Egzamin dyplomowy *Informatyka stosowana* II stopień **od 2022/2023**

Przedmiot	**Nr**	**Pytanie**
~~*Sieci mobilne i komórkowe WLAN*~~	~~1~~	~~Na czym polega idea telefonii komórkowej, na przykładzie sieci GSM?~~
	~~2~~	~~Jakie metody wielodostępu są wykorzystywane w radiowej sieci dostępowej 2G (GSM)?~~
	~~3~~	~~Proszę powiedzieć w jakim celu opracowano technologie GPRS i EDGE?~~
Sieci rozległe	4	Podać przykłady protokołów rutingu klasy IGP (Interior Gateway Protocol) i opisać ich różnice.
	5	Jaki protokół rutingu pracuje w Internecie między systemami autonomicznymi? Proszę powiedzieć co go różni od protokołu OSPF.
	6	Gdzie w sieci jest wykorzystywany protokół MPLS i podać w jakim celu jest stosowany?
~~*Język Fortran 90/95*~~	~~7~~	~~Podstawowe bloki konstrukcyjne Fortranu, komunikacja pomiędzy nimi.~~
	~~8~~	~~Wskaźniki i dynamiczne struktury danych.~~
	~~9~~	~~Przenoszalność (powtarzalność) złożonych obliczeń na różne maszyny, unikalność tych narzędzi w języku Fortran. Parametryzowane typy danych (zakres wartości, precyzja).~~
Zaawansowane techniki programowania obiektowego w C++	10	Objaśnij działanie inteligentnych wskaźników na przykładzie unique_ptr oraz shared_ptr.
	11	Klasy cech w programowaniu generycznym na przykładzie std::numeric_limits
	12	Metaprogramowanie na przykładzie szablonu funkcji potęgowania całkowitego
Programowanie rozproszone i równoległe	13	Na czym polega użycie modelu fork-join w OpenMP?
	14	Co to jest "namiastka"? Proszę o omówienie tego zagadnienia na podstawie technologii RMI.
	15	Proszę omówić problem błędów żywotności w programach współbieżnych.
	16	Na czym polega problem braku widoczności w Java i jak się go rozwiązuje?
	17	Dlaczego metoda stop z klasy Thread w Java uznana jest za metodę przestarzałą i jak można ją zastąpić?
~~*Projektowanie sieci komputerowych*~~	~~18~~	~~Jakie są zalety modularyzacji w budowie sieci komputerowej?~~
	~~19~~	~~Dlaczego w sieciach dużej dostępności powinny być preferowane połączenia typu punkt-punkt o równych kosztach?~~
	~~20~~	~~Proszę wskazać zalety i wady translacji adresów sieciowych.~~
Zaawansowana grafika komputerowa	21	Filtry konwolucyjne dla obrazów rastrowych.
	22	Skalowanie obrazów rastrowych (resampling).
	23	Pojęcie grafu sceny.
~~*Symulacje Monte Carlo i superkomputery*~~	~~24~~	~~Etapy algorytmu Metropolisa dla modelu Isinga.~~
	~~25~~	~~Omówić znaczenie warunku równowagi szczegółowej.~~
	~~26~~	~~Omówić dwie spośród następujących metod szacowania błędów statystycznych: naiwne odchylenie standardowe, jackknife, bootstrap, uwzględnienie autokorelacji.~~
Projektowanie obiektowe	27	Jaka jest różnica między testami wydajnościowymi, obciążeniowych oraz stres testami?
	28	Proszę opisać trzy wzorce behawioralne.
	29	Jak działa prototyp w języku JavaScript?
	30	Proszę opisać cztery podstawowe typy Singletona.
	31	Czym jest dług technologiczny?
Zarządzanie projektami	32	Proszę podać najważniejsze współzależne parametry projektu
	33	Proszę podać etapy procesu zarządzania projektami
	34	Proszę wymienić wartości Manifestu Agile

Metody statystyczne	35	Stan stacjonarny w markowowskich procesach stochastycznych - metody wyznaczania i zastosowania.
	36	Systemy kolejkowe - definicja, typy, notacja Kendalla, współdzielenie procesora, prawo Little'a, przykłady systemów kolejkowych
	37	Modele z ukrytym procesem Markowa - definicja i zastosowanie. Ogólna zasada działania algorytmu Viterbiego.
Zaawansowane interfejsy graficzne	38	Wiązanie danych w WPF
	39	Metody testowania aplikacji posiadających graficzny interfejs użytkownika
	40	Wzorzec MVVM
~~*Bezpieczeństwo w sieciach*~~	~~41~~	Omów zasadę działania firewalla
	~~42~~	Wymień znane metody kryptograficzne
	~~43~~	Omów fazy ataków sieciowych.
Projektowanie wspomagane komputerem	**44**	**Rodzaje i charakterystyka akcji projektowych**
	45	**Narzędzia wykorzystywane w projektowaniu wizualnym i ich charakterystyka**
	46	**Definicja interpretacji grafu**
E-biznes	47	Czym jest monada?
	48	Czym są kontenery oraz obrazy w Dockerze?
	49	Czym jest trait?
	50	Jak działa tzw. Companion Object?
~~*Kryptografia*~~	~~51~~	Proszę wyjaśnić na czym polega kryptografia asymetryczna i jakie ma zastosowania?
	~~52~~	Proszę wyjaśnić czym jest i jakie narzędzia kryptograficzne wykorzystuje protokół TLS?
	~~53~~	Proszę wymienić i krótko scharakteryzować główne przykłady ataków na systemy kryptograficzne.
Głębokie sieci neuronowe	54	Sieci neuronowe: podstawowe informacje. Z czego się składają, jak się je uczy.
	55	Konwolucyjne sieci neuronowe.
	56	Rekurencyjne sieci neuronowe.
	57	Generative adversarial networks.
	58	Adversarial examples.
~~*Komputerowa analiza zagadnień różniczkowych*~~	~~59~~	~~Jawne i niejawne metody Rungego-Kutty i ich obszary stabilności~~
	~~60~~	~~Adaptacyjna zmiana kroku całkowania i zagnieżdżone metody Rungego-Kutty~~
	~~61~~	~~Metody dla dwupunktowych problemów brzegowych~~
~~*Analiza szeregów czasowych*~~	~~62~~	~~Procesy AR, MA, ARMA i ARIMA~~
	~~63~~	~~Szeregi z długoczasowymi korelacjami, wykładnik Hursta i metody jego obliczania~~
	~~64~~	~~Zastosowania falek (wavelets) do kompresji i odszumiania sygnałów~~
Biometria	~~65~~	~~Metody weryfikacji tożsamości użytkownika w systemie informatycznym~~
	~~66~~	~~Jakie cechy powinien mieć biometryczny system identyfikacji/weryfikacji tożsamości.~~
	~~67~~	~~Rozpoznawanie mowy oraz identyfikacja mówcy (za trudne …?)~~
	~~68~~	~~Klasyfikacja statystyczna w rozpoznawaniu obrazów twarzy (pewnie też za trudne …) zamiast tego może być…~~
	~~69~~	~~Jak działa binarny klasyfikator obrazów ?~~
Projektowanie aplikacji internetowych	70	Omów architekturę Model-Widok-Kontroler (MVC) w kontekście działania aplikacji internetowej.
	71	Omów model komunikacji asynchronicznej dla aplikacji internetowych oraz wykorzystanie podejścia REST wg. modelu Richardsona.

	72	Omów model tworzenia aplikacji internetowych w oparciu o architekturę rozproszonych mikroserwisów SOA.
~~*Geometria 3D dla projektantów gier wideo?*~~	~~73~~	~~W jaki sposób modelujemy bryłę widzenia?~~
	~~74~~	~~Przedstaw model oświetlenia Blinna-Phonga~~
	~~75~~	~~Wyjaśnij różnicę między cieniowaniem Gouraud i cieniowaniem Phonga~~
	~~76~~	~~Przestaw zastosowanie metody śledzenia promieni do wyznaczania elementów widocznych na scenie~~
	~~77~~	~~Przedstaw opis krzywych kubicznych. Co to są krzywe Beziera, splajny Catmulla-Roma i krzywe NURBS?~~

Spis treści

A) Sieci rozległe 6
4)Podać przykłady protokołów rutingu klasy IGP (Interior Gateway Protocol) i opisać ich różnice. 6
1. RIP (Routing Information Protocol) 6
2. OSPF (Open Shortest Path First) 7
3. IS-IS (Intermediate System to Intermediate System) 7
4. EIGRP (Enhanced Interior Gateway Routing Protocol) 7
5) Jaki protokół routingu pracuje w Internecie między systemami autonomicznymi? Proszę powiedzieć co go różni od protokołu OSPF. 8
6) Gdzie w sieci jest wykorzystywany protokół MPLS i podać w jakim celu jest stosowany? 8
B) Zaawansowane techniki programowania obiektowego w C++ 9
10) Objaśnij działanie inteligentnych wskaźników na przykładzie unique_ptr oraz shared_ptr. 9
11) Klasy cech w programowaniu generycznym na przykładzie std::numeric_limits 9
12) Meta programowanie na przykładzie szablonu funkcji potęgowania całkowitego 10
C) Programowanie rozproszone i równoległe 11
13) Na czym polega użycie modelu fork-join w OpenMP? 11
14) Co to jest "namiastka"? Proszę o omówienie tego zagadnienia na podstawie technologii RMI 12
15) Proszę omówić problem błędów żywotności w programach współbieżnych. 15
16) Na czym polega problem braku widoczności w Java i jak się go rozwiązuje? 16
17) Dlaczego metoda stop z klasy Thread w Java uznana jest za metodę przestarzałą i jak można ją zastąpić? 18
D) Zaawansowana grafika komputerowa 21
21) Filtry konwolucyjne dla obrazów rastrowych. 21
22) Skalowanie obrazów rastrowych (resampling). 22
23) Pojęcie grafu sceny. 24
E) Projektowanie obiektowe 25
27)Jaka jest różnica między testami wydajnościowymi, obciążeniowych oraz stres testami? 25
1. Testy wydajnościowe (Performance Testing) 25
2. Testy obciążeniowe (Load Testing) 25
3. Testy przeciążeniowe (Stress Testing) 26
28) Proszę opisać trzy wzorce behawioralne. 26
29) Jak działa prototyp w języku JavaScript? 30
30) Proszę opisać cztery podstawowe typy Singletona. 31
31) Czym jest dług technologiczny? 32
F) Zarządzanie projektami 33
32) Proszę podać najważniejsze współzależne parametry projektu 33
33)Proszę podać etapy procesu zarządzania projektami 33
1)Inicjacja (Initiation): 33

2)Planowanie (Planning): ... 33
3)Realizacja (Execution): ... 33
4)Monitorowanie i kontrola (Monitoring and Controlling): ... 34
5)Zamknięcie (Closing): ... 34
34) Proszę wymienić wartości Manifestu Agile ... 34
Ludzi i interakcje ponad procesami i narzędziami. ... 34
Działające oprogramowanie ponad obszerną dokumentacją. ... 34
Współpracę z klientem ponad negocjowaniem umowy. ... 34
Reagowanie na zmianę ponad realizację założonego planu ... 34
G)Metody statystyczne ... 34
35) Stan stacjonarny w markowowskich procesach stochastycznych - metody wyznaczania i zastosowania. ... 34
36) Systemy kolejkowe - definicja, typy, notacja Kendalla, współdzielenie procesora, prawo Little'a, przykłady systemów kolejkowych ... 35
37)Modele z ukrytym procesem Markowa - definicja i zastosowanie. Ogólna zasada działania algorytmu Viterbiego. ... 37
H)Zaawansowane interfejsy Graficzne ... 40
38)Wiązanie Danych (Data Binding) w WPF (Windows Presentation Foundation) ... 40
39) Metody testowania aplikacji posiadających graficzny interfejs użytkownika ... 41
Ręczne testowanie ... 41
Automatyczne testowanie ... 41
Narzędzia ... 41
40) Wzorzec MVVM ... 42
Komponenty ... 42
I) Bezpieczeństwo w sieciach ... 42
41) Omów zasadę działania firewalla ... 42
42) Wymień znane metody kryptograficzne ... 44
43) Omów fazy ataków sieciowych ... 45
J)Projektowanie wspomagane komputerem ... 47
44) Rodzaje i charakterystyka akcji projektowych ... 47
45) Narzędzia wykorzystywane w projektowaniu wizualnym i ich charakterystyka ... 47
46) Definicja interpretacji grafu ... 49
K) E-biznes ... 50
47) Czym jest monada? ... 50
48) Czym są kontenery oraz obrazy w Dockerze? ... 51
1. Obraz (Image) ... 51
2. Kontener (Container) ... 51
49) Czym jest trait? ... 52
50) Jak działa tzw. Companion Object? ... 53
L) Kryptografia ... 54
51) Proszę wyjaśnić na czym polega kryptografia asymetryczna i jakie ma zastosowania? ... 54

52) Proszę wyjaśnić czym jest i jakie narzędzia kryptograficzne wykorzystuje protokół TLS? 55
53) Proszę wymienić i krótko scharakteryzować główne przykłady ataków na systemy kryptograficzne. 56
M) Głębokie sieci neuronowe 57
54) Sieci neuronowe: podstawowe informacje. Z czego się składają, jak się je uczy. 57
55) Konwolucyjne sieci neuronowe..... 58
56) Rekurencyjne sieci neuronowe. 59
57) Generative adversarial networks. 61
58) Adversarial examples. 63
N) Projektowanie aplikacji internetowych 64
70) Omów architekturę Model-Widok-Kontroler (MVC) w kontekście działania aplikacji internetowej. . 64
71) Omów model komunikacji asynchronicznej dla aplikacji internetowych oraz wykorzystanie podejścia REST wg. modelu Richardsona. 66
72) Omów model tworzenia aplikacji internetowych w oparciu o architekturę rozproszonych mikroserwisów SOA. 68

A) Sieci rozległe

4)Podać przykłady protokołów rutingu klasy IGP (Interior Gateway Protocol) i opisać ich różnice.

Protokoły rutingu klasy IGP (Interior Gateway Protocol) są stosowane do zarządzania ruchem wewnątrz jednej autonomicznej systemu (AS) w sieciach komputerowych. Poniżej znajdują się przykłady popularnych protokołów IGP oraz ich kluczowe różnice:

Przykłady protokołów IGP:

1. **RIP (Routing Information Protocol)**
2. **OSPF (Open Shortest Path First)**
3. **IS-IS (Intermediate System to Intermediate System)**
4. **EIGRP (Enhanced Interior Gateway Routing Protocol)**

1. RIP (Routing Information Protocol)

- **Typ protokołu:** Protokół wektora odległości.

- **Maksymalna liczba skoków:** 15 skoków (16 jest traktowane jako nieskończoność, co oznacza, że trasa jest niedostępna).
- **Metryka:** Liczba skoków (hop count).
- **Aktualizacja:** RIP przesyła całą tablicę routingu co 30 sekund, co powoduje duży ruch sieciowy i wolne konwergowanie.
- **Wersje:** RIP v1 (brak obsługi CIDR), RIP v2 (obsługuje CIDR).

Zastosowanie: RIP jest prosty w konfiguracji, ale ma ograniczoną skalowalność i jest rzadko używany w większych sieciach z powodu jego ograniczeń.

2. OSPF (Open Shortest Path First)

- **Typ protokołu:** Protokół stanu łącza.
- **Metryka:** Koszt oparty na przepustowości łącza.
- **Aktualizacja:** OSPF wysyła aktualizacje topologii tylko w przypadku zmian w sieci, co zmniejsza obciążenie sieci.
- **Struktura:** OSPF dzieli sieć na obszary, co umożliwia lepsze skalowanie i zarządzanie dużymi sieciami.
- **Konwergencja:** Szybka konwergencja dzięki algorytmowi Dijkstry.
- **Autoryzacja:** Obsługuje autoryzację i bezpieczeństwo na poziomie każdej wymiany informacji.

Zastosowanie: OSPF jest szeroko stosowany w średnich i dużych sieciach korporacyjnych ze względu na jego elastyczność i wydajność.

3. IS-IS (Intermediate System to Intermediate System)

- **Typ protokołu:** Protokół stanu łącza.
- **Metryka:** Możliwość dostosowania metryk, domyślnie na podstawie przepustowości.
- **Struktura:** IS-IS obsługuje hierarchiczne routowanie poprzez dzielenie sieci na poziomy (Level 1 i Level 2).
- **Protokół warstwy:** IS-IS działa na warstwie łącza danych, co czyni go niezależnym od protokołów warstwy sieciowej (np. IP, IPv6).
- **Elastyczność:** Obsługuje duże sieci z różnymi technologiami, często stosowany w sieciach dostawców usług.

Zastosowanie: IS-IS jest preferowany w dużych sieciach dostawców usług internetowych (ISP) i telekomunikacyjnych ze względu na jego niezależność od warstwy sieciowej i skalowalność.

4. EIGRP (Enhanced Interior Gateway Routing Protocol)

- **Typ protokołu:** Hybrydowy (łączy cechy protokołów stanu łącza i wektora odległości).
- **Metryka:** Złożona metryka bazująca na przepustowości, opóźnieniu, niezawodności, obciążeniu i MTU.
- **Aktualizacja:** EIGRP przesyła tylko zmiany, a nie całą tablicę routingu, co zmniejsza obciążenie sieci.
- **Konwergencja:** Szybka konwergencja dzięki użyciu algorytmu DUAL (Diffusing Update Algorithm).
- **Zarządzanie:** Łatwość zarządzania dzięki uproszczonej konfiguracji w porównaniu do OSPF i IS-IS.

Zastosowanie: EIGRP jest często stosowany w sieciach korporacyjnych, szczególnie tych korzystających z rozwiązań Cisco, ponieważ jest to protokół opracowany przez Cisco i ściśle zintegrowany z ich sprzętem.

Kluczowe różnice:

1. **Typ protokołu:** RIP używa wektora odległości, podczas gdy OSPF i IS-IS używają stanu łącza, a EIGRP jest protokołem hybrydowym.
2. **Skalowalność:** OSPF i IS-IS są bardziej skalowalne niż RIP i EIGRP, co sprawia, że są bardziej odpowiednie do dużych sieci.
3. **Konwergencja:** OSPF, IS-IS, i EIGRP oferują szybszą konwergencję niż RIP.
4. **Metryka:** RIP używa prostej metryki liczby skoków, OSPF i IS-IS opierają się na bardziej zaawansowanych metrykach związanych z przepustowością, a EIGRP wykorzystuje złożoną metrykę.
5. **Zastosowanie:** RIP jest używany w małych sieciach, OSPF i IS-IS w dużych sieciach korporacyjnych i ISP, natomiast EIGRP w sieciach korporacyjnych z rozwiązaniami Cisco.

5) Jaki protokół routingu pracuje w Internecie między systemami autonomicznymi? Proszę powiedzieć co go różni od protokołu OSPF.

BGP (Border Gateway Protocol) jest używany do routingu między różnymi systemami autonomicznymi (AS) w Internecie, podczas gdy **OSPF (Open Shortest Path First)** działa wewnątrz jednego AS.

Główne różnice:

1. **Zakres działania:**
   - **BGP:** Działa między różnymi AS-ami (EGP).
   - **OSPF:** Działa wewnątrz jednego AS (IGP).
2. **Typ protokołu:**
   - **BGP:** Wektor ścieżek (oparty na politykach).
   - **OSPF:** Stan łącza (oparty na kosztach).
3. **Metryka:**
   - **BGP:** Używa atrybutów takich jak AS-path, policy, prefiks.
   - **OSPF:** Używa kosztu (na podstawie przepustowości).
4. **Konwergencja:**
   - **BGP:** Wolniejsza, z naciskiem na stabilność.
   - **OSPF:** Szybka, z naciskiem na adaptację do zmian.
5. **Zastosowanie:**
   - **BGP:** Globalny routing między sieciami w Internecie.
   - **OSPF:** Routing wewnątrz jednej organizacji lub sieci.

6) Gdzie w sieci jest wykorzystywany protokół MPLS i podać w jakim celu jest stosowany?

MPLS (Multiprotocol Label Switching) jest wykorzystywany głównie w sieciach szkieletowych operatorów telekomunikacyjnych oraz dużych korporacyjnych sieciach WAN. MPLS znajduje również zastosowanie w sieciach centrów danych i sieciach usługowych.

Cel stosowania MPLS:

1. **Efektywność routingu:**
   - MPLS przyspiesza przesyłanie danych poprzez używanie etykiet (labels) zamiast tradycyjnych metod routingu IP. Pakiety są kierowane na podstawie prostego przełączania etykiet, co redukuje czas przetwarzania w routerach i zwiększa wydajność sieci.
2. **Jakość usług (QoS):**
   - MPLS umożliwia różnicowanie ruchu sieciowego i zapewnianie odpowiednich poziomów jakości usług. Dzięki MPLS, operatorzy mogą priorytetyzować ruch, np. zapewniając wyższy priorytet dla głosu (VoIP) czy wideo, które są wrażliwe na opóźnienia.

3. **Skalowalność i zarządzanie:**
    - MPLS ułatwia zarządzanie dużymi sieciami poprzez umożliwienie łatwego tworzenia wirtualnych sieci prywatnych (VPN) na tej samej infrastrukturze fizycznej. To pozwala na efektywne zarządzanie wieloma klientami na jednej sieci operatora.
4. **Wieloprotokołowość:**
    - MPLS obsługuje różne protokoły warstwy sieciowej (np. IPv4, IPv6), co czyni go wszechstronnym rozwiązaniem w różnorodnych środowiskach sieciowych.
5. **Redundancja i niezawodność:**
    - Dzięki szybkiemu przełączaniu awaryjnemu, MPLS może zapewniać wysoką dostępność usług, co jest kluczowe w sieciach krytycznych, takich jak sieci bankowe czy telekomunikacyjne.

B) Zaawansowane techniki programowania obiektowego w C++

10) Objaśnij działanie inteligentnych wskaźników na przykładzie unique_ptr oraz shared_ptr.

std::unique_ptr i **std::shared_ptr** to inteligentne wskaźniki w C++, które automatycznie zarządzają dynamicznie alokowaną pamięcią.

- **std::unique_ptr**: Ma unikalne prawo własności nad zasobem, co oznacza, że tylko jeden wskaźnik może zarządzać danym obiektem. Nie można go kopiować, tylko przenosić. Zasób jest zwalniany, gdy wskaźnik wychodzi poza zakres.
- **std::shared_ptr**: Pozwala na współdzielenie zasobu przez wiele wskaźników. Zarządza pamięcią za pomocą licznika referencji, który zwalnia zasób dopiero wtedy, gdy nikt już go nie używa.

Różnice:

- unique_ptr: Lekki, unikalny właściciel zasobu.
- shared_ptr: Współdzielony właściciel, więcej wskaźników może odnosić się do tego samego zasobu.

11) Klasy cech w programowaniu generycznym na przykładzie std::numeric_limits

Klasy cech (ang. **traits classes**) w programowaniu generycznym to wzorce projektowe, które umożliwiają uzyskanie informacji o właściwościach typów, które są używane w kontekście szablonów. Używane są do określania specyficznych cech typów w sposób generyczny i są kluczowe w programowaniu z szablonami (ang. *templates*).

Przykład: std::numeric_limits

std::numeric_limits to przykład klasy cech w C++, która dostarcza informacji o właściwościach typów numerycznych (takich jak int, float, double). Jest częścią standardowej biblioteki C++ i jest specjalizowana dla różnych typów podstawowych.

Działanie std::numeric_limits:

Klasa std::numeric_limits umożliwia dostęp do cech typów numerycznych, takich jak:

- Maksymalna i minimalna wartość danego typu (max(), min()).

- Dokładność (digits).
- Czy typ obsługuje liczby ujemne (is_signed).
- Czy typ jest zmiennoprzecinkowy (is_integer).
- Liczba cyfr w części ułamkowej (digits10).

Przykład użycia:

cpp Skopiuj kod

```cpp
#include <iostream>
#include <limits>  // Zawiera std::numeric_limits

int main() {
    std::cout << "Właściwości typu int:\n";
    std::cout << "Min: " << std::numeric_limits<int>::min() << '\n';
    std::cout << "Max: " << std::numeric_limits<int>::max() << '\n';
    std::cout << "Czy signed: " << std::numeric_limits<int>::is_signed << '\n';

    std::cout << "\nWłaściwości typu double:\n";
    std::cout << "Min: " << std::numeric_limits<double>::min() << '\n';
    std::cout << "Max: " << std::numeric_limits<double>::max() << '\n';
    std::cout << "Dokładność: " << std::numeric_limits<double>::digits10 << '\n';

    return 0;
}
```

Wyjaśnienie:

- **std::numeric_limits<int>::min()**: Zwraca minimalną wartość, jaką może przechowywać typ int.
- **std::numeric_limits<double>::digits10**: Określa liczbę cyfr dziesiętnych, które można przechowywać i odtworzyć bez utraty precyzji w typie double.
- **std::numeric_limits<int>::is_signed**: Informuje, czy typ int jest typem ze znakiem (true dla typów ze znakiem).

Podsumowanie:

Klasy cech, takie jak std::numeric_limits, są używane do uzyskiwania informacji o typach w sposób generyczny. Umożliwiają pisanie bardziej elastycznego i wielokrotnego użytku kodu, który może działać z różnymi typami, dostosowując swoje zachowanie na podstawie właściwości tych typów.

12) Meta programowanie na przykładzie szablonu funkcji potęgowania całkowitego

Metaprogramowanie w C++ polega na pisaniu kodu, który jest wykonywany podczas **kompilacji**, aby generować inny kod lub wykonywać obliczenia. Przykładem tego jest implementacja potęgowania całkowitego za pomocą szablonów funkcji, gdzie wynik obliczeń jest znany już w czasie kompilacji.

Przykład: Potęgowanie całkowite z użyciem szablonów

cpp Skopiuj kod

```cpp
#include <iostream>

// Podstawowa wersja szablonu - potęgowanie przez 0 zawsze zwraca 1
template <int Base, int Exponent>
struct Power {
    static constexpr int value = Base * Power<Base, Exponent - 1>::value;
};

// Specjalizacja dla wykładnika równego 0
template <int Base>
struct Power<Base, 0> {
    static constexpr int value = 1;
};

int main() {
    // Przykład użycia - obliczamy 2^8 w czasie kompilacji
    constexpr int result = Power<2, 8>::value;

    std::cout << "2^8 = " << result << std::endl;

    return 0;
}
```

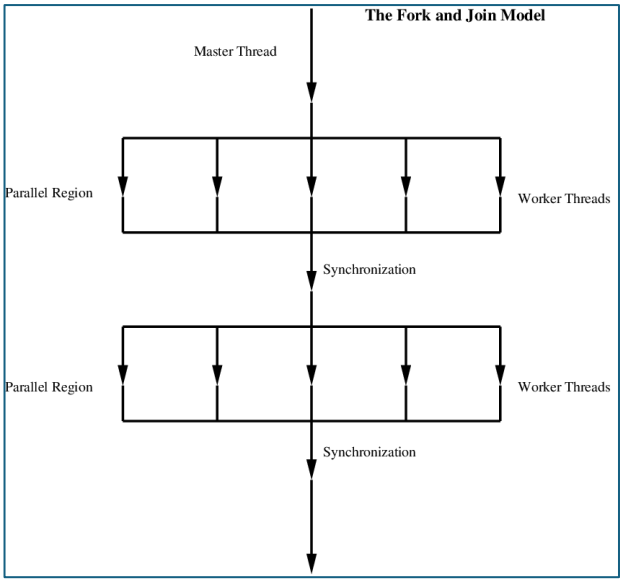
C) Programowanie rozproszone i równoległe

13) Na czym polega użycie modelu fork-join w OpenMP?

Model Fork-Join w OpenMP

Fork-Join to popularny model programowania równoległego, który jest wspierany przez OpenMP, narzędzie do programowania równoległego w C, C++ i Fortran. Model ten działa w następujący sposób:

1. **Sekcja sekwencyjna (Fork):**
   - Program rozpoczyna się jako pojedynczy wątek (wątek główny).
   - W pewnym momencie, napotykając na konstrukcję OpenMP (np. #pragma omp parallel), program „rozgałęzia się” (fork), tworząc wiele wątków, które wykonują równolegle określone zadanie.
2. **Sekcja równoległa:**
   - Wątki pracują równolegle nad różnymi częściami zadania. W OpenMP można kontrolować liczbę wątków oraz sposób podziału zadań między wątki.
   - Każdy wątek wykonuje te same instrukcje, ale może operować na różnych danych.
3. **Sekcja sekwencyjna (Join):**
   - Po zakończeniu sekcji równoległej, wątki kończą swoje zadania i „dołączają” (join) z powrotem do wątku głównego.
   - Program kontynuuje wykonywanie kodu w trybie sekwencyjnym.



Przykład użycia modelu Fork-Join w OpenMP

cpp Skopiuj kod

```cpp
#include <iostream>
#include <omp.h>

int main() {
    int n = 10;
    int sum = 0;

    // Fork: Tworzenie wątków do równoległego wykonania pętli
    #pragma omp parallel for reduction(+:sum)
    for (int i = 0; i < n; i++) {
        sum += i;  // Każdy wątek dodaje do sumy swoją część pracy
    }
    // Join: Dołączenie wątków, kontynuacja w wątku głównym

    std::cout << "Suma: " << sum << std::endl;

    return 0;
}
```

14) Co to jest "namiastka"? Proszę o omówienie tego zagadnienia na podstawie technologii RMI.

"Namiastka" (ang. **stub**) w kontekście technologii RMI (Remote Method Invocation) to ważny element, który umożliwia wywoływanie metod zdalnych obiektów tak, jakby były lokalne.

RMI (Remote Method Invocation)

RMI to technologia w języku Java, która pozwala na wywoływanie metod obiektów znajdujących się na zdalnych maszynach, jakby były lokalne. Dzięki temu możliwa jest komunikacja między obiektami uruchomionymi na różnych JVM (Java Virtual Machines), często na różnych komputerach.

Namiastka to lokalna reprezentacja obiektu zdalnego. Jest to klasa wygenerowana automatycznie, która implementuje te same interfejsy, co zdalny obiekt. Klient, który chce wywołać metodę na zdalnym obiekcie, tak naprawdę wywołuje ją na namiastce.

Jak działa namiastka?

1. **Wywołanie metody:** Kiedy klient wywołuje metodę na obiekcie zdalnym, wywołanie jest przechwytywane przez namiastkę.
2. **Przekazywanie żądania:** Namiastka zajmuje się marshallingiem, czyli serializacją argumentów metody, i wysyła je przez sieć do serwera, na którym znajduje się rzeczywisty obiekt zdalny.
3. **Obsługa przez serwer:** Na serwerze odpowiednik namiastki (ang. **skeleton**) odbiera to wywołanie, deserializuje argumenty i wywołuje odpowiednią metodę na zdalnym obiekcie.
4. **Zwracanie wyniku:** Wynik wywołania jest następnie serializowany i wysyłany z powrotem do klienta, gdzie namiastka deserializuje wynik i zwraca go do oryginalnej metody wywołanej przez klienta.

Przykład działania namiastki w RMI

Załóżmy, że mamy zdalny obiekt, który implementuje interfejs Calculator, z metodą int add(int a, int b).

Krok po kroku:

1. **Serwer:**
    - Zdalny obiekt CalculatorImpl implementuje interfejs Calculator.
    - Serwer rejestruje ten obiekt w rejestrze RMI (RMI Registry), co umożliwia jego odnalezienie przez klientów.
2. **Klient:**
    - Klient wyszukuje zdalny obiekt przez RMI Registry i otrzymuje namiastkę Calculator.
    - Klient wywołuje metodę add(5, 3) na namiastce.
3. **Namiastka:**
    - Namiastka serializuje argumenty (5 i 3) i wysyła je do serwera.
    - Serwer deserializuje te argumenty i wykonuje metodę add(5, 3) na rzeczywistym obiekcie CalculatorImpl.
4. **Wynik:**
    - Wynik (8) jest przesyłany z powrotem do klienta przez sieć.
    - Namiastka deserializuje wynik i zwraca go do kodu klienta, który wywołał metodę.

Podsumowanie

Namiastka w technologii RMI to kluczowy mechanizm, który umożliwia lokalnym klientom wywoływanie metod na zdalnych obiektach w sposób przezroczysty. Namiastka przechwytuje wywołania metod, przekazuje je przez sieć do zdalnego serwera, a następnie odbiera wyniki i zwraca je klientowi. Dzięki temu programista może pracować z obiektami zdalnymi tak, jakby były lokalne, bez konieczności ręcznego zarządzania komunikacją sieciową.

Following is an example of an RMI server program.

```
import java.rmi.registry.Registry;
import java.rmi.registry.LocateRegistry;
import java.rmi.RemoteException;
import java.rmi.server.UnicastRemoteObject;

public class Server extends ImplExample {
   public Server() {}
   public static void main(String args[]) {
      try {
         // Instantiating the implementation class
         ImplExample obj = new ImplExample();

         // Exporting the object of implementation class
         // (here we are exporting the remote object to the stub
         Hello stub = (Hello) UnicastRemoteObject.exportObject(o

         // Binding the remote object (stub) in the registry
         Registry registry = LocateRegistry.getRegistry();

         registry.bind("Hello", stub);
         System.err.println("Server ready");
      } catch (Exception e) {
         System.err.println("Server exception: " + e.toString())
         e.printStackTrace();
      }
   }
}
```

Following is an example of an RMI client program.

```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;

public class Client {
   private Client() {}
   public static void main(String[] args) {
      try {
         // Getting the registry
         Registry registry = LocateRegistry.getRegistry(null);

         // Looking up the registry for the remote object
         Hello stub = (Hello) registry.lookup("Hello");

         // Calling the remote method using the obtained object
         stub.printMsg();

         // System.out.println("Remote method invoked");
      } catch (Exception e) {
         System.err.println("Client exception: " + e.toString())
         e.printStackTrace();
      }
   }
}
```

15) Proszę omówić problem błędów żywotności w programach współbieżnych.

Błędy żywotności w programach współbieżnych to kategoria problemów, które wynikają z nieprawidłowego zarządzania czasem życia zasobów (takich jak obiekty, zmienne, struktury danych) używanych przez różne wątki. Te błędy mogą prowadzić do niestabilności programu, trudnych do zreplikowania błędów, a nawet awarii.

Zakleszczenie (Deadlock)
Jest to najbardziej znany problem związany z żywotnością. Występuje, gdy dwa lub więcej wątków czeka na zasoby zajęte przez siebie nawzajem, co prowadzi do sytuacji, w której żaden z wątków nie jest w stanie kontynuować wykonania.

Głodzenie (Starvation)
W tym przypadku, niektóre wątki są w stanie ciągle korzystać z zasobów i postępować w swoim wykonaniu, podczas gdy inne wątki są blokowane na czas nieokreślony. Problem głodzenia jest często wynikiem niewłaściwego zarządzania priorytetami wątków.

Wyścig (Race Condition)
Chociaż wyścigi najczęściej kojarzone są z błędami bezpieczeństwa i niepoprawnym działaniem, mogą one również wpływać na żywotność programu. Na przykład, jeśli dwie operacje są w wyścigu o dostęp do zasobu i jedna z nich jest niezbędna dla kontynuacji działania programu, wyścig może spowodować, że ten krytyczny zasób nigdy nie zostanie zdobyty.

Usunięcie zasobu używanego przez inny wątek (Use-After-Free):

- **Opis:** Ten błąd występuje, gdy jeden wątek zwalnia zasób (np. pamięć, plik, mutex), który jest nadal używany przez inny wątek. Może to prowadzić do nieoczekiwanych zachowań, ponieważ inny wątek może próbować użyć zwolnionego zasobu, co często prowadzi do błędów typu segmentation fault.

16) Na czym polega problem braku widoczności w Java i jak się go rozwiązuje?

Problem braku widoczności (visibility problem) w programowaniu współbieżnym w języku Java występuje, gdy zmienne współdzielone przez różne wątki nie są prawidłowo widoczne dla tych wątków. Oznacza to, że jeden wątek może nie widzieć zmian dokonanych przez inny wątek w czasie, co prowadzi do niespójności danych i trudnych do przewidzenia błędów.

Na czym polega problem braku widoczności?

Wielowątkowe środowisko uruchomieniowe może przechowywać kopie zmiennych w lokalnej pamięci cache procesora lub w rejestrach, zamiast w głównej pamięci współdzielonej. Gdy jeden wątek modyfikuje zmienną, inny wątek może nie zauważyć tej zmiany, ponieważ odczytuje starą wartość zmiennej z własnej kopii w pamięci cache.

Przykład problemu:

java Skopiuj kod

```java
public class VisibilityProblem {
    private static boolean flag = false;

    public static void main(String[] args) {
        Thread writer = new Thread(() -> {
            flag = true;
        });

        Thread reader = new Thread(() -> {
            while (!flag) {
                // busy-waiting
            }
            System.out.println("Flag has been updated!");
        });

        writer.start();
        reader.start();
    }
}
```

W tym przykładzie wątek reader może nigdy nie wyjść z pętli while (!flag), ponieważ może nigdy nie zobaczyć zmiany wartości flag dokonanej przez wątek writer. Problem wynika z braku synchronizacji między tymi wątkami, przez co nie gwarantuje się, że zmiana wartości flag będzie widoczna dla innych wątków.

Jak rozwiązać problem braku widoczności?

1. **Użycie słowa kluczowego volatile:**
    - Deklarowanie zmiennej jako volatile sprawia, że wszystkie wątki odczytują i zapisują tę zmienną bezpośrednio do pamięci głównej, pomijając pamięć cache. Gwarantuje to, że zmiany wprowadzone przez jeden wątek będą natychmiast widoczne dla innych wątków.

```java
public class VisibilityProblem {
    private static volatile boolean flag = false;

    public static void main(String[] args) {
        Thread writer = new Thread(() -> {
            flag = true;
        });

        Thread reader = new Thread(() -> {
            while (!flag) {
                // busy-waiting
            }
            System.out.println("Flag has been updated!");
        });

        writer.start();
        reader.start();
    }
}
```

W tym przykładzie użycie volatile zapewnia, że zmiana wartości flag dokonana przez wątek writer będzie widoczna dla wątku reader.

2. **Użycie synchronizacji:**

- Synchronizacja bloków kodu za pomocą **synchronized** nie tylko zapewnia wzajemne wykluczanie (jednoczesny dostęp tylko jednego wątku do bloku), ale także synchronizuje widoczność zmian zmiennych pomiędzy wątkami.

- Przykład:

```java
public class VisibilityProblem {
    private static boolean flag = false;

    public static synchronized void setFlag(boolean value) {
        flag = value;
    }

    public static synchronized boolean getFlag() {
        return flag;
    }

    public static void main(String[] args) {
        Thread writer = new Thread(() -> {
            setFlag(true);
        });

        Thread reader = new Thread(() -> {
            while (!getFlag()) {
                // busy-waiting
            }
            System.out.println("Flag has been updated!");
        });

        writer.start();
        reader.start();
    }
}
```

3) Użycie klas z pakietu java.util.concurrent:

- Klasy takie jak AtomicBoolean, ReentrantLock, czy CountDownLatch oferują zaawansowane mechanizmy synchronizacji, które nie tylko zapewniają wzajemne wykluczanie, ale również odpowiednią widoczność zmiennych między wątkami.

Przykład z AtomicBoolean:

java Skopiuj kod

```java
import java.util.concurrent.atomic.AtomicBoolean;

public class VisibilityProblem {
    private static AtomicBoolean flag = new AtomicBoolean(false);

    public static void main(String[] args) {
        Thread writer = new Thread(() -> {
            flag.set(true);
        });

        Thread reader = new Thread(() -> {
            while (!flag.get()) {
                // busy-waiting
            }
            System.out.println("Flag has been updated!");
        });

        writer.start();
        reader.start();
    }
}
```

17) Dlaczego metoda stop z klasy Thread w Java uznana jest za metodę przestarzałą i jak można ją zastąpić?

Metoda **stop**() z klasy Thread w Java została uznana za przestarzałą (deprecated) od wersji JDK 1.2 z powodu licznych problemów, które mogła powodować w kontekście bezpieczeństwa i stabilności aplikacji. Oto główne powody, dla których metoda stop() jest uznana za przestarzałą oraz jak można ją bezpiecznie zastąpić:

Problemy z metodą stop()

1. **Nagłe przerwanie działania wątku:**
    - Metoda stop() powoduje natychmiastowe zatrzymanie wątku, bez względu na to, w którym miejscu swojego wykonania się znajduje. Wątek zostaje przerwany, a wszystkie zablokowane zasoby (takie jak zamknięte pliki, zablokowane mutexy itp.) mogą pozostać w stanie nieoczekiwanym.

2. **Brak możliwości zwolnienia zasobów:**

    - Przerwanie wątku przy użyciu stop() nie daje wątkowi szansy na zakończenie pracy w sposób uporządkowany. Na przykład, bloki finally mogą nie zostać wykonane, co może prowadzić do wycieków pamięci lub innych problemów z zarządzaniem zasobami.

3. **Nieprzewidywalność i trudności w debugowaniu:**

    - Nagłe przerwanie wątku może pozostawić obiekt w niespójnym stanie, co prowadzi do trudnych do wykrycia i naprawienia błędów, ponieważ inne wątki mogą nadal korzystać z częściowo zaktualizowanych danych.

4. **Potencjalne problemy z bezpieczeństwem:**

    - Przerwanie wątku przy użyciu stop() mogło skutkować naruszeniem zasad bezpieczeństwa, jeśli wątek był przerwany w momencie, gdy posiadał dostęp do zasobów, do których inne wątki nie powinny mieć dostępu.

Jak bezpiecznie zastąpić metodę stop()?

Zamiast używania stop(), należy implementować mechanizm łagodnego przerywania wątku, który pozwala wątkowi zakończyć pracę w sposób kontrolowany i bezpieczny. Oto ogólny sposób, jak to zrobić:

1. **Użycie flagi kontrolnej:**

    - Wątek powinien okresowo sprawdzać wartość specjalnej flagi kontrolnej (np. volatile boolean), która informuje, czy powinien zakończyć swoje działanie.

2. **Zakończenie wątku w sposób kontrolowany:**

    - Wątek, gdy zauważy, że flaga została ustawiona, powinien dokończyć bieżącą operację (np. zwolnić zasoby, zamknąć pliki) i zakończyć swoją pracę w sposób uporządkowany.

Przykład:

java Skopiuj kod

```java
public class SafeThread implements Runnable {
    private volatile boolean running = true;

    public void run() {
        while (running) {
            try {
                // Wykonywanie zadania w pętli
                System.out.println("Thread is running...");
                Thread.sleep(1000); // Symulacja pracy wątku
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt(); // Przywrócenie flagi przerwania
                break; // Wyjście z pętli, zakończenie pracy wątku
            }
        }
        System.out.println("Thread has stopped safely.");
    }

    public void stopRunning() {
        running = false;
    }

    public static void main(String[] args) throws InterruptedException {
        SafeThread safeThread = new SafeThread();
        Thread thread = new Thread(safeThread);
        thread.start();

        Thread.sleep(5000); // Pozwól wątkowi działać przez chwilę

        safeThread.stopRunning(); // Bezpieczne zatrzymanie wątku
        thread.join(); // Czekaj na zakończenie wątku
    }
}
```

Wyjaśnienie:

- **Flaga volatile boolean running:** Jest to flaga kontrolna, którą wątek sprawdza w każdej iteracji swojej pętli. Gdy wartość flagi zostanie ustawiona na false, wątek kończy swoją pracę w sposób kontrolowany.

- **stopRunning()**: Ta metoda ustawia flagę running na false, co informuje wątek, że powinien zakończyć działanie.

- **Thread.sleep() i InterruptedException:** Użycie Thread.sleep() symuluje zadania wątku. Gdy wątek zostanie przerwany (InterruptedException), wątek obsługuje to wyjątkiem, co pozwala na jego bezpieczne zakończenie.

Podsumowanie

Metoda stop() z klasy Thread w Java została uznana za przestarzałą z powodu problemów z bezpieczeństwem, stabilnością i przewidywalnością kodu. Zamiast niej, zaleca się użycie mechanizmu łagodnego przerywania wątku, który polega na użyciu flagi kontrolnej i odpowiedniego zarządzania zasobami, aby zapewnić bezpieczne i kontrolowane zakończenie pracy wątku.

D) Zaawansowana grafika komputerowa

21) Filtry konwolucyjne dla obrazów rastrowych.

Filtry konwolucyjne są podstawowymi narzędziami używanymi w przetwarzaniu obrazów, szczególnie w operacjach takich jak wykrywanie krawędzi, wygładzanie, wyostrzanie i wiele innych. Stosuje się je na obrazach rastrowych, które są reprezentowane jako dwuwymiarowe tablice pikseli.

Co to jest filtr konwolucyjny?

Filtr konwolucyjny to macierz (zwana też maską lub jądrem), która jest przesuwana po obrazie, dokonując operacji matematycznej zwanej **konwolucją** na każdej grupie pikseli (często zwanej oknem lub sąsiedztwem). Wynik tej operacji jest zapisywany jako nowa wartość piksela w obrazie wyjściowym.

Jak działa konwolucja na obrazie?

1. **Macierz jądra (kernel)**: Jest to mała macierz, np. 3x3, 5x5, która jest stosowana na każdej pozycji obrazu.
2. **Przesuwanie jądra**: Jądro jest przesuwane po obrazie od lewego górnego do prawego dolnego rogu.
3. **Mnożenie elementów jądra przez wartości pikseli**: W każdym punkcie jądro jest nakładane na odpowiadającą mu grupę pikseli obrazu, a każdy element jądra jest mnożony przez odpowiadającą mu wartość piksela.
4. **Sumowanie**: Wyniki mnożenia są sumowane, a ta suma jest przypisywana do piksela w nowym obrazie.
5. **Przesunięcie**: Jądro przesuwa się do następnej pozycji (zwykle o jeden piksel) i proces jest powtarzany.

Przykłady popularnych filtrów konwolucyjnych

1. **Filtr wygładzający (blur filter)**:
   - **Cel:** Zmniejsza szum i detale, wygładzając obraz.
   - **Przykładowe jądro 3x3:**

$$\frac{1}{9}\begin{pmatrix}1 & 1 & 1\\ 1 & 1 & 1\\ 1 & 1 & 1\end{pmatrix}$$

   - **Efekt:** Każdy piksel w obrazie jest zastępowany średnią z pikseli w jego sąsiedztwie.
2. **Filtr wykrywający krawędzie (edge detection filter)**:
   - **Cel:** Wykrywanie krawędzi, czyli miejsc, gdzie następuje gwałtowna zmiana intensywności pikseli.
   - **Przykładowe jądro Sobela w kierunku poziomym**

$$\begin{pmatrix}-1 & 0 & 1\\ -2 & 0 & 2\\ -1 & 0 & 1\end{pmatrix}$$

   - **Efekt:** Wartości pikseli zmieniają się gwałtownie tam, gdzie są krawędzie, co pozwala je wykryć.
3. **Filtr wyostrzający (sharpening filter)**:
   - **Cel:** Wzmacnia krawędzie i zwiększa kontrast na obrazie.
   - **Przykładowe jądro 3x3:**

$$\begin{pmatrix}0 & -1 & 0\\ -1 & 5 & -1\\ 0 & -1 & 0\end{pmatrix}$$

   - **Efekt:** Krawędzie stają się bardziej wyraźne, a obraz bardziej kontrastowy.

Zastosowanie filtrów konwolucyjnych

Filtry konwolucyjne są powszechnie stosowane w wielu dziedzinach przetwarzania obrazów:

- **Wykrywanie krawędzi:** Stosowane w systemach rozpoznawania obiektów, detekcji ruchu, analizy obrazów medycznych.
- **Wygładzanie obrazów:** Używane do redukcji szumu, poprawy jakości obrazów i przygotowania danych do dalszej analizy.
- **Wyostrzanie obrazów:** Przydatne w poprawie jakości zdjęć, usuwaniu rozmycia, przygotowaniu obrazów do wydruku lub publikacji.

Podsumowanie

Filtry konwolucyjne są kluczowym narzędziem w przetwarzaniu obrazów, pozwalającym na modyfikację i analizę obrazów rastrowych. Dzięki różnym rodzajom filtrów, można uzyskać różne efekty, takie jak wygładzanie, wyostrzanie czy wykrywanie krawędzi, co czyni je niezwykle wszechstronnymi w aplikacjach komputerowego przetwarzania obrazów.

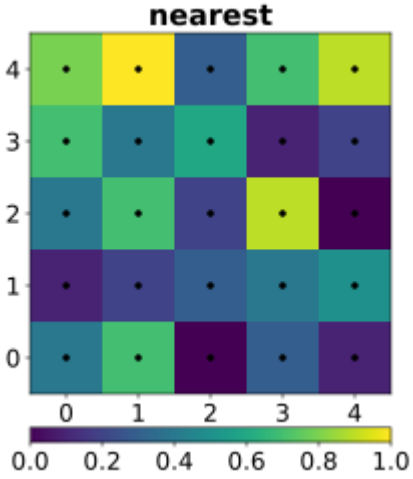
22) Skalowanie obrazów rastrowych (resampling).

Skalowanie obrazów rastrowych to proces zmiany rozmiaru obrazu poprzez dodawanie lub usuwanie pikseli. Proces ten jest znany również jako **resampling** i jest kluczowy w wielu aplikacjach graficznych, od prostych edytorów obrazów po zaawansowane systemy przetwarzania obrazu.

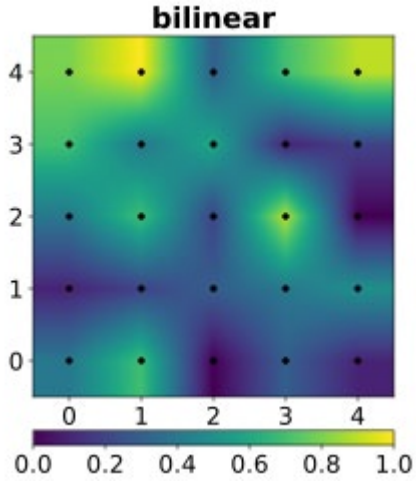
Dlaczego skalowanie obrazów jest trudne?

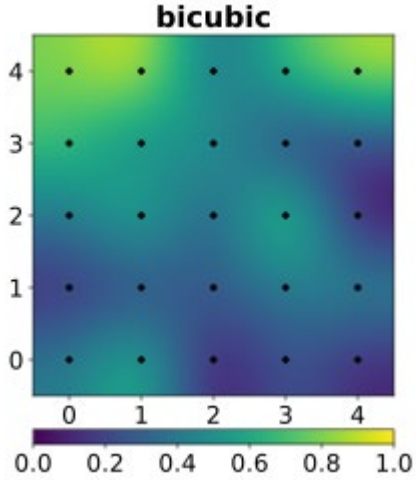
Obraz rastrowy jest zbiorem pikseli, które tworzą siatkę prostokątną. Każdy piksel ma określony kolor i intensywność, które razem tworzą wizualny obraz. Podczas skalowania obrazu, szczególnie gdy zmieniamy jego rozmiar znacząco (np. powiększamy lub zmniejszamy o dużą wartość), musimy dokonać interpolacji, aby uzyskać odpowiednie wartości dla nowych pikseli.

Metody resamplingu (interpolacji)

1. **Najbliższy sąsiad (Nearest Neighbor):**
   - **Opis:** Jest to najprostsza i najszybsza metoda resamplingu. Każdemu pikselowi w nowym obrazie przypisywana jest wartość piksela z oryginalnego obrazu, który jest najbliżej jego pozycji.
   - **Zalety:** Szybkość i prostota.
   - **Wady:** Może prowadzić do widocznych artefaktów, takich jak schodki (efekt "schodków" na krawędziach), szczególnie podczas powiększania.
   - 

2. **Interpolacja dwuliniowa (Bilinear Interpolation):**
   - **Opis:** Każdemu nowemu pikselowi przypisywana jest wartość będąca średnią ważoną czterech najbliższych pikseli w oryginalnym obrazie. Wartości te są interpolowane liniowo w dwóch wymiarach.
   - **Zalety:** Zapewnia lepszą jakość niż metoda najbliższego sąsiada, gładkość obrazu.
   - **Wady:** Większe rozmycie niż przy najbliższym sąsiedzie, ale mniej niż przy bardziej zaawansowanych metodach.

- 

3. **Interpolacja dwusześcienna (Bicubic Interpolation):**
   - **Opis:** Jest to bardziej zaawansowana metoda, w której wartość nowego piksela jest obliczana na podstawie 16 sąsiednich pikseli. Interpolacja jest bardziej skomplikowana i uwzględnia kształt funkcji sześciennej.
   - **Zalety:** Daje bardzo dobre rezultaty, zachowując więcej szczegółów i ostrych krawędzi w porównaniu do interpolacji biliniowej.
   - **Wady:** Jest bardziej czasochłonna i obciążająca obliczeniowo.
   - 

4. **Interpolacja Lanczos (Lanczos Resampling):**
   - **Opis:** Używa funkcji sinc jako jądra do interpolacji, co umożliwia precyzyjniejsze odwzorowanie szczegółów obrazu. Często wykorzystywana przy zaawansowanym przetwarzaniu obrazów.
   - **Zalety:** Może dawać bardzo wysoką jakość obrazu, szczególnie przy dużych zmianach rozmiaru.
   - **Wady:** Wysoka złożoność obliczeniowa, co może być problematyczne przy dużych obrazach.

Wybór metody resamplingu

Wybór metody resamplingu zależy od specyficznych wymagań:

- **Najbliższy sąsiad**: Stosowany głównie w przypadku, gdy szybkość jest ważniejsza niż jakość, np. w małych aplikacjach graficznych lub przy tworzeniu prostych miniatur.
- **Interpolacja biliniowa**: Dobry kompromis między jakością a szybkością, odpowiednia dla większości standardowych zadań związanych z przeskalowaniem obrazu.
- **Interpolacja bikubicza**: Preferowana metoda, gdy jakość jest kluczowa, np. w edytorach zdjęć czy zaawansowanych systemach graficznych.
- **Interpolacja Lanczos**: Używana w zaawansowanym przetwarzaniu obrazów, gdzie wymagany jest najwyższy poziom szczegółowości.

Przykład zastosowania

Wyobraźmy sobie skalowanie zdjęcia o wymiarach 4000x3000 pikseli do miniatury 400x300 pikseli:

- **Najbliższy sąsiad** da szybki, ale często pikselowany obraz.
- **Interpolacja biliniowa** wygładzi obraz, ale może wprowadzić lekkie rozmycie.
- **Interpolacja bikubicza** zapewni ostrzejsze krawędzie i lepszą jakość.
- **Interpolacja Lanczos** może zachować najwięcej szczegółów, ale kosztem dłuższego czasu przetwarzania.

Podsumowanie

Skalowanie obrazów rastrowych (resampling) jest niezbędnym narzędziem w przetwarzaniu grafiki, umożliwiającym zmianę rozmiaru obrazów przy minimalnej utracie jakości. Wybór odpowiedniej metody interpolacji jest kluczowy, aby uzyskać pożądany efekt końcowy, zależnie od wymagań dotyczących jakości, szczegółowości oraz wydajności.

23) Pojęcie grafu sceny.

Graf sceny (ang. **scene graph**) to struktura danych, która jest szeroko stosowana w grafice komputerowej, szczególnie w renderowaniu trójwymiarowych scen. Reprezentuje on hierarchiczną organizację obiektów w scenie, umożliwiając efektywne zarządzanie i manipulowanie elementami, takimi jak modele, światła, kamery, efekty i inne elementy sceny.

Kluczowe cechy grafu sceny

1. **Hierarchia obiektów:**
    - Graf sceny organizuje obiekty w strukturę drzewiastą, gdzie każdy węzeł reprezentuje pewien obiekt lub grupę obiektów. Węzły mogą mieć "dzieci", co pozwala na tworzenie złożonych hierarchii.
    - Na przykład, postać w grze może być reprezentowana jako drzewo w grafie sceny, gdzie korzeń to postać, a gałęzie to części ciała, takie jak głowa, ręce i nogi.
2. **Dziedziczenie transformacji:**
    - W grafie sceny transformacje (np. translacja, rotacja, skalowanie) są dziedziczone w hierarchii. Oznacza to, że jeśli przesuniemy węzeł rodzica, wszystkie jego dzieci również zostaną przesunięte.
    - Przykład: Jeśli przesuniemy korzeń reprezentujący postać, wszystkie części ciała (dzieci) zostaną automatycznie przesunięte razem z nim.
3. **Podział sceny na logiczne grupy:**
    - Graf sceny umożliwia grupowanie obiektów, co ułatwia zarządzanie złożonymi scenami. Grupy mogą być tworzone według różnych kryteriów, np. obiekty należące do tego samego poziomu w grze, obiekty, które mają być renderowane z tym samym materiałem itp.
4. **Optymalizacja renderowania:**
    - Graf sceny pozwala na optymalizację procesu renderowania. Na przykład, jeśli cała gałąź grafu (czyli grupa obiektów) znajduje się poza polem widzenia kamery, można pominąć jej renderowanie, co oszczędza zasoby obliczeniowe.
    - Techniki takie jak culling (usuwanie niewidocznych obiektów) są łatwe do implementacji dzięki strukturze grafu sceny.

Przykład użycia grafu sceny

Rozważmy prostą scenę składającą się z:

- Słońca
- Ziemi, która obraca się wokół Słońca
- Księżyca, który obraca się wokół Ziemi

Graf sceny może wyglądać tak:

```markdown
- Słońce (root)
  - Ziemia (dziecko Słońca)
    - Księżyc (dziecko Ziemi)
```

- **Transformacje:** Jeśli Słońce się przesunie, Ziemia i Księżyc przesuną się razem z nim. Jeśli Ziemia się obróci, Księżyc także się obróci, ponieważ jest jej dzieckiem.

Zastosowanie grafów scen

Grafy scen są wykorzystywane w wielu dziedzinach, takich jak:

- **Gry komputerowe:** Do zarządzania złożonymi scenami, postaciami i obiektami.
- **Symulacje:** W symulacjach fizycznych i wirtualnych światach, gdzie obiekty mogą być dynamicznie dodawane i usuwane.
- **Systemy CAD:** Do organizacji i manipulowania modelami inżynierskimi.

Podsumowanie

Graf sceny jest kluczową strukturą danych w grafice komputerowej, która pozwala na hierarchiczne zarządzanie obiektami w trójwymiarowej scenie. Dzięki możliwości **dziedziczenia** transformacji, grupowania obiektów i optymalizacji renderowania, grafy scen są niezastąpione w tworzeniu złożonych i wydajnych aplikacji graficznych.

E) Projektowanie obiektowe

27)Jaka jest różnica między testami wydajnościowymi, obciążeniowych oraz stres testami?

Testy wydajnościowe, obciążeniowe oraz stres testy są różnymi rodzajami testów, które mają na celu ocenę zachowania systemu pod różnymi warunkami. Każdy z tych testów skupia się na innym aspekcie działania systemu. Oto ich główne różnice:

1. Testy wydajnościowe (Performance Testing)

Cel: Testy wydajnościowe mają na celu zmierzenie, jak szybko i efektywnie system działa pod określonym obciążeniem. Celem jest sprawdzenie, czy system spełnia wymagania wydajnościowe w normalnych warunkach operacyjnych.

Zakres:

- Określenie czasu odpowiedzi na żądania.
- Sprawdzenie przepustowości systemu.
- Ocena zużycia zasobów (CPU, pamięć, I/O).
- Monitorowanie wydajności systemu przy różnej liczbie użytkowników lub operacji.

Przykład: Testowanie aplikacji internetowej, aby upewnić się, że strona ładuje się w mniej niż 2 sekundy, gdy korzysta z niej 1000 użytkowników jednocześnie.

2. Testy obciążeniowe (Load Testing)

Cel: Testy obciążeniowe sprawdzają, jak system zachowuje się pod określonym, zwykle maksymalnym przewidywanym obciążeniem. Celem jest zidentyfikowanie punktu, w którym system zaczyna tracić wydajność lub przestaje funkcjonować poprawnie.

Zakres:

- Symulacja maksymalnej liczby użytkowników lub operacji, które system ma obsługiwać.
- Określenie, jak długo system może utrzymać wydajność przy dużym obciążeniu.
- Identyfikacja wąskich gardeł w systemie przy dużym obciążeniu.

Przykład: Symulowanie 10 000 użytkowników jednocześnie dokonujących transakcji w systemie bankowym, aby sprawdzić, czy system działa poprawnie i jakie są jego granice.

3. Testy przeciążeniowe (Stress Testing)

Cel: Stres testy mają na celu sprawdzenie, jak system zachowuje się poza normalnymi granicami obciążenia. Celem jest ocenienie, jak system reaguje na przeciążenie i jak odzyskuje sprawność po awarii.

Zakres:

- Symulacja ekstremalnych warunków, które wykraczają poza normalne użytkowanie (np. bardzo duża liczba użytkowników, brak zasobów systemowych).
- Identyfikacja, gdzie i jak system ulega awarii.
- Ocena stabilności systemu i jego zdolności do odzyskiwania sprawności po awarii.

Przykład: Przeprowadzenie testu, w którym liczba użytkowników korzystających z aplikacji internetowej gwałtownie rośnie do poziomu, który znacznie przekracza przewidywaną maksymalną liczbę użytkowników, aby zobaczyć, jak system się zachowuje (np. przy 100 000 użytkowników jednocześnie).

Podsumowanie

- **Testy wydajnościowe** oceniają, jak dobrze system działa pod normalnym obciążeniem.
- **Testy obciążeniowe** sprawdzają, jak system radzi sobie z maksymalnym przewidywanym obciążeniem.
- **Stres testy** badają, jak system zachowuje się w ekstremalnych warunkach, które przekraczają jego normalne możliwości.

Każdy z tych testów ma na celu zapewnienie, że system jest odpowiednio przygotowany do działania w różnych scenariuszach i że jego wydajność jest zgodna z oczekiwaniami.

28) Proszę opisać trzy wzorce behawioralne.

Wzorce behawioralne (ang. **behavioral patterns**) to kategoria wzorców projektowych, które zajmują się algorytmami oraz podziałem odpowiedzialności między obiektami. Mają one na celu ułatwienie komunikacji między obiektami w systemie, pozwalając na elastyczną zmianę zachowania programu w czasie jego działania. Oto opis trzech popularnych wzorców behawioralnych:

1. Wzorzec **Obserwator** (Observer)

Cel: Wzorzec Obserwator umożliwia stworzenie zależności "jeden do wielu" między obiektami, tak że zmiana stanu jednego obiektu (obserwowanego) powoduje automatyczne powiadomienie i aktualizację wszystkich jego zależnych obiektów (obserwatorów).

Zastosowanie:

- Kiedy zmiana w jednym obiekcie powinna prowadzić do aktualizacji innych obiektów.
- Kiedy liczba obserwatorów zmienia się dynamicznie w trakcie działania programu.

Przykład:

- W systemie GUI, zmiana w jednym komponencie interfejsu użytkownika (np. zmiana stanu przycisku) może powodować aktualizację wielu innych komponentów (np. etykiet, list, pól tekstowych).

Działanie:

- **Obiekt Obserwowany (Subject):** Przechowuje stan, do którego obserwatorzy mają dostęp, i zarządza listą obserwatorów.
- **Obiekt Obserwator (Observer):** Rejestruje się u obiektu obserwowanego i jest powiadamiany o zmianach.

```
interface Observer {
    void update();
}

class ConcreteObserver implements Observer {
    public void update() {
        System.out.println("Obserwator został powiadomiony!");
    }
}

class Subject {
    private List<Observer> observers = new ArrayList<>();

    public void addObserver(Observer observer) {
        observers.add(observer);
    }

    public void notifyObservers() {
        for (Observer observer : observers) {
            observer.update();
        }
    }

    public void changeState() {
        // Zmiana stanu
        notifyObservers();
    }
}
```

2.

Wzorzec **Strategia** (Strategy)

Cel: Wzorzec Strategia umożliwia definiowanie rodziny algorytmów, umieszczanie ich w oddzielnych klasach, a następnie używanie ich wymiennie. Wzorzec ten pozwala na zmianę algorytmu działania obiektu w czasie działania programu.

Zastosowanie:

- Kiedy mamy wiele wariantów algorytmu i chcemy wybrać jeden z nich w zależności od sytuacji.

- Kiedy chcemy uniknąć wielu instrukcji warunkowych (if-else, switch) związanych z wyborem algorytmu.

Przykład:

- W aplikacji do sortowania danych, możemy mieć różne strategie sortowania (np. quicksort, mergesort, bubblesort), które można zmieniać w zależności od rodzaju danych.

Działanie:

- **Kontekst (Context):** Klasa, która używa strategii do wykonania zadania.
- **Strategia (Strategy):** Interfejs lub klasa bazowa dla wszystkich konkretnych strategii.
- **Konkretny algorytm (Concrete Strategy):** Implementuje specyficzny algorytm, który może być użyty przez kontekst.

```
interface Strategy {
    void execute();
}

class ConcreteStrategyA implements Strategy {
    public void execute() {
        System.out.println("Wykonywanie strategii A");
    }
}

class ConcreteStrategyB implements Strategy {
    public void execute() {
        System.out.println("Wykonywanie strategii B");
    }
}

class Context {
    private Strategy strategy;

    public void setStrategy(Strategy strategy) {
        this.strategy = strategy;
    }

    public void executeStrategy() {
        strategy.execute();
    }
}
```

3. Wzorzec Komenda (Command)

Cel: Wzorzec Komenda przekształca żądanie w obiekt, co pozwala na parametryzację klientów z różnymi żądaniami, opóźnianie lub kolejkę żądań oraz wsparcie operacji cofania.

Zastosowanie:

- Kiedy chcemy zrealizować operacje undo/redo.

- Kiedy chcemy zrealizować polecenia jako obiekty, aby można je było przekazywać, kolejkują, rejestrować czy anulować.

Przykład:

- W edytorze tekstu każde działanie użytkownika (np. pisanie, usuwanie, formatowanie) może być traktowane jako komenda, którą można cofnąć lub powtórzyć.

Działanie:

- **Komenda (Command):** Interfejs deklarujący metodę execute.
- **Konkretny obiekt komendy (Concrete Command):** Implementacja konkretnej komendy.
- **Odbiorca (Receiver):** Obiekt, który rzeczywiście wykonuje żądanie.
- **Wywołujący (Invoker):** Obiekt, który inicjuje wykonanie komendy.

```
interface Command {
    void execute();
}

class Light {
    public void on() {
        System.out.println("Światło włączone");
    }

    public void off() {
        System.out.println("Światło wyłączone");
    }
}

class LightOnCommand implements Command {
    private Light light;

    public LightOnCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.on();
    }
}

class LightOffCommand implements Command {
    private Light light;

    public LightOffCommand(Light light) {
        this.light = light;
    }

    public void execute() {
        light.off();
    }
}

class RemoteControl {
    private Command command;

    public void setCommand(Command command) {
        this.command = command;
    }

    public void pressButton() {
        command.execute();
    }
}
```

Podsumowanie

- **Obserwator**: Umożliwia powiadamianie obiektów o zmianach stanu innego obiektu.
- **Strategia**: Pozwala na wybór algorytmu działania w czasie działania programu.

- **Komenda**: Umożliwia traktowanie poleceń jako obiektów, co pozwala na ich późniejsze wykonanie, anulowanie lub przechowywanie.

Te wzorce projektowe pomagają w tworzeniu elastycznych i łatwych do utrzymania systemów, umożliwiając zarządzanie różnorodnymi scenariuszami zachowań obiektów w programie.

29) Jak działa prototyp w języku JavaScript?

W JavaScript prototyp to mechanizm umożliwiający obiektom dziedziczenie właściwości i metod od innych obiektów. Każdy obiekt w JavaScript ma ukryte odniesienie do swojego prototypu, który jest innym obiektem. Ten układ tworzy tzw. **łańcuch prototypów** (prototype chain), który jest przeszukiwany przez JavaScript, gdy próbujesz uzyskać dostęp do właściwości lub metody obiektu.

Jak działa prototyp?

1. **Łańcuch prototypów:**
   - Gdy odwołujesz się do właściwości lub metody na obiekcie, JavaScript najpierw sprawdza, czy właściwość/metoda istnieje na tym obiekcie. Jeśli nie, przeszukuje jego prototyp. Jeśli prototyp też jej nie zawiera, JavaScript przechodzi do prototypu tego prototypu, i tak dalej, aż do końca łańcucha, gdzie znajduje się null.
2. **Tworzenie prototypów:**
   - Gdy tworzysz obiekt przy użyciu funkcji konstruktora (np. za pomocą new), nowo utworzony obiekt automatycznie dziedziczy właściwości i metody z prototypu tego konstruktora.

```
function Person(name) {
    this.name = name;
}

Person.prototype.greet = function() {
    console.log("Hello, my name is " + this.name);
};

const john = new Person("John");
john.greet(); // "Hello, my name is John"
```

   - W tym przykładzie obiekt john korzysta z metody greet, która jest zdefiniowana w prototypie konstruktora Person.
3. **Dynamiczne dodawanie metod:**
   - Możesz dynamicznie dodawać nowe metody lub właściwości do prototypu, co sprawia, że są one dostępne dla wszystkich instancji obiektu, które dziedziczą z tego prototypu.

```
Person.prototype.sayGoodbye = function() {
    console.log("Goodbye from " + this.name);
};

john.sayGoodbye(); // "Goodbye from John"
```

4. **Dziedziczenie prototypów:**
    - Możliwe jest dziedziczenie prototypów między różnymi konstruktorami. Na przykład, jeśli chcesz, aby Employee dziedziczył od Person, możesz ustawić Employee.prototype na nowy obiekt stworzony z Person.prototype.

```
function Employee(name, jobTitle) {
    Person.call(this, name);
    this.jobTitle = jobTitle;
}

Employee.prototype = Object.create(Person.prototype);
Employee.prototype.constructor = Employee;

const jane = new Employee("Jane", "Engineer");
jane.greet(); // "Hello, my name is Jane"
```

    - W powyższym przykładzie jane dziedziczy metodę greet z prototypu Person.

Podsumowanie

Prototyp w JavaScript to potężny mechanizm umożliwiający obiektom dziedziczenie właściwości i metod od innych obiektów poprzez łańcuch prototypów. Dzięki temu możliwe jest tworzenie obiektów, które mogą współdzielić zachowanie w sposób dynamiczny i efektywny. Prototypy są podstawą dziedziczenia w JavaScript i pozwalają na tworzenie złożonych struktur obiektowych oraz elastycznego, wielokrotnego użytku kodu.

4o

30) Proszę opisać cztery podstawowe typy Singletona.

Singleton Lazy Initialization (Opóźnione Tworzenie):

Wzorzec ten polega na opóźnionym tworzeniu instancji Singletona, tj. instancja jest tworzona dopiero w momencie, gdy jest pierwszy raz żądane jej użycie.

Jest to prosty sposób na oszczędzanie zasobów, ponieważ instancja jest tworzona tylko wtedy, gdy jest naprawdę potrzebna.

Jednak może prowadzić do problemów z wielowątkowością, jeśli wiele wątków próbuje uzyskać dostęp do Singletona jednocześnie.

Singleton Eager Initialization (Natychmiastowe Tworzenie):

W tym przypadku instancja Singletona jest tworzona natychmiast po załadowaniu klasy.

Jest to prosty sposób na zapewnienie, że istnieje tylko jedna instancja Singletona od początku działania programu.

Może być mniej wydajny, jeśli Singleton nie jest używany wcale lub jest używany tylko w późniejszym etapie działania programu.

Singleton Thread-Safe (Singleton z Bezpieczeństwem Wielowątkowości):

Ten rodzaj Singletona zapewnia bezpieczeństwo wielowątkowości, co oznacza, że może być bezpiecznie używany w środowiskach wielowątkowych.

Najczęściej wykorzystuje mechanizmy takie jak blokady (locks) lub mechanizmy inicjalizacji leniwej w sposób bezpieczny.

Jest to ważne, gdy wiele wątków może próbować uzyskać dostęp do Singletona jednocześnie. Singleton z Wykorzystaniem Enumeracji:

Wzorzec Singletona można również zaimplementować przy użyciu typu wyliczeniowego (enum) w niektórych językach programowania.
Enumeracje są w językach takich jak Java gwarantowanej jedności, co sprawia, że jest to prosty i bezpieczny sposób na implementację Singletona.
Jest to najbezpieczniejsza i najprostsza metoda implementacji Singletona.

31) Czym jest dług technologiczny?

Dług technologiczny to pojęcie używane w kontekście programowania i zarządzania projektami informatycznymi, które odnosi się do konsekwencji podejmowania **krótkoterminowych** decyzji, które ułatwiają szybkie dostarczenie produktu lub funkcji, ale mogą prowadzić do większych problemów w **przyszłości**.

Oto kilka kluczowych aspektów długu technologicznego:

1. **Kompleksowość kodu**: Szybkie wprowadzenie nowej funkcji może wymagać pisania kodu, który jest trudny do utrzymania, testowania lub rozszerzania w przyszłości. Przykładem może być brak odpowiedniej dokumentacji, nadmiarowy lub zduplikowany kod, brak testów jednostkowych.

2. **Krótko- vs. długoterminowe decyzje**: Decyzje projektowe podejmowane z myślą o szybkim dostarczeniu rozwiązania mogą prowadzić do sytuacji, gdzie w przyszłości konieczne będzie przeprojektowanie, refaktoryzacja lub przepisywanie dużych części kodu. To z kolei generuje dodatkowe koszty czasowe i finansowe.

3. **Wpływ na rozwój i utrzymanie systemu**: Im dłużej ignorowany jest dług technologiczny, tym trudniej jest go spłacić. Systemy mogą stawać się coraz trudniejsze w utrzymaniu i rozwijaniu, co może wpływać na zdolność zespołu do szybkiego reagowania na zmiany rynkowe lub potrzeby klientów.

4. **Spłacanie długu technologicznego**: Spłata długu technologicznego polega na poświęceniu czasu i zasobów na refaktoryzację kodu, poprawę architektury systemu, dodanie brakujących testów i inne działania, które zwiększają jakość i stabilność systemu.

W skrócie, dług technologiczny jest jak zaciągnięcie pożyczki: daje natychmiastową korzyść w postaci szybkiego postępu, ale z czasem może prowadzić do większych problemów, jeśli nie zostanie odpowiednio zarządzany i spłacony

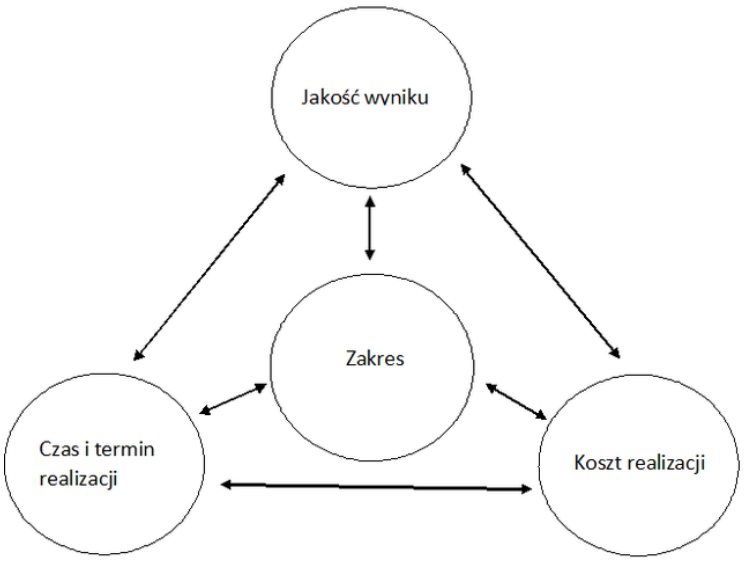
F) Zarządzanie projektami

32) Proszę podać najważniejsze współzależne parametry projektu

Żelazny trójkąt, znany również jako złoty trójkąt, to fundamentalny model zarządzania projektami, który opisuje wzajemne zależności między trzema kluczowymi parametrami: zakresem, czasem i kosztami. Często do tego modelu dodaje się jakość jako czwarty, równie istotny element.

- **Zakres** definiuje, co dokładnie ma zostać wykonane w ramach projektu.
- **Czas** odnosi się do harmonogramu, czyli jak długo projekt będzie realizowany.
- **Koszty** obejmują wszystkie zasoby finansowe potrzebne do realizacji projektu.
- **Jakość** dotyczy spełnienia określonych standardów i wymagań przez końcowy produkt.

Zmiana jednego z tych elementów wpływa na pozostałe. Na przykład, rozszerzenie zakresu projektu zazwyczaj prowadzi do zwiększenia kosztów i wydłużenia czasu realizacji. Kierownik projektu odpowiada za utrzymanie równowagi między tymi parametrami, co jest kluczowe dla sukcesu projektu.
Dodatkowo, **Zasoby** (np. ludzkie, technologiczne) również odgrywają istotną rolę w planowaniu i realizacji projektu, wpływając na koszt, jakość i czas jego wykonania. Zarządzanie tymi współzależnymi parametrami jest podstawowym zadaniem w każdym projekcie, a brak równowagi między nimi może prowadzić do jego niepowodzenia.



33)Proszę podać etapy procesu zarządzania projektami

Etapy procesu zarządzania projektami obejmują sekwencję działań, które pomagają w planowaniu, realizacji i zakończeniu projektu. Poniżej przedstawiam kluczowe etapy tego procesu:

1)Inicjacja (Initiation):

- **Cel**: Określenie potrzeby lub problemu, który ma być rozwiązany przez projekt. Na tym etapie definiuje się wstępne cele, zakres oraz interesariuszy projektu.
- **Działania**: Przygotowanie dokumentu inicjującego projekt, ocena wykonalności projektu, zidentyfikowanie interesariuszy oraz wstępne określenie budżetu i harmonogramu.

2)Planowanie (Planning):

- **Cel**: Opracowanie szczegółowego planu działania, który określi, jak projekt zostanie zrealizowany, w tym harmonogram, budżet, zasoby i ryzyka.
- **Działania**: Tworzenie planu zarządzania zakresem, harmonogramem, kosztami, jakością, zasobami, komunikacją, ryzykiem oraz zaopatrzeniem. Określenie kamieni milowych, kryteriów sukcesu oraz sposobu monitorowania postępów.

3)Realizacja (Execution):

- **Cel**: Wykonanie zaplanowanych działań i dostarczenie produktów lub wyników projektu zgodnie z ustalonymi kryteriami.

- **Działania**: Zarządzanie zespołem projektowym, alokacja zasobów, realizacja zadań zgodnie z harmonogramem, monitorowanie postępów oraz komunikacja z interesariuszami. W razie potrzeby wprowadzanie korekt do planu.

4)Monitorowanie i kontrola (Monitoring and Controlling):

- **Cel**: Nadzór nad realizacją projektu w celu zapewnienia zgodności z planem oraz wprowadzanie niezbędnych zmian.
- **Działania**: Regularne śledzenie postępów, zarządzanie zmianami w projekcie, monitorowanie ryzyk, kontrola budżetu i harmonogramu, raportowanie stanu projektu. Ocena wyników i podejmowanie działań korygujących w razie potrzeby.

5)Zamknięcie (Closing):

- **Cel**: Formalne zakończenie projektu, ocena osiągniętych wyników i rozliczenie zasobów.
- **Działania**: Przekazanie końcowego produktu lub usługi interesariuszom, zakończenie kontraktów, dokumentowanie wyników projektu, przeprowadzenie retrospektywy (lessons learned), rozliczenie zasobów, zamknięcie budżetu oraz archiwizacja dokumentacji projektu.

Każdy z tych etapów jest kluczowy dla zapewnienia, że projekt zostanie zrealizowany zgodnie z założeniami, w ramach określonego czasu, budżetu i zakresu, a także spełni oczekiwania interesariuszy.

34) Proszę wymienić wartości Manifestu Agile

Ludzi i interakcje ponad procesami i narzędziami.
Działające oprogramowanie ponad obszerną dokumentacją.
Współpracę z klientem ponad negocjowaniem umowy.
Reagowanie na zmianę ponad realizację założonego planu.

G)Metody statystyczne

35) Stan stacjonarny w markowowskich procesach stochastycznych - metody wyznaczania i zastosowania.

Metody wyznaczania:

1. Rozwiazanie równania $\pi = \pi * P$. Gdzie P to macierz prawdopodobieństwa przejść miedzy stanami

$$P = \begin{pmatrix} 0.7 & 0.2 & 0.1 \\ 0.1 & 0.6 & 0.3 \\ 0.1 & 0.2 & 0.7 \end{pmatrix}$$

Gdzie π to wektor stanu stacjonarnego, którego chcemy szukać. Otrzymujemy układ równań, z którego możemy znaleźć π.

2. Metoda Monte Carlo

Możemy przeprowadzić symulację, w której zaczynamy od pewnego stanu i korzystamy z matrycy P do przejścia do następnego stanu zgodnie z określonymi prawdopodobieństwami. Po wielu iteracjach obliczamy, jaką część czasu system spędził w każdym ze stanów, co da nam estymację wektora π.

3. Iteracyjne metody numeryczne
Metoda potęgowa polega na wielokrotnym mnożeniu dowolnego wektora przez matrycę P
aż do zbiegnięcia do wektora stacjonarnego π.

Zastosowanie łańcuchów Markova:

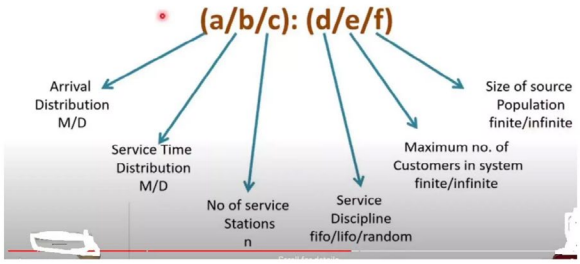
- Modelowanie: Za pomocą łańcuchów Markowa można skutecznie modelować wiele naturalnych procesów i struktur. Na przykład modelując w ten sposób język naturalny można zbudować algorytm kompresji tekstu. Alternatywnie, modeli takich można użyć do generowania losowych tekstów. Modele Markowa pojawiają się też bardzo często w biologii obliczeniowej.

  • PageRank: Imponującym zastosowaniem łańcuchów Markowa jest stworzony przez firmę Google algorytm szeregowania stron PageRank. Algorytm ten bazuje na łańcuchu Markowa, który jest modelem procesu poruszania się użytkownika po zbiorze wszystkich (znanych systemowi) stron WWW.

36) Systemy kolejkowe - definicja, typy, notacja Kendalla, współdzielenie procesora, prawo Little'a, przykłady systemów kolejkowych

Kendall notation

is **the standard system used to describe and classify a queueing node**.



Definicja:
W systemach którymi zajmuje się teoria kolejek, zlecenia (np. klienci w supermarkecie) napływają do punktów obsługi (kasy) i czekają na obsłużenie w poczekalni (kolejka do kasy).

Notacja Kendalla:

Kendala powstała w 1953 roku i dotyczyła notacji, jaką należy stosować przy opisywaniu systemów kolejkowych. Zaproponowana przez Kendala notacja wyglądała następująco: $A/B/c/L/N$. Poszczególne składowe tej notacji miały następujące znaczenie:

- A – rozkład zmiennej losowej τ_1 (czas między kolejnymi zgłoszeniami),
  - M – rozkład wykładniczy,
  - D – rozkład deterministyczny (o stałych odstępach czasu),
  - E_k – rozkład Erlanga rzędu k,
  - G – rozkład ogólny (od General) zdefiniowany przez użytkownika,
- B – algorytm obsługi zgłoszeń (rozkład zmiennej losowej τ_2),
- c – liczba równoległych stanowisk obsługi,
- L – wielkość poczekalni (jeśli nie podane, to $L = \infty$),
- N – wymiar źródła zgłoszeń (jeśli nie podane, to $N = \infty$).

Typy:

- **M/M/1** - system kolejkowy, w którym rozkład czasu pomiędzy kolejnymi zgłoszeniami do systemu oraz rozkład czasu obsługi pojedynczego zgłoszenia są rozkładami wykładniczymi, istnieje jedno stanowisko obsługi i nieskończenie długa kolejka.
- **M/M/c** – system kolejkowy, w którym rozkład czasu pomiędzy kolejnymi zgłoszeniami do systemu oraz rozkład czasu obsługi pojedynczego zgłoszenia są rozkładami wykładniczymi,

istnieje stanowisk obsługi i nie ma kolejki do systemu (każde zgłoszenie przybyłe w momencie, gdy wszystkie stanowiska obsługi są zajęte, jest odrzucane)

- **M/M/1/L** – system kolejkowy, w którym rozkład czasu pomiędzy kolejnymi zgłoszeniami do systemu oraz rozkład czasu obsługi pojedynczego zgłoszenia są rozkładami wykładniczymi, istnieje jedno

stanowisko obsługi i istnieje kolejka, która ma długość

M/M/1/N – system kolejkowy, w którym rozkład czasu pomiędzy kolejnymi zgłoszeniami do systemu oraz rozkład czasu obsługi pojedynczego zgłoszenia są rozkładami wykładniczymi, istnieje jedno

stanowisko obsługi i źródło zgłoszeń ma skończony wymiar równy

współdzielenie procesora:

??????

chat gpt:

W kontekście systemów kolejkowych, współdzielenie procesora odnosi się do możliwości równoczesnej obsługi wielu zadań przez jeden serwer (procesor). Jest to często spotykane w systemach operacyjnych i sieciach komputerowych.

prawo Little'a

Prawo Little'a jest wyrażone wzorem:

$L = \lambda W$

gdzie:

L = średnia liczba rzeczy/klientów w systemie/kolejce;

λ = średnie tempo przybywania (intensywność napływu zgłoszeń);

W = średni czas przebywania w systemie

Przykład: jeżeli oddział banku odwiedza średnio 10 klientów na godzinę (λ) i klie

przykłady systemów kolejkowych

Przykłady Systemów Kolejkowych

- System kolejkowy w supermarkecie - M/M/c, gdzie c to liczba kas.
- Serwer WWW - G/G/1 lub M/M/1, w zależności od charakterystyki ruchu i czasu obsługi.
- Transport Publiczny - Można modelować jako system kolejkowy z różnymi stacjami jako serwerami. - Systemy Produkcyjne - Mogą być modelowane jako serie systemów kolejkowych, gdzie każdy etap produkcji jest jednym systemem.

37)Modele z ukrytym procesem Markowa - definicja i zastosowanie. Ogólna zasada działania algorytmu Viterbiego.

Definicja
Modelu z Ukrytym Procesem Markowa (Hidden Markov Model, HMM)
Model z ukrytym procesem Markowa (HMM) to statystyczny model, który jest rozwinięciem standardowego modelu procesu Markowa. W HMM, obserwacje są generowane przez ukryte stany, które są niewidoczne (ukryte) dla obserwatora. Tylko obserwacje są bezpośrednio mierzalne, ale ukryte stany nie są. Struktura modelu pozwala na estymację ukrytych stanów oraz parametrów modelu na podstawie obserwowanych danych.

Model HMM jest opisany przez:

Alfabet obserwacji - zbiór wszystkich możliwych obserwacji.
Alfabet stanów - zbiór wszystkich ukrytych stanów.
Macierz przejść stanów - prawdopodobieństwa przejścia między ukrytymi stanami.
Macierz prawdopodobieństwa emisji - prawdopodobieństwa generowania obserwacji z każdego ukrytego stanu.
Rozkład początkowy stanów - prawdopodobieństwa początkowych stanów ukrytych.

Przykład z pogodą i obserwacjami
Wyobraź sobie, że masz przyjaciela, który codziennie rano wybiera jeden z dwóch sposobów ubioru: nosi kurtkę lub koszulkę. Nie wiesz, jaka jest pogoda w jego mieście (to jest ukryty stan), ale wiesz, że jego wybór stroju zależy od pogody.
Stany ukryte:

- Słonecznie
- Deszczowo

Nie możesz bezpośrednio zobaczyć, jaka jest pogoda, ale wiesz, co twój przyjaciel nosi. To, co nosi, jest obserwacją, która zależy od pogody.
Obserwacje:

- Kurtka
- Koszulka

Przejścia między stanami:
Załóżmy, że jeśli dziś jest słonecznie, istnieje 80% szans, że jutro też będzie słonecznie, i 20% szans, że będzie deszczowo. Jeśli dziś jest deszczowo, istnieje 70% szans, że jutro również będzie deszczowo, i 30% szans, że będzie słonecznie. To nazywamy prawdopodobieństwami przejść między stanami.
Prawdopodobieństwa emisji:
Jeśli jest słonecznie, twój przyjaciel nosi koszulkę z prawdopodobieństwem 90%, a kurtkę z prawdopodobieństwem 10%. Jeśli jest deszczowo, nosi kurtkę z prawdopodobieństwem 80%, a koszulkę z prawdopodobieństwem 20%.
Zadanie:
Codziennie widzisz, co twój przyjaciel nosi, ale nie wiesz, jaka jest pogoda. Twoim celem jest na podstawie jego ubioru wywnioskować, jaka jest prawdopodobna pogoda w jego mieście (ukryty stan) oraz jak może się ona zmieniać w kolejnych dniach.
Jak to działa?

1. **Obserwacje**: Patrzysz na codzienny wybór stroju (obserwacje).
2. **Ukryty stan**: Na podstawie tego, co nosi, próbujesz zgadnąć, jaka była pogoda danego dnia (ukryty stan).
3. **Przewidywanie**: Na podstawie sekwencji ubrań, które wybierał w poprzednich dniach, próbujesz przewidzieć, jaka będzie pogoda w kolejnych dniach (czyli przejścia między stanami).

Zastosowania

- Rozpoznawanie mowy - HMM są używane do modelowania sekwencji dźwięków i identyfikacji słów czy fraz.

- Bioinformatyka - W analizie sekwencji DNA i białek, HMM są używane do identyfikowania genów i innych ważnych sekwencji.

- NLP (Przetwarzanie języka naturalnego) - HMM są używane w zadaniach takich jak rozpoznawanie jednostek gramatycznych, tłumaczenie maszynowe czy analiza sentymentu.

- Robotyka i Nawigacja - HMM mogą być używane do śledzenia ścieżek czy też do rozpoznawania aktywności opartych na sensorach.

Ogólna zasada działania algorytmu Viterbiego

Algorytm Viterbiego służy do znajdowania najbardziej prawdopodobnej sekwencji stanów ukrytych w modelu z Ukrytym Procesem Markowa (HMM) na podstawie obserwacji. Aby to wyjaśnić na prostym przykładzie, wrócimy do naszego scenariusza z pogodą.

Przypomnienie:

- **Stany ukryte**: Słonecznie (S), Deszczowo (D)
- **Obserwacje**: Kurtka (K), Koszulka (C)

Prawdopodobieństwa przejścia między stanami:

- P(Słonecznie -> Słonecznie) = 0.8
- P(Słonecznie -> Deszczowo) = 0.2
- P(Deszczowo -> Deszczowo) = 0.7
- P(Deszczowo -> Słonecznie) = 0.3

Prawdopodobieństwa emisji (tj. jak pogoda wpływa na ubiór):

- P(Kurtka | Słonecznie) = 0.1
- P(Koszulka | Słonecznie) = 0.9
- P(Kurtka | Deszczowo) = 0.8
- P(Koszulka | Deszczowo) = 0.2

Prawdopodobieństwa początkowe (jak zaczynamy):

- P(Słonecznie na początku) = 0.6
- P(Deszczowo na początku) = 0.4

Zadanie:

Załóżmy, że widzimy następującą sekwencję obserwacji: **Kurtka (K), Koszulka (C), Kurtka (K)**. Chcemy ustalić, jaka była najbardziej prawdopodobna sekwencja stanów ukrytych (pogody) w te dni.

Krok 1: Inicjalizacja

Na początku musimy policzyć prawdopodobieństwo rozpoczęcia od każdego stanu i wygenerowania pierwszej obserwacji (Kurtki).

- Dla stanu **Słonecznie (S)**: P(Słonecznienapoczątku)×P(Kurtka|Słonecznie)=0.6×0.1=0.06P(Słonecznie na początku) \times P(Kurtka | Słonecznie) = 0.6 \times 0.1 = 0.06P(Słonecznienapoczątku)×P(Kurtka|Słonecznie)=0.6×0.1=0.06
- Dla stanu **Deszczowo (D)**: P(Deszczowonapoczątku)×P(Kurtka|Deszczowo)=0.4×0.8=0.32P(Deszczowo na początku) \times P(Kurtka | Deszczowo) = 0.4 \times 0.8 = 0.32P(Deszczowonapoczątku)×P(Kurtka|Deszczowo)=0.4×0.8=0.32

Krok 2: Rekurencja

Teraz przechodzimy przez kolejne obserwacje, licząc prawdopodobieństwa dla każdej możliwej ścieżki.

Dzień 2: Obserwacja - Koszulka (C)

Obliczamy prawdopodobieństwo, że jesteśmy w stanie Słonecznie lub Deszczowo, biorąc pod uwagę drugą obserwację (Koszulka).

- Dla **Słonecznie (S)**:
  - Przyszliśmy ze stanu **Słonecznie (S)**:
    0.06×P(Słonecznie−>Słonecznie)×P(Koszulka|Słonecznie)=0.06×0.8×0.9=0.04320.06 \times P(Słonecznie -> Słonecznie) \times P(Koszulka | Słonecznie) = 0.06 \times 0.8 \times 0.9 =
    0.04320.06×P(Słonecznie−>Słonecznie)×P(Koszulka|Słonecznie)=0.06×0.8×0.9=0.0432
  - Przyszliśmy ze stanu **Deszczowo (D)**:
    0.32×P(Deszczowo−>Słonecznie)×P(Koszulka|Słonecznie)=0.32×0.3×0.9=0.08640.32 \times P(Deszczowo -> Słonecznie) \times P(Koszulka | Słonecznie) = 0.32 \times 0.3 \times 0.9 =
    0.08640.32×P(Deszczowo−>Słonecznie)×P(Koszulka|Słonecznie)=0.32×0.3×0.9=0.0864

Najwyższe prawdopodobieństwo to 0.08640.08640.0864, więc wybraliśmy ścieżkę przez stan Deszczowo -> Słonecznie.

- Dla **Deszczowo (D)**:
  - Przyszliśmy ze stanu **Słonecznie (S)**:
    0.06×P(Słonecznie−>Deszczowo)×P(Koszulka|Deszczowo)=0.06×0.2×0.2=0.00240.06 \times P(Słonecznie -> Deszczowo) \times P(Koszulka | Deszczowo) = 0.06 \times 0.2 \times 0.2 =
    0.00240.06×P(Słonecznie−>Deszczowo)×P(Koszulka|Deszczowo)=0.06×0.2×0.2=0.0024
  - Przyszliśmy ze stanu **Deszczowo (D)**:
    0.32×P(Deszczowo−>Deszczowo)×P(Koszulka|Deszczowo)=0.32×0.7×0.2=0.04480.32 \times P(Deszczowo -> Deszczowo) \times P(Koszulka | Deszczowo) = 0.32 \times 0.7 \times 0.2 =
    0.04480.32×P(Deszczowo−>Deszczowo)×P(Koszulka|Deszczowo)=0.32×0.7×0.2=0.0448

Najwyższe prawdopodobieństwo to 0.04480.04480.0448, więc wybieramy ścieżkę przez Deszczowo -> Deszczowo.

Dzień 3: Obserwacja - Kurtka (K)

Obliczamy prawdopodobieństwo dla trzeciego dnia (obserwacja: Kurtka).

- Dla **Słonecznie (S)**:
  - Przyszliśmy ze stanu **Słonecznie (S)**:
    0.0864×P(Słonecznie−>Słonecznie)×P(Kurtka|Słonecznie)=0.0864×0.8×0.1=0.0069120.0864 \times P(Słonecznie -> Słonecznie) \times P(Kurtka | Słonecznie) = 0.0864 \times 0.8 \times 0.1 =
    0.0069120.0864×P(Słonecznie−>Słonecznie)×P(Kurtka|Słonecznie)=0.0864×0.8×0.1=0.006912
  - Przyszliśmy ze stanu **Deszczowo (D)**:
    0.0448×P(Deszczowo−>Słonecznie)×P(Kurtka|Słonecznie)=0.0448×0.3×0.1=0.0013440.0448 \times P(Deszczowo -> Słonecznie) \times P(Kurtka | Słonecznie) = 0.0448 \times 0.3 \times 0.1 =
    0.0013440.0448×P(Deszczowo−>Słonecznie)×P(Kurtka|Słonecznie)=0.0448×0.3×0.1=0.001344

Najwyższe prawdopodobieństwo to 0.0069120.0069120.006912, więc wybieramy ścieżkę Słonecznie -> Słonecznie -> Słonecznie.

- Dla **Deszczowo (D)**:
  - Przyszliśmy ze stanu **Słonecznie (S)**:
    0.0864×P(Słonecznie−>Deszczowo)×P(Kurtka|Deszczowo)=0.0864×0.2×0.8=0.0138240.0864 \times P(Słonecznie -> Deszczowo) \times P(Kurtka | Deszczowo) = 0.0864 \times 0.2 \times 0.8 =
    0.0138240.0864×P(Słonecznie−>Deszczowo)×P(Kurtka|Deszczowo)=0.0864×0.2×0.8=0.013824

- Przyszliśmy ze stanu **Deszczowo (D)**: 0.0448×P(Deszczowo−>Deszczowo)×P(Kurtka|Deszczowo)=0.0448×0.7×0.8=0.0250880.0448 \times P(Deszczowo -> Deszczowo) \times P(Kurtka | Deszczowo) = 0.0448 \times 0.7 \times 0.8 = 0.0250880.0448×P(Deszczowo−>Deszczowo)×P(Kurtka|Deszczowo)=0.0448×0.7×0.8=0.025088

Najwyższe prawdopodobieństwo to 0.0250880.0250880.025088, więc wybieramy ścieżkę Deszczowo -> Deszczowo -> Deszczowo.

Krok 3: Ścieżka z największym prawdopodobieństwem

Największe prawdopodobieństwo końcowe to 0.0250880.0250880.025088, więc najbardziej prawdopodobna sekwencja stanów to **Deszczowo -> Deszczowo -> Deszczowo**.

Podsumowanie:

Obserwacje: Kurtka, Koszulka, Kurtka

Najbardziej prawdopodobna sekwencja stanów: Deszczowo, Deszczowo, Deszczowo

To jest prosty przykład użycia algorytmu Viterbiego w praktyce.

H)Zaawansowane interfejsy Graficzne

38)Wiązanie Danych (Data Binding) w WPF (Windows Presentation Foundation)

Wiązanie danych w WPF umożliwia łatwą synchronizację pomiędzy warstwą prezentacji (interfejsem użytkownika) a warstwą logiki aplikacji (dane, modele). Pozwala to na oddzielenie warstw, co ułatwia zarządzanie kodem i tworzenie skalowalnych, łatwo testowalnych aplikacji.

Rodzaje Wiązania

OneWay: Aktualizacja danych w kontrolce, gdy zmieniają się dane źródłowe.

TwoWay: Aktualizacja zarówno danych źródłowych, jak i kontrolek, gdy jedno z nich się zmienia.

OneTime: Jednorazowe przypisanie wartości ze źródła do kontrolki.

OneWayToSource: Aktualizacja danych źródłowych, gdy zmienia się wartość w kontrolce.

Składnia XAML

W składni XAML można zdefiniować wiązanie danych przy użyciu atrybutu Binding:

xaml Copy
code
<TextBlock Text="{Binding Path=PropertyName}" />

Przykład Wiązania

Załóżmy, że mamy klasę Person:

csharp Copy
code
```
public class Person : INotifyPropertyChanged
{
   private string _name;

   public string Name
   {
      get { return _name; }
      set
      {
         if (_name != value)
         {
            _name = value;
            OnPropertyChanged("Name");
```

```
        }
      }
    }

    public event PropertyChangedEventHandler PropertyChanged;

    protected virtual void OnPropertyChanged(string propertyName)
    {
       PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
    } }
```

W kodzie XAML możemy związać właściwość Name z kontrolką TextBox:

xaml Copy
code
<TextBox Text="{Binding Name, Mode=TwoWay}" /> W
kodzie C# ustawiamy DataContext:

csharp Copy
code
this.DataContext = new Person { Name = "John" };
Teraz, gdy zmieni się wartość właściwości Name obiektu klasy Person, automatycznie zaktualizuje się również wartość w TextBox, i vice versa.

Wiązanie danych w WPF jest potężnym narzędziem, które znacznie upraszcza prace nad aplikacjami. Dzięki odseparowaniu logiki od prezentacji, możliwe jest też łatwiejsze testowanie i rozwijanie aplikacji.

39) Metody testowania aplikacji posiadających graficzny interfejs użytkownika

Ręczne testowanie

- Testy eksploracyjne: Testujący przeszukuje aplikację, aby znaleźć ewentualne błędy.
- Testy scenariuszowe: Wykonywanie predefiniowanych scenariuszy, które symulują konkretne zadania użytkownika.

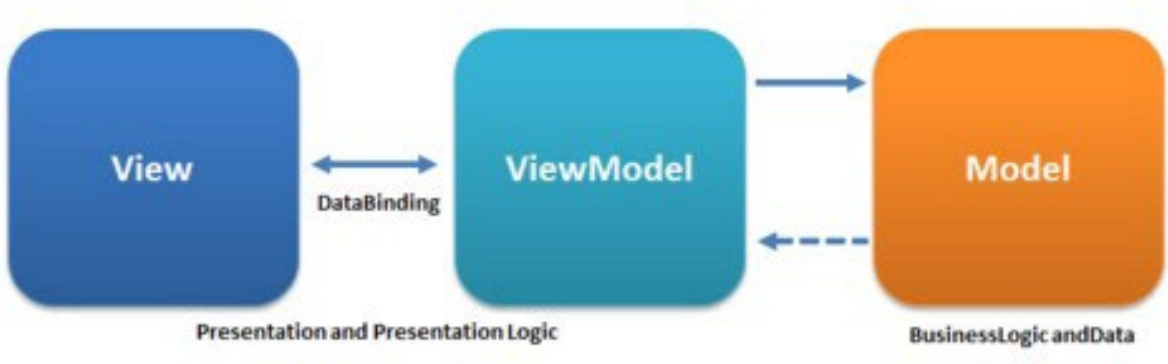
Automatyczne testowanie

T- esty **jednostkowe** dla GUI: Izolowane testy pewnych aspektów interfejsu użytkownika, np. czy przycisk działa jak oczekiwano.
- Testy **funkcjonalne**: Automatyczne testy, które symulują interakcje użytkownika z aplikacją.
- Testy **regresji**: Celem jest sprawdzenie, czy nowo wprowadzone zmiany nie wpłynęły negatywnie na istniejącą funkcjonalność.
- Testy **end-to-end**: Testy, które symulują pełen przepływ aktywności użytkownika, od początku do końca.

Narzędzia

-Selenium: Jeden z najpopularniejszych frameworków do automatycznego testowania aplikacji webowych.
- Appium: Używane głównie do testowania aplikacji mobilnych.
- JUnit, TestNG: Frameworki używane do testów jednostkowych w Javie, które można integrować z Selenium. (Nunit)

40) Wzorzec MVVM



Wzorzec MVVM (Model-View-ViewModel) to architektoniczny wzorzec projektowy, który jest często używany do budowy aplikacji z graficznym interfejsem użytkownika (GUI). Wzorzec ten jest szczególnie popularny w kontekście platform takich jak WPF, Xamarin i Angular. Oto główne składniki wzorca MVVM:

Komponenty

- **Model**: Reprezentuje logikę biznesową i dane. W tym elemencie zwykle odbywa się komunikacja z bazą danych oraz inne operacje związane z danymi.

- **View**: To wszystko, co widzi użytkownik. Odpowiada za wyświetlanie danych i prezentację interfejsu.

- **ViewModel**: Pełni rolę pośrednika między Modelem a Widokiem. Zawiera logikę prezentacji i obsługuje interakcje użytkownika, przekształcając je na akcje dla Modelu.

I) Bezpieczeństwo w sieciach

41) Omów zasadę działania firewalla

Firewall, czyli zapora sieciowa, to system zabezpieczeń, który monitoruje i kontroluje ruch sieciowy (przepływ danych) między dwoma lub więcej sieciami, najczęściej między siecią lokalną (LAN) a Internetem. Główne zadanie firewalla to ochrona sieci przed nieautoryzowanym dostępem i złośliwym oprogramowaniem, poprzez blokowanie lub zezwalanie na określone rodzaje ruchu na podstawie zdefiniowanych reguł.

Zasada działania firewalla

Firewall działa na podstawie zestawu reguł, które określają, jaki ruch jest dozwolony, a jaki ma być blokowany. Te reguły są stosowane na ruch wchodzący (przychodzący do sieci), wychodzący (opuszczający sieć) lub na oba te rodzaje ruchu.

Typy firewalli:

1. **Filtracja pakietów (Packet Filtering)**: To najprostsza forma firewalla, który działa na poziomie warstwy sieciowej. Każdy pakiet danych (mała jednostka informacji przesyłanej przez sieć) jest analizowany pod kątem:
    - Adresu źródłowego i docelowego (IP),
    - Portu źródłowego i docelowego,
    - Protokołu (np. TCP, UDP),
    - Stanu połączenia.

Firewall na tej podstawie decyduje, czy dany pakiet przepuścić czy zablokować. Jest to szybki, ale dość podstawowy sposób ochrony, który nie analizuje zawartości pakietów.

2. **Stanowy filtr pakietów (Stateful Packet Inspection, SPI)**: Tego typu firewall nie tylko analizuje poszczególne pakiety, ale także śledzi stan połączeń. To znaczy, że firewall wie, czy dany pakiet jest częścią już istniejącego, dozwolonego połączenia, czy też jest nową próbą połączenia. Jest to bardziej zaawansowane i bezpieczne niż prosta filtracja pakietów, ponieważ umożliwia kontrolowanie całych sesji komunikacyjnych.

3. **Firewall aplikacyjny (Application Layer Firewall)**: Ten typ firewalla działa na wyższych warstwach modelu OSI, aż do warstwy aplikacji (np. HTTP, FTP). Może analizować ruch w kontekście konkretnej aplikacji, a nie tylko na podstawie adresów IP i portów. Na przykład, firewall aplikacyjny może blokować złośliwe żądania HTTP, analizować treść wiadomości e-mail lub ruchu FTP. Jest bardziej zaawansowany, ale również bardziej zasobożerny.
4. **Firewall proxy**: Działa jako pośrednik między użytkownikiem a Internetem. Firewall proxy przyjmuje żądania od klienta, a następnie przesyła je do odpowiedniego serwera, otrzymuje odpowiedź i przekazuje ją z powrotem do klienta. Taki firewall może analizować zarówno ruch przychodzący, jak i wychodzący, a także cache'ować treści, co zwiększa bezpieczeństwo i wydajność.
5. **Next-Generation Firewall (NGFW)**: To zaawansowany firewall, który łączy funkcje kilku typów zapór, dodając do tego np. systemy wykrywania włamań (IDS/IPS), funkcje głębokiej inspekcji pakietów (DPI) oraz mechanizmy wykrywania i blokowania złośliwego oprogramowania. NGFW analizują ruch nie tylko na podstawie portów i protokołów, ale także kontekstowo, np. rozpoznając konkretne aplikacje (np. Facebook) i użytkowników.

Działanie firewalla - kluczowe komponenty

1. **Reguły (Rules)**: Reguły firewalla to zestawy zasad, które definiują, jaki ruch jest dozwolony, a jaki zablokowany. Każda reguła składa się z kilku elementów, takich jak:
    - Źródłowy i docelowy adres IP,
    - Źródłowy i docelowy port,
    - Protokół (TCP, UDP),
    - Akcja: "Zezwalaj" lub "Blokuj".

Firewall analizuje każdy pakiet danych przechodzący przez sieć i porównuje go z listą reguł, aby zdecydować, co zrobić z tym pakietem.

2. **Inspekcja pakietów (Packet Inspection)**: Podczas inspekcji, firewall analizuje nagłówki pakietów danych, aby zidentyfikować adres IP źródłowy i docelowy, porty, a także protokoły. Firewall może działać na różnych poziomach modelu OSI, co pozwala na inspekcję od podstawowych informacji sieciowych po analizę na poziomie aplikacji.
3. **Blokowanie złośliwego ruchu (Intrusion Detection/Prevention)**: Firewalle nowej generacji (NGFW) mogą wykrywać złośliwe oprogramowanie, wirusy, trojany i inne zagrożenia, bazując na sygnaturach ataków lub anomaliach w ruchu sieciowym. W przypadku wykrycia złośliwego ruchu firewall może go zablokować, zanim osiągnie on cel.

Przykład działania firewalla

Załóżmy, że masz w domu router z wbudowanym firewallem. Ustalono, że firewall blokuje wszystkie połączenia przychodzące z zewnątrz (Internetu), chyba że są one odpowiedzią na wcześniej zainicjowane przez ciebie połączenie.

1. **Próba nawiązania połączenia**:
   Kiedy twój komputer łączy się z witryną internetową, wysyła zapytanie do serwera. Firewall przepuszcza tę prośbę, ponieważ ruch wychodzi z twojej sieci.
2. **Odpowiedź z serwera**:
   Serwer odpowiada na zapytanie, wysyłając dane do twojego komputera. Firewall sprawdza, czy odpowiedź jest zgodna z istniejącym połączeniem i przepuszcza ją.
3. **Nieautoryzowane połączenie**:
   Jeśli ktoś z zewnątrz próbuje nawiązać połączenie z twoim komputerem bez wcześniejszego zapytania (np. atak), firewall blokuje ten ruch, chroniąc sieć przed potencjalnym zagrożeniem.

Podsumowanie

Firewall działa jako bariera ochronna pomiędzy twoją siecią a zewnętrznymi zagrożeniami. Analizuje ruch sieciowy na podstawie zestawu reguł i decyzji, które mogą blokować lub przepuszczać dane. W nowoczesnych firewallach stosowane są zaawansowane technologie, takie jak inspekcja pakietów, analiza aplikacji czy wykrywanie złośliwego oprogramowania, co znacznie podnosi poziom bezpieczeństwa.

42) Wymień znane metody kryptograficzne

Metody kryptograficzne to techniki stosowane do zapewnienia bezpieczeństwa danych poprzez ich szyfrowanie i deszyfrowanie. Oto niektóre z najbardziej znanych metod kryptograficznych:

1. Symetryczne metody kryptograficzne (Symmetric-key cryptography)

W symetrycznej kryptografii ten sam klucz służy do szyfrowania i deszyfrowania danych. Jest to szybka metoda, ale wymaga bezpiecznego przesyłania klucza między nadawcą a odbiorcą.

- **DES (Data Encryption Standard)**: Starszy standard szyfrowania blokowego z kluczem o długości 56 bitów, obecnie uważany za przestarzały z powodu małej długości klucza.
- **3DES (Triple DES)**: Ulepszona wersja DES, która trzykrotnie stosuje algorytm DES, zwiększając poziom bezpieczeństwa.
- **AES (Advanced Encryption Standard)**: Obecny standard szyfrowania przyjęty przez rząd USA. Wykorzystuje klucze o długości 128, 192 lub 256 bitów, oferując wysokie bezpieczeństwo.
- **Blowfish**: Algorytm szyfrowania blokowego o zmiennej długości klucza (od 32 do 448 bitów), szybki i bezpieczny.
- **RC4 (Rivest Cipher 4)**: Szyfr strumieniowy, popularny w przeszłości, ale obecnie uważany za mniej bezpieczny.

2. Asymetryczne metody kryptograficzne (Asymmetric-key cryptography)

W asymetrycznej kryptografii używane są dwa różne klucze: publiczny (do szyfrowania) i prywatny (do deszyfrowania). Jest wolniejsza, ale nie wymaga bezpiecznej wymiany kluczy.

- **RSA (Rivest-Shamir-Adleman)**: Jeden z najstarszych i najbardziej znanych algorytmów asymetrycznych, używa kluczy o długości do 4096 bitów. Stosowany w szyfrowaniu danych oraz podpisach cyfrowych.
- **ECC (Elliptic Curve Cryptography)**: Kryptografia na krzywych eliptycznych oferuje podobny poziom bezpieczeństwa jak RSA przy krótszych kluczach, co czyni ją bardziej wydajną.
- **DSA (Digital Signature Algorithm)**: Algorytm stosowany głównie do tworzenia podpisów cyfrowych, bazuje na kryptografii asymetrycznej.

3. Kryptografia klucza jednorazowego (One-time pad)

Jest to technika, w której klucz jest tak długi, jak szyfrowany tekst, i używany tylko raz. Jeśli klucz jest generowany losowo i użyty raz, ta metoda jest teoretycznie niemożliwa do złamania. Problemem jest jednak praktyczne zarządzanie kluczami.

4. Funkcje skrótu (Hash functions)

Funkcje skrótu nie są używane do szyfrowania, ale do tworzenia "odcisków palca" danych. Wynik działania funkcji skrótu ma stałą długość i jest unikalny dla danej porcji danych.

- **MD5 (Message Digest Algorithm 5)**: Popularna w przeszłości funkcja skrótu, ale obecnie uważana za przestarzałą z powodu wykrytych kolizji.
- **SHA-1 (Secure Hash Algorithm 1)**: Stosowany przez wiele lat, ale obecnie nie zalecany ze względu na wykryte słabości.
- **SHA-2 (Secure Hash Algorithm 2)**: Rodzina funkcji skrótu, obejmująca m.in. SHA-256, SHA-512, które oferują wyższy poziom bezpieczeństwa.
- **SHA-3 (Keccak)**: Nowsza funkcja skrótu, opracowana jako alternatywa dla SHA-2, oferująca większą odporność na ataki.

5. Kryptografia kwantowa

Kryptografia kwantowa to rozwijająca się dziedzina, oparta na zasadach mechaniki kwantowej, w szczególności z zastosowaniem technologii kwantowego rozdzielania klucza (Quantum Key Distribution, QKD). Techniki te zapewniają teoretyczną niemożliwość przechwycenia klucza przez osoby trzecie bez wzbudzania wykrywalnych zakłóceń.

6. Szyfrowanie homomorficzne

Pozwala na wykonywanie operacji matematycznych na zaszyfrowanych danych bez ich deszyfrowania. Zaszyfrowany wynik operacji po odszyfrowaniu jest taki sam, jak gdyby operacje były wykonywane na danych w postaci jawnej. To przydatne w scenariuszach, takich jak obliczenia na danych przechowywanych w chmurze, gdzie prywatność danych musi być zachowana.

7. Kryptografia klucza dzielonego (Secret Sharing)

Polega na podzieleniu klucza na kilka części i rozesłaniu ich do różnych uczestników, tak aby klucz mógł zostać odtworzony tylko wtedy, gdy wystarczająca liczba osób połączy swoje fragmenty. Przykładem jest **Schemat Shamira**.

8. Podpisy cyfrowe (Digital Signatures)

Podpisy cyfrowe służą do uwierzytelniania nadawcy i integralności wiadomości. Bazują na kryptografii asymetrycznej, gdzie prywatny klucz służy do tworzenia podpisu, a publiczny do jego weryfikacji.
Przykłady:

- **RSA** (może być używany zarówno do szyfrowania, jak i podpisów cyfrowych).
- **ECDSA** (oparty na kryptografii krzywych eliptycznych, efektywniejszy przy krótszych kluczach).

9. Kryptografia oparta na tożsamości (Identity-based encryption, IBE)

IBE to metoda szyfrowania, w której publiczne klucze mogą być tożsamościami (np. adresy e-mail). System kluczy prywatnych jest generowany przez centralny urząd (authority), co upraszcza zarządzanie kluczami w niektórych scenariuszach.

Podsumowanie

Znane metody kryptograficzne obejmują szeroki zakres technik, od prostych algorytmów symetrycznych i asymetrycznych po bardziej zaawansowane systemy, takie jak kryptografia kwantowa czy homomorficzna. Każda z tych metod ma swoje zastosowania w zależności od potrzeb związanych z bezpieczeństwem, wydajnością i poziomem zagrożeń.

43) Omów fazy ataków sieciowych.

Ataki sieciowe są często realizowane w sposób zorganizowany i przebiegają według określonego schematu. Zrozumienie faz ataków sieciowych jest kluczowe dla obrony przed nimi. Zwykle atak sieciowy można podzielić na kilka faz, które składają się na cały cykl ataku. Oto najczęściej występujące fazy ataków sieciowych:

1. Rozpoznanie (Reconnaissance)

W tej fazie atakujący stara się zebrać jak najwięcej informacji o celu. Działania te mogą być prowadzone pasywnie (bez bezpośredniej interakcji z celem) lub aktywnie (poprzez interakcję z siecią ofiary).

- **Rozpoznanie pasywne**: Atakujący analizuje publicznie dostępne informacje o celu, takie jak dane WHOIS, rekordy DNS, informacje o firmie dostępne w Internecie.
- **Rozpoznanie aktywne**: W tym przypadku atakujący może skanować porty, próbować mapować sieć i wykrywać usługi, aby zidentyfikować potencjalne luki bezpieczeństwa.

Przykłady technik:

- Skanowanie portów (np. za pomocą narzędzia **Nmap**).
- Analiza zasobów sieciowych (serwery DNS, witryny internetowe).
- Przechwytywanie ruchu sieciowego.

2. Skanowanie i identyfikacja podatności (Scanning and Vulnerability Identification)

Po rozpoznaniu, atakujący przechodzi do bardziej szczegółowego skanowania w poszukiwaniu słabych punktów w infrastrukturze sieciowej ofiary. Atakujący próbuje dowiedzieć się, jakie urządzenia są aktywne, jakie oprogramowanie jest zainstalowane i czy są jakieś znane podatności.

- **Skanowanie portów**: Wykrywanie otwartych portów, które mogą umożliwiać dostęp do serwerów lub aplikacji.
- **Skanowanie podatności**: Atakujący wykorzystuje narzędzia do skanowania (np. **Nessus**, **OpenVAS**), aby zidentyfikować luki w systemie, takie jak nieaktualne oprogramowanie, otwarte usługi lub błędy konfiguracyjne.
- **Fingerprinting systemu operacyjnego**: Analiza systemu w celu zidentyfikowania używanego systemu operacyjnego i jego wersji.

3. Uzyskiwanie dostępu (Gaining Access)

Na tym etapie atakujący wykorzystuje uzyskane informacje, aby spróbować uzyskać dostęp do systemu lub sieci. Wykorzystuje zidentyfikowane podatności, aby przeprowadzić eksploitację, czyli atak, który pozwala na wejście do systemu.
Metody uzyskiwania dostępu:

- **Ataki na podatności oprogramowania**: Wykorzystanie luk w aplikacjach lub systemie operacyjnym (np. zdalne wykonywanie kodu).
- **Ataki brute-force**: Próba złamania hasła przy użyciu siłowego ataku.
- **Ataki phishingowe**: Podszywanie się pod zaufane źródła w celu wyłudzenia danych logowania lub innych informacji umożliwiających dostęp.

Jeśli atakujący z sukcesem uzyska dostęp, często zdobywa uprawnienia konta użytkownika o ograniczonych uprawnieniach.

4. Podniesienie uprawnień (Privilege Escalation)

Po uzyskaniu dostępu do systemu, atakujący stara się zdobyć wyższe uprawnienia, aby uzyskać większą kontrolę nad systemem, np. dostęp do konta administratora.

Przykłady:

- **Wykorzystanie lokalnych luk bezpieczeństwa**: Atakujący może wykorzystać luki w systemie operacyjnym, aby uzyskać prawa administratora.
- **Eksploracja pamięci**: Atakujący może próbować wyciągnąć dane z pamięci RAM systemu, aby uzyskać hasła lub inne poufne informacje.

5. Instalacja złośliwego oprogramowania (Installing Malware)

Na tym etapie atakujący często instaluje dodatkowe złośliwe oprogramowanie, które pomaga w dalszej eksploatacji systemu lub zapewnia możliwość powrotu do systemu w przyszłości.

Przykłady:

- **Backdoory**: Zainstalowanie oprogramowania, które umożliwia powrót do systemu w przyszłości bez konieczności ponownego przeprowadzania ataku.
- **Rootkity**: Oprogramowanie, które ukrywa aktywność atakującego przed użytkownikiem systemu.
- **Trojany**: Złośliwe oprogramowanie, które podszywa się pod legalne aplikacje, ale ma ukryte funkcje umożliwiające kontrolę nad systemem.

6. Zachowanie dostępu (Maintaining Access)

Atakujący próbuje utrzymać kontrolę nad systemem przez dłuższy czas. Może zainstalować dodatkowe narzędzia lub modyfikować konfigurację systemu, aby utrudnić wykrycie jego obecności i zapewnić ciągły dostęp do systemu.

Techniki:

- **Zainstalowanie backdoorów**: Programy pozwalające na zdalny, nieautoryzowany dostęp do systemu w przyszłości.
- **Użycie kont uprzywilejowanych**: Atakujący może stworzyć nowe konta użytkowników z wysokimi uprawnieniami lub zmodyfikować istniejące.

7. Działania destrukcyjne lub kradzież danych (Executing Attack Goals)

To faza, w której atakujący realizuje swoje cele, takie jak:

- **Kradzież danych**: Przesyłanie danych poza system ofiary, np. kradzież informacji finansowych, osobowych, danych klientów.
- **Sabotaż**: Niszcząc systemy, pliki lub całe infrastruktury (np. za pomocą ransomware).
- **Atak typu DDoS**: Zaatakowanie i sparaliżowanie działania sieci lub serwerów, aby utrudnić ich działanie lub całkowicie je zablokować.

8. Wycofanie się (Covering Tracks)

Na końcowym etapie atakujący stara się ukryć swoje ślady, aby utrudnić wykrycie, śledzenie oraz analizę ataku przez administratorów systemów lub specjalistów od cyberbezpieczeństwa.

Techniki ukrywania śladów:

- **Czyszczenie logów**: Usuwanie lub modyfikowanie logów systemowych, aby usunąć ślady obecności atakującego.
- **Ukrywanie backdoorów**: Stworzenie sposobów na powrót do systemu bez pozostawiania śladów.
- **Usuwanie narzędzi atakujących**: Usunięcie złośliwego oprogramowania lub narzędzi, które mogłyby zdradzić obecność atakującego.

Podsumowanie:

Fazy ataków sieciowych to cykliczny proces, w którym atakujący zbiera informacje, uzyskuje dostęp, eskaluje swoje uprawnienia, instaluje złośliwe oprogramowanie, realizuje swoje cele, a następnie ukrywa

ślady swojej działalności. Każda z tych faz wymaga zastosowania odpowiednich technik ochrony i monitorowania, aby skutecznie przeciwdziałać atakom.

J)Projektowanie wspomagane komputerem

44) Rodzaje i charakterystyka akcji projektowych

Akcje:

- **akcje fizyczne** – zmiana w świecie zewnętrznym zgodnie z celem ustalonym w świecie wewnętrznym; polegają na:
  - rysowaniu nowych elementów, np. linii, okręgów,
  - usuwaniu elementów
  - kopiowaniu elementów
- **akcje percepcyjne** – odbywa się w świecie wewnętrznym projektanta:
  - odkrywanie wizualnych cech elementów, np. kształtu, wymiaru,
  - postrzeganie relacji przestrzennych, np. sąsiedztwa,
  - odnajdowanie organizacji elementów, np. podziału na grupy,
  - porównywanie elementów, np. szukanie podobieństw;
- **akcje funkcyjne** - odbywa się w świecie wewnętrznym projektanta;
  - polegają na łączeniu poszczególnych wizualno-przestrzennych cech w obiektach graficznych z pewną treścią lub funkcją albo abstrakcją
- **akcje koncepcyjne** - odbywa się w świecie wewnętrznym projektanta;
  - polegają na określeniu celów i wymagań projektowych,
  - ocenie rozwiązań projektowych (np. estetyki),
  - wyszukiwaniu wiedzy i podobnych rozwiązań

45) Narzędzia wykorzystywane w projektowaniu wizualnym i ich charakterystyka

Narzędzia wykorzystywane w projektowaniu wizualnym i ich charakterystyka:
Projektowanie wizualne opiera się na różnorodnych narzędziach, które wspierają proces tworzenia form, struktur oraz kompozycji. Każde z tych narzędzi pełni inną funkcję i umożliwia realizację różnych zadań w zależności od potrzeb projektanta. Oto główne narzędzia używane w projektowaniu wizualnym oraz ich charakterystyka:

1. Gramatyka kształtu (Shape Grammar Tools):

- **Charakterystyka:** Gramatyka kształtu to system reguł formalnych, które pozwalają na generowanie skomplikowanych form na podstawie prostych operacji geometrycznych, takich jak przesunięcia, obroty czy skalowanie. Narzędzia oparte na tej metodzie umożliwiają projektantom tworzenie złożonych struktur w sposób automatyczny, z zachowaniem pełnej kontroli nad regułami generacyjnymi.
- **Zastosowanie:** Głównie w architekturze, sztuce wizualnej i designie przemysłowym. Przykładem mogą być narzędzia takie jak "Shape Grammar Interpreter", które umożliwiają definiowanie i stosowanie reguł gramatyki kształtu do generowania form.
- **Zalety:** Automatyzacja, iteracyjne podejście, możliwość generowania dużej liczby wariantów projektu na podstawie jednego zestawu reguł.

2. Edytory graficzne:

- **Charakterystyka:** To programy służące do tworzenia, modyfikowania i edytowania grafiki komputerowej, zarówno rastrowej, jak i wektorowej. Edytory graficzne oferują szeroką gamę narzędzi, takich jak pędzle, narzędzia do tworzenia kształtów, manipulowania kolorem, a także zaawansowane funkcje jak warstwy czy filtry.
- **Przykłady:**
  - **Adobe Illustrator:** Program do projektowania grafiki wektorowej, idealny do tworzenia logotypów, ilustracji i ikon.

    - **CorelDRAW:** Alternatywa do Illustratora, szeroko stosowany w projektowaniu komercyjnym i graficznym.
    - **Inkscape:** Darmowy edytor wektorowy, który oferuje wiele funkcji dostępnych w płatnych programach.
- **Zalety:** Precyzja w edytowaniu kształtów i elementów, możliwość tworzenia skalowalnych grafik bez utraty jakości (wektory).

3. Edytory funkcjonalno-strukturalne:

- **Charakterystyka:** Edytory te pozwalają na projektowanie nie tylko pod kątem estetycznym, ale także z uwzględnieniem struktury i funkcji obiektu. Projektanci mogą kontrolować relacje między elementami, definiować parametry i określać funkcje, które muszą pełnić poszczególne części projektu.
- **Przykłady:**
    - **Grasshopper (wtyczka do Rhinoceros 3D):** Narzędzie do parametrycznego modelowania, które pozwala na tworzenie złożonych form opartych na regułach matematycznych i logicznych.
    - **Houdini:** Profesjonalny program do modelowania 3D, szeroko stosowany w branży efektów wizualnych, który pozwala na kontrolowanie każdego aspektu formy przez zmienne i parametry.
- **Zalety:** Zwiększona kontrola nad projektem, integracja formy i funkcji, parametryczne podejście umożliwiające modyfikację projektu bez konieczności jego całkowitej przebudowy.

4. Narzędzia parametryczne:

- **Charakterystyka:** Narzędzia parametryczne umożliwiają projektowanie w oparciu o zmienne, które definiują poszczególne elementy projektu. Dzięki temu projektanci mogą tworzyć projekty, które można dynamicznie modyfikować przez zmianę wartości parametrów, co prowadzi do generowania różnych wariantów projektu bez potrzeby ręcznej edycji.
- **Przykłady:**
    - **Blender:** Otwarty program do modelowania 3D, który oferuje zaawansowane funkcje parametryczne, pozwalając na tworzenie animacji, gier i symulacji.
    - **FreeCAD:** Narzędzie open-source do modelowania parametrycznego, szeroko stosowane w projektowaniu inżynieryjnym.
- **Zalety:** Szybkie generowanie wariantów projektu, dynamiczna kontrola nad formą i funkcją, łatwość w dostosowywaniu projektów.

5. Narzędzia do modelowania 3D:

- **Charakterystyka:** Narzędzia do modelowania 3D pozwalają na tworzenie trójwymiarowych obiektów, które mogą być używane w projektowaniu wizualnym, architekturze, grach, animacji oraz innych dziedzinach. Programy te oferują funkcje takie jak modelowanie bryłowe, powierzchniowe, a także renderowanie, symulacje i animacje.
- **Przykłady:**
    - **Autodesk Maya:** Profesjonalne narzędzie do tworzenia animacji i modelowania 3D, szeroko stosowane w filmie i grach.
    - **Rhinoceros 3D:** Narzędzie do modelowania kształtów w oparciu o krzywe NURBS, popularne w architekturze i wzornictwie przemysłowym.
- **Zalety:** Tworzenie złożonych, realistycznych form, możliwość symulacji i renderowania, szerokie zastosowanie w różnych dziedzinach kreatywnych.

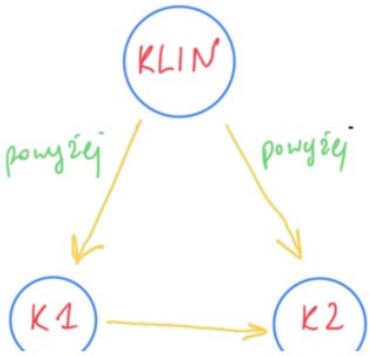
6. Narzędzia do renderingu i wizualizacji:

- **Charakterystyka:** Programy te służą do tworzenia fotorealistycznych obrazów lub animacji z modelów 3D, pozwalając na prezentację projektów w kontekście ich rzeczywistego wyglądu.
- **Przykłady:**
    - **V-Ray:** Popularny silnik renderujący, szeroko stosowany w architekturze i projektowaniu wnętrz.

- **KeyShot:** Narzędzie do szybkiego renderowania i tworzenia fotorealistycznych wizualizacji, idealne do prezentacji projektów produktowych.
- **Zalety:** Realistyczna wizualizacja projektu, możliwość prezentacji ostatecznego wyglądu przed produkcją.

Każde z tych narzędzi ma swoje specyficzne zastosowania i oferuje unikalne funkcje, które wspierają różne aspekty procesu projektowego. Wybór odpowiedniego narzędzia zależy od specyfiki projektu, wymagań funkcjonalnych oraz preferencji projektanta.

46) Definicja interpretacji grafu



Etykietowany graf skierowany (Directed Labeled Graph) to rodzaj grafu, który spełnia kilka szczególnych właściwości, kluczowych w analizie sieci, struktur danych oraz modelowania zależności. Oto jego główne cechy:

Cechy charakterystyczne etykietowanego grafu skierowanego:

1. **Graf skierowany (directed graph, digraph):**
   - **Wierzchołki i krawędzie:** Graf składa się z zestawu wierzchołków (ang. *vertices* lub *nodes*) oraz krawędzi (ang. *edges* lub *arcs*).
   - **Skierowane krawędzie:** Krawędzie są skierowane, co oznacza, że każda krawędź ma określony kierunek – biegnie od jednego wierzchołka do innego. W skrócie, krawędź łącząca wierzchołki uuu i vvv nie jest symetryczna, a wyrażenie (u→v)(u \rightarrow v)(u→v) oznacza, że istnieje krawędź z uuu do vvv, ale niekoniecznie odwrotnie.
   - **Przykład:** Węzeł A może prowadzić do węzła B (A→BA \rightarrow BA→B), ale B→AB \rightarrow AB→A nie musi istnieć.
2. **Etykietowanie (labeling):**
   - **Etykiety krawędzi i wierzchołków:** W etykietowanym grafie każda krawędź i/lub wierzchołek może mieć przypisaną etykietę (ang. *label*). Etykiety mogą być dowolnymi symbolami, liczbami lub ciągami znaków, które w jakiś sposób opisują lub klasyfikują elementy grafu.
   - **Zastosowania etykiet:** Etykiety mogą reprezentować różne cechy lub atrybuty. Na przykład:
     - **Wierzchołki:** Mogą opisywać obiekty, takie jak osoby w sieci społecznościowej.
     - **Krawędzie:** Mogą reprezentować relacje między obiektami, np. "jest przyjacielem", "zależy od", "wskazuje na".
3. **Notacja matematyczna:**
   - Graf skierowany zwykle oznaczany jest jako G=(V,E)G = (V, E)G=(V,E), gdzie:
     - VVV to zbiór wierzchołków.
     - E⊆V×VE \subseteq V \times VE⊆V×V to zbiór skierowanych krawędzi, które są parami uporządkowanymi (u,v)(u, v)(u,v), gdzie u,v∈Vu, v \in Vu,v∈V.
   - Jeśli graf jest etykietowany, dodajemy funkcję etykietowania LLL, która przypisuje etykiety wierzchołkom i/lub krawędziom.

Zastosowania etykietowanego grafu skierowanego:

1. **Modelowanie sieci komputerowych:**
   - Wierzchołki mogą reprezentować urządzenia, takie jak routery lub komputery, a etykietowane krawędzie mogą przedstawiać połączenia z informacją o przepustowości, opóźnieniach lub innych parametrach technicznych.
2. **Grafy przepływu pracy (workflow):**
   - Wierzchołki to zadania, a skierowane krawędzie reprezentują zależności czasowe lub kolejność wykonywania zadań. Etykiety mogą opisywać czas trwania operacji lub zasoby potrzebne do wykonania zadania.
3. **Języki formalne i automaty:**
   - Grafy skierowane etykietowane są często używane do modelowania automatów skończonych, gdzie wierzchołki reprezentują stany, a etykietowane krawędzie wskazują na przejścia między stanami w zależności od wejściowych symboli.
4. **Analiza sieci społecznościowych:**
   - W takim grafie wierzchołki mogą odpowiadać osobom, a skierowane krawędzie mogą reprezentować relacje społeczne (np. "A śledzi B" w mediach społecznościowych). Etykiety na krawędziach mogą wskazywać siłę relacji lub czas jej ustanowienia.
5. **Systemy zależności:**
   - Etykietowane grafy skierowane mogą być wykorzystywane w systemach zależności, na przykład w projektach, gdzie zależności między zadaniami, plikami, komponentami oprogramowania itp. są kluczowe dla poprawnej realizacji procesów.

Przykład:
Rozważmy etykietowany graf skierowany z trzema wierzchołkami A,B,CA, B, CA,B,C i krawędziami A→BA \rightarrow BA→B, B→CB \rightarrow CB→C oraz A→CA \rightarrow CA→C. Etykiety krawędzi mogą np. reprezentować koszty połączeń (np. A→BA \rightarrow BA→B ma etykietę 5, a B→CB \rightarrow CB→C ma etykietę 3).
W takim grafie możemy analizować zależności, ścieżki, przepływy, czy wykorzystywać go w zadaniach optymalizacyjnych, jak znajdowanie najkrótszej drogi między węzłami.
Podsumowanie:
Etykietowane grafy skierowane to potężne narzędzie do reprezentowania złożonych relacji w wielu dziedzinach, od informatyki i inżynierii po nauki społeczne i biologię.

K) E-biznes

47) Czym jest monada?

Monada to sposób, żeby "opakować" wartość i działać na niej, bez konieczności jej "wyjmowania" za każdym razem. Pomaga to radzić sobie z różnymi sytuacjami, takimi jak obsługa błędów, w prosty sposób.

Super prosty przykład: Monada "Może" (Maybe)
Wyobraź sobie, że masz operację, która czasem daje wynik, a czasem nie. Zamiast co chwilę sprawdzać, czy wynik jest dostępny, monada robi to za nas.
Problem:
Chcemy dodać 1 do liczby, ale czasami nie mamy liczby (np. może być pusta). Zamiast co chwilę sprawdzać, czy liczba istnieje, używamy monady.
Przykład:
Mamy liczbę albo pustą wartość:

- Just(5) oznacza, że mamy liczbę 5.
- Nothing oznacza, że nie mamy liczby.

Teraz chcemy dodać 1 do tej liczby, ale nie chcemy martwić się, czy ta liczba istnieje. Monada robi to za nas.

Jak to działa:

1. **Działamy na wartości, jeśli istnieje:**
    - Just(5) → dodaj 1 → mamy Just(6).
2. **Jeśli nie ma wartości (czyli jest Nothing), nic nie robimy:**
    - Nothing → próbujemy dodać 1 → zostaje Nothing.

Monada automatycznie "przepuszcza" wartość lub nic nie robi, jeśli nie ma na czym działać.

Przykład w pseudokodzie:

haskell

Skopiuj kod

```
Just(5) >>= (\x -> Just(x + 1))  -- Wynik: Just(6)
Nothing >>= (\x -> Just(x + 1))  -- Wynik: Nothing
```

Monada sprawia, że nie musisz się martwić o sprawdzanie, czy masz liczbę — ona sama o to dba!

Podsumowanie:

Monada to taki sprytny "pojemnik", który automatycznie sprawdza, czy wartość istnieje, i pozwala działać na tej wartości bez potrzeby ręcznego sprawdzania na każdym kroku.

48) Czym są kontenery oraz obrazy w Dockerze?

W Dockerze **kontenery** i **obrazy** to kluczowe pojęcia, które są używane do tworzenia, uruchamiania i zarządzania aplikacjami w sposób efektywny i przenośny. Oto ich proste wyjaśnienia:

1. Obraz (Image)

Obraz to gotowa paczka, która zawiera wszystko, czego potrzebuje aplikacja do działania: kod źródłowy, biblioteki, zależności, narzędzia, konfiguracje i system operacyjny. Jest to coś w rodzaju „wzorca” dla uruchamiania kontenerów.

- **Nieruchomy:** Obraz jest statycznym bytem — nie zmienia się podczas działania.
- **Zdefiniowane środowisko:** Każdy obraz definiuje konkretne środowisko, w którym aplikacja zostanie uruchomiona.
- **Warstwy:** Obrazy Dockerowe są zbudowane z kilku warstw, gdzie każda warstwa to krok w budowaniu obrazu (np. instalacja systemu operacyjnego, dodanie zależności).

Przykład obrazu:

Wyobraź sobie obraz jako przepis na zupę. Przepis mówi, jakie składniki potrzebujesz (np. pomidory, cebula, przyprawy) i jak przygotować zupę. Tak samo obraz Docker zawiera wszystkie elementy potrzebne do uruchomienia aplikacji.

2. Kontener (Container)

Kontener to działający proces, który jest uruchomiony na podstawie obrazu. Jest to lekki, samodzielny, izolowany fragment, który wykonuje określone zadania.

- **Ruchomy:** Kontener to działający proces, który został uruchomiony na podstawie obrazu.
- **Izolacja:** Każdy kontener działa w swoim własnym, izolowanym środowisku, ale korzysta z zasobów systemu operacyjnego hosta, co sprawia, że jest bardziej wydajny niż tradycyjne maszyny wirtualne.
- **Przenośny:** Możesz przenieść kontener na inny system z Dockerem i uruchomić go tam bez żadnych zmian.

Przykład kontenera:

Jeśli obraz to przepis na zupę, kontener to zupa, którą faktycznie ugotowałeś. Obraz mówi, jak zrobić zupę, a kontener to już gotowa, działająca zupa.

Różnica między obrazem a kontenerem:

- **Obraz**: To przepis na aplikację – zawiera wszystko, czego aplikacja potrzebuje, ale nie wykonuje żadnych działań.
- **Kontener**: To działający „egzemplarz” aplikacji, uruchomiony na podstawie obrazu.

Podsumowanie:

- **Obraz**: Paczka, która zawiera wszystko, co potrzebne do uruchomienia aplikacji.
- **Kontener**: Działający proces aplikacji oparty na obrazie.

Kontenery są bardzo lekkie, przenośne i szybkie, dlatego Docker jest tak popularny w środowiskach deweloperskich i produkcyjnych.

49) Czym jest trait?

Trait to pojęcie w programowaniu obiektowym, które pozwala na współdzielenie funkcjonalności między różnymi klasami, bez potrzeby stosowania wielokrotnego dziedziczenia. Jest to konstrukcja obecna w niektórych językach programowania, takich jak Scala, PHP czy Rust, i służy do „mieszania" (mix-in) funkcji do klas. Cechy (Traits) są używane, aby współdzielić interfejsy i pola pomiędzy klasami. Są bardzo podobne do **interfejsów** w Javie 8. Cechy mogą być rozszerzane przez klasy i obiekty, jednak nie można stworzyć instancji danej cechy. Z tego powodu cechy nie przyjmują parametrów wartości.

1. Trait w języku Scala:

W Scali **trait** jest podobny do abstrakcyjnej klasy, ale z kilkoma istotnymi różnicami. Trait może zawierać zarówno metody abstrakcyjne (bez implementacji), jak i konkretne (z implementacją). Jednak nie może być instancjonowany samodzielnie, czyli nie można utworzyć obiektu bezpośrednio z traitu.

Główne cechy:

- **Wielokrotne dziedziczenie:** Scala wspiera jedynie pojedyncze dziedziczenie klas, ale przy użyciu traitów można uzyskać efekt podobny do wielokrotnego dziedziczenia.
- **Mix-in:** Trait można „zmieszać" (mix-in) z klasą, co oznacza, że klasa może dziedziczyć właściwości i metody z wielu traitów.

Przykład w Scali:

scala
Skopiuj kod

```
trait Flyable {
  def fly(): Unit = println("Flying")
}

trait Swimmable {
  def swim(): Unit = println("Swimming")
}

class Bird extends Flyable with Swimmable

val bird = new Bird
bird.fly()  // Output: Flying
bird.swim() // Output: Swimming
```

W tym przykładzie klasa Bird używa dwóch traitów (Flyable i Swimmable), co pozwala jej dziedziczyć zarówno metodę fly(), jak i swim().

2. Trait w języku PHP:

PHP wspiera jedynie pojedyncze dziedziczenie klas, co oznacza, że klasa może dziedziczyć tylko po jednej klasie nadrzędnej. **Traits** w PHP rozwiązują problem braku wielokrotnego dziedziczenia, umożliwiając dziedziczenie metod i właściwości z wielu źródeł.

Główne cechy:

- **Metody współdzielone:** Traits pozwalają na definiowanie metod, które można używać w wielu klasach.
- **Deklaracja metod:** Trait może zawierać zarówno konkretne metody, jak i abstrakcyjne metody, które muszą być zaimplementowane w klasach korzystających z tego traitu.
- **Dowolny dostęp do metod:** Metody w trait mogą mieć dowolny modyfikator dostępu (public, protected, private).

Przykład w PHP:

```
php
Skopiuj kod
trait Logger {
  public function log($message) {
    echo $message;
  }
}

trait FileWriter {
  public function writeFile($filename, $content) {
    file_put_contents($filename, $content);
  }
}

class MyApp {
  use Logger, FileWriter;
}

$app = new MyApp();
$app->log("Logging message."); // Output: Logging message.
$app->writeFile('file.txt', 'Hello, World!');
```

W tym przykładzie klasa MyApp korzysta z dwóch traitów: Logger i FileWriter. Dzięki temu klasa ta dziedziczy metody z obu traitów.

Podsumowanie:

- **Scala:** Trait w Scali jest zbliżony do abstrakcyjnej klasy i umożliwia wielokrotne dziedziczenie poprzez "mix-in".
- **PHP:** Trait w PHP rozwiązuje problem pojedynczego dziedziczenia, pozwalając na współdzielenie metod między klasami.

W obu przypadkach traity pozwalają na tworzenie elastycznych, wielokrotnego użycia komponentów, unikając problemów związanych z wielokrotnym dziedziczeniem klas.

50) Jak działa tzw. Companion Object?

Companion Object to pojęcie występujące w językach programowania takich jak **Scala** i odnosi się do specjalnego rodzaju obiektu, który jest powiązany z klasą o tej samej nazwie. Umożliwia on definiowanie metod i pól, które mogą być współdzielone przez wszystkie instancje danej klasy. Companion Object pozwala na połączenie funkcji statycznych z funkcjonalnością instancji klasy, co przypomina podejście znane z programowania obiektowego w językach takich jak Java.

Jak działa Companion Object?

1. **Powiązanie z klasą:**
    - Companion Object jest definiowany przy klasie i dzieli z nią tę samą nazwę.
    - Klasa i jej Companion Object mają dostęp do swoich prywatnych pól i metod, co umożliwia bardziej zaawansowaną interakcję między obiektem a instancją klasy.
2. **Metody statyczne:**
    - W przeciwieństwie do Javy, Scala nie posiada wbudowanej koncepcji metod statycznych w klasach. Companion Object jest sposobem na emulację tej funkcjonalności. Wszelkie metody umieszczone w Companion Object są współdzielone między wszystkimi instancjami klasy, co działa jak metody statyczne.
3. **Możliwość tworzenia instancji klasy:**
    - W Companion Object można często znaleźć metodę apply, która umożliwia tworzenie instancji klasy bez wywoływania operatora new. Dzięki temu tworzenie obiektów staje się bardziej zwięzłe i przypomina działanie funkcji fabrycznych.

Przykład działania Companion Object w Scali:

```
class Person(val name: String, val age: Int)

object Person {
  def apply(name: String, age: Int): Person = new Person(name, age)
  def adultAge: Int = 18
}

val john = Person("John", 25)  // Tworzenie instancji klasy Person bez użycia new
println(john.name)             // Output: John
println(Person.adultAge)       // Output: 18
```

Wyjaśnienie:

- **Klasa Person:** Definiuje strukturę obiektów Person, posiadając pola name i age.
- **Companion Object Person:** Ma metodę apply, dzięki której możemy tworzyć instancje klasy Person bez wywoływania new. Dodatkowo definiuje metodę adultAge, która jest współdzielona między wszystkie obiekty typu Person.
- **Tworzenie instancji:** Person("John", 25) wywołuje metodę apply z Companion Object, tworząc obiekt bez potrzeby korzystania z operatora new.

Cechy Companion Object:

1. **Statyczne metody i pola:** Służy do implementacji metod, które powinny być dostępne bez tworzenia instancji klasy (podobnie jak metody statyczne w Javie).
2. **Prywatny dostęp:** Companion Object i klasa, z którą jest związany, mają nawzajem dostęp do swoich prywatnych członków.
3. **Brak operatora new:** Dzięki metodzie apply instancje klas mogą być tworzone bez jawnego użycia operatora new.
4. **Zastępuje klasy narzędziowe:** W Javie często tworzy się klasy zawierające jedynie metody statyczne (np. Math). W Scali funkcje tego typu mogą być realizowane przy użyciu Companion Object.

Podsumowanie:

Companion Object w Scali jest potężnym narzędziem umożliwiającym tworzenie metod współdzielonych przez wszystkie instancje klasy oraz zapewniającym uproszczenie procesu tworzenia obiektów. Umożliwia realizację metod statycznych w bardziej funkcjonalnym stylu i zwiększa spójność kodu dzięki dostępowi do prywatnych członków klasy, z którą jest powiązany.

L) Kryptografia

51) Proszę wyjaśnić na czym polega kryptografia asymetryczna i jakie ma zastosowania?

Kryptografia asymetryczna to sposób szyfrowania danych, w którym wykorzystywane są dwa różne klucze: **klucz publiczny** i **klucz prywatny**. Jeden klucz (publiczny) służy do szyfrowania wiadomości, a drugi klucz (prywatny) do jej odszyfrowania.

Wyobraź sobie taki przykład:

1. **Klucz publiczny** to coś jak **otwarty zamek**, który możesz komuś wysłać. Każdy może użyć tego otwartego zamka, aby zamknąć skrzynkę z wiadomością (zaszyfrować wiadomość).
2. **Klucz prywatny** to **klucz do zamka**, który posiadasz tylko Ty. Tylko Ty możesz otworzyć skrzynkę i przeczytać wiadomość (odszyfrować wiadomość).

Przykład:

- Ania chce wysłać wiadomość do Tomka, ale chce, żeby tylko Tomek mógł ją odczytać.
- Tomek daje Ani **klucz publiczny** (otwarty zamek). Ania używa go, żeby zaszyfrować wiadomość, czyli zamknąć skrzynkę.

- Gdy Tomek otrzymuje tę zaszyfrowaną wiadomość, otwiera ją swoim **kluczem prywatnym** (kluczem do zamka), który tylko on posiada. Dzięki temu tylko Tomek może przeczytać wiadomość, mimo że każdy mógłby ją zaszyfrować.

Zastosowania kryptografii asymetrycznej:

1. **Szyfrowanie e-maili**: Gdy chcesz wysłać poufną wiadomość, używasz klucza publicznego odbiorcy, aby zaszyfrować wiadomość, a on używa swojego klucza prywatnego do jej odczytania.
2. **Podpisy cyfrowe**: Odbiorca może zweryfikować, że wiadomość rzeczywiście pochodzi od Ciebie (używając Twojego klucza publicznego), sprawdzając, czy wiadomość była zaszyfrowana Twoim kluczem prywatnym.
3. **Bezpieczne połączenia w Internecie**: Gdy łączysz się z bankiem lub sklepem online, używany jest klucz publiczny serwera, aby nawiązać bezpieczne połączenie, a serwer używa klucza prywatnego, aby odczytać zaszyfrowane dane.

To pozwala na bezpieczne przesyłanie informacji bez potrzeby wymiany tajnych kluczy przed rozpoczęciem komunikacji.

52) Proszę wyjaśnić czym jest i jakie narzędzia kryptograficzne wykorzystuje protokół TLS?

Protokół TLS (Transport Layer Security) to mechanizm, który zapewnia bezpieczną komunikację w internecie, np. podczas przeglądania stron internetowych, korzystania z bankowości online czy wysyłania e-maili. TLS szyfruje dane, aby nikt nie mógł ich przechwycić ani zmienić podczas przesyłania.

Jak działa TLS na prostym przykładzie:

Wyobraź sobie, że robisz zakupy w sklepie online i chcesz bezpiecznie podać numer karty kredytowej:

1. **Ręka na przywitanie (ustalanie połączenia)**:
    - Ty (klient) mówisz: "Chcę porozmawiać bezpiecznie."
    - Sklep (serwer) odpowiada: "Świetnie! Oto mój **certyfikat** – to taki dowód, że jestem naprawdę tym, za kogo się podaję."
2. **Wymiana kluczy (bezpieczne szyfrowanie)**:
    - Sklep wysyła Ci swój **klucz publiczny** (to jak otwarty zamek). Możesz teraz zaszyfrować swoje dane (np. numer karty) tym kluczem.
    - Ty (klient) zaszyfrowujesz dane i wysyłasz je do sklepu.
    - Tylko sklep ma **klucz prywatny**, który pozwala odszyfrować te dane. Dzięki temu nikt inny nie może ich odczytać.
3. **Bezpieczna transmisja**:
    - Teraz komunikacja jest szyfrowana, czyli cała rozmowa (np. podawanie numeru karty) jest zabezpieczona i tylko Ty i sklep możecie ją zrozumieć.

Narzędzia kryptograficzne używane w TLS:

1. **Szyfrowanie asymetryczne**:
    - W początkowej fazie, kiedy klient (np. przeglądarka) i serwer (np. strona sklepu) wymieniają klucze, używana jest kryptografia asymetryczna (jak w przykładzie z kluczem publicznym i prywatnym). Pozwala to na bezpieczną wymianę kluczy bez wcześniejszej znajomości między stronami.
2. **Szyfrowanie symetryczne**:
    - Po wymianie kluczy, do dalszej szyfrowanej komunikacji wykorzystywane jest **szyfrowanie symetryczne** (oba końce komunikacji używają tego samego tajnego klucza, ale jest on bezpiecznie ustalony dzięki kryptografii asymetrycznej). To szybszy sposób na szyfrowanie danych podczas reszty sesji.
3. **Certyfikaty**:
    - Serwer używa **certyfikatów cyfrowych**, które potwierdzają jego tożsamość (tak jak paszport w świecie rzeczywistym). Certyfikaty są wydawane przez zaufane instytucje, tzw. urzędy certyfikacji (CA).

Superprosty przykład:

- Wyobraź sobie, że Ty (klient) i sklep (serwer) chcecie rozmawiać bezpiecznie.
- Sklep pokazuje Ci dowód osobisty (certyfikat) i mówi: "Zaufaj mi, jestem tym, za kogo się podaję."

- Sklep daje Ci otwarty zamek (klucz publiczny), Ty zamykasz nim wiadomość (szyfrujesz dane) i wysyłasz. Tylko sklep ma klucz do otwarcia (klucz prywatny), więc tylko on może odczytać Twoje dane.

TLS jest używany na co dzień w przeglądarkach internetowych – oznaczany jako "https://" na początku adresu strony.

53) Proszę wymienić i krótko scharakteryzować główne przykłady ataków na systemy kryptograficzne.

Ataki na systemy kryptograficzne mają na celu złamanie zabezpieczeń lub uzyskanie dostępu do zaszyfrowanych danych bez odpowiednich uprawnień. Oto kilka głównych przykładów takich ataków:

1. Atak brute force (atak siłowy):
- **Opis**: Atakujący próbuje odgadnąć klucz szyfrujący poprzez sprawdzenie wszystkich możliwych kombinacji kluczy. Jest to metoda czasochłonna i zasobożerna, szczególnie jeśli długość klucza jest duża.
- **Środki zaradcze**: Wydłużenie długości kluczy, co znacznie zwiększa liczbę możliwych kombinacji do przetestowania.

2. Atak kryptoanalizy:
- **Opis**: Polega na analizie zaszyfrowanych wiadomości (i być może częściowo znanych tekstów jawnych), aby znaleźć słabości algorytmu szyfrowania i wydedukować klucz.
- **Środki zaradcze**: Stosowanie silnych i sprawdzonych algorytmów szyfrujących, które są odporne na takie analizy.

3. Atak man-in-the-middle (atak pośredni):
- **Opis**: Atakujący przechwytuje i modyfikuje komunikację między dwoma stronami bez ich wiedzy. Może np. podszywać się pod jedną ze stron, zmieniać treść komunikacji lub uzyskiwać dostęp do poufnych danych.
- **Środki zaradcze**: Używanie protokołów z autentykacją (np. TLS) oraz wymiana certyfikatów.

4. Atak na algorytmy kryptograficzne (atak na słabości algorytmu):
- **Opis**: Ataki tego typu wykorzystują słabości w samych algorytmach szyfrowania. Np. jeżeli algorytm ma wadę matematyczną, można ją wykorzystać do odczytania danych bez znajomości klucza.
- **Środki zaradcze**: Regularne aktualizacje i stosowanie algorytmów, które przeszły dokładne testy i analizy.

5. Atak z odszyfrowaniem wybranego tekstu jawnego (chosen-plaintext attack):
- **Opis**: Atakujący uzyskuje możliwość wybrania tekstu jawnego, który chce zaszyfrować, a następnie analizuje wynik (tekst zaszyfrowany), aby zdobyć informacje o kluczu szyfrowania.
- **Środki zaradcze**: Algorytmy muszą być odporne na takie ataki, np. stosowanie losowych wartości (IV - wektora inicjalizującego) podczas szyfrowania.

6. Atak z odszyfrowaniem wybranego tekstu zaszyfrowanego (chosen-ciphertext attack):
- **Opis**: Atakujący zdobywa możliwość odszyfrowania wybranych fragmentów tekstu zaszyfrowanego i analizuje wyniki w celu wydedukowania klucza lub mechanizmu szyfrowania.
- **Środki zaradcze**: Wprowadzenie dodatkowych zabezpieczeń w algorytmie szyfrowania, takich jak podpisy cyfrowe czy uwierzytelnianie danych.

7. Atak bocznokanałowy:
- **Opis**: Ten atak nie dotyczy bezpośrednio samego algorytmu, ale sposobu jego implementacji. Atakujący wykorzystuje dane poboczne, takie jak zużycie energii, czas operacji, emisje elektromagnetyczne, aby wydedukować klucz szyfrujący.
- **Środki zaradcze**: Fizyczne zabezpieczenia sprzętu oraz optymalizacja algorytmów pod kątem odporności na takie ataki.

8. Atak powtórzeniowy (replay attack):
- **Opis**: Atakujący przechwytuje zaszyfrowaną wiadomość i wysyła ją ponownie w celu uzyskania tego samego efektu, np. przeprowadzenia tej samej transakcji.

- **Środki zaradcze**: Dodawanie znaczników czasu i unikalnych identyfikatorów do wiadomości, co uniemożliwia ich ponowne wykorzystanie.

9. Atak na hasze (kolizje):

- **Opis**: Atak polega na znalezieniu dwóch różnych wiadomości, które dają ten sam skrót (hash). Może to pozwolić na podsunięcie fałszywego dokumentu jako autentycznego.
- **Środki zaradcze**: Stosowanie algorytmów haszujących odpornych na kolizje (np. SHA-256).

Ataki na systemy kryptograficzne stale się rozwijają, dlatego niezwykle ważne jest ciągłe doskonalenie algorytmów i stosowanie najlepszych praktyk zabezpieczeń.

M) Głębokie sieci neuronowe

54) Sieci neuronowe: podstawowe informacje. Z czego się składają, jak się je uczy.

Sieci neuronowe: podstawowe informacje

Sieci neuronowe to struktury inspirowane działaniem ludzkiego mózgu, które są wykorzystywane w sztucznej inteligencji do rozwiązywania różnorodnych problemów, takich jak rozpoznawanie obrazów, przetwarzanie języka naturalnego, prognozowanie danych i wiele innych. Są one podstawą wielu nowoczesnych technologii AI.

Z czego składają się sieci neuronowe?

1. **Neurony (komórki)**:
   - Podstawowym elementem sieci jest **neuron** (inaczej sztuczny neuron lub perceptron). Neuron odbiera sygnały od innych neuronów, przetwarza je i generuje wyjście, które może zostać przekazane dalej.
2. **Warstwy**: Sieć neuronowa składa się z kilku warstw neuronów:
   - **Warstwa wejściowa**: Przyjmuje dane wejściowe (np. obraz, tekst). Każdy neuron w tej warstwie odpowiada za jedno z wejść.
   - **Warstwy ukryte**: Warstwy między wejściem a wyjściem, które przetwarzają informacje. Liczba warstw i neuronów w nich wpływa na zdolności sieci do rozwiązywania skomplikowanych problemów.
   - **Warstwa wyjściowa**: Generuje wynik końcowy, który może być klasyfikacją, prognozą, decyzją, itp.
3. **Połączenia między neuronami (synapsy)**:
   - Neurony w różnych warstwach są połączone ze sobą. Każde takie połączenie ma swoją **wagę**. Waga określa, jak silny wpływ dany neuron ma na kolejny neuron w sieci.
4. **Funkcja aktywacji**:
   - Każdy neuron ma funkcję aktywacji, która przekształca sumę swoich wejść na określoną wartość wyjściową. Przykłady popularnych funkcji aktywacji to **ReLU** (Rectified Linear Unit), **sigmoida** i **tanh**.

Jak uczy się sieci neuronowe?

Proces uczenia sieci neuronowej to dostosowywanie wag połączeń między neuronami, aby sieć lepiej wykonywała swoje zadanie, na przykład rozpoznawała obiekty na obrazach lub klasyfikowała teksty. Proces ten nazywa się **trenowaniem** sieci.

Kluczowe etapy uczenia sieci neuronowej:

1. **Inicjalizacja**:
   - Na początku każda waga w sieci jest ustawiona losowo. Sieć nie ma jeszcze pojęcia, jak dobrze wykonywać zadanie.
2. **Przekazywanie danych (forward propagation)**:
   - Dane wejściowe są wprowadzane do sieci przez warstwę wejściową, przetwarzane przez warstwy ukryte, a na końcu sieć generuje wynik w warstwie wyjściowej. Ten proces nazywa się **propagacją w przód**.
3. **Funkcja kosztu (loss function)**:
   - Sieć porównuje wynik, który wygenerowała, z rzeczywistą wartością (oczekiwanym wynikiem). Na podstawie tej różnicy (błędu) obliczana jest **funkcja kosztu** – miara, jak bardzo wynik sieci różni się od oczekiwanego wyniku.

4. **Propagacja wsteczna (backpropagation)**:
    - Proces **propagacji wstecznej** polega na przekazywaniu błędów z warstwy wyjściowej z powrotem do wcześniejszych warstw, aby obliczyć, jak zmienić wagi. Sieć "uczy się" na podstawie tych błędów.
5. **Aktualizacja wag**:
    - Wagi są aktualizowane przy użyciu algorytmu **optymalizacji**. Najpopularniejszym algorytmem jest **gradient descent** (spadek gradientu), który minimalizuje funkcję kosztu, dostosowując wagi tak, aby wyniki sieci były coraz lepsze.
    - **Learning rate** (tempo uczenia) kontroluje, jak duże kroki sieć wykonuje podczas aktualizowania wag. Zbyt małe tempo może spowolnić proces uczenia, a zbyt duże może prowadzić do niestabilności.
6. **Powtarzanie procesu**:
    - Proces przekazywania danych, propagacji wstecznej i aktualizacji wag powtarza się wiele razy (tzw. **epoki**), aż sieć osiągnie akceptowalny poziom dokładności.

Typy sieci neuronowych:

- **Sieci wielowarstwowe (MLP – Multi-Layer Perceptron)**: Klasyczne sieci neuronowe, składające się z wielu warstw ukrytych, używane do klasyfikacji i regresji.
- **Sieci konwolucyjne (CNN)**: Używane głównie do przetwarzania obrazów i analizy danych przestrzennych.
- **Sieci rekurencyjne (RNN)**: Wykorzystywane do analizy sekwencji danych, takich jak teksty lub dane czasowe, ponieważ potrafią zapamiętywać informacje z poprzednich kroków.

Podsumowanie

Sieci neuronowe składają się z neuronów, które są połączone wagami i umieszczone w warstwach. Uczą się poprzez dostosowywanie wag na podstawie błędów między przewidywaniami a rzeczywistymi wynikami, co jest osiągane dzięki procesowi propagacji wstecznej i optymalizacji.

55) Konwolucyjne sieci neuronowe.

Konwolucyjne Sieci Neuronowe (CNN) – Podstawowe Informacje

Konwolucyjne sieci neuronowe (CNN, z ang. **Convolutional Neural Networks**) to rodzaj sieci neuronowych, które są szczególnie efektywne w analizie danych przestrzennych, takich jak obrazy, sygnały czy dane o strukturze 2D. CNN zrewolucjonizowały dziedziny związane z przetwarzaniem obrazów i rozpoznawaniem wzorców, np. w systemach rozpoznawania twarzy, wykrywaniu obiektów, medycynie czy autonomicznych pojazdach.

Z czego składa się CNN?

1. **Warstwa konwolucyjna (convolutional layer)**:
    - To podstawowy blok CNN. W tej warstwie stosowane są **filtry (kernels)**, które przesuwają się po obrazie (lub innym wejściu) i wyodrębniają z niego cechy, np. krawędzie, tekstury, kształty. Każdy filtr analizuje obraz pod kątem innych cech.
    - **Operacja konwolucji** polega na mnożeniu wartości pikseli wejściowych przez wartości filtru i sumowaniu wyników. Powstaje tzw. **mapa cech (feature map)**, która jest wynikiem działania filtru na danym fragmencie obrazu.
2. **Funkcja aktywacji (najczęściej ReLU)**:
    - Po przejściu przez warstwę konwolucyjną, dane są przetwarzane przez **funkcję aktywacji**, która wprowadza nieliniowość do modelu, co pozwala sieci lepiej uczyć się złożonych wzorców. Najczęściej stosowaną funkcją aktywacji jest **ReLU** (Rectified Linear Unit), która zastępuje wszystkie wartości ujemne w macierzy cech zerami.
3. **Warstwa subsamplingu (pooling layer)**:
    - Warstwa ta służy do **zmniejszania wymiarów** danych (redukcji rozdzielczości obrazu), co zmniejsza liczbę parametrów i obliczeń, a jednocześnie zachowuje istotne cechy. Najczęściej stosowany jest **max pooling**, gdzie wybierana jest maksymalna wartość z grupy sąsiadujących pikseli (np. z 2x2 pikseli wybierany jest największy).
4. **Warstwy w pełni połączone (fully connected layers)**:

- Na końcu sieci konwolucyjnej stosuje się kilka warstw w pełni połączonych, które działają podobnie do tradycyjnych sieci neuronowych. Służą one do podejmowania ostatecznych decyzji, takich jak klasyfikacja obrazu. Dane z poprzednich warstw są przekształcane w jednowymiarowy wektor (ang. flattening) i przekazywane do tych warstw.

Jak działa CNN? – Prosty przykład

Załóżmy, że mamy obraz kota, który chcemy zaklasyfikować jako "kot". Oto jak działa CNN krok po kroku:

1. **Warstwa konwolucyjna**: Filtry analizują różne cechy obrazu, np. krawędzie uszu, teksturę futra. Każdy filtr przesuwa się po obrazie i generuje mapę cech.
2. **Funkcja aktywacji**: Mapy cech są przepuszczane przez funkcję ReLU, aby wyeliminować wartości ujemne i zachować najważniejsze informacje.
3. **Pooling**: Obraz jest zmniejszany, ale kluczowe informacje (np. kształt głowy kota) zostają zachowane. Dzięki temu sieć jest bardziej odporna na przesunięcia i zniekształcenia.
4. **Warstwy w pełni połączone**: Na końcu sieć przekształca uzyskane dane w jednowymiarowy wektor, który jest wykorzystywany do ostatecznej klasyfikacji. Wynikiem może być stwierdzenie: „To jest kot” z określonym prawdopodobieństwem.

Kluczowe cechy CNN:

1. **Lokalna konwolucja**: Zamiast analizować cały obraz na raz, CNN przetwarzają lokalne fragmenty (np. 3x3 piksele), co sprawia, że są bardziej wydajne i skupiają się na drobnych cechach.
2. **Udostępnianie wag (weight sharing)**: Wszystkie neurony w danej warstwie używają tych samych filtrów, co redukuje liczbę parametrów w sieci, a tym samym zmniejsza ryzyko przeuczenia i przyspiesza proces trenowania.
3. **Hierarchiczna struktura cech**: Na niższych warstwach CNN wyodrębniają proste cechy, takie jak krawędzie, a na wyższych bardziej złożone wzorce, np. kształty całych obiektów.

Proces uczenia CNN

Uczenie CNN odbywa się w taki sam sposób jak tradycyjnych sieci neuronowych, poprzez:

1. **Przekazywanie danych (forward propagation)**: Dane wejściowe (np. obraz) są przetwarzane warstwa po warstwie, aż do uzyskania ostatecznego wyniku (np. klasyfikacji).
2. **Obliczanie funkcji kosztu (loss function)**: Różnica między przewidywanym wynikiem a rzeczywistą etykietą jest obliczana, co pozwala oszacować, jak daleko sieć była od prawidłowego wyniku.
3. **Propagacja wsteczna (backpropagation)**: Błędy są przekazywane z powrotem przez sieć, a wagi filtrów i połączeń są dostosowywane w celu poprawy przyszłych wyników.
4. **Optymalizacja (np. gradient descent)**: Algorytmy optymalizacyjne zmieniają wagi sieci tak, aby zminimalizować błąd.

Zastosowania CNN

1. **Rozpoznawanie obrazów**: CNN są szeroko stosowane w systemach rozpoznawania obiektów, twarzy, pojazdów itp.
2. **Analiza wideo**: Wykorzystywane do analizy klatek wideo, np. w monitoringu, systemach bezpieczeństwa.
3. **Przetwarzanie medyczne**: Stosowane w diagnostyce obrazowej, np. do wykrywania zmian chorobowych na zdjęciach rentgenowskich czy tomograficznych.
4. **Systemy autonomiczne**: CNN są kluczowe w samochodach autonomicznych do identyfikacji przeszkód, znaków drogowych i innych pojazdów.

Podsumowując, konwolucyjne sieci neuronowe to zaawansowane narzędzie przetwarzania danych przestrzennych, które doskonale nadają się do analizy obrazów, dzięki zdolności do automatycznego wyodrębniania cech z danych wejściowych i hierarchicznej struktury przetwarzania informacji.

56) Rekurencyjne sieci neuronowe.

Rekurencyjne Sieci Neuronowe (RNN) – Podstawowe Informacje

Rekurencyjne sieci neuronowe (RNN, z ang. **Recurrent Neural Networks**) to rodzaj sieci neuronowych, które są zaprojektowane do przetwarzania danych sekwencyjnych, takich jak teksty, sygnały dźwiękowe,

dane czasowe czy serie finansowe. W odróżnieniu od tradycyjnych sieci neuronowych, RNN potrafią zapamiętywać informacje z poprzednich kroków (stanów) i uwzględniać je podczas przetwarzania aktualnych danych, co czyni je idealnymi do zadań związanych z danymi o zależnościach czasowych.

Z czego składa się RNN?

1. **Neurony rekurencyjne**:
    - Każdy neuron w RNN ma połączenie zwrotne (rekurencyjne), co oznacza, że jego wyjście może zostać wprowadzone z powrotem jako wejście w następnym kroku czasowym.
    - Dzięki temu RNN mogą pamiętać poprzednie stany i brać je pod uwagę przy podejmowaniu kolejnych decyzji.
2. **Warstwy rekurencyjne**:
    - W RNN dane są przetwarzane w sposób sekwencyjny – na przykład w analizie tekstu każdy neuron przetwarza jedno słowo w zdaniu, korzystając z informacji o poprzednich słowach.
    - Wyjście każdej warstwy jest przekazywane do kolejnej oraz do siebie samej w kolejnych krokach czasowych.
3. **Stan ukryty (hidden state)**:
    - Kluczową cechą RNN jest to, że przechowują one informacje o poprzednich wejściach w **stanie ukrytym**, który jest aktualizowany z każdym krokiem czasowym. Stan ten działa jak "pamięć" sieci.

Jak działa RNN?

RNN przetwarza sekwencje danych krok po kroku, uwzględniając zarówno bieżące dane wejściowe, jak i wcześniejsze stany. Oto jak to działa krok po kroku:

1. **Wejście**: Do RNN wprowadzana jest sekwencja danych (np. zdanie podzielone na słowa). Sieć przetwarza jedno słowo na raz.
2. **Stan ukryty**: Przy każdym kroku czasowym, sieć bierze pod uwagę bieżące słowo oraz poprzedni **stan ukryty**, co pozwala jej "pamiętać", jakie były poprzednie słowa w zdaniu.
3. **Wyjście**: Na każdym kroku czasowym sieć generuje wyjście, które może być interpretowane jako wynik (np. predykcja kolejnego słowa w zdaniu).
4. **Aktualizacja stanu ukrytego**: Stan ukryty jest aktualizowany w każdym kroku, co pozwala sieci na "uczenie się" zależności między wcześniejszymi a bieżącymi danymi.

Przykład działania RNN:

Załóżmy, że mamy zadanie przewidywania kolejnych słów w zdaniu. Na przykład, zaczynamy od zdania "Dzisiaj jest bardzo". Sieć rekurencyjna może przewidzieć, że kolejne słowo będzie "ciepło", ponieważ nauczyła się, że w kontekście wcześniejszych słów istnieje duże prawdopodobieństwo, że to słowo będzie odpowiednie.

Problemy z klasycznymi RNN:

1. **Zanikanie gradientu**: W klasycznych RNN występuje problem z tzw. **zanikaniem gradientu**. Gdy sieć jest bardzo głęboka (długie sekwencje danych), gradienty z propagacji wstecznej (backpropagation) mogą stawać się coraz mniejsze, co utrudnia sieci efektywne uczenie się z bardzo długich sekwencji.
2. **Problemy z pamięcią długoterminową**: Klasyczne RNN mają trudności z zapamiętywaniem informacji na długie odległości czasowe, co ogranicza ich zdolności do pracy z bardzo długimi sekwencjami.

Ulepszone wersje RNN:

Aby poradzić sobie z powyższymi problemami, stworzono ulepszone warianty RNN, takie jak:

1. **LSTM (Long Short-Term Memory)**:
    - **LSTM** to zaawansowana wersja RNN, która rozwiązuje problem zanikającego gradientu i lepiej radzi sobie z pamięcią długoterminową.
    - LSTM wprowadza tzw. **bramki** (gate), które decydują, które informacje mają zostać "zapamiętane", które mają być "zapomniane" i które mają zostać przekazane dalej. Dzięki temu LSTM może lepiej przechowywać istotne informacje przez dłuższy czas.
2. **GRU (Gated Recurrent Unit)**:
    - **GRU** to uproszczona wersja LSTM, która również rozwiązuje problem zanikającego gradientu, ale jest mniej skomplikowana i szybsza w obliczeniach. Wprowadza mniej

bramek niż LSTM, co czyni ją prostszą w implementacji, ale równie efektywną w wielu zadaniach.

Zastosowania RNN:

1. **Przetwarzanie języka naturalnego (NLP)**:
   - RNN są szeroko stosowane w analizie i generowaniu tekstu, tłumaczeniu maszynowym, analizie sentymentu, rozpoznawaniu mowy, generowaniu tekstu itp.
2. **Analiza danych czasowych**:
   - RNN znajdują zastosowanie w analizie danych czasowych, np. prognozowaniu danych finansowych, przetwarzaniu sygnałów audio, analizie sekwencji w biologii itp.
3. **Generowanie sekwencji**:
   - RNN mogą generować sekwencje danych, np. tworzyć nowe fragmenty muzyki, generować opisy obrazów, automatycznie tworzyć teksty lub przewidywać kolejne klatki w filmach.
4. **Rozpoznawanie mowy**:
   - W systemach rozpoznawania mowy RNN są wykorzystywane do przetwarzania sekwencji dźwięków i konwertowania ich na tekst.

Jak uczy się RNN?

1. **Przekazywanie danych (forward propagation)**: Dane wejściowe są przetwarzane krok po kroku przez neurony rekurencyjne, z uwzględnieniem stanu ukrytego z poprzednich kroków czasowych.
2. **Funkcja kosztu (loss function)**: Różnica między przewidywaniami a rzeczywistymi wynikami jest obliczana, aby zmierzyć, jak daleko sieć była od prawidłowego wyniku.
3. **Propagacja wsteczna przez czas (BPTT)**: W RNN używa się specjalnej wersji propagacji wstecznej, zwanej **Backpropagation Through Time (BPTT)**, która bierze pod uwagę zależności czasowe w danych.
4. **Aktualizacja wag**: Wagi w sieci są dostosowywane przy użyciu algorytmów optymalizacyjnych (np. gradient descent), aby zminimalizować błąd i poprawić przyszłe prognozy.

Podsumowanie:

Rekurencyjne sieci neuronowe (RNN) są potężnym narzędziem do przetwarzania sekwencji danych, szczególnie tam, gdzie istotne są zależności czasowe. Pomimo swoich ograniczeń, takich jak problem zanikającego gradientu, RNN i ich bardziej zaawansowane wersje, takie jak LSTM i GRU, są niezwykle efektywne w zadaniach związanych z analizą tekstu, rozpoznawaniem mowy, prognozowaniem danych czasowych i innych sekwencji.

57) Generative adversarial networks.

Generative Adversarial Networks (GAN) – Podstawowe Informacje

Generative Adversarial Networks (GAN) to rodzaj sieci neuronowych wprowadzony w 2014 roku przez Iana Goodfellowa, który rewolucjonizował dziedziny generatywnej sztucznej inteligencji. GAN-y są używane do generowania nowych danych na podstawie istniejących, takich jak obrazy, teksty, dźwięki czy wideo. Ich unikalną cechą jest to, że składają się z dwóch sieci neuronowych, które rywalizują ze sobą, co prowadzi do generowania coraz bardziej realistycznych wyników.

Z czego składa się GAN?

GAN składa się z dwóch głównych komponentów:

1. **Generator**:
   - To sieć neuronowa, której zadaniem jest **tworzenie nowych, syntetycznych danych** (np. obrazów). Generator bierze losowe wartości (tzw. **szum**), a następnie przekształca je w dane podobne do tych rzeczywistych, np. w realistyczne obrazy.
   - Na początku generowane dane mogą wyglądać bardzo nienaturalnie, ale generator uczy się na podstawie rywalizacji z drugą siecią.
2. **Dyskryminator**:
   - To druga sieć neuronowa, której zadaniem jest **rozróżnianie między prawdziwymi danymi (pochodzącymi z rzeczywistego zbioru) a danymi wygenerowanymi przez generator**.
   - Dyskryminator klasyfikuje dane jako "prawdziwe" lub "fałszywe". Uczy się tego na podstawie rzeczywistych danych z zestawu treningowego i prób, które generuje generator.

Jak działa GAN?

GAN działa na zasadzie rywalizacji (stąd "adversarial" w nazwie), gdzie generator i dyskryminator próbują wzajemnie się pokonać. Oto jak to wygląda krok po kroku:

1. **Generator tworzy dane**:
    - Generator zaczyna od losowego szumu (np. losowych wartości) i przekształca go w dane wyjściowe, np. obraz. Na początku obrazy są bardzo niskiej jakości.
2. **Dyskryminator klasyfikuje dane**:
    - Dyskryminator otrzymuje dwa rodzaje danych: dane wygenerowane przez generator oraz dane prawdziwe (z rzeczywistego zbioru). Jego zadaniem jest odróżnienie, które dane są prawdziwe, a które fałszywe.
3. **Rywalizacja (gra zero-jedynkowa)**:
    - Celem generatora jest "oszukanie" dyskryminatora – wygenerowanie takich danych, które dyskryminator uzna za prawdziwe.
    - Celem dyskryminatora jest poprawne rozpoznanie, które dane są fałszywe (wygenerowane), a które są prawdziwe.
4. **Uczenie się**:
    - Oba modele są uczone za pomocą **propagacji wstecznej**. Dyskryminator uczy się lepiej rozróżniać między prawdziwymi a fałszywymi danymi, a generator uczy się tworzyć coraz bardziej realistyczne dane.
    - Z czasem generator staje się tak dobry w tworzeniu realistycznych danych, że dyskryminator ma trudności z ich rozróżnianiem.
5. **Końcowy wynik**:
    - W miarę postępów treningu, generator staje się na tyle skuteczny, że jest w stanie tworzyć dane, które są niemal nieodróżnialne od rzeczywistych, a dyskryminator staje się coraz trudniejszy do oszukania.

Kluczowe cechy GAN:

1. **Rywalizacja (adwersarialność)**: Unikalny sposób treningu, w którym dwa modele rywalizują ze sobą, co prowadzi do stopniowego doskonalenia obu sieci.
2. **Generowanie nowych danych**: GAN może tworzyć dane, które przypominają rzeczywiste dane – od obrazów twarzy, przez muzykę, aż po teksty.
3. **Brak bezpośredniego nadzoru**: GAN nie potrzebuje etykietowanych danych treningowych (jak w klasyfikacji), tylko zbioru danych rzeczywistych.

Zastosowania GAN:

1. **Generowanie obrazów**:
    - GAN są wykorzystywane do generowania realistycznych obrazów, takich jak twarze, krajobrazy, obiekty 3D. Przykładem są systemy, które mogą tworzyć zdjęcia nieistniejących osób (np. "This Person Does Not Exist").
2. **Super-rozdzielczość obrazów (Image Super-Resolution)**:
    - GAN są używane do przekształcania obrazów o niskiej rozdzielczości w obrazy o wysokiej rozdzielczości, co jest przydatne w medycynie, astronomii i innych dziedzinach wymagających analizy szczegółowych obrazów.
3. **Zamiana stylu (Style Transfer)**:
    - GAN mogą zmieniać styl obrazów, np. przekształcać zdjęcia w obrazy malowane w stylu konkretnego artysty lub zmieniać wygląd obiektów.
4. **Tworzenie sztucznej muzyki i dźwięku**:
    - GAN mogą generować nowe utwory muzyczne, syntezować dźwięki lub modyfikować istniejące dźwięki.
5. **Generowanie danych treningowych**:
    - GAN są używane do generowania sztucznych danych, które mogą posłużyć jako dodatkowy zbiór do treningu innych modeli maszynowego uczenia.
6. **Deepfake**:

- GAN są używane w technologii deepfake, która pozwala na realistyczne zmienianie twarzy lub głosu w filmach, co ma swoje kontrowersyjne zastosowania, np. tworzenie fałszywych nagrań.

Wyzwania związane z GAN:

1. **Trudność w trenowaniu**:
   - Trening GAN może być niestabilny. Jeśli dyskryminator jest zbyt silny, generator może mieć trudności z poprawnym uczeniem się, ponieważ dyskryminator będzie zbyt łatwo rozpoznawać fałszywe dane. Z drugiej strony, jeśli generator zdominuje, dyskryminator przestanie dobrze rozróżniać.
2. **Mode collapse**:
   - To problem, w którym generator "nauczy się" tworzyć bardzo podobne dane, zamiast różnorodnych, co prowadzi do utraty różnorodności w generowanych wynikach.
3. **Potencjalne nadużycia**:
   - GAN mogą być używane do nieetycznych celów, takich jak tworzenie deepfake'ów, co stwarza zagrożenia związane z fałszowaniem tożsamości i informacji.

Podsumowanie:

Generative Adversarial Networks to potężne narzędzie do generowania realistycznych danych, które znajdują zastosowanie w wielu dziedzinach, od obrazów, przez dźwięki, po teksty. Ich unikalna architektura, oparta na rywalizacji między generatorem a dyskryminatorem, pozwala na tworzenie danych, które są niemal nieodróżnialne od rzeczywistych. Mimo to, trenowanie GAN wiąże się z wyzwaniami, ale ich potencjał do tworzenia innowacyjnych rozwiązań jest ogromny.

58) Adversarial examples.

Adversarial Examples – Podstawowe Informacje

Adversarial examples (przykłady atakujące) to sztucznie stworzone dane wejściowe, które zostały zmodyfikowane w taki sposób, aby wprowadzić w błąd model sztucznej inteligencji, np. sieć neuronową, mimo że zmiany są niewidoczne lub trudno zauważalne dla człowieka. Przykłady atakujące są szczególnie istotne w kontekście bezpieczeństwa systemów AI, ponieważ pokazują, że nawet niewielkie zmiany w danych wejściowych mogą spowodować drastyczne zmiany w decyzjach modelu.

Jak działają adversarial examples?

Adversarial examples powstają poprzez wprowadzenie małych, celowych zakłóceń do danych wejściowych (np. obrazu), które są na tyle subtelne, że człowiek ich nie zauważy, ale model AI zostanie oszukany i da błędną predykcję.

Przykład działania:

1. **Przykład poprawny**: Sieć neuronowa może poprawnie klasyfikować obraz psa jako "pies".
2. **Przykład atakujący**: Na ten sam obraz można dodać drobne, nieznaczne zakłócenia (niezauważalne dla człowieka), co może spowodować, że sieć neuronowa sklasyfikuje psa jako "samochód" lub "kota".

Jak tworzy się adversarial examples?

Adversarial examples są generowane za pomocą optymalizacji, polegającej na minimalnych zmianach w danych wejściowych w celu maksymalizacji błędów modelu. Przykładowo, popularną metodą generowania takich przykładów jest **Fast Gradient Sign Method (FGSM)**, która dodaje zakłócenia do obrazu na podstawie gradientów błędu sieci.

1. **Krok 1**: Obliczenie gradientu błędu dla danego przykładu wejściowego (np. obrazu psa).
2. **Krok 2**: Dodanie niewielkiej zmiany do pikseli obrazu w kierunku maksymalizacji tego błędu. Zakłócenia są tak subtelne, że dla człowieka obraz wydaje się taki sam.
3. **Krok 3**: Model, mimo że widzi praktycznie ten sam obraz, daje błędną predykcję.

Wpływ na modele AI:

1. **Podatność modeli**: Adversarial examples pokazują, że nawet zaawansowane modele AI, takie jak sieci neuronowe, są podatne na małe zakłócenia, które mogą prowadzić do błędnych decyzji.
2. **Brak intuicyjnej odporności**: Dla człowieka takie zmiany są zwykle nieistotne i niezauważalne, co wskazuje, że modele AI nie mają ludzkiej intuicji w ocenie danych wejściowych.

Zastosowania adversarial examples:

1. **Testowanie odporności systemów AI**: Adversarial examples są używane do testowania bezpieczeństwa modeli AI, aby upewnić się, że są one odporne na takie ataki.
2. **Ataki na systemy AI**: Mogą być wykorzystane w nieetycznych celach, np. do atakowania systemów rozpoznawania twarzy, autonomicznych pojazdów czy systemów bezpieczeństwa, wprowadzając je w błąd.

Przykłady zastosowań w realnym świecie:

1. **Systemy rozpoznawania obrazów**: Wystarczy dodać drobne zmiany do obrazu, aby model, który wcześniej poprawnie klasyfikował obiekt (np. znak stopu), nagle sklasyfikował go jako zupełnie inny obiekt (np. znak ograniczenia prędkości).
2. **Autonomiczne pojazdy**: W przypadku samochodów autonomicznych, niewielkie zakłócenia w obrazie z kamer mogą sprawić, że pojazd zareaguje nieprawidłowo, np. nie rozpozna znaku drogowego, co może prowadzić do niebezpiecznych sytuacji.
3. **Systemy rozpoznawania twarzy**: Niewielkie zmiany w obrazie twarzy mogą spowodować, że system błędnie rozpozna tożsamość osoby.

Sposoby ochrony przed adversarial examples:

1. **Adversarial training**: Polega na trenowaniu modelu na przykładach atakujących, aby zwiększyć jego odporność na takie zakłócenia.
2. **Regularization techniques**: Techniki regularizacji, takie jak Dropout, mogą pomóc w redukcji podatności na zakłócenia, ale nie są one wystarczające same w sobie.
3. **Detekcja przykładów atakujących**: Badacze pracują nad metodami, które pozwalają na wykrywanie i odrzucanie przykładów, które zostały celowo zmienione w celu oszukania modelu.
4. **Zastosowanie bardziej złożonych architektur**: Niektóre architektury sieci neuronowych, takie jak **defensive distillation**, zostały opracowane w celu zwiększenia odporności na ataki adwersarialne.

Podsumowanie:

Adversarial examples to sztuczne, subtelnie zmienione dane, które mogą oszukać modele AI, powodując błędne wyniki. Stanowią one poważne zagrożenie dla bezpieczeństwa systemów opartych na sztucznej inteligencji, zwłaszcza w kontekście krytycznych aplikacji, takich jak autonomiczne pojazdy czy systemy bezpieczeństwa. Trwają prace nad metodami ochrony przed takimi atakami, ale zagadnienie to pozostaje jednym z kluczowych wyzwań w dziedzinie sztucznej inteligencji i uczenia maszynowego.

N) Projektowanie aplikacji internetowych

70) Omów architekturę Model-Widok-Kontroler (MVC) w kontekście działania aplikacji internetowej.

Architektura Model-Widok-Kontroler (MVC) w kontekście aplikacji internetowej

Architektura **Model-Widok-Kontroler** (MVC) to wzorzec projektowy stosowany w aplikacjach internetowych, który pozwala na lepszą organizację kodu poprzez oddzielenie logiki biznesowej, prezentacji i interakcji użytkownika. MVC dzieli aplikację na trzy komponenty: **Model**, **Widok** i **Kontroler**, co umożliwia lepszą strukturę i łatwiejszą konserwację kodu.

1. Model:

- **Opis**: Model reprezentuje dane oraz logikę biznesową aplikacji. Jest odpowiedzialny za zarządzanie danymi, ich przechowywanie, modyfikację i weryfikację. Obejmuje to interakcje z bazą danych oraz wszystkie operacje związane z przetwarzaniem danych.
- **Funkcje**:
    - Odpowiada za przechowywanie danych aplikacji.
    - Zawiera logikę biznesową, np. walidację danych, przetwarzanie transakcji.
    - Wysyła i odbiera dane z bazy danych.
- **Przykład w aplikacji internetowej**: W aplikacji sklepu internetowego Model zarządza produktami, zamówieniami i użytkownikami. Kiedy użytkownik chce dodać produkt do koszyka, Model odpowiada za aktualizację koszyka użytkownika w bazie danych.

2. Widok (View):

- **Opis**: Widok odpowiada za prezentację danych użytkownikowi. Obejmuje interfejs użytkownika (UI), który wyświetla informacje pochodzące z Modelu. Widok nie zawiera logiki biznesowej ani bezpośrednio nie manipuluje danymi, ale prezentuje dane w czytelny sposób, umożliwiając użytkownikowi interakcję.
- **Funkcje**:
    - Prezentuje dane użytkownikowi (np. w formie HTML, CSS, JavaScript).
    - Odbiera dane od Modelu i formatuje je do wyświetlenia.
    - Odbiera dane wejściowe od użytkownika, np. formularze.
- **Przykład w aplikacji internetowej**: Strona produktu w sklepie internetowym jest Widokiem, który pobiera informacje o produkcie z Modelu i wyświetla je w przyjaznym dla użytkownika interfejsie.

3. Kontroler (Controller):

- **Opis**: Kontroler pełni funkcję pośrednika pomiędzy Modelem a Widokiem. Przyjmuje żądania użytkownika (np. kliknięcie przycisku, wypełnienie formularza), przetwarza je i komunikuje się z Modelem w celu pobrania lub zmiany danych, a następnie przekazuje odpowiednie dane do Widoku. Kontroler nie zarządza danymi bezpośrednio, ale koordynuje przepływ informacji.
- **Funkcje**:
    - Odbiera żądania użytkownika (np. z URL).
    - Komunikuje się z Modelem w celu pobrania, zapisania lub aktualizacji danych.
    - Przekazuje dane do Widoku w celu ich prezentacji.
- **Przykład w aplikacji internetowej**: Kiedy użytkownik klika przycisk "Dodaj do koszyka", Kontroler odbiera to żądanie, prosi Model o aktualizację koszyka użytkownika, a następnie przekazuje wyniki tej operacji do Widoku (np. wyświetla zaktualizowany koszyk).

Przepływ danych w aplikacji internetowej opartej na MVC:

1. **Użytkownik wchodzi na stronę** (np. sklepu internetowego) i wykonuje akcję, np. wyszukuje produkt, wypełnia formularz, klika link.
2. **Żądanie trafia do Kontrolera**:
    - Kontroler odbiera żądanie użytkownika (np. dodanie produktu do koszyka).
    - Kontroler decyduje, jakie dane są potrzebne, i komunikuje się z Modelem.
3. **Model przetwarza dane**:
    - Model wykonuje operacje związane z danymi, np. pobiera produkt z bazy danych, aktualizuje koszyk użytkownika.
    - Przetworzone dane są przekazywane z Modelu do Kontrolera.
4. **Kontroler wysyła dane do Widoku**:
    - Kontroler przekazuje dane z Modelu do Widoku w celu ich wyświetlenia użytkownikowi.
5. **Widok prezentuje dane użytkownikowi**:
    - Widok formatuje dane (np. produkt, cena, stan koszyka) i wyświetla je w przeglądarce internetowej w formie HTML/CSS/JavaScript.

Zalety architektury MVC:

1. **Modularność i separacja zadań**:
    - Oddzielenie logiki biznesowej (Model), prezentacji (Widok) i interakcji (Kontroler) sprawia, że kod jest bardziej przejrzysty i łatwiejszy w utrzymaniu.
2. **Łatwość testowania**:
    - Każda część aplikacji może być testowana osobno – logika biznesowa Modelu, interfejs Widoku i logika sterowania Kontrolera.
3. **Reużywalność kodu**:
    - Widoki mogą być wielokrotnie używane z różnymi danymi. Podobnie, ten sam Model może być używany przez różne Kontrolery.
4. **Lepsza organizacja kodu**:
    - Podział na Model, Widok i Kontroler zmniejsza ryzyko powstawania złożonego i trudnego do zarządzania kodu.

Przykład zastosowania MVC w aplikacji internetowej:

Aplikacja do blogowania:

- **Model**: Przechowuje dane o artykułach, autorach i komentarzach w bazie danych.
- **Widok**: Wyświetla artykuły, sekcje komentarzy i formularze dla użytkowników do dodawania komentarzy.
- **Kontroler**: Obsługuje żądania użytkowników, np. dodawanie nowych artykułów, dodawanie komentarzy, edycję artykułów oraz przekazuje dane między Modelem a Widokiem.

Podsumowanie:
Architektura MVC umożliwia łatwiejsze zarządzanie złożonymi aplikacjami internetowymi poprzez podział kodu na trzy niezależne komponenty: Model (zarządza danymi), Widok (prezentuje dane) i Kontroler (steruje przepływem danych). Taka separacja ułatwia rozwój, testowanie, utrzymanie i skalowanie aplikacji, co czyni MVC popularnym wzorcem w nowoczesnym projektowaniu aplikacji internetowych.

71) Omów model komunikacji asynchronicznej dla aplikacji internetowych oraz wykorzystanie podejścia REST wg. modelu Richardsona.

Model komunikacji asynchronicznej dla aplikacji internetowych
Komunikacja asynchroniczna w aplikacjach internetowych to metoda wymiany danych, w której klient (np. przeglądarka) może wysłać żądanie do serwera, nie czekając na natychmiastową odpowiedź. W odróżnieniu od komunikacji synchronicznej, gdzie klient musi czekać na odpowiedź przed kontynuowaniem innych zadań, komunikacja asynchroniczna pozwala na wykonywanie wielu operacji równolegle, co zwiększa efektywność i responsywność aplikacji.
Jak działa komunikacja asynchroniczna?

1. **Klient wysyła żądanie do serwera** (np. za pomocą technologii takich jak AJAX, Fetch API, WebSockets).
2. **Żądanie jest obsługiwane przez serwer**, który przetwarza je i przygotowuje odpowiedź.
3. **Klient nie musi czekać na odpowiedź**, może kontynuować inne operacje (np. interakcje użytkownika, renderowanie widoku).
4. **Serwer wysyła odpowiedź do klienta**, gdy jest gotowa, a klient przetwarza ją asynchronicznie, wyświetlając wyniki lub wykonując odpowiednie akcje.

Przykłady zastosowania:

1. **AJAX (Asynchronous JavaScript and XML)**:
    - Najczęściej używane do aktualizacji części strony internetowej bez konieczności jej pełnego przeładowania. Na przykład, formularz może wysłać dane do serwera, a odpowiedź z aktualizacją stanu może pojawić się na stronie natychmiast, bez przeładowania strony.
2. **WebSockets**:
    - WebSockets umożliwiają dwukierunkową, pełnodupleksową komunikację w czasie rzeczywistym między klientem a serwerem. To podejście jest idealne do aplikacji wymagających natychmiastowych aktualizacji, takich jak czaty, gry online czy systemy giełdowe.
3. **Promise / Async/Await**:
    - Współczesne języki programowania (np. JavaScript) wspierają mechanizmy takie jak Promise i async/await, które pozwalają na prostą obsługę asynchronicznych operacji, takich jak komunikacja z serwerem.

Zalety komunikacji asynchronicznej:

- **Poprawiona responsywność**: Aplikacje mogą reagować na interakcje użytkownika, nawet podczas oczekiwania na odpowiedzi z serwera.
- **Zwiększona wydajność**: Asynchroniczna obsługa wielu żądań równocześnie zmniejsza czas oczekiwania i obciążenie serwera.
- **Lepsza obsługa długotrwałych operacji**: Serwer może obsługiwać skomplikowane zapytania (np. przetwarzanie dużych zbiorów danych), a odpowiedź zostanie zwrócona dopiero, gdy proces zostanie ukończony, bez blokowania aplikacji.

Podejście REST według modelu Richardsona
REST (Representational State Transfer) to styl architektoniczny oparty na zasadach prostych, zasobo-centrycznych interakcji w sieci, głównie z użyciem protokołu HTTP. **Model dojrzałości Richarda**

Richardsona to klasyfikacja, która opisuje stopnie zgodności interfejsu API z zasadami REST. Jest to czteroetapowy model oceny stopnia przestrzegania zasad REST w interfejsach API.

Poziomy dojrzałości REST według modelu Richardsona:

1. **Poziom 0: Usługi z jednym punktem końcowym (endpoint)**
    - Na tym poziomie aplikacja API ma jeden główny punkt końcowy (np. /api), który obsługuje wszystkie żądania. API nie korzysta z zasobów ani metod HTTP (GET, POST, PUT, DELETE). W takim podejściu struktura API przypomina tradycyjne usługi SOAP.
    - **Przykład**: Wszystkie żądania (np. pobieranie danych, wysyłanie danych) trafiają na ten sam URL /api, a typ operacji jest określany np. w treści żądania.
2. **Poziom 1: Zasoby (Resources)**
    - Na tym poziomie API zaczyna wprowadzać **zasoby**. Każdy rodzaj danych lub obiekt (np. użytkownik, produkt) jest reprezentowany przez osobne URI. Każdy zasób ma swój unikalny adres, np. /users dla użytkowników i /products dla produktów.
    - **Przykład**: API ma oddzielne ścieżki dla zasobów:
        - GET /users – pobiera listę użytkowników.
        - GET /users/123 – pobiera szczegóły użytkownika o ID 123.
3. **Poziom 2: Metody HTTP (Verbs)**
    - Na tym poziomie API wykorzystuje różne **metody HTTP** do operacji na zasobach. Każda operacja (czyli metoda HTTP) ma określone znaczenie:
        - **GET** – pobieranie danych.
        - **POST** – tworzenie nowego zasobu.
        - **PUT** – aktualizacja istniejącego zasobu.
        - **DELETE** – usunięcie zasobu.
    - **Przykład**: API używa odpowiednich metod HTTP:
        - POST /users – tworzy nowego użytkownika.
        - PUT /users/123 – aktualizuje użytkownika o ID 123.
        - DELETE /users/123 – usuwa użytkownika o ID 123.
4. **Poziom 3: HATEOAS (Hypermedia As The Engine Of Application State)**
    - Na najwyższym poziomie dojrzałości API zgodne z REST udostępnia **hipermedia** jako sposób nawigacji po dostępnych zasobach. Klient API otrzymuje nie tylko dane, ale także linki do innych zasobów, co pozwala na dynamiczne eksplorowanie API bez potrzeby wcześniejszej znajomości pełnej struktury.
    - **Przykład**: API zwraca dane wraz z linkami do innych dostępnych operacji:

json
Skopiuj kod

```
{
  "id": 123,
  "name": "John",
  "links": {
    "self": "/users/123",
    "delete": "/users/123",
    "update": "/users/123"
  }
}
```

Zalety modelu REST według Richardsona:

1. **Lepsza struktura i organizacja API**: Każdy poziom modelu Richardsona wprowadza bardziej ustrukturyzowany i logiczny sposób na organizowanie interakcji w API, co ułatwia korzystanie i rozumienie interfejsu.
2. **Użycie standardów HTTP**: Wykorzystanie metod HTTP w sposób spójny (np. GET do pobierania, POST do tworzenia) sprawia, że API jest intuicyjne i łatwe w implementacji.
3. **Odpowiednie rozdzielenie zasobów i operacji**: Oddzielając zasoby od operacji, API staje się bardziej czytelne i łatwiejsze w utrzymaniu.

4. **HATEOAS na poziomie 3**: Umożliwia dynamiczne nawigowanie po API i minimalizuje zależność od dokumentacji.

Podsumowanie:

Komunikacja asynchroniczna w aplikacjach internetowych pozwala na efektywne zarządzanie żądaniami między klientem a serwerem, zwiększając wydajność i responsywność aplikacji. **Model dojrzałości Richarda Richardsona** dla REST oferuje stopniowy sposób oceny zgodności API z zasadami REST, od podstawowej struktury zasobów, aż po pełną dynamikę obsługi linków za pomocą HATEOAS, co ułatwia tworzenie wydajnych i skalowalnych interfejsów API.

4o

72) Omów model tworzenia aplikacji internetowych w oparciu o architekturę rozproszonych mikroserwisów SOA.

Model tworzenia aplikacji internetowych w oparciu o architekturę rozproszonych mikroserwisów SOA

Architektura rozproszonych mikroserwisów oraz **SOA (Service-Oriented Architecture)** to popularne podejścia do budowy nowoczesnych aplikacji internetowych, które kładą nacisk na modułowość, skalowalność i łatwość integracji różnych usług. Obie te architektury mają na celu rozbicie monolitycznych aplikacji na mniejsze, autonomiczne komponenty, które mogą być rozwijane, wdrażane i skalowane niezależnie od siebie.

SOA (Service-Oriented Architecture) – Wprowadzenie

SOA to architektura, w której aplikacja składa się z **luźno powiązanych usług**, które komunikują się między sobą poprzez standardowe interfejsy (np. protokół HTTP, SOAP, REST). Każda usługa realizuje konkretne funkcje biznesowe i może być wielokrotnie wykorzystywana przez różne aplikacje. SOA kładzie nacisk na interoperacyjność między różnymi systemami i językami programowania.

Główne cechy SOA:

1. **Usługi jako samodzielne komponenty**:
   - Każda usługa w SOA jest niezależnym komponentem, który realizuje określoną funkcjonalność (np. usługa zarządzania użytkownikami, usługa przetwarzania płatności).
2. **Standardowe protokoły**:
   - SOA opiera się na standardowych protokołach komunikacyjnych (np. HTTP, SOAP, REST) i formatach danych (XML, JSON), co umożliwia integrację między różnymi systemami.
3. **Interoperacyjność**:
   - Usługi w SOA mogą być napisane w różnych językach programowania i działać na różnych platformach, co pozwala na ich łatwą integrację w różnych środowiskach.
4. **Zarządzanie procesami biznesowymi**:
   - SOA pozwala na łączenie usług w złożone procesy biznesowe, które mogą być łatwo zarządzane, monitorowane i modyfikowane.

Mikroserwisy – Wprowadzenie

Mikroserwisy to bardziej zwinne i skalowalne podejście do SOA, w którym aplikacja internetowa jest podzielona na **bardzo małe, niezależne serwisy**, które komunikują się ze sobą za pomocą lekkich protokołów (np. HTTP/REST). W odróżnieniu od SOA, gdzie nacisk kładziony jest na interoperacyjność między różnymi systemami, mikroserwisy są projektowane w celu umożliwienia łatwego skalowania, wdrażania i konserwacji poszczególnych komponentów.

Główne cechy mikroserwisów:

1. **Autonomiczne serwisy**:
   - Każdy mikroserwis jest autonomicznym modułem, który realizuje jedną, dobrze zdefiniowaną funkcję (np. obsługa koszyka, przetwarzanie zamówień, autoryzacja użytkowników). Mikroserwisy mają własne bazy danych i mogą być niezależnie wdrażane.
2. **Niezależne skalowanie i wdrażanie**:
   - Mikroserwisy mogą być wdrażane i skalowane niezależnie od siebie, co zwiększa elastyczność i ułatwia zarządzanie aplikacją. W przeciwieństwie do monolitów, w których każdy element aplikacji jest ściśle powiązany, mikroserwisy mogą być wdrażane w różnych wersjach bez wpływu na inne komponenty.

3. **Luźne powiązanie**:
    - Mikroserwisy komunikują się przez lekkie protokoły (np. HTTP, gRPC, AMQP) i są ze sobą luźno powiązane. Oznacza to, że mogą działać niezależnie od siebie, a problemy z jednym mikroserwisem nie wpływają bezpośrednio na działanie innych.
4. **Podział według domeny biznesowej**:
    - Mikroserwisy są projektowane zgodnie z podziałem na konkretne domeny biznesowe. Zamiast tworzyć jedną dużą aplikację, organizacja może podzielić system na mniejsze komponenty odpowiadające konkretnym funkcjom biznesowym.
5. **Niezależność technologiczna**:
    - Mikroserwisy mogą być rozwijane w różnych technologiach (np. jeden mikroserwis w Pythonie, inny w Javie), co daje dużą swobodę w doborze narzędzi.

Różnice między SOA a mikroserwisami:

1. **Skala i rozmiar usług**:
    - W SOA usługi mogą być większe i realizować bardziej złożone funkcje, podczas gdy mikroserwisy mają tendencję do bycia bardzo małymi i skoncentrowanymi na jednym zadaniu.
2. **Niezależność**:
    - Mikroserwisy są w pełni niezależne, posiadają własne bazy danych i mogą być wdrażane autonomicznie, podczas gdy usługi w SOA mogą dzielić pewne zasoby.
3. **Komunikacja**:
    - W SOA częściej stosuje się bardziej rozbudowane protokoły komunikacji, takie jak SOAP, podczas gdy mikroserwisy preferują lżejsze metody, takie jak REST, gRPC lub komunikacja asynchroniczna za pomocą kolejek.

Model tworzenia aplikacji w oparciu o mikroserwisy SOA

1. Projektowanie domenowe:

- Pierwszym krokiem w tworzeniu aplikacji opartej na mikroserwisach jest **zidentyfikowanie domen biznesowych**. Każda domena biznesowa (np. użytkownicy, zamówienia, produkty) powinna być obsługiwana przez odrębny mikroserwis.

2. Podział na mikroserwisy:

- Domena biznesowa zostaje rozbita na **mikroserwisy**, z których każdy realizuje jedno konkretne zadanie. Na przykład:
    - Serwis autoryzacji użytkowników.
    - Serwis zarządzania produktami.
    - Serwis przetwarzania płatności.

3. Baza danych dla każdego mikroserwisu:

- Każdy mikroserwis ma swoją własną **bazę danych**. To kluczowy aspekt mikroserwisów – zamiast dzielić jedną dużą bazę danych, każdy serwis zarządza swoim fragmentem danych, co minimalizuje zależności.

4. Komunikacja między mikroserwisami:

- Mikroserwisy komunikują się za pomocą **lekkich protokołów**, takich jak:
    - **HTTP/REST** – do synchronicznej komunikacji.
    - **gRPC** – do szybkiej i efektywnej komunikacji między serwisami.
    - **Message Brokers (np. RabbitMQ, Kafka)** – do asynchronicznej wymiany wiadomości między serwisami, co zapewnia odporność na awarie.

5. Orkiestracja i rejestracja usług:

- W przypadku dużej liczby mikroserwisów niezbędne jest wprowadzenie **mechanizmów orkiestracji** i **rejestracji usług**. Narzędzia takie jak **Kubernetes** czy **Docker Swarm** pomagają zarządzać wdrażaniem, skalowaniem i monitorowaniem mikroserwisów.

6. Bezpieczeństwo i autoryzacja:

- W architekturze mikroserwisów każdy serwis może wymagać osobnych mechanizmów **autoryzacji**. Stosuje się tutaj standardy takie jak **OAuth 2.0**, które zapewniają bezpieczne zarządzanie sesjami użytkowników i dostępem do różnych usług.

7. Testowanie i monitorowanie:

- W związku z niezależnością mikroserwisów, testowanie wymaga podejścia specyficznego dla każdego serwisu. Popularnym podejściem jest **testowanie kontraktów**, aby upewnić się, że mikroserwisy komunikują się poprawnie.
- **Monitorowanie** mikroserwisów odbywa się za pomocą narzędzi takich jak **Prometheus**, **ELK Stack** (Elasticsearch, Logstash, Kibana) czy **Grafana**, które pozwalają na analizowanie logów, śledzenie wydajności i diagnozowanie problemów.

Zalety architektury mikroserwisów w modelu SOA:

1. **Skalowalność**: Możliwość niezależnego skalowania poszczególnych mikroserwisów pozwala na lepsze wykorzystanie zasobów i dopasowanie do obciążeń.
2. **Elastyczność technologiczna**: Mikroserwisy mogą być rozwijane w różnych technologiach, co daje zespołom większą swobodę w doborze narzędzi.
3. **Szybszy rozwój**