Spis treści

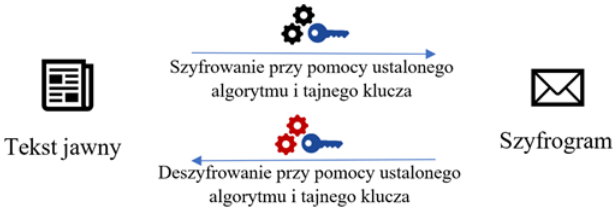
Rozdział I

Pytania - dr. hab. Bogdan Księżopolski

66 Charakterystyka kryptografii symetrycznej oraz asymetrycznej.

Kryptografia symetryczna

W kryptografii symetrycznej szyfrowanie i deszyfrowanie wykonywane jest przy użyciu tego samego klucza. W niektórych algorytmach wykorzystywane są dwa klucze, jednak muszą one być od siebie zależne w taki sposób, że znając jeden z nich, można wygenerować drugi.



Rysunek I.1: Zasada działania kryptografii symetrycznej

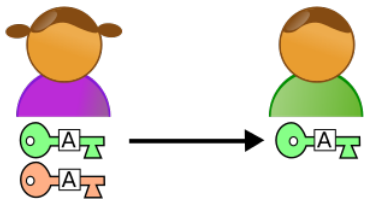
W celu zapewnienia bezpiecznej komunikacji, algorytm szyfrowania musi być tak skonstruowany, żeby odtworzenie tekstu jawnego bez znajomości klucza było zadaniem trudnym obliczeniowo. Dodatkowym wymaganiem jest tajność klucza – przed rozpoczęciem wymiany wiadomości, należy opracować protokół uzgadniania lub przekazywania klucza.

Algorytmy szyfrowania symetrycznego możemy podzielić na algorytmy blokowe i strumieniowe. Pierwsze z nich przekształcają blok danych ustalonej długości, traktując go jako całość, na szyfrogram o tej samej liczbie bitów. Szyfry strumieniowe przyjmują natomiast ciąg (strumień) danych. Algorytmy kryptografii symetrycznej są szybkie, zwykle wymagają też mniejszej mocy obliczeniowej niż algorytmy asymetryczne. Powszechnie stosowanym szyfrem symetrycznych jest **AES**.

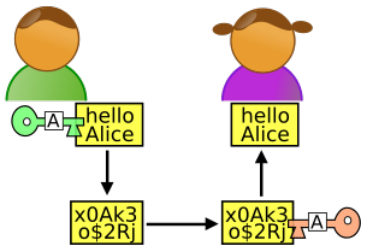
Kryptografia asymetryczna

Kryptografia asymetryczna to rodzaj kryptografii, w którym jeden z używanych kluczy jest udostępniony publicznie. Każdy użytkownik może użyć tego klucza do zaszyfrowania wiadomości, ale tylko posiadacz drugiego, tajnego klucza może odszyfrować taką wiadomość.

Kryptografia asymetryczna opiera się na funkcjach jednokierunkowych – takich, które da się łatwo wyliczyć w jedną stronę, ale bardzo trudno w drugą. Np. mnożenie jest łatwe, a rozkład na czynniki (z ang. faktoryzacja) trudny (na czym przykładowo opiera się **RSA**). Potęgowanie modulo jest łatwe, a logarytmowanie dyskretne jest trudne (na czym opierają się ElGamal, DSA i **ECC**).



Rysunek I.2: Krok 1: Alice przesyła do Boba swój klucz publiczny



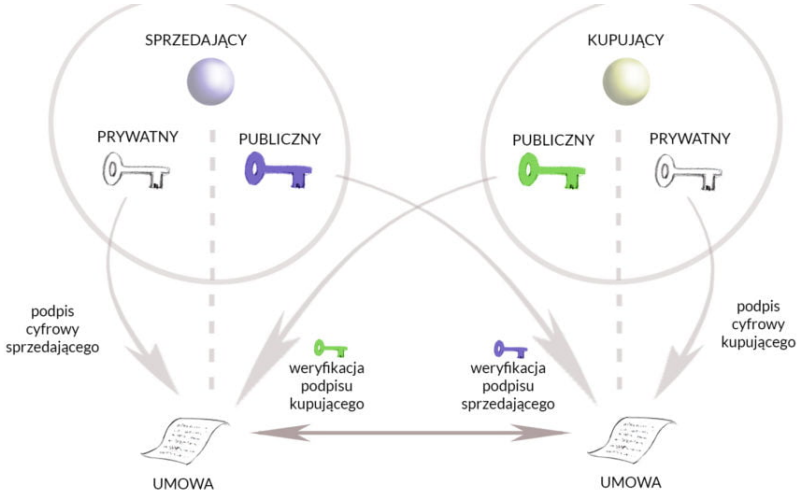
Rysunek I.3: Kroki 2 i 3: Bob szyfruje wiadomość kluczem publicznym Alice, która to następnie otrzymuje zaszyfrowaną wiadomość i rozszyfrowuje ją kluczem prywatnym

Klucz publiczny używany jest do zaszyfrowania informacji, klucz prywat-

ny do jej odczytu. Ponieważ klucz prywatny jest w wyłącznym posiadaniu adresata informacji, tylko on może ją odczytać. Natomiast klucz publiczny jest udostępniony każdemu, kto zechce zaszyfrować wiadomość.

Ponieważ kryptografia asymetryczna jest o wiele wolniejsza od symetrycznej, prawie nigdy nie szyfruje się wiadomości za pomocą kryptosystemów asymetrycznych (również ze względu na ograniczenie wielkości szyfrowanej wiadomości). Zamiast tego szyfruje się jedynie klucz jakiegoś szyfru symetrycznego, takiego jak np. AES. Takie protokoły, łączące elementy kryptografii symetrycznej i asymetrycznej, nazywa się hybrydowymi.

Nadawcy mogą także używać kluczy prywatnych do cyfrowego podpisywania wiadomości. Te podpisy cyfrowe pozwalają odbiorcom uwierzytelnić tożsamość nadawcy i spać spokojnie, wiedząc, że wiadomości nie zostały zmienione od momentu podpisania. W takim przypadku przesyłane informacje mogą być publiczne, a odbiorca może użyć certyfikatu, który towarzyszy tej informacji, aby zweryfikować integralność i autentyczność podpisanej wiadomości.



Rysunek I.4: Jak działa podpis

2 Funkcje skrótu (mieszające) i ich zastosowania.

Funkcja skrótu, funkcja mieszająca lub funkcja haszująca – funkcja przyporządkowująca dowolnie dużej liczbie krótką wartość o stałym rozmiarze,

tzw. skrót nieodwracalny.

W informatyce funkcje skrótu pozwalają na ustalenie krótkich i łatwych do weryfikacji sygnatur dla dowolnie dużych zbiorów danych. Sygnatury mogą chronić przed przypadkowymi lub celowo wprowadzonymi modyfikacjami danych (sumy kontrolne), a także mają zastosowania przy optymalizacji dostępu do struktur danych w programach komputerowych (tablice mieszające).

Szczególną podgrupą funkcji skrótu są funkcje uznawane za bezpieczne do zastosowań kryptologicznych (jak np. SHA-3). Kryptograficzna funkcja skrótu powinna spełniać kombinację następujących kryteriów, w zależności od zastosowania:

- Odporność na kolizje (collision resistance) – brak praktycznej możliwości wygenerowania dwóch dowolnych wiadomości o takim samym skrócie
- Odporność na kolizje konkretnych wiadomości (target collision-resistance, preimage resistance) pierwszego i drugiego rzędu – brak praktycznej możliwości wygenerowania wiadomości o takim samym skrócie jak wskazana wiadomość
- Jednokierunkowość (one-wayness) – brak możliwości wnioskowania o wiadomości wejściowej na podstawie wartości skrótu. Zmiana dowolnego pojedynczego bitu wiadomości powinna zmieniać średnio połowę bitów skrótu w sposób, który nie jest istotnie podatny na kryptoanalizę różnicową.

Przykładowe funkcje skrótu to SHA-1 (SHA128), SHA-2 (SHA256), SHA-3(SHA512), MD5.

9 TODO: Protokoły TCP i UDP – porównanie i zastosowanie.

Lorem ipsum dupa dupa

10 TODO: Adresowanie w warstwie Internetu modelu TCP/IP.

Lorem ipsum dupa dupa

12 TODO: Porównanie modelu OSI i TCP/IP.

Lorem ipsum dupa dupa

13 TODO: Mechanizm enkapsulacji w modelu OSI.

Lorem ipsum dupa dupa

58 TODO: Mechanizm sesji w zarządzaniu stanem aplikacji sieciowej.

Lorem ipsum dupa dupa

59 TODO: Mechanizm gniazd – pojęcie, sposób realizacji i zastosowanie

Lorem ipsum dupa dupa

60 TODO: Metody obsługi wielu klientów równolegle w aplikacjach sieciowych.

Lorem ipsum dupa dupa

61 TODO: Pocztowe protokoły warstwy aplikacji.

Lorem ipsum dupa dupa

62 TODO: Porównanie HTTP i WebSocket.

Lorem ipsum dupa dupa

63 TODO: Atrybuty bezpieczeństwa informacji.

Lorem ipsum dupa dupa

64 TODO: Modele dystrybucji kluczy kryptograficznych.

Lorem ipsum dupa dupa

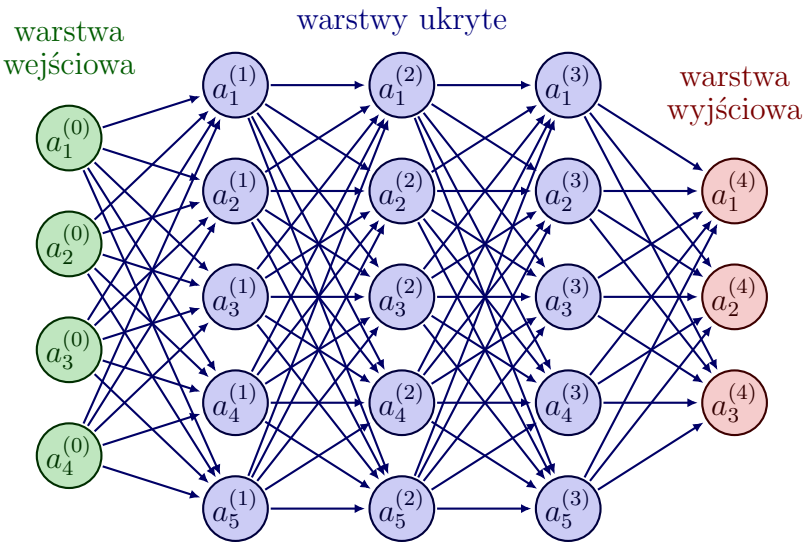
65 TODO: Rodzaje zagrożeń oraz ochrona aplikacji sieciowych.

Lorem ipsum dupa dupa

Rozdział II

Pytania - dr. hab. Grzegorz Wójcik

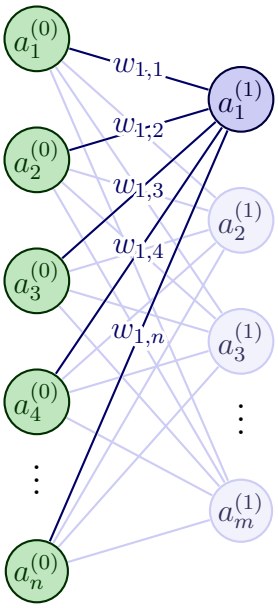
33 Budowa sieci neuronowych



Sieci neuronowe, znane również jako sztuczne sieci neuronowe lub symulowane sieci neuronowe są częścią funkcji uczenia maszynowego i stanowią podstawę algorytmów uczenia głębokiego. Ich nazwa i struktura są wzorowane na ludzkim mózgu i naśladują sposób, w jaki biologiczne neurony komunikują się między sobą.

Sztuczne sieci neuronowe składają się z warstw węzłów, obejmujących warstwę wejściową, jedną lub więcej warstw ukrytych oraz warstwę wyjściową. Każdy węzeł (sztuczny neuron) łączy się z innym i ma powiązaną wagę

oraz próg. Jeśli wyjście dowolnego pojedynczego węzła przekracza określoną wartość progową, węzeł ten jest aktywowany podczas wysyłania danych do kolejnej warstwy sieci. W przeciwnym razie żadne dane nie są przekazywane do następnej warstwy sieci.



$$= \sigma\left(w_{1,0}a_0^{(0)} + w_{1,1}a_1^{(0)} + \ldots + w_{1,n}a_n^{(0)} + b_1^{(0)}\right)$$

$$= \sigma\left(\sum_{i=1}^{n} w_{1,i}a_i^{(0)} + b_1^{(0)}\right)$$

$$\begin{pmatrix} a_1^{(1)} \\ a_2^{(1)} \\ \vdots \\ a_m^{(1)} \end{pmatrix} = \sigma\left[\begin{pmatrix} w_{1,0} & w_{1,1} & \ldots & w_{1,n} \\ w_{2,0} & w_{2,1} & \ldots & w_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ w_{m,0} & w_{m,1} & \ldots & w_{m,n} \end{pmatrix}\begin{pmatrix} a_1^{(0)} \\ a_2^{(0)} \\ \vdots \\ a_n^{(0)} \end{pmatrix} + \begin{pmatrix} b_1^{(0)} \\ b_2^{(0)} \\ \vdots \\ b_m^{(0)} \end{pmatrix}\right]$$

$$a^{(1)} = \sigma\left(\mathbf{W}^{(0)}a^{(0)} + \mathbf{b}^{(0)}\right)$$

O każdym pojedynczym węźle należy myśleć jak o modelu regresji liniowej złożonym z danych wejściowych, wag, odchyleń (lub wartości progowych) i danych wyjściowych. Rysunek powyżej przedstawia właśnie te obliczenia dla jednego neurona. Macierz $\mathbf{W}^{(0)}$ jest macierzą wszystkich wag wchodzących do każdego neurona warstwy $a^{(0)}$ z warstwy poprzedniej, wektor $b^(0)$ to wartości wszystkich *bias-ów* danej warstwy, a σ to funkcja aktywacji danej warstwy.

Wagi pomagają określić znaczenie każdej zmiennej, przy czym większe z nich mają większy wpływ na wynik wyjściowy w porównaniu do innych danych wejściowych. Wszystkie dane wejściowe są następnie mnożone przez swoje odpowiednie wagi, a potem sumowane. Następnie wyniki są przepuszczane przez funkcję aktywacji, która określa wartość wyjściową.

30 TODO: Modele reprezentacji wiedzy.

Lorem ipsum dupa dupa

31 TODO: Mechanizmy wnioskowań.

Lorem ipsum dupa dupa

32 TODO: Metody uczenia maszynowego.

Lorem ipsum dupa dupa

34 TODO: Normalizacja baz danych – pierwsza, druga i trzecia postać normalna.

Lorem ipsum dupa dupa

35 Paweł TODO: Modele baz danych (logiczny, relacyjny, fizyczny).

Lorem ipsum dupa dupa

36 Paweł TODO: Rodzaje zapytań w języku SQL.

Lorem ipsum dupa dupa

37 Paweł TODO: Funkcje w języku SQL.

Lorem ipsum dupa dupa

38 Paweł TODO: Transakcje w bazach danych.

Lorem ipsum dupa dupa

15 TODO: Hermetyzacja, dziedziczenie i polimorfizm w programowaniu obiektowym.

Lorem ipsum dupa dupa

48 TODO: Główne paradygmaty programowania – charakterystyka i przykłady.

Lorem ipsum dupa dupa

17 TODO: Paradygmat i przykłady programowania generycznego (rodzajowego).

Lorem ipsum dupa dupa

Rozdział III

Pytania - reszta

1 Wektory i macierze – definicje i podstawowe operacje.

Macierz to układ liczb, symboli lub wyrażeń zapisanych w postaci prostokątnej tablicy. W algebrze liniowej macierze wprowadza się często jako sposób skondensowanego zapisu układów równań liniowych, co ma na celu wyeliminowanie powtarzających się elementów standardowej notacji układów równań tego rodzaju z wieloma niewiadomymi. Macierze pozwalają również na reprezentowanie przekształceń liniowych w sposób umożliwiający przeprowadzanie obliczeń. Ponieważ wiele przekształceń geometrycznych (jak na przykład obroty przestrzeni $\mathbb{R}^n$ wokół początku układu współrzędnych) są przekształceniami liniowymi, macierze znajdują zastosowanie w geometrii analitycznej i grafice komputerowej.

Przykład zapisu macierzy 3×3 $\begin{bmatrix} 8 & 3 & 2 \\ 5 & 4 & 1 \\ 2 & 9 & 0 \end{bmatrix}$

Macierze $\mathbf{A} = [a_{ij}]$ i $\mathbf{B} = [b_{ij}]$ uważa się za równe, jeśli mają ten sam typ i równe odpowiadające sobie elementy, tzn. dla każdej możliwej pary i, j zachodzi $a_{ij} = b_{ij}$.

Sumę macierzy $\mathbf{A}$ i $\mathbf{B}$ definiuje się „po współczynnikach", tzn. za pomocą wzoru $\mathbf{A} + \mathbf{B} = [a_{ij} + b_{ij}]$ dla wszystkich i, j. Z definicji wynika (ale można napisać wprost), że można dodawać macierzy tylko o takich samych wymiarach.

$$\begin{bmatrix} 8 & 3 & 2 \\ 5 & 4 & 1 \\ 2 & 9 & 0 \end{bmatrix} + \begin{bmatrix} 2 & 2 & 6 \\ 3 & 5 & 7 \\ 1 & 0 & 4 \end{bmatrix} = \begin{bmatrix} 10 & 5 & 8 \\ 8 & 10 & 8 \\ 3 & 9 & 4 \end{bmatrix}$$

Mnożenie przez skalar macierzy **A** oraz liczby c również definiuje się „po współczynnikach", czyli $c\mathbf{A} = [ca_{ij}]$ dla dowolnych i, j.

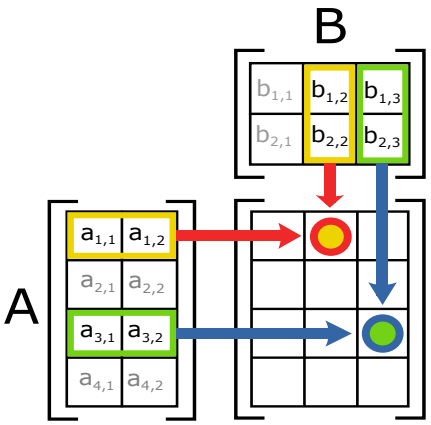
$$2 * \begin{bmatrix} 8 & 3 & 2 \\ 5 & 4 & 1 \\ 2 & 9 & 0 \end{bmatrix} = \begin{bmatrix} 16 & 6 & 4 \\ 10 & 8 & 2 \\ 4 & 18 & 0 \end{bmatrix}$$

Działanie mnożenia macierzy jest zdefiniowane najczęściej jako tzw. iloczyn Cauchy'ego: dla dla macierzy **A** typu $m \times n$ oraz **B** typu $n \times p$ dany jest on jako taka macierz **C** typu $m \times p$, oznaczana **AB**, dla której

$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \cdots + a_{in}b_{nj}$ dla dowolnych i, j.

Mnożenie to jest łączne $(A(BC) = (AB)C)$, ale nie jest przemienne $(AB \neq BA)$.

$$\begin{bmatrix} 2 & 3 & 7 \\ 6 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \\ 5 \end{bmatrix} = \begin{bmatrix} 2*1+3*0+7*5 \\ 6*1+1*0+2*5 \end{bmatrix} = \begin{bmatrix} 37 \\ 16 \end{bmatrix}$$



Rysunek III.1: Schemat mnożenia macierzy A i B

Elementem neutralnym mnożenia macierzy przez siebie jest macierz diagonalna, zawierająca na swojej przekątnej same jedynki.

$$\begin{bmatrix} 2 & 3 & 7 \\ 6 & 1 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 3 & 7 \\ 6 & 1 & 2 \end{bmatrix}$$

Przestawienie bądź transpozycja danej macierzy $\mathbf{A}$, tzn. zamiana jej kolumn i wierszy miejscami (z zachowaniem kolejności). Macierz transponowaną lub przestawioną względem macierzy $\mathbf{A}$ definiuje się jako macierz

$\mathbf{A}^{\mathrm{T}} = [a_{ji}]$ dla wszystkich i, j, przy czym $(\mathbf{AB})^{\mathrm{T}} = \mathbf{B}^{\mathrm{T}}\mathbf{A}^{\mathrm{T}}$ oraz $\left(\mathbf{A}^{\mathrm{T}}\right)^{\mathrm{T}} = \mathbf{A}$.

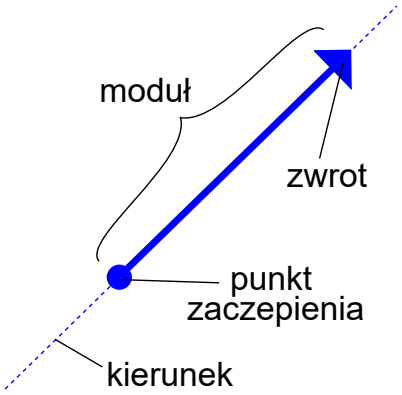
Wyznacznikiem $\det(\mathbf{A})$ lub $|\mathbf{A}|$ macierzy kwadratowej $\mathbf{A}$ nazywa się liczbę kodującą pewne właściwości przekształcenia A reprezentowanego przez tę macierz.

Wyznacznik macierzy stopnia drugiego dany jest wzorem

$$\det \begin{bmatrix} a & b \\ c & d \end{bmatrix} = ad - bc.$$

Wektor jest macierzą o wymiarach $n \times 1$. Reprezentuje on punkt w przestrzeni $\mathbb{R}^n$. Jego podstawowe trzy cechy to:

- długość - czasami inaczej zwana modułem lub wartością
- kierunek - kierunek prostej zawierającej wektor
- zwrot - grot strzałki



Rysunek III.2: Ilustracja wektora

Dodawanie oraz mnożenie przez skalar wektora jest zdefiniowane w ten sam sposób jak w przypadku macierzy.

Iloczyn skalarny dwóch wektorów to **liczba**, którą obliczamy dodając iloczyny odpowiednich współrzędnych.

$$\vec{a} = [2, 1, 3], \vec{b} = [4, 1, 2]$$
$$\vec{a} \circ \vec{b} = 2 * 4 + 1 * 1 + 3 * 2 = 15$$

Iloczyn skalarny możemy również obliczyć znając długości wektorów $|\vec{a}|$ i $|\vec{b}|$ oraz kąt α między nimi:

$$\vec{a} \circ \vec{b} = \|\vec{a}\| * \|\vec{b}\| * \cos\alpha$$

Długość wektora $\vec{a}$ może być zdefiniowana jako pierwiastek iloczynu skalarnego z samym sobą.

$$\|\vec{a}\| = \sqrt{\vec{a} \circ \vec{a}}$$

Iloczyn wektorowy - działanie dwuargumentowe przyporządkowujące parze wektorów przestrzeni $\mathbb{R}^3$ pewien wektor tej przestrzeni.

Iloczyn wektorowy $\mathbf{a} \times \mathbf{b}$ wektorów **a** i **b** określa się następująco:

- jeśli wektory **a** i **b** są liniowo zależne, to $\mathbf{a} \times \mathbf{b} = 0$
- jeśli wektory **a** i **b** nie są liniowo zależne, to $\mathbf{a} \times \mathbf{b} = \mathbf{c}$, gdzie **c** jest wektorem prostopadłym do płaszczyzny wyznaczonej przez **a** i **b**.

6 Sposoby cyfrowej reprezentacji liczby całkowitej i rzeczywistej.

Liczby całkowite

Kod ZM (kod znak-moduł) Sprawa w kodzie ZM jest w miarę prosta i klarowna. Najstarszy bit b_{n-1} dla n-bitowej liczby jest bitem znaku i określa czy liczba jest dodatnia czy ujemna:

- 0 - liczba dodatnia,
- 1 - liczba ujemna.

Bity od b_{n-1} do b_0 odpowiadają za kodowanie wartości samej liczby. Wzór na obliczenie wartości liczby zakodowanej w **ZM**:

$$L_{ZM} = (-1)^{b_{n-1}} \cdot (b_{n-2}2^{n-2} + ... + b_2 2^2 + b_1 2^1 + b_0 2^0)$$

Przykładowe kodowanie liczby na ośmiu bitach w kodzie **ZM**:

$$26 \longrightarrow \mathbf{0}0011010$$
$$-26 \longrightarrow \mathbf{1}0011010$$

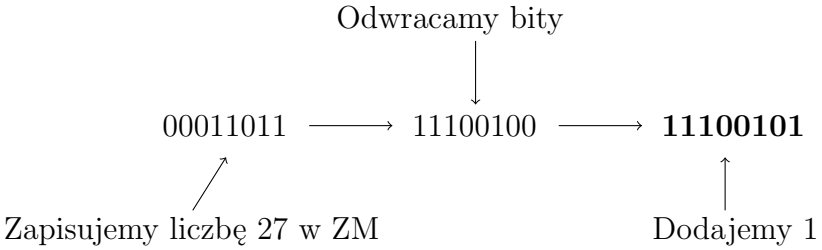
Proste, logiczne, fajne. Pytania, problemy? To jedziemy dalej.

Kod U2 (kod uzupełnień do 2) Tutaj sprawa się nieco komplikuje z zapisem liczb ujemnych. Bit b_{n-1} ma wagę -2^{n-1} co sprawia, że musimy bitowo tak jakby zapisać odwrotność liczby, którą chcemy reprezentować jako ujemna (i dodać 1, żeby się wszystko zgadzało). W zapisie liczb dodatnich zapis jest identyczny jak w **ZM** - na najstarszym bicie musimy tylko zachować 0.

Istnieje prosty algorytm konwersji na U2 z wykorzystaniem ZM:

1. Zapisać moduł liczby w ZM,
2. Dokonać inwersji bitów (0 na 1 i 1 na 0),
3. Zwiększ wynik dodając 1.

Przykład z liczbą -27 na 8 bitach:



Liczby rzeczywiste

Zapis stałopozycyjny Do zapisu liczby stałoprzecinkowej przeznaczona jest z góry określona liczba bitów, a pozycję przecinka ustala się arbitralnie, w zależności od wymaganej dokładności, wolne bity uzupełniając zerami. Do reprezentacji liczb ze znakiem stosuje także kod U2.

Liczba $6,25 = 110,01_{(2)}$ zapisana na 8 bitach gdy częśd ułamkowa zajmuje 3 najmłodsze bity, ma postać:

$$\underbrace{00110}_{\text{część całkowita}}\overbrace{010}^{\text{część ułamkowa}}$$

A w reprezentacji U2 będzie miała postać:

11001110

Część całkowita liczby zachowuje się identycznie jak w przypadku zwykłych liczb całkowitych, natomiast bity w części ułamkowej posiadają wagi 2^{-1}, 2^{-2}, itd. - czyli $\frac{1}{2}$, $\frac{1}{4}$, ..., więc ilość bitów w części ułamkowej wpływa na precyzję zapisu.

Zapis zmiennopozycyjny Liczba zmiennoprzecinkowa jest komputerową reprezentacją liczb rzeczywistych zapisanych w postaci wykładniczej o podstawie 2. Przykładowa notacja:

$$(-1)^Z \cdot M \cdot 2^C = (-1)^Z \cdot (1+m) \cdot 2^{c-BIAS}$$

gdzie:

$(-1)^Z$ - znak liczby

$M = 1 + m$ - znormalizowana mantysa (liczba spełniająca warunek: $1 \leqslant M \leqslant 2$). Ponieważ przed przecinkiem stoi zawsze 1, więc można ją przedstawić w postaci $1+m$, gdzie m jest liczbę ułamkową: $0 \leqslant m \leqslant 1$)

$C = c - BIAS$ - cecha (liczba całkowita), która dzięki zastosowaniu stałej BIAS pozwoli przedstawid cechę w postaci różnicy c-BIAS (c jest liczbą całkowitą dodatnią, tzw spolaryzowana cechę)

$BIAS$ - stała (liczba całkowita BIAS zależna od danej implementacji – rozwiązuje problem znaku cechy)

Kodujemy wyłącznie:

z - bit znaku

m - mantysę pomniejszoną o 1

c - cechę przesuniętą o BIAS

Załóżmy, że operujemy następującym zmiennopozycyjnym formatem zapisu liczby rzeczywistej:

- na zapis przeznaczamy 16 bitów
- najstarszy bit (b_{15}) to bit znaku (będziemy stosowad kod ZM)

- kolejne 6 bitów (b_9-b_{14}) to mantysa
- pozostałe bity (b_0-b_8) są przeznaczone na zapis cechy i przyjmijmy, że BIAS=9

Przedstawimy liczbę +0,0224609375 w powyższym formacie. Naszą liczbę zapisujemy w systemie binarnym w postaci wykładniczej o podstawie 2, przesuwamy przecinek zapisując ją w notacji wykładniczej:

$$0,0224609375 = 0,0000010111_{(2)} = 1,0111_{(2)} \cdot 2^{-6}$$

Z tego wynika, że:

- Znak: $(-1)^0$
- Mantysa: $1.\underline{\mathbf{0111}}_2$
- Cecha: $-6 = 3 - 9 = 11_2 - BIAS$

Oto liczba 0,0224609375 zapisana w zadanym formacie:

$$\underbrace{0}_{\text{bit znaku}}\overbrace{0111100}^{\text{mantysa}}\underbrace{000000011}_{\text{cecha}}$$

53 Deklaratywne programowanie w logice: klauzule Horne'a, nawracanie.

Logika Hoare'a – formalizm matematyczny służący do opisu poprawności algorytmów. Trójka $\{P\}C\{Q\}$ oznacza, że fragment kodu C, o ile na wejściu będzie miał stan spełniający warunek P, oraz zakończy swoje działanie, to na wyjściu da stan spełniający warunek Q. Formułę P nazywamy warunkiem wstępnym, a formułę Q nazywamy warunkiem końcowym.

Przykład: do instrukcji przypisania $x := 5$ możemy dopisać następujące warunki wstępne i końcowe:

$$\{\text{true}\}x := 5\{x = 5\}$$

co oznacza, że przy dowolnym stanie przed wykonaniem instrukcji, po wykonaniu instrukcji będziemy mieli stan, w którym zmiennej x jest przypisana wartość 5.

Prawdą jest też bardziej skomplikowana formuła:

$$\{x = y + z\}\{\text{if } x < y \text{ then } z := -z\}\{x \leqslant y + z\}$$

3 TODO: Problemy rekurencyjne i ich rozwiązywanie.

Lorem ipsum dupa dupa

5 TODO: Pozycyjne systemy liczbowe i konwersje pomiędzy nimi.

Lorem ipsum dupa dupa

7 TODO: Typ, zmienna, obiekt i zarządzanie pamięcią.

Lorem ipsum dupa dupa

8 Paweł TODO: Instrukcje sterujące przepływem programu.

Lorem ipsum dupa dupa

11 TODO: Porównanie zadań przełącznika (switcha) i routera.

Lorem ipsum dupa dupa

14 TODO: Obiekt i klasa w wybranym języku programowania zorientowanym obiektowo.

Lorem ipsum dupa dupa

16 TODO: Interfejsy i klasy abstrakcyjne w programowaniu obiektowym.

Lorem ipsum dupa dupa

18 TODO: Algorytmy sortowania.

Lorem ipsum dupa dupa

19 TODO: Strategia „dziel i zwyciężaj" budowania algorytmów.

Lorem ipsum dupa dupa

20 TODO: Algorytmy typu zachłannego.

Lorem ipsum dupa dupa

21 TODO: Algorytmy z nawrotami.

Lorem ipsum dupa dupa

22 TODO: Grafy, drzewa, kopce – charakterystyka i przykłady zastosowania.

Lorem ipsum dupa dupa

47 Paweł TODO: Definicja i klasy złożoności obliczeniowej – czasowej i pamięciowej.

Lorem ipsum dupa dupa

56 TODO: Kodowanie liczb ze znakiem w systemie U2, generowanie liczby ze znakiem przeciwnym, dodawanie i odejmowanie.

Lorem ipsum dupa dupa

Rozdział IV

Pytania których raczej nie dostaniemy

28 TODO: Różnice pomiędzy obsługą zdarzeń w przerwaniach sprzętowych a obsługą zdarzeń w pętli programowej.

Lorem ipsum dupa dupa

29 TODO: Powody i przykłady stosowania mikrokontrolerów zamiast typowych komputerów.

Lorem ipsum dupa dupa

39 TODO: Standardowe metodyki procesu wytwórczego oprogramowania.

Lorem ipsum dupa dupa

40 TODO: Metodyki zwinne – SCRUM.

Lorem ipsum dupa dupa

41 TODO: Testowanie oprogramowania.

Lorem ipsum dupa dupa

42 TODO: Diagramy UML.

Lorem ipsum dupa dupa

43 TODO: Wzorce projektowe programowania obiektowego.

Lorem ipsum dupa dupa

44 TODO: Definicja funkcji obliczalnej (częściowo rekurencyjnej).

Lorem ipsum dupa dupa

45 TODO: Maszyna Turinga jako model procesów obliczalnych.

Lorem ipsum dupa dupa

46 TODO: Zagadnienia nierostrzygalne w kontekście obliczalności.

Lorem ipsum dupa dupa

49 TODO: Gramatyki bezkontekstowe – definicje, charakterystyki i przykłady.

Lorem ipsum dupa dupa

50 TODO: Analiza leksykalna, syntaktyczna i semantyczna kodu.

Lorem ipsum dupa dupa

51 TODO: Rodzaje błędów w kontekście analizy leksykalnej, syntaktycznej i semantycznej kodu.

Lorem ipsum dupa dupa

52 TODO: Deklaratywne programowanie funkcyjne: rachunek lambda, monady.

Lorem ipsum dupa dupa

54 TODO: Podstawowe układy systemu mikroprocesorowego i sposób wymiany informacji pomiędzy nimi.

Lorem ipsum dupa dupa

55 TODO: Dekoder, multiplekser i demultiplekser: budowa, zasada, działania, przeznaczenie, zastosowanie.

Lorem ipsum dupa dupa

57 TODO: Budowa i zasada działania generatora obrazu w systemie mikroprocesorowym.

Lorem ipsum dupa dupa

23 TODO: Wielowarstwowa organizacja systemów komputerowych.

Lorem ipsum dupa dupa

24 TODO: System operacyjny – charakterystyka, zadania, klasyfikacja.

Lorem ipsum dupa dupa

25 TODO: Procesy i wątki – charakterystyka i problemy.

Lorem ipsum dupa dupa

26 TODO: Zarządzanie pamięcią operacyjną w systemie operacyjnym.

Lorem ipsum dupa dupa

27 TODO: Organizacja systemu plików i pamięci zewnętrznej.

Lorem ipsum dupa dupa

3 TODO: Podstawowe charakterystyki statystyki opisowej i matematycznej.

Lorem ipsum dupa dupa