i, S_z, q_j, L/P)$	S_0	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
	L/R	ruch głowicy o jedną komórkę w lewo (L) lub w prawo (R)

S_0 i q_i są tzw. **częścią identyfikacyjną instrukcji**. Maszyna Turinga wykonuje tyle różnych instrukcji, ile zdefiniujemy części identyfikacyjnych - w programie nie może być dwóch różnych instrukcji o identycznej części identyfikacyjnej.

S_z , q_j i L/P są tzw. **częścią operacyjną**, która określa jakie działanie podejmuje dana instrukcja. Części operacyjne różnych instrukcji mogą być takie same - oznacza to jedynie, iż instrukcje te wykonują dokładnie to samo działanie.

Przykład instrukcji

$$(A, q_0, B, q_0, \mathbf{R})$$

Jeżeli odczytanym przez głowicę symbolem z taśmy będzie symbol A , a układ sterowania znajduje się w stanie q_0 , to głowica zamieni ten symbol na B , stan wewnętrzny nie zmieni się (pozostanie dalej q_0), a głowica przesunie się do sąsiedniej komórki po prawej stronie.

Hipoteza Churcha-Turinga

Formalna Definicja

Każdy problem, który może być intuicyjnie uznany za obliczalny, jest rozwiązywalny przez maszynę Turinga.

Sformułowanie "intuicyjnie uznany za obliczalny" uniemożliwia przeprowadzenie matematycznego dowodu tej hipotezy.

Bardziej praktyczna definicja

Każdy problem, dla którego przy nieograniczonej pamięci oraz zasobach istnieje efektywny algorytm jego rozwiązywania, da się rozwiązać na maszynie Turinga

Trzecie równoważne sformułowanie

Każdy nieinteraktywny program może być zredukowany do rozwiązującej go maszyny Turinga, a ta może być wyrażona w każdym [zupełnym w sensie Turinga](#) języku programowania.

Dlatego równoważne sformułowanie tej hipotezy mówi, że każdy istniejący algorytm można wyrazić w każdym zupełnym języku programowania.

Zapis Formalny MT

$$\text{MT} = \langle Q, \Sigma, \Gamma, s, b, F, \delta \rangle$$

Q – skończony zbiór stanów (q_0 – stan początkowy),

Σ – skończony zbiór symboli wejściowych

$\Gamma \supseteq \Sigma$ – skończony zbiór dopuszczalnych symboli,

$s \in Q$ – stan początkowy

$b \in \Gamma \setminus \Sigma$ – symbol pusty

$F \subseteq Q$ – zbiór stanów końcowych

$\delta : Q \times \Gamma \longrightarrow Q \times (\Gamma \times \{L, R, S\})$ – funkcja częściowa, zwana funkcją przejść, gdzie k jest liczbą taśm, L to przesunięcie w lewo, R przesunięcie w prawo, a S to brak przesunięcia.

42. Usługa translacji adresów w sieci TCP/IP.

Definicja Formalna

Translacja Adresów Sieciowych (Network Address Translation, NAT) – technika przesyłania ruchu sieciowego poprzez router, która wiąże się ze zmianą źródłowych lub docelowych adresów IP, zwykle również numerów portów TCP/UDP pakietów IP podczas ich przepływu. Zmieniane są także sumy kontrolne (zarówno w pakiecie IP, jak i w segmencie TCP/UDP), aby potwierdzić wprowadzone zmiany.

Alternatywna Definicja

NAT jest procesem modyfikującym informację o adresie IP w nagłówku pakietu IP, w momencie przesyłania ruchu przez urządzenie sieciowe. W większości konfiguracji, NAT podmienia prywatne adresy wewnętrz sieci na adresy IP publiczne, udostępniane przez dostawcę usługi dostępu do internetu. Taki zabieg pozwala komputerom w sieci domowej czy firmowej współdzielić połączenie internetowe. Dodatkowo, uzyskuje się zwiększy poziom bezpieczeństwa sieci, ponieważ dostęp do sieci wewnętrznej z zewnątrz jest mocno ograniczony.

Dwa podstawowe typy NAT:

- **SNAT** (Source Network Address Translation) to technika polegająca na zmianie adresu źródłowego pakietu IP na jakiś inny. Stosowana często w przypadku podłączenia sieci dysponującej adresami prywatnymi do sieci Internet. Wtedy router, przez który podłączono sieć, podmienia adres źródłowy prywatny na adres publiczny (najczęściej swój własny).
- **DNAT** (Destination Network Address Translation) - to technika polegająca na zmianie adresu docelowego pakietu IP na jakiś inny. Stosowana często w przypadku, gdy serwer, który ma być dostępny z Internetu ma tylko adres prywatny. W tym przypadku router dokonuje translacji adresu docelowego pakietów IP z Internetu na adres tego serwera.

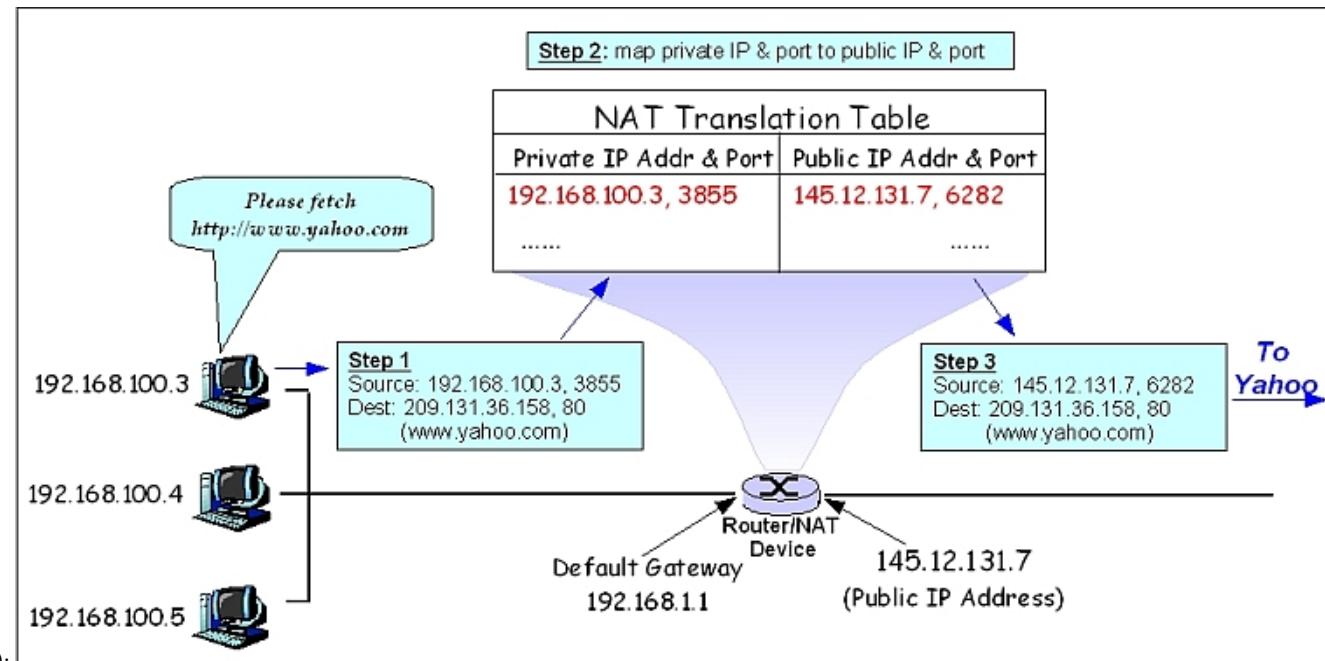
Wyróżniamy trzy rodzaje SNAT:

- **Statyczny NAT:** Udostępnia odzorowanie 1-1 między adresami zewnętrznymi a adresami lokalnymi (czyli każdy komputer lokalny ma swoje IP, a serwer tylko pośredniczy w przekazywaniu pakietów). Takie stałe mapowanie jest najbardziej odpowiednie dla hostów, które muszą być dostępne poza siecią. Jest to najbardziej odpowiednie do zapewnienia dostępu do serwerów takich jak serwery poczty elektronicznej i serwery internetowe.
- **Dynamiczny NAT:** Serwer dysponuje pulą adresów IP, które przyporządkowuje lokalnym jednostkom dynamiczne w odpowiedzi na ich żądania skierowane do sieci zewnętrznej
- **PAT: Port address translation** - jest jednym z najczęściej używanych systemów NAT. Wiele połączeń z różnych wewnętrznych hostów jest multipleksowane w celu utworzenia jednego publicznego adresu IP, który wykorzystuje różne numery portów źródłowych. Maksymalnie 65 536 połączeń wewnętrznych można przetłumaczyć na jeden publiczny adres IP. Sprawia to, że jest on bardzo skuteczny w sytuacjach, gdy dostawca usług przydzielił tylko jeden publiczny adres IP.

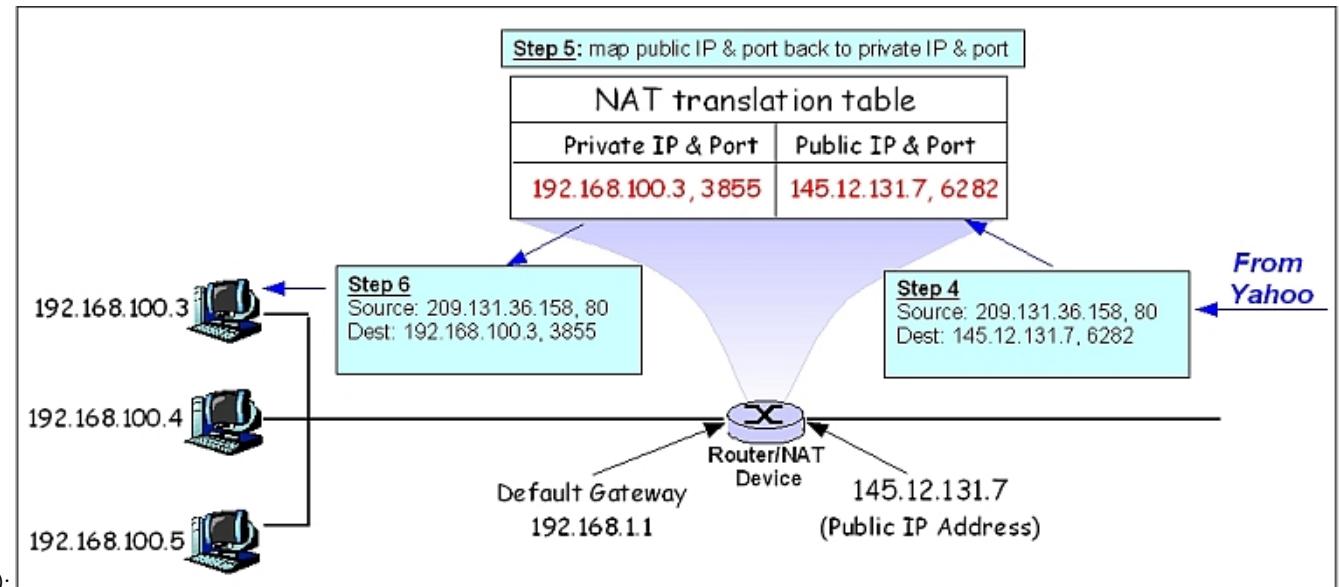
Kiedy komputer z sieci lokalnej wysyła zapytanie do sieci, urządzenie NAT zmienia adres nadawcy pakietu (i czasem numer portu) na publiczny adres IP.

W momencie, gdy wraca do nas odpowiedź na ten pakiet urządzenie NAT przypisuje pakietowi odpowiedni adres lokalnego węzła. Odwzorowania między pakietami a adresami zapamietywane są w tablicy translacji NAT.

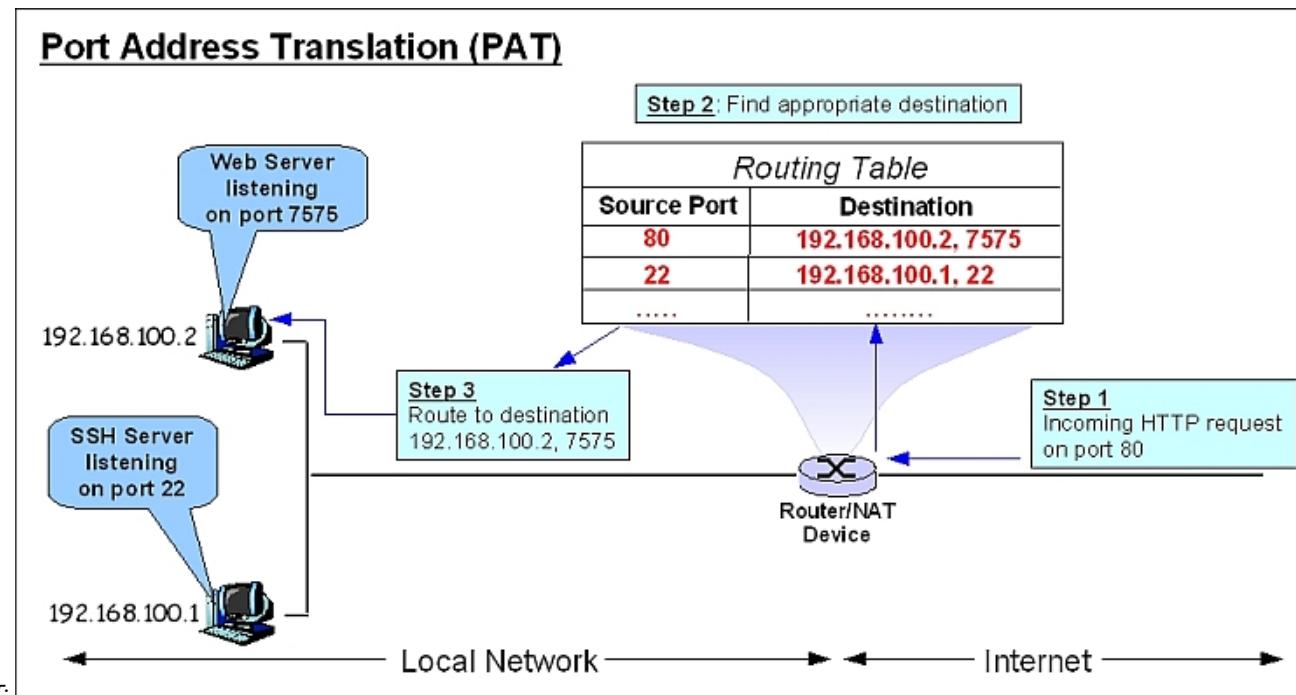
Przykłady działania SNAT'U



Wysyłanie request'u przez (Statyczny/Dynamiczny SNAT):



Otrzymanie odpowiedzi przy (Statycznym/Dynamicznym SNAT):



Otrzymanie odpowiedzi z włączanym PAT:

43. Mechanizm trasowania (ang. routing) pakietów w Internecie.

Ogólne pojęcie

Trasowanie (Routing) - to mechanizm wyznaczania trasy i przesyłania pakietów danych w sieci, od stacji nadawczej do stacji odbiorczej.

Intersieć - to minimum dwie sieci fizyczne połączone ze sobą za pomocą routera.

Trasowaniem zajmuje się urządzenie zwane routerem: może to być zwykły komputer jak i urządzenie specjalnie dedykowane tylko do tego zadania, tzw. *router sprzętowy*.

Trasowanie umożliwia danym z jednej sieci lokalnej dotrzeć do innej sieci lokalnej, która może znajdować się w dowolnym miejscu na świecie. Trasa może prowadzić przez wiele sieci pośrednich, tak więc routing jest jakby *spoiwem łączącym Internet w całość*. Bez routowania cały ruch danych byłby ograniczony do jednej fizycznej sieci.

UWAGA

Trasowanie realizowane jest w **warstwie trzeciej (sieciowej)** modelu OSI. Wyznaczane trasy pakietów danych muszą być jak **najbardziej optymalne** – czyli możliwie najszybsze, ale umożliwiające dostarczenie wszystkich pakietów.

Troche bardziej szczegółowo o pakietach

Pakiet to jednostka informacji, której źródłem i przeznaczeniem jest warstwa Sieciowa (warstwa 3) modelu OSI. Pakiet składa się z trzech elementów:

- **Nagłówka** warstwy Sieciowej,
- **Danych** warstwy wyższej,
- **Końcówki** warstwy Sieciowej.

Nagłówek i końcówka zawierają informację sterującą przeznaczoną dla warstwy 3 w stacji odbiorczej. Można powiedzieć, że dane z wyższej warstwy są otoczone (kapsułkowane) przez nagłówek i końcówkę warstwy 3.

Datagram jest jednostką informacji, której źródłem i przeznaczeniem jest warstwa Sieciowa (warstwa 3) modelu OSI, używającą bezpołączeniowej obsługi sieci. Pakiet (połączeniowa obsługa sieci) = datagram (beopołączeniowa)

Etapy trasowania:

1. Host generuje pakiety i decyduje, czy dostarczyć je bezpośrednio do adresata, czy przesłać do routera.
2. Obowiązkiem routera przy przekazywaniu pakietu dalej do celu jest obniżenie o jeden wartości TTL (ang. Time To Live, czas życia). Datagram IP, który trafia do routera z wartością 1 (a zostanie ona zmniejszona na tym routerze do 0) w polu TTL zostanie utracony, a do źródła router odsyła data gram ICMP z kodem TTL Exceeded.
3. Router decyduje, czy przesłać pakiety bezpośrednio do adresata, czy do routera pośredniczącego (i ew. do którego routera, gdy jest ich kilka).

Tablica Routingu

Router przechowuje tzw. **tablicę routingu**, dzięki której wie, jak kierować ruchem. Najważniejsze informacje zawarte w tablicy to adresy sąsiednich routerów i adresy sieci docelowych.

Aby dotrzeć do sieci	Wyślij do urządzenia o adresie
10.1.1.0	10.1.2.2
10.1.2.0	10.1.2.2

Aby dotrzeć do sieci Wyślij do urządzenia o adresie

10.1.3.0

Bezpośrednio połączony

Oprócz tego w tablicy mogą się też znaleźć informacje o **całościowym koszcie (metryce)** wysłania daną trasą pakietu (jest to pewna liczba przypisana trasie przez protokoły routingu), **nazwy** **czy adresy interfejsów sieciowych**, przez które dany pakiet jest kierowany do sieci, **flagi** opisujące właściwości danej ścieżki (H - ścieżka do konkretnego komputera, a nie np. do kolejnego routera, U – ścieżka jest drożna i działa bez problemów), **licznik** określający czas, jaki upłynął od ostatniego uaktualnienia informacji o trasie.

Pakiet danych przechodzi pomiędzy kolejnymi sieciami. Takie kolejne przejście nazywane jest **przeskokiem** lub **hop-em**. Tablica routingu zawarta w routerze lub w komputerze sieciowym zawiera właśnie przyporządkowania adresów **dotyczące jednego hopu!**

Routing Table:

Destination	Gateway	Flags	Ref	Use	Interface
127.0.0.1	127.0.0.1	UH	0	10565	lo0
153.19.51.64	153.19.51.66	U	3	12105	hme0
224.0.0.0	153.19.51.66	U	3	0	hme0
default	153.19.51.126	UG	0	264196	

Rysunek 1. Przykładowa tablica routingu

Routing Statyczny i Dynamiczny

Pod względem sposobu wypełniania danymi tych tablic, dzielimy routing na statyczny i dynamiczny.

Statyczny - administrator ręcznie wpisuje wszystkie adresy do tablicy routingu. Najprostszą formą budowania informacji o topologii sieci są ręcznie podane przez administratora trasy definiujące routing statyczny. Przy tworzeniu takiej trasy wymagane jest jedynie podanie adresu sieci docelowej, interfejsu, przez który pakiet ma zostać wysłany oraz adresu IP następnego routera na trasie.

Zalety

- Router przesyła pakiety przez z góry ustalone interfejsy bez konieczności każdorazowego obliczania tras, co zmniejsza zajętość cykli procesora i pamięci.
- Informacja statyczna nie jest narażona na deformację spowodowaną zanikiem działania dynamicznego routingu na routerach sąsiednich.
- Dodatkowo zmniejsza się zajętość pasma transmisji, gdyż nie są rozsyłane pakiety rozgłoszeniowe protokołów routingu dynamicznego
- Dla małych sieci jest to doskonałe rozwiązanie, ponieważ nie musimy posiadać zaawansowanych technologicznie i rozbudowanych sprzętowo routerów.
- Routing statyczny zapewnia również konfigurację tras domyślnych, nazywanych *bramkami ostatniej szansy (gateway of the last resort)*. Jeżeli router uzna, iż żadna pozycja w tablicy routingu nie odpowiada poszukiwanemu adresowi sieci docelowej, korzysta ze statycznego wpisu, który spowoduje odesłanie pakietu w inne miejsce sieci.)

Wady

- Routing statyczny wymaga jednak od administratora sporego nakładu pracy w początkowej fazie konfiguracji sieci.
- Nie jest również w stanie reagować na awarie poszczególnych tras.

Dynamiczny - routery samodzielnie zbierają informacje i aktualizują zapisy w tablicy.

Ponieważ statyczne systemy trasowania nie mogą reagować na zmiany w sieci, to generalnie nie są one przydatne do stosowania w sieciach dużych, gdzie zmiany następują praktycznie ciągle. Dlatego większość obecnie stosowanych algorytmów trasowania to algorytmy dynamiczne, dostosowujące się do zmiennych warunków występujących w sieci, na drodze analizy aktualizujących komunikatów trasowania. W wypadku, gdy aktualizujący komunikat trasowania wskazuje, że w sieci wystąpiły zmiany, oprogramowanie trasujące ponownie oblicza trasy i wysyła do routerów nowe komunikaty aktualizujące. W ślad za tym komunikaty, przenikając przez sieć, stymulują routery do uruchomienia algorytmów trasowania i zmieniają ich tablice trasowania.

Protokoły trasowania dynamicznego są wykorzystywane przez routery do pełnienia trzech podstawowych funkcji:

- Wyszukiwania nowych tras
- Przekazywanie do innych routerów informacji o znalezionych trasach
- Przesyłania pakietów za pomocą owych routerów.

Kategorie protokołów trasowania

Podział protokołów:

- Podział ze względu na charakter wymienianych informacji:
 - Protokoły wektora odległości (lub dystans - wektor)
 - Protokoły stanu łączna
 - Hybrydowe
- Podział ze względu na obszary zastosowań:
 - Protokoły wewnętrzne
 - Protokoły zewnętrzne

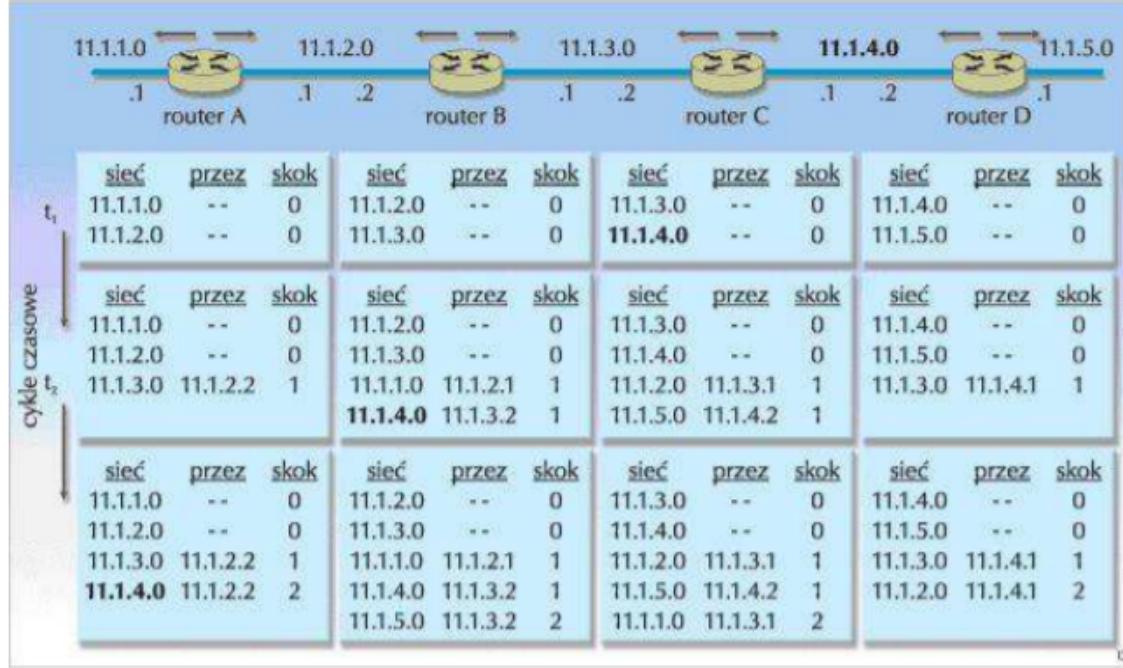
Protokoły wektora odległości:

Trasowanie może być oparte na algorytmach wektora odległości (nazywanych również [algorytmami Bellmana-Forda](#)). Nazwa pochodzi stąd, iż poszczególne routery prezentowane są jako wektory zawierające dwie informacje: dystans oraz wektor wyznaczający kierunek. **Dystans** opisuje całkowity koszt/metrykę danej trasy i wyrażany jest za pomocą pewnej liczby, natomiast **Kierunek** definiowany jest poprzez adres następnego skoku.

Etapy działania protokołu:

1. Przy starcie router tworzy tablicę routingu zawierającą informacje tylko o jego bezpośrednich sąsiadach i kosztach/metrykach dotarcia do nich.
2. Wysyła tą tablicę tylko do swoich sąsiadów, którzy uzupełniają swoje tablice routingu o informacje, które pozyskali z tej właśnie przysłanej.

Router nie widzi poza swojego sąsiada i informacje o innych sieciach, nieprzyłączonych do niego bezpośrednio, uzyskuje tylko dzięki nim. Nazywa się to **routingiem przez plotkowanie**.



Rysunek 2. Przykład działania protokołu wektora odległości

Zalety:

- Protokoły wektora odległości są łatwe w konfiguracji i bardzo dobrze nadają się do zastosowania w małych sieciach.

Wady:

- Niestety, jednym z ich podstawowych problemów jest tzw. **zbieżność**, czyli powolne reagowanie na zmiany zachodzące w topologii sieci, na przykład wyłączenie lub włączenie pewnych segmentów - zerwanie łącza zostaje odzwierciedlone w tabelach routingu poszczególnych routerów dopiero po pewnym czasie. Czas, po którym wszystkie routery mają spójne i aktualnione tabele routingu nazywany jest **czasem zbieżności**.
- Kolejną wadą protokołów wektora odległości jest generowanie dodatkowego ruchu w sieci poprzez cykliczne rozgłaszczenie pełnych tabel routingu, nawet wówczas, gdy w topologii sieci nie zachodzą żadne zmiany.
- Protokoły tej grupy nie są też odporne na powstawanie pętli między routerami (zarówno między bezpośrednimi sąsiadami, jak i pętli rozległych), co skutkuje wzajemnym odsyłaniem sobie pakietów z informacją o tej samej sieci.

Przykładowe protokoły: **RIP, EBGP**.

Trasowanie na podstawie stanu łączna

Algorytmy trasowania na podstawie stanu łącza, ogólnie określane jako protokoły "najpierw najkrótszą ścieżką" (ang. [SPF shortest path first](#)), utrzymują złożoną bazę danych opisującą topologię sieci. W odróżnieniu od protokołów wektora odległości, protokoły stanu łączego zbierają i przechowują pełną informację na temat routerów sieci, a także o sposobie ich połączenia.

W protokołach stanu łączego każdy router przechowuje kompletną bazę danych o topologii sieci z informacjami o koszcie pojedynczych ścieżek w obrębie sieci oraz o stanie połączeń. Informacje te kompletowane są poprzez rozsyłanie tzw. pakietów LSA (*Link-State Advertisement*) o stanie łączego.

Etapy działania protokołu:

1. Każdy router wysyła informację o bezpośrednio do niego podłączonych sieciach oraz o ich stanie (włączone lub wyłączone).
2. Dane te są następnie rozsyłane od routera do routera, każdy router pośredni zapisuje u siebie kopię pakietów LSA, ale nigdy ich nie zmienia.
3. Po pewnym czasie (czasie zbieżności) każdy router ma identyczną bazę danych o topologii (czyli mapę sieci) i na jej podstawie tworzy drzewo najkrótszych ścieżek SPF (shortest path first) do poszczególnych sieci.
4. Router zawsze umieszcza siebie w centrum (korzeniu) tego drzewa, a ścieżka wybierana jest na podstawie kosztu dotarcia do docelowej sieci - najkrótsza trasa nie musi pokrywać się z trasą o najmniejszej liczbie skoków. Do wyznaczenia drzewa najkrótszych ścieżek stosowany jest [algorytm E.W. Dijkstry](#).

Zalety:

- Reagowanie na zmiany w topologii sieci. Po zmianie stanu łączego router generuje nowy pakiet LSA, który rozsyłany jest od routera do routera, a każdy router otrzymujący ten pakiet musi przeliczyć od nowa drzewo najkrótszych ścieżek i na jego podstawie zaktualizować tabelę routingu.
- Protokoły stanu łączego nazywane są też protokołami "cichymi", ponieważ w przeciwieństwie do protokołów wektora odległości nie rozsyłają cyklicznych ogłoszeń, a dodatkowy ruch generują tylko przy zmianie stanu łączego. Ze względu na sposób działania i swoje cechy protokoły stanu łączego przeznaczone są do obsługi znacznie większych sieci niż protokoły wektora odległości.

Wady

- Do wad protokołów stanu łączego zaliczyć można zwiększone zapotrzebowanie na pasmo transmisji w początkowej fazie ich działania (zanim "ucichną"), gdy routery rozsyłają między sobą pakiety LSA. Wspomniane obniżenie wydajności ma charakter przejściowy, ale jest niestety mocno odczuwalne.
- Dodatkowo ze względu na złożoność obliczeń drzewa SPF, protokoły stanu łączego mają zwiększone wymagania dotyczące procesora i pamięci RAM routera (zwłaszcza przy większych sieciach). Z tego powodu routery skonfigurowane do obsługi trasowania na postawie stanu łączego są stosunkowo drogie. Typowym przedstawicielem tej grupy protokołów jest *OSPF (Open Shortest Path First)*

Przykładowe protokoły: OSPF, IS-IS, IDRP

Hybrydowe Trasowanie

Ostatnią formą trasowania dynamicznego jest praca *hybrydowa*. Choć istnieją "otwarte" zrównoważone protokoły hybrydowe, ta forma trasowania jest niemal całkowicie związana z zastrzeżonym produktem jednej firmy Cisco Systems, Inc. Protokół o nazwie **EIGRP** (ang. [Enhanced Interior Gateway Routing Protocol](#)) został zaprojektowany z zamiarem połączenia najlepszych cech protokołów opartych na wektorze odległości i stanie łączego, przy jednoczesnym ominięciu ich ograniczeń wydajności i innych wad.

Protokoły wewnętrzne i zewnętrzne

Potrzebna informacja:

System autonomiczny – grupa sieci i routerów pod wspólną administracją (korporacje, uczelnie). Routery wewnętrz systemu autonomicznego dowolnie zarządzają trasami. Każdy system autonomiczny wybiera router lub routery przeznaczone do komunikacji z innymi systemami autonomicznymi. Odpowiadają one za przekazywanie informacji o osiągalności sieci wewnątrz „swojego” systemu do innych systemów.

Routery odpowiedzialne za komunikację z innymi systemami autonomicznymi nazywane są routerami zewnętrznymi albo brzegowymi (exterior gateways), routery działające wewnątrz systemu – wewnętrzny (interior gateways).

Zewnętrzne:

EGP(Exterior Gateway Protocol)

- Router może uzgodnić z innym routerem, że będą „sąsiadami”, tzn. będą wymieniać informacje o trasach.
- Router sprawdza co jakiś czas czy jego sąsiedzi działają.
- Sąsiedzi wymieniają komunikaty pozwalające zaktualizować tablice routingu. Komunikat taki zawiera listę znanych danemu routerowi sieci i odległości do nich

Inny protokoł tego typu: (E) *BGP (Exterior Border Gateway Protocol)*

Wewnętrzne:

Grupę protokołów używanych przez routery wewnątrz systemu autonomicznego określa się nazwą *IGP (Interior Gateway Protocols)*

Przykładowe protokoły z tej grupy:

- **RIP**
- **HELLO**
- **OSPF**

RIP - Routing Information Protocol

- Implementacja algorytmu wektor-odległość dla sieci lokalnych
- Odległość mierzona jako „*hop count*”
- Liczba routerów między rozważanymi sieciami
- Przeznaczony dla niewielkich sieci – odległość 16 traktowana jest jako nieskończoność

HELLO

- Protokół bazujący na algorytmie „wektor odległość”
- Do oceny odległości używa opóźnień (tj. czasu potrzebnego na dostarczenie komunikatu za pośrednictwem sieci), a nie liczby routerów pośredniczących

Miary trasowania

W jaki sposób algorytmy trasowania decydują o tym, że jedna trasa jest preferowana bardziej niż inna? Rozróżnia się obecnie następujące miary trasowania:

- długość ścieżki
- niezawodność
- opóźnienie
- szerokość pasma
- obciążenie
- koszt komunikacji

Długość ścieżki jest najczęściej używaną miarą trasowania. Niektóre protokoły trasowania zezwalają administratorowi sieci na arbitralne przypisywanie kosztów każdemu łączu sieciowemu. W takim wypadku koszt ścieżki jest sumą kosztów związanych z każdym łączem składającym się na ścieżkę. Inne protokoły trasowania natomiast używają miary hop count, rozumianej jako liczba przejść pakietu przez urządzenia intersieciowe - np. routery - od stacji nadawczej do stacji odbiorczej.

Niezawodność, w kontekście algorytmów trasowania, odnosi się do skuteczności każdego łącza (określonego liczbą przekłamanych bitów). Niektóre łącza mogą ulegać uszkodzeniom częściej od innych. Po uszkodzeniu sieci niektóre łącza można naprawić szybciej i prościej niż inne.

Opóźnienie trasowania oznacza czas potrzebny do przesłania pakietu od stacji nadawczej do stacji odbiorczej w intersieci.

Szerokość pasma odnosi się do dostępnej pojemności ruchu w określonym łączu

Obciążenie to stopień zajętości zasobu sieciowego, np. routera. Obciążenie zależy od wielu czynników, np. stopnia wykorzystania procesora czy liczby pakietów przetwarzanych w czasie jednej sekundy.

Koszt komunikacji jest ważną miarą trasowania, przede wszystkim dlatego, że niektóre firmy nie dbają o wydajność. Nawet wtedy, gdy opóźnienia są duże, przesyłają pakiety przez własne linie zamiast korzystać z sieci publicznych, za które się płaci tylko w czasie ich używania.

44. Usługi nazewnicze sieci TCP/IP.

Ogólny Zarys

Usługi nazewnicze wykorzystywane są do dystrybuowania informacji. One tłumaczą nazwy hostów na adresy IP. Internetowym standardem jest DNS, ale w pewnych sytuacjach wykorzystywane są NIS i WINS.

DNS(ang. Domain Name System, system nazw domenowych) to system serwerów oraz protokół komunikacyjny zapewniający zamianę adresów znanych użytkownikom Internetu na adresy zrozumiałe dla urządzeń tworzących sieć komputerową. Dzięki wykorzystaniu DNS nazwa mnemoniczna, np. pl.wikipedia.org, może zostać zamieniona na odpowiadający jej adres IP, czyli 145.97.39.135.

Adresy DNS składają się z domen internetowych rozzielonych kropkami. Dla przykładu w adresie Wikipedii *org* oznacza domenę funkcjonalną organizacji, wikipedia domenę należącą do fundacji Wikimedia, a *pl* polską domenę w sieci tej instytucji. W ten sposób możliwe jest budowanie hierarchii nazw, które porządkują Internet.

Strona Techniczna

Podstawą technicznego systemu DNS jest ogólnoświatowa sieć serwerów przechowujących informacje na temat adresów domen. Każdy wpis zawiera nazwę oraz odpowiadającą jej wartość, najczęściej adres IP. System DNS jest podstawą dla rozwiązywania nazw hostów w Internecie.

DNS to również protokół komunikacyjny opisujący sposób łączenia się klientów z serwerami DNS. Częścią specyfikacji protokołu jest również zestaw zaleceń, jak aktualizować wpisy w bazach domen internetowych. Na świecie jest wiele serwerów DNS, które odpowiadają za obsługę poszczególnych domen internetowych. Domeny mają strukturę drzewiastą, na szczytce znajduje się 13 głównych serwerów (*root servers*) obsługujących domeny najwyższego poziomu (*TLD – top level domains*).

Serwery najwyższego poziomu z reguły posiadają tylko odwołania do odpowiednich serwerów DNS odpowiedzialnych za domeny niższego rzędu, np. serwery główne (obsługujące między innymi [TLD.com](#)) wiedzą, które serwery DNS odpowiedzialne są za domenę [example.com](#). Serwery DNS zwracają nazwę serwerów odpowiedzialnych za domeny niższego rzędu. Możliwa jest sytuacja, że serwer główny odpowiada, że dane o domenie [example.com](#) posiada serwer [dns.example.com](#). W celu uniknięcia zapętlenia w takiej sytuacji serwer główny do odpowiedzi dołącza specjalny rekord (tak zwany [glue record](#)) zawierający także adres IP serwera niższego rzędu (w tym przypadku [dns.example.com](#)).

Wewnątrz każdej domeny można tworzyć tzw. subdomeny - stąd mówimy, że system domen jest 'hierarchiczny'. Przykładowo wewnątrz domeny .pl utworzono wiele domen:

- regionalnych jak 'opole.pl', 'dzierzoniow.pl' czy 'warmia.pl'
- funkcjonalnych jak 'com.pl', 'gov.pl' czy 'org.pl'
- należących do firm, organizacji lub osób prywatnych jak 'onet.pl' czy 'zus.pl'

Nazwy domen i poszczególnych komputerów składają się z pewnej liczby nazw, oddzielonych kropkami. Ostatnia z tych nazw jest domeną najwyższego poziomu. Każda z tych nazw może zawierać litery, cyfry lub znak '-'. Od niedawna w nazwach niektórych domen można używać znaków narodowych (IDN) takich jak 'ą' czy 'ż', ale większość współczesnych programów nie przewiduje możliwości wykorzystania takich funkcji. *Wewnątrz każdej z poddomen można tworzyć dalsze poddomeny, np. w domenie '[wikipedia.org](#)' można utworzyć [domenę pl.wikipedia.org](#).*

DNS, jako system organizacyjny, składa się z dwóch instytucji - IANA i ICANN. Nadzorujące ogólne zasady przyznawania nazw domen i adresów IP. Jednak te dwie instytucje nie są w stanie zajmować się całym światem i dlatego cedują swoje uprawnienia na szereg lokalnych instytucji i firm.

Najważniejsze cechy DNS:

- Nie ma jednej centralnej bazy danych adresów IP i nazw. Najważniejszych jest 13 głównych serwerów (klastrów) rozmieszczonych na wielu kontynentach
- Serwery DNS przechowują dane tylko wybranych domen.
- Każda domena powinna mieć co najmniej 2 serwery DNS obsługujące ją, jeśli więc nawet któryś z nich będzie nieczynny, to drugi może przejąć jego zadanie.
- Każda domena posiada jeden główny dla niej serwer DNS (tzw. master), na którym to wprowadza się konfigurację tej domeny, wszystkie inne serwery obsługujące tę domenę są typu slave i dane dotyczące tej domeny pobierają automatycznie z jej serwera głównego po każdej zmianie zawartości domeny.
- Serwery DNS mogą przechowywać przez pewien czas odpowiedzi z innych serwerów (*ang. caching*), a więc proces zamiany nazw na adresy IP jest często krótszy niż w podanym przykładzie.
- Na dany adres IP może wskazywać wiele różnych nazw. Na przykład na adres IP 207.142.131.245 mogą wskazywać nazwy pl.wikipedia.org oraz de.wikipedia.org
- Czasami pod jedną nazwą może kryć się więcej niż 1 adres IP po to, aby jeśli jeden z nich zawiedzie, inny mógł spełnić jego rolę.
- Przy zmianie adresu IP komputera pełniącego funkcję serwera WWW, nie ma konieczności zmiany adresu internetowego strony, a jedynie poprawy wpisu w serwerze DNS obsługującym domenę.
- Protokół DNS posługuje się do komunikacji serwer-klient głównie protokołem UDP, serwer pracuje na porcie numer 53, przesyłanie domeny pomiędzy serwerami master i slave odbywa się protokołem TCP na porcie 53.

Rodzaje zapytań DNS

- **Rekurencyjne**

Zmusza serwer do znalezienia wymaganej informacji lub zwrócenia wiadomości o błędzie. Ogólną zasadą jest, że zapytania od resolwera (program, który potrafi wysyłać zapytania do serwerów DNS) do serwera są typu rekurencyjnego, czyli resolwer oczekuje podania przez serwer adresu IP poszukiwanego hosta. Wykonywanie zapytań rekurencyjnych pozwala wszystkim uczestniczącym serwerom zapamiętać odwzorowanie (ang. *DNS caching*), co podnosi efektywność systemu.

- **Iteracyjne**

Wymaga od serwera jedynie podania najlepszej dostępnej mu w danej chwili odpowiedzi, przy czym nie musi on łączyć się jeszcze z innymi serwerami. Zapytania wysyłane pomiędzy serwerami są iteracyjne, przykładowo wiarygodny serwer domeny org nie musi znać adresu IP komputera www.pl.wikipedia.org, podaje więc najlepszą znaną mu w tej chwili odpowiedź, czyli adresy serwerów autorytatywnych dla domeny wikipedia.org

Odpowiedzi na zapytania

- **Autorytatywne**

Dotyczące domeny w strefie, nad którą dany serwer ma zarząd, pochodzą one bezpośrednio z bazy danych serwera; jest to pozytywna odpowiedź zwracana do klienta, która w komunikacie DNS zawiera ustawiony bit uwierzytelniania (AA – Authoritative Answer) wskazujący, że odpowiedź została uzyskana z serwera dokonującego bezpośredniego uwierzytelnienia poszukiwanej nazwy

- **Nieautorytatywne**

Dane które zwraca serwer pochodzą spoza zarządzanej przez niego strefy; odpowiedzi nieautorytatywne są buforowane poprzez serwer przez czas TTL wyrażony w sekundach, wyspecyfikowany w odpowiedzi, a następnie po upływie czasu są usuwane

Komunikaty DNS

Zapytania i odpowiedzi DNS są najczęściej transportowane w pakietach [UDP](#). Każdy komunikat musi się zatrzymać w jednym pakiecie UDP (standardowo 512 oktetów, ale wielkość tę można zmieniać pamiętając również o ustaleniu takiej samej wielkości w [MTU](#) – Maximum Transmission Unit). W innym przypadku przesyłany jest protokołem [TCP](#) i poprzedzony dwubajtową wartością określającą długość zapytania i długość odpowiedzi (bez wliczania tych dwóch bajtów).

Format Komunikatu DNS:

NAGŁÓWEK - (Header)

ZAPYTANIE - (Question) do serwera nazw

ODPOWIEDŹ - (Answer) zawiera rekordy będące odpowiedzią

ZWIERZCHNOŚĆ - (Authority) wskazuje serwery zwierzchnie dla domeny

DODATKOWE - (Additional) sekcja informacji dodatkowych

45. Zarządzanie konfiguracją urządzenia w sieci TCP/IP.

Ex. 1	Ex. 2	Ex. 3
Ts	Ts	Ts

46. Wirtualne sieci lokalne.

Wirtualna sieć lokalna, VLAN (ang. virtual local area network) - sieć komputerowa wydzielona logicznie w ramach innej, większej sieci fizycznej.

Wirtualne sieci lokalne (Virtual Local Area Networks, VLANs) umożliwiają podział większej fizycznej sieci komputerowej na logiczne, odizolowane segmenty. **Kształtowanie przepływu ruchu między sieciami VLAN odbywa się w warstwie 3. modelu OSI.**

Virtual LAN dzieli fizyczne łącza na logiczne segmenty, ale sposób zaprojektowania wirtualnej sieci lokalnej zależy od administratora, a raczej przyjętych w organizacji założeń w zakresie kształtowania przepływu ruchu oraz wymaganego poziomu bezpieczeństwa. W ten sposób na jednym fizycznym przełączniku można utworzyć dwie (lub więcej) odizolowane od siebie sieci lokalne.

Tylko urządzenia przynależące do tej samej sieci VLAN mogą komunikować się ze sobą, każda sieć VLAN tworzy bowiem niezależną domenę rozgłoszeniową. Przełączniki przekazują ruch transmisji pojedynczej (unicast), grupowej (multicast) i rozgłoszeniowej (broadcast) tylko w ramach jednego segmentu sieci LAN. Poza izolacją segmentów sieci podejście to pozwala też ograniczyć zalewanie portów przełącznika rozgłoszeniami z protokołów ARP i DHCP, które nigdy nie przekraczają granic sieci VLAN.

Mechanizm routingu między VLAN, choć wymaga zastosowania dodatkowych urządzeń, pozwala kształtować przepływy ruchu między poszczególnymi segmentami sieci komputerowej. Mowa tutaj o kontroli dostępu, filtrowaniu ruchu na porcie sieciowym czy zapewnianiu jakości usług (QoS).

Praktyczne zastosowanie

Sieć VLAN może służyć do segmentacji według struktury organizacyjnej. W instytucjach publicznych komputery pracowników działów finansowych i HR nie powinny komunikować się ze względów bezpieczeństwa z urządzeniami pozostałego personelu biurowego. Z kolei w firmie produkcyjnej technologia VLAN może odizolować ruch sieci komputerowej udostępnianej pracownikom biurowym od sieci komputerowej wykorzystywanej w wydziałach produkcyjnych na potrzeby zbierania danych i sterowania maszynami.

Inne praktyczne zastosowanie sieci VLAN to segmentacja ruchu sieciowego ze względu na jego typ. Podejście to sprawdzi się w każdej instytucji, nawet gdy nie ma jawniej potrzeby izolowania ruchu według struktury organizacyjnej. Oddzielne VLAN stosuje się dla serwerów, punktów końcowych (stacje robocze, laptopy), drukarek, urządzeń mobilnych (strategia BYOD), telefonów VoIP, sieci Wi-Fi dla gości, sieci zarządzania (management) czy strefy DMZ.

Protokół IEEE 802.1Q

VLAN to wydzielona logicznie sieć komputerowa **warstwy 2. (łącza danych)** modelu OSI. Grupuje logicznie porty jednego lub wielu przełączników sieciowych niezależnie od ich położenia. Podstawowym, powszechnie stosowanym protokołem oznaczania ramek i trunkingu jest IEEE 802.1Q. Protokół ten, nazywany także Dot1q, stał się branżowym standardem definiującym sposób obsługi VLAN w sieciach Ethernet.

Działanie sieci VLAN bazuje na dodawaniu 4-bajtowych znaczników (tagów) wewnątrz nagłówka ramek Ethernet, które pozwalają urządzeniom sieciowym sterować przepływem ruchu. Znacznik ten, o nazwie 802.1Q Header, umieszczany jest między polem adresu źródłowego (Source MAC) a polem wskazującym na typ ramki/długość (EtherType/Size). Pierwsze dwa bajty tego znacznika (Tag Protocol ID, TPID) mają stałą wartość 0x8100 i umożliwiają przełącznikowi odróżnienie znakowanej ramki 802.1Q od ramki nieznakowanej, która w tym miejscu miałaby pole EtherType/Size. Pozostałe dwa bajty (Tag Control Information, TCI) zawierają informacje służące do oznaczenia priorytetu ramki (definiowany w standardzie 802.1p), standardu sieci LAN

(Ethernet lub Token Ring) oraz numeru wirtualnej sieci (VLAN ID), do której przynależy dana ramka. Wspomniane pole VLAN ID, stanowiące identyfikator sieci wirtualnej, ma długość 12 bitów i pozwala skutecznie przypisać ramkę do właściwego segmentu VLAN. W rezultacie na przełączniku można zdefiniować maksymalnie do 4096 sieci VLAN, z czego dwie są zarezerwowane do innych celów, a VLAN 1 pełni funkcję sieci natywnej.

W tym miejscu warto też wspomnieć o innym protokole znakowania i trunkingu. InterSwitch Link (ISL) to własnościowy protokół Cisco używany w przełącznikach tej firmy. Oryginalna ramka Ethernet pozostaje niezmieniona, jest bowiem kapsułkowana w ramce ISL, której nagłówek zawiera znacznik VLAN ID. Protokół ISL został uznany za przestarzały, nie powinien być dalej używany. Co więcej, nie jest wspierany przez najnowsze przełączniki Cisco.

Punkty końcowe mogą komunikować się ze sobą w ramach jednej sieci VLAN. Przekazywanie ruchu sieciowego między sieciami VLAN wymaga zastosowania routera lub przełącznika działającego w warstwie 3. (sieci) modelu OSI.

47. Technologie redundantne w sieciach komputerowych.

Ex. 1	Ex. 2	Ex. 3
Ts	Ts	Ts

48. Metody optymalizacji zapytań SQL.

Klasyfikacja metod optymalizacji

Są 2 rodzaje optymalizacji:

- Optymalizacja Dynamiczna
- Optymalizacja Statyczna

W zależności od liczby optymalizowanych zapytań mamy następny podział:

- Optymalizacja pojedynczego zapytania
- Jednoczesna optymalizacja zbioru zapytań

Pierwsza z podanych klasyfikacji wyróżnia optymalizację statyczną i optymalizację dynamiczną. Optymalizacja statyczna polega na znalezieniu „najlepszego” planu wykonania zapytania, przed rozpoczęciem wykonywania zapytania. W trakcie realizacji zapytania plan wykonania zapytania nie ulega już zmianie – stąd nazwa optymalizacja statyczna. Optymalizacja dynamiczna polega na znalezieniu „najlepszego” planu wykonania zapytania, przed rozpoczęciem wykonywania zapytania, ale później, w trakcie wykonywania zapytania jego plan wykonania może ulegać zmianie. Aktualnie, komercyjne systemy baz danych zapewniają jedynie optymalizację statyczną, choć efektywność takiej optymalizacji jest najczęściej niższa aniżeli efektywność optymalizacji dynamicznej. Optymalizacja dynamiczna jest jednak znacznie bardziej kosztowna. Druga z podanych klasyfikacji wyróżnia optymalizację pojedynczego zapytania oraz jednoczesną optymalizację wielu zapytań. W przypadku optymalizacji pojedynczego zapytania, optymalizacji podlega tylko jedno zapytanie. W przypadku jednoczesnej optymalizacji wielu zapytań, częściowe wyniki wykonania jednego zapytania mogą być wykorzystane przez wiele innych zapytań, co prowadzi do minimalizacji czasu wykonania zbioru zapytań. W chwili obecnej systemy komercyjnych baz danych zapewniają jedynie optymalizację pojedynczego zapytania.

Ogólny proces optymalizacji zapytań:

- Transformacja zapytania SQL do postaci drzewa wyrażenia logicznego:
 - Identyfikacja bloków zapytania (odpowiadających zagnieżdżonym zapytaniom lub perspektywom)
- Faza przepisywania zapytania:
 - Zastosowania **transformacji algebraicznych** w celu uzyskania tańszego planu wykonania zapytania
- Optymalizacja bloku: zdefiniowania porządku wykonywania operacji połączenia
- Zakończenie optymalizacji: wybór uszeregowania

W wyniku zastosowania transformacji algebraicznych uzyskujemy zbiór najlepszych planów wykonania pojedynczych bloków zapytania. Pozostaje jeszcze problem połączenia bloków, w szczególności, problem zdefiniowania porządku wykonywania operacji połączenia. Wybór kolejności wykonywania operacji połączenia, tzn. wybór uszeregowania operacji połączenia, kończy proces optymalizacji zapytania.

Szczegółowy opis faz przetwarzania danych

Dekompozycja

Pierwszą fazą przetwarzania zapytania jest **dekompozycja** zapytania. Celem procesu dekompozycji zapytania jest transformacja zapytania wyrażonego w języku wysokiego poziomu na wyrażenie *algebry relacji* i weryfikacja syntaktycznej i semantycznej poprawności zapytania. Proces dekompozycji składa się z następujących etapów:

- analiza zapytania
- normalizacja zapytania
- analiza semantyczna zapytania
- upraszczanie zapytania
- restrukturyzacja zapytania

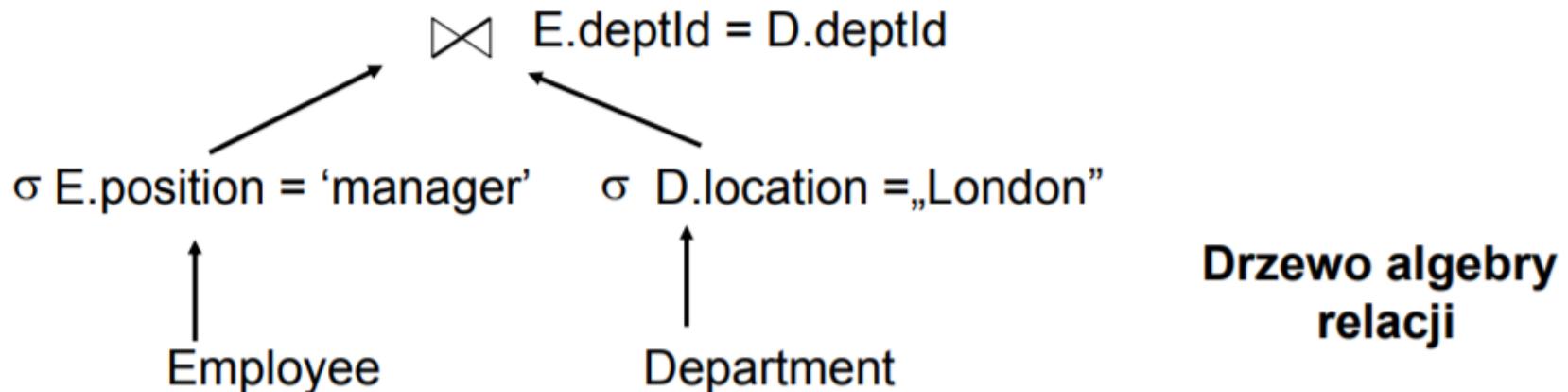
Analiza zapytania

Celem etapu analizy jest analiza syntaktyczna poprawności zapytania. W skład tej analizy wchodzi *weryfikacja poprawności atrybutów i relacji* (czy w bazie danych występują wyspecyfikowane w zapytaniu relacje i atrybuty, czy zapytanie poprawnie specyfikuje typy danych).

Następnie, zapytanie wyrażone w języku SQL jest transformowane do postaci reprezentacji wewnętrznej (wyrażenia algebry relacji, które można zapisać w postaci drzewa, jak na przykładzie poniżej), bardziej adekwatnej do procesu dalszego przetwarzania zapytania.

Przykład zapytania i odpowiadającego drzewa:

```
Select *
From Employee E, Department D
Where E.deptId = D.DeptId
And E.position = 'manager' and D.location = 'London';
```



Drzewo algebry relacji

Normalizacja

Kolejnym etapem fazy dekompozycji jest normalizacja zapytania. Celem etapu normalizacji zapytania jest przekształcenie wewnętrznej reprezentacji zapytania do znormalizowanej postaci koniunkcyjnej lub dysjunkcyjnej. W fazie tej sekwencja predykatów selekcji jest przekształcana do normalnej postaci koniunkcyjnej lub normalnej postaci dysjunkcyjnej. Postać dysjunkcyjna jest, najczęściej, mniej efektywna, gdyż wymaga niezależnego wartościowania poszczególnych składowych wyrażenia. Przykłady postaci koniunkcyjnej i dysjunkcyjnej wyrażenia zapytania:

- Dowolny predykat (w SQL) można przekształcić do jednej z dwóch postaci:

- Normalnej postaci koniunkcyjnej

(position=„manager” or salary > 1000) and deptId =100

- Normalnej postaci dysjunkcyjnej

(position=„manager” and salary > 1000) or deptId =100

Kolejnym, ważnym, etapem dekompozycji zapytania jest etap analizy semantycznej zapytania. Celem analizy semantycznej zapytania jest odrzucenie niepoprawnie sformułowanych lub sprzecznych zapytań. Zapytanie jest niepoprawnie sformułowane, jeżeli jego elementy składowe nie prowadzą do generacji wyniku. Zapytanie jest sprzeczne, jeżeli jego predykaty nie mogą być spełnione przez żadną krotkę w bazie danych. Przykładem klauzuli, która jest sprzeczna jest wyrażenie:
position = 'manager' and position = 'assistant'.

Zakładając, że baza danych jest w **1NF**, nie istnieje w bazie danych żadna krotka, któraaby jednocześnie spełniała oba predykaty. Wartość sprzecznej klauzuli interpretujemy jako wartość **FALSE**. W związku z tym, wyrażenie zawierające sprzeczną klauzulę można uprościć.

Przykładowo, wyrażenie

(position = 'manager' and position = 'assistant') or salary > 1000;

ze względu na sprzeczność klauzuli : „position = 'manager' and position = 'assistant'" można uprościć do postaci „salary > 1000”

Analiza semantyczna zapytania(2)

Niestety, algorytmy oceny poprawności semantycznej zapytań istnieją tylko dla pewnej klasy zapytań, nie zawierających dysjunkcji i negacji. W jaki sposób rozwiązywany jest problem zapytań niepoprawnie sformułowanych oraz zapytań sprzecznych? Rozwiążanie problemu zapytań niepoprawnie sformułowanych opiera się na konstrukcji tak zwanego **grafu połączenia relacji**. W grafie tym, wierzchołki odpowiadają relacjom, natomiast luki odpowiadają operacjom połączenia wyspecyfikowanych w zapytaniu. Dodatkowo, graf połączenia relacji zawiera wierzchołek reprezentujący wynik zapytania. Jeżeli graf połączenia relacji nie jest spójny, to zapytanie jest niepoprawnie sformułowane.

Rozwiążanie problemu zapytań *niepoprawnie sformułowanych* opiera się na konstrukcji tak zwanego **grafu połączeń atrybutów**.

Przykłady tworzenia tych grafów (str: 10-12)

Upraszczanie zapytania

Kolejnym etapem fazy dekompozycji jest *upraszczanie zapytań*. Celem tego etapu jest identyfikacja wyrażeń redundantnych, eliminacja wspólnych podwyrażeń, i transformacja zapytania do równoważnej postaci, ułatwiającej dalsze przekształcanie zapytania. Transformacja zapytania do postaci równoważnej polega na zastosowaniu znanych reguł algebraii relacji.

Restrukturyzacja

Kolejnym etapem fazy dekompozycji jest etap *restrukturyzacji*, czy też *transformacji* zapytania. Zanim jednak przejdziemy do przedstawienia podstawowych reguł transformacji, wróćmy na chwilę do problemu konstrukcji podstawowych bloków zapytania. Tradycyjne podejście do konstrukcji bloków zapytania opera się na zastosowaniu transformacji algebraicznych, jednakże, zbiór stosowanych transformacji różni się zasadniczo dla różnych systemów komercyjnych.

Co więcej, nie wszystkie transformacje gwarantują minimalizację czasu wykonania danego bloku. W ostatnim czasie, coraz częściej, konstrukcja bloków opiera się na optymalizacji kosztowej, w której, dla każdego bloku, konstruujemy możliwe plany wykonania danego bloku i szacujemy koszt i rozmiar wykonania każdego planu. Ostatecznie wybierany jest plan wykonania o najniższym szacowanym koszcie. Do zakończenia procesu optymalizacji pozostaje jeszcze znalezienie najlepszego drzewa operacji połączenia, łączącego wyniki wykonania bloków zapytania. Tradycyjne podejście do problemu znajdowania najlepszego drzewa operacji połączenia (nazywane często podejściem w stylu systemu R) polega na zastosowaniu *algorytmu programowania dynamicznego*.

Operacje

Każdy plan wykonania zapytania jest częściowo uporządkowanym zbiorem operacji. W skład tego zbioru operacji wchodzą: operacja skanowania, selekcji, projekcji, połączenia, produktu kartezjańskiego, operacje grupowania i agregacji. Problem znalezienia najlepszego planu wykonania zapytania obejmuje, z jednej strony, określenie kolejności wykonania operacji wchodzących w skład zapytania, z drugiej, określenia metody wykonania poszczególnych operacji. Przykładowo, mamy dwie metody dostępu do relacji: *bezpośrednie skanowanie (odczyt)* relacji lub dostęp do relacji poprzez *skanowanie indeksu założonego na relacji*. Podstawowa reguła optymalizacji mówi, że wszystkie operacje unarne (projekcja i selekcja) należy przesunąć w dół drzewa zapytania, tzn. *wykonywać w pierwszej kolejności*. Operacje te charakteryzują się silną własnością redukcji (filtrowania) przetwarzanych danych. Redukując rozmiar przetwarzanych danych, operacje unarne prowadzą do poprawy efektywności wykonywania operacji binarnych. Dlatego, operacje binarne (połączenie, produkt kartezjański) należy przesunąć w kierunku korzenia drzewa zapytania. Dla operacji binarnych, np. połączenia, poza określeniem kolejności ich wykonywania, należy wybrać również metodę ich wykonania (dla połączenia - nested loop, sort-merge, hash-join). Najczęściej, na końcu planu wykonania zapytania znajdują się operacje grupowania i agregacji.

Reguły transformacji oparte na algebrze relacji (Strony: 17-20)

Ważne

Zapytania zawierające skorelowane podzapytania zagnieżdżone są kosztowne w realizacji, gdyż wymagają sprawdzenia, dla każdej krotki zapytania zewnętrznego, czy spełniony jest dla tej krotki warunek podzapytania skorelowanego. Klasyczna metoda transformacji takich zapytań polega na przepisaniu zapytania w taki sposób, aby usunąć zagnieżdżenie (ang. unnesting). Usunięcie zagnieżdżenia polega na **zastąpieniu zagnieżdżenia operacją połączenia**.

Przykłady transformacji zagnieżdzonych zapytań (Strony: 21-26)

Zagadnienie optymalizacji jest zagadnieniem trudnym i istnieje bardzo wiele, specyficznych, reguł transformacji dla różnych typów zapytań. Co więcej, nie zawsze jest możliwe przetransformowanie zapytań w taki sposób, aby nie zawierało podzapytań (szczególnie dla podzapytań skorelowanych). *W szczególnych przypadkach, gdy czas realizacji zapytania jest nieakceptowalny, można zastosować technikę redukcji rozmiarów relacji uczestniczących w zapytaniu opartą o sekwencję operacji półpołączenia*. Technika ta jest wykorzystywana do optymalizacji zapytań rozproszonych w systemach rozproszonych baz danych.

49. Modele uwierzytelniania, autoryzacji i kontroli dostępu do systemów komputerowych.

Uwierzytelnianie

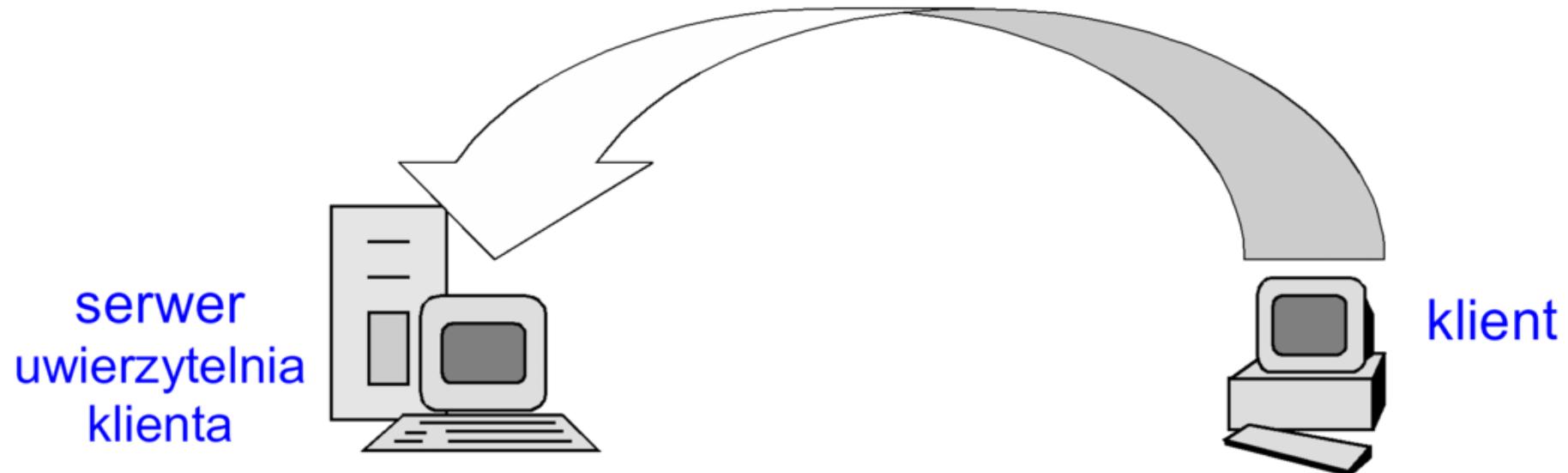
Uwierzytelnianie (ang. authentication) - proces polegający na potwierdzeniu zadeklarowanej tożsamości podmiotu biorącego udział w procesie komunikacji. Celem uwierzytelniania jest uzyskanie określonego poziomu pewności, że dany podmiot jest w rzeczywistości tym, za który się podaje.

W systemach informatycznych stosuje się następujące rodzaje uwierzytelniania:

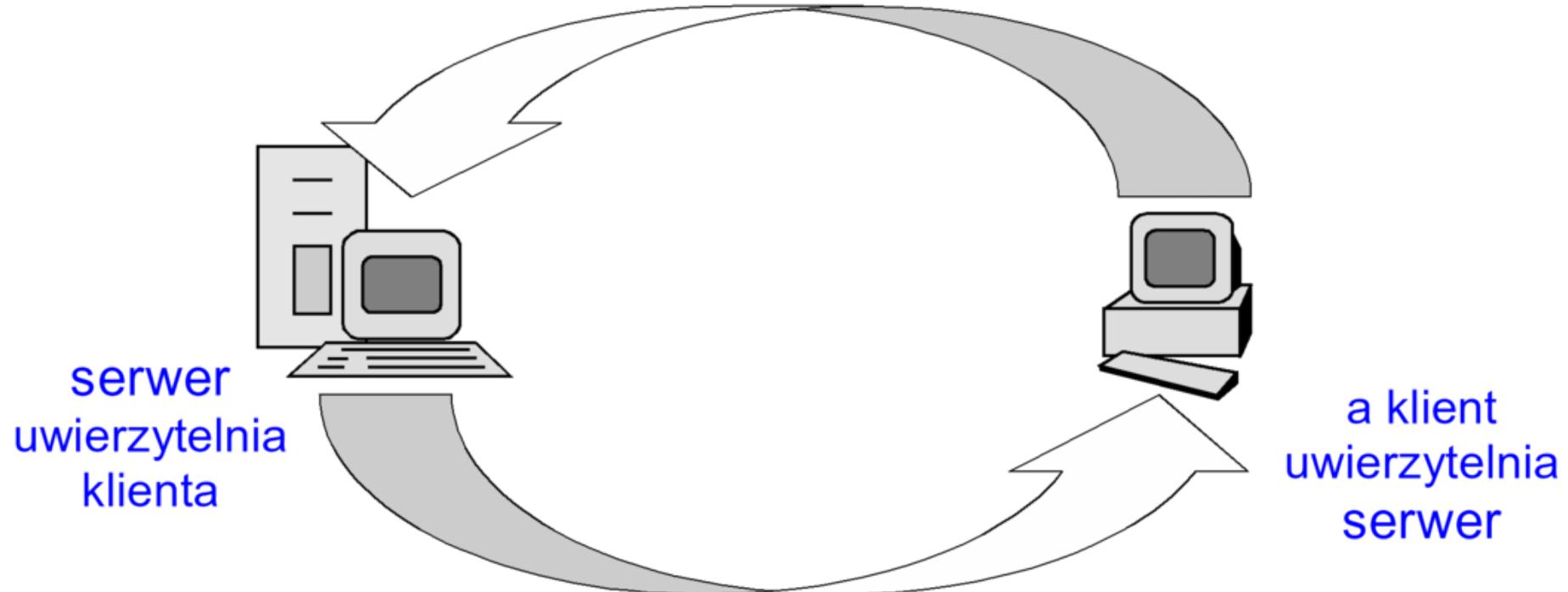
1. *Uwierzytelnianie jednokierunkowe* - polega na uwierzytelnieniu jednego podmiotu (uwierzytelnianego), np. klienta aplikacji, wobec drugiego (uwierzytelniającego) – serwera.

Uwierzytelnienie następuje poprzez zweryfikowanie danych uwierzytelniających przekazanych przez podmiot uwierzytelniany. Typowymi danymi uwierzytelniającymi są np. identyfikator użytkownika i jego hasło dostępu.

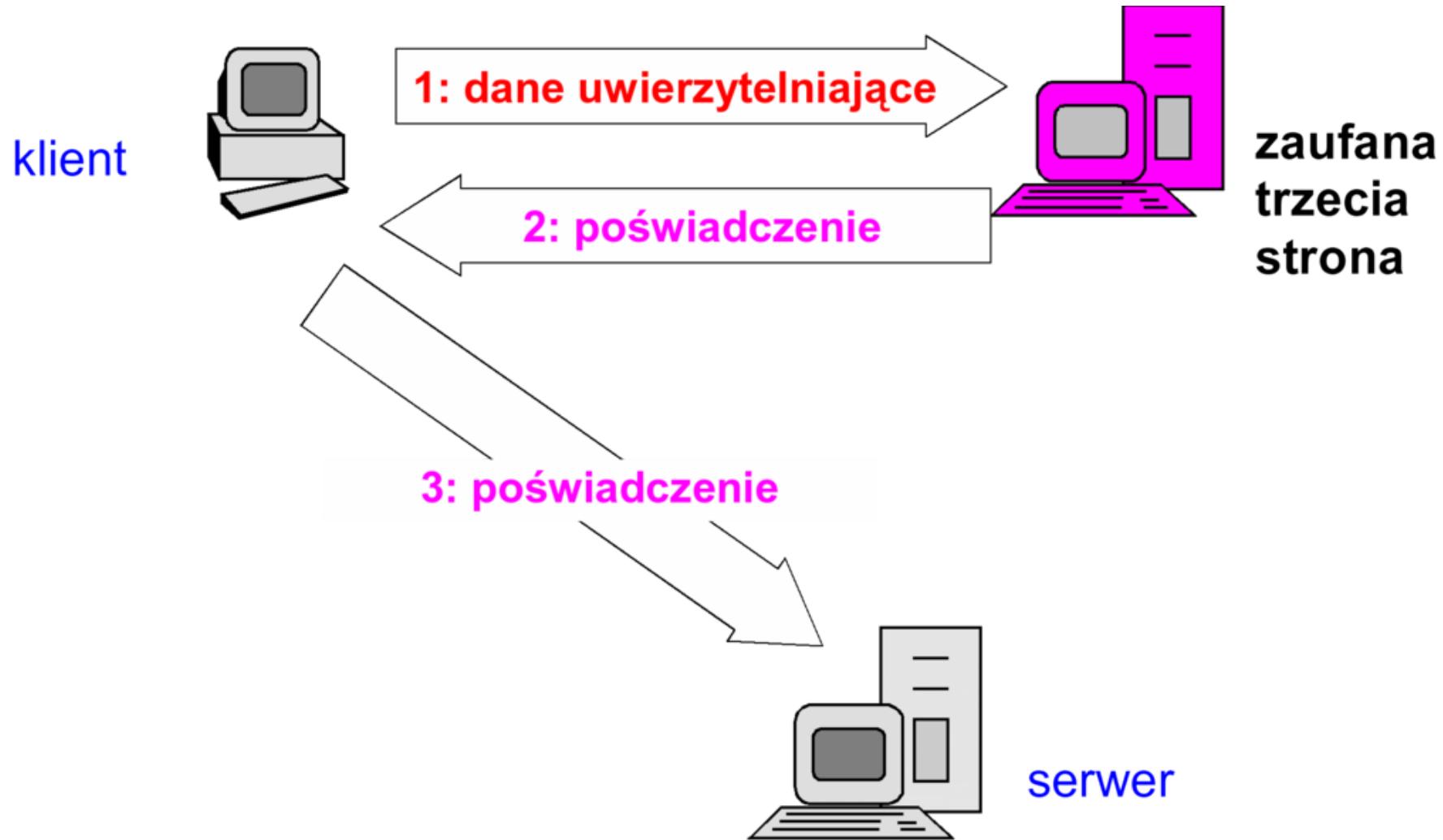
dane uwierzytelniające



2. *Uwierzytelnianie dwukierunkowe* - polega na kolejnym lub jednoczesnym uwierzytelnieniu obu podmiotów (które są wzajemnie i naprzemiennie uwierzytelnianym oraz uwierzytelniającym). Jeżeli wzajemne uwierzytelnianie następuje sekwencyjnie (np. najpierw klient wobec serwera, a później serwer wobec klienta), mówimy o uwierzytelnianiu dwuetapowym, natomiast jednoczesne uwierzytelnienie obu stron nazywamy jednoetapowym.



3. *Uwierzytelnianie z udziałem zaufanej trzeciej strony* – włącza w proces uwierzytelniania trzecią zaufaną stronę, która bierze na siebie ciężar weryfikacji danych uwierzytelniających podmiotu uwierzytelnianego. Po pomyślnej weryfikacji podmiot uwierzytelniany otrzymuje poświadczenie, które następnie przedstawia zarządcy zasobu, do którego dostępu żąda (serwerowi). Podstawową zaletą tego podejścia jest przesunięcie newralgicznej operacji uwierzytelniania do wyróżnionego stanowiska, które można poddać szczególnie podwyższonemu zabezpieczeniu. Należy też podkreślić potencjalną możliwość wielokrotnego wykorzystania wydanego poświadczenia (przy dostępie klienta do wielu zasobów, serwerów). Zaufana trzecia strona może być lokalna dla danej sieci komputerowej (korporacyjnej) lub zewnętrzna (wykorzystująca infrastrukturę uwierzytelniania dostępną w sieci rozległej np. publiczne urzędy certyfikujące).

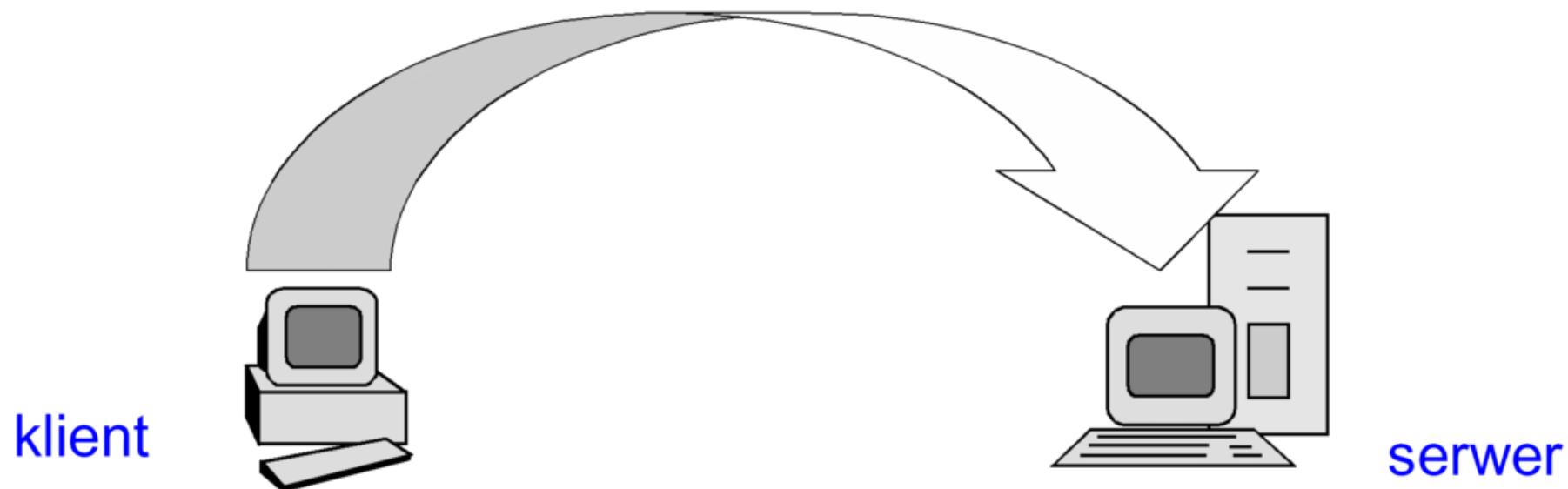


Mechanizmy uwierzytelniania

Klasyczne uwierzytelnianie użytkownika

W przypadku wielu współczesnych środowisk informatycznych, systemów operacyjnych lub systemów zarządzania bazami danych, funkcjonuje klasyczny mechanizm uwierzytelniania poprzez hasło. Proces uwierzytelniania rozpoczyna klient żądając zarejestrowania w systemie (login). Serwer pyta o identyfikator (nazwę) użytkownika, a następnie o hasło i decyduje o dopuszczeniu do sieci. W większości przypadków nazwa użytkownika i hasło są przesyłane tekstem jawnym, co stanowić może kolejny problem zapewnienia poufności, jaką właśnie mamy osiągnąć stosując opisywany mechanizm. Stąd też takie klasyczne podejście nadaje się do wykorzystania jedynie w ograniczonej liczbie przypadków, kiedy np. mamy uzasadnioną skądinąd pewność wykluczenia możliwości podsłuchu danych uwierzytelniających.

hasło



Hasła nie są

najefektywniejszą, ani najbezpieczniejszą formą weryfikacji tożsamości użytkownika, z następujących powodów:

- Hasło można złamać
- Odgadnąć, np. metodą przeszukiwania wyczerpującego (brute-force attack) lub słownikową (dictionary attack) - często hasła są wystarczająco nieskomplikowane by ułatwiło to odgadnięcie ich przez atakującego
- Podsłuchać w trakcie niezabezpieczonej transmisji
- Hasła się starzeją - czas przez który możemy z dużą pewnością polegać na tajności naszego hasła skraca się nieustannie, przez co hasła wymagają systematycznych zmian na nowe

Zdalne potwierdzanie tożsamości

W środowisku sieci TCP/IP wypracowano mechanizm prostego potwierdzania tożsamości użytkownika, który żąda zdalnego uwierzytelniania. W tym celu powstał standard RFC 1413 opisujący usługę o nazwie identd. Niezależnie od jej aktualnej przydatności i powszechności warto zdawać sobie sprawę z istoty jej działania, którą łatwo opisać w następujący sposób:

- Użytkownik uruchamia klienta usługi i nawiązuje połączenie z serwerem
- Serwer kontaktuje się z wydzielonym serwerem - identd, pracującym na stacji klienta (113/tcp) w celu poświadczania nazwy (lub identyfikatora) użytkownika wykorzystującego usługę



Należy też zdawać sobie sprawę z potencjalnych zagrożeń jakie niesie udostępnianie przez usługę ident informacji o przynależności procesów dokonujących komunikacji sieciowej (nie tylko klientów). W standardzie RFC 1413 oraz w praktycznych implementacjach **nie realizuje się bowiem uwierzytelniania podmiotu żądającego informacji z tej usługi**, może ona być zatem również nadużyta przez potencjalnego włamywacza.

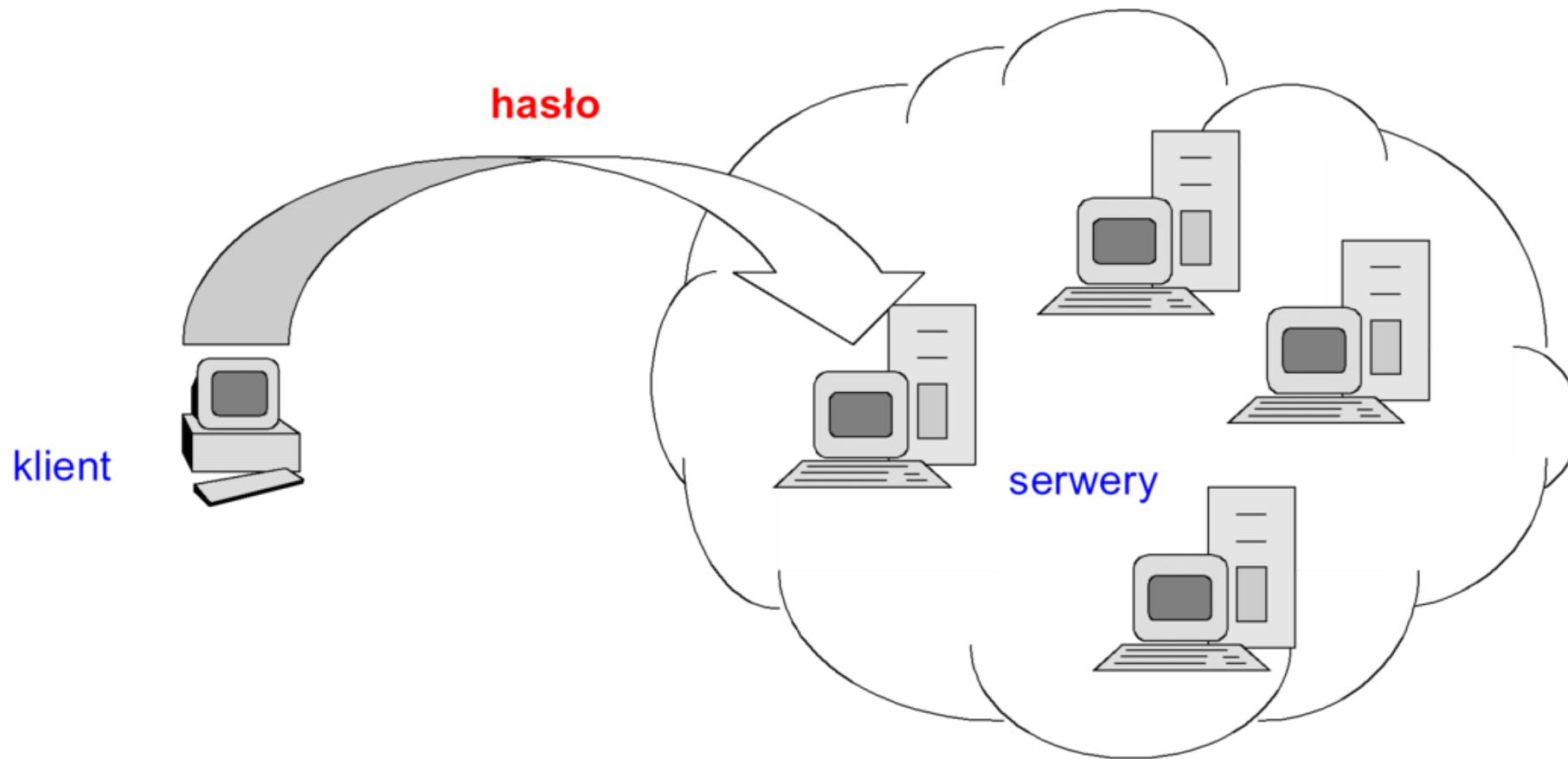
Uwierzytelnianie jednokrotne (SSO – single sign-on)

Procedury uwierzytelniania jednokrotnego są częściowym rozwiązaniem problemu ochrony danych uwierzytelniających przed złamaniem w systemie wielozasobowym, np. sieci komputerowej z wieloma serwerami.

Ideą procedury uwierzytelniania jednokrotnego jest minimalizacja ilości wystąpień danych uwierzytelniających w systemie - hasło powinno być podawana jak najrzadziej. Zgodnie z tą zasadą, jeśli jeden z komponentów systemu (np. system operacyjny) dokonał pomyślnie uwierzytelniania użytkownika, pozostałe komponenty (np. inne systemy lub zarządcy zasobów) ufać będą tej operacji i nie będą samodzielnie wymagać ponownie danych uwierzytelniających. Przy tym jest możliwe teoretycznie, że wszystkie komponenty samodzielnie korzystają z

odmiennych mechanizmów uwierzytelniana. Wówczas, dodatkowo po pierwszorzazowym uwierzytelnieniu użytkownika, system może oddelegować specjalny moduł do przechowywania odrębnych danych uwierzytelniających użytkownika i poświadczania w przyszłości jego tożsamości wobec innych komponentów systemu.

chemat SSO przedstawia poniższy rysunek. W przedstawionej na rysunku sytuacji tylko jeden serwer dokonuje uwierzytelniania klienta, reszta ufa uwierzytelnianiu dokonanemu przez ten serwer.



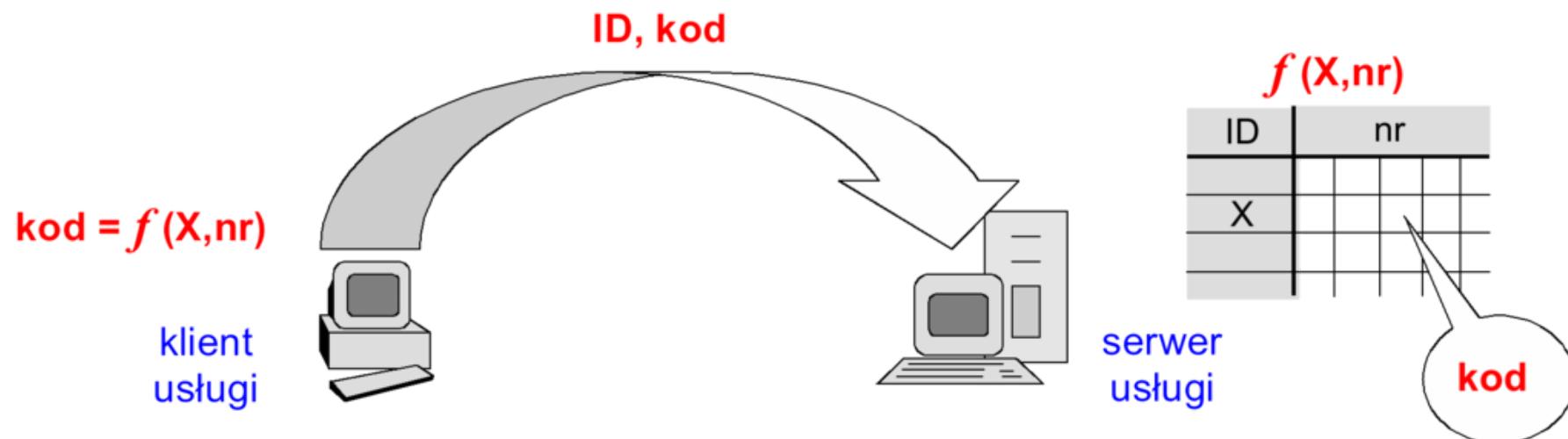
Hasła jednorazowe

Istota wykorzystania haseł jednorazowych wynika zamiaru ochrony ich przed przechwyceniem i nieautoryzowanym wykorzystaniem, w przyszłości. Jednak nie polega na zapewnieniu ich poufności w transmisji lecz na uczynieniu ich de facto bezwartościowymi po przechwyceniu. Opiera się na, jak sama nazwa wskazuje, tylko użyciu danej postaci hasła tylko raz. Hasła jednorazowe mają przy każdym kolejnym uwierzytelnieniu inną postać. Raz przechwycone hasło jednorazowe nie jest przydatne, bowiem przy kolejnym uwierzytelnieniu będzie obowiązywać już inne. *Komunikacja między podmiotami procesu uwierzytelniania może być zatem jawną*. Stosujące takie hasła procedury uwierzytelniania muszą jedynie oferować brak możliwości odgadnięcia na podstawie jednego z haseł, hasła następnego.

Hasła jednorazowe generowane są przy pomocy listy haseł, synchronizacji czasu lub metody zwołanie-odzew. Dostępne są najczęściej w następujących postaciach: listy papierowe, listy-zdrapki, tokeny programowe i tokeny sprzętowe.

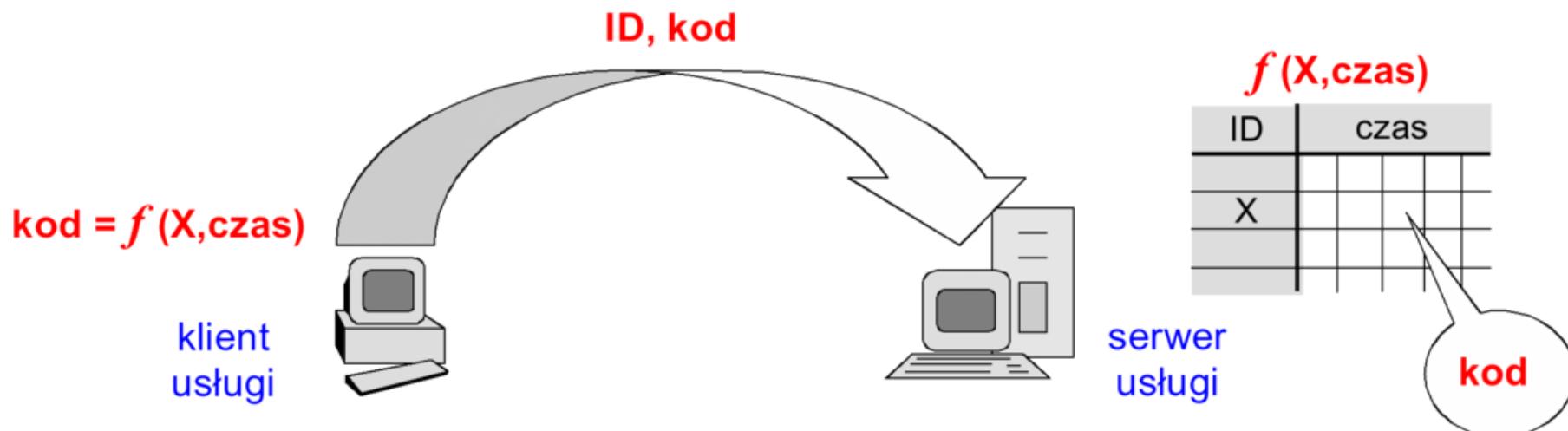
Lista haseł

Listy haseł to najprostsza i najtańsza metoda identyfikacji metodą haseł jednorazowych. Użytkownik otrzymuje listę zawierającą ponumerowane hasła. Ta sama lista zostaje zapisana w bazie systemu identyfikującego. W trakcie logowania użytkownik podaje swój identyfikator, a system prosi o podanie hasła z odpowiednim numerem. Klient za każdym razem posługuje się kolejnym niewykorzystanym hasłem z listy.



Metoda synchronizacji czasowej

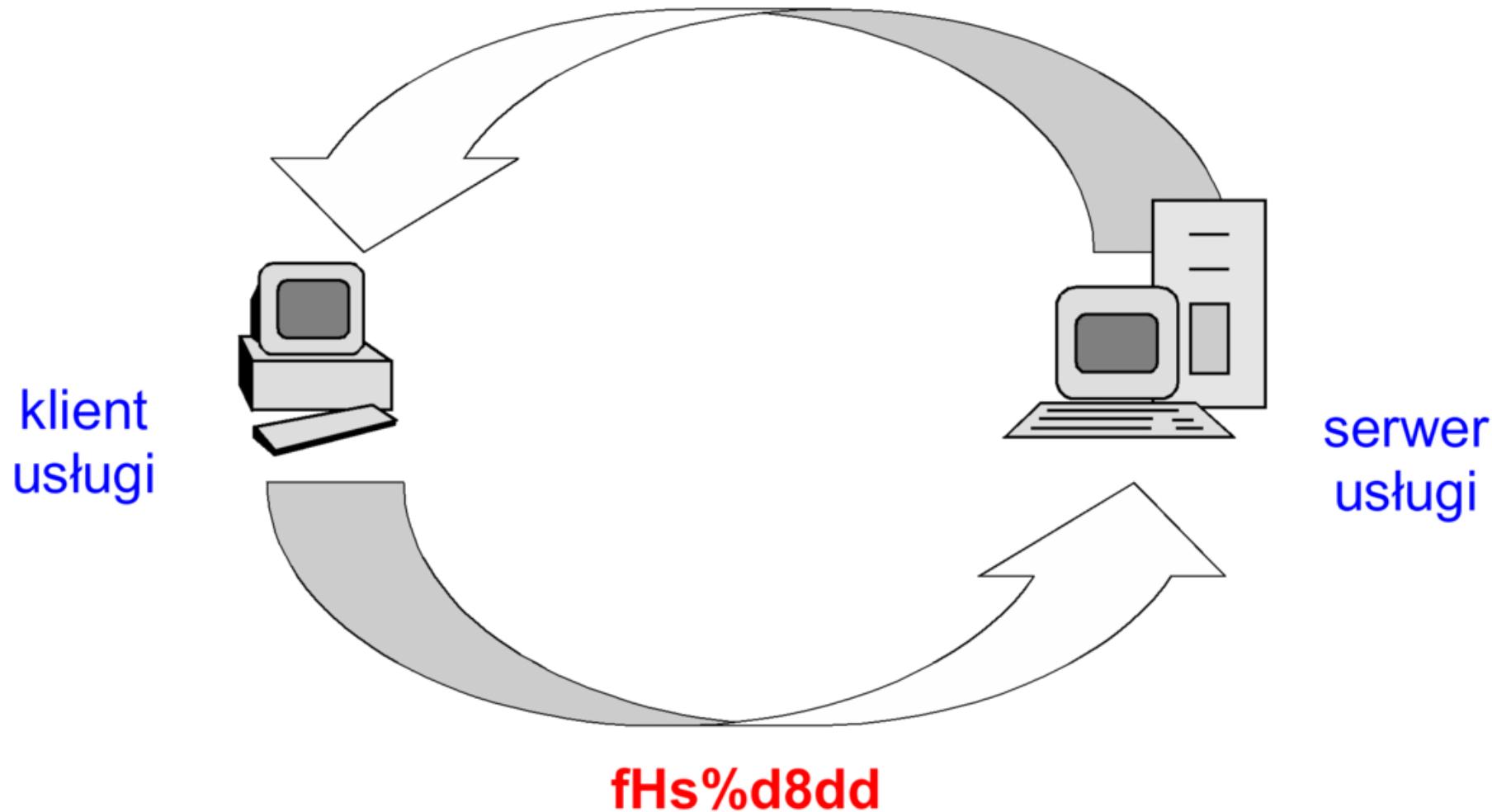
W metodzie z synchronizacją czasu (*time synchronization*) klient generuje unikalny kod w funkcji pewnego parametru X użytkownika (identyfikatora, kodu pin, hasła, numeru seryjnego karty identyfikacyjnej) oraz bieżącego czasu. Serwer następnie weryfikuje otrzymany od klienta kod korzystając z identycznej funkcji (z odpowiednią tolerancją czasu).



Metoda "Zwołanie-Odzew"

Natomiast w metodzie zwołanie-odzew (challenge-response) serwer pyta o nazwę użytkownika, a następnie przesyła unikalny ciąg („zwołanie”). Klient koduje otrzymany ciąg (np. swoim hasłem lub innym tajnym parametrem pełniącym rolę klucza) i odsyła jako „odzew”. Serwer posługując się identycznym kluczem weryfikuje poprawność odzachu.

18d78\$3fx337A&4665%8



Metoda Tokenów

Tokeny programowe to specjalne programy generujące hasła. W zależności od implementacji program na podstawie kwantu czasu lub zwołania serwera generuje hasło jednorazowe, które weryfikuje serwer.

Token sprzętowy jest małym przenośnym urządzeniem spełniającym wszystkie funkcje tokenu programowego.

Pewną ciekawostką zyskującą na popularności jest wykorzystanie telefonu komórkowego w uwierzytelnianiu za pomocą haseł jednorazowych. Cały proces polega przesyłaniu hasła jednorazowego z serwera na telefon w postaci wiadomości SMS. W tym przypadku rola telefonu jako swoistego tokena sprowadza się tylko do medium odbierającego i wyświetlającego dane.

Inne mechanizmy uwierzytelniania

Do uwierzytelniania użytkowników można wykorzystać również przedmioty, których posiadaniem musi się wykazać uwierzytelniany. Mogą to być np. karty magnetyczne, karty elektroniczne czy tokeny USB. Ponadto, w przypadku ludzi, można posłużyć się również cechami osobowymi wynikającymi z odmienności parametrów niektórych naturalnych składników organizmu (uwierzytelnianie biometryczne), takich jak [m.in.:](#)

- Klucz DNA
- Geometria twarzy
- Termogram twarzy
- Termogram dłoni
- Odcisk palca (dermatoglify)
- Tęczówka oka

Autoryzacja i kontrola dostępu

Autoryzacja i kontrola dostępu zaczyna się tam, gdzie kończy się uwierzytelnianie. Kiedy podmiot zabezpieczeń SP (Security Principal) podejmuje próbę uzyskania dostępu do chronionego obiektu lub usługi, proces autoryzacji przejmuje jego tożsamość i używa jej do określenia jego uprawnień.

Zadania autoryzacji i kontroli dostępu legalnych użytkowników należą do podstawowych funkcji systemów operacyjnych czy systemów zarządzania bazą danych oraz środowisk przetwarzania rozproszonego. W większości przypadków te funkcje są realizowane podobnie.

Jeżeli SP próbuje uzyskać dostęp (np. do pliku na serwerze WWW), usługa autoryzacji kwerenduje bazę danych, aby określić, jakie uprawnienia związane z tym plikiem ma SP. System kontroli dostępu jest programem lub procesem egzekwującym uprawnienia i przywileje. Generalnie, część autoryzacyjna systemu określa, że SP może tylko czytać dany plik, a system kontroli dostępu w rzeczywistości zapewnia, że nie może tego pliku zmodyfikować.

Modele kontroli dostępu opisują ogólne metody używane w poszczególnych domenach bezpieczeństwa do kontroli dostępu na styku SP i żądanej usługi czy obiektu.

Trzy modele kontroli dostępu do danych:

• **DAC (Discretionary Access Control)**

Uznaniowa kontrola dostępu jest tradycyjną metodą kontroli: tożsamość SP ma przyznane (pośrednio lub jawnie) jedno lub więcej uprawnień do obiektu. W tym systemie uprawnienia są przechowywane z każdym obiektem. Kiedy SP zamierza uzyskać dostęp do obiektu lub usługi, mechanizm kontroli dostępu otrzymuje jego tożsamość i wtedy kwerenduje obiekt (lub tablice uprawnień) w celu sprawdzenia, jakie uprawnienia mu przydzielono.

• **MAC (Mandatory Access Control)**

System obowiązkowej kontroli dostępu jest oparty na kategoryzujących etykietach bezpieczeństwa, które są przypisywane wszystkim SP i obiektom. SP z określoną etykietą bezpieczeństwa może wykonywać operacje (czytania, pisania lub usuwania) tylko na tych obiektach, które mają taką samą lub niższą w hierarchii etykietę MAC. Firma może np. etykietować całą zawartość według kategorii: "tajna", "poufna", "prywatna", "publiczna" i tylko niektórym pracownikom przypisać dostęp do wszystkich kategorii zawartości. Jedną z

głównych wad MAC jest to, że większość SP musi być dodatkowo ograniczana na zasadzie niezbędności dostępu do poszczególnych zawartości z danej kategorii - użytkownik dopuszczony do informacji tajnych niekoniecznie musi mieć dostęp do wszystkich tajnych dokumentów. W praktyce MAC sprawdza się jako warstwa innego modelu kontroli dostępu.

- **RBAC (Role-Based Access Control)**

W tym systemie uprawnienia określa się na podstawie pełnionej roli i zakresu działania SP. W systemie RBAC role są tworzone i przydzielane zgodnie ze sprecyzowanymi uprawnieniami i przywilejami niezbędnymi do ich pełnienia. SP są przydzielane jedna lub więcej ról i ma on możliwość wykonywania jedynie tego, co zostało przypisane do danej roli. W porównaniu z innymi modelami, RBAC precyzyjniej wyznacza najmniejszy, niezbędny zbiór uprawnień SP.

Główna różnica między modelem dostępu RBAC a DAC polega na tym, że grupy DAC mają na ogół określać ogólną przynależność (np. zespół działu kadr), podczas gdy wyznacznikiem RBAC są działania (np. wykonywane przez pracownika działu kadr, działu płac itp.).

Wiele modeli RBAC (w tym sporo zaimplementowanych jako warstwa nad systemami DAC) dopuszcza tylko szczegółowe uprawnienia w trakcie wykonywania przez SP koniecznych działań. I tak np. w tym modelu pracownik zajmujący się wypłatami ma dostęp do bazy danych płac tylko wtedy, gdy korzysta z aplikacji płacowej.

W domenie bezpieczeństwa opartej na DAC pracownik działu płac będzie miał prawdopodobnie uprawnienia do zapisów oraz odczytu całej bazy danych, i te uprawnienia będą stałe oraz niezależne od aplikacji.

Systemy RBAC są dużo bardziej bezpieczne. Preferuje je większość ekspertów ds. bezpieczeństwa, ale również wymagają znaczco większego wysiłku po stronie administrowania, podejmowania decyzji o uprawnieniach, określania delegacji ról. W codziennym użytkowaniu RBAC wymaga większej automatyzacji i stosowania globalnych praktyk zarządzania.

W praktyce rzadko można spotkać domenę bezpieczeństwa, gdzie zastosowano tylko jeden model kontroli dostępu. System operacyjny Windows jest zbudowany wokół DAC, ale już w systemie Vista Microsoft dodał MAC (stosując obowiązkową kontrolę integralności), a RBAC jest dostępny na poziomie sieciowym w Active Directory. Wiele domen opartych na DAC i RBAC umożliwia także klasyfikowanie (etykietowanie) danych, podobnie jak systemy oparte na MAC, co pomaga ustalić właściwe uprawnienia i inne zabezpieczenia, które mają być stosowane na kolejnym poziomie ochrony ważnych danych.

Krótko o systemach rozliczeniowych i audycie

Aby zapewnić odpowiedni poziom bezpieczeństwa, niezbędne jest rejestrowanie pomyłnych lub podejmowanych prób dostępu, a także działań po uzyskaniu dostępu. Rozliczanie jest zazwyczaj uważane za bardziej ogólną metodę rejestracji niż audyt. System rozliczeniowy może rejestrować tylko pojedyncze pomyślne logowania (na sesję), liczbę przesłanych danych lub całkowity czas aktywności sesji. Z audytem wiąże się dużo bardziej szczegółowy poziom kontroli, pozwalający śledzić każdą wykonywaną przez SP akcję (lub próbę wykonania) - przeglądane lub modyfikowane pliki i foldery - oraz rejestrować czas wydarzenia.

Audyt i system rozliczania dodatkowo komplikują systemy współdzielone, tożsamości i hasła. Silny system AAA wymaga unikatowych tożsamości, aby każda indywidualna akcja mogła zostać zarejestrowana oddzielnie. Poziom rozliczalności i audytu może być ustalony przez administratora, aczkolwiek zależy częściowo od systemu AAA i używanych protokołów. Dobry system rozliczania i audytu śledzi każdą akcję wykonywaną przez każdego SP - od tworzenia obiektu aż do jego usunięcia (włączając w to zdarzenia logowania i zmiany w dzienniku zdarzeń audytu).

[Więcej o tym tutaj](#) (Strony 2-5)

50. Teoretyczne modele komputerów: automaty skończone, automaty ze stosem, maszyny Turinga i odpowiadające im klasy języków formalnych.

Ważne definicje

Alfabet

Alfabetem nazywamy dowolny niepusty zbiór skooczony. Elementy alfabetu nazywami symbolami.

Słowa

Słowem

I Wyrażenia regularne

Def

Niech będzie dany zbiór $\Pi = \{ \emptyset, \lambda, +, \cdot, ^*, (,) \}$ oraz alfabet Σ , przy czym $\Sigma \cap \Pi = \emptyset$.

Wyrażeniem regularnym nad alfabetem Σ nazywamy każde słowo $A \in (\Sigma, \Pi)^*$ spełniające jeden z poniższych warunków:

1. $A = \emptyset$
2. $A = \lambda$
3. $A = x \in \Sigma$
4. **A jest postaci $A = (B) + (C)$ lub $A = (B) \cdot (C)$ lub $A = (B)^*$, gdzie B, C są wyrażeniami regularnymi nad Σ**

Rodzinę wyrażeń regularnych nad alfabetem Σ oznaczamy przez **WR**(Σ)

(lub **WR** jeśli nie będzie wątpliwości dotyczących alfabetu)

II Języki regularne

Def

Niech będzie dany alfabet Σ oraz rodzina wyrażeń regularnych **WR** zdefiniowanych nad tym alfabetem. Każdemu wyrażeniu regularnemu $A \in \text{WR}$ przyporządkowujemy język $L(A)$ za pomocą definicji rekurencyjnej ze względu na budowę wyrażenia regularnego **A**:

$$L(A) = \begin{cases} \emptyset & \text{gdy } A = \emptyset \\ \{\lambda\} & \text{gdy } A = \lambda \\ \{x\} & \text{gdy } A = x \in \Sigma \\ L(B) + L(C) & \text{gdy } A = B + C, \text{ gdzie } B, C \in \text{WR} \\ L(B) \bullet L(C) & \text{gdy } A = B \bullet C, \text{ gdzie } B, C \in \text{WR} \\ (L(B))^* & \text{gdy } A = B^*, \text{ gdzie } B \in \text{WR} \end{cases}$$

Mówimy, że język $L(A)$ jest językiem generowanym przez wyrażenie regularne **A**, natomiast o słowach należących do języka $L(A)$ mówimy, że są generowane przez wyrażenie regularne **A**.

Def.

Językiem regularnym nazywamy każdy język formalny L nad danym alfabetem, dla którego istnieje wyrażenie regularne \mathbf{A} takie, że: $L=L(A)$. Klasę języków regularnych oznaczamy przez **JR**.

Każdy język regularny może być generowany przez wiele wyrażeń regularnych. Def. Mówimy, że wyrażenia regularne \mathbf{A} i \mathbf{B} są równoważne, gdy generują ten sam język, tzn

$$A \equiv B \iff L(A) = L(B)$$

AUTOMAT SKOŃCZONY Rabina-Scotta

Automatem skończonym (typu **Nas-lambda**) nazywamy uporządkowaną piątkę

$M = (Q, \Sigma, \delta, S_{start}, F)$, gdzie :
 Q jest skończonym zbiorem stanów,
 Σ jest alfabetem symboli wejściowych,
 $\delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow P(Q)$ jest funkcją przejścia ($\delta(s, a)$ to stan, do którego przechodzi automat ze stanu s po wczytaniu znaku a).
Funkcję przejście δ można przedstawić w tabeli lub za pomocą grafu.
 $s_{start} \in Q$ – stan początkowy automatu,
 $F \subset Q$ – zbiór stanów końcowych (automat przechodząc do tego stanu akceptuje dotychczas przeczytane słowo)

Język L złożony ze wszystkich słów akceptowanych przez automat skończony \mathbf{M} nazywamy generowanym przez automat \mathbf{M} i oznaczamy przez $L(M)$

$$L(M) = \{A \in \Sigma^* : \hat{\delta}(s_0, A) \cap F \neq \emptyset\}$$

Gdzie:

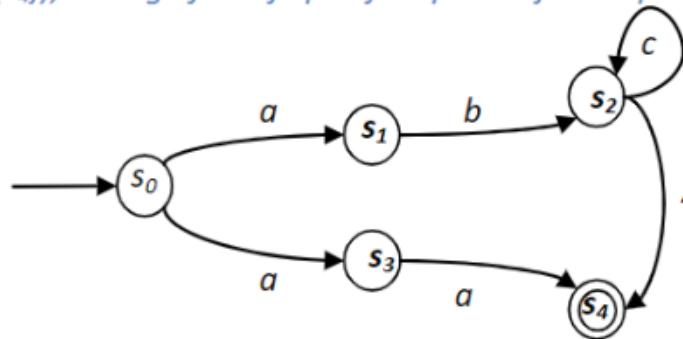
$\hat{\delta}$ – rozszerzona funkcja przejścia

Zbiór wszystkich języków generowanych przez automaty skończone oznaczamy symbolem **ZJNAS-λ**.

Przykład automatu skończonego:

NP. Dany jest automat $M = (\{s_0, s_1, s_2, s_3, s_4\}, \{a, b, c\}, \delta, s_0, \{s_4\})$, którego funkcja przejść opisana jest za pomocą tabeli i grafu:

δ	a	b	c	λ
s_0	{ s_1, s_3 }	\emptyset	\emptyset	\emptyset
s_1	\emptyset	{ s_2 }	\emptyset	\emptyset
s_2	\emptyset	\emptyset	{ s_2 }	{ s_4 }
s_3	{ s_4 }	\emptyset	\emptyset	\emptyset
s_4	\emptyset	\emptyset	\emptyset	\emptyset



Rozszerzona funkcja przejść:

$$\hat{\delta}(s_0, \lambda) = \{s_0\}$$

$$\hat{\delta}(s_2, \lambda) = \{s_2, s_4\}$$

$$\hat{\delta}(s_0, 1) = \{s_1, s_3\}$$

$$\hat{\delta}(s_1, b) = \{s_2, s_4\}$$

$$\hat{\delta}(s_1, bcccc) = \{s_2, s_4\}$$

$$\hat{\delta}(s_0, ac) = \emptyset$$

Akceptacja słów:

A=ab jest akceptowane przez automat $L(M)$ ponieważ

$$\hat{\delta}(s_0, ab) = \{s_2, s_4\} \quad i \quad \{s_2, s_4\} \cap F = \{s_2, s_4\} \cap \{s_4\} \neq \emptyset$$

B=a nie jest akceptowane przez automat $L(M)$ ponieważ

$$\hat{\delta}(s_0, a) = \{s_1, s_3\} \quad i \quad \{s_1, s_3\} \cap F = \{s_2, s_4\} \cap \{s_4\} = \emptyset$$

C=ac nie jest akceptowane przez automat $L(M)$ ponieważ

$$\hat{\delta}(s_0, ac) = \emptyset \quad i \quad \emptyset \cap F = \emptyset$$

Język akceptowany przez ten automat:

$$L(M) = \{aa, ab, abc, abcc, abccc, abcccc, \dots\}$$

Deterministyczny, zupełny automat skończony Rabina-Scotta

Automatem skończonym type **DAS** nazywamy uporządkowaną piątkę:

$M = (Q, \Sigma, \delta, S_{start}, F)$, gdzie :
 Q jest skończonym zbiorem stanów,
 Σ jest alfabetem symboli wejściowych,
 $\delta : Q \times \Sigma \rightarrow Q$ jest funkcją przejścia ($\delta(s, a)$ to stan,
do którego przechodzi automat ze stanu s po wczytaniu znaku a).
Funkcję przejść δ można przedstawić w tabeli lub za pomocą grafu.
 $s_{start} \in Q$ – stan początkowy automatu,
 $F \subset Q$ – zbiór stanów końcowych (automat przechodząc do tego stanu
akceptuje dotychczas przeczytane słowo)

Język Lzłożony ze wszystkich słów akceptowanych przez automat \mathbf{M} typu DAS nazywamy generowanym przez automat M oznaczamy przez $L(M)$

$$L(M) = \{A \in \Sigma^* : \hat{\delta}(s_0, A) \in F\}$$

Zbiór wszystkich języków generowanych przez automaty typu DAS oznaczamy symbolem **ZJDAS**.

TW. 3

Jeżeli

$$M = (Q, \Sigma, \delta, S_{start}, F)$$

jest automatem typu **DAS**, to generuje on język L wtedy i tylko wtedy, gdy automat

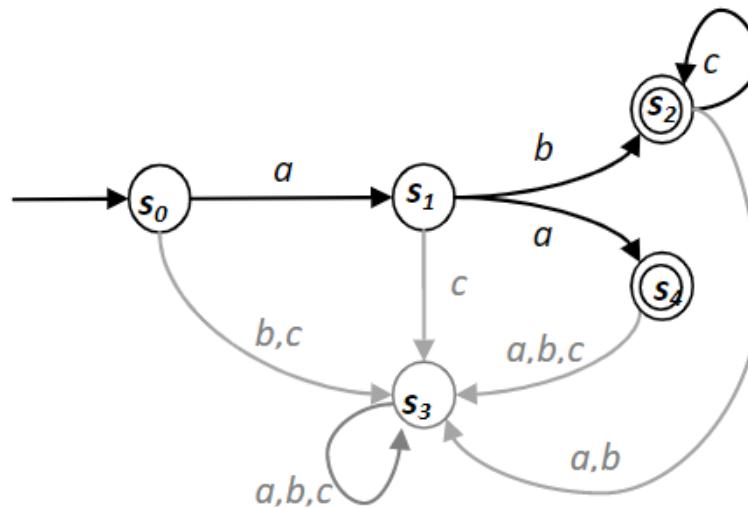
$$M' = (Q, \Sigma, \delta, S_{start}, Q \setminus F)$$

generuje język L' .

Przykład automatu typu DAS:

NP. Dany jest automat $M = (\{s_0, s_1, s_2, s_3, s_4\}, \{a, b, c\}, \delta, s_0, \{s_2, s_4\})$, którego funkcja przejść opisana jest za pomocą tabeli i grafu:

δ	a	b	c
s_0	s_1	s_3	s_3
s_1	s_4	s_2	s_3
s_2	s_3	s_3	s_2
s_3	s_3	s_3	s_3
s_4	s_3	s_3	s_3



Rozszerzona funkcja przejść:

$$\hat{\delta}(s_0, \lambda) = s_0$$

$$\hat{\delta}(s_2, \lambda) = s_2$$

$$\hat{\delta}(s_0, 3) = s_1$$

$$\hat{\delta}(s_1, 2) = s_2$$

$$\hat{\delta}(s_1, bc) = \delta(\delta(s_1, b), c) = \delta(s_2, c)$$

$$\hat{\delta}(s_0, acb) = \delta(\delta(\delta(s_0, a), c), b) = \delta(\delta(s_1, c), b) = \delta(s_3, b) = s_3$$

Akceptacja słów:

A=abcc jest akceptowane przez automat $L(M)$ ponieważ

$$\hat{\delta}(s_0, abcc) = \delta(\delta(\delta(\delta(s_0, a), b), c), c) = \delta(\delta(\delta(s_1, b), c), c) = \delta(\delta(s_2, c), c) = \delta(s_2, c) = s_2 \quad i \quad s_2 \in F$$

B=a nie jest akceptowane przez automat $L(M)$ ponieważ $\hat{\delta}(s_0, a) = s_1 \quad i \quad s_1 \notin F$

C=ac nie jest akceptowane przez automat $L(M)$ ponieważ $\hat{\delta}(s_0, ac) = s_3 \quad i \quad s_3 \notin F$

Język akceptowany przez ten automat to $L(M) = \{aa, ab, abc, abcc, abccc, \dots\}$.

Ważne Tw. Zbiory języków generowanych przez automaty typu DAS i NAS- λ są sobie równe:

$$ZJDAS = ZJNAS - \lambda$$

Def Gramatyką bezkontekstową nazywamy uporządkowaną czwórkę:

$G = (V, \Sigma, P, S)$, gdzie :

V jest skończonym zbiorem zmiennych (tzw. symboli nieterminalnych),
 Σ jest alfabetem gramatyki (tzw. zbiorem symboli nieterminalnych),
 P jest zbiorem par słów postaci (α, β) , gdzie $\alpha \in V, \beta \in (\Sigma \cup V)^*$,
zwanych produkcjami i zapisywanych w postaci $\alpha \Rightarrow \beta$
 S jest wyróżnioną zmienną początkową ze zbioru V
(tzw. głową gramatyki)

Def Językiem generowanym przez gramatykę \mathbf{G} , nazywamy zbiór

$$L = \{A \in \Sigma^* : S \hat{\Rightarrow} A\}.$$

Słowo **A** należy do języka **L** opisanego przez daną gramatykę **G**, jeśli istnieje ciąg produkcji prowadzący od symbolu zmiennej początkowej **S** do danego słowa. Mówimy wówczas, że słowo **A** jest wyprowadzone w gramatyce **G**.

Def

Językiem Bezkontekstowym nazywamy język, dla którego istnieje gramatyka bezkontekstowa generująca ten język. Zbiór języków generowanych przez gramatyki bezkontekstowe oznaczamy przez **ZJB**.

Def

Automatem ze Stosem(AZS) nazywamy uporządkowaną siódemkę:

$M = (Q, \Sigma, \Gamma, \delta, S_{start}, \perp, F)$, gdzie :

Q jest skończonym zbiorem stanów,
 Σ jest alfabetem symboli wejściowych (terminalne),
 Γ jest alfabetem stosu
 $\delta : Q \times (\Sigma \cup \lambda) \times \Gamma \rightarrow P(Q \times \Gamma)^*$ jest funkcją przejścia
 $S_{start} \in Q$ – stan początkowy stosu,
 $\perp \in \Gamma$ – stan początkowy stosu,
 $F \subset Q$ – zbiór stanów końcowych automatu.

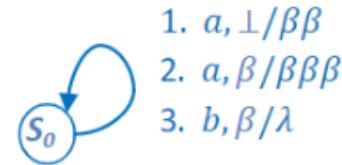
Opisem chwilowym automatu **M** nazywamy każdą trójkę (S, A, B) , gdzie:

$S \in Q$ – jest stanem,
 $A \in \Sigma^*$ – jest słowem do wczytania pozostającym na taśmie,
 $\beta \in \Gamma^*$ jest słowem znajdującym się na stosie.

Przykładowy auyomat ze Stosem:

NP. Dany jest automat ze stosem $M=(Q, \Sigma, \Gamma, \delta, s_{\text{start}}, \perp, F) = (\{S_0\}, \{a,b\}, \{\perp, \beta\}, \delta, S, \perp, \emptyset)$, gdzie δ jest opisana za pomocą następujących przejść:

1. $(S_0, a, \perp) \rightarrow (S_0, \beta\beta)$
2. $(S_0, a, \beta) \rightarrow (S_0, \beta\beta\beta)$
3. $(S_0, b, \beta) \rightarrow (S_0, \lambda)$



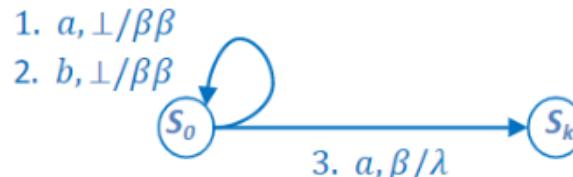
Rozważmy słowo $A=aabbba$ i kolejne opisy chwilowe:

$$(S_0, aabbba, \perp) \xrightarrow{1} (S_0, abbb, \beta\beta) \xrightarrow{2} (S_0, bbbb, \beta\beta\beta\beta) \xrightarrow{3} (S_0, bbb, \beta\beta\beta) \xrightarrow{3} (S_0, bb, \beta\beta) \xrightarrow{3} (S_0, b, \beta) \xrightarrow{3} (S_0, \lambda, \lambda)$$

Całe słowo A z taśmy zostało wczytane a stos został opróżniony. Powiemy w takiej sytuacji, że słowo A jest akceptowane przez automat M poprzez pusty stos.

NP. Dany jest automat ze stosem $M=(Q, \Sigma, \Gamma, \delta, s_{\text{start}}, \perp, F) = (\{S_0, S_k\}, \{a,b\}, \{\perp, \beta\}, \delta, S_0, \perp, \{S_0, S_k\})$, gdzie δ jest opisana za pomocą następujących przejść:

1. $(S_0, a, \perp) \rightarrow (S_0, \beta\beta)$
2. $(S_0, b, \perp) \rightarrow (S_0, \beta\beta)$
3. $(S_0, a, \beta) \rightarrow (S_k, \lambda)$



Rozważmy słowo $A=ba$ i kolejne opisy chwilowe:

$$(S_0, ba, \perp) \xrightarrow{2} (S_0, a, \beta\beta) \xrightarrow{3} (S_k, \lambda, \beta)$$

Całe słowo A z taśmy zostało wczytane a automat znalazł się w jednym ze stanów końcowych. Powiemy w takiej sytuacji, że słowo A jest akceptowane przez automat M poprzez stan końcowy (stos może nie być opróżniony).

Tw. 17

Zbiór języków akceptowanych przez automaty ze stosem jest równy zbiorowi języków bezkontekstowych.

Maszyny Turinga

Def. Maszyną Turinga (MT) nazywamy uporządkowany układ:

$M = (Q, \Sigma, \Gamma, \delta, s_{start}, \square, F)$, gdzie :
 Q jest skończonym zbiorem stanów,
 Σ jest alfabetem maszyny
 Γ jest alfabetem taśmy, przy czym $\Gamma \supset \Sigma$
 $\delta : D \rightarrow P(Q \times \Gamma \times \{L, R\})$ (gdzie $D \subset Q \times \Gamma$) jest funkcja
przejść maszyny
 $s_{(start)} \in Q$ jest stanem startowym maszyny,
 $\square \in \Gamma$ jest symbolem pustej komórki (poza skońzoną ilością komórek
wszystkie pozostałe komórki nieskończonej taśmy zawierają symbol pustej
komórki),
 $F \subset Q$ jest zbiorem stanów końcowych maszyny.

Def

Język **L** złożony ze wszystkich słów akceptowanych przez maszynę Turinga **M** nazywamy językiem **generowanym przez maszynę M** i oznaczamy przez $L(M)$:

$$L(M) = \{A \in \Sigma^* : s_{start} A \Rightarrow \dots \Rightarrow \Gamma_1 s \Gamma_2 \in \Gamma * i \text{ stanu } s \in F\}$$

Zbiór wszystkich języków generowanych przez maszyny Turinga oznaczamy symbolem **ZJMT** i nazywamy **rekurencyjnie przeliczalnymi**.

UWAGA

Jeżeli $\delta : D \rightarrow Q \times \Gamma \times \{L, R\}$ (gdzie $D \subset Q \times \Gamma$) to Maszyna Turinga
jest maszyną deterministyczną.

UWAGA

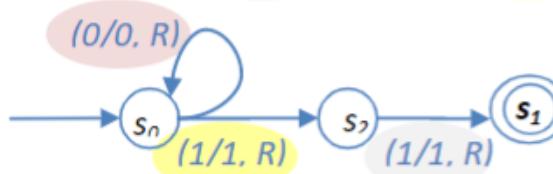
Maszyny Turinga można przedstawić m.in. za pomocą:

- tabeli,
- grafu,
- kodu maszyny Turinga (o ile: $E=\{0,1\}$, przekształcimy maszynę do postaci maszyny Turinga o jednym stanie końcowym s_1 i ustalimy kolejność kodowania ruchów maszyny Turinga).

Przykład Maszyny Turinga

NP. Dana jest maszyna Turinga $M=(Q, \Sigma, \Gamma, \delta, s_0, \square, F) = (\{s_0, s_1, s_2\}, \{0, 1\}, \{\square, a, b\}, \delta, s_0, \{s_1\})$, gdzie δ jest opisana równoważnie za pomocą tabeli, grafu i kodu maszyny Turinga: 0101010100110100100010010011000100100100100

δ	0	1	\square
s_0	$(s_0, 0, R)$	$(s_2, 1, R)$	
s_2		$(s_1, 1, R)$	



Rozważmy słowo $A=00110010111$ i kolejne opisy chwilowe:

$$(s_0 00110010111) \xrightarrow{1} (0s_0 0110010111) \xrightarrow{1} (00s_0 110010111) \xrightarrow{2} (001s_2 10010111) \xrightarrow{3} (0011s_1 0010111)$$

Słowo nie zostało wczytane z taśmy ale głowica przeszła od stanu początkowego gdy czytany był pierwszy symbol słowa do stanu końcowego s_1 . Zatem słowo $A=00110010111$ jest akceptowane przez tą maszynę Turinga.

Podobnie akceptowane są słowa 11, 110, 011. Ogólnie $L(M)=L(0^*11(0+1)^*)$

Przykład MT akceptującej słowa i MT obliczającej:

NP.

δ	0	1	X	\square
s_0	$(s_0, 0, R)$	$(s_1, 1, R)$		
s_1	$(s_1, 0, R)$	$(s_2, 1, R)$		
s_2	(s_3, x, L)	$(s_2, 1, R)$	(s_2, x, R)	(s_4, \square, L)
s_3	(s_2, x, L)	$(s_3, 1, L)$	(s_3, x, L)	
s_4		$(s_4, 1, L)$	(s_4, x, L)	(s_k, \square, R)

Maszyna Turinga akceptująca słowa
języka DODAWANIE = $\{0^i 10^j 10^{i+j} : i, j \in \mathbb{N}\}$

δ	0	1	\square
s_0	$(s_0, 0, R)$	$(s_0, 0, R)$	(s_1, \square, L)
s_1	(s_k, \square, R)		

Maszyna Turinga obliczająca sumę $i+j$
po wprowadzeniu na taśmę słowa $0^i 10^j$

(na taśmie pozostałe wyłącznie $i+j$ zer)

MT

Języki

MT	Języki
Maszyny Turinga	
Niedeterministyczne maszyny Turinga	ZJRP
Wielotaśmowe maszyny Turinga	zbiór języków
Uniwersalna maszyna Turinga	rekursywnie przeliczalnych (rekurencyjnie przeliczalnych)
Właściwe MT - maszyny Turinga zatrzymująca się dla każdego słowa po skończonej ilości ruchów	ZJRK
MT (automaty) liniowo ograniczone – maszyny Turinga w których $<,> \in \Gamma$ i głowica przesuwa się tylko między symbolami $< i >$ wyznaczającymi początek i koniec słowa.	Zbiór języków rekursywnych (rekurencyjnych)
	ZJK
	Zbiór języków kontekstowych.

Tw.

Istnieje język formalny, który nie jest rekursywnie przeliczalnym, tzn. nie jest akceptowany przez żadną MT.