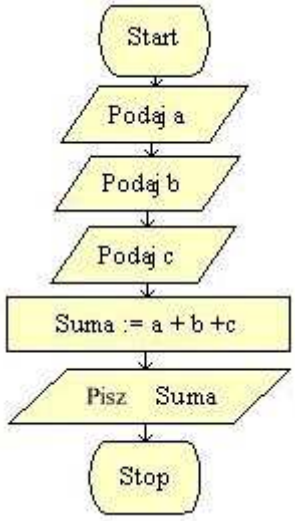
1. Algorytm. Własności algorytmu. Prezentacja algorytmu.



Algorytm — jednoznacznie sformułowany sposób postępowania, który w skończonej liczbie kroków umożliwia rozwiązanie zadania określonej klasy. Zakodowany w wybranym języku programowania zamienia się w program komputerowy.

Każdy algorytm:

- Posiada dane wejściowe pochodzące z dobrze zdefiniowanego zbioru
- Produkuje pewien wynik
- Każdy krok algorytmu musi być jednoznacznie określony
- Jest skończony
- Powinien być efektywny tzn. wykonywać swoją pracę w jak najkrótszym czasie i wykorzystując jak najmniejszą ilość pamięci
- Powinien dawać poprawne wyniki

2. Złożoność algorytmów.

Złożoność algorytmu rozumiemy jako ilość niezbędnych zasobów do wykonania algorytmu. Złożoność dzielimy na czasową oraz pamięciową.

W przypadku złożoności czasowej, z reguły wyróżnimy pewną operację dominującą, a czas będziemy traktować jako liczbę wykonanych operacji dominujących. Złożoność algorytmu może być rozumiana w sensie złożoności najgorszego przypadku lub złożoności średniej. Złożoność najgorszego przypadku nazywamy złożonością pesymistyczną - jest to maksymalna złożoność dla danych tego samego rozmiaru n. $g(n) = n$ liniowa, $g(n) = n^2$ kwadratowa, gdy $g(n)$ jest wielomianem to złożoność wielomianowa.

Złożoność pamięciowa jest miarą ilości wykorzystanej pamięci. Jako tę ilość najczęściej przyjmuje się użytą pamięć maszyny abstrakcyjnej. Możliwe jest również obliczanie rozmiaru potrzebnej pamięci fizycznej wyrażonej w bitach lub bajtach.

3. Problemy sortowania. Przykłady algorytmów sortowania i ich złożoność.

Sortowanie – polega na uporządkowaniu zbioru danych względem pewnych cech charakterystycznych każdego elementu tego zbioru.

Złożoność algorytmów sortowania jest określana jako liczba wykonywanych porównań i zamian elementów, wyrażona w zależności od liczby porządkowanych elementów. Złożoność najlepszych algorytmów sortowania jest proporcjonalna do $n\log_2 n$, gdzie n jest liczbą porządkowanych elementów.

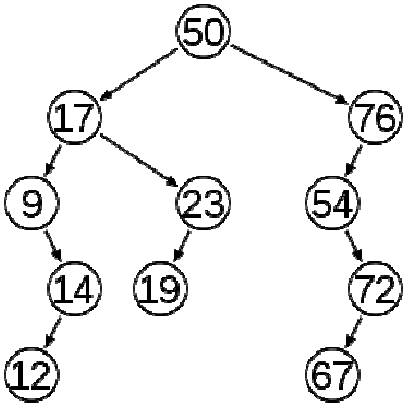
Przykładowe algorytmy sortowania

- **sortowanie bąbelkowe (ang. *bubblesort*)** **$O(n^2)$**
- sortowanie przez wstawianie (ang. *insertion sort*) $O(n^2)$
- sortowanie przez scalanie (ang. *merge sort*) $O(n\log n)$
- sortowanie przez wybieranie (ang. selection sort) $O(n^2)$

<u>Sortowanie bąbelkowe</u>

Polega na porównywaniu dwóch kolejnych elementów i zamianie ich kolejności, jeżeli zaburza ona porządek, w jakim się sortuje tablicę. Sortowanie kończy się, gdy podczas kolejnego przejścia nie dokonano żadnej zmiany.

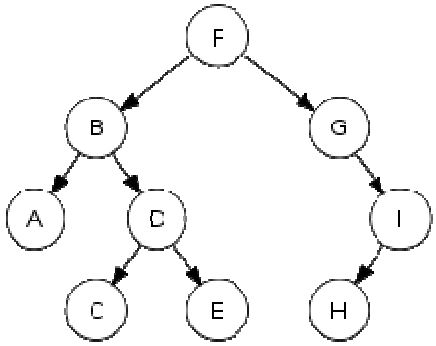
4. Drzewa przeszukiwań binarnych. Sposób wykonywania na nich podstawowych operacji (dodawanie, wyszukiwanie, usuwanie).



Drzewo przeszukiwań binarnych (BST- binary search tree)
Oprócz pola wartości drzewo przeszukiwań binarnych posiada jeszcze dwa pola: **L** i **P**, wskazujące odpowiednio na lewy i prawy następnik.

- element **w lewej gałęzi to jest mniejszy**
- element **w prawej gałęzi to jest większy**
- wierzchołki znajdujące się bezpośrednio pod danym węzłem nazywamy synami (lub dziećmi).
- wierzchołki, które nie mają potomków nazywane są liśćmi.

Przeszukiwanie drzewa:



Wszerz:

Kolejność: F B G A D I C E H

W głąb:

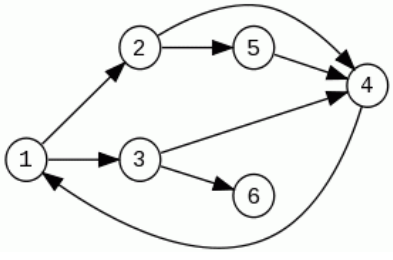
- *Preorder*: korzeń, lewe poddrzewo, prawe poddrzewo. F B A D C E G I H
- *Inorder*: lewe poddrzewo, korzeń, prawe poddrzewo. A B C D E F G H I
- *Postorder*: lewe poddrzewo, prawe poddrzewo, korzeń. A C E D B H I G F

Usuwanie elementów:
Jeżeli usuwany element nie ma następników to można zwolnić przydzieloną mu pamięć. Jeśli element do usunięcia ma jeden następnik należy go połączyć (następnik) z poprzednikiem usuwanego elementu. Jeśli element ma dwa następniki można go usunąć na dwa sposoby:

- połączyć poprzednik elementu z wierzchołkiem o najmniejszej wartości z prawego poddrzewa usuwanego
- połączyć poprzednik elementu z wierzchołkiem o największej wartości z lewego poddrzewa

5. Algorytmy przeszukiwania grafu.

Przeszukiwanie grafu to odwiedzenie w usystematyzowany sposób wszystkich wierzchołków grafu w celu zebrania potrzebnych informacji.
Powszechnie stosuje się dwie podstawowe metody przeszukiwania grafów:



przeszukiwanie wszerz (BFS) - (ang. Breadth-first search, w skrócie BFS) – Algorytm zaczyna od korzenia i odwiedza wszystkie połączone z nim węzły. Następnie odwiedza węzły połączone z tymi węzłami i tak dalej, aż do odnalezienia celu.
przeszukiwanie w głąb (DFS) - (ang. Depth-first search, w skrócie DFS) – Przeszukiwanie zaczyna się od korzenia i porusza się w dół do samego końca gałęzi, po czym wraca się o jeden poziom i próbuje kolejne gałęzie itd.

Przeszukiwanie w głąb (DFS):

Zaczynamy od wierzchołka 1, odwiedzamy go i wrzucamy na stos wszystkie jego następniki, w kolejności od tego z największym indeksem:

Odwiedzone: 1; Stos: 3, 2;

Bierzemy wierzchołek ze stosu, odwiedzamy go i znów wrzucamy wszystkie jego nieodwiedzone jeszcze następniki na stos:

Odwiedzone: 1, 2; Stos: 3, 5, 4;
Odwiedzone: 1, 2, 4; Stos: 3, 5;
Odwiedzone: 1, 2, 4, 5; Stos: 3;
Odwiedzone: 1, 2, 4, 5, 3; Stos: 6;
Odwiedzone: 1, 2, 4, 5, 3, 6; Stos: ;

Stos jest pusty zatem zakończyliśmy przeszukiwanie grafu w głąb.

Przeszukiwanie wszerz (BFS):

Zaczynamy od wierzchołka 1, odwiedzamy go i wrzucamy do kolejki wszystkie jego następniki, w kolejności od tego z najmniejszym indeksem:

Odwiedzone: 1; Kolejka: 2, 3;

Bierzemy wierzchołek z kolejki, odwiedzamy go i znów wrzucamy wszystkie jego nieodwiedzone jeszcze następniki do kolejki:

Odwiedzone: 1, 2; Kolejka: 3, 4, 5;
Odwiedzone: 1, 2, 3; Kolejka: 4, 5, 6;
Odwiedzone: 1, 2, 3, 4; Kolejka: 5, 6;
Odwiedzone: 1, 2, 3, 4, 5; Kolejka: 6;
Odwiedzone: 1, 2, 3, 4, 5, 6; Kolejka: ;

Kolejka jest pusta zatem zakończyliśmy przeszukiwanie grafu w szerz.

6. Abstrakcyjne struktury danych: listy, kolejki, stosy. Ich implementacja komputerowa.

Stos

Liniowa struktura danych, w której dane dokładane są na wierzch stosu i z wierzchołka stosu są pobierane (bufor typu **LIFO**, *Last In, First Out*; *ostatni na wejściu, pierwszy na wyjściu*).

Implementacja Strukturami danych służącymi do reprezentacji stosu mogą być tablice (gdy znamy maksymalny rozmiar stosu), tablice dynamiczne lub listy.

Listy

Dynamiczna struktura danych składająca się z podstruktur wskazujących na następniki i/lub poprzedniki. Typowa lista jest łączona **jednostronnie** - komórki zawierają tylko odnośnik do kolejnej komórki. Innym przypadkiem jest lista **dwustronna**, gdzie komórki zawierają także odnośnik do poprzednika.

Implementacja listy

Tablicowa Jak wskazuje nazwa, lista zaimplementowana w ten sposób opiera się na tablicy obiektów (lub rekordów) danego typu.

Wskaźnikowa W tej implementacji każdy obiekt na liście musi (co nie było konieczne w wersji tablicowej) zawierać dodatkowy element: wskaźnik do innego obiektu tego typu. Wynika to z faktu, że to wskaźniki są podstawą nawigacji w tym typie listy, a dostęp do jej elementów jest możliwy wyłącznie przez wskaźnik.

Kolejki

Dane dopisywane są na końcu kolejki, a z początku kolejki pobierane są dane do dalszego przetwarzania (bufor typu **FIFO**, *First In, First Out*; *pierwszy na wejściu, pierwszy na wyjściu*).

Implementacja tablicowa - Jak wskazuje nazwa, kolejka zaimplementowana w ten sposób opiera się na tablicy obiektów (lub rekordów) danego typu.

http://tnij.org/stosy_kolejki

7. Zasada działania i sposoby implementacji tablic haszowanych.

Idea

Tablica mieszająca to jeden ze sposobów realizacji tablicy asocjacyjnej, tj. struktury danych służącej do przechowywania informacji, w taki sposób aby możliwy był do nich szybki dostęp. Stosuje się je w sytuacjach, w których kolejność kluczy nie ma znaczenia, natomiast istotna jest jak największa efektywność przeszukiwania.

Zasada działania Tablice mieszające opierają się na zwykłych tablicach indeksowanych liczbami - dostęp do danych jest bardzo szybki, nie zależy od rozmiaru tablicy ani położenia elementu. W tablicy mieszającej stosuje się funkcję mieszającą (funkcję skrótu, funkcję haszującą), która dla danego klucza wyznacza indeks w tablicy.

1. Funkcja skrótu:

Dzielenie: h(x) = x mod m zbiór kluczy składa się z liczb naturalnych, a m jest rozmiarem tablicy haszującej. Zakładając, że tablica haszująca ma 10 elementów, 149 powinno się znaleźć na pozycji dziewiątej, bo 149 mod 10 = 9.

Wycinanie: do obliczenia adresu wykorzystuje się tylko część klucza. Może to być np. fragment środkowy, albo ostatnie cztery pozycje, albo konkretnie wskazane pozycje. Na ogół pomijamy te fragmenty klucza, które nie rozróżniają dostatecznie dobrze elementów.
Na przykład, gdybyśmy chcieli zapamiętać numery telefonów stacjonarnych, to nie ma sensu brać pod uwagę prefiksu 89, gdyż jest on identyczny dla wszystkich przypadków.

2. Problem kolizji:

Wartość funkcji mieszającej obliczonej dla klucza elementu wstawianego do tablicy pokrywa się z wartością funkcji obliczoną dla klucza jakiegoś elementu już znajdującego się w tej tablicy.

Metoda łańcuchowa - Polega na przechowywaniu elementów nie bezpośrednio w tablicy, lecz na liście związanej z danym indeksem. Wstawiane elementy dołącza się do jednego z końców listy.

Adresowanie otwarte - W podejściu tym nowy element wstawia się w innym miejscu niż wynikałoby to z wartości funkcji mieszającej. Nowa lokalizacja określana jest przez dodanie do wartości funkcji mieszającej wartości tzw. **funkcji przyrostu** $p(i)$, gdzie i oznacza numer próby (to znaczy ile razy wstawienie się nie powiodło ze względu na kolizję).

8. Idea algorytmu zachłannego. Przykład.

Algorytm zachłanny w celu wyznaczenia rozwiązania w każdym kroku dokonuje zachłannego, tj. najlepiej rokującego w danym momencie wyboru rozwiązania częściowego. Innymi słowy algorytm zachłanny nie patrzy czy w kolejnych krokach jest sens wykonywać dane działanie, dokonuje decyzji lokalnie optymalnej, dokonuje on wyboru wydającego się w danej chwili najlepszym, kontynuując rozwiązanie podproblemu wynikającego z podjętej decyzji.

Istnieje wiele problemów, dla których udowodnić można, że rozwiązanie zachłanne jest zawsze optymalne - między innymi znajdowanie minimalnego drzewa rozpinającego, czy znajdowanie najkrótszej ścieżki między dwoma punktami w grafie.

Przykład – Algorytm Kruskala - wyznacza minimalne drzewo rozpinające dla grafu nieskierowanego ważonego, o ile jest on spójny. Innymi słowy, znajduje drzewo zawierające wszystkie wierzchołki grafu, których waga jest najmniej możliwa.

9. Idea Dijkstry algorytmu znajdowania najkrótszej ścieżki.

Algorytm Dijkstry, służy do znajdowania najkrótszej ścieżki z pojedynczego źródła w grafie o nieujemnych wagach krawędzi. Jest często używany w trasowaniu. Mając dany graf z wyróżnionym wierzchołkiem (*źródłem*) algorytm znajduje odległości od źródła do wszystkich pozostałych wierzchołków. Łatwo zmodyfikować go tak, aby szukał wyłącznie ścieżki do jednego ustalonego wierzchołka, po prostu przerywając działanie w momencie dojścia do wierzchołka docelowego.

http://tnij.org/dijkstra_gif

10. Pozycyjne systemy liczbowe (binarny, dziesiętny, szesnastkowy). Prezentacja liczb w komputerze.

Posiadają symbole (cyfry) tylko dla kilku najmniejszych liczb naturalnych: 0, 1, 2, ..., g-1, gdzie g to tzw. podstawa systemu. Cyfry te są umieszczane na ściśle określonych pozycjach i są mnożone przez odpowiednią potęgę g.

DEC - System dziesiętny podstawą pozycji są kolejne potęgi liczby 10. Do zapisu liczb potrzebne jest więc w nim 10 cyfr: 0, 1, 2, 3, 4, 5, 6, 7, 8, 9.

zapis "5045,7" wynika z:

$$5\times10^3+0\times10^2+4\times10^1+5\times10^0+7\times10^{-1} = 5000+0+40+5+0,7 = 5045,7$$

Pozycyjny, dziesiętny system liczbowy jest obecnie na świecie podstawowym systemem stosowanym niemal we wszystkich krajach.

BIN - System binarny (dwójkowy) podstawą jest liczba 2. Do zapisu liczb potrzebne są więc tylko dwie cyfry: 0 i 1.

Powszechnie używany w elektronice cyfrowej, przyjął się też w informatyce.

liczba zapisana w dziesiętnym systemie liczbowym jako 10, w systemie dwójkowym przybiera postać 1010

$$1 \cdot 2^3 + 0 \cdot 2^2 + 1 \cdot 2^1 + 0 \cdot 2^0 = 8 + 2 = 10.$$

HEX - System heksadecymalny (szesnastkowy)

Szesnastkowy system – podstawą jest liczba 16.

W najpowszechniejszym standardzie poza cyframi dziesiętnymi od 0 do 9 używa się pierwszych sześciu liter alfabetu łacińskiego: A, B, C, D, E, F (dużych lub małych). Cyfry 0-9 mają te same wartości co w systemie dziesiętnym, natomiast litery odpowiadają następującym wartościom: A = 10, B = 11, C = 12, D = 13, E = 14 oraz F = 15.

Liczba zapisana w dziesiętnym systemie liczbowym jako 1000, w systemie szesnastkowym przybiera postać 3E8, gdyż:

$$3 \times 16^2 + 14 \times 16^1 + 8 \times 16^0 = 768 + 224 + 8 = 1000$$

System szesnastkowy sprawdza się szczególnie przy zapisie dużych liczb takich jak adresy pamięci, zakresy parametrów itp.

11. Budowa komputera

Komputer to urządzenie elektroniczne służące do przetwarzania wszelkich informacji, które da się zapisać w formie ciągu cyfr albo sygnału ciągłego.

Architektura stworzonej w 1945 przez Johna von Neumanna. Dzieliła ona komputer na trzy części:

- procesor (w ramach którego wydzielona bywa część sterująca oraz część arytmetyczno-logiczna)
- pamięć komputera (zawierająca dane i sam program)
- urządzenia wejścia/wyjścia

Większość współczesnych komputerów składa się z trzech podstawowych elementów:

- procesora – podzielonego na część arytmetyczno-logiczną czyli układu, który faktycznie wykonuje wszystkie konieczne obliczenia oraz część sterującą
- pamięci RAM – (od ang. Random Access Memory) czyli układy scalone, które przechowują program i dane (umożliwia to m.in. samo modyfikację programu) oraz bieżące wyniki obliczeń procesora i stale, na bieżąco wymienia dane z procesorem
- urządzeń wejścia/wyjścia – które służą do komunikacji komputera z otoczeniem.

12. Struktura procesora.

W funkcjonalnej strukturze procesora można wyróżnić takie elementy, jak:

- bardzo szybką pamięć podręczną (cache)
- zespół rejestrów do przechowywania danych i wyników, rejestry mogą być ogólnego przeznaczenia lub mają specjalne przeznaczenie,
- jednostkę arytmetyczną do wykonywania operacji obliczeniowych na danych,
- układ sterujący przebiegiem wykonywania programu,
- inne układy, w które producent wyposaża procesor w celu usprawnienia jego pracy.

Jednym z parametrów procesora jest rozmiar elementów budujących jego strukturę. Im są one mniejsze, tym niższe jest zużycie energii, napięcie pracy oraz wyższa możliwa do osiągnięcia częstotliwość pracy.

13. Cykl rozkazowy procesora

Cykl rozkazowy procesora to pewien schemat powtarzających się czynności, realizujących wykonywanie instrukcji programu. Na cykl rozkazowy składają się dwie fazy:

- faza pobrania,
- faza wykonania.

W fazie pobrania procesor pobiera kod rozkazu z komórki pamięci. Adres pod którym znajduje się rozkaz przechowywany jest w liczniku rozkazów. Wartość tego licznika wysyłana jest na magistralę adresową a licznik rozkazów zostaje zwiększony tak, aby wskazywał na następny rozkaz przeznaczony do wykonania.
Po fazie pobrania następuje **faza wykonania**, w której procesor dekoduje rozkaz i określa jego typ. Układ sterowania generuje odpowiednie sygnały sterujące, realizujące dany rozkaz. Jeżeli rozkaz wymaga pobrania argumentów, to przed jego wykonaniem są one pobierane z pamięci i umieszczane w odpowiednich rejestrach.

14. Metody odwzorowania pamięci głównej w pamięci podręcznej.

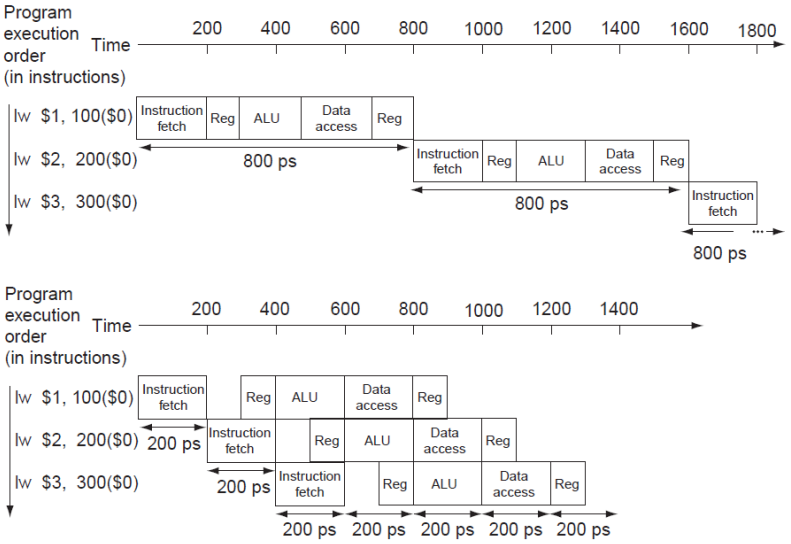
Odwzorowanie bezpośrednie - najprostsza metoda, polega na odwzorowywaniu każdego bloku pamięci głównej na tylko jeden możliwy wiersz pamięci podręcznej.

Odwzorowywanie skojarzeniowe umożliwia ładowanie każdego bloku pamięci głównej do dowolnego wiersza pamięci podręcznej. W tym przypadku sterujące układy logiczne pamięci podręcznej interpretują adres pamięci po prostu jako **znacznik i pole słowa**. Pole znacznika jednoznacznie określa blok pamięci głównej.

Odwzorowywanie sekcyjno-skojarzeniowe pamięć podręczna jest dzielona na v sekcji, z których każda składa się z k wierszy. W przypadku odwzorowywania sekcyjno-skojarzeniowego blok Bj może być odwzorowywany na dowolny wiersz sekcji i. W tym przypadku sterujące układy logiczne pamięci podręcznej interpretują adres pamięci jako trzy pola: **znacznik, sekcja i słowo.**

15. Przetwarzanie potokowe.

Przetwarzanie potokowe — (Używanie w celu zwiększenia wydajności) sposób pracy procesora polegający na wykonywaniu kolejnych instrukcji, przy czym rozpoczęcie następnej nie wymaga zakończenia poprzedniej. Dzięki temu, że przetwarzanie odbywa się w rozdzielnych blokach system może przetwarzać jednocześnie tyle danych ile zdefiniowano bloków.



Wadą rozwiązania jest dłuższy czas oczekiwania na zakończenie pierwszej operacji po chwili przestoju, gdyż dla potoku o długości n etapów minimalny czas oczekiwania wynosi n cykli zegarowych. Jedynym sposobem ograniczenia tego problemu jest zwiększanie częstotliwości taktowania układu oraz likwidowanie przerw w przetwarzaniu.

16. Porównanie różnych architektur sieci komputerowych

Architektura sieci – określa sposoby realizacji przekazu informacji pomiędzy urządzeniami końcowymi. Przeważnie organizowana warstwowo tworząc warstwową architekturę sieci.

Ważniejsze architektury warstwowe:

ISO-OSI (ISO - Open Systems Interconection), TCP/IP (Transmission Control Protocol/Internet Protocol)
SNA (Systems Network Architecture) firmy IBM, DNA (Digital Network Architecture) firmy DEC

Model OSI dwa systemy mogą wymieniać informacje między sobą pod warunkiem, że w obu przypadkach zaimplementowano te same protokoły komunikacyjne.

Warstwy modelu ISO/OSI:

1 APLIKACJI - zajmuje się specyfikacją interfejsu, który wykorzystują aplikacje do przesyłania danych do sieci;

2 PREZENTACJI - zapewnia tłumaczenie danych, definiowanie ich formatu oraz odpowiednią składnię;

3 SESJI - zadaniem jest zarządzanie przebiegiem komunikacji, podczas połączenia miedzy dwoma komputerami;

4 TRANSPORTOWA - segmentuje dane oraz składa je w tzw. strumień, zapewnia całościowe połączenie między stacjami;

5 SIECIOWA - podstawowe zadania to przesyłanie danych pomiędzy węzłami sieci wraz z wyznaczaniem trasy przesyłu, łączenie bloków informacji w ramki na czas ich przesyłania a następnie stosowny ich podział;

6 ŁĄCZA DANYCH - odpowiada za obsługę ramek, czyli stałej długości pakietów danych, do jej zadań należy wykrywanie wszelkich błędów, które wystąpiły w warstwie fizycznej, oraz ich usuwanie;

7 FIZYCZNA - odpowiedzialna za przesyłanie sygnałów w elektryczny, elektromagnetyczny, mechaniczny, optyczny czy też inny sposób, wykorzystując do tego fizyczne medium komunikacyjne (przewody miedziane, światłowody)

Model TCP/IP - podział całego zagadnienia komunikacji sieciowej na szereg współpracujących ze sobą warstw (ang. *layers*). Każda z nich może być tworzona przez programistów niezależnie, jeżeli narzucimy pewne protokoły według których wymieniają się one informacjami. Założenia modelu TCP/IP są pod względem organizacji warstw zbliżone do modelu OSI. Jednak ilość warstw jest mniejsza i bardziej odzwierciedla prawdziwą strukturę Internetu.

Model TCP/IP składa się z czterech warstw:

1 APLIKACJI - najwyższy poziom, w którym pracują użyteczne dla człowieka aplikacje takie jak, np. serwer WWW czy przeglądarka internetowa. Obejmuje ona zestaw gotowych protokołów, które aplikacje wykorzystują do przesyłania różnego typu informacji w sieci.

2 TRANSPORTOWA - zapewnia pewność przesyłania danych oraz kieruje właściwe informacje do odpowiednich aplikacji.

3 INTERNETU - przetwarzane są datagramy posiadające adresy IP. Ustalana jest odpowiednia droga do docelowego komputera w sieci.

4 DOSTĘPU DO SIECI - zajmuje się przekazywaniem danych przez fizyczne połączenia między urządzeniami sieciowymi. Najczęściej są to karty sieciowe lub modemy. Dodatkowo warstwa ta jest czasami wyposażona w protokoły do dynamicznego określania adresów IP.

Model OSI	**Model TCP/IP**
został wymyślony przed wynalezieniem odpowiadających mu protokołów. Taka kolejność oznacza, że model nie był ukierunkowany na konkretny zbiór protokołów i stał się przez to bardziej ogólny.	Model jest opisem istniejących już protokołów. Jedyny problem jest taki że nie pasuje do żadnego innego stosu protokołów.
7 warstw	4 warstwy
Funkcjonalność warstw jest zbliżona. Np. w obu modelach warstwy od dołu aż do transportowej włącznie mają zapewnić dwupunktową, niezależną od sieci usługę transportową procesów, które chcą się komunikować.	
Warstwy powyżej transportowej w obu modelach są zorientowanymi na aplikację użytkownikami usługi transportowej.	

17. Funkcje warstwy łącza danych modelu OSI.

Nagłówek	Pole treści	Stopka

Druga warstwa modelu OSI. Jest ona odpowiedzialna za końcową zgodność przesyłanych danych.

Jest odpowiedzialna za upakowywanie instrukcji, danych itp. w tzw. ramki. Ramka jest strukturą rodzimą, czyli właściwą dla warstwy łącza danych, która zawiera ilość informacji wystarczającą do pomyślnego przesłania danych przez sieć lokalną do ich miejsca docelowego.

Pomyślna transmisja danych zachodzi wtedy, gdy dane osiągają miejsce docelowe w postaci niezmienionej w stosunku do postaci, w której zostały wysłane. Ramka musi więc również zawierać mechanizm umożliwiający weryfikowanie integralności jej zawartości podczas transmisji.

Wysoka jakość transmisji wymaga spełnienia następujących dwóch warunków:

- Węzeł początkowy musi odebrać od węzła końcowego potwierdzenie otrzymania każdej ramki w postaci niezmienionej.
- Węzeł docelowy przed wysłaniem potwierdzenia otrzymania ramki musi zweryfikować integralność jej zawartości.

Warstwa łącza danych jest również odpowiedzialna za ponowne składanie otrzymanych z warstwy fizycznej strumieni binarnych i umieszczanie ich w ramkach.

18. Funkcje warstwy sieci modelu OSI.

Warstwa sieci modelu OSI odpowiada za określenie trasy pomiędzy nadawcą a odbiorcą.
Kierowanie przepływem pakietów w sieci. Zapewnia ona, że pakiety przesyłane między komputerami nie łączącymi się bezpośrednio, będą przekazywane z komputera na komputer, aż osiągną adresata. Proces znajdowania drogi w sieci nazywa się trasowaniem lub rutowaniem. Nie wymaga się aby pakiety pomiędzy ustalonymi punktami poruszały się za każdym razem po tej samej drodze. Warstwa ta nie posiada żadnych mechanizmów wykrywania oraz korygowania błędów. Dopuszcza się gubienie pakietów przez tę warstwę - jest jedynym wyjściem gdy router - węzeł dokonujący trasowania lub pewien segment sieci są przeciążone i nie mogą przyjmować nowych pakietów.

19. Dynamiczne przydzielanie adresów.

Protokół DHCP opisuje trzy techniki przydzielania adresów IP:

- **przydzielanie ręczne** oparte na tablicy adresów MAC oraz odpowiednich dla nich adresów IP. Jest ona tworzona przez administratora serwera DHCP. W takiej sytuacji prawo do pracy w sieci mają tylko komputery zarejestrowane wcześniej przez obsługę systemu.
- **przydzielanie automatyczne**, gdzie wolne adresy IP z zakresu ustalonego przez administratora są przydzielane kolejnym zgłaszającym się po nie klientom.
- **przydzielanie dynamiczne**, pozwalające na ponowne użycie adresów IP. Administrator sieci nadaje zakres adresów IP do rozdzielenia. Wszyscy klienci po starcie systemu automatycznie pobierają swoje adresy na pewien czas (lease). Protokół DHCP minimalizuje również możliwe źródła błędów.

20. Protokoły połączeniowe i bezpołączeniowe.

Protokół połączeniowy to protokół komunikacyjny, w którym przed rozpoczęciem wymiany komunikatów strony muszą nawiązać połączenie, w którego ramach może zostać wynegocjowany sam protokół. Po zakończeniu łączności połączenie musi ulec likwidacji. W wyniku budowy połączenia zostaje utworzona pojedyncza ścieżka między stacją nadawczą a stacją odbiorczą. W fazie przesyłania dane są transmitowane przez utworzoną ścieżkę w sposób sekwencyjny. Dane docierają do stacji odbiorczej w kolejności ich wysyłania przez stację nadawczą.

Cecha	UDP	TCP
połączeniowy	nie	tak
nagłówek zawiera długość segmentu	tak	nie
sumy kontrolne	opcjonalne	tak
potwierdzenie pozytywne	nie	tak
retransmisje i kontrola czasu	nie	tak
wykrywanie powtórzeń	nie	tak
numery sekwencyjne	nie	tak
kontrola przepływu	nie	tak

Protokół bezpołączeniowy - typ protokołu sieciowego, umożliwia przesyłanie bez ustanawiania połączenia z odbiorcą. Nie buduje się jedynej ścieżki między stacją nadawczą i odbiorczą, pakiety do stacji odbiorczej mogą docierać w innej kolejności niż są wysyłane przez stację nadawczą, ze względu na to, że mogą być przesyłane różnymi trasami. W usłudze bezpołączeniowej dane przepływają przez trwałe połączenia między węzłami sieci, a każdy pakiet jest obsługiwany indywidualnie i niezależnie od innych pakietów danego komunikatu. Jest to możliwe pod warunkiem, że każdy pakiet jest kompletnie zaadresowany, to znaczy, że każdy z nich ma swój adres stacji nadawczej i stacji odbiorczej.

21. Tryby FTP.

FTP działa w dwóch trybach: aktywnym i pasywnym, w zależności od tego, w jakim jest trybie, używa innych portów do komunikacji.

- **jeżeli połączenie FTP działa w trybie aktywnym** to używa portu 21 dla poleceń – zestawiane przez klienta i portu 20 do przesyłu danych – zestawiane przez serwer;
- **jeżeli połączenie FTP pracuje w trybie pasywnym** to wykorzystuje port 21 dla poleceń i port o numerze powyżej 1024 do transmisji danych – obydwa połączenia zestawiane są przez klienta.

Aktywny tryb FTP jest wygodny dla administratora serwera FTP, jednak kłopotliwy dla klienta. Serwer FTP stara się nawiązać połączenie z nieuprzywilejowanym portem klienta, co najprawdopodobniej zostanie udaremnione przez firewall po stronie klienta.
Tryb **pasywny** jest wygodny dla klienta, lecz kłopotliwy dla administratora serwera FTP. Oba połączenia nawiązuje klient, ale jedno z nich dotyczy nieuprzywilejowanego portu, i najprawdopodobniej będzie zablokowane przez firewall po stronie serwera.

22. Podstawowe pojęcia baz danych. Baza Danych. Funkcje bazy danych. Właściwości bazy danych. Modele baz danych.

Baza danych – zbiór informacji opisujący wybrany fragment rzeczywistości.

Właściwości baz danych:

- **współdzielenie danych** – wielu użytkowników tej samej bazy,
- **integracja danych** – baza nie powinna mieć powtarzających się, bądź zbędnych danych,
- **niezależność danych** – dane niezależnie od aplikacji wykorzystujących te dane.
- **integralność danych** – dokładne odzwierciedlenie rzeczywistości,
- **trwałość danych** – dane przechowywane przez pewien czas,
- **bezpieczeństwo danych** – dostęp do bazy lub jej części przez upoważnionych użytkowników,

Funkcje bazy danych:

- aktualizujące – dokonują zmian na danych,
- zapytań – wydobywają dane z bazy danych.

Model baz danych to zbiór zasad (specyfikacji), opisujących strukturę danych w bazie danych. Określane są również dozwolone operacje.

Model hierarchiczny: W modelu hierarchicznym dane są przechowywane na zasadzie rekordów *nadrzędnych-podrzędnych*, tzn. rekordy przypominają strukturę drzewa. Każdy rekord (z wyjątkiem głównego) jest związany z dokładnie jednym rekordem nadrzędnym.

Model relacyjny: Dane w takim modelu przechowywane są w tabelach, z których każda ma stałą liczbę kolumn i dowolną liczbę wierszy. Każda tabela (relacja) ma zdefiniowany klucz danych (key) - wyróżniony atrybut lub kilka takich atrybutów, którego wartość jednoznacznie identyfikuje dany wiersz.

Model obiektowy: Dane w tym modelu przechowywane są jako instancje/obiekty zdefiniowanych klas. Każda klasa może mieć zdefiniowany unikalny zbiór atrybutów, a również super klasę (przodka) po którym dziedziczy atrybuty. Obiekty jednej klasy posiadają ten sam zbiór atrybutów a różnić się mogą tylko ich wartościami.

Model sieciowy: Model danych operujący pojęciami typów rekordów i typów kolekcji (opisów związków "jeden do wielu" między dwoma typami rekordów). Związki asocjacyjne pomiędzy danymi są reprezentowane poprzez powiązania wskaźnikowe. Struktura danych tworzy więc graf, czyli sieć.

23. System zarządzania bazami danych. Funkcje systemu. Przykłady SZDB.

System zarządzania bazą danych, SZBD (ang. Database Management System, DBMS) – oprogramowanie bądź system informatyczny służący do zarządzania bazą danych.

Funkcje SZBD:

- **optymalizacja zapytań** – takie przekształcenie zapytań kierowanych do bazy, aby oczekiwanie na odpowiedź było możliwie najkrótsze
- **zapewnienie integralności danych** – uniemożliwienie przejścia bazy do stanu, który nie istnieje w modelowanej rzeczywistości
- **zarządzanie współbieżnym dostępem wielu użytkowników**, tak aby ich działania nie kolidowały ze sobą
- **odporność na awarie** – możliwość odtworzenia poprawnego stanu bazy po wystąpieniu awarii
- **ochrona danych** – uniemożliwienie dostępu do danych bez autoryzacji

Przykłady SZBD:

- MySQL: silniki MyISAM, InnoDB
- Access: silnik Microsoft Jet
- Kexi (konkurent Access): silnik SQLite

24. Model relacyjny baz danych. Relacje, klucze główne i obce, integralność referencyjna.

Model relacyjny – model organizacji danych bazujący na matematycznej teorii mnogości, w szczególności na pojęciu relacji. Reprezentacją relacji jest dwuwymiarowa tabela złożona z kolumn (atrybutów) i wierszy (krotek). W modelu relacyjnym przyjmuje się następujące założenia o tabeli:

- Liczba kolumn (atrybutów) jest z góry ustalona
- Z każdą kolumną (atrybutem) jest związana jej nazwa oraz dziedzina, określająca zbiór wartości, jakie mogą występować w danej kolumnie
- Na przecięciu wiersza i kolumny znajduje się pojedyncza (atomowa) wartość należąca do dziedziny kolumny
- Wiersz (krotka) reprezentuje jeden rekord informacji
- W modelu relacyjnym kolejność wierszy (krotek) może się zmieniać

Relacje - związki między tabelami.

- **Jeden – do – wielu** (1 - ∞) wierszowi (krotce) w tabeli A może odpowiadać wiele zgodnych wierszy (krotek) w tabeli B, ale wierszowi w tabeli B może odpowiadać tylko jeden zgodny wiersz w tabeli A
- **Wiele – do – wielu** (∞ - ∞) wierszowi (krotce) w tabeli A może odpowiadać wiele zgodnych wierszy (krotek) w tabeli B i na odwrót. Relacje taką tworzy się definiując trzecią tabelę, zwaną tabelą skrzyżowań, której klucz podstawowy zawiera zarówno klucz obcy z tabeli A, jak i z tabeli B.
- **Jeden – do – jednego** (1 - 1) wierszowi (krotce) w tabeli A może odpowiadać nie więcej niż jeden zgodny wiersz (krotka) w tabeli B i na odwrót. Relacja jeden-do-jednego jest tworzona, jeśli obie powiązane kolumny są kluczami podstawowymi lub mają ograniczenia UNIQUE.

Klucz główny - dla każdej tabeli w modelu relacyjnym musi być określony, jednoznaczny identyfikator, nazywany kluczem głównym (podstawowym). Klucz główny składa się z jednej lub ze zbioru kolumn (atrybutów), w których wartości w jednoznaczny sposób identyfikują cały wiersz (krotkę). Oznacza to, że wartości znajdujące się w kolumnie będącej kluczem głównym nie mogą się powtarzać i muszą być unikatowe.

Klucz obcy - jest to zbiór złożony z jednej kolumny lub więcej kolumn, w których wartości występują, jako wartości ustalonego klucza głównego lub jednoznacznego w tej samej lub innej tabeli i są interpretowane, jako wskaźniki do wierszy w tej drugiej tabeli.

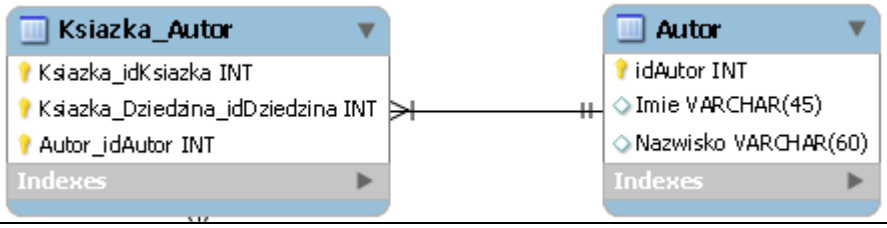
Zasada integralności referencyjnej w relacyjnej bazie danych wymaga, aby wartości klucza obcego tabeli podrzędnej były puste (null) lub odpowiadały wartościom klucza podstawowego tabeli nadrzędnej.

25. Modelowanie baz danych. Diagram związków encji.

Model bazy danych to zbiór zasad (specyfikacji), opisujących strukturę danych w bazie danych. Definiuje się strukturę danych poprzez specyfikację reprezentacji dozwolonych w modelu obiektów oraz ich związków
Hierarchiczny model danych W modelu hierarchicznym dane są przechowywane na zasadzie rekordów *nadrzędnych-podrzędnych*, tzn. rekordy przypominają strukturę drzewa. Każdy rekord (z wyjątkiem głównego) jest związany z dokładnie 1 rekordem nadrzędnym. Przykładem takiego modelu może być struktura katalogów na dysku twardym komputera.
Relacyjny model danych (RDBMS) Dane w takim modelu przechowywane są w tabelach, z których każda ma stałą liczbę kolumn i dowolną liczbę wierszy. Każda tabela (relacja) ma zdefiniowany klucz danych (key) - wyróżniony atrybut lub kilka takich atrybutów, którego wartość jednoznacznie identyfikuje dany wiersz.

Specyfikując schemat klasy obiektów w języku ODL, opisujemy trzy rodzaje właściwości obiektów:
Atrybuty - przyjmują wartości typów pierwotnych takich jak całkowity lub tekstowy, albo typów złożonych powstających z pierwotnych;
Związki - referencje do obiektów pewnej klasy, albo kolekcje (zbiory) takich referencji;
Metody - funkcje operujące na obiektach danej klasy.

Diagram Związków Encji
Modele danych można opracowywać na różnych poziomach szczegółowości wykorzystując technikę diagram związków encji.

Komponent	Opis
Encja	Rzecz mająca znaczenie, rzeczywista lub wymyślona, o której informacje należy znać lub przechowywać.
Atrybut	Element informacji służący do klasyfikowania, identyfikowania, kwalifikowania, określania ilości lub wyrażania stanu encji.
Związek	Znaczący sposób, w jaki mogą być ze sobą powiązane dwie rzeczy tego samego typu lub różnych typów.
Przykład	

Uwaga: encje opisuje się za pomocą rzeczowników lub wyrażeń rzeczownikowych w liczbie pojedynczej.

26. Język baz danych SQL. Podjęzyki DDL, DML, DCL.

1. SQL interakcyjny wykorzystywany jest przez użytkowników w celu bezpośredniego pobierania lub wprowadzania informacji do bazy. Przykładem może być zapytanie prowadzące do uzyskania zestawienia aktywności kont w miesiącu. Wynik jest wówczas przekazywany na ekran, z ewentualną opcją przekierowania go do pliku lub drukarki.

2. Statyczny kod SQL (Static SQL) nie ulega zmianom i pisany jest wraz z całą aplikacją, podczas której pracy jest wykorzystywany. Nie ulega zmianom w sensie zachowania niezmiennej treści instrukcji, które jednak zawierać mogą odwołania do zmiennych lub parametrów przekazujących wartości z lub do aplikacji. Statyczny SQL występuje w dwóch odmianach.

> **a. Embedded SQL (Osadzony SQL)** oznacza włączenie kodu SQL do kodu źródłowego innego języka. Większość aplikacji pisana jest w takich językach jak C++ czy Java, jedynie odwołania do bazy danych realizowane są w SQL. W tej odmianie statycznego SQL-a do przenoszenia wartości wykorzystywane są zmienne.
>
> **b. Język modułów**. W tym podejściu moduły SQL łączone są z modułami kodu w innym języku. Moduły kodu SQL przenoszą wartości do i z parametrów, podobnie jak to się dzieje przy wywoływaniu podprogramów w większości języków proceduralnych. Jest to pierwotne podejście, zaproponowane w standardzie SQL. Embedded SQL został do oficjalnej specyfikacji włączony nieco później.

3. Dynamiczny kod SQL (Dynamic SQL) generowany jest w trakcie pracy aplikacji. Wykorzystuje się go w miejsce podejścia statycznego, jeżeli w chwili pisania aplikacji nie jest możliwe określenie treści potrzebnych zapytań – powstaje ona w oparciu o decyzje użytkownika. Tę formę SQL generują przede wszystkim takie narzędzia jak graficzne języki zapytań. Utworzenie odpowiedniego zapytania jest tu odpowiedzią na działania użytkownika.

DML (Data Manipulation Language) służy do wykonywania operacji na danych – do ich umieszczania w bazie, kasowania, przeglądania, zmiany. Najważniejsze polecenia z tego zbioru to:

* **SELECT** * **INSERT** * **UPDATE** * **DELETE**

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** * **DROP** * **ALTER**

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

* **GRANT** * **REVOKE**

27. Instrukcja SELECT.

Polecenie SELECT nazywa się także „zapytaniem". Polecenie to pobiera krotki z relacyjnej bazy danych i zwraca w postaci zbioru odczytanych krotek (zbiór taki może być też pusty), opcjonalnie może je przetworzyć przed zwróceniem. Mówiąc prościej polecenie SELECT służy do odczytywania danych z bazy danych i dokonywaniu na nich prostych obliczeń i przekształceń.
SELECT [DISTINCT] {atrybut1, atrybut2, ...}
FROM {nazwa relacji}
[WHERE {waruek}] [ORDER BY {{wyrażenie5 [ASC|DESC], alias1 [ASC|DESC]}] [LIMIT {limit}];

28. Instrukcje DDL i DCL.

DDL (Data Definition Language - Język definiowania danych) definiuje struktury danych, na których operują instrukcje DML.
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 - Język kontrolowania danych) ma zastosowanie do nadawania uprawnień do obiektów bazodanowych.
GRANT– (GRANT {ALL | lista_uprawnień} TO użytkownik})-przyznaje użytkownikowi uprawnienia
REVOKE – (REVOKE {ALL | lista_uprawnień} TO użytkownik})-zabiera uprawnienia, które zostały wcześniej przyznane
DENY- (DENY {ALL | lista_uprawnień} TO użytkownik})-Bezpośrednio zabiera uprawnienia

29. Indeksy w bazach danych, podział indeksów , B^+-drzewo.

Indeks jest mechanizmem poprawienia szybkości dostępu do danych bez zmiany struktury ich przechowywania. Indeks zazwyczaj jest założony na jednym polu rekordu danych.
Rekord indeksu ma strukturę: <wartość pola, adres w pliku danych> i jest zapisywany do pliku indeksu w kolejności zgodnej z wartością pola indeksowanego.
Indeksy dzielimy na:

- **indeks główny** – założony na atrybucie porządkującym unikalnym,
- **grupujący** – założony na atrybucie porządkującym nieunikalnym,
- **pomocniczy** – założony na atrybucie nieporządkującym.
- **Indeks gęsty** - posiada rekord indeksu dla każdego rekordu indeksowanego pliku danych
- **Indeks rzadki** - posiada rekordy tylko dla wybranych rekordów indeksowanego pliku danych (wskazuje na blok rekordów, a nie poszczególne rekordy)

W miarę wzrostu rozmiarów indeksu wydłuża się czas wyszukiwania po tym indeksie, dlatego wiele systemów zarządzania bazą danych stosuje pewną formę indeksu wielopoziomowego:

- **statyczne wielopoziomowe indeksy** – modyfikowanie zawartości pliku z danymi powoduje, że struktura indeksu staje się nieefektywna
- **dynamiczne wielopoziomowe indeksy** – B^+-Drzewo

B+ drzewo jest zrównoważoną strukturą drzewiastą, w której wierzchołki wewnętrzne służą do wspomagania wyszukiwania, natomiast wierzchołki liści zawierają rekordy indeksu ze wskaźnikami do rekordów w plikach danych. Zrównoważenie struktury oznacza, że odległość (liczba poziomów) od korzenia do dowolnego liścia jest zawsze taka sama. W celu zapewnienia odpowiedniej efektywności realizacji zapytań przedziałowych wierzchołki liści stanowią listę dwukierunkową.
CREATE INDEX <nazwa indeksu> ON <nazwa tabeli> (<nazwa kolumny>)

30. Normalizacja., cel normalizacji, postać normalna Boyce'a-Codda.

Normalizacja bazy danych jest to proces mający na celu eliminację powtarzających się danych w relacyjnej bazie danych. Główna idea polega na trzymaniu danych w jednym miejscu, a w razie potrzeby linkowania do danych. Taki sposób tworzenia bazy danych zwiększa bezpieczeństwo danych i zmniejsza ryzyko powstania niespójności (w szczególności problemów anomalii).
Relacja R jest w PNBC wtedy, i tylko wtedy gdy dla każdej nietrywialnej zależności zbioru A1...An -> B zbiór ten jest nadkluczem R.

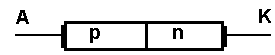
31. Transakcje, własności transakcji.

Transakcja jest sekwencją logicznie powiązanych operacji na bazie danych, która przeprowadza bazę danych z jednego stanu spójnego w inny stan spójny. Typy operacji na bazie danych obejmują: odczyt i zapis danych oraz zakończenie połączone z akceptacja (zatwierdzeniem) lub wycofaniem transakcji.

Atomicity - atomowość - zbiór operacji wchodzących w skład transakcji jest niepodzielny, to znaczy albo zostaną wykonane wszystkie operacje transakcji albo żadna.
Consistency - spójność - transakcja przeprowadza bazę danych z jednego stanu spójnego do innego stanu spójnego. W trakcie wykonywania transakcji baza danych może być przejściowo niespójna. Transakcja nie może naruszać ograniczeń integralnosciowych.
Isolation - izolacja - transakcje są od siebie logicznie odseparowane. Transakcje oddziałują na siebie poprzez dane. Mimo współbieżnego wykonywania, transakcje widza stan bazy danych tak, jak gdyby były wykonywane w sposób sekwencyjny.
Durability - trwałość - wyniki zatwierdzonych transakcji nie mogą zostać utracone w wyniku wystąpienia awarii systemu. Zatwierdzone dane w bazie danych, w przypadku awarii, musza być odtwarzalne.

32. Diody półprzewodnikowe. Tranzystory.

Diodą półprzewodnikową nazywamy element wykonany z półprzewodnika (np. krzem, german), zawierającego jedno złącze – najczęściej p-n z dwiema końcówkami wyprowadzeń.
Oznaczenia: A – anoda K – katoda n – warstwa ujemna(elektrony) p – warstwa dodatnia



rys. 1. schemat blokowy diody

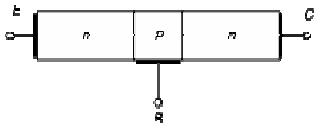
rys. 2. symbol didy p-n

Dioda **przepuszcza prąd w jednym kierunku** (od anody do katody), natomiast w kierunku przeciwnym - w minimalnym stopniu
Diody stosowane są w układach analogowych i cyfrowych.
Diody półprzewodnikowe stosuje się w układach prostowania prądu zmiennego, w układach modulacji i detekcji, przełączania, generacji i wzmacniania sygnałów elektrycznych.
Każda dioda ma pewną **częstotliwość graniczną**, po przekroczeniu której nie zachowuje się jak dioda, lecz jak kondensator

Tranzystor - trójelektrodowy półprzewodnikowy element elektroniczny, posiadający zdolność *wzmacniania* sygnału elektrycznego

- **Tranzystory bipolarne**, w których prąd wyjściowy jest funkcją prądu wejściowego (sterowanie prądowe).
- **Tranzystory unipolarne** (tranzystory polowe), w których prąd wyjściowy jest funkcją napięcia (sterowanie napięciowe)

Tranzystor posiada trzy końcówki przyłączone do warstw półprzewodnika, nazywane:
emiter (ozn. E), **baza** (ozn. B), **kolektor** (ozn. C)



Są wykorzystywane do budowy wzmacniaczy różnego rodzaju. Z tranzystorów buduje się także bramki logiczne realizujące podstawowe funkcje boolowskie, Tranzystory są także podstawowym budulcem wielu rodzajów pamięci półprzewodnikowych (RAM, ROM itp.

33. Układy scalone.

Układ scalony jest zminiaturyzowanym układem elektronicznym, który może zawierać w sobie miliony elementów elektronicznych. Płytki krzemowe bo na nich najczęściej budowane są układy scalone stanowią podłoże półprzewodnikowe dla elementów elektronicznych jak diody, kondensatory, tranzystory lub rezystory.

Ze względu na sposób wykonania układy scalone dzieli się na główne grupy:

- **monolityczne**, w których wszystkie elementy, zarówno elementy czynne jak i bierne, wykonane są w monokrystalicznej strukturze półprzewodnika. Większość stosowanych obecnie układów scalonych jest wykonana właśnie w tej technologii.
- **hybrydowe** – na płytki wykonane z izolatora nanoszone są warstwy przewodnika oraz materiału rezystywnego, które następnie są wytrawiane, tworząc układ połączeń elektrycznych oraz rezystory. Do tak utworzonych połączeń dołącza się indywidualne, miniaturowe elementy elektroniczne (w tym układy monolityczne).

W najnowszych technologiach, w których między innymi produkowane są procesory Intel i AMD, minimalna długość bramki wynosi 22nm.

34. Układy impulsowe.

Układ sterowania lub regulacji, w którym sygnały wejściowe (sterujące), a często również sygnały wyjściowe (wielkości sterowane) mogą zmieniać swoje wartości tylko w pewnych chwilach czasu — uzależnionych od różnych czynników (np. gdy sygnał wejściowy przekracza pewną wartość).
Układy dyskretne opisywane są za pomocą równań różnicowych. Transmitancja operatorowa układów dyskretnych opiera się o przekształcenie Z. Transmitancją impulsową układu dyskretnego nazywa się stosunek transformaty Z odpowiedzi układu do transformaty Z sygnału wejściowego.

35. Układy cyfrowe.

Układy cyfrowe to rodzaj układów elektronicznych, w których sygnały napięciowe przyjmują tylko określoną liczbę poziomów, którym przypisywane są wartości liczbowe. Najczęściej (choć nie zawsze) liczba poziomów napięć jest równa dwa, a poziomom przypisywane są cyfry 0 i 1, wówczas układy cyfrowe realizują operacje zgodnie z algebrą Boole'a i z tego powodu nazywane są też **układami logicznymi**. Obecnie układy cyfrowe budowane są w oparciu o bramki logiczne realizujące elementarne operacje znane z algebry Boola: iloczyn logiczny (AND, NAND), sumę logiczną (OR, NOR), negację NOT, różnicę symetryczną (XOR) itp.
Zalety układów cyfrowych:

- Możliwość **bezstratnego** kodowania i przesyłania informacji
- Zapis i przechowywanie informacji cyfrowej jest prostsze.
- Mniejsza wrażliwość na zakłócenia elektryczne.
- Możliwość tworzenia układów programowalnych, których działanie określa program komputerowy

Wady układów cyfrowych:

- Są skomplikowane zarówno na poziomie elektrycznym, jak i logicznym i obecnie ich projektowanie wspomagają komputery
- Chociaż są bardziej odporne na zakłócenia, to **wykrywanie przekłamań** stanów logicznych wymaga dodatkowych zabezpieczeń (patrz: kod korekcyjny) i też nie zawsze jest możliwe wykrycie błędu.

Ze względu na sposób przetwarzania informacji rozróżnia się:

- **układy kombinacyjne** – układy „bez pamięci", w których sygnały wyjściowe są zawsze takie same dla określonych sygnałów wejściowych;
- **układy sekwencyjne** – układy „z pamięcią", w których stan wyjść zależy nie tylko od aktualnego stanu wejść, ale również od stanów poprzednich.

36. Pamięci półprzewodnikowe, magnetyczne, optyczne.

Rodzaj pamięci	Cechy

Pamięci półprzewodnikowe RAM, EEPROM	• Bardzo szybkie – czas dostępu rzędu ns (nanosekund)
	• Bardzo duży transfer danych – rzędu GB/s (Gigabajtów na sekundę)
	• Duża cena za jednostkę pojemności
	• Możliwość bezpośredniego sprzężenia z mikroprocesorami
	• Zapis, odczyt, zachowywanie danych:
	  ○ Utrata informacji po odłączeniu zasilania (RAM)
	  ○ Tylko do odczytu (ROM)
	  ○ Trwały odczyt i zapis (EEPROM, Flash)

Pamięci dyskowe (magnetyczne - dysk twardy, taśma magnetyczna, optyczne - dysk CDROM, taśma filmowa)	• Wolne – duży czas dostępu rzędu ms (milisekund) • Średni transfer danych – rzędu dziesiątek MB/s (megabajtów na sekundę) • Niewielka cena za jednostkę pojemności • Wymagają specjalnych układów (interfejsowych – pośredniczących) • Podtrzymują dane bez zasilania • Możliwość realizacji jako pamięci wymiennych – nośnik jest oddzielony od czytnika

37. Przetworniki analogowo-cyfrowe i cyfrowo-analogowe.

Przetwornik C/A przetwarzający sygnał cyfrowy na sygnał analogowy w postaci prądu elektrycznego lub napięcia o wartości proporcjonalnej do tej liczby (równoważny sygnał analogowy).
Taki przetwornik ma n wejść i jedno wyjście.
Przetwornik A/C to układ służący do zamiany sygnału analogowego (ciągłego) na reprezentację cyfrową (sygnał cyfrowy). Dzięki temu możliwe jest przetwarzanie ich w urządzeniach elektronicznych opartych o architekturę zero-jedynkową oraz gromadzenie na dostosowanych do tej architektury nośnikach danych. Proces ten polega na uproszczeniu sygnału analogowego do postaci skwantowanej (dyskretnej), czyli zastąpieniu wartości zmieniających się płynnie do wartości zmieniających się skokowo w odpowiedniej skali (dokładności) odwzorowania.

38. Metody pomiarów wielkości elektrycznych.

Do wykonania pomiarów elektrycznych niezbędne są odpowiednie przyrządy pomiarowe:

1. **woltomierz** miernik napięcia podłączany równolegle,
2. **amperomierz** miernik natężenia podłączamy szeregowo obwód zamknięty
3. **omomierz** miernik oporu podłączamy równolegle obwód otwarty
4. **watomierz** miernik mocy podłączamy szeregowo gdy prądowy obwód
   podłączamy równolegle gdy napięciowy obwód

39. Metody pomiarów wielkości nieelektrycznych.

Wielkości nieelektryczne mierzy się dziś metodami elektrycznymi. Obecnie tylko te metody umożliwiają wykonywanie pomiarów wielkości szybko zmieniających się w czasie, prowadzenie tych pomiarów zdalnie i automatycznie. Obecnie metodami elektrycznymi mierzy się następujące wielkości: geometryczne (długość, kąt), kinetyczne (prędkość, przyspieszenie), dynamiczne (siłę, ciężar), akustyczne (natężenie dźwięku), optyczne (natężenie oświetlenia), cieplne (temperaturę), fizykochemiczne materii (lepkość, wilgotność)
W pomiarach tych wielkości nieelektryczne muszą być zamienione na wielkości elektryczne. Do tego służą **przetworniki pomiarowe**. Wielkości elektryczne pojawiające się na wyjściu przetwornika są następnie mierzone metodami elektrycznymi, przy czym przyrządy pomiarowe mogą być wyskalowane w jednostkach mierzonych wielkości nieelektrycznych.

40. Generatory kwarcowe – czas i częstotliwość w komputerze

Generatory kwarcowe - generatory drgań elektrycznych małej mocy, ale wielkiej częstości, charakteryzujące się niezwykle dużą stabilnością drgań. Wykorzystano w nich własności piezoelektryczne kryształu kwarcu. Umożliwiają wytwarzanie sygnału o częstotliwościach od 10 kHz do 200 MHz. Taki typ generatora jest stosowany w układach taktujących, układach impulsowych, we wzorcach częstotliwości i czasu, a także w zegarach kwarcowych.

Czas i częstotliwość w komputerze: Częstotliwość przerwań zegarowych zależna jest od sposobu zaprogramowania licznika. Tak, więc im wyższa częstotliwość przerwań zegarowych tym większa dokładność pomiaru czasu w systemie. Jednak częstotliwość taka, nie może być też zbyt wysoka, gdyż obsługa przerwania zegarowego absorbuje określony ułamek mocy procesora. Częstotliwość przerwań zegarowych jest kompromisem pomiędzy dokładnością pomiaru czasu, a narzutem na obsługę przerwań zegarowych. W większości stosowanych obecnie systemów częstotliwość przerwań leży pomiędzy 10Hz, a 1000Hz.

41. Grafy. Grafy eulerowskie i hamiltonowskie.

Graf to zbiór *wierzchołków*, które mogą być połączone *krawędziami*, w taki sposób, że każda krawędź kończy się i zaczyna w którymś z wierzchołków.

Graf hamiltonowski to graf zawierający ścieżkę (drogę) przechodzącą przez **każdy wierzchołek dokładnie jeden raz** zwaną ścieżką Hamiltona. W szczególności grafem hamiltonowskim jest graf zawierający cykl Hamiltona, tj. zamkniętą ścieżkę Hamiltona. W niektórych źródłach graf zawierający tylko ścieżkę Hamiltona nazywany jest grafem *półhamiltonowskim*.

Graf eulerowski, **graf Eulera** – graf eulerowski odznacza się tym, że da się w nim skonstruować cykl Eulera, czyli drogę, która przechodzi przez każdą jego **krawędź dokładnie raz** i wraca do punktu wyjściowego. Liczba krawędzi stykających się z danym wierzchołkiem nazywana jest jego stopniem. Jeżeli wszystkie wierzchołki grafu nieskierowanego mają stopień parzysty, to znaczy, że da się skonstruować zamkniętą ścieżkę Eulera nazywaną cyklem Eulera. Jeżeli najwyżej dwa wierzchołki mają nieparzysty stopień, to możliwe jest zbudowanie tylko takiej ścieżki Eulera, która nie jest zamknięta. Graf zawierający cykl Eulera jest nazywany grafem eulerowskim, a graf posiadający jedynie ścieżkę Eulera nazywany jest półeulerowskim.

Graf skierowany - ruch może odbywać się tylko w kierunkach wyznaczonych przez krawędzie. Poruszanie się "pod prąd" jest zabronione. Każdy wierzchołek posiada pewną liczbę krawędzi wejściowych nazywaną stopniem wchodzącym. Analogicznie ilość krawędzi wychodzących to stopień wychodzący. Skierowany graf eulerowski definiowany jest jako graf silnie spójny, w którym dla każdego wierzchołka grafu liczba krawędzi wchodzących jest równa ilości krawędzi wychodzących.

42. Kolorowanie grafów, definicja liczby chromatycznej grafu i indeksu chromatycznego.

Kolorowanie grafu polega na przyporządkowaniu koloru każdemu węzłowi w grafie przy wykorzystaniu jak najmniejszej liczby kolorów (liczby chromatycznej). Jednakże przy założeniu, że dwa różne ale przyległe do siebie węzły nie będą miały tego samego koloru. Takie kolorowanie stosuje się często na mapach do oznaczenia terenów lub do określenia jak składować chemikalia bo jak wiadomo niektóre substancje po zetknięciu się ze sobą mogą spowodować eksplozje.

Liczba chromatyczna - jest to liczba kolorów (liczb) niezbędna do optymalnego klasycznego (wierzchołkowego) pokolorowania grafu, czyli najmniejsza możliwa liczba k taka, że możliwe jest legalne pokolorowanie wierzchołków grafu k kolorami. Oznacza się ją symbolem $\chi(G)$.

Indeks chromatyczny grafu określa minimalną liczbę kolorów wystarczającą do prawidłowego pokolorowania krawędzi grafu. Indeks chromatyczny grafu jest równy liczbie chromatycznej jego grafu krawędziowego.

43. Ciąg Fibonacciego. Definicja rekurencyjna i wzór ogólny.

Ciąg Fibonacciego – ciąg liczb naturalnych określony rekurencyjnie w sposób następujący:
Wzór ogólny:

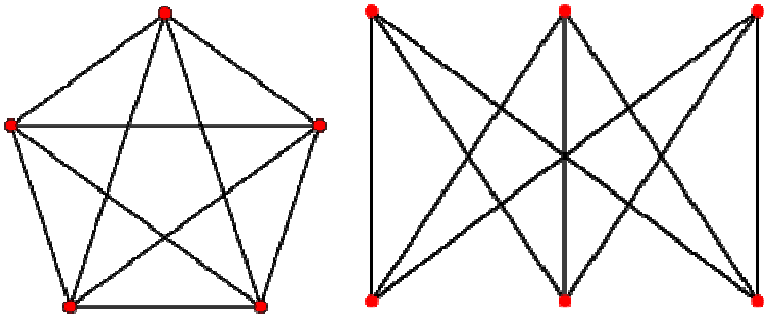
$$F_n = \frac{1}{\sqrt{5}}\left(\frac{1+\sqrt{5}}{2}\right)^n - \frac{1}{\sqrt{5}}\left(\frac{1-\sqrt{5}}{2}\right)^n \qquad F_n := \begin{cases} 0 & \text{dla } n = 0; \\ 1 & \text{dla } n = 1; \\ F_{n-1} + F_{n-2} & \text{dla } n > 1. \end{cases}$$

Kolejne liczby tego ciągu to**: 0, 1, 1, 2, 3, 5, 8, 13, 21...**

44. Grafy planarne.

Graf planarny – graf, który można narysować na płaszczyźnie tak, by krzywe obrazujące krawędzie grafu nie przecinały się ze sobą. Odwzorowanie grafu planarnego na płaszczyznę o tej własności nazywane jest jego rysunkiem płaskim. Graf planarny o zbiorze wierzchołków i krawędzi zdefiniowanym poprzez rysunek płaski nazywany jest grafem płaskim.

Kryterium Kuratowskiego
Dwa minimalne grafy, które nie są planarne, to K_5 i $K_{3,3}$. Twierdzenie Kuratowskiego mówi, że graf skończony jest planarny wtedy i tylko wtedy, gdy nie zawiera podgrafu homeomorficznego z grafem K_5 ani z grafem $K_{3,3}$.



Wzór Eulera Dowolny rysunek płaski grafu planarnego wyznacza spójne obszary płaszczyzny zwane ścianami. Dokładnie jeden z tych obszarów, zwany ścianą zewnętrzną, jest nieograniczony.
Zgodnie z wzorem Eulera, jeżeli G jest grafem spójnym i planarnym, to | V | + | S| − | E | = 2, gdzie V - zbiór wierzchołków, E - zbiór krawędzi, S - zbiór ścian dowolnego rysunku płaskiego grafu G.

Wnioski ze wzoru Eulera:
Jeżeli G jest planarny, to $|E| \leq 3 \cdot |V| - 6$.
Jeżeli G jest planarny, to wierzchołek o najmniejszym stopniu jest stopnia co najwyżej 5.

Zgodnie z twierdzeniem o czterech barwach, graf planarny daje się zawsze pokolorować przy użyciu co najwyżej czterech kolorów.

45. Drzewa spinające grafu. Minimalne drzewa spinające.

Drzewem rozpinającym grafu G nazywamy drzewo, które zawiera wszystkie wierzchołki grafu G, zaś zbiór krawędzi drzewa jest podzbiorem zbioru krawędzi grafu. Konstrukcja drzewa rozpinającego polega na usuwaniu z grafu tych krawędzi które należą do cykli.
Minimalne drzewo rozpinające jest to drzewo rozpinające danego grafu o najmniejszej z możliwych wag.
Algorytmy znajdujące dla zadanego grafu minimalne drzewo rozpinające:

- **Algorytm Prima** (nazywany też algorytmem Dijkstry-Prima)
- **Algorytm Kruskala**

46. Typy zmiennych. Podział. Przykłady w wybranym języku programowania.

Typ liczbowy Zmienne typu liczbowego dzielimy przede wszystkim na liczby **całkowite** (są to liczby bez części dziesiętnych) i **zmiennoprzecinkowe** nazywane również **rzeczywistymi** (te liczby mają już części dziesiętne). W zależności jaki typ wybierzemy mamy w każdym do wyboru kilka typów różniących się ilością zajmowanych bajtów w pamięci i przedziałem wartości. Jedynie typ decimal może występować jako typ całkowity (dokładny) lub zmiennoprzecinkowy.

int i = 10; *long l = 13456;* *byte b = 100;* *float f = 3.14;* *double d = 45.349134;*

Typ znakowy Zmienne typu znakowego służą do przechowywania pojedynczego znaku, umieszcza się go w parze pojedynczych cudzysłowów. Jeśli chcemy by zapisać więcej znaków trzeba zadeklarować tablicę typu char.

```
string B = "B                    // zmienna typu string przechowująca literę B.
char literaB = Char.Parse("B");  // zamiana typu string na char.
```

Typ łańcuchowy (string) Typ string jest typem obiektowym, reprezentującym jednowymiarowy zbiór znaków.

string s = "Imię"; *string s1 = "234";* *string s2 = "Mam na imię Adam i mam 21 lat.";*

Tablice to ciąg zmiennych jednego typu. Ciąg taki posiada jedną nazwę a do jego poszczególnych elementów odnosimy się przez numer (indeks).

```
typ nazwa_tablicy[rozmiar]; int tablica[20];
```

Typ wskaźnikowy Wskaźnik zawiera adres zmiennej. Deklaracja zmiennych typu wskaźnikowego zawiera operator * po nazwą typu, na który ma wskazywać. Kiedy chcemy uzyskać adres zmiennej, poprzedzamy ją operatorem &. Kiedy chcemy się odwołać do elementu wskazywanego przez wskaźnik, piszemy przed nazwą wskaźnika *.

47. Rodzaje pętli. Uwarunkowanie zastosowań. Problem równoważności pętli w wybranym języku programowania.

Pętla, z definicji, to element języka programowania pozwalający na wielokrotne, kontrolowane wykonywanie danego fragmentu kodu.

Zasadniczo, w języku C/C++ zdefiniowane są trzy rodzaje pętli:

- Pętla warunkowa do…while
- Pętla warunkowa while…
- Pętla krokowa for…

Zaczynając od pętli warunkowych, pętla do… i pętla while… są sobie równoważne pod względem konstrukcyjnym. Różnica tkwi w miejscu sprawdzania warunku. Odrębnym rodzajem jest pętla krokowa for…. Odrębnym, ponieważ pętla jest wykonywana zadaną ilość razy kontrolowaną przez licznik, tworzony specjalnie na potrzeby tej pętli.

Równoważność między pętlą krokową for… i pętlami warunkowymi while… i do… jest tylko częściowa. Pętlę for… zawsze można zaimplementować za pomocą pętli while… lub do…, przenosząc bezpośrednio warunek kończący lub zanegowany warunek początkowy i dodając do instrukcji wewnątrz pętli, instrukcję inkrementującą lub dekrementującą licznik. Implementacja w drugim kierunku jednak nie zawsze jest możliwa.

48. Zmienne typu adresowego (wskaźniki). Implementacja w wybranym języku programowania.

Zmienne typu adresowego, zamiast faktycznej wartości danego typu, przechowują adres fizycznej pamięci, przechowujący właściwą zmienną z wartością. Wykorzystywane są najczęściej, jeśli funkcja ma operować na głównej zmiennej, a nie standardowo na jej kopii utworzonej na potrzeby funkcji. W przypadku deklarowania wskaźnika dla tablicy, pod tą zmienną znajduje się adres pierwszej komórki tej tabeli. Aby dostać się do kolejnych komórek tabeli należy inkrementować odpowiednio zmienną wskaźnika.

Implementacja wskaźnika dla zmiennej typu int.	int* wsk;
Implementacja wskaźnika dla bloku dowolnego typu pamięci	void* wsk;
wskaźnik 3-elementowej tablicy liczb całkowitych,	int (*ptab)[3];
wskaźnik do 3-elementowej tablicy wskaźników,	int *(*ptab)[3];
Tablica 4 wskaźników liczb całkowitych	int *tab[4]; lub int (*tab[4]);
stały wskaźnik typu całkowitego	int *const pc
wskaźnik stałej typu całkowitego	int const *pxc

49. Funkcje (z wzmianką o procedurach). Przekazywanie parametrów przez wartość i referencję lub adres.

Funkcja jest to fragment kodu, który może być wywołany wielokrotnie w różnych miejscach programu. Każda funkcja powinna zwracać wynik wykonanego kodu. Procedura nie zwraca wartości.

typ nazwa_funkcji (**parametry**){
    // instrukcje
}

```
void f1(int c, int d){
    c++;
    d++;
}
```

```
void f2(int &x, int &y){
    x++;
    y++;
}
```

Przekazywanie parametrów przez wartość (funkcja f1) – pracujemy na kopiach zmiennych a i b, czyli nie zostaną one zmienione w głównym programie.

Przekazywanie parametrów przez referencję (funkcja f2) – do funkcji przekazujemy adresy zmiennych a i b, czyli pracujemy na wartościach przechowywanych w ich adresach.

50. Cechy programowania strukturalnego. Kluczowe różnice między programowaniem strukturalnym a obiektowym.

Cechy: hierarchiczna i modułowa struktura programu, możliwość analizy poprawności programu, łatwość modyfikacji, redukcja błędów i czasu pisania programu, możliwość wykorzystania wcześniej napisanych modułów do innych programów, metody: analityczna, czyli podejście "z góry na dół" (ang. top-down), syntetyczna, czyli podejście "z dołu do góry" (ang. bottom-up)

Do kluczowych różnic między programowaniem strukturalnym a obiektowym zaliczamy:

- program pisany strukturalnie – zawiera „bloki”, gdzie program pisany obiektowo zawiera obiekty
- dane i procedury w programowaniu strukturalnym nie są ze sobą bezpośrednio powiązane
- programy strukturalne nie posiadają takich cech jak: abstrakcja, hermetyzacja, polimorfizm, dziedziczenie – charakterystyczne dla programowania obiektowego
- programowanie obiektowe może być oparte na wzorach projektowych – gdzie strukturalne ma sztywną metodę pisania

51. Zalety języków programowania obiektowo orientowanych. Składniki klasy. Konstruktory i destruktory. Podać przykłady.

Zalety programowania obiektowego

a) łatwość przekładania poszczególnych wymogów na poszczególne moduły systemu;

b) możliwość wielokrotnego zastosowania poszczególnych modułów systemu;

c) każdy moduł odpowiedzialny jest za konkretne zadanie, nie występuje dublowanie funkcjonalności co pozwala szybko i skutecznie poprawiać powstałe błędy.

Klasa *jest to grupa danych i funkcji, które na tych danych operują. Tworząc klasę tworzymy nowy typ danych, którego składnikami są inne typy danych.*

Składnik klasy *to element należący do danej klasy. Składnikami klasy mogą być: pola, konstruktory, destruktory, metody, właściwości, indeksatory, delegacje, operatory jak również zagnieżdżone: klasy, struktury listy wyliczeniowe czy interfejsy.*

a) public - oznacza, że składniki deklarowane po tej etykiecie są dostępne z każdego miejsca programu,

b) private - oznacza, że składniki deklarowane po tej etykiecie dostępne są tylko dla funkcji składowych tej klasy; funkcje globalne (zwykłe) nie mają dostępu do tych składników, a więc z funkcji main również nie ma dostępu do tych danych,

c) protected - tak jak w przypadku private z tą tylko różnicą, że dostęp do takich składników mają jeszcze klasy wywodzące się z tej klasy

```
class osoba{
        public:
        char imie[20];
        char nazwisko[30];
        int wiek;
        osoba(char imie, char nazwisko, int wiek)
        {
                this->imie=imie;
                this->nazwisko=nazwisko;
                this->wiek=wiek;
        };          // konstruktor z parametrami
        virtual ~osoba();       // destruktor wirtualny, bezparametrowy
    };
osoba osoba1(Jan, Kowalski,30); // wywołanie konstruktora
osoba1.osoba::~osoba(); // wywołanie destruktora
```

Konstruktor jest to funkcja w klasie, wywoływana w trakcie tworzenia każdego obiektu danej klasy. Należy dodać że każda klasa ma swój konstruktor. Nawet jeżeli nie zadeklarujemy go jawnie zrobi to za nas kompilator (stworzy wtedy konstruktor bezparametrowy).

Destruktor jest natomiast funkcją, którą wykonuje się w celu zwolnienia pamięci; następuje niszczenie obiektu danej klasy.

52. Zastosowanie składników statycznych w klasie. Statyczne funkcje składowe. Podać przykłady.

Czasami zachodzi potrzeba dodania elementu, który jest związany z *klasą*, ale nie z konkretną *instancją* tej klasy. Możemy wtedy stworzyć element **statyczny**. Element **statyczny** jest właśnie elementem, który jest powiązany z *klasą*, a nie z *obiektem* tej klasy, czyli np. statyczna metoda nie może się odwołać do niestatycznej zmiennej lub funkcji.

Elementy **statyczne** poprzedza się podczas definicji słówkiem *static*. **Statyczne** mogą być zarówno *funkcje*, jak i *pola* należące do klasy.

```
class Klasa {
    protected:
        static int iloscInstancji; // pole statyczne
    public:
        Klasa() {
            iloscInstancji++;
        }
        virtual ~Klasa() {
            iloscInstancji--;
        }
        static int IloscInstancji() {
            return iloscInstancji;
        }
};
```

W powyższym przykładzie ponadto istnieje **metoda statyczna**. Z takiej metody nie można się odwołać do niestatycznych elementów klasy. Zarówno do **klasy statycznej** jak do **statycznego pola** możemy się odwołać nawet jeżeli nie został stworzony żaden obiekt klasy *Klasa*.

Odwołanie się do **metody statycznej** *IloscInstancji* z programu wymaga następująco:

```
int i=Klasa::IloscInstancji();
```

Gdyby zaś pole *iloscInstancji* było publiczne, a nie chronione, to moglibyśmy się do niego odwołać poprzez:

```
int i=Klasa::iloscInstancji;
```

Statyczne funkcje składowe

Takie funkcje można wywołać nawet wtedy, gdy nie istnieje jeszcze żaden obiekt klasy. Do jej nazwy z zewnątrz klasy odwołujemy się poprzez nazwę klasy za pomocą operatora zasięgu, czyli „czterokropka" (' ::'), albo za pomocą operatora wyboru składowej (kropka). Ponieważ funkcja statyczna *nie* jest wywoływana na rzecz obiektu, ale jak funkcja globalna, nie można w niej odwoływać się do **this** ani do żadnych składowych niestatycznych - te bowiem istnieją tylko wewnątrz konkretnych obiektów i w każdym z nich mogą być różne. Można natomiast w funkcjach statycznych klasy odwoływać się do składowych statycznych tej klasy: innych funkcji statycznych i zmiennych klasowych (określanych przez statyczne pola klasy).

53. Dziedziczenie. Dostęp do składników klasy. Kolejność wywołania konstruktorów. Przypisanie i inicjalizacja obiektów w warunkach dziedziczenia.

Dziedziczenie to w programowaniu obiektowym operacja polegająca na stworzeniu nowej klasy na bazie klasy już istniejącej. Stosowana najczęściej przy tworzeniu potomnych klas specjalizowanych, lub przy tworzeniu klas użytkowych z klasy abstrakcyjnej.

Dostęp do składników klasy głównej jest zależny od tego w jakich sekcjach są one zdefiniowane. Do składników z sekcji private danej klasy mają dostęp jedynie metody i przyjaciele tej klasy. Aby więc klasa dziedzicząca mogła naprawdę dziedziczyć z klasy głównej, wszystkie typy danych w klasie głównej muszy być typu public lub protected.

```
class Ryba : public Zwierze {
      public:
          Ryba();
          void plyn();};
```

Pierwszy wiersz jest właściwym dziedziczeniem, zapisem jak nazywa się klasa potomna i z jakiej klasy dziedziczy. Najpierw wykonuje się konstruktor klasy z której się dziedziczy, a następnie instrukcje konstruktora klasy dziedziczącej. Przy wywoływaniu funkcji z klasy potomnej lub głównej, gdy funkcje publiczne w obu przypadkach mają taką samą nazwę, należy poprzedzać je zapisem przynależności do przestrzeni odpowiedniej klasy: KlasaPotomna.funkcja1(); KlasaGlowna.funkcja1();

54. Polimorfizm. Deklarowanie funkcji wirtualnych. Mechanizm wywołania. Podać przykłady.

Polimorfizm - mechanizmy pozwalające programiście używać wartości, zmiennych i podprogramów na kilka różnych sposobów. Inaczej mówiąc jest to możliwość wyodrębnienia wyrażeń od konkretnych typów. Podczas pisania programu wygodnie jest traktować nawet różne dane w jednolity sposób. Niezależnie czy należy wydrukować liczbę czy napis, czytelniej (zazwyczaj) jest gdy operacja taka nazywa się po prostu *drukuj*, a nie *drukuj_liczbę* i *drukuj_napis*. Jednak napis musi być drukowany inaczej niż liczba, dlatego będą istniały dwie implementacje polecenia *drukuj* ale nazwanie ich wspólną nazwą tworzy wygodny abstrakcyjny interfejs niezależny od typu drukowanej wartości.
Funkcje wirtualne to specjalne funkcje składowe, które przydają się szczególnie, gdy używamy obiektów posługując się wskaźnikami lub referencjami do nich. Dla zwykłych funkcji z identycznymi nazwami to, czy zostanie wywołana funkcja z klasy podstawowej, czy pochodnej, zależy od typu wskaźnika, a nie tego, na co faktycznie on wskazuje. Dysponując funkcjami wirtualnymi będziemy mogli użyć prawdziwego **polimorfizmu** - używać metod klasy pochodnej wszędzie tam, gdzie spodziewana jest klasa podstawowa. W ten sposób będziemy mogli korzystać z metod klasy pochodnej korzystając ze wskaźnika, którego typ odnosi się do klasy podstawowej.

```
class Baza
{
public:
  void pisz()
  {
    std::cout << "Tu funkcja pisz z klasy Baza" << std::endl;
  }
};

class Baza2 : public Baza
{
public:
  void pisz()
  {
    std::cout << "Tu funkcja pisz z klasy Baza2" << std::endl;
  }
};
```

Jeżeli teraz w funkcji main stworzymy wskaźnik do obiektu typu Baza, to możemy ten wskaźnik ustawiać na dowolne obiekty tego typu. Można też ustawić go na obiekt typu pochodnego, czyli Baza2:

```
int main()
{
  Baza*  wsk;
  Baza   objB;
  Baza2  objB2;

  wsk = &objB;
  wsk -> pisz();

  wsk = &objB2; // Teraz ustawiamy wskaźnik wsk na obiekt typu pochodnego
  wsk -> pisz();
  return 0;
}
```

Po skompilowaniu na ekranie zobaczymy dwa wypisy: "Tu funkcja pisz z klasy Baza". Stało się tak dlatego, że wskaźnik jest do typu Baza.

Można jednak określić żeby kompilator nie sięgał po funkcję z klasy bazowej, ale sam się zorientował na co wskaźnik pokazuje. Do tego służy przydomek *virtual*, a funkcja składowa nim oznaczona nazywa się wirtualną.

```
class Baza
{
public:
  virtual void pisz()
  {
    cout << "Tu funkcja pisz z klasy baza" << endl;
  }
};

class Baza2 : public Baza
{
public:
  void pisz()
  {
    cout << "Tu funkcja pisz z klasy Baza2" << endl;
  }
};
```

Gdy funkcja jest oznaczona jako wirtualna, kompilator nie przypisuje na stałe wywołania funkcji z tej klasy, na którą pokazuje wskaźnik, już podczas kompilacji.

55. Definiowanie szablonu klasy. Korzystanie z szablonu.

Szablonów klas używamy, jeśli planujemy używać podobnych/identycznych struktur dla różnych typów danych.

Definiowanie szablonu klasy:

```
template <class T>
class Liczba
{
   public:
     T dodaj(T zmienna, T zmienna2){
              return zmienna + zmienna2;
     }
}
```

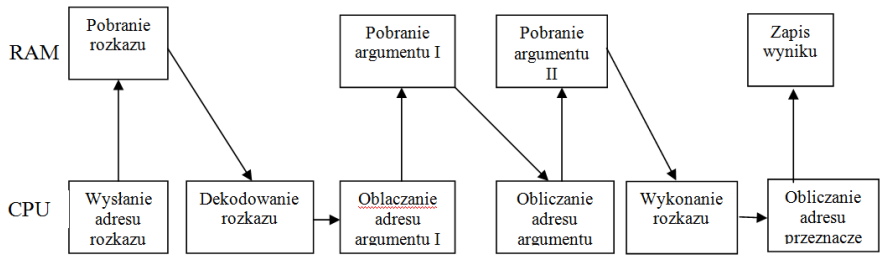
Korzystanie z szablonu:

```
   Liczba <int> liczba_int;
   Liczba <float> liczba_float;

   cout<<liczba_int.dodaj(1, 2);
   cout<<liczba_float.dodaj(1.0, 2.0);
```

56. Cykl pracy procesora przy wykonaniu programu.

Typowy cykl rozkazowy w systemie o architekturze von Neumanna zaczyna się od pobrania rozkazu z pamięci i przesłania go do rejestru rozkazów (ang. instruction register). Rozkaz jest następnie dekodowany i realizowany (może spowodować pobranie argumentów z pamięci i umieszczenie ich w innym rejestrze wewnętrznym). Po wykonaniu rozkazu na argumentach jego wynik można z powrotem przechować w pamięci.
Zauważmy, że jednostka pamięci „widzi" tylko strumień adresów pamięci. Nie jest jej znany sposób, w jaki one powstały (licznik rozkazów, indeksowanie, modyfikacje pośrednie, adresy literalne itp.) ani czemu służą.



57. Procesy, zarządzanie procesami.

Proces jest to po prostu egzemplarz wykonywanego programu. Należy odróżnić jednak proces od wątku. Każdy proces posiada własną przestrzeń adresową, natomiast wątki posiadają wspólną sekcję danych.
Każdemu procesowi przydzielone zostają zasoby, takie jak: procesor, pamięć, dostęp do urządzeń wejścia-wyjścia, pliki.
Każdy proces posiada tzw. "rodzica". W ten sposób tworzy się swego rodzaju drzewo procesów. Proces może (ale nie musi) mieć swoje procesy potomne. Za zarządzanie procesami odpowiada jądro systemu operacyjnego. Wykonanie musi przebiegać sekwencyjnie.
Może przyjmować kilka stanów: działający, czekający na udostępnienie przez system operacyjny zasobów, przeznaczony do zniszczenia, właśnie tworzony itd. -proces zombie
W skład procesu wchodzi: kod programu, licznik rozkazów, stos, dane
Zarządzanie procesami

- System operacyjny odpowiada za następujące działania dotyczące zarządzania procesami: tworzenie i usuwanie
- planowanie porządku wykonywania
- mechanizmy synchronizacji, komunikacji
- usuwanie zakleszczeń

58. Metody przydziału pamięci operacyjnej procesowi. Organizacja pamięci wirtualnej.

First-Fit - Jest to najprostsza możliwa strategia. Wybierany jest pierwszy (pod względem adresów) wolny obszar, który jest wystarczająco duży.

Best-Fit - Wybieramy najmniejszy wolny obszar, który jest wystarczająco duży.

Worst-fit - Przydzielamy pamięć zawsze z największego wolnego obszaru (oczywiście, o ile jest on wystarczająco duży).

Segmentacja Pamięć wykorzystywana przez proces, z logicznego punktu widzenia, nie stanowi jednego spójnego obszaru. Zwykle składa się z kilku segmentów. Typowe segmenty to: kod programu, zmienne globalne, stos i sterta. System operacyjny może więc przydzielać procesom pamięć nie w postaci jednego spójnego bloku, ale kilku takich bloków, segmentów. Co więcej, proces może w trakcie działania prosić o przydzielenie lub zwolnienie segmentów.

Stronicowanie Podział wirtualnej przestrzeni adresowej procesu na strony. Strona to obszar ciągłej pamięci o stałym rozmiarze (zazwyczaj 4kB). Rzeczywista pamięć operacyjna podzielona jest na ramki,

których rozmiar odpowiada wielkości stron. System operacyjny według uznania może przydzielać ramkom strony pamięci lub pozostawiać je puste.

Pamięć wirtualna jest techniką programową a także sprzętową gospodarowania pamięcią operacyjną RAM pozwalającą na przydzielanie pamięci dla wielu procesów, zwalnianie jej i powtórne przydzielanie, w ilości większej niż rzeczywista ilość pamięci fizycznej zainstalowanej w komputerze poprzez przeniesienie danych z ostatnio nie używanej pamięci do pamięci masowej (np. twardego dysku), w sytuacji gdy procesor odwołuje się do danych z pamięci przeniesionej na dysk przesuwa się te dane do pamięci w wolne miejsce, a gdy brak wolnej pamięci zwalnia się ją przez wyżej opisane przerzucenie jej na dysk.

59. Funkcje systemowe – podstawowe kategorie, przykłady.

Programy użytkowników mają dostęp do usług systemu operacyjnego poprzez tzw. funkcje systemowe. Funkcje te są widoczne w języku programowania podobnie jak funkcje z tego języka, a ich konkretny mechanizm wywoływania jest ukryty przed programistą.

Funkcje udostępniane przez system operacyjny możemy pogrupować w następujący sposób:

- **operacje na procesach** Ta grupa funkcji obejmuje m.in.: tworzenie nowych procesów, uruchamianie programów, zakończenie się procesu, przerwanie działania innego procesu, pobranie informacji o procesie, zawieszenie i wznowienie procesu, oczekiwanie na określone zdarzenie, przydzielanie i zwalnianie pamięci,
- **operacje na plikach** Ta grupa funkcji obejmuje takie operacje, jak: utworzenie pliku, usunięcie pliku, otwarcie pliku, odczyt z pliku, zapis do pliku, pobranie lub zmiana atrybutów pliku.
- **operacje na urządzeniach** Zamówienie i zwolnienie urządzenia, odczyt z i zapis do urządzenia, pobranie lub zmiana atrybutów urządzenia, (logiczne) przyłączenie lub odłączenie urządzenia.
- **informacje systemowe** Funkcje te obejmują pobieranie i/lub ustawianie najrozmaitszych informacji systemowych, w tym: czasu i daty, wersji systemu operacyjnego, atrybutów użytkowników itp.
- **komunikacja** Ta grupa funkcji obejmuje przekazywanie informacji między procesami. Spotykane są dwa schematy komunikacji: przekazywanie komunikatów i pamięć współdzielona. Przekazywanie komunikatów polega na tym, że jeden proces może wysłać (za pośrednictwem systemu operacyjnego) pakiet informacji, który może być odebrany przez drugi proces. Mechanizm pamięci współdzielonej polega na tym, że dwa procesy uzyskują dostęp do wspólnego obszaru pamięci.

60. Szeregowanie procesów. Wybrane algorytmy szeregowania.

Proces szeregujący (ang. scheduler) to część jądra zajmująca się przydzielaniem procesom procesora, zgodnie z pewną polityką, za pomocą pewnych mechanizmów.

FIFO - algorytm powszechnie stosowany, jeden z prostszych w realizacji, dający dobre efekty w systemach ogólnego przeznaczenia; zadanie wykonuje się aż nie zostanie wywłaszczone przez siebie lub inne zadanie o wyższym priorytecie

Planowanie priorytetowe - wybierany jest proces o najwyższym priorytecie. W tej metodzie występuje problem nieskończonego blokowania (procesu o niskim priorytecie przez procesy o wysokim priotytecie). Stosuje się tu postarzanie procesów, polegające na powolnym podnoszeniu priorytetu procesów zbyt długo oczekujących.

Planowanie rotacyjne ('round-robin) - Planista przydziela każdemu z procesów kwant czasu i jeżeli nie zdąży się wykonać to ląduje na końcu kolejki FCFS.

Algorytm wywłaszczający - Algorytm, który może przerwać wykonanie procesu i przenieść go z powrotem do kolejki.

Algorytm niewywłaszczający - Algorytm, w którym procesy przełączają się dobrowolnie. Proces aktywny jest przenoszony do kolejki procesów oczekujących tylko wtedy, gdy sam przerwie swe działanie; dopóki tego nie uczyni (lub nie zakończy działania), żaden inny proces nie otrzyma dostępu do procesora.

61. Synchronizacja procesów współbieżnych. Semafory.

Procesy współbieżne to takie procesy, które są wykonywane jednocześnie w systemie operacyjnym. Można je podzielić na dwie kategorie: niezależne oraz współpracujące. Procesy współpracujące mogą wpływać na inne procesy lub podlegać ich oddziaływaniom. Mogą również dzielić wspólne dane, do których trzeba zagwarantować zsynchronizowany dostęp, aby zapewnić ich spójność, tzn. nie można dopuścić do sytuacji, w której wykonywane na nich zmiany zaburzają się wzajemnie.
Jednym z narzędzi do synchronizacji jest semafor. Semafor to zmienna całkowita, która przyjmuje pewną początkową wartość (zazwyczaj 1) i na której można wykonać tylko dwie operacje: *wait* i *signal*. Są to operacje niepodzielne. Jeśli jeden proces zmienia wartość semafora, to żaden inny proces nie może jednocześnie tego robić.
Każdy proces przed wejściem do sekcji krytycznej wywołuje funkcję *wait*, która sprawdza wartość semafora. Jeżeli wartość jest większa od zera, to proces wchodzi do sekcji krytycznej, zmniejsza wartość semafora o jeden i modyfikuje dane. Proces, który zakończy wykonywanie sekcji krytycznej wywołuje funkcję *signal*, która dla klasycznego semafora zwiększa jego wartość o jeden. W przypadku rozwiązania semafora z kolejką procesów czekających, funkcja *signal* zwiększa wartość semafora o jeden i wznawia jeden z procesów czekających w kolejce.
Istnieje też druga konstrukcja semafora: semafor binarny. Różni się on od wcześniejszego, zwanego semaforem zliczającym, tym, że przyjmuje tylko dwie wartości: 0 lub 1.

62. Cykle projektowania i życia oprogramowania.

strategiczna *W tej fazie podejmowane są decyzje strategiczne odnośnie podejmowania przedsięwzięcia projektowego: zakresu, kosztów, czasu realizacji itp.*
analizy *(ang. analysis), w której budowany jest logiczny model systemu,*
dokumentacj*i, w której wytwarzana jest dokumentacja użytkownika. Opracowywanie dokumentacji przebiega równolegle z produkcją oprogramowania. Faza ta praktycznie rozpoczyna się już w trakcie określania wymagań.*
określania wymagań*, w której określane są cele oraz szczegółowe wymagania wobec systemu,*
projektowania *(ang. design), w której powstaje szczegółowy projekt systemu spełniającego ustalone wcześniej wymagania,*
implementacji/kodowania *(ang. implementation/coding) oraz testowania modułów, w której projekt zostaje zaimplementowany w konkretnym środowisku programistycznym oraz wykonywane są testy poszczególnych modułów,*
testowania, *w której następuje integracja poszczególnych modułów połączona z testowaniem poszczególnych podsystemów oraz całego SI,*
instalacji*, w której następuje przekazanie systemu użytkownikowi,*
konserwacji, *w której oprogramowanie jest wykorzystywane przez użytkownika (ów), a producent dokonuje konserwacji SI (a przede wszystkim oprogramowania) – wykonuje modyfikacje polegające na usuwaniu błędów, zmianach i rozszerzaniu funkcji systemu;*
likwidacja*, w której wykonuje się czynności związane z zakończeniem użytkowania SI*

63. Klasyfikacja narzędzi wspierających wytwarzanie oprogramowania.

1) **Zintegrowane środowisko programistyczne** jest to aplikacja lub zespół aplikacji (środowisko) służących do tworzenia, modyfikowania, testowania i konserwacji oprogramowania.

   - pakiet Microsoft Visual Studio *(popularny na systemach rodziny Windows)*
   - Eclipse i NetBeans
   - Zend Studio *(rozwiązanie dedykowane dla jezyka PHP)*

2) **CASE** (**Computer-Aided Software Engineering**,) - oprogramowanie używane do komputerowego wspomagania projektowania oprogramowania.

   - narzędzia do modelowania w języku UML i podobnych
   - narzędzia do zarządzania konfiguracją zawierające system kontroli wersji
   - narzędzia do re factoringu

3) **Systemy kontroli wersji** służą do śledzenia zmian głównie w kodzie źródłowym oraz pomocy programistom w łączeniu i modyfikacji zmian dokonanych przez wiele osób w różnych momentach.
   Git, Mercurial, SVN

4) **Kompilatory**(ang. *compiler*) - programy służący do automatycznego tłumaczenia kodu napisanego w jednym języku (języku *źródłowym*) na równoważny kod w innym języku (języku *wynikowym*)
   Microsoft Visual Studio, GNU Compiler for Java

5) **Narzędzia wspomagające kompilację**
   Ccache, Distcc, Scratchbox

6) **Narzędzia wspomagające budowę aplikacji**
   Autotools, Autoconf, Autoheader, Automake

7) **Narzędzia do analizy programów (Debuggery)**
   programy komputerowy służący do dynamicznej analizy innych programów, w celu odnalezienia i identyfikacji zawartych w nich błędów, zwanych z angielskiego bugami (robakami).
   GNU DebuggerSoftICE

64. Metody oraz strategie testowania oprogramowania.

Testowanie oprogramowania – proces związany z wytwarzaniem oprogramowania. Jest on jednym z procesów kontroli jakości oprogramowania.
Wyróżniamy testy:
1. Statyczne: proces pozwalający na wyszukanie błędów od fazy zbierania wymagań biznesowych poprzez konstruowanie kodu, aż po dostarczenie produktu, jednak bez uruchamiania aplikacji.
Testy statyczne przeprowadzane są przy użyciu:

- **Przegląd** jest procesem lub spotkaniem podczas którego produkt/system jest przedstawiany członkom zespołu projektowego, użytkownikom oraz innym zainteresowanym stronom w celu uzyskania komentarzy lub aprobaty dla danego rozwiązania.
- **Analiza statyczna** jest to analiza struktury kodu źródłowego lub kodu skompilowanego bez jego uruchomienia. Analiza statyczna może odbywać się na etapie budowania aplikacji, ponieważ jej wyniki są dostępne już podczas kompilacji kodu źródłowego.

2. Dynamiczne:

- **Testy funkcjonalne** (testy czarnej skrzynki). Tester nie ma dostępu do kodu testowanej aplikacji. Testy wykonywane są przez osobę, która nie tworzyła aplikacji w opaciu o dokumentację oraz założenia funkcjonalne.
- **Testy strukturalne**. Przykładem testów strukturalnych są testy jednostkowe (unit tests), które polegają na stworzeniu kodu sprawdzającego poprawność działania właściwego kodu aplikacji. Do zdefiniowania zasad przejścia przez ścieżki aplikacji, używane są kryteria pokrycia pętli i warunków.

65. Instalacja i konserwacja oprogramowania.

Przez **instalację** (**wdrożenie)** systemu informatycznego rozumiemy przekazanie systemu użytkownikowi, po czym następuje etap funkcjonowania SI i jego konserwacji (usuwania błędów, modyfikacji).

Proces wdrożenia zależy od tego, czy wytwarzane oprogramowanie jest seryjnym (komercyjnym) – sprzedawanym w wielu egzemplarzy, czy też jest stworzono na konkretne zamówienie (jest dedykowanym SI). W tym ostatnim wypadku na etapie wdrożenia dokonuje się:

- Szkolenie przyszłych administratorów systemu oraz jego użytkowników.
- Instalacja sprzętu i oprogramowania.
- Wprowadzanie rzeczywistych danych do baz danych.
- Usuwanie znalezionych błędów w programach i dokumentacji.
- Przekazanie systemu użytkownikowi.

Konserwacja oprogramowania polega na wykonaniu ewentualnych modyfikacji oprogramowania. Tymi modyfikacjami mogą być:

- Dostosowanie oprogramowania do zmian zachodzących w środowisku pracy klienta (zmian pewnych funkcji systemu).
- Dostosowanie oprogramowania do zmian sprzętu, urządzeń technicznych.
- Poprawianie jakości oprogramowania.
- Usuwanie odnalezionych błędów.

66. Zagadnienia etyczne i prawne związane z procesem wytwarzania i użytkowania oprogramowania.

Przykłady nieetycznych zachowań w obszarze IT obejmują:

- przejmowanie cudzych pomysłów i projektów,
- kradzież informacji, przedkładanie plagiatów,
- złośliwe usuwanie dokumentów,
- handel danymi,
- zamieszczanie informacji obraźliwych,
- celowe przeciążanie serwerów

Do często spotykanych błędów w kontraktach publicznych na oprogramowanie zalicza się:

- Deklarowane wirtualne koszty, tj. takie, które pozwolą wygrać przetarg,
- Nierealny termin dostawy,
- Nieprawidłowy harmonogram etapów,
- Świadomie niekompletne wymagania,
- Brak doświadczenia dostawcy w danej dziedzinie zastosowań,
- Długi okres wyczekiwania na poprawę zgłaszanych defektów,
- Brak alternatywnych procedur na wypadek niepowodzenia przedsięwzięcia programowego,
- Niewystarczająca troska o walidację danych.

67. Etapy cyklu życia systemu informatycznego i ich charakterystyka

Określenie założeń i celów systemu - Etap ten jest konieczny przed rozpoczęciem pracy analityków i projektantów systemu. W deklaracji założeń i celów należy przedstawić problem, który powinien zostać rozwiązany oraz obszar działania, którego on dotyczy.
Zebranie informacji i studium możliwości - Efektem tego etapu jest ocena technicznych możliwości rozwiązania problemów i realizacji założeń projektu. Mogą tutaj zostać zaprezentowane różne warianty rozwiązań w ogólnym zarysie wraz z szacunkiem kosztów, efektów i możliwości realizacji. Celem tego etapu jest dostarczenie danych niezbędnych do podjęcia decyzji o kontynuowaniu prac nad projektem.
Analiza systemu - Faza ta obejmuje stworzenie modelu logicznego przepływu danych i procesów niezbędnych do realizacji opisanych funkcji. Efektem tej fazy jest model logiczny systemu uzupełniony o specyfikację algorytmów przetwarzania, słowniki danych i modele danych.
Projektowanie ogólne – Po analizie systemu, projektanci mogą przystąpić do konstrukcji logicznej struktury systemu. Następuje wybór optymalnej platformy sprzętowej, struktury bazy danych, itp. Efektem tej fazy są różne warianty projektowanego systemu, wraz z ich oceną dotyczącą kosztu, wpływu na organizacje i prognozowanych efektów wdrożenia.
Projektowanie szczegółowe - Po akceptacji modelowego rozwiązania niezbędne jest stworzenie wymagań dotyczących koniecznych zakupów sprzętu i oprogramowania. Następujące pytania wymagają odpowiedzi: jakie programy należy napisać, jaki sprzęt należy kupić, jaka powinna być struktura bazy danych oraz systemu plików, jaki powinien być harmonogram wdrożenia
Wdrożenie - W czasie wdrożenia, zaprojektowany system jest fizycznie tworzony. Programy są pisane przez programistów i instalowane na zakupionym sprzęcie. Przeprowadzane są testy działania systemu i współpracy różnych programów. Tworzona jest struktura bazy danych, następuje przeniesienie danych historycznych ze starych systemów do nowego systemu.
Przejście na nowy system - Jest to etap, w którym przedsiębiorstwo zaczyna wykorzystywać nowy system w działalności operacyjnej.
Eksploatacja oraz ocena działania systemu W tym momencie firma posiada już uruchomiony i wykorzystywany w codziennej pracy system.

68. Modele organizacyjne wytwarzania systemów informatycznych i ich charakterystyka – zalety i wady.

Model	Fazy	Zalety	Wady
MODEL KASKADOWY- polega na wykonywaniu podstawowych czynności jako odrębnych faz projektowych, w porządku jeden po drugim. Każda czynność to kolejny schodek (kaskada)	- określenia wymagań, - analizy, - projektowania, - implementacji/kodowania - testowania i integracji, - konserwacji	- prostota, - ułatwia planowanie i monitorowanie postępów	- może być stosowany gdy jest możliwość precyzyjnego określenia wymagań; - narzuca twórcom SI ścisłą kolejność wykonywania prac - długa przerwa w kontaktach z klientem. - wysoki koszt następstw błędów popełnianych we wstępnych fazach.
MODEL SPIRALNY proces tworzenia ma postać spirali, której każda pętla reprezentuje jedną fazę procesu.	- planowania, - analizy ryzyka, - konstrukcji, - atestowania- rozpoczęcie nowego cyklu możliwe po akceptacji dotychczasowych prac przez klienta	- regularne kontakty z klientem, - minimalizacja ryzyka porażki, - realizacja przyrostowa, - z punktu widzenia klienta- pełna kontrola postępów prac	- koszty realizacji większe niż w modelu kaskadowym, - pokusa odkładania realizacji szczegółów na później
PROTOTYPOWANIE – MODEL PRZYROSTOWY sposób na uniknięcie zbyt wysokich kosztów błędów popełnionych w fazie określania wymagań.	- ogólne określenie wymagań, - budowa prototypu, - weryfikacja działającego prototypu przez klienta, - uszczegółowienie wymagań - realizacja pełnego systemu według modelu kaskadowego	- szybkie stworzenie działającej wersji systemu - korekta/weryfikacja specyfikacji w oparciu o prototyp - możność prowadzenia szkoleń w trakcje konstrukcji	- koszty budowy prototypu, - kłopoty percepcyjne klienta (prototyp działa- dlaczego system tak długo powstaje?)

69. Istota i znaczenie fazy strategicznej w realizacji przedsięwzięć informatycznych.

Znaczenie fazy strategicznej

- Uniknięcie nieporozumień, nieścisłości dzięki precyzyjnie zdefiniowanym celom projektu - z punktu widzenia klienta.
- Oszacowanie rozmiaru systemu
- Oszacowanie pracochłonności
- Oszacowanie harmonogramu

Faza strategiczna : Ocenia się czy rozpoznane potrzeby klientów mogą być spełnione przy obecnych technologiach sprzętu i/lub oprogramowania.

70. Metody identyfikacji wymagań na systemy informatyczne i ich charakterystyka.

Metodyki

- Postulują przebieg procesu projektowania
- Wyznaczają zadania i kolejność ich wykonywania - co, kiedy i dlaczego powinno być wykonane
- Wskazują zastosowanie odpowiednich technik

Strukturalne

*Dane i procesy modelowane osobno
*Wykorzystują w zasadzie tylko proste typy danych
*Dobrze dostosowane do modelu relacyjnego danych
*Podstawowe techniki
diagramy związków encji (ERD)
hierarchie funkcji (FHD)
diagramy przepływu danych (DFD)

Obiektowe

*Dane i procesy są modelowane łącznie!!!
*Wykorzystują złożone typy danych
*Dostosowane do obiektowego modelu danych
*W przypadku realizacji opartej na relacyjnej b.d. wynikowy obiektowy model danych musi być odpowiednio przekształcony
*Podstawowe techniki
diagramy klas UML
przypadki użycia,
modele dynamiczne UML

71. Znaczenie modelowania obiektowego funkcjonowania informatyzowanej organizacji w ustalaniu wymagań na system informatyczny.

Obiektowe modelowanie funkcjonowania organizacji określane jest mianem modelowania biznesowego. Służy do opisu wszystkiego, co składa się na daną organizację (dokumentacji, procedur i procesów biznesowych). Dzięki modelom biznesowym można odpowiedzieć na 6 kluczowych pytań dla każdej organizacji biznesowej: co, jak, dlaczego, kto, gdzie i kiedy.
Modelowanie biznesowe pozwoli zrozumieć czym zajmuje się dane przedsiębiorstwo, czemu akurat tym i czemu akurat w taki sposób. Ponadto można stwierdzić za co odpowiedzialne są konkretne osoby w analizowanej firmie, jaka jest ich rola, czy też zakres współpracy z innymi pracownikami.
Zasadniczym celem budowy modelu biznesowego jest utworzenie takiego obrazu organizacji, który będąc opisem rzeczywistości stanie się podstawą szkieletu aplikacji (opisem tego szkieletu).
Zrozumienie istoty procesów biznesowych jest podstawą specyfikacji wymagań oraz analizy i projektowania systemów informatycznych. Wiele metodyk przypisuje temu zagadnieniu wysoki priorytet.

72. Pożądana zawartość dokumentu „Wymagania na system informatyczny".

1. **Wprowadzenie** – cele, zakres i kontekst systemu
2. **Opis wymagań funkcjonalnych** - opisują funkcje wykonywane przez system. Mogą być opisywane za pomocą: Języka naturalnego, Języka naturalnego strukturalnego, Tablic i Formularzy, Diagramów blokowych, Diagramów kontekstowych i DPU
3. **Opis wymagań niefunkcjonalnych -** Wymagania niefunkcjonalne – opisują ograniczenia, przy których system ma realizować swoje funkcje;
4. **Opis ewolucji systemu –** rozwój systemu
5. **Model logiczny systemu, opracowany przy użyciu np. języka UML** - Model logiczny to zbiór informacji określający zachowanie się systemu. elementy: lista funkcji systemu, określenie obiektów zew. systemu, określenie współzależności między funkcjami systemu
6. **Słownik terminów niejednoznacznych na udziałowców przedsięwzięcia projektowego** - Słownik zawiera terminy, które mogą być niejednoznacznie interpretowane przez uczestników przedsięwzięcia informatycznego- co może być przyczyną nieporozumień podczas jego realizacji;

73. Podstawowe rezultaty fazy modelowania systemu.

1) diagram klas
2) diagram przypadków użycia SI
3) diagramy sekwencji
4) diagramy stanów
5) diagramy czynności
6) raport zawierający definicje i opisy klas, atrybutów, związków, metod, itd.
7) słownik danych, zawierający specyfikację modelu poprawiony dokument opisujący wymagania
8) harmonogram fazy projektowania

74. Podstawowe zadania realizowane w procesie budowy obiektowego modelu systemu informatycznego.

Proces tworzenia modelu obiektowego:

- Identyfikacja klas i obiektów
- Identyfikacja związków pomiędzy klasami
- Identyfikacja i definiowanie pól (atrybutów)
- Identyfikacja i definiowanie metod i komunikatów

Czynności te są wykonywane iteracyjnie. Kolejność ich wykonywania nie jest ustalona i zależy zarówno od stylu pracy, jak i od konkretnego problemu.

75. Rodzaje modyfikacji wprowadzanych w fazie pielęgnacji systemu informatycznego.

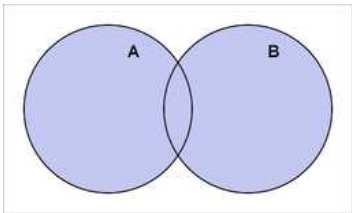
Faza pielęgnacji rozpoczyna się po fazie testowania, kiedy klient rozpoczyna użytkowanie systemu. Jest to najczęściej najdłuższa faza cyklu życia oprogramowania, gdyż obejmuje okres eksploatacji oprogramowania jak i jego utrzymania.
Pielęgnacja oprogramowania polega na wprowadzeniu modyfikacji właściwości użytkowych. Istnieją trzy rodzaje takich modyfikacji:

- **Modyfikacje poprawiające** – polegają na usuwaniu z oprogramowania błędów powstałych bądź wykrytych w poprzednich fazach
- **Modyfikacje ulepszające** – polegają na poprawie jakości oprogramowania,
- **Modyfikacje dostosowujące** – polegają na dostosowaniu oprogramowania do zmian zachodzących w wymaganiach użytkownika lub w środowisku komputerowym

76. Działania na zbiorach.

Suma zbiorów

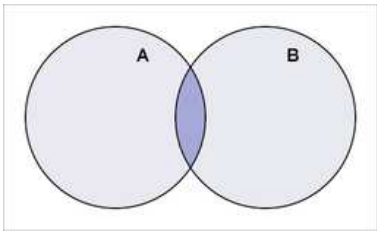
Sumą zbiorów *A* i *B* nazywamy zbiór tych elementów, które należą do zbioru *A* lub do zbioru *B*, matematycznie zapisujemy ją tak: $A \cup B = \{x : x \in A \vee x \in B\}$.



Iloczyn zbiorów

Iloczynem zbioru *A* i *B* nazywamy zbiór tych elementów, które należą jednocześnie do zbioru *A* i do zbioru *B*, formalnie zapisujemy ją tak: $A \cap B = \{x : x \in A \wedge x \in B\}$.

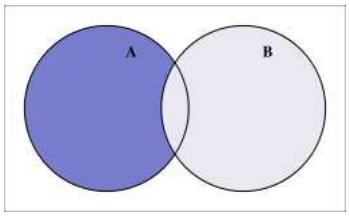
Iloczyn zbiorów nazywany jest także częścią wspólną zbiorów lub przekrojem zbiorów.



Różnica zbiorów

Różnicą zbiorów A i B nazywamy zbiór tych elementów, które należą do zbioru *A*, a które nie należą do zbioru *B*, możemy ją zapisać tak: $A \backslash B = \{x : x \in A \wedge x \notin B\}$.

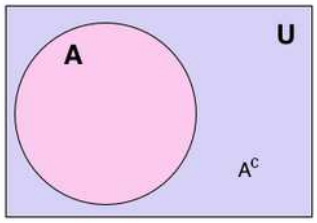
Różnica zbiorów *A* i *B* zapisywana jest też *A* − *B*.



Dopełnienie zbioru

Dopełnieniem zbioru A z przestrzeni *U* nazywamy zbiór tych elementów przestrzeni *U*, które **nie należą** do zbioru *A*. Dopełnienie zbioru *A* oznaczamy jako *A*' lub A^c. Dopełnienie możemy zapisać tak: $A' = \{x : x \in U \wedge x \notin A\}$.

Z definicji dopełniania wynika także, że jest to po prostu różnica przestrzeni *U* i zbioru *A*: $A' = U \backslash A$. Zbiór *U* zwany jest zbiorem **uniwersum**. Czasami zamiast *U* używa się innego oznaczenia przestrzeni np. *X*.



$(A \cup B)' = A' \cap B'$ – *I prawo De Morgana*
$(A \cap B)' = A' \cup B'$ – *II prawo De Morgana*
$A \cup B = B \cup A$ – przemienność dodawania zbiorów
$A \cap B = B \cap A$ – przemienność mnożenia zbiorów
$(A \cup B) \cup C = A \cup (B \cup C)$ – łączność dodawania zbiorów
$(A \cap B) \cap C = A \cap (B \cap C)$ – łączność mnożenia zbiorów
$A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$ – rozdzielność dodawania zbiorów względem mnożenia
$A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$ – rozdzielność mnożenia zbiorów względem dodawania

77. Rachunek zdań.

W klasycznym rachunku zdań przyjmuje się założenie, że każdemu zdaniu można przypisać jedną z dwu wartości logicznych - prawdę albo fałsz, które umownie przyjęto oznaczać 1 i 0. Klasyczny rachunek zdań jest więc dwuwartościowym rachunkiem zdań.
Wartość logiczną zdań określa funkcja prawdy, związana z każdym spójnikiem zdaniowym. Wartość ta zależy wyłącznie od prawdziwości lub fałszywości zdań składowych. Szczególną rolę w rachunku zdań odgrywają takie zdania złożone, dla których wartość logiczna jest równa 1, niezależnie od tego, jakie wartości logiczne mają zdania proste, z których się składają. Takie zdania nazywa się prawami rachunku zdań lub tautologiami.
Zmienne zdaniowe: p, q, r, s, itd.
Funktory: koniunkcja, alternatywa, równoważność, implikacja, itd.
Znaki pomocnicze: nawiasy: (,), [,], {, }.

a. prawo przemienności koniunkcji
$(p \wedge q) \Leftrightarrow (q \wedge p)$
b. prawo przemienności alternatywy
$(p \vee q) \Leftrightarrow (q \vee p)$
c. prawo łączności koniunkcji
$[(p \wedge q) \wedge r] \Leftrightarrow [p \wedge (q \wedge r)]$
d. prawo łączności alternatywy
$[(p \vee q) \vee r] \Leftrightarrow [p \vee (q \vee r)]$
e. prawo rozdzielności koniunkcji względem alternatywy
$[p \wedge (q \vee r)] \Leftrightarrow [(p \wedge q) \vee (p \wedge r)]$
f. prawo rozdzielności alternatywy względem koniunkcji
$[p \vee (q \wedge r)] \Leftrightarrow [(p \vee q) \wedge (p \vee r)]$
g. pierwsze prawo De Morgana
h. (prawo zaprzeczenia koniunkcji)
$\neg(p \wedge q) \Leftrightarrow (\neg p \vee \neg q)$
i. drugie prawo De Morgana
j. (prawo zaprzeczenia alternatywy)
$\neg(p \vee q) \Leftrightarrow (\neg p \wedge \neg q)$

78. Działania na macierzach.

Dodawanie i mnożenie przez skalar
Mnożenie macierzy przez stałą (skalar, element pierścienia) oraz dodawanie macierzy definiuje się następująco:
- $a \cdot (a_{ij}) = (a \cdot a_{ij})$
- $(a_{ij}) + (b_{ij}) = (a_{ij} + b_{ij})$.

Słownie: mnożenie macierzy przez skalar polega na wymnożeniu przez niego wszystkich jej elementów, a dodawanie macierzy na dodaniu odpowiadających sobie elementów. Analogicznie definiuje się różnicę macierzy.
Mnożenie przez skalar spełnia warunki:
- $(a+b) \cdot A = a \cdot A + b \cdot A$,
- $a \cdot (A+B) = a \cdot A + a \cdot B$,
- $(ab) \cdot A = a \cdot (b \cdot A)$,
- $1 \cdot A = A$,

Przykład mnożenia przez skalar

$$3 \cdot \begin{bmatrix} 1 & 2 & 0 \\ -1 & -4 & 9 \end{bmatrix} = \begin{bmatrix} 3 \cdot 1 & 3 \cdot 2 & 3 \cdot 0 \\ 3 \cdot (-1) & 3 \cdot (-4) & 3 \cdot 9 \end{bmatrix} = \begin{bmatrix} 3 & 6 & 0 \\ -3 & -12 & 27 \end{bmatrix}$$

79. Układy równań liniowych – twierdzenie Kroneckera-Capelliego, wzory Cramera.

Układy równań liniowych

Równaniem liniowym o n niewiadomych $x_1, x_2, \ldots, x_n$ nazywamy równanie postaci

$a_1x_1 + a_2x_2 + a_3x_3 + \ldots + a_nx_n = b$ **Układem dwóch równań liniowych o dwóch niewiadomych** nazwiemy wyrażenie postaci

$$\begin{cases} c_1x + b_1y = r_1 \\ c_2x + b_2y = c_2 \end{cases}$$

Układ n równań liniowych o n niewiadomych ma postać

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{n1}x_1 + a_{n2}x_2 + a_{n3}x_3 + \ldots + a_{nn}x_n = b_n \end{cases}$$

Wzory Cramera

Najpierw wyznaczmy tzw wyznacznik główny W

$$W = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \ldots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & a_{nn} \end{vmatrix}$$

Następnie utwórzmy $n$wyznaczników $W_1, W_2, W_3 \ldots, W_n$

$$W_1 = \begin{vmatrix} b_1 & a_{12} & a_{13} & \ldots & a_{1n} \\ b_2 & a_{22} & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ b_n & a_{n2} & a_{n3} & \ldots & a_{nn} \end{vmatrix}, \quad W_2 = \begin{vmatrix} a_{11} & b_1 & a_{13} & \ldots & a_{1n} \\ a_{21} & b_2 & a_{23} & \ldots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & b_n & a_{n3} & \ldots & a_{nn} \end{vmatrix}, \ldots.$$

$$W_n = \begin{vmatrix} a_{11} & a_{12} & a_{13} & \ldots & b_1 \\ a_{21} & a_{22} & a_{23} & \ldots & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \ldots & b_n \end{vmatrix}$$

Dla w. w. wyznaczników zachodzą 3 przypadki

1. $W \neq 0 \Rightarrow x_1 = \frac{W_1}{W}, x_2 = \frac{W_2}{W}, x_3 = \frac{W_3}{W}, \ldots, x_n = \frac{W_n}{W}$
2. $W = 0 \wedge$nie każdy $W_i = 0, \ i = 1, 2, 3, \ldots, n \Rightarrow$układ jest sprzeczny
3. $W = 0 \wedge W_1 = 0, W_2 = 0, W_3 = 0, \ldots, W_n = 0 \Rightarrow$Układ jest nieoznaczony $\Leftrightarrow$ ma nieskończenie wiele rozwiązań.

Twierdzenie Kroneckera-Capelli'ego

Dla układu $m$równań liniowych (jednorodnych albo niejednorodnych) o n niewiadomych ($m < n \vee m = n \vee m > n$)

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 + \ldots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 + \ldots + a_{2n}x_n = b_2 \\ \ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots \\ a_{m1}x_1 + a_{m2}x_2 + a_{m3}x_3 + \ldots + a_{mn}x_n = b_m \end{cases}$$

utwórzmy macierz W ze współczynników $a_{11}, a_{12}, \ldots, a_{1n}, \ldots, a_{mn}$

$$W = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} \end{bmatrix}$$

Przez $r(W)$ oznaczamy **rząd** macierzy W - największy stopień (różny od 0) minora z tej macierzy. (Jeśli wszystkie elementy macierzy wynoszą 0 to $r(W) = 0$).

Dopisując do macierzy W kolumnę wyrazów wolnych, otrzymujemy macierz uzupełnioną U.

$$U = \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} & b_1 \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} & b_2 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{m1} & a_{m2} & a_{m3} & \dots & a_{mn} & b_m \end{bmatrix}$$

Twierdzenie Kroneckera-Capelli'ego mówi: warunkiem koniecznym i wystarczającym rozwiązalności układu równań liniowych jest równość rzędu macierzy W współczynników układu i rzędu macierzy uzupełnionej U
$r(W) = r(U) = r$
Gdy:
1. $r = n \Rightarrow$układ ma **dokładnie jedno rozwiązanie**
2. $r < n \Rightarrow$układ ma **nieskończenie wiele rozwiązań** zależnych od $n - r$ parametrów

80. Pojęcie relacji i funkcji.

	RELACJA	**FUNKCJA**
Definicja	Relacją dwuczłonową R nazywa się dowolny podzbiór iloczynu kartezjańskiego A × B dowolnych zbiorów A i B. **R ⊂ A × B** Własności relacji: • zwrotna (Z), jeśli ∀x ∈ A : xRx • przeciwzwrotna (PZ), jeśli ∀x ∈ A : ~ xRx • symetryczna (S), jeśli ∀x ∈ A, y ∈ A : xRy ⇒ yRx • antysymetryczna (AS), jeśli ∀x ∈ A, y ∈ A : xRy ∧ yRx ⇒ x = y • przechodnia (P), jeśli ∀x, y, z ∈ A : xRy ∧ yRz ⇒ xR	Przyporządkowanie każdemu elementowi zbioru X dokładnie jednego elementu ze zbioru Y. Funkcja jest relacją dwuczłonową. Funkcje można określić za pomocą: - grafu - wykresu - wzoru - zbioru - tabelki - opisu słownego
Pojęcia związane	R jest relacją równoważności, gdy 1) R jest zwrotne, 2) R jest symetryczne, 3) R jest przechodnie, R jest relacją częściowego porządku, gdy 1) R jest zwrotne, 2) R jest słabo antysymetryczne, 3) R jest przechodnie, R jest relacją liniowego porządku, gdy 1)R jest relacją częściowego porządku, 2)R jest relacją spójną,	**Dziedzina funkcji** - zbiór X **Zbiór wartości funkcji** - zbiór Y **Miejsce zerowe funkcji** - argument x dla którego funkcja przyjmuje wartość 0, na wykresie jest to punkt przecięcia z osią X **Monotoniczność funkcji** - określa czy funkcja jest rosnąca czy malejąca. Funkcja jest **rosnąca** gdy wraz ze wzrostem argumentów rosną wartości funkcji Funkcja jest **malejąca** gdy wraz ze wzrostem argumentów maleją wartości funkcji

81. Własności relacji: relacje porządkujące; relacje równoważności.

Relacją w A_1 x ... x A_n (iloczyn kartezjański) nazywamy dowolny zbiór R ⊂ A_1 x ... x A_n.
Nazywamy ją relacją n-argumentową. Relacje dwuargumentowe to R ⊂ A x B.
xRy ↔ (x,y) ∈ R
Np. $R = \{x \in R^2 : x < y\}$

Relacja równoważności
R⊂ X^2 jest relacją
- Zwrotną $\Lambda_{x \in X} xRx$
- Symetryczną $\Lambda_{x,y \in X}(xRy \rightarrow yRx)$
- Przechodnią $\Lambda_{x,y,z \in X}(xRy \wedge yRz \rightarrow xRz)$

Np.
xRy ↔ x=y na zbiorze R
xRy ↔ |x|=|y| na zbiorze R

Relacje porządkujące:
Częściowo porządkująca
R⊂ X^2 jest relacją
- Zwrotną $\Lambda_{x \in X} xRx$
- Słabosymetryczną $\Lambda_{x,y \in X}(xRy \wedge yRx \rightarrow x = y)$
- Przechodnią $\Lambda_{x,y,z \in X}(xRy \wedge yRz \rightarrow xRz)$

Porządkująca i dodatkowo spójna
Relacja porządkująca, która jest spójna $\Lambda_{x,y \in X}(x \neq y \rightarrow xRy \vee yRx)$
Przykłady xRy ↔ x≤y

82. Własności funkcji: miejsca zerowe, ciągłość, pochodna.

Ciągłość
Niech funkcja f będzie określona w pewnym otoczeniu U punktu x_0
Funkcję f nazywamy ciągłą w punkcie x_0, jeżeli istnieje jej granica w tym punkcie i
$\lim_{x \to x_0} f(x) = f(x_0)$
Funkcja f jest ciągła w przedziale otwartym (a, b), jeżeli jest ciągła w każdym punkcie $x_0 \in$(a,b)
Funkcja f jest ciągła w przedziale domkniętym <a, b>, jeżeli spełnia następujące warunki
- Jest ciągła w (a,b)
- $\lim_{x \to a^+} f(x) = f(a)$ (funkcja prawostronnie ciągła w punkcie a),
- $\lim_{x \to a^-} f(x) = f(b)$ (funkcja lewostronnie ciągła w punkcie b).

Miejsce zerowe (pierwiastek) funkcji to argument, dla którego dana funkcja przyjmuje wartość 0.

Niech $U \subset \mathbb{R}$ będzie przedziałem otwartym i funkcja $f : U \to \mathbb{R}$.

Jeśli dla pewnego $x_0 \in U$ istnieje skończona granica ilorazu różnicowego

$$\lim_{x \to x_0} \frac{f(x) - f(x_0)}{x - x_0} = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h},$$

to mówimy, że f jest **różniczkowalna** w punkcie x_0.
Wartość powyższej granicy nazywamy **pochodną funkcji** f w punkcie x_0 i oznaczamy symbolem $f'(x_0)$.

83. Zmienna losowa i jej charakterystyki liczbowe.

Pojęcie zmiennej losowej: Intuicyjne można powiedzieć, że zmienna losowa (związana z pewnym doświadczeniem), to taka zmienna, która w wyniku doświadczenia przyjmuje **wartość** liczbową zależną od przypadku (nie dając ą się ustalić przez przeprowadzeniem doświadczenia).
Wartość oczekiwana. Obliczenie wartości oczekiwanej zmiennej losowej bezpośrednio za pomocą funkcji prawdopodobieństwa lub gęstości prawdopodobieństwa zmiennej losowej X przedstawia następujący wzór :
1. Zmienna losowa dyskretna – Niech X będzie zmienną losową typu dyskretnego. Wartością oczekiwaną nazywa się sumę iloczynów wartości tej zmiennej losowej oraz prawdopodobieństw z jakimi są one przyjmowane.
2. Zmienna losowa ciągła - Jeżeli X jest zmienną losową typu ciągłego zdefiniowaną na przestrzeni probabilistycznej (Ω, P, F), to wartość oczekiwaną zmiennej losowej definiuje się jako całkę.

$$E(x)=\begin{cases}\sum_{i=1}^{n} x_i p_i & 1\\ \int_{-\infty}^{+\infty} XdP & 2\end{cases}$$

Wariancja zmiennej losowej. jest średnią arytmetyczną kwadratów odchyleń (różnic) poszczególnych wartości cechy od wartości oczekiwanej.

1. $D^2X = E(X^2)-(EX)^2,$ 2. $D^2X = E[X-E(X)]^2,$

Odchylenie standardowe zmiennej losowej. Odchylenie standardowe zmiennej losowej X jest to pierwiastek z jej wariancji i opisany jest wzorem:

$\sigma = \sqrt{D^2X},$

84. Cel i rodzaje programowania deklaratywnego.

Programowanie deklaratywne - polega na określeniu oczekiwanych własności rozwiązania które ma być uzyskane bez podawania szczegółowego algorytmu jego uzyskania.

Programowanie funkcyjne:
- nie ma zmiennych (stanu programu) - tylko argumenty funkcji
- nie ma trwałych efektów ubocznych wykonania funkcji
- nie ma instrukcji iteracyjnych
- instrukcjom iteracyjnym odpowiada rekurencja,
- do rozwiązania bardziej złożonych problemów definiuje sie liczne funkcje pomocnicze

Przykładowe języki: LISP, Haskell, można ten styl stosować w większości języków strukturalnych

Programowanie w logice:
- Program to sformułowanie własności rozwiązania w języku formalnym pozwalającym na definiowanie predykatów.
- Wykonanie programu polega na znalezieniu takich danych, dla których można "udowodnić" spełnienie wymaganej własności .
- Zadaniem kompilatora i interpretera programu logicznego jest znalezienie ciągu operacji prowadzących do rozwiązania problemu (udowodnienia własności

Przykładowe języki: Prolog, SQL

85. Definicje unifikatora (podstawienia uzgadniającego), najogólniejszego unifikatora, algorytm unifikacji i twierdzenie o unifikacji.

Podstawienie uzgadniające dwóch termów jest to takie podstawienie, które czyni te termy identycznymi.
Najbardziej ogólne podstawienie uzgadniające dwóch termów to takie podstawienie uzgadniające, w którym wspólna instancja jest w najbardziej ogólnej postaci.
Algorytm unifikacji: Ideę algorytmu unifikacji można tłumaczyć jako kolejne przeszukiwania struktury wyrażeń do unifikacji w celu znalezienia niespójnych elementów i zastąpienie jednego z nich innym.

86. Programy Horna, SLD-rezolucja, odpowiedzi poprawne, odpowiedzi obliczone, poprawność i zupełność SLD-rezolucji.

Klauzula i program Horna
Klauzula Horna jest to dowolna klauzula, zawierająca co najwyżej jeden literał pozytywny.
Programem Horna nazywamy skończony zbiór klauzul Horna.
SLD-rezolucja, jest podstawową regułą wnioskowania wykorzystywaną w programowaniu deklaratywnym. SLDrezolucja to algorytm służący do udowodnienia formuły logiki pierwszego rzędu z zestawu klauzul Horna. Jest oparta na liniowej rezolucji.
Def. Odpowiedź poprawna jest pojęciem deklaratywnym, związanym ze znaczeniem programu.
Def. Odpowiedź obliczona jest pojęciem proceduralnym związanym z wykonaniem programu.
O poprawności rezolucji liniowej Jeśli na podstawie metody rezolucji ze zbioru klauzul zostanie wyprowadzona klauzula pusta, to zbiór klauzul jest niespełnialny.
O zupełności rezolucji liniowej Odpowiedzi poprawne pokrywają się ze wszystkimi możliwymi konkretyzacjami odpowiedzi obliczonych.

87. Zasada rezolucji liniowej w programowaniu w logice.

1. Rezolucja liniowa:
Rezolucja liniowa jest to strategia wnioskowania za pomocą reguły rezolucji, w której każdą, kolejną parę przesłanek tworzą:
• rezolwenta,
• dowolny aksjomat, negacja dowodzonej hipotezy lub dowolna rezolwenta skonstruowana wcześniej.

2. Przykład:
KOBIETA(Ewa)
MEZCZYZNA(Adam)
~MEZCZYZNA(x)\/~KOBIETA(y)\/~LUBI_WINO(y)\/KOCHA(x,y)
~KOBIETA(x)\/LUBI_WINO(x)

Naszym zadaniem jest, na podstawie powyższych formuł udowodnić prawdziwość formuły
KOCHA(Adam, Ewa).
Pierwszym krokiem jest zanegowanie tej formuły, a następnie dodanie jej w takiej formie do zbioru formuł (przy założeniu, że zbiór ten jest niesprzeczny), z których będziemy prowadzić wywód. Po tym kroku zbiór formuł wygląda następująco:
(1) KOBIETA(Ewa)
(2) MEZCZYZNA(Adam)
(3) ~MEZCZYZNA(x)\/~KOBIETA(y)\/~LUBI_WINO(y)\/KOCHA(x,y)
(4) ~KOBIETA(x)\/LUBI_WINO(x)
(5) ~ KOCHA(Adam, Ewa)
Teraz można już prowadzić wywód. Naszym celem jest uzyskanie klauzuli pustej.

(6) ~KOBIETA(y)\/~LUBI_WINO(y)\/KOCHA(Adam,y) -otrzymane z formuł 2 i 3, przy zastosowaniu reguły nr 2.
(7) LUBI_WINO(Ewa) -otrzymane z formuł 2 i 3, przy zastosowaniu reguły nr 2.
(8) ~KOBIETA(Ewa)\/KOCHA(Adam,Ewa) - otrzymane z formuł 6 i 7, przy zastosowaniu reguły nr 2. (9) KOCHA(Adam,Ewa) -otrzymane z formuł 1 i 8, przy zastosowaniu reguły Modus ponens.
(10) [] -otrzymane z formuł 5 i 9, przy zastosowaniu reguły nr 4.
Jak widać udało nam się uzyskać klauzulę pustą. Świadczy to o tym, iż formuła KOCHA(Adam, Ewa) jest niespełniona, zatem formuła KOCHA(Adam, Ewa) jest spełniona, co należało dowieść.

88. Budowa programu w Prologu: klauzule (fakty, reguły), definicje predykatów. Sposób realizacji programu.

PROGRAM zbiór procedur; kolejność procedur w programie nie jest istotna.
PROCEDURA ciąg klauzul definiujących jeden predykat kolejność klauzul w procedurze jest istotna
KLAUZULA fakt lub reguła.
FAKT Bezwarunkowo prawdziwe stwierdzenie o istnieniu pewnych powiazań (relacji) miedzy obiektami.
Budowa: symbol_relacji (obiekt1, ..., obiektn)
Przykłady **student(marcin)**. Marcin jest studentem.
lubi(ewa, marek). Ewa lubi Marka
REGUŁA: Warunkowe stwierdzenie istnienia pewnych powiazań (relacji) miedzy obiektami.
symbol_relacji (obiekt1, ..., obiektn) :- symbol_relacji_1 (obiekt1, ..., obiektn1),
symbol_relacji_2 (obiekt1, ..., obiektn2)…
Przykład: **powierzchnia (X, Y) :- dlugosc (X, D), szerokosc (X, S), Y is D*S.**
Powierzchnia Y prostokata X jest równa iloczynowi długosci D i szerokosci S tego prostokata.
Rozpoczęcie działania programu polega na wywołaniu dowolnej procedury, które jest nazywane w Prologu zadawaniem pytań lub podawaniem celu.
Celem działania programu jest uzyskanie odpowiedzi na Pytanie o prawdziwość podanych faktów lub polecenie znalezienia nazw obiektów będących w podanej relacji z innymi.
Postać ?- symbol_relacji_1 (obiekt1, ..., obiektn1),
symbol_relacji_2 (obiekt1, ..., obiektn2).
Przykład
?- **lubi (marta, jan).** - Czy Marta lubi Jana?
lubi(marta, X). – Kogo lubi Marta?
lubi(X, marta) –Kto lubi Martę?

89. Co to są systemy (zestawy) funkcjonalnie pełne? Podać przykładowe.

Systemem funkcjonalnie pełnym nazywa się taki zbiór operatorów (funkcji), którymi można przedstawić dowolne wyrażenie logiczne (dowolną funkcję).

Funkcje sumy, iloczynu i negacji tworzą tzw. podstawowy system funkcjonalnie pełny. Nie jest to jednak system minimalny. Systemy funkcjonalnie pełne tworzą również:

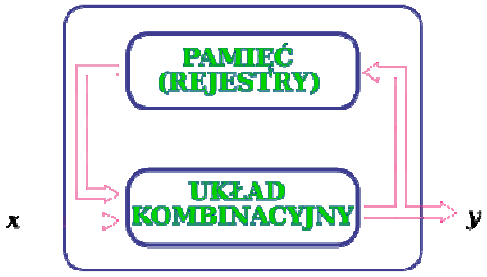
- iloczyn i negacja (suma może zostać wyeliminowana dzięki prawu De Morgana)
- suma i negacja (analogicznie jak wyżej)
- funkcja Sheffer'a (NAND) (jak wyżej oraz ponieważ $\overline{a} = \overline{a \cdot a}$)
- funkcja Pierce'a (NOR) ($\overline{a} = \overline{a + a}$)

90. Podaj rodzaje układów sekwencyjnych oraz różnice w procedurach ich projektowania.

Układ sekwencyjny

Charakteryzuje się tym, że stan wyjść y zależy od stanu wejść x oraz od poprzedniego stanu, zwanego stanem wewnętrznym, pamiętanego w zespole rejestrów (pamięci).

Jeżeli stan wewnętrzny nie ulega zmianie pod wpływem podania różnych sygnałów X, to taki stan nazywa się stabilnym.



Rozróżnia się dwa rodzaje układów sekwencyjnych:

1. asynchroniczne
2. synchroniczne

Układy asynchroniczne

W układach asynchronicznych zmiana sygnałów wejściowych X natychmiast powoduje zmianę wyjść Y. W związku z tym układy te są szybkie, ale jednocześnie podatne na zjawisko hazardu i wyścigu. Zjawisko wyścigu występuje, gdy co najmniej dwa sygnały wejściowe zmieniają swój stan w jednej chwili czasu (np. 11b -> 00b).

Układy synchroniczne

W układach synchronicznych zmiana stanu wewnętrznego następuje wyłącznie w określonych chwilach, które wyznacza sygnał zegarowy (ang. clock). Każdy układ synchroniczny posiada wejście zegarowe oznaczane zwyczajowo symbolami C, CLK lub CLOCK. Charakterystyczne dla układów synchronicznych, jest to, iż nawet gdy stan wejść się nie zmienia, to stan wewnętrzny - w kolejnych taktach zegara - może ulec zmianie.

Różnice polegają na konieczności eliminacji zjawisk szkodliwych w układach asynchronicznych (wyścigów i hazardów).

91. Jakie znasz elementy pamięciowe stosowane układach sekwencyjnych?

Układy synchroniczne produkuje się z przerzutników typu D, T, i JK.
Układy asynchroniczne produkuje się z przerzutników RS lub bez przerzutników (bramki ze sprzężeniem zwrotnym)

92. Co oznacza termin mikroprogramowanie? Do czego służy?

Układ mikroprogramowany to jedno z możliwych rozwiązań stosowanych w technice cyfrowej dla projektów o dużej złożoności. **Projekt dekomponowany jest na dwie części: układ operacyjny (wykonawczy) i układ sterujący.** Układ sterujący można zaprojektować jako układ mikroprogramowany. Bity sterujące, wraz z innymi odpowiednimi bitami służącymi do napisania mikroprogramu, umieszcza się w pamięci ROM.

93. Podaj znane zapisy liczbowe i zakres ich stosowania.

Zapisy można podzielić na **stałopozycyjne (liczby całkowite)** i **zmiennopozycyjne (liczby rzeczywiste)**.
Stałopozycyjne to: ZNAK-MODUŁ, U1 i U2. Najpowszechniejszy jest U2 ze względu na najprostsze algorytmy operacji arytmetycznych.
Zmiennopozycyjny zapis polega na reprezentowaniu liczby za pomocą trzech grup bitów:
1 bit znaku, 23-bitowej mantysy i np. 8-bitowego wykładnika.

8 bitów	char	−128 — +127 (ze znakiem)
16 bitów	short int	−32 768 — +32 767 (ze znakiem)
32 bity	int	−2 147 483 648 — +2 147 483 647 (ze znakiem)

32 bity	float
64 bity	double

94. Co to są mikrokontrolery?

Mikrokontroler - system mikroprocesorowy zrealizowany w postaci pojedynczego układu scalonego, zawierającego jednostkę centralną (CPU), pamięć RAM oraz na ogół pamięć programu i rozbudowane układy wejścia-wyjścia. Określenie *mikrokontroler* pochodzi od głównego obszaru zastosowań, jakim jest sterowanie urządzeniami elektronicznymi.
Mikrokontroler stanowi użyteczny i całkowicie autonomiczny system mikroprocesorowy, nie wymagający użycia dodatkowych elementów, których wymagałby do pracy tradycyjny mikroprocesor. Mikrokontrolery wykorzystuje się powszechnie w sprzęcie AGD, układach kontrolno-pomiarowych, w przemysłowych układach automatyki, w telekomunikacji itp.

95. Jakie parametry są charakterystyczne dla pamięci dynamicznych?

Podstawowymi parametrami opisującymi każdy typ pamięci są:
- Szybkość
- Pojemność
- Pobór mocy
- Koszt

Dla pamięci dynamicznej charakterystycznymi parametrami są:
wymagany czas odświeżania (8-64 ms)
wymaganą liczbę cykli odświeżania (pierwiastek z liczby bitów).

Pojemność każdej pamięci, to ilość informacji którą dana pamięć jest w stanie przechować.
Szybkość jest parametrem określającym jak często procesor, lub inne urządzenie mogą z niej korzystać.

Czasem dostępu - access time. Jest to czas upływający od momentu w którym wysyłane jest żądanie dostępu do pamięci do czasu w którym informacja zwrotna ukaże się na jej wyjściu.
Czasem cyklu - cycle time. Jest to czas najkrótszy, który upływa między dwoma kolejnymi żądaniami dostępu do pamięci.
Szybkością transmisji - Szybkość transmisji mierzymy liczbą bitów, bądź bajtów, którą jesteśmy w stanie przesłać pomiędzy urządzeniem, a pamięcią.
Również pobór mocy stanowi bardzo ważny parametr, który staje się problemem przy budowaniu urządzeń zasilanych bateryjnie, jak komputery kieszonkowe, laptopy.

96. Podaj tryby adresowania dla rozkazów mikrokontrolera i odpowiadający im czas trwania cyklu instrukcyjnego wraz z omówieniem kolejnych cykli maszynowych.

Trybem adresowania nazywa się sposób określenia miejsca przechowywania argumentów rozkazu.
Rozróżniamy następujące tryby adresowania:

- **adresowanie natychmiastowe**, w którym argument rozkazu zawarty jest w kodzie rozkazu. **2 cykle**
- **adresowanie bezpośrednie**, w którym kod rozkazu zawiera adres komórki pamięci przechowującej argument rozkazu. **4 cykle**
- **adresowanie rejestrowe**, w którym w kodzie rozkazu określony jest rejestr zawierający argument rozkazu. **1 cykl**
- **adresowanie pośrednie** (rejestrowe pośrednie), w którym kod rozkazu zawiera określenie rejestru, bądź rejestrów zawierających adres komórki pamięci z argumentem rozkazu. **2 cykle**

Rodzaje cykli:

- Pobranie kodu operacyjnego (fetch)
- Odczyt z pamięci (memory read)
- Zapis do pamięci (memory write)
- Odczyt z urządzenia wejściowego (IOread)
- Zapis do urządzenia wyjściowego (IOwrite)
- Cykl przyjęcia przerwania (interrupt)
- Cykl zatrzymania (HALT)

97. Jakie znasz rodzaje transmisji szeregowej i na czym one polegają?

Asynchroniczna: Transmisja ta pozwala na transmisję 1 bajtu z parametrami. Zegary w nadajniku i odbiorniku ustawione są na tą samą częstotliwość. Transmisja rozpoczyna się od przesłania bitu startu, następnie przesyłany jest znak i transmisję kończy bit stopu.
Synchroniczna: W transmisji synchronicznej specjalna preambuła dokonuje zsynchronizowania nadawcy i odbiorcy. Po synchronizacji następuje przesłanie danych.

98. Porównaj pod względem szybkości znane rozwiązania operacji mnożenia w systemach wbudowanych.

Mnożenie w układzie kombinacyjnym (układ iteracyjny gdzie wynik pojawia się po czasie opóźnienia wnoszonym przez bramki) jest najszybsze.
Mnożenie w układzie sekwencyjnym (tyle kroków ile jest bitów, a każdy krok to dodawanie warunkowe i przesunięcie) jest wolniejsze.
Mnożenie programowe (bez żadnego sprzętu) jest najwolniejsze.

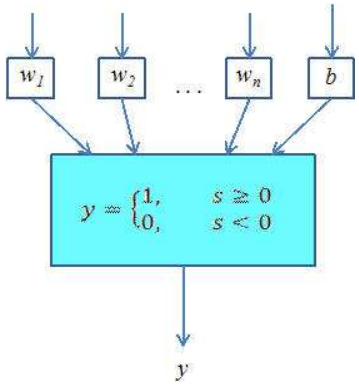
99. Model obliczeniowy perceptronu – możliwości i ograniczenia.

Perceptron - posiada wejścia (*n* zmiennych wejściowych $x_1,...,x_n$) przyjmujące wartości rzeczywiste lub binarne, będące odpowiednikiem dendrytów. Wejścia te albo są połączone z innymi perceptronami, albo stanowią wejścia (np. czujniki) całego układu. Wyjście stanowi pojedynczą wartość 0 lub 1.
Parametry wewnętrzne perceptronu to *n* wag połączeń $w_1,...,w_n$ (liczb rzeczywistych), symulujących wagi synaps łączących naturalne neurony. Ostatnim parametrem jest wartość odchylenia *b* (ang. *bias*) odpowiadająca za nieliniowe przekształcenie wejść w wyjście.

Zasada działania

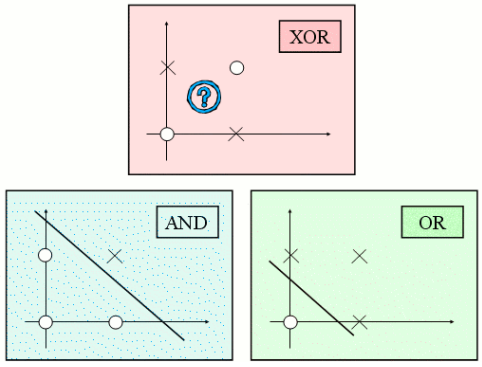
Do każdego *i*-tego wejścia perceptronu przypisana jest waga w_i. Dla danych stanów wejściowych $x_1,...,x_n$ liczymy sumę ważoną:

$$s = \sum_{i=1}^{n} x_i w_i + b$$



Jeżeli *s* jest większa lub równa 0, to ustawiamy wyjście $y=1$, zaś w przeciwnym przypadku ustawiamy $y=0$. Perceptron opisany jest więc jednoznacznie przez zbiór wag$w_1,...,w_n$ oraz wartość odchylenia *b*. Schemat działania:

Pojedynczy perceptron nie potrafi odróżniać zbiorów nieseparowalnych liniowo, tzn. takich, że między punktami z odpowiedzią pozytywną i negatywną nie da się poprowadzić prostej rozgraniczającej. Jednym z najprostszych przykładów takich zbiorów jest funkcja logiczna XOR (alternatywa wykluczająca). Jak widać na poniższych rysunkach, funkcje AND i OR dają zbiory separowalne liniowo, a więc możliwe do realizacji perceptronem.



100. Metody uczenia sieci neuronowych.

Pod pojęciem uczenia sieci rozumiemy wymuszenie na niej określonej reakcji na zadane sygnały wejściowe. Uczenie jest konieczne tam, gdzie brak jest informacji doświadczalnych o powiązaniu wejścia z wyjściem lub jest ona niekompletna, co uniemożliwia szczegółowe zaprojektowanie sieci. Uczenie może być realizowane krok po kroku lub poprzez pojedynczy zapis.

Uczenie z nadzorem

Stosuje się tylko wówczas, gdy istnieje możliwość zweryfikowania poprawności odpowiedzi udzielanych przez sieć; oznacza to, że dla każdego wektora wejściowego musi być znana dokładna postać wektora wyjściowego (pożądana odpowiedź). Celem uczenia pod nadzorem jest minimalizacja odpowiednio zdefiniowanej funkcji celu

Uczenie bez nadzoru

Stosuje się wówczas gdy nie znamy oczekiwanych odpowiedzi na zadany wzorzec. Podczas uczenia bez nauczyciela pożądana odpowiedz nie jest znana. Ze względu na brak informacji o poprawności, czy niepoprawności odpowiedzi sieć musi się uczyć poprzez analizę reakcji na pobudzenia. W trakcie analizy parametry sieci podlegają zmianom, co nazywamy samoorganizacją.

101. Mechanizm działania algorytmu genetycznego.

Algorytm genetyczny - rodzaj algorytmu przeszukującego przestrzeń alternatywnych rozwiązań problemu w celu wyszukania rozwiązań najlepszych.
Sposób działania algorytmów genetycznych nieprzypadkowo przypomina zjawisko ewolucji biologicznej. Obecnie zalicza się go do grupy algorytmów ewolucyjnych.

Najczęściej działanie algorytmu przebiega następująco:

1. Losowana jest pewna populacja początkowa.
2. Populacja poddawana jest ocenie (**selekcja**). Najlepiej przystosowane osobniki biorą udział w procesie reprodukcji.
3. Genotypy wybranych osobników poddawane są operatorom ewolucyjnym:
   - są ze sobą kojarzone poprzez złączanie genotypów rodziców (**krzyżowanie**),
   - przeprowadzana jest mutacja, czyli wprowadzenie drobnych losowych zmian.
4. Rodzi się drugie (kolejne) pokolenie i algorytm powraca do kroku drugiego, jeżeli nie znaleziono dostatecznie dobrego rozwiązania. W przeciwnym wypadku uzyskujemy wynik.

102.Definicja entropii informacji i wybrane zastosowanie tego pojęcia.

Entropia – w ramach teorii informacji jest definiowana jako średnia ilość informacji, przypadająca na pojedynczą wiadomość ze źródła informacji. Innymi słowy jest to średnia ważona ilości informacji niesionej przez pojedynczą wiadomość, gdzie wagami są prawdopodobieństwa nadania poszczególnych wiadomości.
Wzór na entropię:

$$H(x) = \sum_{i=1}^{n} p(i) \log_r \frac{1}{p(i)} = -\sum_{i=1}^{n} p(i) \log_r p(i)$$

gdzie *p(i)* – prawdopodobieństwo zajścia zdarzenia *i*,
n – liczba wszystkich zdarzeń danej przestrzeni.

- przy ocenie bezpieczeństwa bezwarunkowego systemów szyfrowania
- w uczeniu maszynowym przez zastosowanie ukrytych modeli Markowa (HMM)
- porównywanie modeli językowych w lingwistyce komputerowej
- rekonstrukcja obrazu w tomografii komputerowej dla zastosowań medycznych

103.Metody generacji reguł systemu decyzyjnego, minimalne, pokrywające, wyczerpujące.

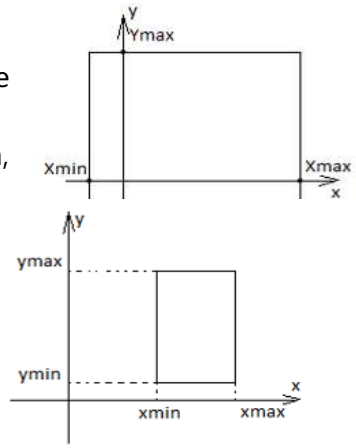
Reguły systemu decyzyjnego są jednym z narzędzi reprezentacji wiedzy. Opisują one związki między atrybutami warunkowymi, a atrybutem decyzyjnym. Systemy decyzyjne oparte na regułach zawierają zwykle wiele reguł, z których każda może być oparta na innym zestawie atrybutów warunkowych.
Metody minimalne - metody tworzenia systemu decyzyjnego poprzez wygenerowanie minimalnej ilości reguł ze zbioru treningowego. Przykład: LEM2
Metody pokrywające - idea: nauczyć sie reguły, po czym usunąć obiekt, który ta reguła pokrywa - powtarzać aż do wyczerpania zbioru obiektów. Przykład: Pokrywanie sekwencyjne.
Metody wyczerpujące - Polega na przeszukiwaniu wszystkich deskryptorów warunkowych obiektów treningowych począwszy od kombinacji długości 1 w celu znalezienia kombinacji , która jest niesprzeczna w systemie decyzyjnym i która nie zawiera w sobie reguł niższego rzędu. Przykład: EXHAUSTIVE: z najmniejszą ilością deskryptorów.

104. Podaj znaczenie oraz omów wzajemne związki między terminami: przestrzeń urządzenia, przestrzeń operacyjna urządzenia, powierzchnia obrazowania, macierz adresowalna, jednostka rastru, krok kreślaka.

Przestrzeń urządzenia (obrazującego) - to obszar zdefiniowany przez układ współrzędnych, ograniczony pojemnością rejestrów pozycji x i y w urządzeniu graficznym
Przestrzeń operacyjna - to obszar przestrzeni urządzenia którego zawartość została odwzorowana na powierzchni obrazowania
Powierzchnia obrazowania - nośnik, na którym mogą pojawiać się obrazy i rysunki, np. ekran lampy kineskopowej lub papier w ploterze.
Macierz adresowalna - to macierz utworzona z punktów adresowalnych określająca wielkość obrazu który można przesłać do generatora obrazu
Jednostka rastru - jednostką rastru jest piksel

105. Zdefiniuj okno i widok oraz oknowanie i obcinanie; podaj co to są współrzędne znormalizowane i do czego one służą.

Okno – ograniczony obszar w obrębie przestrzeni obrazowania, który może być rozszerzony do całej przestrzeni obrazowania.
Widok – ograniczony obszar w obrębie przestrzeni operacyjnej urządzenia, który prezentuje zawartość okna. Widok może być rozszerzony do całej przestrzeni operacyjnej. Obszar (prostokątny) na urządzeniu graficznym
Okienkowanie – wyizolowanie dowolnej części wyświetlonego obrazu .
Obcinanie – wyznaczanie elementów wewnątrz okna.
Współrzędne znormalizowane – Często przejście od współrzędnych rzeczywistych do współrzędnych danego urządzenia graficznego jest wykonywane dwustopniowo. Najpierw przechodzi do współrzędnych znormalizowanych, to znaczy do kwadratu [0,1]x[0,1], a stad do współrzędnych urządzenia.



Widok we wspolrzednych urzadzenia graficznego.

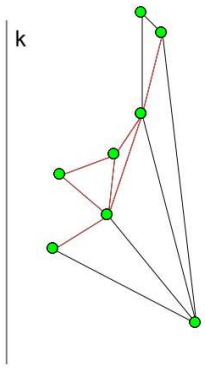
106. Podaj ideę algorytmu Cohena-Sutherlanda obcinania odcinka do prostokątnego okna i jego trzy pierwsze kroki; podaj w jakich współrzędnych działa ten algorytm i dlaczego?

Algorytm Cohena-Sutherlanda jest analitycznym algorytmem obcinania dwuwymiarowych odcinków przez prostokąt obcinający, którego boki są równoległe do osi układu współrzędnych.
Algorytm Cohena-Sutherlanda polega na przypisaniu każdemu końcowi odcinka czterobitowego kodu określającego położenie tego punktu względem prostokątnego okna. Dokładniej, bity są numerowane od prawego: **Kod(P) = $b_4b_3b_2b_1$** b_1 – gdy P leży na lewo od okna b_2 - gdy P leży na prawo od okna b_3 - gdy P leży poniżej okna b_4 - gdy P powyżej okna
a w przeciwnym razie odpowiednie bity kodu mają wartość 0.
Algorytm przebiega następująco:
1. Wybierany jest koniec odcinka leżący poza prostokątem, a więc mający niezerowy kod; jeśli kody obu końców są niezerowe to można wybrać dowolny z nich.
2. Wyznaczany jest punkt przecięcia odcinka z jedną z prostych. Wybór prostych determinuje kod wybranego końca.
3. Następnie odcinek jest przycinany do tego punktu - koniec wybrany w punkcie pierwszym jest zastępowany przez punkt przecięcia.

107. Sformułuj zadanie i ogólne warunki trójkątowania wielokąta, warunki trójkątowania naturalnego oraz podaj ideę trójkątowania wielokąta monotonicznego.

Triangulacja wielokątów jest przeprowadzana w celu uproszczenia kształtów powierzchni, przez co ułatwia się rozwiązywanie zadań typu: wypełnianie obszarów, określenie zasłaniania, oświetlenia i przecinania się obiektów 3W.

Zadanie triangulacji formułuje się jako podział wielokąta zwykłego na sumę nie nakładających się na siebie trójkątów, których wierzchołkami mogą być tylko wierzchołki danego wielokąta.



Ogólne warunki:

1. Liczba trójkątów powinna być jak najmniejsza (tj. równa n-2 dla n-kąta)
2. trójkąty nie mogą się nakładać
3. ich wierzchołkami mogą być tylko wierzchołki wielokąta

W praktyce polega to na przeglądaniu odpowiednio uporządkowanych wierzchołków i tam, gdzie będzie to możliwe poprowadzenie przez kolejny wierzchołek przekątnej. Przekątna odetnie trójkąt pozostawiając prostszy wielokąt do dalszej triangulacji.

Triangulacja wielokąta monotonicznego - Algorytm (zachłanny, metoda „zamiatania")

1. Porządkujemy wierzchołki wzdłuż kierunku wyznaczonego przez prostą k (względem której wielokąt jest monotoniczny). Wymaga to czasu liniowego, gdyż jest to równoważne ze scalaniem dwóch uporządkowanych ciągów.
2. Poruszając się wzdłuż prostej łączymy krawędzią punkt, do którego doszliśmy z wszystkimi możliwymi punktami, które zostały już odwiedzone, ale jeszcze nie zostały odcięte od reszty wielokąta przez krawędź triangulacji.

108.Zdefiniuj przekształcenie 3-punktowe. Do czego ono służy?.

Przekształcenie trzypunktowe

Jest ono wykorzystywane przy budowie scen trójwymiarowych. Dane są trzy nie współliniowe punkty: P1, P2, P3 i Q1, Q2, Q3.

Szukamy takiej izometrii (przekształcenia), która będzie spełniała następujące warunki:

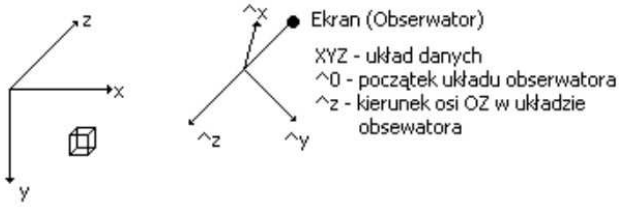
- Odwzorowuje punkt P1 w Q1,
- Kierunek P=P2-P1 w kierunek Q=Q2-Q1,
- Transformuje płaszczyznę wyznaczoną przez : P1, P2, P3 w płaszczyznę wyznaczoną przez Q1, Q2, Q3

109. Podaj wzajemne położenie układów współrzędnych: danych i obserwatora.

Układ współrzędnych obserwatora nie jest określony jednoznacznie. Przyjmuje się, że jego początek O' pokrywa się z początkiem O układu danych.

Aby przekształcić układ danych do układu obserwatora należy:

1. Określić kierunki osi w układzie obserwatora
2. Przedstawić dane we współrzędnych układu obserwatora



110. Wymień we właściwej kolejności operacje, jakie należy wykonać, aby obrócić punkt *P* o kąt φ dookoła prostej zadanej przez dwa punkty P_1 i P_2.

1) Przesunięcie, aby punkt P_1 znalazł się w początku układu współrzędnych.
2) Obrót wokół osi OX.
3) Obrót wokół osi OY. Obroty (etap 2. i 3.) zapewniają, że zadana oś obrotu (prosta) pokryje się (z uwzględnieniem zwrotów) z osią OX układu współrzędnych.
4) Realizacja zadanego obrotu o kąt wokół osi OX.
5) Obrót będący operacją odwrotną do operacji 3.
6) Obrót będący operacją odwrotną do operacji 2.
7) Przesunięcie odwrotne do przesunięcia 1.

111. Opisać 3 podstawowe obszary uzależnień komputerowych.

Środkami uzależniającymi są:

1. gry komputerowe,
2. Internet,
3. programowanie.

Pozbawieni dostępu do komputera przeżywają stany identyczne z zespołem abstynenckim, są pobudzeni, cierpią na zaburzenia snu, popadają w stany depresyjne, fantazjują na temat Internetu. Uzależnienia komputerowe można podzielić na pierwotne i wtórne. W pierwszym przypadku mamy do czynienia z potrzebą przeżycia emocji, uzyskania efektu pobudzenia, sprawdzenia swoich umiejętności. W drugim przypadku komputer jest traktowany jako forma ucieczki od rzeczywistości. Ponadto należy pamiętać, że ten rodzaj uzależnienia ma negatywne konsekwencje dla zdrowia.

Uzależnienie od komputera u dziecka można stwierdzić, gdy:

- udaje, że się uczy, kiedy tymczasem godzinami siedzi przed ekranem monitora,
- w czasie gry lub przeglądania stron WWW wpada w stan przypominający trans,
- próby ograniczenia czasu spędzanego przy komputerze napotykają na silny opór, nawet agresję,
- niechętnie spotyka się z kolegami, brak mu przyjaciół, woli komputer niż ruch na świeżym powietrzu,
- nie potrafi odpoczywać, nie wzrusza się.

112. Wymienić przynajmniej 3 polskie ustawy dotyczące środowiska informatycznego.

Dla środowiska informatycznego najważniejszymi regulacjami prawnymi, które weszły w życie w minionych kilku latach, są ustawy:

- łączności z 1990 roku,
- prawie autorskim i prawach pokrewnych z 1994 roku,
- zamówieniach publicznych z 1994 roku,
- ochronie danych osobowych z 1997 roku.

113. Jaka jest zasadnicza różnica między ochroną własności intelektualnej i ochroną patentową?

Podstawową różnicą prawa patentowego jako systemu ochrony partykularnej jest wyłączenie możliwości uzyskania patentu przez różne podmioty na wynalazki, które są do siebie podobne.
Czyli mając coś opatentowane nie możesz stworzyć czegoś podobnego posiadającego podobne algorytmy, funkcje itp a w prawie autorskim chronisz tylko program a nie algorytm itd.
Jak cos jest chronione patentem to jest chronione jako patent i własność intelektualna, a jak cos jest chronione jako własność intelektualna, to niekoniecznie jest chronione patentem.
Prawo autorskie chroni tylko formę (kod źródłowy i wynikowy). Nie chroni jednak odkryć, idei, procedur, metod, zasad działania, koncepcji matematycznych.

114. Opisać na czym polega szpiegostwo komputerowe.

Szpiegostwo definiuje się jako działanie przestępne dokonywane na szkodę określonego państwa, polegające na wykonywaniu odpowiednich czynności na rzecz obcego wywiadu, a w szczególności na zbieraniu, przekazywaniu informacji organom obcego wywiadu lub na gromadzeniu i przechowywaniu informacji w celu przekazania obcemu wywiadowi.

Artykuł 130 – paragrafy 1 do 4 określają szpiegostwo komputerowe jako:

- Działanie w obcym wywiadzie przeciwko Rzeczypospolitej Polskiej
- Działanie w obcym wywiadzie lub biorąc w nim udział, udziela informacji, które mogą wyrządzić szkodę Rzeczypospolitej Polskiej,
- Kto, w celu udzielenia obcemu wywiadowi wiadomości określonych w § 2, gromadzi je lub przechowuje, włącza się do sieci komputerowej w celu ich uzyskania albo zgłasza gotowość działania na rzecz obcego wywiadu przeciwko Rzeczypospolitej Polskiej,
- Kierowanie lub tworzenie organizacji obcego wywiadu