

1. Omówić struktury sieci komputerowych. Proszę wymienić i krótko scharakteryzować poznane struktury sieci komputerowych, oraz porównać je ze sobą pod względem: bezpieczeństwa, niezawodności i efektywności. (A-01)

Poznane struktury sieci komputerowych:

- sieć klient-serwer,
- sieć równorzędna typu peer-to-peer,
- sieć LAN,
- sieć WAN,
- sieć CAN,
- sieć WAN.

Sieć klient-serwer

Architektura systemu komputerowego, w szczególności oprogramowania, umożliwiająca podział zadań. Polega to na ustaleniu, że serwer zapewnia usługi dla klientów, zgłaszających do serwera żądania obsługi. Najczęściej spotykanymi, podstawowymi serwerami, które działają w oparciu o architekturę klient-serwer są: serwer WWW, serwer poczty elektronicznej, serwer plików oraz serwer aplikacji. Na ogół z usług jednego serwera korzysta wielu klientów jednocześnie. Ogólnie jeden klient może w tym samym czasie korzystać z usług wielu serwerów.

Sieć peer-to-peer

To sieć, która służy do dzielenia się plikami pomiędzy użytkownikiem internetu przy użyciu odpowiedniej aplikacji. Wszystkie urządzenia są równe w hierarchii. Przy takiej architekturze nie istnieje centralny administrator zarządzający pozostałymi komputerami. Każde urządzenia działa jako host. Może więc w procesie wymiany informacji jednocześnie pełnić funkcję klienta pobierającego zasoby cyfrowe oraz funkcję serwera, który je udostępnia.

Sieć LAN(Local Area Network)

Lokalna sieć komputerowa, która zwykle łączy co najmniej dwa komputery i urządzenia peryferyjne znajdujące się na niewielkim obszarze, najczęściej w jednym budynku, np. bloku, biurze, zakładzie produkcyjnym, laboratorium, bibliotece czy uczelni. Urządzenia tworzące sieć LAN powinny mieć dostęp do protokołu TCP/IP oraz być połączone kablem Ethernet. Taka sieć cechuje się sporą stabilnością połączenia i wysokim transferem.

Sieć WAN(Wide Area Network)

Typem sieci komputerowej o największym zasięgu jest WAN. Umożliwia transfer danych pomiędzy komputerami oddalonymi nawet o tysiące kilometrów, w tym znajdującymi się na różnych kontynentach. Przykładem takiej sieci jest Internet. Połączenie można nawiązać z użyciem takich protokołów jak m.in. ADSL, DSL, ISDN, ATM, HDLC, SMDS czy PPP. Korzystanie z międzynarodowej sieci WAN wymaga wykupienia usługi u operatora.

Sieć CAN(Campus Area Network)

Pośrednia sieć pomiędzy MAN a LAN. Zwykle jest zakładana w ośrodkach akademickich lub małych firmach. Zasięg sieci obejmuje co najwyżej kilka budynków, które nie są od siebie zbyt oddalone. Maksymalna prędkość transmisji wynosi 1 Mb/s i jest możliwa do osiągnięcia na dystansie nieprzekraczającym 40 metrów. Sieć CAN wyróżnia się dużą stabilnością oraz odpornością na zakłócenia sygnału.

Sieć PAN(Personal Area Network)

Osobista, prywatna sieć komputerowa o najmniejszym zasięgu. Umożliwia wymianę danych pomiędzy różnymi urządzeniami – komputerami, tabletami, smartfonami, drukarkami itp., należącymi do określonej osoby bądź gospodarstwa domowego. Mogą one łączyć się ze sobą niezależnie lub z wykorzystaniem sieci wyższego poziomu. Sieć PAN może działać w sposób bezprzewodowy bądź przewodowy, np. z użyciem kabli USB. Transfer danych jest możliwy na bliskich odległościach, zwykle do kilku metrów. Technologie używane w tej sieci to np. Bluetooth, Wireless USB czy IrDA.

Porównanie wymienionych wyżej struktur sieci komputera:

Tabela 1 Tabela porównująca efektywność różnych struktur sieciowych

	Klient-serwer	P2P	LAN	WAN	CAN	PAN
E*	Jeden lub kilka komputerów udostępniające usługi oraz wiele komputerów korzystających	Wiele komputerów pełniących funkcję serwera i komputera korzystającego	Zasięg lokalny, na kondygnacji, ewentualnie kilka budynków	Zasięg rzędu setek lub tysięcy kilometrów	Zasięg kilku sieci lokalnych	Zasięg osobisty, kilka metrów

*E – efektywność

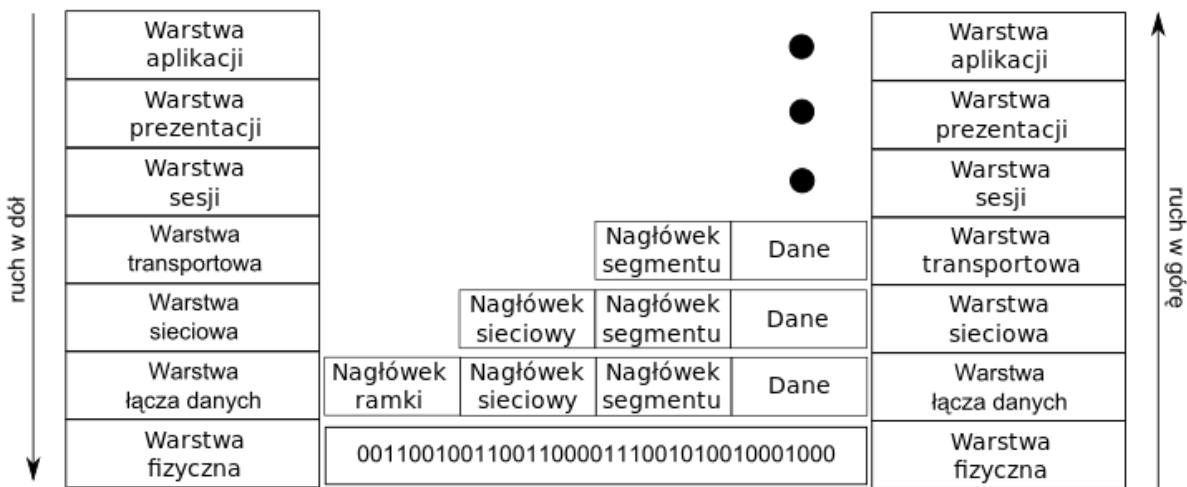
A2. Omówić model ISO/OSI. Proszę wymienić i krótko scharakteryzować warstwy modelu ISO/OSI, oraz przedstawić jakie urządzenia sieciowe i usługi, pracują w poszczególnych warstwach.

Wzajemna komunikacja urządzeń w sieci komputerowej składa się z kilku etapów, z kilku elementów. Każdy z nich jest tak samo ważny, ponieważ na każdym z nich realizowane są zadania niezbędne do poprawnej komunikacji. Etapy te określone są przez tak zwane modele warstwowe.

Istnieją dwa warstwowe modele, są nimi model protokołów TCP/IP oraz model odniesienia ISO/OSI. Z jednej strony są do siebie podobne, z drugiej jednak każdy z modeli charakteryzuje się nieco innym podejściem do sposobu komunikacji. Najlepszym przykładem na zobrazowanie tego są systemy operacyjne. Część użytkowników korzysta z systemu Windows, część z jednej z dystrybucji Linuxa, a jeszcze inni z MacOS. Każdy z tych systemów jest inny, każdy w inny sposób realizuje zadania sieciowe, ale w efekcie końcowym na każdym z tych systemów strona WWW czy też e-mail będą wyglądały tak samo, a przynajmniej podobnie. Tak więc do najważniejszych korzyści wynikających z zastosowania modeli warstwowych zaliczymy:

- ✓ łatwiejsze określenie reguł i zasad komunikacji (te reguły i zasady to protokoły komunikacyjne),
- ✓ możliwość współdziałania ze sobą urządzeń sieciowych i oprogramowania różnych producentów,
- ✓ umożliwienie zastosowania jednego rozwiązania innym składnikom sieci bez konieczności zmian w innych warstwach(np. wymieniając kartę sieciową, nie musimy zmieniać kabli Ethernet)
- ✓ wprowadza niezależność poszczególnych rodzajów danych, które są wykorzystywane w sieciach(część struktury sieci oparte na światłowodach, a reszta na Wi-Fi i wzajemnie się uzupełniają)
- ✓ możliwość łatwiejszego zrozumienia całego procesu komunikacji,
- ✓ możliwość zarządzania procesem komunikacji.

Model OSI opisuje drogę danych od aplikacji w systemie jednej stacji roboczej do aplikacji w systemie drugiej. Przed wysłaniem dane wraz z przekazywaniem do niższych warstw sieci zmieniają swój format, co nosi nazwę procesu kapsułkowania (enkapsulacji).



Rysunek 0-1. Enkapsulacja danych

Na Rysunku 1. można zauważyc jak wraz z przenoszeniem kombinacji składającej się z danych i nagłówka warstwy poprzedniej w dół stacji wysyłającej (lewa strona) ulega ona kapsułkowaniu pod nagłówkiem warstwy kolejnej.

- W warstwie transportu dane obejmują właściwe dane oraz nagłówek segmentu.
- Natomiast w warstwie sieciowej dane oprócz właściwych danych i nagłówka segmentu dodatkowo wzbogacone są o nagłówek sieciowy, który zawiera adresy logiczne: źródłowy i docelowy. Adresy te pozwalają wyznaczyć drogę tych pakietów między dwiema stacjami, które pracują w odległych sieciach.
- W warstwie łącza danych pakiet z poprzedniej warstwy wzbogacony jest dodatkowo o nagłówek ramki, który określa sposób przekazania danych przez interfejs sieciowy do sieci fizycznej.
- Ostatnia warstwa – fizyczna – ramka z poprzedniej warstwy przekształcana jest do postaci pozwalającej przesyłać informację przewodem sieciowym lub za pomocą innego nośnika.

Dane wędrują do stacji docelowej i tam są ponownie przekształcane, najpierw z bitów na nagłówek ramki oraz pozostałe dane. Kiedy dane wędrują do wyższych warstw, to właśnie nagłówki są wykorzystywane do określenia w jaki sposób dane mają zostać przekazane wyższym warstwom. W związku z tym, po dotarciu danych do wyższej warstwy nagłówek warstwy poprzedniej jest zdejmowany.



Rysunek 0-2. Model ISO/OSI

Model OSI	Model TCP/IP	PROTOKÓŁ / urządzenie sieciowe
Warstwa aplikacji	Warstwa aplikacji	HTTP, TELNET SSH, FTP POP3, SMTP IMAP, DNS
Warstwa prezentacji		
Warstwa sesji		
Warstwa transportowa	Warstwa transportowa	TCP, UDP
Warstwa sieci	Internet	IP, RIP OSPF, EIGRP /Router, Switch
Warstwa łącza danych	Warstwa dostępu do sieci	ETHERNET / Switch, Most, Hub
Warstwa fizyczna		Media transmisyjne (skrętka, światłowód), kodowanie

Rysunek 3. Zestawienie warstw wraz z działającymi w nich protokołami oraz urządzeniami

1. Warstwy wyższe

Wyróżniamy trzy warstwy górne, czyli warstwę aplikacji, prezentacji i sesji. Ich zadaniem jest współpraca z oprogramowaniem realizującym zadania zlecone przez użytkownika systemu komputerowego. Tworzą one pewien interfejs, który pozwala na komunikację z warstwami niższymi. Ta sama warstwa realizuje dokładnie odwrotne zadanie w zależności od kierunku przepływu danych. Przyjmijmy, że dane przepływają w dół Modelu OSI, kiedy płyną od użytkownika do urządzeń sieciowych oraz w górę w przeciwnym wypadku.

7. Warstwa aplikacji

Warstwa aplikacji jest warstwą najwyższą, zajmuje się specyfikacją interfejsu, który wykorzystują aplikacje do przesyłania danych do sieci (poprzez kolejne warstwy modelu ISO/OSI). Warstwa aplikacji udostępnia użytkownikom możliwość korzystania z usług sieciowych, takich jak WWW, poczta elektroniczna, wymiana plików, połączenia terminalowe czy komunikatory, udostępnianie zasobów (plików, drukarek). Na tym poziomie rezydują procesy sieciowe dostępne bezpośrednio dla użytkownika.

6. Warstwa prezentacji

Podczas ruchu w dół zadaniem warstwy prezentacji jest przetworzenie danych od aplikacji do postaci kanonicznej zgodnej ze specyfikacją OSI-RM, dzięki czemu niższe warstwy zawsze otrzymują dane w tym samym formacie. Kiedy informacje płyną w góre, warstwa prezentacji tłumaczy format otrzymywanych danych na zgodny z wewnętrzną reprezentacją systemu docelowego. Wynika to ze zróżnicowania systemów komputerowych, które mogą w różny sposób interpretować te same dane. Dla przykładu bity w bajcie danych w niektórych procesorach są interpretowane w odwrotnej kolejności niż w innych. Warstwa ta odpowiada za kodowanie i konwersję danych oraz za kompresję / dekompresję; szyfrowanie/deszyfrowanie. Warstwa prezentacji obsługuje np. MPEG, JPG, GIF itp.

5. Warstwa sesji

Zapewnia aplikacjom na odległych komputerach realizację wymiany danych pomiędzy nimi. Kontroluje nawiązywanie i zrywanie połączenia przez aplikację. Jest odpowiedzialna za poprawną realizację zapytania o daną usługę. Do warstwy tej można zaliczyć funkcje API udostępniane programiście przez bibliotekę realizującą dostęp do sieci na poziomie powyżej warstwy transportowej takie jak np. biblioteka strumieni i gniazdek BSD.

2. Warstwy niższe

Najniższe warstwy zajmują się odnajdywaniem odpowiedniej drogi do celu, gdzie ma być przekazana konkretna informacja. Dzielą również dane na odpowiednie dla urządzeń sieciowych pakiety określane często skrótem PDU (ang. Protocol Data Unit). Dodatkowo zapewniają weryfikację bezbłędności przesyłanych danych. Ważną cechą warstw dolnych jest całkowite ignorowanie sensu przesyłanych danych. Dla warstw dolnych nie istnieją aplikacje, tylko pakiety/ramki danych.

4. Warstwa transportowa

Warstwa transportowa segmentuje dane oraz składa je w tzw. strumień. Warstwa ta zapewnia całościowe połączenie między stacjami: źródłową oraz docelową, które obejmuje całą drogę transmisji. Następuje tutaj podział danych na części, które są kolejno indeksowane i wysyłane do docelowej stacji. Na poziomie tej warstwy do transmisji danych wykorzystuje się dwa protokoły TCP (ang. *Transmission Control Protocol*) oraz UDP (ang. *User Datagram Protocol*). W przypadku gdy do transmisji danych wykorzystany jest protokół TCP stacja docelowa po odebraniu segmentu wysyła potwierdzenie odbioru. W wyniku niedotarcia któregoś z segmentów stacja docelowa ma prawo zlecić ponowną jego wysyłkę (kontrola błędów transportu). W przeciwieństwie do protokołu TCP w protokole UDP nie stosuje się potwierdzeń. Protokół UDP z racji konieczności transmisji mniejszej ilości danych zazwyczaj

jest szybszy od protokołu TCP, jednakże nie gwarantuje dostarczenia pakietu. Oba protokoły warstwy transportowej stosują kontrolę integralności pakietów, a pakiety zawierające błędy są odrzucane.

3. Warstwa sieciowa

Zapewnia metody ustanawiania, utrzymywania i rozłączania połączenia sieciowego. Obsługuje błędy komunikacji. Ponadto jest odpowiedzialna za trasowanie pakietów w sieci, czyli wyznaczenie optymalnej trasy dla połączenia. W niektórych warunkach dopuszczalne jest gubienie pakietów przez tę warstwę. W skład jej obiektów wchodzą min.: rutery

2. Warstwa łącza danych

Zapewnia niezawodność łącza danych. Definiuje mechanizmy kontroli błędów w przesyłanych ramkach lub pakietach - CRC (Cyclic Redundancy Check). Jest ona ściśle powiązana z warstwą fizyczną, która narzuca topologię. Warstwa ta często zajmuje się również kompresją danych. W skład jej obiektów wchodzą sterowniki urządzeń sieciowych, np.: sterowniki kart sieciowych oraz mosty i przełączniki.

1. Warstwa fizyczna

Zapewnia transmisję danych pomiędzy węzłami sieci. Definiuje interfejsy sieciowe i medium transmisji. Określa m.in. sposób połączenia mechanicznego, elektrycznego, standard fizycznej transmisji danych. W skład jej obiektów wchodzą min.: przewody, karty sieciowe, modemy, wzmacniacze, koncentratory.

(A-03) Omówić adresowanie sieci. Proszę omówić na czym polega adresowane sieci komputerowej oraz przedstawić zasadnicze różnice pomiędzy protokołami: IPv4 i IPv6.

Skrót:

Adres- Jest to wyrażenie pozwalające na lokalizację danego elementu w wybranym zbiorze, w przypadku sieci komputerowych takimi elementami są: same sieci, urządzenia sieciowe takie jak routery, przełączniki oraz najliczniejsza grupa urządzenia hostów np. komputery osobiste.

Adres fizyczny (Adres MAC):

- Zawarty w drugiej warstwie modelu ISO/OSI,
- 48-bitowa liczba zapisywana szesnastkowo,
- 08-BE-AC-1F-70-58 Przykładowy adres.

Adres logiczny (Adres IP)

- Zawarty w trzeciej warstwie modelu ISO/OSI, dla modelu TCP/IP warstwa druga,

-IPv4:

- 32-bitowa liczba zapisana jako 4 oddzielne oktety z wartościami z przedziału 0-255,
- maska podsieci wyodrębnia adres podsieci oraz adres hosta w tej podsieci,
- klasy adresów publicznych A-E

-IPv6:

-128-bitowa liczba, każde 16 bitowe pole jest zapisane szesnastkowo i oddzielone dwukropkiem.

-rodzaje adresów Unicast, Multicast, Anycast

-zasięg określa jakiej części sieci ten adres dotyczy,

Różnice:

Różnica	IPv4	IPv6
Długość adresu	32 bity	128 bitów
Sposób zapisu	4 oktety z wartościami z zakresu 0-255 192.168.1.15	Pełen zapis składa się z 8 16-bitowych pól zapisanych szesnastkowo. 2001:0db8:0000:0001:0000:0000:0001 Adresy te można także uproszczyć w zapisie 2001:db8:0:1::1
Maska adresu	Używana do oddzielenia części sieciowej od części hosta.	Nie używana, stosowane są długości prefiksów
Zasięg adresu	Pojęcie nie ma zastosowania, dla adresów pojedynczych.	Adresy pojedyncze mają zdefiniowane dwa zasięgi, w tym segmentowy i globalny; adresy grupowe mają 14 zasięgów.
Czas życia adresu	Stosowany tylko w przypadku przydzielonych protokołem DHCP	Podział na dwa czasy życia: preferowany i poprawny. Preferowany <= poprawnemu.

Adres nieokreślony	Niezdefiniowany	Zdefiniowany jako ::/128
Liczba dostępnych adresów	$2^{32} = 4\ 294\ 967\ 296$	$2^{128} = 3.4 \times 10^{38}$
Sposób konfiguracji	Ręczny lub DHCP	Samokonfiguracja
Wykorzystanie IPsec	Nieobowiązkowa implementacja	Obowiązkowa implementacja

(A-03).1 Adresowanie

Adresowanie sieci- w celu umożliwienia komunikacji pomiędzy wieloma urządzeniami w sieci, każde takie urządzenie potrzebuje unikatowego adresu, który pomoże w zidentyfikowaniu urządzenia w sieci.

(A-03).2. Adres fizyczny (Adres MAC)

Adresowanie fizyczne- ma miejsce na drugiej warstwie modelu ISO/OSI (łącza danych), często jest też nazywane adresowaniem sprzętowym ponieważ adres ten zapisany jest w pamięci ROM karty sieciowej. Adres fizyczny nazywany również MAC składa się to 48-bitowy numer zakodowany w systemie szesnastkowym.

(A-03).3. Adres logiczny (Adres IP)

Adres logiczny- występuje w trzeciej warstwie modelu odniesienia ISO/ OSI, czyli w warstwie sieciowej. Każdy komputer w sieci Internet ma unikatowy adres IP, którego przydział jest administrowany przez odpowiednie organizacje (IANA, ICANN). Adres IP jest adresem logicznym interfejsu sieciowego (karty sieciowej). Każde urządzenie pracujące w sieci (komputer, tablet, telefon komórkowy itp.) musi taki adres mieć, aby komunikować się z inni urządzeniami. Przełączniki oraz punkty dostępowe mogą mieć przypisany adres IP; z kolei routery muszą mieć przynajmniej jeden adres IP. Adres IP nazywany jest adresem logicznym, ponieważ nie jest przypisywany urządzeniu na stałe i może się zmieniać

(A-03).4 Protokół IPv4

IPv4- Zapewnia podstawowe funkcje potrzebne do przesyłania pakietów, przesyłanie odbywa się przy użyciu datagramów, protokół ten zapewnia także funkcjonalność pozwalającą na fragmentację i ponowne złożenie długich datagramów. Miejsce źródłowe i docelowe informacji jest identyfikowane przez adresy o stałej długości.

Adres w protokole IPv4 to 32-bitowa liczba zapisywana jako 4 oktety o wartościach z przedziału 0-255.

Z adresami protokołu IPv4 nierożłączne jest także pojęcie masek podsieci. Maska podsieci jest to liczba, która pozwala podzielić adres na adres podsieci oraz adres hosta, jest to 32-bitowa liczba, w której w zapisie binarnym 0 nie może poprzedzać żadnej 1.

Dla adresów IPv4 wydzielamy 5 klas adresów publicznych:

Klasa A- część sieci składa się z 8 bitów (I oktet), a część hosta z pozostałych 24 bitów (II, III, IV oktet). Pierwszy bit adresu zawsze jest ustalony na 0. Maksymalnie można zaadresować:

$$2^{24} - 2 \text{ hosty} = 16777214 \text{ hostów},$$

Klasa B- część sieci składa się z 16 bitów (I, II oktet), a część hosta z pozostałych 16 bitów (III, IV oktet). Pierwsze 2 bity adresu zawsze są ustawione na 10. Maksymalnie można zaadresować:

$$2^{16} - 2 \text{ hosty} = 65534 \text{ hostów},$$

Klasa C- część sieci składa się z 24 bitów (I, II, III oktet), a część hosta z pozostałych 8 bitów (IV oktet). Pierwsze 3 bity adresu zawsze są ustawione na 110. Maksymalnie można zaadresować:

$$2^8 - 2 \text{ hosty} = 254 \text{ hosty},$$

Klasa D- pierwsze 4 bity adresu zawsze są ustawione na 1110,

Klasa E- pierwszy 4 bity adresu zawsze są ustawione na 1111.

Adresy prywatne

Adresy prywatne, to adresy wydzielone w każdej klasie adresów IP, które nie są przydzielane hostom w Internecie. Adresy takie najczęściej wykorzystuje się do adresowania w sieciach lokalnych. Ruch kierowany do sieci prywatnej nie jest przepuszczany przez routery. Aby sieci prywatne (budowane w oparciu o adresy prywatne) mogły łączyć się z sieciami publicznymi (np. Internetem), muszą wykorzystywać translację NAT lub Proxy Server.

Zakresy adresów prywatnych IP:

- 10.0.0.0 - 10.255.255.255
- 172.16.0.0 - 172.31.255.255
- 192.168.0.0 - 192.168.255.255

(A-03).5 Protokół IPv6

IPv6- jest to następca protokołu IPv4. W porównaniu do swojego poprzednika zwiększała została liczba wykorzystywanych bitów na adres dzięki czemu można było zastosować więcej poziomów hierarchii, ilość możliwych do przypisania adresów została znacznie zwiększała.

Adres w tym protokole to 128-bitowa liczba zapisywana w postaci ośmiu 16-bitowych części oddzielonych od siebie dwukropkiem.

W adresacji wykorzystywanej w protokole IPv6 używane są trzy typy adresów:

- adresy unicast – identyfikujące pojedynczy interfejs; pakiety, które są kierowane na ten typ adresu dostarczane są tylko do odbiorcy,
- adresy multicast – identyfikujące grupę interfejsów (mogą one należeć do różnych węzłów), pakiety wysyłane na ten adres dostarczane są do interfejsów w tej grupie,
- adresy anycast – podobnie jak adresy multicast, identyfikują grupę interfejsów, jednak pakiet wysyłany na ten adres dostarczany jest tylko do najbliższego węzła (węzeł ten jest wyznaczany przez protokół routingu)

W protokole IPv6 w odróżnieniu do IPV4 nie stosujemy masek do oddzielenia adresu podsieci od adresu hosta tylko stosujemy długość prefiku zapisywane są zazwyczaj wraz z adresem po znaku „/” np. 2001:db8:abcd:ffff::/64.

Charakterystyczną cechą protokołu jest fakt, że zostały zdefiniowane zakresy adresów. W przypadku adresów unicastowych wyróżniane są następujące zakresy:

- adresy lokalne dla łącza (link-local address) – są to adresy wykorzystywane tylko do komunikacji w jednym segmencie sieci lokalnej lub przy połączeniu typu point-to-point. Routery nie przekazują pakietów z tego rodzaju adresem. Z puli pozostałych adresów wyróżniane są przez prefiks FE80::/10. Każdy interfejs musi mieć przydzielony co najmniej jeden adres lokalny dla łącza, nawet jeżeli posiada adres globalny lub unikatowy adres lokalny.
- unikatowe adresy lokalne (unique local address) – są to adresy będące odpowiednikami adresów prywatnych wykorzystywanych w protokole IPv4. Z puli pozostałych adresów wyróżniane są przez prefiks FC00::/7. Od adresów lokalnych łącza odróżnia je także prefiks routingu.
- adresy globalne (global unicast address) – widoczne w całym Internecie, są odpowiednikami adresów publicznych stosowanych w IPv4; do adresów tego typu należą adresy nie wymienione w pozostałych punktach.

Źródła:

<https://datatracker.ietf.org/doc/html/rfc4291> [Dostęp:22.05.2022]

<https://datatracker.ietf.org/doc/html/rfc791> [Dostęp:22.05.2022]

<https://datatracker.ietf.org/doc/html/rfc796> [Dostęp:22.05.2022]

<http://www.crypto-it.net/pl/teoria/protokoly-tcp-ip.html> [Dostęp:22.05.2022]

<https://pasja-informatyki.pl/sieci-komputerowe/adresowanie-w-sieci/> [Dostęp:22.05.2022]

<http://kti.eti.pg.gda.pl/ktilab/ipv6-adr/ipv6-adresy.pdf> [Dostęp:22.05.2022]

<https://www.ibm.com/docs/pl/i/7.2?topic=6-comparison-ipv4-ipv6> [Dostęp:22.05.2022]

http://www.nss.et.put.poznan.pl/study/projekty/sieci_komputerowe/tcpip/html/adresowanie/adres_ow.htm [Dostęp:22.05.2022]

<https://sieci.infopl.info/index.php/adresowanie/klasyip> [Dostęp: 22.05.2022]

<https://datatracker.ietf.org/doc/html/rfc2460> [Dostęp:22.05.2022]

Protokół DHCP

- DHCP (Dynamic Host Configuration Protocol) jest najpopularniejszym protokołem automatycznego przydzielania IPv4.
- Działa w architekturze klient-serwer
- Serwer DHCP odpowiada za przydzielanie adresów, tworzy maskę podsieci, oraz wyznacza czas jaki dany adres może być przypisany do jednego klienta
- Po podłączeniu do sieci to klient prosi serwer DHCP o przydelenie jednego z wolnych adresów
- Bardzo często rolę serwera DHCP pełni router
- Do przydzielania adresów IPv6 wykorzystywany jest DHCPv6

Protokół DHCPv4 – metody przydzielania adresów

- Ręczna alokacja – Administrator przypisuje wstępnie przydzielony adres IPv4 dla klienta, a DHCPv4 przekazuje ten adres IPv4 do urządzenia.
- Automatyczna alokacja – DHCPv4 automatycznie przypisuje statyczny adres IPv4 na stałe do urządzenia, wybierając go z puli dostępnych adresów.
- Dynamiczna alokacja – DHCPv4 - dynamicznie przydziela, lub inaczej - dzierżawi, adres IPv4 z puli dostępnych adresów, na określony przez serwer czas, lub do momentu gdy klient nie potrzebuje już tego adresu.

IPv6 – typy automatycznej konfiguracji

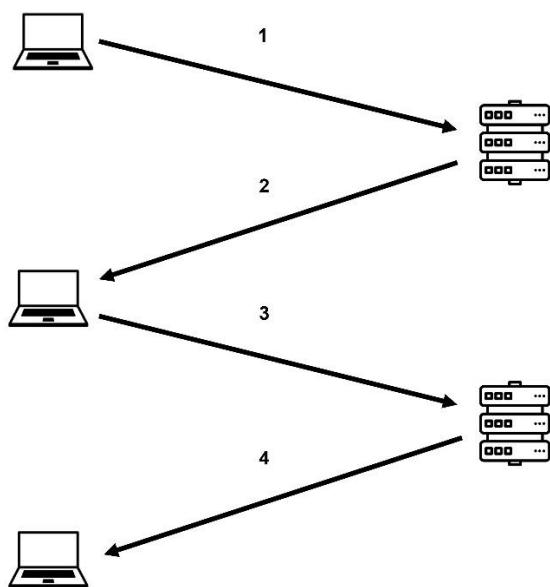
- **Stanowa automatyczna konfiguracja.** Ten typ konfiguracji wymaga pewnego poziomu interwencji człowieka, ponieważ wymaga serwera Dynamic Host Configuration Protocol dla protokołu IPv6 (DHCPv6) do instalowania węzłów i administrowania nim. Serwer DHCPv6 przechowuje listę węzłów, do których dostarcza informacje o konfiguracji. Przechowuje również informacje o stanie, dzięki czemu serwer wie, jak długo każdy adres jest w użyciu i kiedy może być dostępny do ponownego przypisania.
- **Bez stanowa automatyczna konfiguracja.** Ten typ konfiguracji jest odpowiedni dla małych organizacji i użytkowników indywidualnych. W takim przypadku każdy host określa swoje adresy na podstawie zawartości odebranych anonsów routera. Używając standardu IEEE EUI-64 do definiowania części adresu z identyfikatorem sieci, uzasadnione jest założenie unikatowości adresu hosta w linku.

Protokół DHCPv4 - komunikaty

- DHCPDISCOVER – klient DHCP wysyła ten komunikat w trybie broadcast, aby znaleźć serwer DHCP,
- DHCPOFFER – serwery DHCP odpowiadają tym komunikatem na komunikat DHCP, oferując dzierżawę adresu IP klientowi,
- DHCPREQUEST — klient akceptuje pierwszą ofertę wysłaną przez DHCPOFFER, która oferowała dzierżawę adresu IP, żąda tego adresu,
- DHCPACK — jeżeli adres, którego żąda klient może być użyty, to serwer akceptuje to żądanie wysyłając właśnie ten komunikat,
- DHCPNAK — jeżeli adres, którego żąda klient nie może być użyty, to serwer wysyła komunikat DHCPNAK, po tym klient musi rozpocząć cały proces komunikacji z serwerem od nowa,
- DHCPDECLINE — jeżeli klient określi, że oferowana mu konfiguracja parametrów jest nieprawidłowa, wysyła serwerowi komunikat DHCPDECLINE, po tym klient musi rozpocząć cały proces komunikacji z serwerem od nowa,
- DHCPRELEASE — klient wysyła ten komunikat, aby odrzucić przydzielony mu przez serwer adres IP i anulować dzierżawę adresu IP.

Protokół DHCPv4 – otrzymywanie adresu IP

1. Poszukiwanie serwera DHCP
2. Oferta DHCP
3. Żądanie DHCP
4. Potwierdzenie DHCP



Poszukiwanie serwera DHCP

Klient chcący się połączyć z serwerem wysyła do sieci lokalnej pakiety rozgłoszeniowe zaadresowane do wszystkich odbiorców. Procedura ta nosi nazwę DHCP DISCOVER – odkrywanie DHCP. Czasami routery są konfigurowane, aby przekazywały pakiety DHCP do właściwego serwera w innej podsieci. Pakiety mają adres docelowy rozgłoszeniowy 255.255.255.255 i zawierają prośbę o ostatnio używany adres IP (np. 192.168.1.100). Może ona zostać zignorowana przez serwer.

Oferta DHCP

Oferta DHCP (ang. DHCP Offer) jest składana przez serwer, który określa właściwą konfigurację klienta na podstawie sprzętowego adresu urządzenia sieciowego określonego w polu CHADDR (w sieci lokalnej to adres MAC). W polu YIADDR serwer przekazuje klientowi jego adres IP.

Żądanie DHCP

Żądanie DHCP (ang. DHCP Request) jest wysyłane przez klienta, który już rozpoznał serwer DHCP, ale chce uzyskać inne parametry konfiguracji. Może np. ponownie zażądać adresu IP 192.168.1.100. RFC 2131 ↓ wprowadza dodatkowo zapytanie typu DHCPINFORM. Klient stosuje je, gdy ma już przypisany adres IP (np. ręcznie), lecz nadal nie zna pozostałych wymaganych parametrów. W odpowiedzi serwer wysyła pakiet potwierdzenia DHCP z pustym polem YIADDR oraz nieustawionym czasem dzierżawy adresu.

Potwierdzenie DHCP

Potwierdzenie DHCP (ang. DHCP Acknowledge) jest wysyłane jako odpowiedź na żądanie. Zakłada się, że reakcją klienta na potwierdzenie będzie odpowiednie skonfigurowanie interfejsu sieciowego.

Odświeżanie DHCP

Elementem przydzielenia klientowi adresu IP przez serwer DHCP jest przyznanie dodatkowo tzw. czasu dzierżawy (lease). Określa on czas ważności ustawień. W tle pracują dwa zegary – T1 odmierza połowę czasu użytkowania, zaś T2 – 87,5% pełnego czasu użytkowania. Obie wartości można zmienić w opcjonalnych ustawieniach serwera DHCP – jeśli takie funkcje zostały zaimplementowane. Po upływie czasu T1 klient wysyła komunikat DHCPREQUEST do serwera i pyta, czy serwer może przedłużyć czas użytkowania. Stan ten określa się jako renewing status. Z reguły serwer odpowiada wiadomością DHCPACK i przydziela nowy czas użytkowania. Serwer resetuje wówczas zegary T1 i T2.

Jeżeli po upływie czasu T2 klient nie otrzyma wiadomości DHCPACK, rozpoczyna się tak zwany rebinding status. Klient musi wysłać komunikat DHCPREQUEST, żeby uzyskać przedłużenie czasu użytkowania. Serwer może odpowiedzieć na to żądanie potwierdzeniem DHCPACK. Jeżeli jednak i to żądanie pozostanie bez odpowiedzi, klient musi zażądać nowego adresu IP. Wkracza wówczas ponownie opisany na początku mechanizm, który rozsyła zapytania do wszystkich serwerów DHCP w sieci.

Protokół DHCPv6 – otrzymywanie adresu IP

- **Wymiana informacji pomiędzy klientem a serwerem DHCPv6, jest zbliżona w swym działaniu do zasad protokołu DHCP względem protokołu IPv4, istnieje jednak pomiędzy nimi pewna różnica wynikająca z np. braku adresu rozgłoszeniowego w protokole IPv6.**

- Proces wymiany danych pomiędzy klientem a serwerem DHCPv6 wygląda następująco:
 - Klient wysyła zapytanie SOLICIT (multicast address).
 - Serwer odpowiada komunikatem ADVERTISE (unicast address).
 - Klient kolekcjonuje otrzymane komunikaty ADVERTISE, zebrane od wielu serwerów DHCPv6. W celu wybrania najlepszej propozycji względem propagowanego prefiku.
 - Klient potwierdza wybranie adresu IPv6 za pomocą komunikatu REQUEST.
 - Serwer potwierdza rezerwację adresu za pomocą komunikatu CONFIRM.
- Oprócz tradycyjnej cztero-fazowej negocjacji adresu IPv6, istnieje jeszcze jedna opcjonalna, dwuetapowa negocjacja (Rapid two-way message Exchange), w której klient wraz z serwerem DHCPv6 wymieniają jedynie komunikaty SOLICIT oraz REPLY.

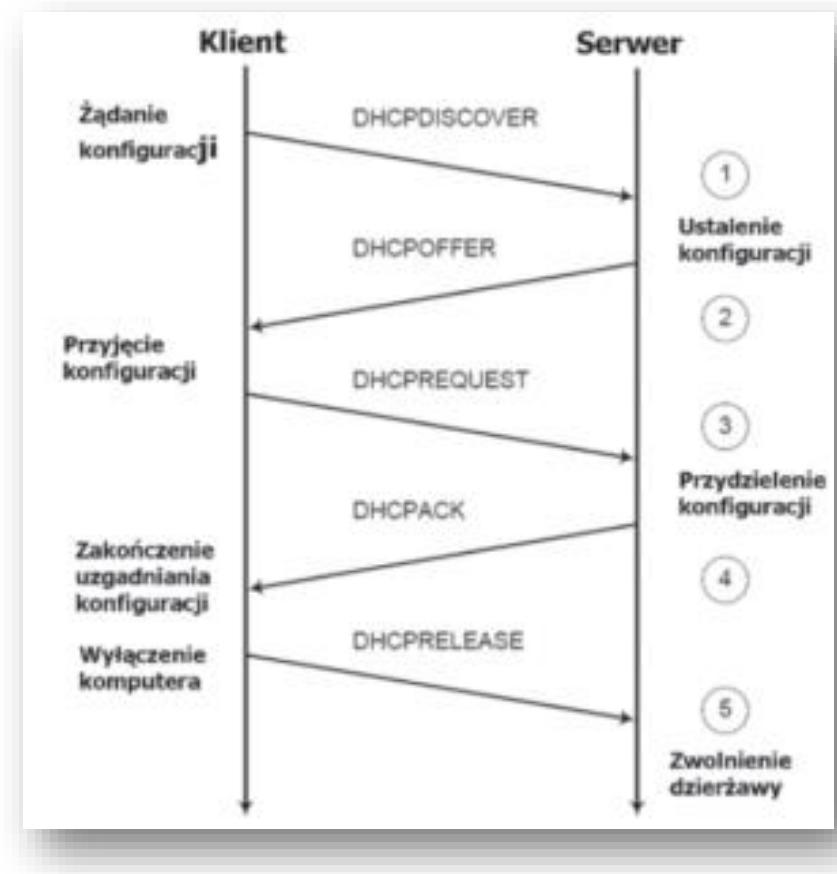
Statyczny a dynamiczny adres IP

- Statyczny adres IP to adres, który pozostaje niezmienny. Gdy urządzeniu zostanie przypisany statyczny adres IP, adres ten zwykle pozostaje niezmienny do czasu wycofania urządzenia z użytku lub zmiany architektury sieci. Statyczne adresy IP są zwykle używane przez serwery lub inne ważne urządzenia.
- Dynamiczny adres IP jest zmienny w trakcje użytkowania internetu. Może on zostać zmieniony przez serwer DHCP po upłynięciu czasu dzierżawy adresu lub czas ten może zostać przedłużony.

(A-05): Omówić serwer DHCP. Proszę wyjaśnić rolę serwera DHCP w sieci komputerowej, oraz czym się różni dynamiczne i statyczne przyznawanie adresów w sieci.

Czym jest serwer DHCP?

- DHCP jest protokołem dynamicznego konfigurowania hostów, działającym jako usługa. Umożliwia podłączonym do sieci komputerom na pobieranie z serwera takich danych jak adresu IP, maski podsieci, adresu bramy czy serwera DNS w sposób dynamiczny.
- Serwer DHCP jest urządzeniem, które za pośrednictwem protokołu komunikuje się z klientami i przydziela im odpowiednią konfigurację w sposób dynamiczny lub statyczny (zależnie od zasad zdefiniowanych przez administratora).



Rola serwera DHCP w sieci

- Serwer za sprawą wykorzystania protokołu DHCP może w sposób błyskawiczny zaktualizować parametry hostów DHCP – to doprowadzi do zmiany opcji serwera bądź zakresu, który występuje na tym serwerze. Taki zabieg umożliwi rozpoczęcie czynności w sposób pojedynczy bądź ogólnie zastosowany dla wszystkich klientów biorących udział w protokole DHCP. Protokół DHCP to możliwość wydzierżawienia adresów IP klientom na wyznaczony czas – później błyskawicznie zostaje on odnowiony jako dzierżawa IP, kiedy to klient wystąpi z prośba odmowy.

Przydzielanie statyczne vs dynamiczne

STATYCZNE	DYNAMICZNE
Statyczne przypisanie adresacji do każdego urządzenia w sposób ręczny	Automatyczne przydzielanie adresów
Każde urządzenie ma stały adres	Adres jest przydzielany automatycznie z dzierżawy i jest możliwe określenie jej czasu
Proces ręcznego przypisywania jest dłuższy	Przydzielanie adresów dzieje się automatycznie, więc jest szybsze
Do adresacji statycznej konieczna jest znajomość klas adresów, maski podsieci, bramy domyślnej i adresów DNS	Znajomość parametrów sieci nie jest konieczna
W razie awarii lub chęci wprowadzenia zmian może być konieczne sprawdzenie konfiguracji każdego urządzenia	W przypadku awarii lub konfliktu adresów może być konieczne wyeliminowanie problemów lub ręczne przydzielenie adresów statycznych spoza puli adresów DHCP

1. Omówić wirtualne sieci prywatne. Proszę wyjaśnić jak działa i do czego służy VPN, oraz omówić jak może przysłużyć się bezpieczeństwu pracy w sieci

Wirtualna sieć prywatna (ang. Virtual private network – VPN) – jest to szyfrowany tunel przez który płynie ruch w ramach sieci prywatnej pomiędzy odbiorcą a nadawcą za pośrednictwem sieci publicznej. Określenie *wirtualna* oznacza, że sieć ta istnieje jedynie jako struktura logiczna, działająca w rzeczywistości w ramach sieci publicznej.

► Wymagana infrastruktura

Do Prawidłowego działania połączenia VPN potrzebne są:

- **Urządzenie** użytkownika (telefon, komputer) – to na nim będzie się korzystać z VPN.
 - **Klient VPN** – specjalny program, który wie jak połączyć się i komunikować z serwerem VPN.
 - **Serwer VPN** – pełni rolę pośrednika pomiędzy użytkownikiem, a globalną siecią internet.
 - **Połączenie z Internetem** – żeby VPN działał trzeba oczywiście mieć aktywne połączenie z siecią.
- Urządzenie, za pomocą klienta VPN, łączy się z serwerem VPN. W tym celu serwer VPN musi autoryzować i uwierzytelnić użytkownika. Po poprawnej weryfikacji następuje utworzenie tunelu VPN, przez który będą przesyłane szyfrowane dane, niedostępne dla osób trzecich. Za szyfrowanie odpowiada protokół VPN.

Poprzez tunel VPN, dane docierają do serwera VPN na którym następuje ich odszyfrowanie. Następnie serwer VPN wysyła je w to miejsce w sieci, do którego mają dotrzeć.

Serwer VPN ma swój własny adres IP, za pomocą którego *przedstawia się* w sieci. To ten adres staje się identyfikatorem użytkownika, który łączy się za pomocą tej usługi. Zawartość tego, co jest przesyłane przez tunel VPN jest niewidoczna dla tych osób, które chcieliby podsłuchać transmisje na trasie użytkownik – serwer VPN.

Szyfrowanie w VPN

- Tunel VPN, który powstaje pomiędzy urządzeniem użytkownika, a serwerem VPN **jest zabezpieczony przed dostępem osób trzecich przez silne szyfrowanie**. Oznacza to, że dane, które nim płyną będą wyglądać dla osób postronnych jako ciąg przypadkowych znaków. Bez odpowiednich kluczy i algorytmów, nie uda się odczytać zawartości.
- Te klucze posiada tylko serwer VPN oraz aplikacja kliencka na komputerze użytkownika. Podczas aktywnego połączenia z VPN, przez cały czas i na bieżąco następuje ciągłe szyfrowanie i deszyfrowanie danych. Do szyfrowania sieci VPN zazwyczaj stosowany jest standard AES-256, czyli szyfr o długości 256 bitów.
- Protokoły VPN

Protokoły VPN odpowiadają za prawidłowy przebieg komunikacji pomiędzy klientem VPN na

urządzeniu użytkownika, a serwerem VPN. **Są zbiorem zasad, które określają w jaki sposób będą szyfrowane dane, jak przesyłane i przetwarzane.** Zostało opracowanych kilka protokołów VPN, które różnią się od siebie rozwiązaniami, szyfrowaniem i sposobem implementacji.

► Najczęściej używane protokoły VPN to:

- WireGuard
- OpenVPN
- IKEv2

- Zastosowania

Jednym z podstawowych zastosowań VPN jest całkowite szyfrowanie połączenia z siecią. Inną ważną cechą jest to, że za pomocą VPN można ukryć własny adres IP oraz lokalizację urządzenia (tzw. Anonimizacja adresu IP).

Cechą ta sprawia, że VPN to świetne zastosowanie w obszarach takich jak:

- zabezpieczenie wrażliwych i poufnych danych. Zarówno dane osobowe pracowników, jak i ustalenia poczynione między firmą a jej kontrahentami są bezpieczne – szansa, że zostaną one pozyskane przez osoby trzecie w wyniku ataków jest mała.
- możliwość ominięcia cenzury. Jest to szczególnie istotne, jeżeli firma wysyła pracowników w delegacje, np. do Chin. Tam korzystanie z Google czy mediów społecznościowych (w tym przede wszystkim Facebooka) jest mocno utrudnione.
- ochrona przed śledzeniem. Łącząc się z siecią w sposób standardowy, dostawca usług internetowych otrzymuje informacje choćby o przeglądanych stronach. Wszelką działalność swojego klienta może nie tylko śledzić, ale również rejestrować. Usługa VPN chroni przed śledzeniem ze strony choćby dostawcy Internetu.

(A-08) Omówić bazy danych. Proszę wymienić i krótko omówić podzbiory języka SQL, oraz wyjaśnić zasadę działania relacyjnych baz danych.

W skrócie

Baza danych – zorganizowane dany – najczęściej tabele 2D z kolumnami/atributami/polami i wierszami/rekordami/danymi.

Podział ze względu na model danych

- relacyjny – oddzielenie logicznej struktury danych od fizycznych struktur pamięci
- zorientowane obiektywne
- semistrukturalne
- hierarchiczne
- sieciowe
- postrelacyjne
- NoSQL

Podzbiory SQL

- DDL – CREATE, DROP, PRIMARY KEY - tworzenie i definiowanie struktur danych
- DCL – GRANT, REVOKE – uprawnienia do obiektów
- DML – INSERT, UPDATE – operacje na danych
- DQL _ SELECT – zapytania

Typy relacji: 1:1, 1:n, n:n

Joiny: inner, full, left, right

Definicje

Baza danych to zbiór danych, będący równocześnie abstrakcyjnym, informatycznym modelem wybranego fragmentu rzeczywistości. Większość typów baz danych organizuje dane w formie dwuwymiarowych tabel, gdzie kolumny to atrybuty (pola), a wiersze to rekordy (dane). Podczas projektowania bazy danych należy zapobiec redundancji danych, która poza kosztami utrzymania oraz zwiększonego czasu zapytań, może powodować niespójność danych.

Model relacyjny oznacza oddzielenie logicznej struktury danych (tabel, widoków i indeksów) od fizycznych struktur pamięci. Pozwala to na zarządzanie fizycznym przechowywaniem danych bez wpływu na strukturę logiczną. Wymaga to przechowywanie wartości danych jako typów prostych, oraz ustawienie unikalnego atrybutu na każdej tabeli.

Bazy danych - zbiór danych zapisanych zgodnie z określonymi regułami. W węższym znaczeniu obejmuje dane cyfrowe gromadzone zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizowanego do gromadzenia i przetwarzania tych danych.

Podzbiory języka SQL:

- SQL DDL - odpowiada za tworzenie i definiowanie struktur danych
 - CREATE DATABASE,
 - DROP TABLE,

- PRIMARY KEY
- SQL DCL - odpowiada za uprawnienia do obiektów bazodanowych
 - GRANT
 - REVOKE
- SQL DML - operacje na danych
 - INSERT
 - UPDATE
- SQL DQL - zapytania
 - SELECT

Podział ze względu na model danych

Typ	Opis	Schemat	Zalety	Wady
<u>Relacyjne</u>	Relacja jest zdefiniowana jako zbiór krotek, które mają takie same atrybuty. Krotka zwykle reprezentuje obiekt i związane z nim informacje. Obiekty są zwykle obiektami fizycznymi/materialnymi albo pojęciami. Relację przedstawia się zwykle jako tabelę, która jest zorganizowana w wiersze i kolumny. Wszystkie dane związane z konkretnym atrybutem mają tą samą dziedzinę i spełniają te same ograniczenia.		- Łatwość użycia - Elastyczność - Bezpieczeństwo	- Wydajność - Niezgodność pomiędzy językiem manipulacji danymi a językami programowania
Zorientowane obiektowo	Obiektowa baza danych to bazy danych, w których informacje są reprezentowane w postaci obiektów jak ma to miejsce w programowaniu obiektowym. Pozwala to zachować spójność między językiem programowania i danymi przechowywanymi w bazie przez ten sam model reprezentacji danych.		- Aplikacje zorientowane obiektowo wymagają mniej kodu.	- Mniejsza efektywność przy prostych, nieskomplikowanych danych i prostych relacjach.
Semistrukturalne	Dane semistrukturalne to model danych oparty na drzewach. Reprezentacja danych jest bardziej elastyczna niż w relacyjnych Bazach matematyczną modelu wybrano graf skierowany. Schemat bazy jest często wpisany bezpośrednio w dane, można to określić jako dane „samo-opisujące się”. Taka struktura bardzo dobrze pasuje do plików w XML.		- Dostarczają elastyczny format	- Duże skomplikowanie przy
<u>Hierarchiczne</u>	W modelu hierarchicznym dane są przechowywane na zasadzie rekordów nadzędnych i podrzędnych, tzn. rekordy przypominają strukturę drzewa. Każdy rekord (z wyjątkiem głównego) jest związany z dokładnie jednym rekordem nadzędnym. Dane w takim modelu są znajdowane na zasadzie wyszukiwania rekordów podrzędnych względem rekordu nadzędnego. Przykładem takiego modelu może być struktura katalogów na dysku twardym komputera.		- Bezpośrednie powiązanie tabel. - Integralność odwołań.	- Brak możliwości dodawania nowych rekordów do tabeli podrzędnej. - Nadmierna ilość danych przez bark obsługi złożonych relacji.
<u>Sieciowe</u>	Model sieciowy bazy danych powstał po to aby rozwiązać problemy modelu hierarchicznego. Jego struktura wizualnie przedstawia odwrócone drzewo. Różnica pomiędzy modelem hierarchicznym a sieciowym polega na tym, że w modelu sieciowym drzewa mogą dzielić ze sobą gałęzie a każde drzewo stanowi część ogólnej struktury bazy danych.		- Dostęp do interesujących nas danych można uzyskać poruszając się po kolekcjach. - Przeszukiwanie możemy zacząć od dowolnej tabeli a następnie poruszać się w góre lub w dół.	- Niemożność zmiany struktury bazy danych bez ponownego tworzenia obsługujących ją programów. - Bardzo skomplikowana struktura.

Postrelacyjne	W modelu postrelacyjnym zakłada się, że są to bazy relacyjne poszerzone na przykład o elementy obiektowości, obsługę XML, rozwiązań analitycznych, zapytania historyczne. Model postrelacyjny powstał w wyniku rozszerzenia relacyjnego modelu baz danych o elementy ułatwiające opisanie skomplikowanej rzeczywistości. Są to na przykład złożone struktury danych, zagnieżdżone relacje, atrybuty wirtualne, abstrakcyjne typy danych czy funkcje rozszeralne.			
NoSQL	Bazy danych, w których można przechowywać, organizować i wyszukiwać dane w inny sposób niż w tabelach relacyjnych.		<ul style="list-style-type: none"> - potrafią przetwarzać dane niestrukturalne - tańsze i prostsze w utrzymaniu (szczególnie w przypadku prostych baz klucz-wartość) 	<ul style="list-style-type: none"> - Brak normalizacji powoduje redundancję. - Brak mechanizmów transakcyjnych.

SQL (Structured Query Language)

SQL (Structured Query Language) - Światowy standard przeznaczony do definiowania, operowania i sterowania danymi w relacyjnych bazach danych.

SQL zapewnia obsługę:

- Zapytań - wyszukiwanie danych w bazie
- Operowania danymi - wstawianie, modyfikowanie i usuwanie
- Definiowania danych - dodawanie do bazy danych nowych tabel
- Sterowania danymi - ochrona przed niepowołanym dostępem

Podzbiory języka SQL

Definicja	Przykłady
SQL DDL (Data Definition Language - język definicji danych) - Światowy standard przeznaczony do definiowania, operowania i sterowania danymi w relacyjnych bazach danych.	<pre>CREATE DATABASE nazwa bazy;DROP DATABASE nazwa bazy; CREATE TABLE nazwa tabeli (nazwa kolumny typ(długość) not null);Możliwe atrybuty do wykorzystania przy tworzeniu tabeli: NOT NULL - pole musi posiadać wartość, może posiadać również wartość NULL.UNSIGNED - pole liczbowe o wartości nieujemnej. AUTO_INCREMENT – automatyczne numerowanie dodanego rekordu. PRIMARY KEY – definicja klucza podstawowego tabeli. Najczęściej występuje wspólnie z opcją AUTO_INCREMENT DROP TABLE nazwa tabeli; TRUNCATE TABLE nazwa tabeli;</pre>
SQL DCL (Data Control Language – język kontroli nad danymi) - stosujemy w przypadku nadania / odebrania uprawnień do obiektów bazodanowych.	<pre>GRANT – nadanie uprawnień do obiektu lub globalnie konkretnemu użytkownikowi. GRANT ALL PRIVILEGES ON TEST TO PKANIA WITH GRANT OPTION; Wydanie w/w zapytania powoduje przyznanie wszystkich możliwych uprawnień do tabeli TEST dla użytkownika PKANIA z dodatkową opcją pozwalającą mu również nadawać prawa do tej tabeli. REVOKE – odbieranie uprawnień. REVOKE ALL PRIVILEGES ON TEST FROM PKANIA;</pre>
SQL DML (Data	INSERT – umieszczenie danych w bazie,

Manipulation Language -język manipulacji danymi -wykonanie operacji na danych (dodawanie, kasowanie, przeglądanie oraz dokonywanie zmian).	<i>INSERT INTO tabela (kolumna,kolumna..) VALUES (wartość,wartość,..);</i> Możliwa jest również składnia skrócona polecenia insert w postaci: <i>INSERT INTO tabela VALUES (wartość,wartość,..);</i> Jednakże do ogólnego stosowania zaleca się pierwszy z powyższych zapisów. <i>UPDATE tabela SET kolumna1=wartość1, kolumna_n=wartość_n WHERE kolumna=wartość;</i> <i>DELETE – usunięcie danych z bazy.</i> <i>DELETE FROM tabela WHERE warunek;</i>
SQL DQL (Data Query Language – językdefiniowania zapytań) -Jest to język zapytań do bazy danych.	Polecenie do wybierania danych – <i>SELECT</i> . <i>SELECT kolumna,kolumna_n,.. FROM tabela;</i> Sortowanie wg wartości kolumny(kolumn). Sortowanie rosnąco - słowo kluczowe ASC (ascend), sortowanie malejaco - słowo kluczowe DESC (descend). <i>SELECT kolumna,kolumna_n,.. FROM tabela ORDER BY kolumna DESC</i> <i>SELECT kolumna,kolumna_n,.. FROM tabela ORDER BY kolumna ASC</i> Dodanie kryteriów wyszukiwania: <i>SELECT kolumna,kolumna_n,.. FROM tabela WHERE kolumna='wartość';</i> Warunki logiczne mogą składać się z operatorów takich jak :znak równości (=) negacja (!=) znak mniejszości (<) lub (<=) znak większości (>) lub (>=)

Budowa relacyjnych baz danych

Teoria relacyjna		Model ER	Relacyjne bazy	Aplikacje
Relacja	Encja	Tabela	Klasa	
Krotka	Instancja	Wiersz	Instancja klasy (obiekt)	
Atrybut	Atrybut	Kolumna	Właściwość, atrybut	
Dziedzina	Dziedzina/Typ	Typ danych	Typ danych	

Typy relacji

Relacja jeden do jednego:

- każdemu wystąpieniu jednej z dwóch encji towarzyszy wystąpienie drugiej encji pozostającej z nim w równoważnym związkę;
- typ rzadko spotykany, ponieważ większość informacji powiązanych w ten sposób byłoby zawartych w jednej tabeli;
- stosowana do podziału tabeli z wieloma polami, do części tabeli ze względów bezpieczeństwa albo do przechowywania informacji odnoszącej się tylko do podzbioru tabeli głównej.

Relacja jeden do wielu:

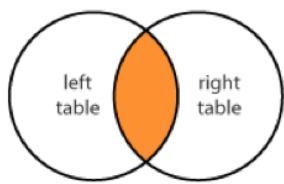
- dla każdej instancji jednej encji istnieje wiele instancji drugiej encji pozostającej z nią w związku;
- realizowana jest poprzez utworzenie atrybutu w encji po stronie wiele (tzw. klucz obcy, ponieważ jest on głównym kluczem w innej tabeli) aby umieścić w nim klucz encji znajdującej się po stronie jeden.
- jest najbardziej powszechnym typem relacji.

Relacja wiele do wielu:

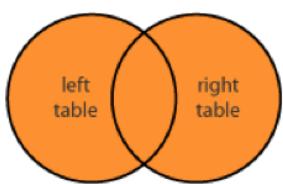
- w tym typie relacji rekord w tabeli A może mieć wiele dopasowanych do niego rekordów z tabeli B i tak samo rekord w tabeli B może mieć wiele dopasowanych do niego rekordów z tabeli A. Jest to możliwe tylko przez zdefiniowanie trzeciej tabeli (nazywanej tabelą łączną), której klucz podstawowy składa się z dwóch pól - kluczy obcych z tabeli A i B;
- Relacja wiele-do-wielu jest definiowana jako dwie relacje jeden-do-wielu z trzecią tabelą.

Joiny

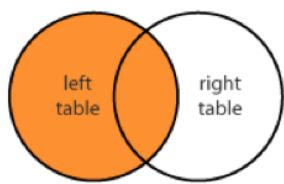
INNER JOIN



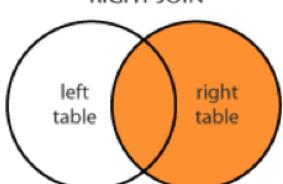
FULL JOIN



LEFT JOIN



RIGHT JOIN



(A-09) Omówić oprogramowanie szkodliwe. Proszę wymienić i krótko scharakteryzować najpopularniejsze rodzaje szkodliwego oprogramowania, oraz przedstawić sposoby i narzędzia służące do obrony przed poszczególnymi typami.

W skrócie

Szkodliwym oprogramowaniem aplikacja, bądź fragment kodu, który zaprojektowano, aby złamać zabezpieczenia naszego urządzenia w celu uszkodzenia, dezaktywacji, bądź kradzieży danych.

- Wirus
- Robak
- Koń trojański
- Spyware
- Ransomware
- Rootkit
- Exploit
- Keylogger

Definicje

Szkodliwym oprogramowaniem (ang. malware) nazywamy każdą aplikację, bądź fragment kodu, który zaprojektowano, aby złamać zabezpieczenia naszego urządzenia w celu uszkodzenia, dezaktywacji, bądź kradzieży danych.

Rodzaj	Opis	Forma obrony
Wirus	Wirus to fragment kodu, który automatycznie dodaje się do innych istniejących plików podczas uruchomienia zarażonego programu. Zwykle występuje w formie załącznika w wiadomości e-mail, czy też odnośniki do innych plików w sieci.	- Działania prewencyjne: Ostrożność względem otwieranych stron, załączników wiadomości. Nieinstalowanie aplikacji z nieznanych źródeł. - Firewall - Antywirus
Robak	Tak samo jak wirus powiela się samodzielnie. Nie nadpisuje obecnych plików, tylko tworzy własne kopie. Może tego dokonać tylko poprzez sieć.	- Działania prewencyjne - Firewall (Wychwyci próbę nawiązania połączenia z innymi urządzeniami w sieci) - Antywirus
Koń trojański	Koń Trojański to program podszywający się pod niegroźna aplikację. Po uruchomieniu koń trojański wykonuje w tle operacje szkodliwe (np. otwarcie określonego portu, czy instalacja wirusów).	- Działania prewencyjne (Koń Trojański potrafi skutecznie wyłączyć ochronę antywirusową, co może spowodować jeszcze większe nasilenie się w komputerze szkodliwego oprogramowania) - Antywirus - Firewall
Spyware	Spyware to szkodliwe oprogramowanie służące do szpiegowania użytkownika i gromadzenia wszelkich informacji na jego temat tj.: dane osobowe, hasła, numery kart płatniczych, adresy mailowe, adresy odwiedzanych stron, czy zainteresowania, bez jego wiedzy i zgody.	- Działania prewencyjne (Występuje często jako dodatkowy i ukryty komponent większego programu, odporny na usuwanie i ingerencję użytkownika) - Antywirus - Zabezpieczenie newralgicznych danych - Użycie programów szyfrujących.
Ransomware	Ransomware jest oprogramowaniem do internetowego wymuszania pieniędzy. Ma za zadanie uniemożliwić dostęp do systemu lub odczyt zapisanych danych, a następnie wyświetlenie komunikatu o żądaniu „okupu” za przywrócenie stanu pierwotnego. Do zakażenia może dojść poprzez ściągnięcie i uruchomienie zainfekowanego pliku, odwiedzanie podejrzanych stron www i klikanie przez użytkownika zainfekowanej reklamy.	- Działania prewencyjne - Antywirus i regularne skany - Regularne aktualizowanie swojego oprogramowania (Ransomware wykorzystuje Exploity, które korzystają z luk w oprogramowaniu. Aktualizowanie blokuje potencjalne luki, które mogą zostać wykorzystane) - Częste tworzenie kopii zapasowych - Odłączanie dodatkowych nośników od urządzenia

1. Omówić bazy danych. Proszę wymienić i krótko omówić podzbiory języka SQL, oraz wyjaśnić zasadę działania relacyjnych baz danych.

Baza danych – zbiór danych zapisanych zgodnie z określonymi regułami. W węższym znaczeniu obejmuje dane cyfrowe gromadzone zgodnie z zasadami przyjętymi dla danego programu komputerowego specjalizowanego do gromadzenia i przetwarzania tych danych. Program taki (często pakiet programów) nazywany jest „systemem zarządzania bazą danych”

Ogólny podział baz danych :

- **bazy relacyjne** – dane są powiązane, wiele tabel może współpracować ze sobą. W bazach relacyjnych wiele tabel danych może współpracować ze sobą (są między sobą powiązane). Bazy relacyjne posiadają wewnętrzne języki programowania, wykorzystujące zwykle SQL do operowania na danych, za pomocą których tworzone są zaawansowane funkcje obsługi danych. Relacyjne bazy danych (jak również przeznaczony dla nich standard SQL) oparte są na kilku prostych zasadach:

1. Wszystkie wartości danych oparte są na prostych typach danych.
2. Wszystkie dane w bazie relacyjnej przedstawiane są w formie dwuwymiarowych tabel (w matematycznym jargonie noszących nazwę „relacji”). Każda tabela zawiera zero lub więcej wierszy (w tymże jargonie – „krotki”) i jedną lub więcej kolumn („atrybutów”). Na każdy wiersz składają się jednakowo ułożone kolumny wypełnione wartościami, które z kolei w każdym wierszu mogą być inne.
3. Po wprowadzeniu danych do bazy, możliwe jest porównywanie wartości z różnych kolumn, zazwyczaj również z różnych tabel, i scalanie wierszy, gdy pochodzące z nich wartości są zgodne. Umożliwia to wiązanie danych i wykonywanie stosunkowo złożonych operacji w granicach całej bazy danych.
4. Wszystkie operacje wykonywane są w oparciu o algebrę relacji, bez względu na położenie wiersza tabeli. Nie można więc zapytać o wiersze, gdzie ($x=3$) bez wiersza pierwszego, trzeciego i piątego. Wiersze w relacyjnej bazie danych przechowywane są w porządku zupełnie dowolnym – nie musi on odzwierciedlać ani kolejności ich wprowadzania, ani kolejności ich przechowywania.

5. Z braku możliwości identyfikacji wiersza przez jego pozycję pojawia się potrzeba obecności jednej lub więcej kolumn niepowtarzalnych w granicach całej tabeli, pozwalających odnaleźć konkretny wiersz. Kolumny te określa się jako „klucz podstawowy” (ang. primary key) tabeli.

- **bazy obiektowe** – dane są przechowywane w strukturze obiektowej, cechą charakterystyczną takiej bazy jest to, że przechowuje obiekty o dowolnych strukturach wraz z przypisanymi do nich procedurami;
- **bazy relacyjno-obiektowe** – dane są przechowywane w strukturze obiektowej, ale są powiązane ze sobą tak jak w bazach relacyjnych;
- **bazy strumieniowe** – dane są przetwarzane jako strumienie danych, model zakłada, że niektóre lub wszystkie napływające do systemu dane nie są dostępne w dowolnej chwili (system zarządzania taką bazą nazywany jest DSMS - ang. Data Stream Management System);
- **bazy temporalne** – są podobne do baz relacyjnych, z tą różnicą, że każdy rekord posiada swój znacznik czasowy (tzw. stempel) określający, czy w danym czasie zapisana wartość jest prawdziwa;
- **bazy nierelacyjne** – dane są zapisywane w formie klucz-wartość i nie są ze sobą powiązane, w takiej bazie najczęściej nie ma wymagania, żeby obiekty były jednorodne pod względem struktury.

Składnia SQL

Użycie SQL, zgodnie z jego nazwą, polega na zadawaniu zapytań do bazy danych.

Zapytania można zaliczyć do jednego z czterech głównych podzbiorów:

- SQL DML (ang. Data Manipulation Language – „język manipulacji danymi”),
- SQL DDL (ang. Data Definition Language – „język definicji danych”),
- SQL DCL (ang. Data Control Language – „język kontroli nad danymi”).
- SQL DQL (ang. Data Query Language – „język definiowania zapytań”).

DML (Data Manipulation Language) służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania oraz dokonywania zmian.

Najważniejsze polecenia z tego zbioru to:

INSERT – umieszczenie danych w bazie,

UPDATE – zmiana danych,

DELETE – usunięcie danych z bazy.

Dane tekstowe muszą być zawsze ujęte w znaki pojedynczego cudzysłowa (').

Dzięki **DDL** (Data Definition Language) można operować na strukturach, w których dane są przechowywane – czyli np. dodawać, zmieniać i kasować tabele lub bazy. Najważniejsze polecenia tej grupy to:

CREATE (np. CREATE TABLE, CREATE DATABASE, ...) – utworzenie struktury (bazy, tabeli, indeksu itp.),

DROP (np. DROP TABLE, DROP DATABASE, ...) – usunięcie struktury,

ALTER (np. ALTER TABLE ADD COLUMN ...) – zmiana struktury (dodanie kolumny do tabeli, zmiana typu danych w kolumnie tabeli).

DCL (Data Control Language) ma zastosowanie do nadawania uprawnień do obiektów bazodanowych. Najważniejsze polecenia w tej grupie to:

GRANT – służące do nadawania uprawnień do pojedynczych obiektów lub globalnie konkretnemu użytkownikowi (np. GRANT ALL PRIVILEGES ON EMPLOYEE TO PIOTR WITH GRANT OPTION – przyznanie wszystkich praw do tabeli EMPLOYEE użytkownikowi PIOTR z opcją pozwalającą mu nadawać prawa do tej tabeli).

REVOKE – służące do odbierania wskazanych uprawnień konkretnemu użytkownikowi (np. REVOKE ALL PRIVILEGES ON EMPLOYEE FROM PIOTR – odebranie użytkownikowi wszystkich praw do tabeli EMPLOYEE).

DENY – służące do zabraniania wykonywania operacji

DQL (Data Query Language) to język formułowania zapytań do bazy danych. W zakres tego języka wchodzi jedno polecenie – SELECT. Często SELECT traktuje się jako część języka DML, ale to podejście nie wydaje się właściwe, ponieważ DML z definicji służy do manipulowania danymi – ich tworzenia, usuwania i aktualniania. Na pograniczu obu języków znajduje się polecenie SELECT INTO, które dodatkowo modyfikuje (przepisuje, tworzy) dane.

Przykładowe zapytania

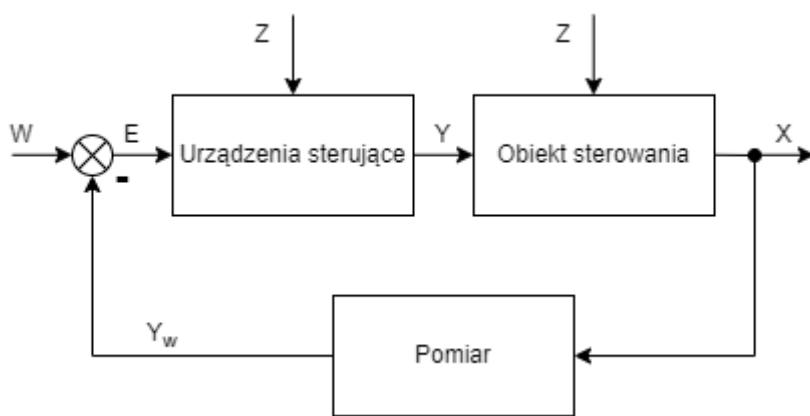
```
SELECT *
  FROM pracownicy
 WHERE pensja > 2000
 ORDER BY staz DESC;
```

Zwraca tabelę (listę) utworzoną ze wszystkich kolumn (*) tabeli „pracownicy” (**FROM pracownicy**) zawierającą pracowników, których pensja jest większa niż 2000 (**WHERE pensja > 2000**) i sortuje wynik malejąco według parametru **staz** (**ORDER BY staz DESC**).

B1. Omówić układ automatycznej regulacji. Proszę narysować i omówić schemat układu automatycznej regulacji, oraz przedstawić jakie cele stawia się układom automatycznej regulacji.

Cel automatyzacji procesów produkcyjnych - uniezależnienie wydajności i kosztów produkcji oraz jakości wytwarzanych wyrobów od wkładu pracy i kwalifikacji oraz ciągłej obserwacji personelu obsługującego urządzenia produkcyjne.

Układy automatycznej regulacji (UAR) są to układy sterowania posiadające sprzężenie zwrotne, których zadaniem jest sterowanie procesem (sygnałami wyjściowymi) w zależności od doprowadzonych sygnałów wejściowych. Schemat blokowy takiego układu przedstawiono na Rysunku 3.



Rysunek 0-1. Schemat układu automatycznej regulacji

X – wielkość regułowana (np. temperatura, ciśnienie, poziom);

Y – wielkość nastawiająca (np. przepływ);

Y_w -wartość mierzona

Z – wielkość zakłócająca (np. temperatura);

W – wartość zadana wielkości regułowanej;

$E=W-X$ – uchyb regulacji; $E=W-Y_w$ w wypadku układu z urządzeniem pomiarowym

Klasyfikacja układów automatycznego sterowania:

- Układ stabilizujący - układ o stałej wartości zadanej, który ma za zadanie zapobiegać zmianom wielkości regułowanej przy zmieniających się w czasie zakłócenach.
- Układ programowy - wartość zadana $w(t)$ jest z góry określona funkcją czasu, czyli zmieniającą się według pewnego programu $w = f(t)$ (rozruch silnika maszyny wyciągowej, w której obroty silnika mają narastać liniowo)
- Układ nadążający (śledzący) - wartość zadana $w(t)$ jest funkcją czasu przy czym jest ona nieznana ($w = ?$); zmiany tej funkcji nie zależą od procesu zachodzącego wewnętrznie do układu, ale związane są ze zjawiskami występującymi na zewnątrz.

Ważnym elementem układu automatycznej regulacji jest regulator. Regulator jest to urządzenie działające w układzie automatycznej regulacji wytwarzające sygnał sterujący na podstawie sygnału uchybu regulacji.

Poprawne działanie układu regulacji zależy od doboru odpowiedniego typu regulatora do sterowanego obiektu. Przyjęty typ regulatora określa zasadę regulacji, tzn. zależność wiążącą sygnał odchyłki e z sygnałem sterującym u . W regulatorach ciągłych ta zależność opiera się na proporcjonalności, całkowaniu i różniczkowaniu odchyłki e .

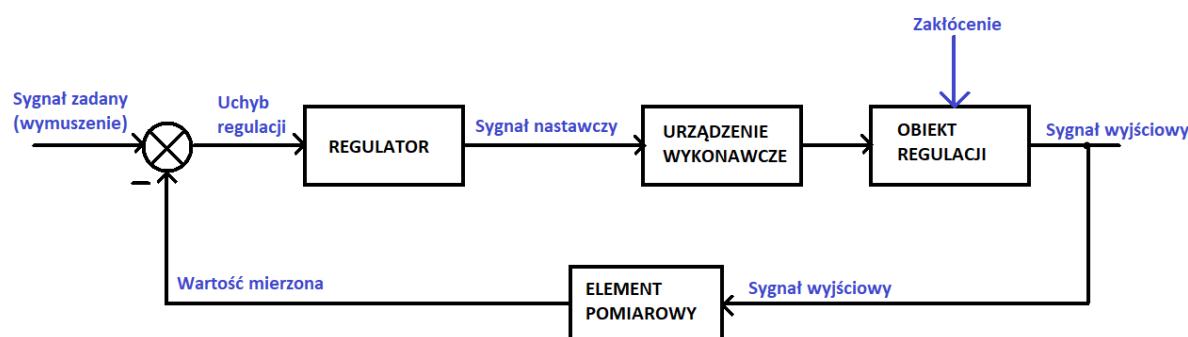
Najczęściej stosowane w praktyce typy regulatorów to:

- Regulator proporcjonalny P,
- Regulator całkujący I,
- Regulator proporcjonalno – całkujący PI,
- Regulator proporcjonalno – różniczkujący PD,
- Regulator proporcjonalno – całkującą – różniczkujący PID

Przy dobieraniu wartości nastaw najważniejszym wymaganiem jest zapewnienie stabilności układu, tzn. że po pobudzeniu stałym niezerowym sygnałem odpowiedź układu musi dążyć do ustalonej wartości. Istnieje wiele metod pozwalających sprawdzić stabilność. Jedna z nich opiera się na sprawdzeniu znaków wszystkich pierwiastków wielomianu charakterystycznego układu (mianownika transmitancji operatorowej). Jeśli wszystkie pierwiastki są ujemne, to układ będzie stabilny.

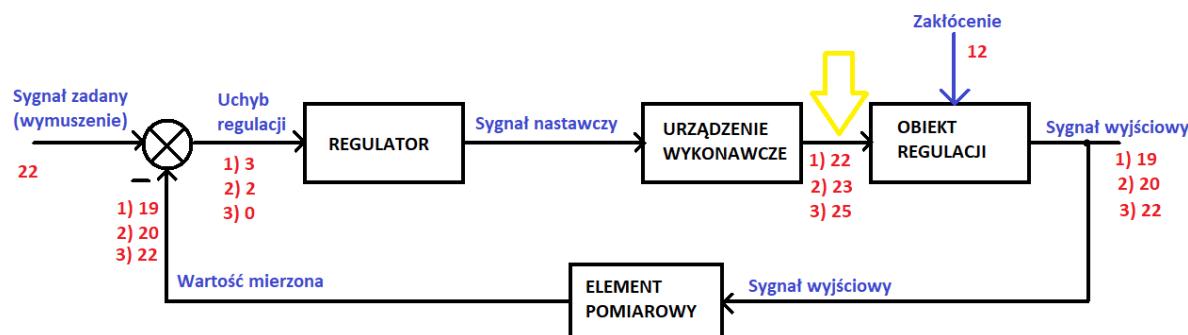
2. Omówić układ automatycznej regulacji. Proszę narysować i omówić schemat układu automatycznej regulacji, oraz przedstawić jakie cele stawia się układom automatycznej regulacji. B-01

Sam układ regulacji to połączenie elementów automatyki, które współdziałają ze sobą realizując wyznaczone zadanie. Układ automatycznej regulacji wyróżnia ujemne sprzężenie zwrotne, dzięki któremu sygnał wyjściowy jest dodatkowo przetwarzany przez element pomiarowy i wartość mierzona jest odejmowana od sygnału. Wynikiem tej różnicy jest sygnał w postaci uchybu regulacji. Poniżej został przedstawiony rysunek ilustrujący taki układ.



Rysunek 1 Schemat układu automatycznej regulacji ze sprzężeniem zwrotnym

Sprzężenie zwrotne ujemne – z cybernetycznego punktu widzenia stanowi fundamentalny mechanizm samoregulacyjny, zachodzi, gdy informacja o rozbieżności pomiędzy wartością faktyczną i referencyjną parametru układu wykorzystywana jest do zniwelowania tej różnicy. Ogólny wysoki poziom ujemnego sprzężenia zwrotnego sprzyja stabilności układu.



Rysunek 2 Schemat układu automatycznej regulacji z przykładowymi wartościami

W pierwszym kroku przepisałem wartości uzyskane w strukturze otwartej, czyli sygnał zadany, zakłócenie i temperaturę docierającą do obiektu regulacji z urządzenia wykonawczego. Żółta strzałka wskazuje początek analizy układu. Pierwsza uzyskana wartość w tym miejscu wynosi 22 stopnie. W konsekwencji na wyjściu otrzymaliśmy 19 stopni. Sygnał ten jest przetwarzany przez element pomiarowy i obliczany jest uchyb, który wynosi 3 stopnie. W konsekwencji regulator tak steruje urządzeniem wykonawczym, że na wyjściu uzyskujemy drugą wartość, czyli 23 stopnie. Powoduje to zwiększenie sygnału wyjściowego do 20 stopni, a uchyb wynosi w tym przypadku 2 stopnie. Wciąż występuje różnica między sygnałem zadanym, a wyjściowym, zatem regulator ponownie zwiększa sygnał sterujący urządzeniem wykonawczym, który generuje temperaturę 25 stopni. Efektem tego jest uzyskanie temperatury na wyjściu zgodnej z sygnałem zadanym, a zatem uchyb wynosi teraz zero.

Klasyfikacja układów automatycznego sterowania:

- a) Układ stabilizujący – układ o stałej wartości zadanej, który ma za zadanie zapobiegać zmianom wielkości regulowanej przy zmieniających się w czasie zakłócenach.
- b) Układ programowy – wartość zdania $w(t)$ jest z góry określona funkcją czasu, czyli zmieniającą się według pewnego programu $w=f(t)$ (rozruch silnika maszyny wyciągowej, w której obroty silnika mają anastać liniowo)
- c) Układ nadążany śledzący – wartość zadana $w(t)$ jest funkcją czasu przy czym jest ona nieznana ($w=?$); zmiany tej funkcji nie zależą od procesu zachodzącego wewnątrz układu, ale związane są ze zjawiskami występującymi na zewnątrz.

B-03

- ▶ Regulator PI – w automatyce, regulator składający się z członu proporcjonalnego P o wzmacnieniu K_V oraz całkującego I o czasie całkowania T_i . Transmitancję określa wzór:
- ▶ $G_{PI}(s) = K_p \left(1 + \frac{1}{T_i s}\right)$
- ▶ Regulatory typu PI pozwalają na eliminację wolnozmennych zakłóceń, co przekłada się na zerowy uchyb ustalony, niemożliwy do osiągnięcia w regulatorach typu P lub typu PD. Wzmocnienie członu całkującego musi być jednak ograniczone, ponieważ wprowadza on ujemne przesunięcie fazowe, które osłabia tłumienie uchybu regulacji
- ▶ Regulator PI jak sama nazwa wskazuje jest połączeniem regulatora proporcjonalnego P i całkującego I. Łączy on w sobie ich zalety, a mianowicie szybkość reakcji regulatora P i dokładność regulacji dla regulatora I. Układ regulacji wyposażony w PI jest zatem dokładny i szybki.
- ▶ Regulatory PI eliminują wolnozmienne zakłócenia, co pozwala uzyskać zerowy uchyb ustalony, niemożliwy do osiągnięcia w regulatorze P lub PD. Ich wadą jest natomiast pogorszenie stabilności układu i problemy z ograniczeniem całkowania.
- ▶ Dobór nastaw ma na celu otrzymanie stabilnego układu regulacji spełniającego przyjęte kryterium jakości regulacji. Korzysta się przy tym z tzw. bezpośrednich wskaźników jakości regulacji określanych na podstawie przebiegu zmiennej procesowej w zamkniętym układzie regulacji podczas skokowych zmian wartości zadanej lub wielkości zakłócenia.

Najpopularniejsze to:

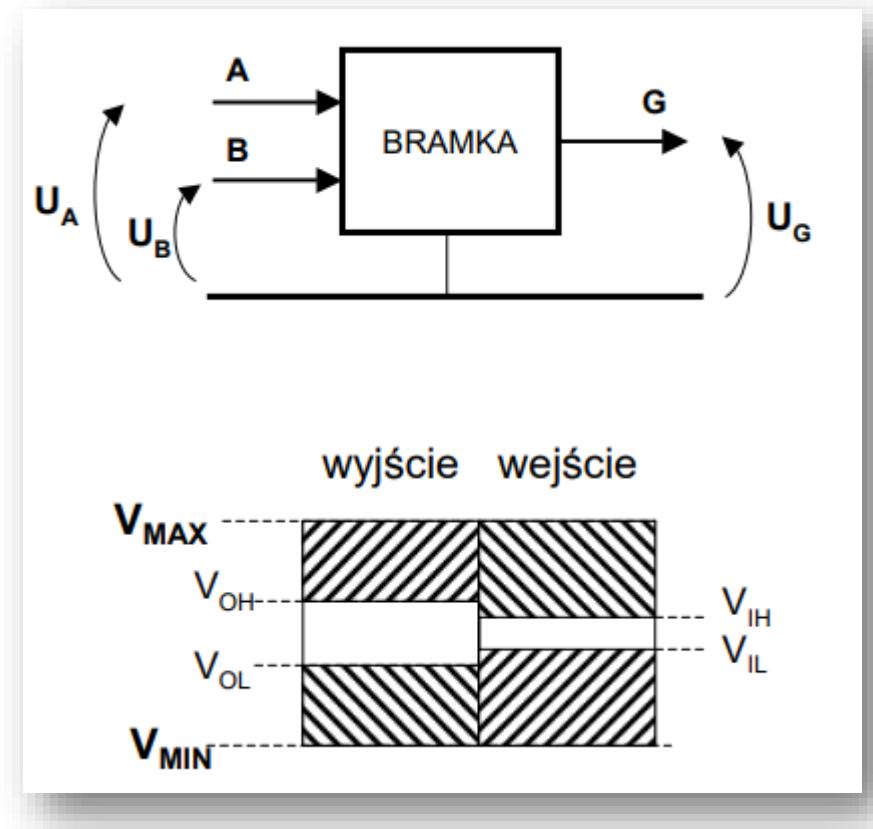
- ▶ Czas ustalania(czas regulacji) – przedział czasu od chwili podania skokowej zmiany wartości zadanej lub zakłócenia do chwili od której różnica między wartością zmiennej procesowej a jej wartością ustaloną nie przekracza $\pm\Delta$, gdzie za Δ przyjmuje się najczęściej 1, 2 lub 5% wartości ustalonej
- ▶ Przeregulowanie – stosunek:
 - ▶ Przy skokowej zmianie wartości zadanej $\sigma = \frac{e_2}{e_1} * 100\%$
 - ▶ Przy skokowej zmianie zakłócenia $\sigma = \frac{e_4}{e_3} * 100\%$
 - ▶ Gdzie: e_1 i e_3 to maksymalne początkowe uchyby regulacji, e_2 i e_4 to maksymalne uchyby regulacji o znaku przeciwnym
- ▶ Uchyb w stanie ustalonym – poziom uchybu po ustaniu procesów przejściowych
- ▶ Pierwsza metoda Zieglera-Nicholsa - metoda cyklu granicznego polega na wyłączeniu w układzie zamkniętym działania całkującego i różniczkującego oraz takim ustawieniu współczynnika wzmacnienia, przy którym w układzie powstają oscylacje o stałej amplitudzie.
- ▶ Druga metoda Zieglera-Nicholsa – polega na analizie odpowiedzi obiektu na wymuszenie skokowe. Odpowiedź aperiodyczna obiektu aproksymowano za pomocą odcinka osi czasowej (opóźnienie L) i półprostej o nachyleniu α

(B-05): Omówić bramki logiczne.

Proszę wymienić i krótko scharakteryzować działanie bramek logicznych, oraz przedstawić różnice w budowie i projektowaniu układów kombinacyjnych i sekwencyjnych.

Czym są bramki logiczne?

- Bramką logiczną nazywamy prosty obwód elektroniczny realizujący funkcję logiczną. Pewien zakres napięcia odpowiada stanowi logicznemu 0, a inny zakres stanowi logicznemu 1.
- Zwyczajnie stanowi 0 przypisujemy niższe napięcie niż stanowi 1, dlatego stan 0 nazywamy stanem logicznym **niskim**, a 1 – **wysokim**.
- Sterowane mogą być tylko wejścia bramki.



Algebra Boole'a

- Algebra Boole'a to rodzaj algebry ogólnej, stosowany w matematyce, informatyce teoretycznej oraz elektronice cyfrowej. Jej nazwa pochodzi od nazwiska matematyka, filozofa i logika George'a Boole'a. Mówiąc o niej, mówią o tym, że dana może przyjąć tylko jedną z dwóch możliwych wartości – **true** lub **false** (1 lub 0). Na danych można wykonywać poniższe operacje:
- Prawa Algebry Boole'a:

Prawo	Działanie
-------	-----------

Łączność	$a \wedge (b \wedge c) = (a \wedge b) \wedge c$
Przemienność	$a \vee b = b \vee a$
Rozdzielność	$a \vee (b \wedge c) = (a \vee b) \wedge (a \vee c)$
Odwrotność	$a \vee \neg a = 1$

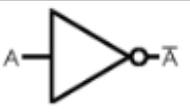
X AND Y		
X	Y	XY
0	0	0
0	1	0
1	0	0
1	1	1

X OR Y		
X	Y	X+Y
0	0	0
0	1	1
1	0	1
1	1	1

NOT X	
X	\bar{X}
0	1
1	0

Rodzaje bramek logicznych

- Podstawowymi elementami logicznymi, stosowanymi powszechnie w budowie układów logicznych, są elementy realizujące funkcje logiczne: sumy, iloczynu i negacji. Są to odpowiednio bramki **OR**, **AND** i **NOT**. Za pomocą dwóch takich bramek (**OR** i **NOT** lub **AND** i **NOT**) można zbudować układ realizujący dowolną funkcję logiczną, układy takie nazywa się układami zupełnymi.
- Bramki **NAND** (negacja koniunkcji) oraz **NOR** (negacja sumy logicznej) nazywa się funkcjonalnie pełnymi, ponieważ przy ich użyciu (tzn. samych **NAND** lub samych **NOR**) można zbudować układ realizujący dowolną funkcję logiczną.
- Inną często stosowaną bramką logiczną jest **XOR**, która wykorzystywana jest w układach arytmetyki takich jak sumatory czy subtraktory.

FUNKCJA LOGICZNA	SYMBOL LOGICZNY	WYRAŻENIE ALGEBRAICZNE	TRUTH TABLE		
			Inputs		Output
		A	B	Y	
AND		$A \cdot B = Y$	0	0	0
			0	1	0
			1	0	0
			1	1	1
OR		$A + B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	1
NOT <i>(Inverter)</i>		$A = \bar{A}$	0		1
NAND		$\overline{A \cdot B} = Y$	0	0	1
			0	1	1
			1	0	1
			1	1	0
NOR		$\overline{A + B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	0
XOR		$A \oplus B = Y$	0	0	0
			0	1	1
			1	0	1
			1	1	0
XNOR		$\overline{A \oplus B} = Y$	0	0	1
			0	1	0
			1	0	0
			1	1	1

Układy kombinacyjne vs sekwencyjne

- Układy kombinacyjne stanowią podstawowy element budulcowy układów większej skali integracji. W szczególności wyróżnia się: dekodery, multipleksery, demultipleksery, komparatory i układy arytmetyczne. Charakteryzują się tym, że stan wyjść zależy **wyłącznie** od stanu wejść.
- Układem sekwencyjnym nazywamy jeden z rodzajów układów cyfrowych charakteryzujący się tym, że stan wyjść „y” zależy od stanu wejść „x” w danej chwili oraz wejść stanu poprzedniego, zwanego stanem wewnętrznym, pamiętanego w zespole rejestrów. Układy te stosuje się do tworzenia np. przerutników.
- Różnicą zatem jest to, że stan wyjść w układzie kombinacyjnym zależy tylko od stanu wejść, a w układach sekwencyjnych stan wyjść zależy od stanu wejść w danej chwili oraz wejść stanu wewnętrznego.

- 2. Omówić aktuatorów. Proszę wymienić rodzaje poznanych aktuatorów, oraz porównać dwa wybrane rodzaje pod względem siły i drogi nastawiania.

Element wykonawczy, urządzenie wykonawcze, organ wykonawczy, człon wykonawczy, moduł wykonawczy, aktuator – urządzenie występujące w układach regulacji, które na podstawie sygnału sterującego wypracowuje sygnał wejściowy do obiektu regulacji. W języku branżowym, są one nazywane „Wyrobnikami” (zgodnie ze standardem KNX).

- Działanie aktuatora

Z technicznego punktu widzenia podjęciem aktuatora rozumie się zestawienie przetwornika energii z członem nastawczym mocy. Człon nastawiający mocy łączy wschodząca energię W (zwykle elektryczną) z sygnałem nastawiającym Y. W ten sposób powstaje zmodulowana energia, która przez przetwornik jest transformowana w odpowiedni rodzaj energii wielkości nastawiającej (najczęściej energię mechaniczną). Zatem w układach mechatronicznych aktuatorzy stanowią człon łączący przetwarzanie informacji z mechanicznym układem podstawowym.

Porównanie dwóch wybranych aktuatorów

- Porównanie dwóch wybranych aktuatorów względem siły i drogi nastawiania.
- Silnik krokowy (aktuator elektromechaniczny) – pozwala na uzyskanie niewielkiej siły nastawiania. Nie działa natychmiastowo, lecz wymaga pewnej drogi nastawiania do uzyskania żądanej siły. Zaletą jest to, że może utrzymywać siłę nastawiania na znacznym dystansie.
 - Aktuator hydrauliczny – pozwala na uzyskanie bardzo dużych sił nastawiania, lecz wymaga do tego znacznej drogi.
 - Aktuator piezoelektryczny – wykorzystuje zmianę kształtu piezoelektryka pod wpływem pola elektrycznego. Duże siły nastawiania przy małej drodze, co powoduje dużą dokładność takich aktuatorów.

Omówić roboty przemysłowe. Proszę wymienić i krótko omówić działanie typowych członów konstrukcyjnych manipulatorów przemysłowych, oraz omówić rodzaje układów współrzędnych służących do opisu sterowania robotem

Klasyfikacja

Rodzaje robotów przemysłowych można rozróżnić na podstawie różnych wyznaczników. Można klasyfikować biorąc pod uwagę zasadnicze cechy ich budowy, sposób sterowania, a także przeznaczenie, rodzaj napędu.

Klasyfikacja ze względu na przeznaczenie

- Roboty przemysłowe
- Roboty naukowo-szkoleniowe
- Roboty medyczne
- Roboty wojskowe
- Roboty badawcze

Klasyfikacja ze względu na sterowanie

- Robot sekwencyjny – posiada układ sterowania pozwalający na wykonanie kolejno zaprogramowanych ruchów i czynności
- Robot realizujący trajektorie – realizujący ustaloną procedurę sterowanych ruchów zgodnie z instrukcją programu
- Robot adaptacyjny – posiada sensoryczny lub adaptacyjny układ sterowania, możliwe są korekty zaprogramowanych ruchów w zależności od otoczenia
- Teleoperator – robot ze sterowaniem zdalnym, realizowanym przez operatora

Klasyfikacja ze względu na budowę

- Robot monolityczny – robot o niezmienialnej konstrukcji mechanizmu
- Robot modułowy – producent dostarcza konkretne zespoły ruchu, z których można skonstruować daną jednostkę
- Robot pseudo-modułowy – stała konstrukcja z możliwością wymiany poszczególnych modułów

Klasyfikacja ze względu na rodzaj napędu

- Napędy Pneumatyczne – np. Silnik wahadłowy
- Napędy Hydrauliczne – np. Silnik liniowy lub obrotowy

- Napędy Elektryczne – np. Silnik elektryczny prądu stałego lub przemiennego

Klasyfikacja ze względu na rodzaj napędu

- Napędy Pneumatyczne – np. Silnik wahadłowy
- Napędy Hydrauliczne – np. Silnik liniowy lub obrotowy
- Napędy Elektryczne – np. Silnik elektryczny prądu stałego lub przemiennego

Klasyfikacja ze względu na rodzaj napędu

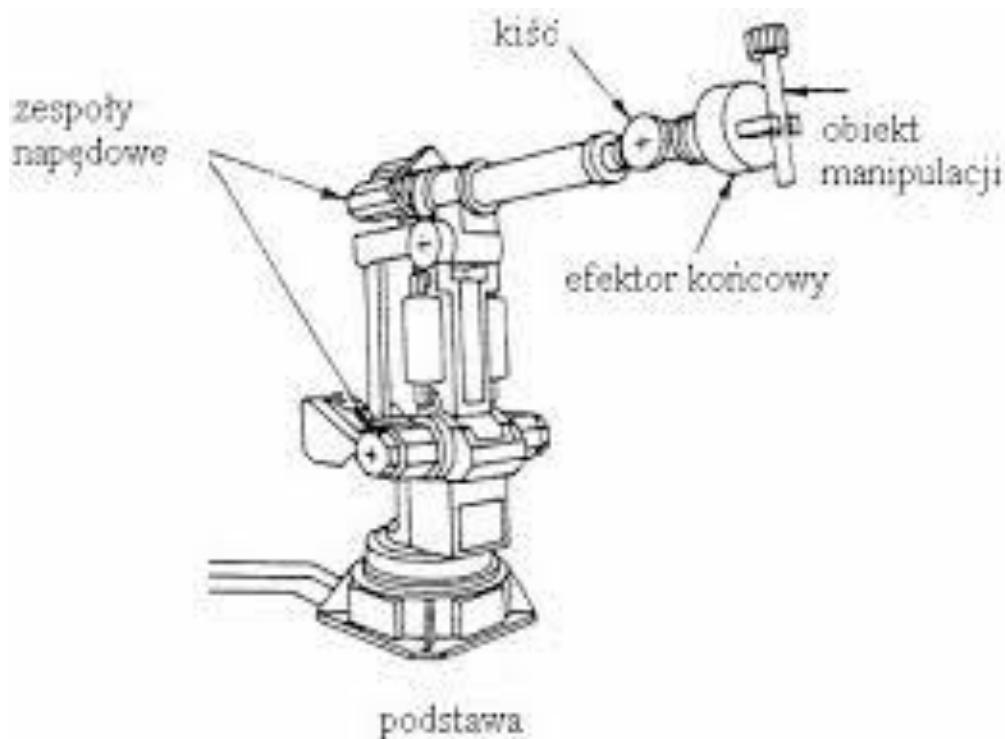
- Napędy Pneumatyczne – np. Silnik wahadłowy
- Napędy Hydrauliczne – np. Silnik liniowy lub obrotowy
- Napędy Elektryczne – np. Silnik elektryczny prądu stałego lub przemiennego

Manipulatory przemysłowe

Narzędzie, które ułatwia pracę z komponentami i produktami gotowymi o wadze od kilku kilogramów do nawet 1,5 tony. Standardowo manipulator przemysłowy jest zintegrowany z systemami równoważenia wagi transportowanego ładunku. Dzięki temu umożliwia precyzyjne przenoszenie oraz manipulowanie różnymi ciężarami, jednocześnie nie wymagając zaangażowania siły fizycznej operatora.

Budowa manipulatora

- Podstawa – płyta lub inna konstrukcja stanowiąca pierwszy człon układu kinematycznego robota
- Zespoły napędowe – połączone czloni i napędzane przeguby ustawiające położenie kiścia
- Kiść – zespół połączonych członów i napędzanych przegubów między ramieniem, a elementem roboczym
- Efektor końcowy – urządzenie przeznaczone do chwycenia i utrzymywania obiektu manipulacji albo do bezpośredniego wykonania operacji technologicznej realizowanej przez robota

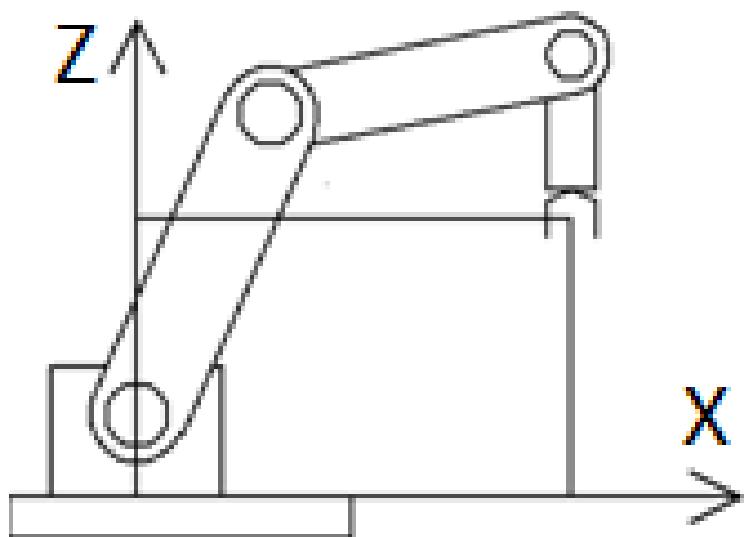


Rodzaje układów współrzędnych służących do opisu sterowania robotem

Aby zrozumieć istotę ruchu robota, należy rozróżnić proces programowania (sterowanie ręczne) od pracy robota w cyklu automatycznym. Ogólnie rzecz biorąc, podczas programowania poruszamy robotem, zmieniając odpowiednio położenie i orientację w przestrzeni tzw. punktu TCP, który w praktyce jest końcówką narzędzia zamocowanego na robocie. Znaczenie tutaj mają układy współrzędnych

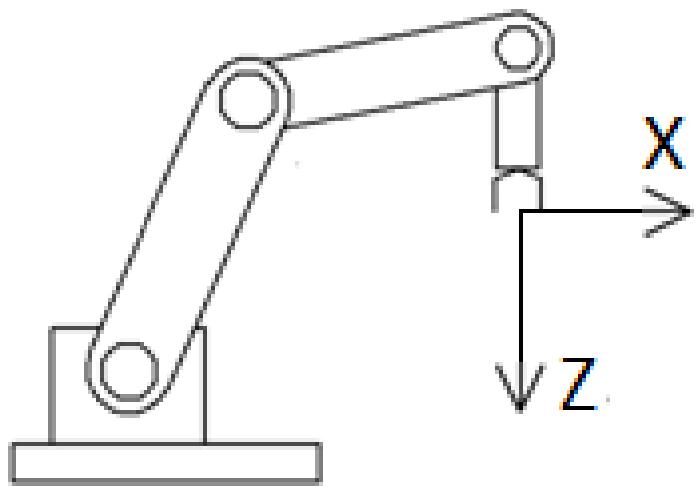
Base

Wzdłuż osi układu kartezjańskiego (XYZ) związanego z podstawą robota (nazywany też WORLD)



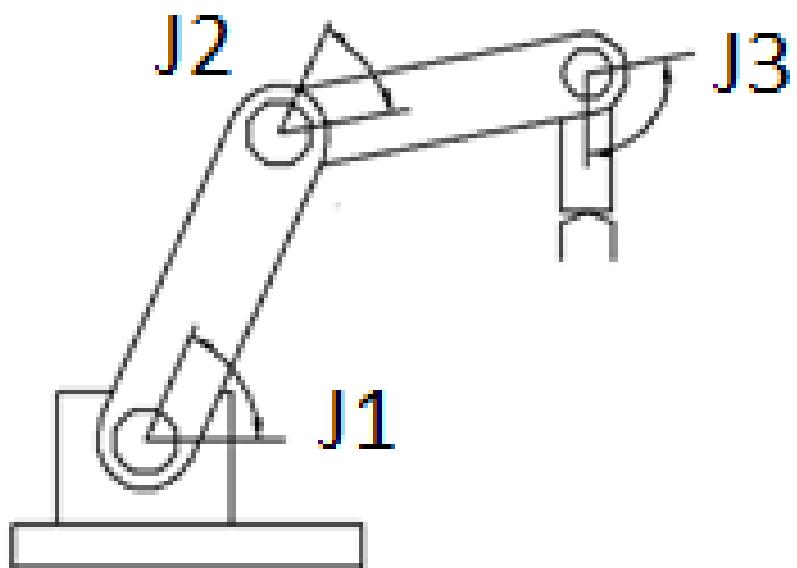
Tool

Wzdłuż osi układu kartezjańskiego (XYZ) związanego z końcówką narzędzia robota



Joint

Obroty poszczególnych osi robota



2. Omówić podstawowe pasywne elementy elektryczne. Proszę omówić przeznaczenie podstawowych pasywnych elementów elektrycznych, oraz omówić ich modelowanie w zależności od typu obwodu.

Elementy pasywne (elektroniczne bierne) to elementy elektryczne nie wytwarzające energii elektrycznej. **Elementami biernymi** nazywamy takie elementy elektryczne/ elektroniczne, które tylko i wyłącznie pobierają energię elektryczną. Elementy bierne mają możliwość gromadzenia energii.

Według podstawowego podziału dzielimy je na rezystory, kondensatory, cewki, transformatory.

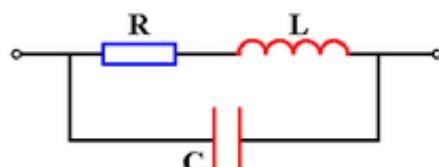
Rezystor

Rezystory to najczęściej pojawiające się elementy pasywne w układach elektronicznych. Ich podstawowym celem zastosowania jest ograniczenie przepływu prądu elektrycznego. Najważniejszym parametrem, jaki opisuje rezystory, jest ich rezystancja, która jest mierzona w omach i zależy od rezystywności materiału, z którego rezystory są wykonane, a także ich wymiarów geometrycznych. Rezystancja jest współczynnikiem proporcjonalności między prądem płynącym przez rezistor a napięciem pomiędzy jego wyprowadzeniami. Rezystory stosuje się do:

- zabezpieczenia przed przeciążeniem elementu
- regulacji parametrów urządzenia (rezistor nastawny)
- wytwarzania ciepła (np. w kuchenkach czy piecach oporowych)
- zmniejszenia napięcia



• Schemat rezystora



Schemat zastępczy rezystora

$$u = Ri$$

$$i = \frac{u}{R}$$

R – rezystancja właściwa

C – pojemność, rzędu pF

L – indukcyjność, rzędu nH

Cewka

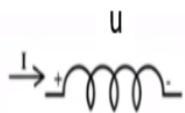
Podobnie jak rezistory, cewki są elementami, które nie wytwarzają energii elektrycznej, ale mogą ją magazynować i uwalniać. Najważniejszym parametrem cewek jest indukcyjność, która jest mierzona w henrach. Indukcyjność określa się jako iloraz strumienia pola magnetycznego do prądu płynącego przez cewkę. W najprostszym ujęciu, cewka jest przewodem uformowanym w zwoje – indukcyjność cewki wzrasta wraz z ilością zwojów i opisuje ona zdolność cewki do magazynowania energii w polu magnetycznym. Jeśli uzwojenie cewki nawiniemy na rdzeń wykonany z materiału ferromagnetycznego, wówczas wzrośnie jej indukcyjność. Dla prądu stałego, cewka jest zwykłym przewodem wykazującym charakter rezystancyjny. Natomiast dla przebiegów elektrycznych zmiennych w czasie, istotną rolę odgrywa reaktancja indukcyjna cewki, której wartość rośnie wprost proporcjonalnie do częstotliwości prądu przez nią płynącego. Cewki stanowią fundamentalny element konstrukcyjny transformatorów, dławików, przekaźników elektromechanicznych, przetworników elektromagnetycznych, a także innych podzespołów indukcyjnych stosowanych w elektronice.

Cewki stosuje się do:

- blokowania przepływu prądu przemiennego w obwodzie,
- zwierania prądu (napięcia) stałego,
- pomiaru upływu czasu na podstawie zanikania przepływu prądu,
- budowania obwodów oscylacyjnych,
- budowania filtrów na określone częstotliwości,
- sprzęgania stopni wzmacniaczy,
- obniżania lub podwyższania napięcia.

Podstawowymi parametrami cewki są jej indukcyjność oraz częstotliwość rezonansowa.

Indukcyjność to inaczej zdolność cewki do przechowywania energii w postaci pola magnetycznego wywoływanego przez przepływ prądu. Indukcyjność mierzy się w Henrach i wyraża jako stosunek chwilowego napięcia do zmiany prądu w czasie.



$$u = L \frac{di}{dt} \quad i = \frac{1}{L} \int_0^T U dt + io$$

Cewka charakteryzuje się tym, że w obwodach prądu zmiennego sinusoidalnego, w stanie ustalonym napięcie na cewce wyprzedza o 90° prąd płynący w cewce.

Impedancia idealnej cewki równa iloczynowi jej reaktancji i jednostki urojonej:

$$ZL = jXL$$

Reaktancja cewki wyraża wzór:

$$XL = \omega L$$

gdzie:

ω – pulsacja,
L - indukcyjność cewki

Kondensatory

Podobnie jak cewki, kondensatory są również elementami pasywnymi, które wykazują zdolność do magazynowania i uwalniania energii. Z tą różnicą, że kondensator magazynuje energię w polu elektrycznym. W podstawowym zarysie, kondensator stanowi układ dwóch elektrod oddzielonych względem siebie wkładem z materiału dielektrycznego. Najważniejszym parametrem określającym kondensator, jest jego pojemność elektryczna, która jest mierzona w faradach. W najprostszym ujęciu, pojemność elektryczna jest stosunkiem ładunku elektrycznego zmagazynowanego między elektrodami kondensatora do napięcia pomiędzy tymi okładzinami. Pojemność kondensatorów zależy także od ich wymiarów geometrycznych oraz przenikalności elektrycznej materiału dielektryka między elektrodami. Najczęściej dielektryk jest ceramiczny lub wykonany polimerów i tworzyw sztucznych, takich jak np. poliester czy polipropylen – szczególnie ten drugi rodzaj

kondensatorów jest wykonany z większą dokładnością i dlatego często używa się ich w produkcji sprzętu audio i aparatury pomiarowej. W przeciwieństwie do cewek, kondensatory dla prądu stałego wykazują właściwości zbliżone do przerwy w obwodzie, kiedy są naładowane. Natomiast dla przebiegów zmiennych w czasie, wykazują reaktancję pojemnościową, której wartość jest odwrotnie proporcjonalna względem częstotliwości prądu przez kondensator płynącego. Większość kondensatorów nie ma określonej polaryzacji, natomiast w przypadku większości kondensatorów elektrolitycznych polaryzacja ma znaczenie i trzeba zawsze na nią zwracać uwagę podczas montażu.

Cewka jest elementem inercyjnym – gromadzi energię w wytwarzanym polu magnetycznym. W połączeniu z kondensatorem tworzy obwód rezonansowy (jeden z fundamentalnych obwodów elektronicznych).

Cewki zasilane prądem stałym, zwane elektromagnesami, są wykorzystywane do wytwarzania pola magnetycznego lub jego kompensacji, na przykład przy rozmagnesowaniu i pomiarach pola magnetycznego.

$$u = \frac{1}{C} \frac{di}{dt} \quad i = \frac{1}{L} \int_0^T U dt + io$$

Kondensator charakteryzuje się tym, że (dla sygnałów sinusoidalnych) napięcie jest opóźnione w fazie względem prądu o kąt $\pi/2$. Z tego względu impedancja kondensatora jest liczbą zespoloną i opisana jest wzorem:

$$Z = \frac{1}{j\omega C} = \frac{-j}{\omega C} = \frac{-j}{2\pi f C}$$

gdzie:

ω – pulsacja,

f – częstotliwość (w hercach),

j – jednostka urojona.

C – pojemność kondensatora

Reaktancja pojemnościowa wyraża się wzorem:

$$X_C = \frac{-1}{\omega C} = \frac{-1}{2\pi f C}$$

3. Omówić instrukcje sterujące w programowaniu strukturalnym.

Proszę wymienić i omówić zastosowanie wszystkich instrukcji sterujących w języku C i C++, oraz zaproponować jak można dokonać optymalizacji ich przetwarzania. C-01

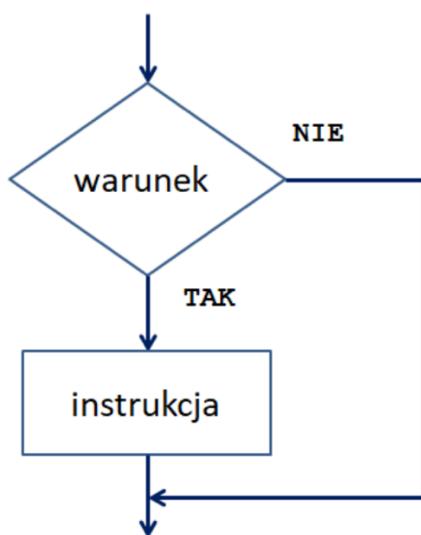
Instrukcje sterujące służą do sterowania przebiegiem programu. Dzięki nim są podejmowane decyzje o wykonaniu tych czy innych instrukcji programie. Decyzje te zależą od wyniku sprawdzenia określonych warunków, czyli od ustalenia, czy warunek jest prawdziwy czy nie.

Instrukcje sterujące w języku C/C++:

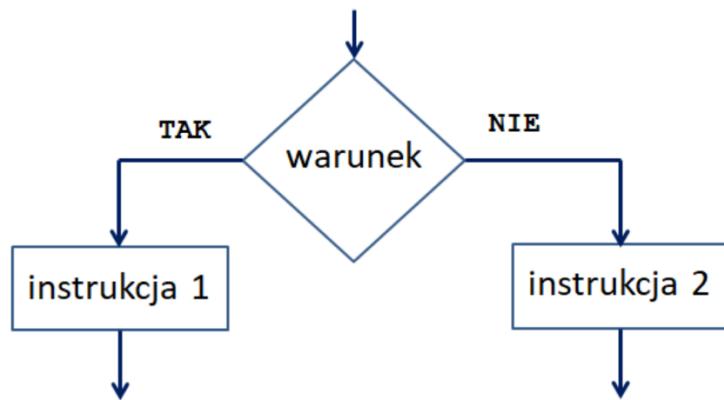
- instrukcja warunkowa *if*,
- instrukcja *switch*,
- pętla iteracyjna *while*,
- pętla iteracyjna *do ...while*,
- pętla iteracyjna *for*.

Instrukcja warunkowa – if.

Umożliwia kontrolę wykonywania poszczególnych instrukcji w obrębie programu w zależności od spełnienia warunku.



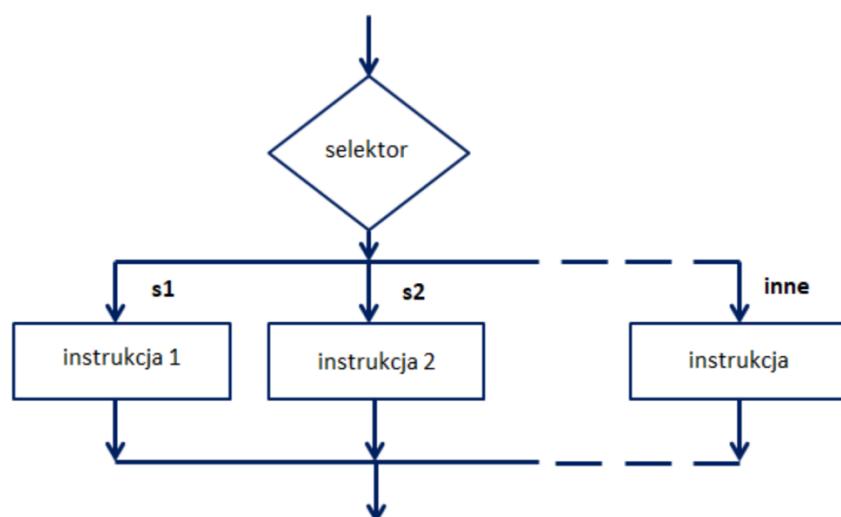
Rysunek 1 Schemat instrukcji warunkowej *if(niepełnej)*



Rysunek 2 Schemat instrukcji warunkowej if(pełnej)

Instrukcja switch

Służy do podejmowania wielowariantowych decyzji.



Rysunek 3 Schemat blokowy instrukcji switch

```

switch (selektor)
{
    case s1:
        instrukcja 1;
        break;
    case s2:
        instrukcja 2;
        break;
    .....
    default:
        instrukcja;
        break;
}

```

Rysunek 4 Schemat implementacji instrukcji switch

Selektor jest to zmienna, która może przyjmować różne wartości.

Opis działania instrukcji:

Jeśli wartość selektora odpowiada wartości podanej w jednej z etykiet case(s1, s2,...), wtedy instrukcje są wykonywane począwszy od tej etykiety.

Wykonanie kończy się po napotkaniu instrukcji break – następuje wyjście z instrukcji switch. Jeśli wartość selektora nie zgadza się z żadną z wartości podanych przy etykietach case to wykonają się instrukcje umieszczone po etykiecie default (może się znajdować w dowolnym miejscu instrukcji switch, ale zazwyczaj podaje się ją na końcu).

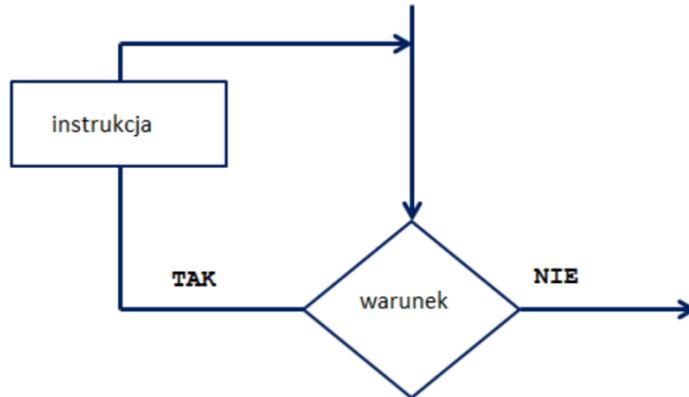
Instrukcja break powoduje natychmiastowe przerwanie instrukcji . Jeśli mamy do czynienia z kilkoma pętlami zagnieżdzonymi to instrukcja break powoduje przerwanie tylko tej pętli, w której bezpośrednio tkwi.

Pętla iteracyjna while

Pętla to konstrukcja programowania, która umożliwia cykliczne wykonywanie ciągu instrukcji określona liczbę razy, do momentu zajścia pewnych warunków, dla każdego elementu kolekcji lub w nieskończoność.

W języku C++ do zbudowania pętli wykorzystuje się następujące instrukcje:

- while – wykorzystywana jeśli liczba powtórzeń ma zależeć od warunku pętli;
- do ...while – wykorzystuje się jeśli dany blok instrukcji musi być wykonany przynajmniej raz;
- for – wykorzystuje się gdy konieczne jest określenie liczby powtórzeń danego bloku instrukcji.



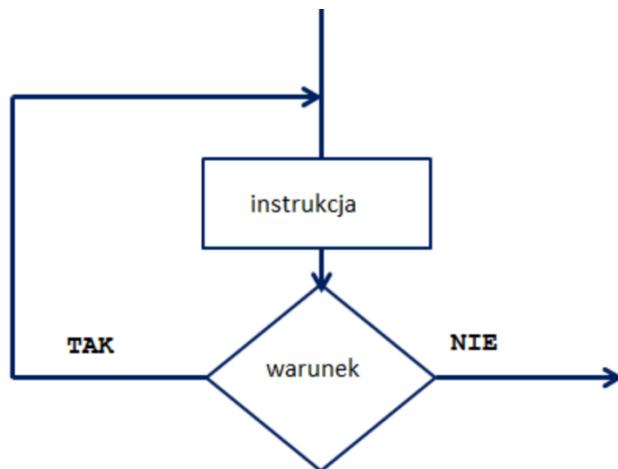
Rysunek 5 Schemat blokowy pętli while

Opis działania pętli:

Tak jak w przypadku instrukcji warunkowej, najpierw oblicza się wartość warunku. Jeśli wynik jest zerowy, to następuje wyjście z pętli. Jeśli wynik jest wartością niezerową, to jest wykonywana instrukcja (blok instrukcji), a następnie ponownie jest sprawdzana wartość warunku. Pętla zakończy działanie jeśli warunek nie zostanie spełniony – i to jest jedyny powód przerwania jej działania.

Uwaga: Pewnym niebezpieczeństwem jest warunek, który będzie zawsze prawdziwy – pętla może działać w nieskończoność.

Pętla do ...while

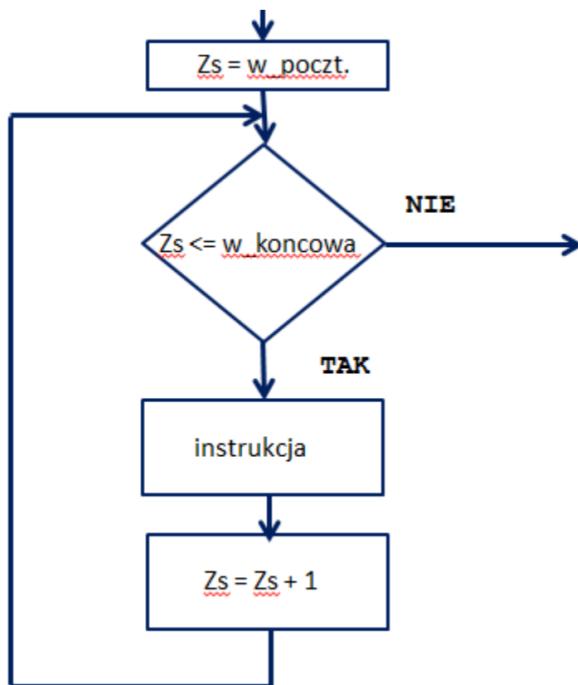


Rysunek 6 Schemat blokowy pętli do ...while

Instrukcja, która znajduje się wewnętrz pętli zostanie wykonana przed sprawdzeniem warunku. Po wykonaniu pierwszy raz instrukcji zostanie sprawdzony warunek i jeśli będzie spełniony pętla będzie wykonywać kolejne powtórzenia instrukcji , ale jeśli nie będzie spełniony pętla się zakończy.

Pętla iteracyjna for

Pętla for służy do obsługi pętli o określonej liczbie powtórzeń.



Rysunek 7 Schemat blokowy instrukcji for

Opis działania pętli:

Instrukcja ma trzy argumenty:

- $Zs = w_{pocz}$ – określa wartość początkową zmiennej sterującej pętli - Zs
- $Zs \leq w_{koncowa}$ – wyznacza wartość końcową zmiennej sterującej
- $Zs = Zs + 1$ – sposób zmiany wartości zmiennej sterującej (inkrementacja)

Optymalizacja powyższych instrukcji sterujących może polegać na:

- skróceniu kodu jeżeli to możliwe,
- stosowanie zmiennych,
- wykluczanie z głównej pętli niepotrzebnych funkcji,
- rozszczepienie pętli poprawia lokalizację odniesienia,
- odwrócenie działania pętli w zależności od wyznaczonych celów,
- oddzielne wykonanie pojedynczej iteracji, zamiast wyznaczać dodatkowe warunki w pętli,
- odpowiednie wykonanie pętli do indeksów tablicy.

(C-01) Omówić instrukcje sterujące w programowaniu strukturalnym. Proszę wymienić i omówić zastosowanie wszystkich instrukcji sterujących w języku C i C++, oraz zaproponować jak można dokonać optymalizacji ich przetwarzania.

- **Instrukcja warunkowa If**

Instrukcja warunkowa if umożliwia uzależnienie wykonania instrukcji od spełnienia pewnego warunku. Warunek ten jest reprezentowany przez wyrażenie umieszczone po słowie if. Wyrażenie to przybiera wartość logiczną prawda lub wartość fałsz (wszystkie wartości różne od 0 są w języku C traktowane jako prawda, wartość 0 jako fałsz). Instrukcja ta ma następującą postać: if (wyrażenie) akcja; Wyrażenie musi być zawarte w nawiasach. Aby wykonać if, najpierw oblicza się wyrażenie i określa się czy jest prawdziwe czy też fałszywe. Jeżeli wyrażenie jest prawdziwe, zostaje wykonana akcja i sterowanie przechodzi do instrukcji programu umieszczonej po akcji. Jeżeli wyrażenie jest fałszywe, akcja zostaje pominięta i następuje przejście do instrukcji umieszczonej po akcji. Część if określona jako akcja składa się z pojedynczej instrukcji lub instrukcji złożonej (pewnej liczby instrukcji umieszczonych w nawiasach {})

- **Pętla while**

Instrukcja while jest instrukcją iteracyjną, umożliwiającą powtarzanie pewnego ciągu operacji. Instrukcja while ma następującą postać: while (wyrażenie) akcja W instrukcji while najpierw określa się czy wyrażenie ma wartość logiczną prawda czy też fałsz. C wymaga by wyrażenie było zawarte w nawiasach. Jeżeli wyrażenie ma wartość logiczną prawda, wykonywana jest akcja i następuje powrót do obliczania wyrażenia. Wyrażenie jest ponownie testowane i jeżeli dalej jest prawdziwe, znowu wykonywana jest akcja i następuje kolejny powrót do obliczania wyrażenia. Jeżeli jednak przy którymś obliczaniu wyrażenia, stwierdzone zostanie, że wyrażenie ma wartość logiczną fałsz, następuje natychmiastowe przejście do instrukcji znajdującej się po instrukcji while. Akcja może się składać z jednej instrukcji lub też szeregu instrukcji zawartych w

nawiasach { }. Użycie tych nawiasów jest konieczne, jeśli akcja obejmuje więcej niż jedną instrukcję.

- **Pętla do while**

Instrukcja do while jest podobna do instrukcji while, z tą różnicą, że wyrażenie kontrolujące pętlę jest na końcu pętli. Cała pętla wykonywana jest przynajmniej raz. Do akcji while (wyrażenie); Przy wykonywaniu pętli do while wykonywana jest najpierw akcja, a następnie wykonywane jest sprawdzenie, czy wyrażenie wartość logiczną prawda też nie. Jeżeli ma wartość prawda, następuje powrót do początku pętli. Jeżeli w którymś momencie wyrażenie przybiera wartość fałsz, wykonywana jest pierwsza instrukcja występująca po pętli. Część pętli oznaczona jako akcja może być jedną instrukcją lub też grupą instrukcji zamkniętą w nawiasy { }, czyli instrukcją złożoną. Instrukcja do while jest użyteczna wtedy, gdy test kontynuacji pętli w sposób naturalny znajduje się na końcu pętli. Przykładem takiej sytuacji może być weryfikacja formatu wartości wprowadzanej przez użytkownika z klawiatury. Po wprowadzeniu ciągu znaków odbywa się sprawdzanie formatu. Jeżeli format jest niedopuszczalny, użytkownik zachęcany jest do wprowadzenia innego ciągu znaków albo też, jeśli format jest poprawny, a podana wartość jest nieprawidłowa, wprowadzenia innej wartości.

- **Switch**

Instrukcja switch jest instrukcją wyboru wielowariantowego, jest stosowana, gdy warunki wyboru mają wartość całkowitą, instrukcja ta może zastąpić wielokrotne if else.

Wyrażenie po słowie switch, zwane wyrażeniem wyboru lub selektorem, musi przyjmować wartości całkowite. Etykiety są całkowitymi wartościami stałymi lub wyrażeniami stałymi. Jeżeli podczas wykonywania instrukcji switch jedna z etykiet ma wartość równą wartości selektora (oznaczonego jako wyrazenieCalkowite), to wykonanie instrukcji switch rozpoczyna się od wykonania instrukcji znajdującej się przy tej etykiecie. Jeżeli w instrukcji switch występuje słowo default (ang. domyślny), instrukcje umieszczone po słowie default są wykonywane, gdy żadna z etykiet nie ma wartości równej selektorowi. Etyketa default jest opcjonalna i jeżeli nie

występuje, to następuje przejście do pierwszej instrukcji po instrukcji switch. Jeżeli chce się uzyskać wykonanie tylko jednej konkretnej instrukcji, należy po każdym wariantie umieścić słowo break wraz ze średnikiem. Spowoduje to natychmiastowe przerwanie wykonywania instrukcji switch po zrealizowaniu instrukcji związanej z danym wariantem.

- **Pętla for**

Instrukcja for, zwana też pętlą for, jest instrukcją iteracyjną podobnie jak instrukcje while i do while. Stosowana jest ona do wielokrotnego powtarzania pewnego segmentu kodu, głównie wtedy gdy ilość powtórzeń jest znana.

Wyrażenie wyr1 jest stosowane do inicjalizacji pętli, wyrażenie wyr2 jest używane do testowania warunku kontynuacji pętli, wyrażenie wyr3 służy do aktualniania pętli. Akcja stanowi treść (ciało) pętli.

```
for(wyr1; wyr2; wyr3){ Akcja;}
```

- **GoTo**

Instrukcja goto to tzw. instrukcja skoku powodująca przekazanie bezwarunkowe przekazanie sterowania do punktu w programie opatrzonego etykietą. Etykieta jest identyfikatorem i zakończona jest dwukropkiem. Etykietę można umieścić przed każdą instrukcją w funkcji, w której występuje instrukcja goto.

Stosowanie instrukcji goto nie jest zalecane, gdyż zwykle jej wielokrotne użycie powoduje, iż program staje się mniej czytelny jak również trudniejszy do modyfikacji. Istnieją jednak sytuacje, gdzie instrukcja goto jest wygodnym rozwiązaniem.

- **Break**

Instrukcja break kończy wykonywanie najbliższej otaczającej pętli lub instrukcji warunkowej, w której się pojawią. Jeśli po końcu przerwanej instrukcji występuje kolejna, sterowanie przechodzi do niej.

- **Continue**

Instrukcja continue jest podobna do instrukcji break w tym sensie, że również przerywa działanie pętli, jednak pętla nie ulega zakończeniu, lecz następuje przeskok do wykonania wyrażeń sterujących pętlą.

Dla pętli while i pętli do while następuje natychmiastowe przejście do obliczenia wyrażenia sterującego pętlą, czyli dla pętli while na początek pętli, a dla pętli do while na koniec pętli.

Omówić podstawowe terminy programowania obiektowego. Proszę zdefiniować pojęcia: Klasa, instancja, metoda, konstruktor i akcesor, oraz wytłumaczyć zasadę hermetyzacji(enkapsulacji) w odniesieniu do pojedynczej klasy i korzyści płynące z jej stosowania

Klasa i instancja

Jest bezpośrednim szablonem na stworzenie obiektu. Definiuje ona właściwości oraz metody (funkcje). Obiekt jest więc instancją określonej klasy. Możemy stworzyć dowolną liczbę instancji każdej klasy i każdy z nich może zachowywać inne wartości poszczególnych właściwości. Inaczej można powiedzieć na przykładzie budowy domu, że klasa jest elastycznym projektem z miejscem na dane takimi jak ilość drzwi, ilość okien, ilość pokoi czy przeznaczenie, natomiast obiekt jest konkretnym domem zbudowanym na podstawie projektu z określoną konkretną liczbą drzwi, okien, pokoi i przeznaczeniem.

Konstruktor

Konstruktory są specjalnymi metodami danej klasy, wywoływanymi podczas tworzenia jej instancji. Podstawowym zadaniem konstruktora jest więc utworzenie obiektu klasy. Możemy wyróżnić kilka rodzajów konstrktorów takich jak:

- Konstruktor domyślny – można wywołać go bez podawania parametrów,
- Konstruktor zwykły – można wywołać podając co najmniej jeden parametr,
- Konstruktor kopiący – można wywołać go podając jako argument referencję od innego obiektu, wówczas nowopowstały obiekt jest stworzony na podstawie referencji do innego obiektu

Z konstruktorami bezpośrednio związane są destruktory, służące do usunięcia obiektu z pamięci urządzenia. Pod względem funkcjonalnym jest to przeciwnieństwo konstruktora.

Hermetyzacja (enkapsulacja)

Jest to jedno z podstawowych założeń paradygmatu programowania obiektowego polegająca na ograniczeniu bezpośredniego dostępu do niektórych pól obiektów. Zapewnia, że obiekt nie może zmienić stanu wewnętrznego innych obiektów w nieoczekiwany sposób. Tylko własne metody, zwane akcesorami, są uprawnione do zmiany ich stanu. O dostępności pól i metod decydują specyfikatory dostępu takie jak między innymi:

- Public – składowe widoczne są globalnie, dla wszystkich,
- Private – składowe widoczne są tylko dla danej klasy,
- Protected – składowe widoczne tylko dla danej klasy oraz klas dziedziczących

W pojedynczej klasie dzięki hermetyzacji jesteśmy w stanie zabezpieczyć nasze zmienne przez nieuchcianymi zmianami i wewnętrzne metody służące tylko do operacji wewnątrz naszego obiektu przed wykorzystaniem ich w przypadkowy inny sposób w innym kontekście.

Akcesor

Zgodnie z zasadą hermetyzacji z poprzedniego slajdu pola klas nie powinny być bezpośrednio dostępne na zewnątrz. Służą nam do tego akcesory, czyli metody typu get i set z możliwością odczytu i edycji naszych pól.

[C-04]

Omówić typy interakcji w programowaniu obiektowym. Proszę wymienić i krótko omówić podstawowe typy relacji między klasami (z uwypukleniem różnic między nimi), oraz wyjaśnić jakie korzyści i problemy związane są z ich występowaniem w programie obiektowym.

Typy relacji pomiędzy klasami

1. Zależność (dependency)
2. Asocjacja (association)
3. Agregacja (aggregation)
4. Kompozycja (composition)
5. Dziedziczenie / uogólnienie (inheritance / generalization)

Zależność

- Najsłabsza forma związku pomiędzy klasami
- Krótkotrwałe i przelotne korzystanie z obiektów danej klasy
- Przykład – użycie obiektu klasy jako parametru operacji

Asocjacja

- Tymczasowe powiązanie obiektów klas
- Brak bezpośredniego wzajemnego posiadania instancji
- Brak kontroli czasu istnienia obiektów między sobą
- Realizacja za pomocą klas asocjacyjnych (parametry powiązania)
- Wiązanie instancji na poziomie użycia klas (zamiast ich definicji) + reużywalność

Agregacja

- Relacja silniejsza niż typowa asocjacja
- Powiązanie typu „has a” (posiadanie)
- Istnienie właściciela (container) i obiektu podległego (contained object)
- Możliwość bycia posiadanym przez wielu właścicieli (czas istnienia)
- Obiekty podległe mogą istnieć samodzielnie

Kompozycja

- Zwana również agregacją całkowitą, relacja typu „is part of”
- Życie obiektu podrzędnego jest całkowicie kontrolowane przez właściciela
- Obiekt podrządzny nie może istnieć samodzielnie (z reguły nie ma to sensu)
- Najsilniejsza relacja pomiędzy klasami z różnych hierarchii

Dziedziczenie

- Najsilniejsza ze wszystkich możliwych relacji – „is a”
- Tworzenie hierarchii klas, od ogólnych do szczegółowych
- Eliminuje duplikowanie kodu w klasach o pokrywającej się funkcjonalności
- Polimorfizm – możliwość zastąpienia klasy bazowej, wyspecjalizowaną klasą pochodną
- Istnienie problemu diamentowego – tylko przy dziedziczeniu wielobazowym

Wskaźnik i referencja

- Wskaźnik to zmienna przechowująca adres pamięci.
- Przed użyciem wskaźnika musi zostać on zainicjalizowany poprzez pobranie adresu istniejącej zmiennej lub dynamiczną alokacją pamięci.
- Wskaźnik może mieć postać generyczną (void), co pozwala na wskazywanie na określony obszar pamięci bez ustalania typu wskaźnika.
- Wskaźnik może zostać zainicjalizowany jako pusty za pomocą nullptr.
- Referencja w swym działaniu przypomina wskaźnik. Różnica polega jednak na tym, że do referencji można przypisać adres tylko raz, a jej dalsze używanie niczym się nie różni od używania zwykłej zmiennej (nie trzeba używać operatorów adresowania i dereferencji).
- Referencja musi być inicjalizowana podczas deklaracji.

Operatory adresowania i dereferencji

- Operator adresowania – operator pozwala na pobranie adresu obiektu na jaki ma wskazywać wskaźnik, symbolem tego operatora jest &
- Operator dereferencji – inaczej operator wyłuskania wartości, pozwala on na dostanie się do wartości zmiennej znajdującej się pod adresem przechowywanym przez wskaźnik, symbolem tego operatora jest *

```
int zmieniona = 1;  
int *wskaźnik;  
wskaźnik = &zmieniona;  
*wskaźnik = 2;  
std::cout << *wskaźnik;
```

Wskaźnik - void

Wskaźnik do dowolnego typu może zostać przypisany do typu void*. Wskaźnik void* może być przypisany do innego wskaźnika typu void*. Dozwolone są porównania między tymi wskaźnikami (==, !=). Inne operacje są niedozwolone. Wskaźnik do typu void reprezentuje wskaźnik do fragmentu pamięci, ale bez znajomości typu (wskaźnik do „surowej” pamięci).

```

void f(int* pi)
{
    void* pv = pi;    // niejawna konwersja z int* do void*
    *pv;    // błąd!
    pv++;   // błąd! nieznany rozmiar
    int* ptr2pi = static_cast<int*>(pv);
    (*ptr2pi)++;
}

```

Wkaźnik – arytmetyka wskaźników

Dzięki wskaźnikom możliwe jest uzyskiwanie wielu wyników z funkcji.

- Wskaźniki mogą udostępniać dostęp do dowolnego elementu tablicy.
- Na wskaźnikach można wykonywać operacje arytmetyczne.
- Jeśli do wskaźnika dodamy wartość całkowitą n, przesunie się o n elementów; operator ++ umożliwia przesunięcie wskaźnika o jeden element.
- Wyrażenie będące sumą wskaźnika i wartości n wskazuje n-ty element za elementem wskazywanym przez wskaźnik.
- Jeśli odejmiemy od siebie wartości dwóch wskaźników, uzyskamy liczbę całkowitą reprezentującą odległość między elementami wskazywanymi przez te wskaźniki.

```

int values[10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}; // tablica 10 elementów typu int
for (int* p = values; p < values + 10; ++p)
{
    std::cout << "index: " << p - values << " value: " << *p << std::endl;
}
int x = values[5];
*(values + 5) = 10;

```

index: 0 value: 0	index: 1 value: 1	index: 2 value: 2	index: 3 value: 3
index: 4 value: 4	index: 5 value: 5	index: 6 value: 6	index: 7 value: 7
index: 8 value: 8	index: 9 value: 9		

Wkaźnik – funkcje

Wskaźnik funkcyjny – wskaźnik na adres funkcji w pamięci

```
int dodaj(int a, int b){return a + b;}
int odejmij(int a, int b){return a - b;}
int (*foo)(int, int); // deklaracja wskaźnika do funkcji przyjmującej dwa argumenty
int main(void)
{
    int wynik;
    foo = &dodaj; // zapisanie adresu funkcji dodaj() do wskaźnika
    wynik = foo(5, 4); // wywołanie funkcji zapisanej do wskaźnika
    printf("Wynik dodawania = %d\n", wynik);
    foo = &odejmij;
    wynik = foo(5, 4);
    printf("Wynik odejmowania = %d\n", wynik);
    return 0;
}
```

Wkaźnik – pamięć dynamiczna

W języku c++ do alokacji i dealokacji pamięci służą operatory **new** i **delete**.

W języku c do alokacji pamięci służą funkcje **malloc**, **calloc** i **realloc**, a do jej zwalniania funkcja **free**.

- funkcja **malloc** (ang. memory allocation) przydziela obszar w pamięci wolnej,
- za pomocą funkcji, **calloc**, możemy zaalokować obszar pamięci i od razu zainicjować go zerami,
- funkcja **realloc** zmienia rozmiar bloku pamięci dynamicznej wcześniejszej zaalokowanego.

```
int size = 5;  
int* m = new int[size];  
for (int i = 0; i < size; ++i)  
    m[i] = 2 * i;  
delete[] m;  
  
int size = 5;  
int* m = (int*)malloc(size * sizeof(int));  
for (int i = 0; i < size; ++i)  
    m[i] = 2 * i;  
void free(void* m);
```

Wkaźnik – const

Przy wykorzystywaniu właściwości `const` dla wskaźnika, w grę wchodzą dwa obiekty: sam wskaźnik i wskazywany przez niego obiekt. Słowo kluczowe `const` stojące przed deklaracją wskaźnika odnosi się do obiektu wskazywanego przez ten wskaźnik. Aby jako stały zadeklarować sam wskaźnik, należy użyć operatora `*const` zamiast `*`.

```
char *const c; // stały wskaźnik do char – niezmienny adres  
char const* c; // wskaźnik do const char – niezmienna wartość char  
const char* c; // wskaźnik do const char – niezmienna wartość char
```

(C-05): Omówić operacje wskaźnikowe w C/C++.

Proszę wyjaśnić co to jest wskaźnik, jakie operacje można na nim wykonać i czym różni się od referencji, oraz omówić zastosowania wskaźników i referencji w C++.

Czym jest wskaźnik?

- **Wskaźnikiem** nazywamy specjalną zmienną, która jest przeznaczona do przechowywania (wskażania) adresu innej zmiennej lub dowolnego obszaru w pamięci.
- Najczęściej używane są wskaźniki **zdefiniowane** zawierające adres innej zmiennej. Taki wskaźnik posiada informację o adresie i typie zmiennej. Ogólna postać definicji takiego wskaźnika:

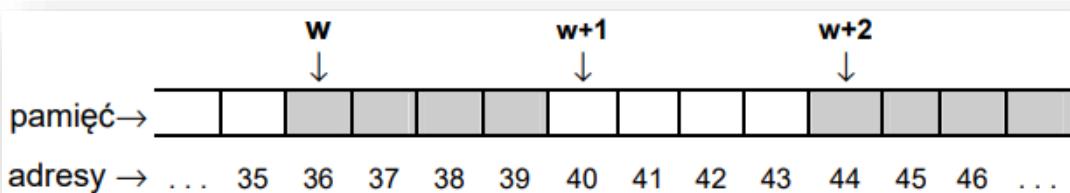
typ_danych * id_wskaznika;

- Można również korzystać ze wskaźników **niezdefiniowanych** (anonimowych). Taki wskaźnik zawiera tylko informację o adresie początku obszaru pamięci (bez określenia typu wskazywanych danych). Definicja takiego wskaźnika ma postać:

void * id_wskaznika;

Operacje wskaźnikowe

- Na wskaźnikach mogą być wykonywane następujące operacje:
 - **Przypisania (=)**
 - **wsk = wskaźnik_zmienne_lub_obszaru_pamieci**
 - **Operacje porównania (==, !=, <, >, <=, >=)**
 - **wsk1 == wsk2** // sprawdzenie czy zmienne zawierają te same adresy
 - **wsk1 < wsk2** // sprawdzenie czy zmienność wsk1 zawiera adres mniejszy od wsk2
 - **Operacja odejmowania wskaźników tego samego typu**
(tzw. Wyznaczanie odległości między dwoma adresami w pamięci):
 - odległość: **N * sizeof(typ_elementu_wskazywanego)**
 - **Operacje powiększania lub pomniejszania wskaźnika o liczbę całkowitą (+, -, ++, --, +=, -=): tylko dla wskaźników zdefiniowanych (!)**
 - Przykład dla **int *w** – powiększenie wskaźnika o wartość N powoduje przesunięcie bajtów w kierunku rosnących adresów:



Referencja a wskaźnik

- Referencja to wartość, która zawiera informacje o położeniu innej wartości w pamięci. Referencja nie jest kopią zmiennej, ale tą samą zmienną pod inną nazwą. W C++ używana przez operator adresu (referencji) &.
 - Przykład: **typ *wskaźnik = &zmienna**
- W odróżnieniu od wskaźników, zarządzanie referencjami realizowane jest wyłącznie przez kompilator lub interpreter, a programista nie posiada żadnych informacji o konkretnym sposobie implementacji referencji.
- We wskaźnikach jest możliwość **ponownego przypisania** zmiennej, natomiast raz przypisana referencja odnosi się do tej samej zmiennej.
- Wskaźnik ma swój **własny adres** w pamięci, a referencja ma ten sam adres, jaki posiada zmienność, do której jest ona przypisana.
- Wskaźnik **może być nullem (nullptr)**, natomiast referencja nigdy.

Zastosowanie wskaźników i referencji w C++

- Najczęstszymi zastosowaniami wskaźników są m. in.:
 - *Tablice wskaźnikowe ze względu na ulepszoną pracę z tablicami pod względem wydajności*
 - *Parametry w funkcjach mogą zmieniać wartość przesyłanych do nich argumentów*
 - *Dostęp do specjalnych komórek pamięci*
 - *Dynamiczna rezerwacja obszarów pamięci*
 - *Praca na obiektach dynamicznych i implementacja struktur danych tj. lista wskaźnikowa, drzewa*
- Głównymi zastosowaniami referencji są:
 - *Działanie jako funkcjonalny parametr podczas przekazywania przez referencję.*
 - *Podczas przekazywania przez referencję, funkcja działa na oryginalnej kopii, a nie klonie zmiennej.*
 - *Zmiany dokonane wewnętrz funkcji mają odzwierciedlenie poza nią.*

- 3. Omówić złożoność obliczeniową algorytmów.
Proszę wyjaśnić co to jest złożoność przestrzenna i czasowa algorytmów, oraz wymienić i omówić klasy złożoności algorytmów.

Złożoność obliczeniowa algorytmu określa, jak wydajny jest algorytm, ile musi on wykonać operacji w zależności ilości danych oraz ile potrzebuje do tego pamięci. Często zdarza się, że dany problem algorytmiczny można rozwiązać kilkoma metodami, czyli algorytmami o różnej złożoności obliczeniowej.

- Złożoność obliczeniową możemy podzielić na:
- Złożoność przestrzenna, inaczej zwana złożonością pamięciową algorytmu wyznaczana jest na podstawie ilości potrzebnej pamięci do rozwiązania problemu. Na przykład, jeśli dla danych wejściowych złożonych z n elementów nasz algorytm będzie zużywał dwa razy tyle pamięci, ile zużyłby na te n elementów, to mówimy, że złożoność pamięciowa wynosi $2n$, natomiast, jeżeli do rozwiązania problemu potrzebuje stałą niewielką ilość komórek pamięci, to mówimy, że taka złożoność jest stała. Oczywiście, im mniejsza złożoność, tym lepiej.
 - Przyjętą miarą złożoności czasowej jest liczba operacji podstawowych w zależności od rozmiaru wejścia. Pomiar rzeczywistego czasu zegarowego jest mało użyteczny ze względu na silną zależność od sposobu realizacji algorytmu, użytego kompilatora oraz maszyny, na której algorytm wykonujemy. Dlatego w charakterze czasu wykonania rozpatruje się zwykle liczbę operacji podstawowych (dominujących). Operacjami podstawowymi mogą być na przykład: podstawienie, porównanie lub prosta operacja arytmetyczna.



Klasy złożoności algorytmów

Notacja dużego O

Aby odpowiedzieć na pytanie czym są klasy złożoności algorytmów, należy najpierw wspomnieć o notacji dużego O. **Notacja dużego**

O mówi nam o szybkości wykonania algorytmu. Przez szybkość mamy na myśli **ilość operacji**, którą komputer musi wykonać, **niekoniecznie zaś sam czas wykonania**. Niemniej jednak, spadek szybkości lubi iść w parze ze wzrostem czasu wykonania algorytmów.

- Im mniejsza złożoność algorytmu, tym jest on bardziej wydajny. Poniżej wypisane są klasy złożoności od najwydajniejszej do najwolniejszej oraz szacowana liczba operacji, którą algorytm wykona dla 100 elementów:

1. $O(1)$ - stała (od jednego do kilku operacji),
2. $O(\log n)$ - logarytmiczna (7 operacji),
3. $O(n)$ - liniowa (100 operacji),
4. $O(n \log n)$ - n razy logarytm z n (700 operacji),
5. $O(n^2)$ - kwadratowa (10 000 operacji),
6. $O(n^3)$ - sześcienna (1000 000 operacji),
7. $O(n!)$ - n silnia ($1*2*3*4*...*100$ operacji),
8. $O(2^n)$ - wykładnicza (2^{100} operacji).

Klasy złożoności algorytmów – inny podział

- Klasy złożoności algorytmów – Są to tak zwane klasy problemów, podział ten grupuje te zagadnienia do rozwiązymania których potrzebna jest podobna ilość zasobów. Zamiennie z określeniem klasa problemu można stosować pojęcia klasa złożoności obliczeniowej
- Wyróżniamy następujące klasy:
 - P – polynominal - Jest to klasa problemów, dla których istnieje wielomianowy algorytm rozwiązania na Deterministycznej Maszynie Turinga (w skrócie DTM). Rozwiążanie takiego problemu jest możliwe w wielomianowym czasie. Przykładem takich

problemów jest sortowanie elementów oraz algorytm Euklidesa do znajdowania Największego Wspólnego Dzielnika

- NP - nondeterministic polynominal - Wielomianowy algorytm rozwiązania istnieje jedynie na Niedeterministycznej Maszynie Turinga, a rozwiąznie nie jest możliwe w czasie wielomianowym. Różnica pomiędzy problemami P i NP polega na tym, że w przypadku P znalezienie rozwiązania ma mieć złożoność wielomianową, podczas gdy dla NP sprawdzenie podanego z zewnątrz rozwiązania ma mieć taką złożoność.
- NP – zupełne - Problemy te zawierają się w klasie problemów NP. Problemy NP-zupełne można traktować jako najtrudniejsze problemy klasy NP (z punktu widzenia wielomianowej rozwiązywalności).
- Silne NP – zupełne
Problemy nierostrzygnięte

3. Omówić struktury danych. Proszę wymienić i krótko scharakteryzować poznane struktury danych i wyjaśnić jaką jest podstawowa różnica pomiędzy strukturami statycznymi i dynamicznymi.

Struktury danych jest to sposób przechowywania danych w pamięci komputera. Na strukturach danych operują algorytmy. Każdego rodzaju program oraz algorytm operuje na różnego rodzaju danych. Wydajność tych systemów zależy głównie od rodzaju przetwarzania danych oraz wykorzystanych do tego procesu ich struktur. Najprościej mówiąc struktura danych jest pojemnikiem na określone dane, ma on za zadanie gromadzenie tych danych oraz układanie ich w odpowiedni sposób.

Podstawowe struktury danych:

- **Rekord** - jest jedną z prostszych struktur danych, która świetnie nadaje się do prezentowania informacji o jednym obiekcie. Ponieważ w trakcie działania algorytmu nie zmienia on swego rozmiaru ani struktury, mówimy, że jest to **statyczna struktura danych**.

```
1 {
2 name:      "Gdańsk",
3 province:  "Pomorskie",
4 population: 500000
5 }
```

W Javie możemy w tym celu wykorzystać klasę:

```
1 public class City {
2     private String name;
3     private String province;
4     private long population;
5     public City(String name, String province, long
6     population) {
7         this.name = name;
8         this.province = province;
9         this.population = population;
10    }
11    City gdansk = new City("Gdańsk", "Pomorskie", 500000);
```

- **Tablica** - Tablice bardzo dobrze sprawdzają się w sytuacji, gdy mamy pewien zbiór danych tego samego typu, który nie będzie modyfikowany podczas działania algorytmu. Rozmiar tego typu struktur musimy jednak określić już na samym początku, podczas podawania wstępnych parametrów.

Struktury danych – Tablica

```
1 City[] cities = new City[100];  
2 cities[0] = new City("Gdańsk", "Pomorskie", 500000);
```

- **Lista** - Lista, podobnie jak tablica, przeznaczona jest do przechowywania większej ilości danych. Z tym wyjątkiem, że lista umożliwia łatwe dodawanie nowych elementów. Listę można postrzegać jako łańcuszek połączonych ze sobą danych. Wiedząc, gdzie znajduje się początek, możemy znaleźć kolejne ogniska takiej struktury.

Istnieją dwie popularne realizacje struktury listy: tablicowa oraz wskaźnikowa. W Javie obie implementują interfejs *List*, zatem w wielu przypadkach mogą być one używane zamiennie. Na ostateczną decyzję, którą implementację wybrać wpływ będą miały poszczególne parametry dostosowane do konkretnego projektu i naszych potrzeb (a czasem również preferencji).

Lista tablicowa – ArrayList

Już sama nazwa wskazuje, że do implementacji tej listy została wykorzystana tablica. Jest to tak naprawdę zwykła tablica wyposażona w dodatkowe funkcje, które ukrywają złożoną logikę i umożliwiają np. usunięcie elementu ze środka listy, czy zwiększenie jej rozmiaru.

```
1 List list = new ArrayList();  
2 list.add("A");  
3 list.add("B");  
4 String b = (String) list.get(1);
```

Lista wskaźnikowa – LinkedList

Implementacja wskaźnikowa wymusza dodanie do każdego elementu listy nowej wartości: **wskaźnika, czyli referencji**, dzięki czemu zrealizowana jest lokalna nawigacja w liście. W przeciwieństwie do tablicy logiczna kolejność elementów w liście wskaźnikowej może być inna niż jej fizyczne ułożenie w pamięci.

```
1 List list = new java.util.LinkedList();
2 list.add("A");
3 list.add("B");
4 String b = (String) list.get(1);
```

Lista jednokierunkowa:

Najprostszą formą listy jest **lista jednokierunkowa**, w której każdy element zawiera referencję do kolejnego (lub poprzedniego) komponentu w liście lub do wartości pustej (`NULL`), jeżeli jest to ostatni jej element.

Lista dwukierunkowa:

jest rozwinięciem listy jednokierunkowej. W tym wypadku poszczególny rekord w liście zawiera referencję do poprzedniego i kolejnego elementu. Dzięki takiej implementacji można swobodnie poruszać się po liście w górę i w dół.

Lista cykliczna:

W tej liście ostatni element jest powiązany z pierwszym, a pierwszy – z ostatnim. Dzięki temu tworzy się zamknięty cykl i możemy poruszać się wokoło. W związku z tym taka lista charakteryzuje się również brakiem początkowego wskaźnika (głowa listy – `HEAD`) oraz ostatniego (ogon listy – `TAIL`).

Stos (Stack) – LIFO

Stos nazywany jest również kolejką LIFO, czyli 'ostatni na wejściu, pierwszy na wyjściu' (z ang. Last In, First Out). Bardzo dobrze nadaje się do sytuacji, gdy chcemy przeprowadzać operacje najpierw na najnowszych danych, a dopiero później – na starszych. Najłatwiej wyobrazić sobie tę strukturę jako stos talerzy do pozmywania ułożonych jeden na drugim. Zaczynamy od obiektów znajdujących się na samej górze, jednak gdy ktoś w trakcie wykonywania przez nas czynności przyniesie jeszcze brudne naczynia, to wylądują one na samym szczycie stosu.

Natomiast – aby dostać się do elementów znajdujących się na samym dole – trzeba najpierw ściągnąć wszystkie inne.

Podstawowe operacje na stosie to:

push (obiekt) – dodanie obiektu na wierzch stosu;

pop () – ściągnięcie jednego elementu ze stosu i zwrócenie jego wartości.

```
1 Deque<String> stack = new java.util.LinkedList();
2 stack.push("A");
3 stack.push("B");
4 stack.pop();
5 String a = stack.pop();
```

Kolejka (Queue) – FIFO

Przeciwieństwem stosu LIFO jest **kolejka typu FIFO** (ang. *First In, First Out*), czyli 'pierwszy na wejściu, pierwszy na wyjściu'. W tym przypadku zaczynamy przetwarzanie danych od elementów, które pojawiły się jako pierwsze. Jest to odpowiednik zwykłej kolejki w sklepie.

```
1 java.util.Queue<String> queue = new
2 java.util.LinkedList();
3 queue.add("A");
4 queue.add("B");
```

Szczególną odmianą tej struktury danych jest **kolejka priorytetowa**. W tej implementacji konkretne rekordy dodatkowo posiadają przypisany priorytet, który wpływa na kolejność późniejszego pobierania elementów z listy. Najpierw przetwarzane są najpilniejsze elementy (o najwyższym priorytecie), a dopiero potem te mniej ważne.

Tego typu kolejki spotyka się przede wszystkim w systemach, gdzie przetwarzane są różnego rodzaju zadania i zależy nam na ich odpowiednim uporządkowaniu.

Zbiór (Set)

Zbiór to kolekcja elementów, która nie może zawierać duplikatów.

W Javie najczęściej wykorzystywane są dwie implementacje: *HashSet* oraz *TreeSet*.

```
1 Set<String> hashSet = new HashSet<>();
2 hashSet.add("c1");
3 hashSet.add("a1");
4 hashSet.add("b1");
5 hashSet.add("b1");
6 Set<String> treeSet = new TreeSet<>();
7 treeSet.add("c1");
8 treeSet.add("a1");
9 treeSet.add("b1");
10 treeSet.add("b1");
11 System.out.println(hashSet); // [a1, c1, b1]
12 System.out.println(treeSet); // [a1, b1, c1]
```

Główna różnica między tymi dwiema implementacjami polega na tym, że **TreeSet wyznacza kolejność elementów (domyślnie jest ona rosnąca)**, a **HashSet – nie**. W szczególnych przypadkach może się jednak zdarzyć, że HashSet również posortuje elementy rosnąco, jednak implementacja tego nie gwarantuje. Chcąc dodać do Set'a instancje swojej własnej klasy, musisz pamiętać o odpowiednim zaimplementowaniu metod: `hashCode` oraz `equals`. Jeżeli nie zachowasz założeń kontraktu między tymi metodami, możesz spodziewać się bardzo dziwnych błędów. Więcej na ten temat przeczytasz w podlinkowanym artykule.

Dodatkowo w przypadku TreeSet klasy przechowywanych obiektów muszą implementować interfejs: `java.lang.Comparable`.

Mapa (Map)

Mapa to specyficzna kolekcja, która przechowuje pary danych składające się z klucza oraz przypisanej wartości. Klucz w obrębie mapy musi być unikatowy, natomiast wartości mogą się powtarzać.

```
1 Map<String, Integer> map = new HashMap<>();
2 map.put("Gdańsk", 10);
3 map.put("Warszawa", 5);
4 map.put("Warszawa", 7);
5 System.out.println(map); // {Gdańsk=10, Warszawa=7}
```

Na powyższym przykładzie widać, że ponowne przypisanie wartości dla tego samego klucza powoduje nadpisanie poprzedniej wartości.

Różnica pomiędzy strukturami statycznymi i dynamicznymi.

Dynamiczne struktury danych w przeciwieństwie do statycznych (takich jak np. rekord) pozwalają na zmianę ich struktury już podczas działania algorytmu. Jest to bardzo wygodne, ponieważ na starcie bardzo rzadko kiedy wiemy ile i jakie dokładnie dane będziemy przetwarzać. Taka swoboda ma jednak swoją cenę – tego typu struktury danych są zwyczajnie bardziej skomplikowane. Dostęp do przechowywanych w nich danych odbywa się często przy pomocy specjalnych wskaźników, a modyfikacje struktury przy pomocy odpowiednich metod.

Przykłady struktur dynamicznych to np. stos, kolejka, lista jedno i dwukierunkowa, drzewo itp.

(C-08) Omówić struktury danych. Proszę wymienić i krótko scharakteryzować poznane struktury danych i wyjaśnić jaka jest podstawowa różnica pomiędzy strukturami statycznymi i dynamicznymi.

Statyczne struktury danych:

- **Struktura prosta** (typy proste) np. int, char, wskaźnik itp.
- **Tablica**

Struktura danych umożliwiająca przechowywanie N wartości tego samego typu. Tablica służy do przechowywania dużej ilości danych tego samego typu. Tablica, to jakby pudełko, podzielone na komórki. W każdej komórce tablicy można przechować jakąś wartość (liczbę tekst itp.). Ponieważ tą wartością może być również tablica - można więc tworzyć tablice wielowymiarowe. Podczas deklarowania tablic wielowymiarowych należy uważać na ilość zajmowanej przez nie pamięci. Komórki pamięci można numerować w dowolny, ale kolejny sposób.

- **Rekord (struct)**

Typ danych umożliwiający przechowywanie wartości różnego typu w spójnym obszarze pamięci

Dynamiczne struktury danych:

- **Lista**

Lista to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż można z łatwością dodawać, usuwać, modyfikować lub podmieniać poszczególne elementy. Listy są wobec tego zwykle implementowane z wykorzystaniem wskaźników, które wiążą ze sobą rekordy (obiekty, klasy), zawierające w sobie dane oraz wskaźniki lub indeksy wskazujące poprzedni oraz następny rekord w liście, za wyjątkiem elementu pierwszego i ostatniego, których odpowiednie wskaźniki nie są ustawione. Lista posiada dwa wyróżnione wskaźniki, zwane głową (head) i ogonem (tail), które

wskazują odpowiednio na początek i koniec listy, czyli pierwszy i ostatni element listy. Wszystkie operacje na liście, tj. dodawanie, usuwanie, modyfikowanie i podmienianie elementów, mogą być wykonywane w dowolnym miejscu listy.

Zbiór elementów tego samego typu ułożonych liniowo – może dynamicznie zmieniać rozmiar, pozwala na dostęp do poszczególnych elementów.

Operacje do wykonania na liście:

- Dodanie elementu na końcu
- Odczyt dowolnego elementu
- Usunięcie dowolnego elementu
- Znalezienie dowolnego elementu
- Wstawienie dowolnego elementu

- **Kolejka**

Kolejka to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż elementy można z dodawać tylko na jej końcu, a usuwać je tylko z jej początku. Kolejka to szczególny przypadek listy pojedynczo wiązanej, jednokierunkowej, ograniczający operacje na niej. Dodawanie nowego elementu na końcu kolejki wiąże się ze zmianą wskaźnika końca kolejki i dotychczas ostatniego elementu kolejki, ustawiając je na nowo dodanym elemencie. Usuwanie elementu z przodu kolejki powoduje przesunięcie wskaźnika początku kolejki na następny element.

Wiele elementów tego samego typu, zmienia rozmiar, dostęp do pierwszego elementu

- Inaczej to kolejka typu first in first out (FIFO)
- Pierwszy wchodzi -> pierwszy wychodzi

- Dodawanie wartości: każda dodawana wartość umieszczana jest na końcu struktury
- Usuwanie wartości: każda usuwana/pobierana wartość pochodzi z początku kolejki

- **Stos**

Stos to dynamiczna, liniowa struktura danych składająca się z elementów powiązanych ze sobą w taki sposób, iż elementy można dodawać lub usuwać tylko z góry stosu. Stos to szczególny przypadek Listy pojedynczo wiązanej, ograniczający operacje na niej. Powiązanie poszczególnych elementów w stosie jest jednokierunkowe. Stos posiada jeden wskaźnik sterujący, zwany górną głowicą, który wskazuje element na jego wierzchu. Tylko na tym elemencie można położyć kolejny albo go zdjąć ze stosu.

Wiele elementów tego samego typu, zmienia rozmiar, dostęp do ostatnio dodanego elementu.

- Inaczej to kolejka typu last in first out (LIFO)
- Dodawanie wartości: każda dodawana wartość umieszczana jest na początku
- Usuwanie wartości: każda usuwana/pobierana wartość pochodzi z początku kolejki

- **Drzewo/sterta**

Struktura hierarchiczna zbudowana z elementów zwanych węzłami, w których przechowywane są dane, łącznikiem węzłów są krawędzie. Każdy węzeł drzewa ma swojego rodzica (węzeł nadzędny) zwanego ojcem za wyjątkiem korzenia. Każdy węzeł ma synów (dzieci), którzy są węzłami podległymi względem danego węzła. Wyjątkiem są liście które nie mają synów. Wszystkie węzły między korzeniem a liśćmi są węzłami podległymi.

- **Graf**

Graf to struktura danych składająca się z wierzchołków i krawędzi, łączących dwa wierzchołki.

Grafem nieskierowanym nazywamy parę $G = (V, E)$, gdzie V jest niepustym i skończonym zbiorem wierzchołków (węzłów), a E jest dowolnym skończonym zbiorem krawędzi, rozpoczynających się w jednym a kończących się w drugim wierzchołku, reprezentowanych w postaci nieuporządkowanych par elementów zbioru V .

Grafem skierowanym nazywamy parę $G = (V, D)$, gdzie V jest niepustym i skończonym zbiorem wierzchołków, a D jest dowolnym skończonym zbiorem krawędzi skierowanych, reprezentowanych w postaci uporządkowanych par elementów zbioru V .

Grafem ważonym nazywamy trójkę $G = (V, E, w)$, gdzie $G = (V, E)$ jest grafem, a $w: E \rightarrow R$ jest funkcją określającą wagę krawędzi.

Dynamiczne struktury danych charakteryzują się zmiennością swoich struktur podczas procesu obliczeniowego. Zmiany te dotyczą przede wszystkim ich rozmiarów (ilości zajętej pamięci). Dla dynamicznych struktur danych kompilator nie może "z góry" przydzielić określonej ilości pamięci podczas dokonywanej translacji kodu, gdyż jej ilość może się zmieniać w trakcie wykonywania programu.

Statyczne struktury danych są powiązane z konkretnym miejscem i rozmiarem pamięci już w momencie komplikacji, a więc muszą być znane przed jej rozpoczęciem.

(C-10) Omówić algorytmy sortowania. Proszę wymienić i krótko scharakteryzować poznane algorytmy sortowania danych, oraz przedstawić ich słabe i mocne strony, w kontekście właściwości i stopnia uporządkowania danych.

Ze względu na złożoność pamięciową algorytmy sortujące dzielimy na dwie podstawowe grupy:

1. **Algorytmy sortujące w miejscu** (ang. in place) - wymagają stałej liczby dodatkowych struktur danych, która nie zależy od liczby elementów sortowanego zbioru danych (ani od ich wartości). Dodatkowa złożoność pamięciowa jest zatem klasy $O(1)$. Sortowanie odbywa się wewnątrz zbioru. Ma to bardzo istotne znaczenie w przypadku dużych zbiorów danych, gdyż mogłoby się okazać, iż posortowanie ich nie jest możliwe z uwagi na brak pamięci w systemie. Większość opisanych tu algorytmów sortuje w miejscu, co jest ich bardzo dużą zaletą.
2. **Algorytmy nie sortujące w miejscu** - wymagają zarezerwowania w pamięci dodatkowych obszarów, których wielkość jest zależona od liczby sortowanych elementów lub od ich wartości. Tego typu algorytmy są zwykle bardzo szybkie w działaniu, jednakże okupione to jest dodatkowymi wymaganiami na pamięć. Zatem w pewnych sytuacjach może się okazać, iż taki algorytm nie będzie w stanie posortować dużego zbioru danych, ponieważ system komputerowy nie posiada wystarczającej ilości pamięci RAM.

Algorytmy sortujące dzieli się również na dwie grupy:

1. **Algorytmy stabilne** - zachowują kolejność elementów równych. Oznacza to, iż elementy o tych samych wartościach będą występować w tej samej kolejności w zbiorze posortowanym, co w zbiorze nieposortowanym. Czasami ma to znaczenie, gdy sortujemy rekordy bazy danych i nie chcemy, aby rekordy o tym samym kluczu zmieniały względem siebie położenie.
2. **Algorytmy niestabilne** - kolejność wynikowa elementów równych jest nieokreślona (zwykle nie zostaje zachowana).

- **Sortowanie bąbelkowe** - Bubble Sort

Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Podstawową zaletą sortowania bąbelkowego jest to, że jest popularny i łatwy do wdrożenia. Ponadto w sortowaniu bąbelkowym elementy są zamieniane na miejscu bez dodatkowego tymczasowego przechowywania, więc zapotrzebowanie na miejsce jest minimalne. Główną wadą tego typu bąbelków jest to, że nie radzi sobie dobrze z listą zawierającą ogromną liczbę przedmiotów. Wynika to z tego, że sortowanie bąbelkowe wymaga $n \cdot n$ -kwaterowych kroków przetwarzania dla każdej liczby elementów do posortowania. Jako taki, sortowanie bąbelkowe nadaje się głównie do nauczania akademickiego, ale nie do rzeczywistych zastosowań.

Algorytm sortowania bąbelkowego przy porządkowaniu zbioru nieposortowanego ma klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie przez wybór** - Selection Sort

Idea algorytmu sortowania przez wybór jest bardzo prosta. Założymy, iż chcemy posortować zbiór liczbowy rosnąco. Zatem element najmniejszy powinien znaleźć się na pierwszej pozycji. Szukamy w zbiorze elementu najmniejszego i wymieniamy go z elementem na pierwszej pozycji. W ten sposób element najmniejszy znajdzie się na swojej docelowej pozycji.

W identyczny sposób postępujemy z resztą elementów należących do zbioru. Znów wyszukujemy element najmniejszy i zamieniamy go z elementem na drugiej pozycji. Otrzymamy dwa posortowane elementy. Procedurę kontynuujemy dla pozostałych elementów dotąd, aż wszystkie będą posortowane.

Główną zaletą sortowania selekcyjnego jest to, że działa dobrze na małej liście. Ponadto, ponieważ jest to algorytm sortowania na miejscu, nie jest wymagane dodatkowe tymczasowe miejsce do przechowywania poza tym, co jest potrzebne do przechowywania oryginalnej listy. Podstawową wadą tego rodzaju selekcji jest jej niska wydajność w przypadku ogromnej listy przedmiotów.

Podobnie jak sortowanie bąbelkowe, sortowanie selekcji wymaga $n \cdot k$ -kwadratowej liczby kroków do sortowania n elementów. Ponadto na jego wydajność łatwo wpływa początkowe zamówienie elementów przed procesem sortowania. Z tego powodu sortowanie według wyboru jest odpowiednie tylko dla listy kilku elementów, które są w losowej kolejności.

Algorytm sortowania przez wybór posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie przez wstawianie** - Insertion Sort

Algorytm sortowania przez wstawianie można porównać do sposobu układania kart pobieranych z talii. Najpierw bierzemy pierwszą kartę. Następnie pobieramy kolejne, aż do wyczerpania talii. Każdą pobraną kartę porównujemy z kartami, które już trzymamy w ręce i szukamy dla niej miejsca przed pierwszą kartą starszą (młodszą w przypadku porządku malejącego). Gdy znajdziemy takie miejsce, rozsuwamy karty i nową wstawiamy na przygotowane w ten sposób miejsce (stąd pochodzi nazwa algorytmu - sortowanie przez wstawianie, ang. Insertion Sort). Jeśli nasza karta jest najstarsza (najmłodsza), to umieszczamy ją na samym końcu. Tak porządkujemy karty. A jak przenieść tę ideę do świata komputerów i zbiorów liczbowych?

Algorytm sortowania przez wstawianie będzie składał się z dwóch pętli. Pętla główna (zewnętrzna) symuluje pobieranie kart, czyli w tym wypadku elementów zbioru. Odpowiednikiem kart na ręce jest tzw. lista uporządkowana (ang. sorted list), którą sukcesywnie będziemy tworzyli na końcu zbioru (istnieje też odmiana algorytmu umieszczająca listę uporządkowaną na początku zbioru). Pętla sortująca (wewnętrzna) szuka dla pobranego elementu miejsca na liście uporządkowanej. Jeśli takie miejsce zostanie znalezione, to elementy listy są odpowiednio rozsuwane, aby tworzyć miejsce na nowy element i element wybrany przez pętlę główną trafia tam. W ten sposób lista uporządkowana rozrasta się. Jeśli na liście uporządkowanej nie ma elementu większego od wybranego, to element ten trafia na koniec listy. Sortowanie zakończymy, gdy pętla główna wybierze wszystkie elementy zbioru.

Główną zaletą tego rodzaju wstawiania jest jego prostota. Wykazuje również dobrą wydajność w przypadku małej listy. Sortowanie wstawiania jest algorymem sortowania na miejscu, więc zapotrzebowanie na miejsce jest minimalne. Wadą tego rodzaju wstawiania jest to, że nie działa on tak dobrze, jak inne, lepsze algorytmy sortowania. Ponieważ $n \cdot k$ -kwadratowych kroków jest wymaganych dla każdego n elementu do posortowania, sortowanie wstawiania nie radzi sobie

dobrze z ogromną listą. Dlatego sortowanie wstawianie jest szczególnie przydatne tylko podczas sortowania listy kilku elementów.

Algorytm sortowania przez wstawianie posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie szybkie** - Quick Sort

Szybkie sortowanie działa na zasadzie dziel i zwyciężaj. Po pierwsze, dzieli listę elementów na dwie listy podrzędne w oparciu o element przestawny. Wszystkie elementy w pierwszej podlistie są ustawione tak, aby były mniejsze niż oś obrotu, natomiast wszystkie elementy w drugiej podlistie są ustawione tak, aby były większe niż oś obrotu. Ten sam proces partycjonowania i aranżacji jest wykonywany wielokrotnie na powstałych podlistach, dopóki cała lista elementów nie zostanie posortowana.

Szybkie sortowanie jest uważane za najlepszy algorytm sortowania. Wynika to z jego znacznej przewagi pod względem wydajności, ponieważ jest w stanie poradzić sobie z ogromną listą przedmiotów. Ponieważ sortuje się w miejscu, nie jest również wymagane dodatkowe miejsce do przechowywania. Nieznaczną wadą szybkiego sortowania jest to, że jego działanie w najgorszym przypadku jest podobne do średnich wyników sortowania bąbelkowego, wstawiania lub selekcji. Ogólnie rzecz biorąc, szybkie sortowanie zapewnia najbardziej efektywną i powszechnie stosowaną metodę sortowania listy o dowolnym rozmiarze.

W przypadku typowym algorytm ten jest najszybszym algorytmem sortującym z klasy złożoności obliczeniowej $O(n \log n)$ - stąd pochodzi jego popularność w zastosowaniach. Musimy jednak pamiętać, iż w pewnych sytuacjach (zależnych od sposobu wyboru piwotu(element podziałowy) oraz niekorzystnego ułożenia danych wejściowych) klasa złożoności obliczeniowej tego algorytmu może się degradować do $O(n^2)$, co więcej, poziom wywołań rekurencyjnych może spowodować przepełnienie stosu i zablokowanie komputera. Z tych powodów algorytmu sortowania szybkiego nie można stosować bezmyślnie w każdej sytuacji tylko dlatego, iż jest uważany za jeden z najszybszych algorytmów sortujących - zawsze należy przeprowadzić analizę możliwych danych wejściowych właśnie pod kątem przypadku niekorzystnego.

- **Dwukierunkowe sortowanie bąbelkowe** - Bidirectional Bubble Sort

Dwukierunkowe sortowanie bąbelkowe oparte jest na spostrzeżeniu, iż każdy obieg wewnętrznej pętli sortującej umieszcza na właściwym miejscu element najstarszy, a elementy młodsze przesuwa o 1 pozycję w kierunku początku zbioru. Jeśli pętla ta zostanie wykonana w kierunku odwrotnym, to wtedy najmłodszy element znajdzie się na swoim właściwym miejscu, a elementy starszy przesuną się o jedną pozycję w kierunku końca zbioru. Połączmy te dwie pętle sortując wewnętrznie naprzemiennie w kierunku normalnym i odwrotnym, a otrzymamy algorytm dwukierunkowego sortowania bąbelkowego.

Wykonanie pętli sortującej w normalnym kierunku ustali maksymalną pozycję w zbiorze, od której powinna rozpocząć sortowanie pętla odwrotna. Ta z kolei ustali minimalną pozycję w zbiorze, od której powinna rozpocząć sortowanie pętla normalna w następnym obiegu pętli zewnętrznej. Sortowanie możemy zakończyć, jeśli nie wystąpiła potrzeba zamiany elementów w żadnej z tych dwóch pętli.

Optymalna klasa złożoności obliczeniowej dla algorytmu sortowania bąbelkowego dwukierunkowego wynosi $O(n)$. Za ogólną klasę złożoności uznaje się $O(n^2)$. Sortowanie bąbelkowe dwukierunkowe jest algorytmem sortowania w miejscu oraz jest algorytmem stabilnym.

(C-10) Omówić algorytmy sortowania. Proszę wymienić i krótko scharakteryzować poznane algorytmy sortowania danych, oraz przedstawić ich słabe i mocne strony, w kontekście właściwości i stopnia uporządkowania danych.

Ze względu na złożoność pamięciową algorytmy sortujące dzielimy na dwie podstawowe grupy:

1. **Algorytmy sortujące w miejscu** (ang. in place) - wymagają stałej liczby dodatkowych struktur danych, która nie zależy od liczby elementów sortowanego zbioru danych (ani od ich wartości). Dodatkowa złożoność pamięciowa jest zatem klasy $O(1)$. Sortowanie odbywa się wewnątrz zbioru. Ma to bardzo istotne znaczenie w przypadku dużych zbiorów danych, gdyż mogłoby się okazać, iż posortowanie ich nie jest możliwe z uwagi na brak pamięci w systemie. Większość opisanych tu algorytmów sortuje w miejscu, co jest ich bardzo dużą zaletą.
2. **Algorytmy nie sortujące w miejscu** - wymagają zarezerwowania w pamięci dodatkowych obszarów, których wielkość jest zależona od liczby sortowanych elementów lub od ich wartości. Tego typu algorytmy są zwykle bardzo szybkie w działaniu, jednakże okupione to jest dodatkowymi wymaganiami na pamięć. Zatem w pewnych sytuacjach może się okazać, iż taki algorytm nie będzie w stanie posortować dużego zbioru danych, ponieważ system komputerowy nie posiada wystarczającej ilości pamięci RAM.

Algorytmy sortujące dzieli się również na dwie grupy:

1. **Algorytmy stabilne** - zachowują kolejność elementów równych. Oznacza to, iż elementy o tych samych wartościach będą występować w tej samej kolejności w zbiorze posortowanym, co w zbiorze nieposortowanym. Czasami ma to znaczenie, gdy sortujemy rekordy bazy danych i nie chcemy, aby rekordy o tym samym kluczu zmieniały względem siebie położenie. Przykład: sortowanie bąbelkowe, przez scalanie, przez wstawianie.
2. **Algorytmy niestabilne** - kolejność wynikowa elementów równych jest nieokreślona (zwykle nie zostaje zachowana). Przykład: sortowanie szybkie.

- **Sortowanie bąbelkowe** - Bubble Sort

Algorytm sortowania bąbelkowego jest jednym z najstarszych algorytmów sortujących. Zasada działania opiera się na cyklicznym porównywaniu par sąsiadujących elementów i zamianie ich kolejności w przypadku niespełnienia kryterium porządkowego zbioru. Operację tę wykonujemy dotąd, aż cały zbiór zostanie posortowany.

Podstawową zaletą sortowania bąbelkowego jest to, że jest popularny i łatwy do wdrożenia. Ponadto w sortowaniu bąbelkowym elementy są zamieniane na miejscu bez dodatkowego tymczasowego przechowywania, więc zapotrzebowanie na miejsce jest minimalne. Główną wadą tego typu bąbelków jest to, że nie radzi sobie dobrze z listą zawierającą ogromną liczbę przedmiotów. Wynika to z tego, że sortowanie bąbelkowe wymaga $n \cdot n$ -kwaterowych kroków przetwarzania dla każdej liczby elementów do posortowania. Jako taki, sortowanie bąbelkowe nadaje się głównie do nauczania akademickiego, ale nie do rzeczywistych zastosowań.

Algorytm sortowania bąbelkowego przy porządkowaniu zbioru nieposortowanego ma klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie przez wybór** - Selection Sort

Idea algorytmu sortowania przez wybór jest bardzo prosta. Założymy, iż chcemy posortować zbiór liczbowy rosnąco. Zatem element najmniejszy powinien znaleźć się na pierwszej pozycji. Szukamy w zbiorze elementu najmniejszego i wymieniamy go z elementem na pierwszej pozycji. W ten sposób element najmniejszy znajdzie się na swojej docelowej pozycji.

W identyczny sposób postępujemy z resztą elementów należących do zbioru. Znów wyszukujemy element najmniejszy i zamieniamy go z elementem na drugiej pozycji. Otrzymamy dwa posortowane elementy. Procedurę kontynuujemy dla pozostałych elementów dotąd, aż wszystkie będą posortowane.

Główną zaletą sortowania selekcyjnego jest to, że działa dobrze na małej liście. Ponadto, ponieważ jest to algorytm sortowania na miejscu, nie jest wymagane dodatkowe tymczasowe miejsce do przechowywania poza tym, co jest potrzebne do przechowywania oryginalnej listy. Podstawową wadą tego rodzaju selekcji jest jej niska wydajność w przypadku ogromnej listy przedmiotów.

Podobnie jak sortowanie bąbelkowe, sortowanie selekcji wymaga $n \cdot k$ -kwadratowej liczby kroków do sortowania n elementów. Ponadto na jego wydajność łatwo wpływa początkowe zamówienie elementów przed procesem sortowania. Z tego powodu sortowanie według wyboru jest odpowiednie tylko dla listy kilku elementów, które są w losowej kolejności.

Algorytm sortowania przez wybór posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie przez wstawianie** - Insertion Sort

Algorytm sortowania przez wstawianie można porównać do sposobu układania kart pobieranych z talii. Najpierw bierzemy pierwszą kartę. Następnie pobieramy kolejne, aż do wyczerpania talii. Każdą pobraną kartę porównujemy z kartami, które już trzymamy w ręce i szukamy dla niej miejsca przed pierwszą kartą starszą (młodszą w przypadku porządku malejącego). Gdy znajdziemy takie miejsce, rozsuwamy karty i nową wstawiamy na przygotowane w ten sposób miejsce (stąd pochodzi nazwa algorytmu - sortowanie przez wstawianie, ang. Insertion Sort). Jeśli nasza karta jest najstarsza (najmłodsza), to umieszczamy ją na samym końcu. Tak porządkujemy karty. A jak przenieść tę ideę do świata komputerów i zbiorów liczbowych?

Algorytm sortowania przez wstawianie będzie składał się z dwóch pętli. Pętla główna (zewnętrzna) symuluje pobieranie kart, czyli w tym wypadku elementów zbioru. Odpowiednikiem kart na ręce jest tzw. lista uporządkowana (ang. sorted list), którą sukcesywnie będziemy tworzyli na końcu zbioru (istnieje też odmiana algorytmu umieszczająca listę uporządkowaną na początku zbioru). Pętla sortująca (wewnętrzna) szuka dla pobranego elementu miejsca na liście uporządkowanej. Jeśli takie miejsce zostanie znalezione, to elementy listy są odpowiednio rozsuwane, aby tworzyć miejsce na nowy element i element wybrany przez pętlę główną trafia tam. W ten sposób lista uporządkowana rozrasta się. Jeśli na liście uporządkowanej nie ma elementu większego od wybranego, to element ten trafia na koniec listy. Sortowanie zakończymy, gdy pętla główna wybierze wszystkie elementy zbioru.

Główną zaletą tego rodzaju wstawiania jest jego prostota. Wykazuje również dobrą wydajność w przypadku małej listy. Sortowanie wstawiania jest algorymem sortowania na miejscu, więc zapotrzebowanie na miejsce jest minimalne. Wadą tego rodzaju wstawiania jest to, że nie działa on tak dobrze, jak inne, lepsze algorytmy sortowania. Ponieważ $n \cdot k$ -kwadratowych kroków jest wymaganych dla każdego n elementu do posortowania, sortowanie wstawiania nie radzi sobie

dobrze z ogromną listą. Dlatego sortowanie wstawianie jest szczególnie przydatne tylko podczas sortowania listy kilku elementów.

Algorytm sortowania przez wstawianie posiada klasę czasowej złożoności obliczeniowej równą $O(n^2)$. Sortowanie odbywa się w miejscu.

- **Sortowanie szybkie** - Quick Sort

Szybkie sortowanie działa na zasadzie dziel i zwyciężaj. Po pierwsze, dzieli listę elementów na dwie listy podrzędne w oparciu o element przestawny. Wszystkie elementy w pierwszej podlistie są ustawione tak, aby były mniejsze niż oś obrotu, natomiast wszystkie elementy w drugiej podlistie są ustawione tak, aby były większe niż oś obrotu. Ten sam proces partycjonowania i aranżacji jest wykonywany wielokrotnie na powstałych podlistach, dopóki cała lista elementów nie zostanie posortowana.

Szybkie sortowanie jest uważane za najlepszy algorytm sortowania. Wynika to z jego znacznej przewagi pod względem wydajności, ponieważ jest w stanie poradzić sobie z ogromną listą przedmiotów. Ponieważ sortuje się w miejscu, nie jest również wymagane dodatkowe miejsce do przechowywania. Nieznaczną wadą szybkiego sortowania jest to, że jego działanie w najgorszym przypadku jest podobne do średnich wyników sortowania bąbelkowego, wstawiania lub selekcji. Ogólnie rzecz biorąc, szybkie sortowanie zapewnia najbardziej efektywną i powszechnie stosowaną metodę sortowania listy o dowolnym rozmiarze.

W przypadku typowym algorytm ten jest najszybszym algorytmem sortującym z klasy złożoności obliczeniowej $O(n \log n)$ - stąd pochodzi jego popularność w zastosowaniach. Musimy jednak pamiętać, iż w pewnych sytuacjach (zależnych od sposobu wyboru piwotu(element podziałowy) oraz niekorzystnego ułożenia danych wejściowych) klasa złożoności obliczeniowej tego algorytmu może się degradować do $O(n^2)$, co więcej, poziom wywołań rekurencyjnych może spowodować przepełnienie stosu i zablokowanie komputera. Z tych powodów algorytmu sortowania szybkiego nie można stosować bezmyślnie w każdej sytuacji tylko dlatego, iż jest uważany za jeden z najszybszych algorytmów sortujących - zawsze należy przeprowadzić analizę możliwych danych wejściowych właśnie pod kątem przypadku niekorzystnego.

- **Dwukierunkowe sortowanie bąbelkowe** - Bidirectional Bubble Sort

Dwukierunkowe sortowanie bąbelkowe oparte jest na spostrzeżeniu, iż każdy obieg wewnętrznej pętli sortującej umieszcza na właściwym miejscu element najstarszy, a elementy młodsze przesuwa o 1 pozycję w kierunku początku zbioru. Jeśli pętla ta zostanie wykonana w kierunku odwrotnym, to wtedy najmłodszy element znajdzie się na swoim właściwym miejscu, a elementy starszy przesuną się o jedną pozycję w kierunku końca zbioru. Połączmy te dwie pętle sortując wewnętrznie naprzemiennie w kierunku normalnym i odwrotnym, a otrzymamy algorytm dwukierunkowego sortowania bąbelkowego.

Wykonanie pętli sortującej w normalnym kierunku ustali maksymalną pozycję w zbiorze, od której powinna rozpocząć sortowanie pętla odwrotna. Ta z kolei ustali minimalną pozycję w zbiorze, od której powinna rozpocząć sortowanie pętla normalna w następnym obiegu pętli zewnętrznej. Sortowanie możemy zakończyć, jeśli nie wystąpiła potrzeba zamiany elementów w żadnej z tych dwóch pętli.

Optymalna klasa złożoności obliczeniowej dla algorytmu sortowania bąbelkowego dwukierunkowego wynosi $O(n)$. Za ogólną klasę złożoności uznaje się $O(n^2)$. Sortowanie bąbelkowe dwukierunkowe jest algorytmem sortowania w miejscu oraz jest algorytmem stabilnym.

- **Sortowanie przez scalanie** - Merge Sort

Algorytm sortowania przez scalanie, to kolejny już z algorytmów, który opiera się na zasadzie „dziel i zwyciężaj”. Można wręcz powiedzieć, że wykorzystuje ją w dość bezpośrednim rozumieniu. Główna zasada działania polega na rekurencyjnym dzieleniu tablicy na podtablice. Dzielenie kończymy, w którym, każda z podtablic w danej grupie jest tablicą jednoelementową. Łączymy je kolejno porównując wartości ich elementów.

Złożoność algorytmu wynosi $n \cdot \log n$, a więc jest on znacznie wydajniejszy niż sortowanie bąbelkowe, przez wstawianie czy przez selekcję, gdzie złożoność jest kwadratowa.

Zalety algorytmu

- prostota implementacji
- wydajność
- stabilność
- algorytm sortuje zbiór **n-elementowy** w czasie proporcjonalnym do liczby $n \log n$ bez względu na rodzaj danych wejściowych

Wady algorytmu

- podczas scalania potrzebny jest dodatkowy obszar pamięci przechowujący kopie podtablic do scalenia

Omówić sposoby kopiowania instancji w programowaniu obiektowym. Proszę omówić różnice pomiędzy kopią płytka, a kopią głęboką, oraz wyjaśnić jakie są konsekwencje stosowania jednej i drugiej

Sposoby kopiowania

Kopiowanie obiektów jest operacją wykonywaną bardzo często: przekazując argumenty przez wartość, zwracając wyniki obliczeń, przechowując elementy w kontenerach i w wielu innych sytuacjach są tworzone kopie. Jeżeli obiekt jest dostępny pośrednio, na przykład przez wskaźnik, to można wykonać kopię wskaźnika albo całego obiektu. W związku z tym możemy wyróżnić dwa rodzaje kopiowania:

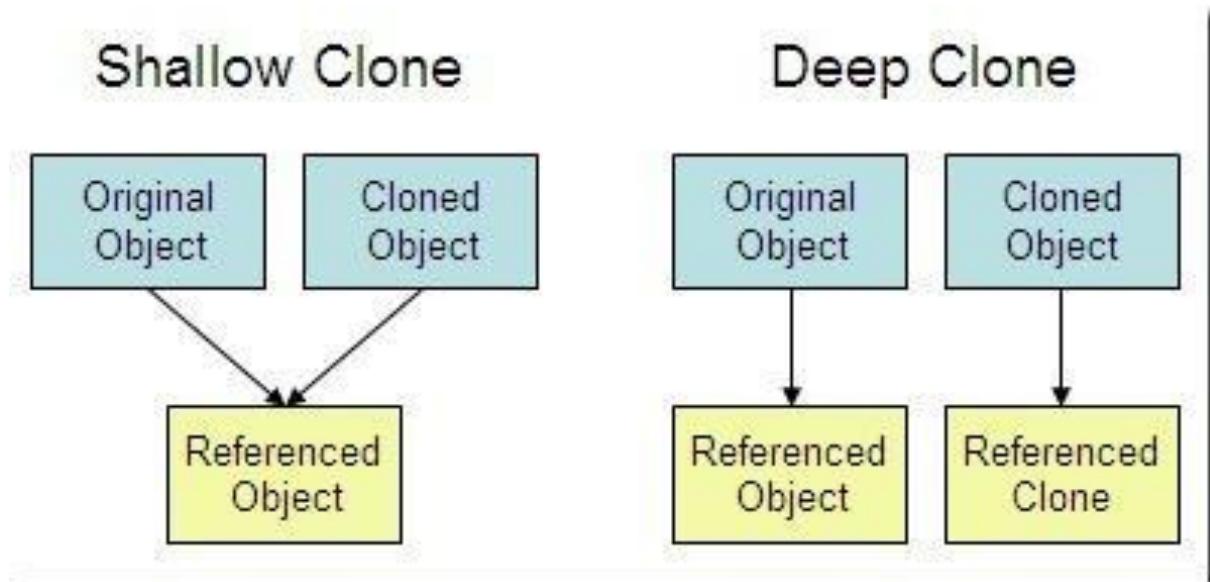
- Kopiowanie płytkie,
- Kopiowanie głębokie

Kopiowanie płytkie

Kopią płytka (Shell Copy) nazywa się kopiowanie jedynie obiektów pośredniczących, wskaźników, referencji itp. Kopia taka jest tworzona szybko, ponieważ wskaźnik lub inny obiekt pośredniczący jest zazwyczaj małym obiektem. Po wykonaniu płytkiej kopii ten sam obiekt jest dostępny z wielu miejsc, obiekt wskazywany nie jest do końca kopiowany, zmiana jego stanu będzie widoczna we wszystkich kopiacach.

Kopiowanie głębokie

Głęboka kopia (Deep copy) oznacza rzeczywiste kopiowanie obiektów wskazywanych. Tworzenie takiej kopii zajmuje więcej czasu, i zasobów pamięci ale obiekt i kopia są od siebie niezależne. Zmiany obiektu nie mają wpływu na kopię.



Różnice i konsekwencje

Kopia płytka

- Przechowuje odniesienia do oryginalnego adresu pamięci
- Odzwierciedla zmiany wprowadzone w nowym/skopiowanym obiekcie w oryginalnym obiekcie
- Przechowuje kopię oryginalnego obiektu przez odniesienia do obiektów
- Jest szybsza

Kopia głęboka

- Przechowuje kopie wartości obiektu
- Nie odzwierciedla zmian wprowadzonych w nowym/skopiowanym obiekcie w oryginalnym obiekcie
- Przechowuje kopię oryginalnego obiektu przez tworzenie nowych obiektów
- Jest wolniejsza

(D-01) Omówić sposoby kopiowania instancji w programowaniu obiektowym. Proszę omówić różnicę pomiędzy kopią płytka, a kopią głęboką, oraz wyjaśnić jakie są konsekwencje stosowania jednej i drugiej.

1. Kopiowanie w programowaniu obiektowym- tworzenie nowej instancji obiektu z identycznymi parametrami pól.

2. Kopia płytka- Jest to jedna z metod kopiowania obiektu, znana również jako metoda pole po polu w momencie gdy kopowany obiekt posiada pole będące innym obiektem to skopiowana zostaje referencja do tej instancji.

3. Kopia głęboka- Jest to druga z metod kopiowania instancji, gdy wywołana zostaje kopia głęboka podobnie jak w przypadku kopi płytkiej kopujemy każde pole, ale w momencie napotkania na pole typu nieprostego to nie jest kopowana referencja a wywoływane jest kopia głęboka dla tego obiektu.

4. Kopia leniwa- Jest to połączenie kopiowania płytkego oraz głębokiego.

5. Konsekwencje stosowania kopi płytkiej- mniejsze zużycie pamięci oraz szybsze działanie, zmiany na współdzielonych obiektach odzwierciedlane są na oryginałe oraz kopi.

6. Konsekwencje stosowania kopi głębokiej- większe zużycie pamięci przez skopiowane elementy spowodowane pełnym kopiowaniem typów nie prostych będących polami instancji. Zmiana wartości w skopiowanej tablicy niepodwójne zmiany w oryginalnej tablicy.

Różnice

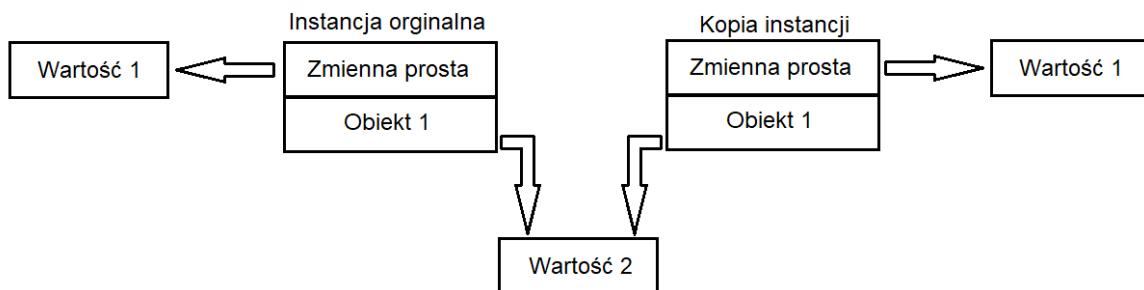
Kopia płytka	Kopia Głęboka
Kopia przechowuje referencje do skopiowanych pól.	Kopia przechowuje także kopie swoich pól.
Zmiany w obiektach odzwierciedlane są na oryginałe oraz kopi.	Zmiany w obiektach działają tylko na instancji, na której zostały wywołane.
Szybsze	Wolniejsze
Zużywa mniej pamięci.	Zużywają więcej pamięci.

1.

W programowaniu obiektowym kopiowanie obiektów to tworzenie kopii istniejącego obiektu, jednostki danych w programowaniu obiektowym. Istnieje kilka sposobów kopiowania obiektu, najczęściej za pomocą konstruktora kopującego lub klonowania. Kopiowanie odbywa się głównie po to, aby można było modyfikować lub przenosić kopię lub zachować obecną wartość. Podczas gdy w prostych przypadkach kopiowanie można wykonać poprzez przydzielenie nowego, niezainicjowanego obiektu i skopiowanie wszystkich pól z oryginalnego obiektu, w bardziej złożonych przypadkach nie skutkuje to pożądanym zachowaniem.

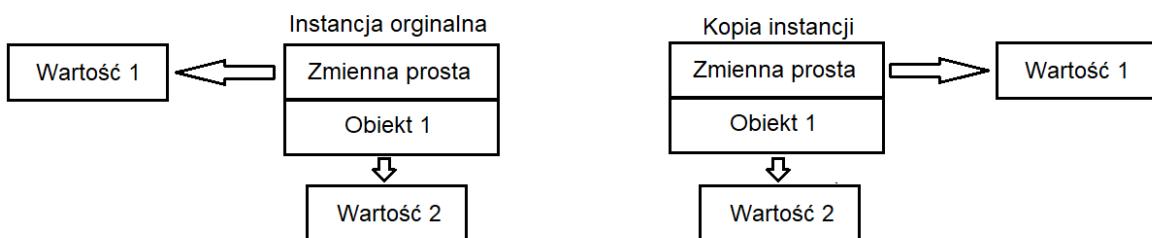
2.

Kopia płytka to stworzenie nowego obiektu, który zawiera te same pola co oryginał, z kopiami tych samych wartości: W przypadku stosunkowo prostych obiektów działa to dobrze. Jeśli jednak nasze obiekty zawierają inne obiekty, to skopiowane zostaną tylko referencje do nich. To z kolei oznacza, że obie kopie zawierają odniesienia do tej samej wartości w pamięci, wraz z zaletami i wadami, które to pociąga za sobą.



Rys.1.Wizualizacja działania kopiowania płytkego.

3. Kopia głęboka polega na stworzeniu nowego obiektu, którego wartości są identyczne do oryginalnego, ale zamiast kopiowania odwołań do obiektu to tworzone są nowe obiekty a odniesienia do nich są przechowywane tylko w nowej instancji, powoduje to, że obiekty znajdujące w kopiowanej instancji są niezależne od tych z oryginału. Kopiowanie głębokie odbywa się rekurencyjnie to znaczy, że najpierw tworzony jest obiekt kopiowany a następnie jest on wypełniany kopiami obiektów podrzędnych.



Rys.2.Wizualizacja działania kopiowania głębokiego.

4.Kopia leniwa to połączenie kopii płytkej oraz głębokiej podczas używania tej metody na początku procesu kopiowania jest uruchamiane kopiowanie płytke, używany jest także licznik, który zlicza ile obiektów dzieli ze sobą informacje. W momencie modyfikacji jednej z instancji program sprawdza licznik czy dany obiekt nie jest współdzielony z inną instancją a jeżeli jest to wtedy uruchamiane są metody kopiowania głębokiego.

5.Konsekwencje stosowania kopi płytkej:

-Mniejsze zużycie pamięci oraz szybsze działanie- jest to spowodowane tym, że obiekty znajdujące się w instancji nie są kopiowane, ale tylko kopiowane są referencje do tego obiektu.

-Zmiany na obiektach, które są współdzielone przez kilka instancji zostaną odzwierciedlone na wszystkich instancjach.

```
from copy import copy

class Obiekt:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

original = Obiekt(1,2,[*range(20)])
kopia_p = copy(original)

print(f"Orginal: \na={original.a} b={original.b} c:{original.c}")
print(f"Kopia płytka: \na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")

original.a = 11
original.c.append(1111)

print(f"\nZmiana wartosci pola a w instancji oryginalu na {original.a} oraz dodanie do listy wartosci: {original.c[-1]}")
print(f"Orginal: \na={original.a} b={original.b} c:{original.c}")
print(f"Kopia płytka: \na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")

kopia_p.a = 5
kopia_p.c.append(5555)

print(f"\nZmiana wartosci pola a w instancji kopi płytkiej na {kopia_p.a} oraz dodanie do listy wartosci: {kopia_p.c[-1]}")
print(f"Orginal: \na={original.a} b={original.b} c:{original.c}")
print(f"Kopia płytka: \na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")
```

Rys.3.Przykładowy kod w języku python wykorzystujący kopię płytkę.

```
Orginal:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Kopia płytka:
a=1 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartosci pola a w instancji oryginalu na 11 oraz dodanie do listy wartosci: 1111
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]
Kopia płytka:
a=1 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]

Zmiana wartosci pola a w instancji kopi płytkiej na 5 oraz dodanie do listy wartosci: 5555
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
Kopia płytka:
a=5 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
```

Rys.4.Wynik działania kodu.

6.Konsekwencje stosowania kopi głębokiej.

-Większe zużycie pamięci oraz wolniejsze działanie, spowodowane jest to tym, że w czasie kopiowania instancji dla każdego typu nieprostego uruchomiana jest własna kopia głęboka.

-Zmiany w obiekcie oryginalnym oraz kopowanym nie wpływają na siebie, jest to spowodowane tym, że każdy obiekt w kopi nie jest w żaden sposób połączony z oryginałem.

```
from copy import copy, deepcopy

class Obiekt:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

orginal = Obiekt(1,2,[*range(20)])
kopia_g = deepcopy(orginal)

print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")

orginal.a = 11
orginal.c.append(1111)

print(f"\nZmiana wartości pola a w instancji oryginalu na {orginal.a} oraz dodanie do listy wartości: {orginal.c[-1]}")
print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")

kopia_g.a = 5
kopia_g.c.append(5555)

print(f"\nZmiana wartości pola a w instancji kopi głębokiej na {kopia_g.a} oraz dodanie do listy wartości: {kopia_g.c[-1]}")
print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")
```

Rys.5.Przykładowy kod wykorzysujący kopię głęboką.

```
Orginal:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Kopia głęboka:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartości pola a w instancji oryginalu na 11 oraz dodanie do listy wartości: 1111
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]
Kopia głęboka:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartości pola a w instancji kopi głębokiej na 5 oraz dodanie do listy wartości: 5555
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]
Kopia głęboka:
a=5 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 5555]
```

Rys.6.Wynik działania kodu.

```

from copy import copy, deepcopy

class Obiekt:
    def __init__(self, a, b, c):
        self.a = a
        self.b = b
        self.c = c

orginal = Obiekt(1,2,[*range(20)])
kopia_g = deepcopy(orginal)
kopia_p = copy(orginal)

print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia płytki:\na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")

orginal.a = 11
orginal.c.append(1111)

print(f"\nZmiana wartości pola a w instancji oryginalu na {orginal.a} oraz dodanie do listy wartości: {orginal.c[-1]}")
print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia płytki:\na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")

kopia_p.a = 5
kopia_p.c.append(5555)

print(f"\nZmiana wartości pola a w instancji kopi płytkiej na {kopia_p.a} oraz dodanie do listy wartości: {kopia_p.c[-1]}")
print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia płytki:\na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")
print(f"Kopia głęboka:\na={kopia_g.a} b={kopia_g.b} c:{kopia_g.c}")

kopia_g.a = 8
kopia_g.c.append(8888)

print(f"\nZmiana wartości pola a w instancji kopi głębokiej na {kopia_g.a} oraz dodanie do listy wartości: {kopia_g.c[-1]}")
print(f"Orginal:\na={orginal.a} b={orginal.b} c:{orginal.c}")
print(f"Kopia płytki:\na={kopia_p.a} {kopia_p.b} c:{kopia_p.c}")

```

Rys.7.Przykład kodu wykorzystującego kopie płytka i głęboką.

```

Orginal:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Kopia płytka:
a=1 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
Kopia głęboka:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartości pola a w instancji oryginalu na 11 oraz dodanie do listy wartości: 1111
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]
Kopia płytka:
a=1 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111]
Kopia głęboka:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartości pola a w instancji kopi płytkiej na 5 oraz dodanie do listy wartości: 5555
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
Kopia płytka:
a=5 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
Kopia głęboka:
a=1 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Zmiana wartości pola a w instancji kopi głębokiej na 8 oraz dodanie do listy wartości: 8888
Orginal:
a=11 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
Kopia płytka:
a=5 2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 1111, 5555]
Kopia głęboka:
a=8 b=2 c:[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 8888]

```

Rys.8.Wynik działania programu.

Źródła:

<https://depot.ceon.pl/bitstream/handle/123456789/3155/sdj2010prototype.pdf?sequence=1&isAllo wed=y> [Dostęp:22.05.2022]

<https://javastart.pl/baza-wiedzy/efektywne-programowanie/plytkie-i-glebokie-kopiowanie- klonowanie-obiektow> [Dostęp:22.05.2022]

<https://www.geeksforgeeks.org/difference-between-shallow-and-deep-copy-of-a-class/> [Dostęp 22.05.2022]

<https://www.baeldung.com/cs深深拷贝-vs-浅拷贝> [Dostęp:22.05.2022]

<https://www.javatpoint.com/shallow-copy-vs-deep-copy-in-java> [Dostęp:22.05.2022]

**(D-02) Omówić sposoby kopiowania danych. Proszę wyjaśnić działanie kopii binarnej i element-po-
elemencie, oraz wyjaśnić kiedy możemy/musimy korzystać z danego typu kopii.**

Do podstawowych metod kopiowania danych w programowaniu możemy zaliczyć kopiowanie element po elemencie i kopiowanie binarne. Pierwsze polega na tym że użytkownik jawnie definiuje, które dane mają zostać skopiowane oraz czy podczas kooptowania chce dokonać rzutowania typów danych. Cała operacja w programowaniu wygląda następująco $X = B$. Czyli do zmiennej X przypisujemy zawartość zmiennej B. Druga metoda natomiast różni się od pierwszej tym że Typ kopiowanych danych jest nieistotny, gdyż memcpy kopiuje same bajty. Z tego powodu jej użycie do kopiowania obiektów typów niebędących POD(Passive data structure) nie jest zalecane, ponieważ zdefiniowane przez użytkownika konstruktory i operatory przypisania nie zostaną wywołane. Dodatkowym warunkiem pomyślnym zakończenia operacji jest fakt iż BLOKI PAMIĘCI NIE MOGĄ NA SIEBIE ZACHODZIĆ. Kopiowana jest zadeklarowana ilość bajtów z miejsca wskazywanego przez użytkownika (miejsca początkowego) do pamięci wskazywanej przez użytkownika(miejsce docelowe). Aby wykonać tą operację niezbędne jest użycie funkcji memcpy() w języku c++. Funkcja memcpy() przyjmuje trzy argumenty. Pierwszym jest: wskaźnik na pamięć, do której nastąpi kopiowanie. Drugim jest wskaźnik na pamięć, z której nastąpi kopiowanie. Ostatni to liczba bajtów do skopiowania. Więc dokładna składnia funkcji memcpy() wygląda następująco:

```
void * memcpy( void * destination, const void * source, size_t num );
```

Możesz używać memcpy tylko wtedy, gdy kopiowane obiekty nie mają jawnych konstruktorów, tak jak ich członkowie (tzw. POD, "zwykłe stare dane"). Można więc wywołać memcpy dla float, ale źle jest dla, np. std::string. Kopiowanie jawnie za pomocą element po elemencie jest mniej narażone na błędy(np. zapełnienie stosu) oraz jest bardziej czytelne.

D-03

- ▶ Wartości logiczne
- ▶ Liczby całkowite
- ▶ Liczby rzeczywiste
- ▶ Znaki iłańcuchy znakowe
- ▶ Obrazy cyfrowe, filmy
- ▶ Treści audio
- ▶ Wartości logiczne w komputerze są przedstawiane za pomocą systemu 0/1 gdzie:
 - ▶ 0 oznacza false
 - ▶ 1 oznacza true
- ▶ Wszystkie liczby naturalne mają identyczny kod (na danej liczbie bitów). Najstarszy bit jest bitem znaku i dla liczb dodatnich ma wartość zero a dla liczb ujemnych 1. Na pozostałych bitach jest zapisana liczba.
- ▶ Kod ZM (znak-moduł) – wszystkie bity są bitami modułu kodowanej liczby poza najstarszym bitem który jest bitem znaku. Bit znaku zapisuje się jako $(-1)^{bn-1}$ W tym sposobie kodowania występują dwie reprezentacje zera oraz nie można na nim stosować naturalnych operacji arytmetycznych.
- ▶ Kod U2 (uzupełnienie do 2) – najstarszy bit odpowiada za znak ale ma wagę -2^{n-1} , kodowanie to jest obecnie najpopularniejszą reprezentacją liczb całkowitych. Operacje dodawania i odejmowania są w nim wykonywane naturalnie.
- ▶ Do zapisu liczby stałoprzecinkowej przeznaczona jest z góry określona liczba bitów, a pozycję przecinka ustala się arbitralnie, w zależności od wymaganej dokładności, wolne bity uzupełniając zerami. Do reprezentacji tych liczb stosuje się kod U2, np.:
 - ▶ Liczba $6,25=110,01_{(2)}$ zapisana na 8 bitach gdy część ułamkowa zajmuje 3 najmłodsze bity.
- ▶ Zapis zmiennoprzecinkowy jest komputerową reprezentacją liczb rzeczywistych zapisanych w postaci wykładniczej o podstawie 2, np.:
 - ▶ $0,0224609375 = 0,0000010111_{(2)} = 1,0111_{(2)} * 2^{-6}$
- ▶ Znaki alfanumeryczne to znaki, które mogą być wprowadzone za pomocą klawiatury komputera.
- ▶ Przykładowe standardy kodowania:
 - ▶ ASCII
 - ▶ 8-bitowy kod przyporządkowujący liczby z zakresu 0-255
 - ▶ Litery alfabetu angielskiego (65-90 dla dużych liter)
 - ▶ Cyfry (48-57)
 - ▶ Znaki przestankowe
 - ▶ Inne symbole oraz polecenia sterujące
 - ▶ UNICODE – uniwersalny standard kodowania znaków, dzięki któremu można wyświetlać znaki charakterystyczne dla różnych języków, używa opisu literowego i liczbowej wartości dla każdego kodowanego znaku. Wartość 16 bitowa jest definiowana jako liczba w systemie szesnastkowym wraz z przedrostkiem U (np. U+0041 przedstawia A).
 - ▶ Reprezentacja zależy od programu, w jakim obraz został utworzony. Wyróżnia się:

- ▶ Grafikę rastrową (bitmapy) – im większa liczba pikseli składa się na obraz tym lepsza jego jakość. W pamięci komputera zapisane są w postaci bitów cechy charakterystyczne każdego piksela, położenie oraz kolor.
- ▶ Grafikę wektorową – obraz składa się z prostych obiektów graficznych (wektorów), łuków, prostych, zamalowanych obszarów. W pamięci komputera wektor zapisany jest w postaci równania
- ▶ Standardy kolorów:
 - ▶ RGB – na zapis koloru jednego piksela przeznacza się 24 bity i traktuje się jako wypadkową trzech kolorów składowych: red, green, blue
 - ▶ CMYK - na zapis przeznacza się 24 bity, lecz traktuje się jako wypadkową czterech kolorów składowych: cyan, magenta, yellow, kolor bazowy – zwykle czarny
- ▶ Reprezentacja zależy od programu, w jakim obraz został utworzony. Wyróżnia się:
 - ▶ Grafikę rastrową (bitmapy) – im większa liczba pikseli składa się na obraz tym lepsza jego jakość. W pamięci komputera zapisane są w postaci bitów cechy charakterystyczne każdego piksela, położenie oraz kolor.
 - ▶ Grafikę wektorową – obraz składa się z prostych obiektów graficznych (wektorów), łuków, prostych, zamalowanych obszarów. W pamięci komputera wektor zapisany jest w postaci równania
- ▶ Standardy kolorów:
 - ▶ RGB – na zapis koloru jednego piksela przeznacza się 24 bity i traktuje się jako wypadkową trzech kolorów składowych: red, green, blue
 - ▶ CMYK - na zapis przeznacza się 24 bity, lecz traktuje się jako wypadkową czterech kolorów składowych: cyan, magenta, yellow, kolor bazowy – zwykle czarny
- ▶ Podstawą zapisu cyfrowego, czyli digitalizacji dźwięku jest próbkowanie, czyli badanie parametrów dźwięku w ścisłe określonych odstępach czasu.
- ▶ Częstotliwość próbkowania to inaczej liczba próbek pobranych w ciągu sekundy
- ▶ Im większa częstotliwość próbkowania tym lepsza jakość dźwięku
- ▶ Dla człowieka są zapisywane dźwięki w skali 2x większej niż zasięg słyszalnych dźwięków dla człowieka czyli $2 \times 20000 \text{ Hz} = 40000 \text{ Hz}$
- ▶ Każda próbka może być kodowana z określona dokładnością. Poziom tej dokładności zależy od liczby bitów przeznaczonych na zakodowanie próbki.
- ▶ Cechy fali dźwiękowej zapisywane w czasie próbkowania to:
 - ▶ Wysokość dźwięku (zależna od częstotliwości drgań)
 - ▶ Natężenie dźwięku (zależne od amplitudy drgań)

[D-04]

Omówić konstrukcje wskaźnikowe w C / C++. Proszę wyjaśnić co to jest wskaźnik na wskaźnik i jakie są jego zastosowania, oraz omówić jak tworzone są wielowymiarowe tablice dynamiczne w C++.

Wskaźnik pojedyczny

- Najczęściej spotykany rodzaj wskaźnika
- Może wskazywać na dowolny typ danych (za wyjątkiem innych wskaźników)
- Przechowuje wartość liczbową o rozmiarze zależnym od architektury systemu operacyjnego (32-bit → 4 bajty, 64-bit → 8 bajtów, ...)
- Definiowany za pomocą operatora pobrania adresu (&)

Wskaźnik podwójny (wskaźnik na wskaźnik)

- Wskazuje tylko na wskaźniki pojedyncze
- Tworzony jest przez użycie operatora wyłuskania adresu na istniejącym wskaźniku
- Rozmiar identyczny do wskaźnika pierwszego rzędu (*)
- Dereferencja pojedyncza a wielokrotna (podwójna)

Zastosowania wskaźnika na wskaźnik

- Dynamiczne tablice dwuwymiarowe
- Dynamiczne jednowymiarowe tablice obiektów złożonych
- Przekazywanie wskaźników do funkcji

Tworzenie wielowymiarowych tablic dynamicznych C++ (1)

- Technika alokacji fragmentarycznej
- Tablica nadrzędna typu wskaźnikowego – rząd wskaźnika równy liczbie wymiarów tablicy (np. dwuwymiarowa → **, trójwymiarowa → ***)
- Każdy element tej tablicy jest analogiczną tablicą typu wskaźnikowego rzędu o jeden niższego aż do uzyskania typu wskazywanego
- Relatywnie duża elastyczność, niska szybkość działania
- Łatwe tworzenie tablic nieregularnych

Tworzenie wielowymiarowych tablic dynamicznych (2)

- Technika alokacji ciągłej
- Tablica nadrzędna również typu wskaźnikowego
- Pierwszy element tej tablicy (indeks zerowy) przechowuje faktyczną tablicę wartości (typ wskazywany)
- Rozmiar tablicy z wartościami jest równy iloczynowi wymiarów
- Każdy kolejny wskaźnik w tablicy nadrzędnej wskazuje na różne pozycje w tej samej tablicy
- Utrudnione tworzenie tablic nieregularnych
- Zdecydowanie trudniejsza alokacja tablic o 3 lub więcej wymiarach

(D-05): Omówić działanie mechanizmu rekurencji. Proszę wyjaśnić, jak działa mechanizm rekurencji, jak projektuje się algorytmy rekurencyjne i jakie są ograniczenia ich działania, oraz podać przykład nietrywialnego algorytmu rekurencyjnego.

Rekurencja

- Rekurencja zwana **rekursją**, polega na wywołaniu przez funkcję samej siebie. Rekurencja postępuje zgodnie z techniką dzielenia i wywołania algorytmu, co oznacza, że problem jest dzielony na podrzędne problemy tego samego lub pokrewnego typu, aż problem podrzędny stanie się na tyle prosty, że można go rozwiązać bezpośrednio.
- Algorytmy rekurencyjne zastępują w pewnym sensie iteracje. Niekiedy problemy rozwiązywane tą techniką będą nieznacznie wolniejsze (wiąże się to z wywoływaniem funkcji), natomiast rozwiązanie niektórych problemów jest dużo prostsze w implementacji.

Algorytmy rekurencyjne

- **Aby zaimplementować algorytm rekurencyjny, problem musi spełniać następujące warunki:**
 - *Można go podzielić na prostszą wersję tego samego problemu*
 - *Ma jeden lub więcej przypadków bazowych, których wartości są znane*
- **Kluczowe kwestie podczas projektowania algorytmu rekurencyjnego**
 - *Obsługa prostych zdarzeń bez użycia rekurencji*
 - *Unikanie zapętlenia – cykli: upewnienie się, że mamy do czynienia z coraz mniejszymi lub prostszymi problemami*
 - *Każde wywołanie modułu reprezentuje kompletną obsługę danego zadania*
- **Ograniczenia w działaniu algorytmów**
 - *Prędkość działania: algorytmy wykorzystywane przy dużych obiektach są mało efektywne, jeśli chodzi o czas pracy ze względu na obciążenie pamięci*
 - *Głębokość rekurencji: niektóre kompilatory ograniczają głębokość celem ochrony przed nieskończonymi rekurencjami*
 - *Mogliwość przepelenia stosu procesu interpretera: możliwość zatrzymania procesu*

Przykład przeszukiwania binarnego rozwiązany rekurencją

- **Wyszukiwanie binarne** jest to algorytm, który potrafi wyszukać w krótkim czasie żądany element. Przeszukiwana tablica musi być **posortowana**. Algorytm polega na zauważaniu zakresu gdzie może znajdować się szukany element.
- Przymijmy cztery argumenty: lista - tablica do przeszukania, szukana - element do znalezienia, lewy_index i prawy_index - indeksy zakresu. Wyliczamy środek zakresu. Jeśli środkowy element jest równy szukanemu to zwracamy indeks srodek. Jeśli lewy_index = prawy_index i nie zwróciliśmy wyniku oznacza to, że dany element nie istnieje i zwracamy -1. Jeśli nie znaleźliśmy poszukiwanej wartości oraz L ≠ P to zmieniamy zakres wywołując funkcję z odpowiednimi parametrami i zwracając odpowiedni wynik.

```
1 int wyszukaj_pom(double* lista, double szukana, int lewy_index, int prawy_index) {
2     int srodek = (lewy_index + prawy_index) / 2;
3     if(lista[srodek] == szukana)
4         return srodek;
5     if(lowy_index >= prawy_index)
6         return -1;
7     if(lista[srodek] > szukana)
8         return wyszukaj_pom(lista, szukana, lewy_index, srodek - 1);
9     return wyszukaj_pom(lista, szukana, srodek + 1, prawy_index);
10 }
```

Bibliografia

- Edward Morgan Forster „Wykład nr 3. Temat: Wskaźniki i referencje”.
- <http://www.fpga.agh.edu.pl/russek/tul/PDF/bramki.pdf>
- <https://www-users.mat.umk.pl/~zssz/nsi2011/ab.pdf>
- [https://encyklopedia.biolog.pl/index.php?haslo=Referencja_\(informatyka\)](https://encyklopedia.biolog.pl/index.php?haslo=Referencja_(informatyka))
- http://marek.piasecki.staff.iiar.pwr.wroc.pl/dydaktyka/on/W05_wskazniki.pdf
- <https://www.codeit-project.eu/pl/algorithmic-design-recursion/?chapter1>

4. Omówić testowanie aplikacji. Proszę wymienić i krótko scharakteryzować poznane rodzaje testów aplikacji oraz przedstawić jakie aspekty programu mogą podlegać testowaniu.

- Testowanie oprogramowania to proces związany z wytwarzaniem oprogramowania. Jest to część procesów zarządzania jakością oprogramowania. Testowanie ma na celu weryfikację oraz walidację oprogramowania. Weryfikacja oprogramowania pozwala skontrolować, czy wytwarzane oprogramowanie jest zgodne ze specyfikacją. Walidacja sprawdza, czy oprogramowanie jest zgodne z oczekiwaniami użytkownika. Testowanie oprogramowania może być wdrożone w dowolnym momencie wytwarzania oprogramowania (w zależności od stosowanej metody). W podejściu kaskadowym zgodnym z modelem V wysiłek zespołu testerskiego zaczyna się wraz z definicją wymagań i jest kontynuowany po zaimplementowaniu zdefiniowanych wymagań. Nowsze metody wytwarzania oprogramowania (np. metody zwinne) rozkładają wysiłek testerski równomiernie na poszczególne iteracje i skupiają się na testach jednostkowych oraz automatyzacji weryfikacji wykonywanych przez członków zespołu.

Podział rodzajów testowania

Testy aplikacji można podzielić na kilka sposobów.

Poziomy testów oprogramowania – są przeprowadzane na różnych poziomach oprogramowania. Pozwalają znaleźć błędne fragmenty kodu w każdej fazie projektu. Są to:

- Testy jednostkowe: są one przeprowadzane na poziomie aplikacji zbliżonym do kodu źródłowego. Polegają na testowaniu pojedynczych

metod, obiektów modułów lub komponentów. Ich ogromną zaletą jest możliwość pełnej automatyzacji procesu testowania.

- Testy integracyjne: sprawdzają jakość współpracy i komunikacji pomiędzy poszczególnymi modułami aplikacji, takimi jak komunikacja między bazą danych a aplikacją czy między serwerem a aplikacją kliencką.
- Testy systemowe: analizują kompletny, zintegrowany system. Dzięki nim możliwa jest ocena działania całego systemu w kontekście wymagań biznesowych, technicznych, funkcjonalnych czy architektonicznych.
- Testy akceptacyjne: badają system pod kątem realizacji wymagań klienta.

Testy niefunkcjonalne: Testy te sprawdzają niefunkcjonalne atrybuty aplikacji. Określają parametry testowanego oprogramowania. Są to:

- Testy wydajnościowe – określają wydajność systemu. Wykonywane są poprzez uruchomienie jak największej ilości działań, które mocno obciążają testowane oprogramowanie. Sprawdzają one wykorzystanie zasobów, skalowalność i niezawodność aplikacji. Parametrem wynikowym jest tutaj czas odpowiedzi, który determinuje wydajność programu.
- Testy obciążeniowe – Testy obciążenia mają na celu testowanie systemu poprzez ciągłe i równomierne zwiększanie obciążenia systemu, aż do osiągnięcia wartości progowej. Jest to podzbiór testów wydajnościowych.
- Testy użyteczności – sprawdzają oprogramowanie pod kątem intuicyjności i łatwości w obsłudze.
- Testy niezawodności – sprawdzają oprogramowanie pod kątem sprawności funkcjonalności w określonych warunkach.

- Testy funkcjonalne – zajmują się zewnętrznym zachowaniem oprogramowania i nazywane są również czarnoskrzynkowymi są odpowiedzialne za testowanie zabezpieczeń programu oraz jego współdziałania z konkretnymi modułami. W tym wypadku nie jest wymagana znajomości testowanego kodu. Testowane są zwracane dane wyjściowe po wprowadzeniu danych wejściowych.
- Testy regresyjne – testy wykonywane po zakończeniu innych procesów testowych. Wykonywane w celu potwierdzenia poprawnego i sprawnego działania programu.

Aspekty programu podlegające testowaniu:

- Niezawodność aplikacji
- Efektywność działania aplikacji
- Bezpieczeństwo programu
- Skalowalność i przenosalność
- Użyteczność dla użytkownika

(D-06) Omówić testowanie aplikacji. Proszę wymienić i krótko scharakteryzować poznane rodzaje testów aplikacji, oraz przedstawić jakie aspekty programu mogą podlegać testowaniu.

1. Czym jest?

Testowanie aplikacji jest procesem weryfikacji zgodności działania implementacji oprogramowania z określonymi funkcjonalnościami. Testom może zostać poddane zarówno cała aplikacja, jak i również jej wybrane elementy. Jej celem jest wykrycie błędów i niedopatrzeń.

Sprawdzenie poprawności programu w pewnych kontrolnych warunkach/scenariuszach i kontrolowanie czy wyniki są zgodne z oczekiwanyimi.

2. Rodzaje testów ze względu na cel wykonania:

- funkcjonalne: Zajmują się zewnętrznym zachowaniem oprogramowania i nazywane są również czarnoskrzynkowymi (black-box). Są odpowiedzialne za testowanie zabezpieczeń programu oraz jego współdziałania z konkretnymi modułami. Osoba testująca nie zna budowy programu poddanego testom. Głównym przedmiotem testu nie jest budowa wewnętrzna programu, lecz jego założenia funkcjonalne i zewnętrzne interfejsy.

Testy funkcjonalne mają wykrywać błędy implementacji funkcjonalności zawartych w specyfikacji wymagań. Testy czarnej skrzynki mają większą szansę wykrycia błędów, ale jednocześnie nie dostarczają precyzyjnej informacji na temat przyczyny wystąpienia błędu w programie. Do testów funkcjonalnych zaliczyć możemy między innymi testy eksploracyjne.

- niefunkcjonalne: Testy te opierają się na sprawdzaniu atrybutów niefunkcjonalnych. Pozwalają określić parametry testowanego programu i zrozumieć, jak dokładnie on działa. Możemy zaliczyć do nich między innymi testowanie:

- wydajnościowe, które określają wydajność systemu.

- obciążeniowe, pokazuje w jaki sposób oprogramowanie zachowuje się podczas wysokiego obciążenia.

- użyteczności, sprawdza czy aplikacja jest łatwa w obsłudze dla użytkowników,

- niezawodności, czyli testuje oprogramowanie pod kątem sprawności funkcjonalności w określonych warunkach.

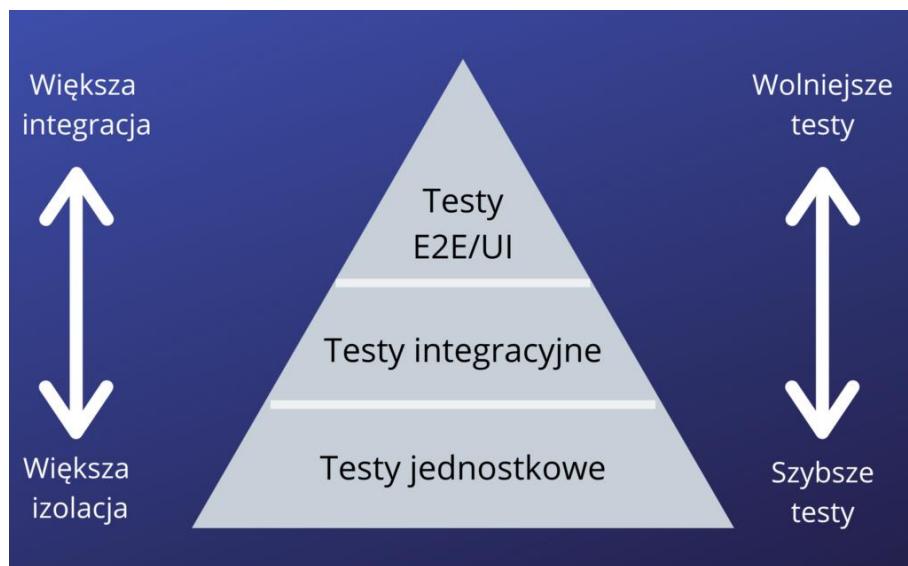
- strukturalne: Testy te nazywane są białośkrzynkowymi (white-box) – Podobnie jak w przypadku testów niefunkcjonalnych i czarnoskrzynkowych, testy strukturalne mogą być przeprowadzane na każdym poziomie testowania. Testy białośkrzynkowe warto przeprowadzić tuż po testowaniu opartym na specyfikacji, aby zwiększyć dokładność pomiarów. Ich zadaniem jest przetestowanie każdej możliwej ścieżki wykonania testowanego komponentu. Upewniamy się, że wszystkie elementy oprogramowania zostały pokryte przez testy.

- potwierdzające (regresywne): Po zakończeniu procesów testowych oraz wykryciu i naprawie usterek należy jeszcze wykonać retest potwierdzający sprawne działanie programu. Tym właśnie jest testowanie regresywne. Polega ono przede wszystkim na ponownym sprawdzeniu oprogramowania pod kątem znalezienia ukrytych wad i błędów. Testy potwierdzające mogą być wykonywane dla wszystkich typów testów i poziomów testowania. Co więcej, ze względu na to, że wymagają powtarzalności, można je łatwo zautomatyzować.

3. Rodzaje testów ze względu na poziom:

- testy jednostkowe: są przeprowadzane na poziomie aplikacji zbliżonym do kodu źródłowego. Polegają na testowaniu pojedynczych metod, obiektów, modułów lub komponentów. Ich ogromną zaletą jest możliwość pełnej automatyzacji procesu testowania oraz znajdowanie błędów natychmiast po ich pojawienniu się. Testy jednostkowe są wykonywane w izolacji. Nie komplikujemy całego systemu, a jedynie potrzebny fragment kodu produkcyjnego. Wykonują się szybko. Pojedynczy test trwa kilka milisekund, a cały zestaw do kilku sekund.
 - testy integracyjne: sprawdzają jakość współpracy pomiędzy poszczególnymi modułami programu. Dzięki nim możemy dowiedzieć się na przykład, czy interakcja między aplikacją a bazą danych jest na właściwym poziomie. Testy integracyjne mogą wykrywać błędy w interfejsach i połączeniach między obiektami. Warto zaznaczyć, że im większy zakres integracji, tym trudniej wskazać miejsce występowania błędu.
- Testy tego typu występują zawsze po testowaniu jednostkowym. Są też bardziej skomplikowane, ponieważ wymagają uruchomienia wielu elementów aplikacji. Testy integracyjne umożliwiają sprawdzenie programu finalnego. Wykonują się dłużej niż testy jednostkowe.
- testy systemowe: analizują kompletny, zintegrowany system. Dzięki nim możliwa jest ocena działania całego systemu w kontekście postawionych wymagań biznesowych, technicznych, funkcjonalnych oraz dotyczących architektury oprogramowania. Testy systemowe przeprowadzane są w środowisku zbliżonym do produkcyjnego. Mogą wyłapać nietrywialne błędy współpracy między modułami, niezrozumienie wymagań, czy problemy specyficzne dla środowiska produkcyjnego.
 - testy akceptacyjne - na tym poziomie system badany jest pod kątem realizacji wymagań klienta. Testy wykonywane przez klienta, które mają dać mu informację potrzebną do podjęcia świadomej decyzji o odbiorze bądź odrzuceniu produktu. Zwykle tego typu testy klient realizuje u siebie, przy wykorzystaniu swoich zasobów ludzkich w postaci testerów, potencjalnych użytkowników systemu, lub z przy współpracy z niezależnym zespołem testerskim. Testy akceptacyjne weryfikują system pod kątem zgodności z założeniami i wymaganiami, które zostały postawione przed fazą developmentu.

4. Piramida testowania



5. Testy end-to-end

Testy E2E to testy, które naśladują zachowanie użytkownika końcowego aplikacji. Tester wciela się w rolę użytkownika końcowego i przechodzi ścieżki, jakimi mógłby się on poruszać. Testują całą funkcjonalność od początku do końca (end-to-end) i wykonywane są w docelowym środowisku produkcyjnym. Sprawdzają, czy wszystkie systemy i ich elementy wspólnie działają poprawnie i nie posiadają błędów. Testy E2E są najbardziej czasochłonne, a co za tym idzie, najdroższe, dlatego w piramidzie znajdują się na samym szczycie. Mogą być zautomatyzowane.

6. Testy eksploracyjne

Testy eksploracyjne to jednocośne uczenie się, projektowanie testów i proces ich wykonywania. Można powiedzieć, że w tych testach planowanie, analiza, projektowanie i wykonywanie testów odbywa się razem i natychmiast.

(D-07) Omówić testy jednostkowe. Proszę wyjaśnić czym jest test jednostkowy, jakie stawiamy mu wymagania, oraz przed-stawić założenia metody Test Driven Development i korzyści z jej stosowania.

1. Czym są?

Testy modułowe wykonujemy na etapie wytwarzania oprogramowania (równocześnie z pracami developerskimi). Zwykle testy te wykonują sami programiści, którzy rozwijają aplikację. Robią to w swoim kodzie w środowisku developerskim – testy te można rozumieć jako „pierwszy front” weryfikacji jakości oprogramowania. Testy jednostkowe w swojej koncepcji skupiają się na weryfikacji małych, wyizolowanych konstrukcji programistycznych, np. fragmentów kodu takich jak pojedyncze funkcje, metody. Mają one za zadanie sprawdzić czy dany fragment kodu realizuje dokładnie zakładaną czynność, logikę. Bardzo ważną cechą testów modułowych jest to, że testowane konstrukcje programistyczne, fragmenty kodu – powinny działać w odizolowaniu od innych funkcjonalności, czyli mówiąc prościej, powinny być niezależne od innych wytwarzonych funkcjonalności.

2. Wymagania testów jednostkowych (F.I.R.S.T)

Zbiór pięciu zasad kryjących się pod akronimem S.O.L.I.D. jest każdemu programiście/programistce dobrze znany. Mniej osób natomiast kojarzy akronim F.I.R.S.T., który określa pięć podstawowych reguł dotyczących wytwarzania testów jednostkowych.

F (Fast) – testy powinny działać szybko. W przypadku, gdy testy jednostkowe będą wykonywały się długo sprawimy że będziemy je uruchamiać rzadziej lub wcale. Im mniejsza częstotliwość uruchamiania testów, tym później dowiadujemy się o błędach i wprowadzamy poprawkę.

I (Independent) – testy powinny być niezależne od siebie. W ciele testu nie powinna następować konfiguracja warunków, która wpływa na wykonanie następnego testu. Jeśli testy są odizolowane od siebie wtedy można je uruchomić w dowolnej kolejności, a wyniki z testów zawsze będą takie same.

R (Repeatable) – testy powinny być powtarzalne w każdym środowisku. Uruchamiając testy na swoim komputerze, na laptopie koleżanki, serwerze buildów powinniśmy za każdym razem otrzymać ten sam rezultat.

S (Self-Validating) – rezultat testu powinien być jednoznaczny (test powiodł się lub nie).

T (Timely) – testy powinny być pisane bezpośrednio przed tworzeniem testowanego kodu produkcyjnego.

3. Dobre praktyki

Ustalenie zasad z zespołem odpowiednich praktyk takich jak:

- nazewnictwo metod testowych np. should..when,
- wzorzec Arrange-Act-Assert (AAA) i Given-When-Then (GWT),
- klasie z kodem produkcyjnym odpowiada jedna klasa z testami jednostkowymi,
- testy jednostkowe nie powinny zawierać instrukcji warunkowych i pętli,
- w testach wyłapujemy jak najbardziej szczegółowe wyjątki,

- testujemy tylko publiczne metody.

4. Korzyści testów jednostkowych

- tanie,

- szybkie,

- sprawdzać nasz kod w izolowany sposób,

- pozwalają na wykrycie błędów i defektów bardzo szybko,

- weryfikacja działania bez uruchamiania systemu.

5. Metoda TDD – Test Driven Development

TDD (Test-Driven Development) jest techniką tworzenia oprogramowania (nie jest to technika pisania testów), w której główną ideą jest w pierwszej kolejności pisanie testów do nieistniejącej funkcjonalności, a dopiero potem napisanie kodu implementującego tę funkcjonalność. TDD nie jest jedyną słuszną i dobrą techniką tworzenia programowania. Można pisać testy równolegle z pisaniem kodu logiki biznesowej, można je pisać po implementacji, lecz wtedy nie jest to już TDD. W Test-Driven Development testy piszemy zawsze przed implementacją.

Cykl życia TDD:

1. Napisanie testu.
2. Uruchomienie napisanego testu.
3. Napisanie minimalnego kodu do przejścia testu
4. Refaktoryzacja kodu
5. Powtarzamy cykl.

Zalety TDD:

- większość błędów zostanie wykrytych podczas testów przed wprowadzeniem kodu w środowisko testowe lub produkcyjne,

- duże pokrycie kodu testami (możliwe, że nawet 100%),

- pisanie tylko potrzebnego kodu,

- łatwiejsze do utrzymania,

- łatwiejsze do refaktoryzacji,

- testy dokumentują kod.

4.Omówić wzorce projektowe. Proszę podać przykład i zdefiniować pojęcie wzorca projektowego, oraz wyjaśnić jakie korzyści płyną ze stosowania wzorów projektowych.

Wzorzec projektowy (ang. design pattern) – uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. Pokazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz utrzymanie kodu źródłowego. Jest opisem rozwiązania, a nie jego implementacją. Wzorce projektowe stosowane są w projektach wykorzystujących programowanie obiektowe.

Wzorzec MVC (od angielskich słów Model, View i Controller) opracowano pod koniec lat 70 jako wzorzec realizacji interfejsu użytkownika w języku Smalltalk-80. Następnie wykorzystano go również w innych językach i systemach zorientowanych obiektywnie.

Trzy składowe tego modelu to:

Model - Główny nacisk kładzie na logikę aplikacji i logikę biznesową. Model powinien być zaprojektowany w taki sposób, aby był niezależny od wybranego rodzaju prezentacji oraz systemu obsługi akcji użytkownika.

Widok - zarządza graficzną lub tekstową prezentacją modelu. Widok pobiera informacje z modelu, ilekroć zostaje powiadomiony o jego zmianie.

Kontroler - jest odpowiedzialny za reagowanie na akcji użytkownika (np. kliknięcia myszką), odwzorowując je na operacje zawarte w modelu oraz na zmiany widoku.

Elementy wzorca to:

Wzorzec projektowy składa się z czterech podstawowych elementów:

- **nazwy wzorca;**

- **problemu** – opisuje sposoby rozpoznawania sytuacji, w których możemy zastosować dany wzorzec oraz warunki jakie muszą zostać spełnione, by jego zastosowanie miało sens;

- **rozwiązań** – opisuje elementy rozwiązania: ich relacje, powiązania oraz obowiązki, zawiera także wskazówki implementacyjne dla różnych technologii;

- **konsekwencji** – zestawienie wad i zalet stosowania wzorca, uwzględniające informacje o jego brakach oraz kosztach rozwoju i utrzymania systemu wykorzystującego dany wzorzec.

Korzyści:

Wzorce projektowe mogą przyspieszyć proces rozwoju oprogramowania przez

dostarczenie wypróbowanych rozwiązań dla problemów, które mogą nie być oczywiste na początku procesu projektowego. Często zagadnienia te wiążą się z ewolucją oczekiwania względem projektowanego systemu: rozszerzeniem jego funkcjonalności, zmianą sposobu i formatu wprowadzanych danych czy dostosowaniem aplikacji do różnych klas użytkowników. Nieuwzględnienie ich na początku procesu rozwoju produktu programistycznego powoduje często konieczność gruntownego przebudowywania zaawansowanego lub gotowego już oprogramowania.

(D-10) Omówić sieci neuronowe. Proszę przedstawić i omówić budowę neuronu w sztucznej sieci neuronowej, oraz wymienić i omówić metody uczenia sztucznej sieci neuronowych.

W skrócie

Sieci neuronowe – neurony połączone synapsami. 3 warstwy: wejściowa, ukryta, wyjściowa. Wyjścia to odpowiedniki dendrytów.

Funkcja aktywacji: skok jednostkowego, liniowa, nieliniowa

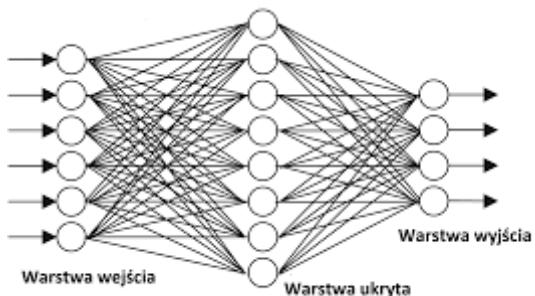
Neuron: wejścia, wagi, blok sumujący, blok aktywacji, wyjście

Metody uczenia sieci neuronowych: uczenie nadzorowane, uczenie nienadzorowane, uczenie przez wzmacnianie.

Podział ze względu na kierunek przesyłu danych: sieci jednokierunkowe, sieci rekurencyjne, samoorganizujące się mapy.

Sieci neuronowe

Sieci neuronowe - struktury składające się z neuronów połączonych synapsami. Sztuczne sieci neuronowe składają się z trzech typów warstw: wejściowej (zbiera dane i przekazuje je dalej), ukrytej (tu szukane są powiązania między neuronami, czyli zachodzi proces uczenia się) i wyjściowej (gromadzi wnioski, wyniki analizy). Sieć neuronowa może składać się z dowolnej liczby warstw. W technologii informacyjnej (IT) sieć neuronowa to sprzęt lub oprogramowanie (może być jedno i drugie) wzorowane na działaniu neuronów w ludzkim mózgu. Często są jedynym rozwiązaniem dla matematycznego opisania obiektów nieliniowych.



Wejścia to odpowiedniki dendrytów, lub ściślej: sygnały przez nie nadchodzące. Wagi to cyfrowe odpowiedniki modyfikacji dokonywanych na sygnałach przez synapsy. Blok sumujący to odpowiednik jądra, blok aktywacji to wzgórek aksonu, a wyjście - to akson. Oczywiście liczba wejść nie musi wynosić trzy, jest ona dowolna.

Proces przetwarzania sygnału w sztucznym neuronie w sposób ogólny można przedstawić następująco:

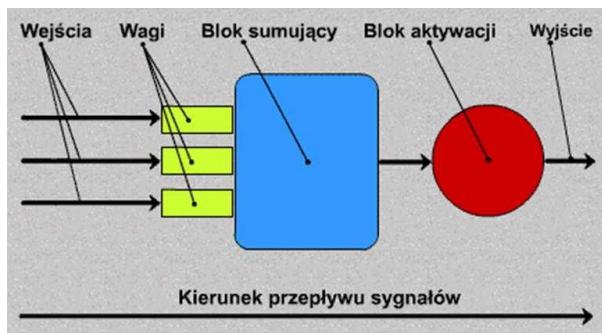
Wejścia dostarczają sygnał, który następnie jest mnożony przez współczynniki wag, następnie w bloku sumowania następuje sumowanie pomnożonych sygnałów. Wynikiem tego otrzymujemy sygnał zwany potencjałem membranowym. Następnie sygnał przetworzony zostaje w bloku aktywacji, który w zależności od potrzeb może być opisany różnymi funkcjami - zwany funkcjami aktywacji. Wartość funkcji aktywacji jest sygnałem wyjściowym neuronu i propagowana jest do neuronów warstwy następnej. Funkcja aktywacji przybiera jedną z trzech postaci:

- skoku jednostkowego - tzw. funkcja progowa

- liniowa
- nielinowa

Podział ze względu na charakter rezultatu:

- Sieci klasyfikacyjne
- Sieci regresyjne



Wartości wejściowe - wartości danych pierwotnych podawanych z zewnątrz lub sygnały pośrednie (w innej warstwie niż wejściowa).

Współczynniki wagowe - odpowiednia waga (siła) wejścia.

Wartość progowa - pojedyncza wartość progowa, która określa jak silne musi być pobudzenie, aby doszło do zapłonu/aktywizacji neuronu.

Blok sumujący - w neuronie dochodzi do obliczenia sumy wartości wejściowych poprzedzanych przez odpowiednie współczynniki wagowe, a następnie odejmowania jest od niej wartość progowa. Ważona suma wejść określa pobudzenie neuronu.

Funkcja aktywacji - Funkcja wykorzystywana do transformacji poziomu aktywacji jednostki (neuronu) w sygnał wyjściowy. Zachowanie neuronu i całej sieci neuronowej jest silnie uzależnione od rodzaju użytej funkcji aktywacji.

Metody uczenia sieci neuronowych

- uczenie nadzorowane (ang. supervised learning), to takie, kiedy zbiór danych dostarczany maszynie do nauki zawiera również oczekiwana odpowiedź. Na przykład zdjęcia różnych kwiatków, a do tego nazwa każdego z nich. Albo zestaw maili z informacją, który z nich to spam a który nie. Dzięki takiemu nauczaniu oczekujemy, że po pokazaniu zdjęcia, którego wcześniej nie było w zbiorze danych, dowiemy się, jaki jest to kwiatek (wcześniej musiały być inne zdjęcia tego kwiatka, żeby komputer miał się skąd tego nauczyć). A nowy mail trafi albo do skrzynki odbiorczej albo do spamu.
- uczenie nienadzorowane (ang. unsupervised learning), to takie, kiedy nie dostarczamy żadnych odpowiedzi, tylko zestaw danych. Na przykład dostarczamy zdjęcia różnych kwiatków, ale nie mamy na ich temat żadnych więcej informacji. Oczekujemy, że zostaną podzielone na jakieś grupy i każde nowe zdjęcie trafi do grupy, gdzie znajdują się kwiatki do niego podobne. Co to znaczy „podobne”? Wybór należy do maszyny uczącej się. Zwykle na początku podaje się informację, na ile grup byśmy chcieli podzielić nasze dane.

- uczenie przez wzmacnianie (ang. reinforcement learning) to takie, kiedy system działa w środowisku zupełnie nieznanym. Brak jest zarówno określonych danych wejściowych jak i wyjściowych. Jedyną informacją, jaką otrzymuje maszyna ucząca się jest tzw. sygnał wzmacnienia. Sygnał ten może być albo pozytywny (nagroda) albo negatywny (kara). Metodę tę można inaczej nazwać metodą prób i błędów. Przykładem może być gra w nową grę, której reguły nie znamy. Po skończonej grze dowiadujemy się, czy wygraliśmy, czy przegraliśmy (nagroda/kara). W kolejnych grach powinno iść coraz lepiej.

Podział ze względu na kierunek przesyłu danych

Cechą wspólną wszystkich sieci neuronowych jest to, że na ich strukturę składają się neurony połączone ze sobą synapsami. Z synapsami związane są wagи, czyli wartości liczbowe, których interpretacja zależy od modelu.

- Sieci jednokierunkowe.
- Sieci rekurencyjne.
- Samoorganizujące się mapy.

Sieci jednokierunkowe to sieci neuronowe, w których nie występuje sprzężenie zwrotne, czyli pojedynczy wzorzec lub sygnał przechodzi przez każdy neuron dokładnie raz w swoim cyklu. Najprostszą siecią neuronową jest pojedynczy perceptron progowy, opracowany przez McCullocha i Pittsa w roku 1943.

W bardziej zaawansowanych rozwiązaniach stosuje się funkcje przejścia. Najpopularniejszą klasę funkcji stosowanych w sieciach neuronowych stanowią funkcje sigmoidalne, np. tangens hiperboliczny. Sieć zbudowana z neuronów wyposażonych w nieliniową funkcję przejścia ma zdolność nieliniowej separacji wzorców wejściowych. Jest więc uniwersalnym klasyfikatorem.

Do uczenia perceptronów wielowarstwowych stosuje się algorytmy spadku gradientowego, między innymi algorytm propagacji wstecznej.

Sieci jednokierunkowe dzielą się na jednowarstwowe, dwuwarstwowe i wielowarstwowe. Sieci jednowarstwowe mogą rozwiązać jedynie wąską klasę problemów. Sieci dwu i wielowarstwowe mogą rozwiązać znacznie szerszą klasę i są pod tym względem równoważne, jednak stosuje się do nich inne algorytmy uczenia (dla wielowarstwowych są one prostsze).

Mianem **sieci rekurencyjnej** określa się sieć, w której połączenia między neuronami stanowią graf z cyklami. Wśród różnorodności modeli rekurencyjnych sztucznych sieci neuronowych wyróżnić można:

- sieć Hopfielda – układ gęsto połączonych ze sobą neuronów (każdy z każdym, ale bez połączeń zwrotnych) realizującą dynamikę gwarantującą zbieżność do preferowanych wzorców
- maszyna Boltzmanna – opracowana przez Geoffa Hintona i Terry'ego Sejnowskiego stochastyczna modyfikacja sieci Hopfielda; modyfikacja ta pozwoliła na uczenie neuronów ukrytych i likwidację wzorców pasożytniczych kosztem zwiększenia czasu symulacji.
- Sieci Hopfielda i maszyny Boltzmanna stosuje się jako pamięci adresowane kontekstowo, do rozpoznawania obrazów, rozpoznawania mowy, a także do rozwiązywania problemów minimalizacji (np. problemu komiwojażera)

Samoorganizujące się mapy (Self Organizing Maps, SOM), zwane też sieciami Kohonena, to sieci neuronów, z którymi są stowarzyszone współrzędne na prostej, płaszczyźnie lub w dowolnej n-wymiarowej przestrzeni.

Uczenie tego rodzaju sieci polega na zmianach współrzędnych neuronów, tak, by dążyły one do wzorca zgodnego ze strukturą analizowanych danych. Sieci zatem „rozpinają się” wokół zbiorów danych, dopasowując do nich swoją strukturę.

Sieci te stosowane są do klasyfikacji wzorców, np. głosek mowy ciągłe, tekstu, muzyki. Do najciekawszych zastosowań należy rozpinanie siatki wokół komputerowego modelu skanowanego obiektu.