**Pytania obowiązujące na egzaminie dyplomowym magisterskim**

Kierunek: Informatyka Stosowana

Specjalność: Modelowanie i Technologie Informacyjne

Zawartość

[1.Omów wpływ architektury procesora i układu pamięć-procesor na wydajność obliczeń sekwencyjnych 4](#_Toc341829785)

[2.Podaj techniki optymalizacji obliczeń sekwencyjnych. Jakie są etapy przy tworzeniu kodu sekwencyjnego wysokiej wydajności? 4](#_Toc341829786)

[3.Z czego składa się czas wykonania obliczeń równoległych? Jakie czynniki wpływają na wydajność obliczeń równoległych? Jak można optymalizować tę wydajność? 6](#_Toc341829787)

[4.Omów modele cyklu życia oprogramowania. 7](#_Toc341829788)

[4.1 Model kaskadowy 7](#_Toc341829789)

[4.2 Model spiralny 8](#_Toc341829790)

[4.3 Model ewolucyjny 9](#_Toc341829791)

[4.4 Prototypowanie 9](#_Toc341829792)

[4.5 Wytwarzanie odkrywcze 9](#_Toc341829793)

[4.6 Wytwarzanie przyrostowe 9](#_Toc341829794)

[4.7 Model komponentowy 9](#_Toc341829795)

[5.Przedstaw wybraną metodologię tworzenia oprogramowania (np. RUP, XP). 10](#_Toc341829796)

[5.1 RUP (Rational Unified Process) 10](#_Toc341829797)

[5.2 Programowanie ekstremalne 11](#_Toc341829798)

[5.3 Agile programming 13](#_Toc341829799)

[6.Scharakteryzuj następujące techniki ponownego wykorzystania kodu: komponenty, wzorce, refaktoryzacja, programowanie aspektowe, RAD. 13](#_Toc341829800)

[7.Czym jest język UML? Omów wybrany rodzaj diagramów UML. 14](#_Toc341829801)

[8.Scharakteryzuj model przetwarzania równoległego z pamięcią wspólną i środowisko programowania OpenMP 15](#_Toc341829802)

[Model przetwarzania równoległego z pamięcią wspólną (Parallel Random Access Machine) 15](#_Toc341829803)

[OpenMP (Open Multi-Processing) 16](#_Toc341829804)

[9.Jak mierzymy i wyrażamy wydajność obliczeń równoległych? Czym jest skalowalność i jak ją wyrażamy? 18](#_Toc341829805)

[10.Wyjaśnij ideę gradientowych metod optymalizacji. 19](#_Toc341829806)

[11.Wyjaśnij różnice pomiędzy deterministycznymi i niedeterministycznymi metodami optymalizacji. 19](#_Toc341829807)

[12.Omów ideę algorytmów genetycznych. 20](#_Toc341829808)

[13.Wyjaśnij zasadę budowy i działania sztucznego neuronu. 20](#_Toc341829809)

[14.Wyjaśnij ideę uczenia sieci neuronowej metodą wstecznej propagacji błędu. 21](#_Toc341829810)

[15.Omów główne elementy systemu ekspertowego. 22](#_Toc341829811)

[16.Opisz algorytmy rozwiązania zagadnienia programowania liniowego. 22](#_Toc341829812)

[Metoda geometryczna 23](#_Toc341829813)

[Metoda Simpleks 23](#_Toc341829814)

[17.Co rozumiesz pod pojęciem logistyka? Cele i zadania logistyki w przedsiębiorstwie. 23](#_Toc341829815)

[18.Co to jest prognozowanie? Wymień i krótko opisz modele prognozowania ilościowego. 24](#_Toc341829816)

[19.Co to jest relacyjna baza danych? Opisz rodzaje relacji oraz podaj przykład każdego z typów relacji. Wyjaśnij pojęcie "klucza podstawowego", jakie spełnia on funkcje, oraz podaj przykład "klucza prostego", "klucza wielokrotnego" i klucza obcego. 25](#_Toc341829817)

[Relacyjna baza danych 25](#_Toc341829818)

[Typy relacji 26](#_Toc341829819)

[Klucz podstawowy 26](#_Toc341829820)

[20.Co to jest zapytanie w bazie danych, opisz rodzaje zapytań oraz ich przeznaczenie. Podaj polecenia w języku SQL tworzące te zapytania. 27](#_Toc341829821)

[21.[puste]Podstawowe równania teorii sprężysto - plastycznych odkształceń. 27](#_Toc341829822)

[22.[puste]Podstawy teorii plastycznego płynięcia. 27](#_Toc341829823)

[23.Ogólne zasady metody elementów skończonych. 28](#_Toc341829824)

[24.Przedstaw podstawowe cechy oraz zasady implementacji technologii XML. Omów kilka wybranych technologii stworzonych w oparciu o XML. 28](#_Toc341829825)

[25.Opisz sposób implementacji oprogramowania oparty o architekturę Model-View-Controller. 30](#_Toc341829826)

[26.Przedstaw technologię Single Sign-On (SSO) oraz przykładowe narzędzia wykorzystywane do pojedynczego logowania. 30](#_Toc341829827)

[27.Omów podstawowe metody algorytmiczne: metodę "dziel i zwyciężaj", metodę zachłanną, programowania dynamicznego. Podaj dla każdej z nich przykłady zastosowania. 31](#_Toc341829828)

[Metoda dziel i zwyciężaj. 31](#_Toc341829829)

[Algorytm zachłanny. 31](#_Toc341829830)

[Programowanie dynamiczne. 31](#_Toc341829831)

[28.Złożoność obliczeniowa algorytmu. Zdefiniuj pojęcie notacji asymptotycznej. Podaj klasy złożoności wraz z przykładami algorytmów. 33](#_Toc341829832)

[Złożoność obliczeniowa algorytmu. 33](#_Toc341829833)

[Notacja asymptotyczna. 33](#_Toc341829834)

[Klasy złożoności. 34](#_Toc341829835)

[29.[puste]Zdefiniuj pojęcie analizy wrażliwości. Przedstaw przykłady zastosowania. 34](#_Toc341829836)

[30.[puste]Zagadnienie uwarunkowania problemów odwrotnych. Metody regularyzacji. 34](#_Toc341829837)

# 1.Omów wpływ architektury procesora i układu pamięć-procesor na wydajność obliczeń sekwencyjnych

Na ogólną wydajność procesora wpływają następujące czynniki:  
  
**Wewnętrzna Architektura procesora** – są to szerokość magistrali, rejestrów i jednostek ALU oraz sposób współpracy procesora z pamięcią operacyjną i urządzeniami I/O. W procesorach wielordzeniowych ważny jest mechanizm zarządzania zadaniami rozdzielanymi między rdzenie.  
  
**Szybkość zegara pracy** – Im wyższa częstotliwość wyrażana w hercach Hz, megahercach MHz, gigahercach GHz, tym szybciej procesor może wykonywać operację  
  
**Wielkość pamięci Cache** – Najnowsze procesory mają po kilka MB pamięci podręcznej Cache. Podzielone są na trzy poziomy L1, L2, L3. Zwiększenie ilości pamięci Cache pozwala przyśpieszyć pracę procesora bez konieczności modyfikowania wewnętrznej struktury  
  
**Dodatkowe funkcje** – mają one za zadanie poprawiać możliwości CPU.

# 2.Podaj techniki optymalizacji obliczeń sekwencyjnych. Jakie są etapy przy tworzeniu kodu sekwencyjnego wysokiej wydajności?

Optymalizacji kodu dokonuje się zazwyczaj ze względu na jeden z dwóch czynników:

– rozmiar kodu

– szybkość działania kodu (wydajność)

Optymalizację przeprowadzić można:

– “ręcznie” stosując odpowiednie techniki

– wykorzystując opcje optymalizującego kompilatora

– przekazując, jeśli jest taka możliwość, wykonanie części kodu procedurom zoptymalizowanych bibliotek

• Opłacalność wyboru jednego z powyższych sposobów zmienia się w czasie i zależy od szeregu czynników, takich jak np.:

– istnienie i jakość zoptymalizowanych bibliotek

– wiek i typowość sprzętu, na którym dokonywane są obliczenia

– typowość optymalizowanego programu

• Efekt optymalizacji wydajności programów uzyskuje się najczęściej poprzez:

– redukcję liczby wykonywanych operacji

– optymalizację dostępu do pamięci

– umożliwienie sprawniejszego przetwarzania potokowego przez procesor

• Ten ostatni cel może być uzyskiwany np. w efekcie usunięcia pojawiających się w programach zależności danych

• Optymalizacja dotycząca zmiennych i wyrażeń:

– *constant folding* (zwijanie stałych)

– *copy propagation* (propagacja kopii)

– *strength reduction* (redukcja złożoności wyrażeń)

– *variable renaming* (przemianowanie zmiennych)

– *common subexpression elimination* (eliminacja powtarzających się podwyrażeń)

• Optymalizacja wykonania pętli:

– *induction variable simplification* (uproszczenie wyrażeń zawierających indeks pętli)

– *loop invariant code motion* (usunięcie poza pętle kodu niezależnego od iteracji)

– *loop interchange* (zamiana kolejności wykonywania pętli)

– *loop fusion* (łączenie pętli)

– *loop fission* (rozdzielanie pętli)

– *loop unrolling* (rozwijanie pętli)

– *blocking* (grupowanie instrukcji ze względu na dostęp do pamięci podręcznej)

• Optymalizacja na poziomie instrukcji:

– *dead code removal* (usuwanie nieosiągalnego lub produkującego zbędne dane kodu)

– *tailrecursion elimination* (eliminacja rekursji ogonowej)

– *inlining* (wplatanie procedur – rozwijanie w miejscu wywołania)

– *software prefetching* – pobieranie z wyprzedzeniem realizowane programowo

– *software pipelining* – przetwarzanie potokowe na poziomie kodu źródłowego

– i wiele innych

Kolejność postępowania przy optymalizacji:

– wybrać najlepszy algorytm (uwzględnić czy jest podatny na optymalizację wydajności, np. zrównoleglenie)

– zaimplementować (jeśli nie istnieją zoptymalizowane procedury biblioteczne)

– zbadać profil czasu wykonania programu (optymalizować ręcznie tylko fragmenty mające istotny wpływ na wydajność całości)

– eksperymentować z różnymi kombinacjami optymalizacji ręcznej i opcji kompilatora

– przeglądać generowane pliki w języku asemblera szukając możliwości przyspieszenia działania kodu

– ewentualnie modyfikować kod asemblera

– porównywać osiągniętą wydajność z maksymalną teoretyczną wydajnością wykorzystywanego sprzętu

Optymalizacja ręczna wspomagająca działanie kompilatora:

– usuwać przeszkody dla optymalizacji automatycznej:

• zależności danych (zwłaszcza w pętlach)

• złożone (np. pośrednie) adresowanie tablic w pętlach

• niestandardowe postacie instrukcji sterujących pętli

• częste (np. wewnątrz pętli) operacje złożone, jak:

– wywołania funkcji

– instrukcje warunkowe

– operacje wejścia wyjścia

– zbyt krótkie pętle (lepiej ręcznie rozwinąć)

Optymalizacja ręczna wspomagająca działanie kompilatora:

– przeprowadzać optymalizację, która może być zbyt skomplikowana dla kompilatora:

• reorganizowanie kodu (zmiana kolejności pętli, łączenie, dzielenie)

• optymalizacja dostępu do pamięci

– pamiętać, że ostateczny efekt optymalizacji składa się z:

• eliminacji zbędnych działań

• usprawnienia przetwarzania potokowego i ew. wektorowego

• optymalnego wykorzystania hierarchii pamięci

# 3.Z czego składa się czas wykonania obliczeń równoległych? Jakie czynniki wpływają na wydajność obliczeń równoległych? Jak można optymalizować tę wydajność?

Czas wykonania na pojedynczym i-tym procesorze

– Ti = Tiobl + Tikomunikacji + Tijałowy

• Całkowity czas wykonania równoległego

– T|| = maxi(Ti)

– T|| = Σi Ti /p (uzupełnienie czasem jałowym)

• Możliwe jest zmniejszenie czasu wykonania dla systemów, w których daje się osiągnąć nakładanie się obliczeń i komunikacji

Elementy składowe czasu wykonania:

– Ti obl – czas obliczeń na pojedynczym procesorze

– Najczęściej zakłada się, że algorytm równoległy wykonuje te same operacje co algorytm sekwencyjny, lub ewentualnie także pewne dodatkowe operacje wynikłe z realizacji równoległej

• Σi Ti obl = Tsekw + Tplus

– W teorii (np. do analiz wydajności) często zakłada się idealny rozkład operacji pomiędzy procesory

• Ti obl = (Tsekw + Tplus)/p

– W praktyce maksymalny czas obliczeń na pojedynczym procesorze jest zazwyczaj większy niż wynikający z założenia idealnego zrównoważenia obciążenia (chyba że ma miejsce przyspieszenie ponadliniowe)

# 4.Omów modele cyklu życia oprogramowania.

## 4.1 Model kaskadowy

Model, w którym każda kolejna faza następuje dopiero po zakończeniu fazy poprzedniej. W klasycznym modelu kaskadowym (*waterfall model* ) nie ma powrotu do poprzedniej fazy.

Fazy:

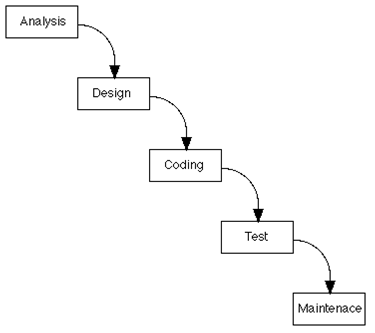
* Określenie wymagań (*requirements*)
* Projektowanie systemu (*system design*)
* Implementacja i testowanie modułów (podsystemów)
* Testowanie połączeń modułów i całości systemu
* Użytkowanie i pielęgnacja (konserwacja, *maintenance)*

**Zalety**: zidentyfikowanie podstawowych faz i uporządkowanie procesu tworzenia oprogramowania

**Wady**: rygorystyczne określenie następstwa faz, co może utrudniać realizacje projektu

**Warianty**:

* Realizacja kierowana dokumentami: przejście do kolejnej fazy następuje po zatwierdzeniu zbioru dokumentów
* Model spiralny (pkt. 4.2)
* Model ewolucyjny (pkt. 4.3)



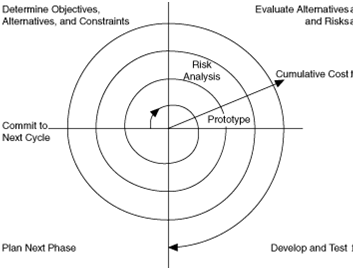
Rysunek 2: Model kaskadowy

## 4.2 Model spiralny

Model zaproponowany jako ogólny schemat rozwoju systemów i oprogramowania. Wyróżnia sie cztery podstawowe etapy: określanie celu, analiza alternatyw i ryzyka, realizacja i walidacja, planowanie następnego etapu.

**Wady**: słabo ujmuje specyfikę wytwarzania oprogramowania,

**Zalety**: jawne uwzględnienie alternatyw i ryzyka



Rysunek 1: Model spiralny

## 4.3 Model ewolucyjny

Model, którego celem jest poprawienie modelu kaskadowego przez rezygnację ze ścisłego, liniowego następstwa faz. Pozostawia sie te same czynności, z możliwością powrotu do poprzednich faz. Tym samym umożliwia się adaptację zmian i korekcję błędów. Model wymaga dodatkowych strategii dla uporządkowania procesu

## 4.4 Prototypowanie

* Prototypowanie jest techniką w ramach wytwarzania ewolucyjnego, w której pojawia się nowa faza tworzenia oprogramowania, poza wymienionymi dotychczas – faza tworzenia prototypu
* Prototyp jest niepełnym systemem, spełniającym cześć wymagań, przeznaczonym do przetestowania rozwiązań wykorzystanych do jego wytworzenia
* Z założenia prototyp nie wchodzi w skład ostatecznego systemu (ostateczny system budowany jest od podstaw po zaakceptowaniu rozwiązań zastosowanych w prototypie)

## 4.5 Wytwarzanie odkrywcze

Wytwarzanie odkrywcze (*exploratory development)* jest wariantem modelu ewolucyjnego, w którym iteracje dotyczą całego cyklu wytwarzania oprogramowania

* Istotą wytwarzania odkrywczego jest stała współpraca z klientem, który otrzymuje kolejne, coraz bogatsze wersje systemu i na ich podstawie określa i uszczegóławia swoje wymagania
* Wytwarzanie odkrywcze dobrze radzi sobie z występującym powszechnie faktem zmiany wymagań przez klientów
* Wytwarzanie odkrywcze polega na stałej modyfikacji kodu, bez odrzucania dotychczas wytworzonego oprogramowania (przeciwnie do prototypowania)

## 4.6 Wytwarzanie przyrostowe

* W wytwarzaniu przyrostowym najpierw następuje określenie wymagań, po czym całość systemu dzielona jest na kolejne „przyrosty” (*increments*), każdorazowo dające się testować, rozrastające się wersje systemu (pierwsze wersje zazwyczaj ujmują podstawowe funkcjonalności systemu)
* Problemem podstawowym wytwarzania przyrostowego jest określenie „przyrostów”, tak aby były one istotnymi fragmentami oprogramowania, a mimo to każdą z wersji dawało się niezależnie testować i oceniać

## 4.7 Model komponentowy

* W modelu komponentowym idee ponownego użycia kodu posunięte są najdalej
* Po fazie określania wymagań następuje faza analizy możliwości wykorzystania istniejących, gotowych komponentów i ewentualna faza modyfikacji wymagań, w konsekwencji zastosowania komponentów
* W fazie projektowania uwzględnia się już znalezione komponenty oraz ewentualnie nowe, związane z techniczną realizacją (implementacją)
* Projekt oprogramowania wykonywany jest tak, aby te spośród wytwarzanych elementów, które się do tego nadają, mogły być ponownie wykorzystane jako komponenty
* W fazie wytwarzania kodu zwraca się szczególną uwagę na interfejsy pomiędzy modułam i komponentami
* Testowanie jest w dużej mierze testowaniem integracji poszczególnych komponentów
* Mimo zalet związanych z wykorzystaniem gotowych, przetestowanych modułów, wytwarzanie oprogramowania w oparciu o komponenty (*component based software development*) stwarza specyficzne trudności:
  + wymagania narzucane przez gotowe komponenty mogą być niezgodne z wymaganiami klientów
  + modyfikacje kodu mogą być utrudnione przez brak kontroli nad pochodzącymi z zewnątrz komponentami

# 5.Przedstaw wybraną metodologię tworzenia oprogramowania (np. RUP, XP).

## 5.1 RUP (Rational Unified Process)

**Zasady RUP:**

* iteracyjne i przyrostowe tworzenie oprogramowania; sterowane ryzykiem i priorytetami, ułatwiające integrację całości kodu i dostosowanie do zmieniających się wymagań
* zarządzanie wymaganiami; we współpracy z klientem i w oparciu o przypadki użycia
* stosowanie architektury opartej na komponentach
* graficzne modelowanie oprogramowania; różne perspektywy spojrzenia na system, użycie UML
* kontrola i weryfikacja jakości oprogramowania przez cały czas procesu wytwarzania
* zarządzanie zmianami w oprogramowaniu

**Fazy projektu RUP**

* faza początkowa (*inception*) – wstępne określenie wymagań, ryzyka, kosztu, harmonogramu, a także architektury systemu
* faza opracowania (*elaboration*) – ustalenie wymagań (większości przypadków użycia), architektury systemu oraz planu całego procesu wytwarzania systemu
* faza konstrukcji (*construction*) – tworzenie systemu (kolejnych komponentów), w trakcie następuje oddanie pierwszej (i być może dalszych) wersji użytkownikowi
* faza przekazania (*transition*) – system jest przekazywany użytkownikowi, wdrażany, szkoleni są pracownicy obsługi systemu, następuje walidacja i końcowe sprawdzenie jakości

**RUP wyróżnia także „dyscypliny”, grupy zadań**

wykonywanych przez pracowników:

* modelowanie biznesowe
* wymagania
* analiza i projektowanie
* implementacja
* testowanie
* wdrożenie
* zarządzanie konfiguracją i zmianami
* zarządzanie projektem (zarządzanie ryzykiem, planowanie iteracji, monitorowanie postępów)
* organizacja środowiska (m.in. narzędzi)

RUP jest dobrze opisane: <http://pl.wikipedia.org/wiki/Rational_Unified_Process>.

## 5.2 Programowanie ekstremalne

**Podstawowe zasady XP:**

* oprogramowanie jest rozwijane w krótkich cyklach i w ciągłej interakcji z klientem, po każdym cyklu może nastąpić zmiana założeń co do dalszej pracy, co więcej może nastąpić rewizja już napisanego kodu (*refactoring*)
* każdy przyrost dostarcza konkretną funkcjonalność określaną na podstawie scenariuszy ("opowieści użytkownika", *user stories*)
* już pierwsze wydanie zawiera system o pewnej całościowej strukturze realizujący istotne dla klienta funkcje
* kolejne wydania (*releases*) kodu następują co kilka miesięcy (maximum), iteracje mają po kilka tygodni, obowiązuje planowanie zadań kilka dni do przodu, przy założeniu, że kolejność działań jest określana przez priorytety ważności
* przed przystąpieniem do kodowania opracowywane są szczegółowe testy mające za zadanie sprawdzenie wszelkich aspektów poprawności wprowadzanych zmian
* każda zmiana jest od razu integrowana z całością kodu oraz testowana – nowe oraz opracowane wcześniej testy (zautomatyzowane) są powtarzane codziennie (lub nawet kilka razy dziennie), żeby sprawdzić czy nie zostały wprowadzone błędy
* w ciągu całego procesu gromadzona jest informacja zwrotna służąca usprawnieniu procesu i ostatecznego kodu
* ważna jest specyficzna organizacja miejsca pracy
* kodowanie realizowane jest zgodnie z przyjętymi przez cały zespół regułami (*coding standards*)
* programowanie odbywa się parami (zgodnie ze szczegółowo ustalonymi zasadami), częste są także dyskusje pomiędzy członkami całego zespołu tworzącego kod
* dokumentacja oprogramowania składa się z:
  + komentarzy w kodzie
  + opisu testów
  + niewiele więcej

**Projektowanie kodu w ramach XP:**

* pojęcie „metafory” zamiast architektury kodu – próba syntetycznego, przenośnego wyrażenia istoty logicznej struktury kodu (np. metafora koszyka na zakupy w programach *ecommerce*)
* szczegółowe planowanie kolejnych działań w ramach iteracji – „gra w planowanie”, *planning game*
* nacisk na prostotę struktury kodu w każdym momencie (realizacja całej funkcjonalności kodu przy użyciu jak najmniejszej liczby klas i metod, bez duplikacji – zasada *once and only once*)
* ciągła refaktoryzacja dla poprawy struktury kodu (*designing through refactoring*)
* karty CRC używane do projektowania klas

**Gra w planowanie:**

* we wzajemnej interakcji pomiędzy przedstawicielami klienta i programistami powstaje schemat kolejnych działań
* klient decyduje o:
  + zakresie funkcjonalności (co system ma robić, *user stories*)
  + priorytetach funkcji (do sterowania wydaniami)
  + składzie konkretnych wydań
  + datach wydań
* wytwórca (programista) decyduje o:
  + oszacowaniu koniecznych zasobów do realizacji
  + technicznych konsekwencjach podjętych decyzji
  + organizacji procesu wytwarzania oprogramowania
  + szczegółowym harmonogramie realizacji wydania

**Cykl programowania w XP:**

* początek: napisz test
* skompiluj test (powinno się nie udać – nie ma jeszcze kodu)
* zaimplementuj tyle, aby udało się skompilować
* wykonaj test (powinien się nie udać – kod nie jest kompletny)
* zaimplementuj tyle, aby przejść pomyślnie test
* wykonaj test
* dokonaj refaktoryzacji, dla uproszczenia, usunięcia powtarzalności
* przeprowadź wszystkie istniejące testy
* przejdź do następnego zadania i zacznij od początku

**Testy w XP:**

* zawsze izolowane (sprawdzające jeden aspekt, fragment kodu)
* zautomatyzowane – możliwe do włączenia w zestaw zadań sprawdzających kod, uruchamianych po każdej modyfikacji (do kilku razy dziennie)
* dzielące się na:
  + funkcjonalne – inspirowane przez klienta i transformowane w testy jednostkowe przez specjalnie oddelegowanego członka zespołu
  + jednostkowe – tworzone przez programistów do testowania poszczególnych metod
  + inne: symulujące losowe dane wejściowe, symulujące najgorsze przypadki, porównujące różne wersje systemu

**Uczestnicy procesu XP:**

* klient
* programiści
* tester – współpracuje z klientem przy pisaniu scenariuszy i uruchamia testy sprawdzające realizacje scenariuszy
* śledzący (*tracker*) – sprawdza postęp prac oraz weryfikuje wszelkie oszacowania
* trener (*coach*) – nadzoruje proces tworzenia oprogramowania przez zespół, pomaga, kontroluje, itp.
* konsultant (okazjonalnie)
* szef – podejmuje strategiczne decyzje, zdobywa zasoby, reprezentuje zespół na zewnątrz

**Do problemów, na które natyka się programowanie ekstremalne należą:**

* niedostosowanie do wymagań niektórych klientów (wymagających kontraktów, rozbudowanej specyfikacji, bogatej dokumentacji itp.)
* niedostosowanie do struktur organizacyjnych tradycyjnych (dużych) firm software'owych (mających np. biura rozproszone w różnych częściach świata)
* niedostosowanie do mentalności niektórych programistów
* trudność utrzymania prostej struktury kodu, w miarę jego wzrostu dokonywanego metodami ewolucyjnymi
* trudność zapewnienia wymagań bezpieczeństwa i niezawodności dla niektórych rodzajów systemów

Dobry opis: <http://pl.wikipedia.org/wiki/Programowanie_ekstremalne>.

## 5.3 Agile programming

**Metody zwinne:**

* Jednostki i interakcje ponad procesy i narzędzia
* Działające oprogramowanie ponad wyczerpującą dokumentację
* Współpraca z klientem ponad negocjacje kontraktu
* Reagowanie na zmiany ponad realizowanie planu

Opisane tutaj: <http://pl.wikipedia.org/wiki/Programowanie_zwinne>.

# 6.Scharakteryzuj następujące techniki ponownego wykorzystania kodu: komponenty, wzorce, refaktoryzacja, programowanie aspektowe, RAD.

*(Inżynieria oprogramowania wykład 10.)*

**Komponenty** są to elementy większe niż pojedyncze klasy, lecz mniejsze niż frameworki. Podstawową cechą jest zdolność do niezależnego montowania bez konieczności kompilacji. Komponent musi być w pełni określony przez interfejs, który realizuje.

Przykłady komponentów: CORBA, EJB, (D)COM(+). Rozwijaną ostatnio alternatywa dla komponentów są *webservices*. Elementy środowisk komponentowych: serwer usług, aplikacja kliencka, middleware. W przypadku webservices występuje jeszcze repozytorium usług.

**Wzorce projektowe** z założenia przedstawiają szkic rozwiązania, który należy uszczegółowić i dostosować do konkretnego kontekstu. Są realizacją koncepcji ponownego użycia, ale w odniesieniu do idei, a nie konkretnego kodu. Rozwiązują konkretne problemy często pojawiające sie w praktyce. Standard prezentacji wzorców:

• nazwa

• przeznaczenie – zwięzła chrakterystyka motywacji i celu

• inne nazwy

• uzasadnienie stosowania – ogólny opis problemu i jego rozwiązania przy pomocy wzorca

• stosowalność – różne konteksty użycia

• struktura – najczęściej w postaci diagramu UML

• uczestnicy – szczegółowy opis klas i obiektów

• współpraca

• konsekwencje – aspekty zastosowania wzorca

• implementacja

• przykłady

• wzorce pokrewne

**Refaktoryzacja** jest to poprawa projektu klas i ich struktury bez zmiany funkcjonalności. Podstawowe kierunki refaktoryzacji:

• uproszczenie struktury

• zwiększenie elastyczności

• zmniejszenie dużych klas

Przeczytaj też: <http://pl.wikipedia.org/wiki/Refaktoryzacja>.

**Programowanie aspektowe** zakłada, ze w programach pojawiają się elementy dwóch typów:

• elementy dające się hermetyzować w jednostki o ściśle określonej funkcjonalności, dobrze oddzielone od innych

• elementy typu „cross-cut”, które przecinają jednostki hermetyzacji i określają pewien aspekt

grupy jednostek np. organizacja dostępu, komunikacja, synchronizacja, obsługa błędów itp.

Przeczytaj też: <http://pl.wikipedia.org/wiki/Programowanie_aspektowe>.

**RAD** (Rapid Application Development) jest to technika szybkiego konstruowania programów, w których istotna role odgrywa UI, natomiast mniejszą przetwarzanie danych. Programy mają zbliżoną strukturę, wiele czynności daje się zautomatyzować. Typowe jest korzystanie z narzędzi CASE oraz posługiwanie się programowaniem graficznym (*visual programming*). RAD zakłada tzw. *time boxing* – tworzenie oprogramowania w przedziałach czasowych o ściśle określonej długości. Oprogramowanie tworzone jest przez mały zespół, który wytwarza oprogramowanie jako serie prototypów. Tworzenie opiera sie głównie na predefiniowanych elementach, API, bibliotekach, komponentach etc.

# 7.Czym jest język UML? Omów wybrany rodzaj diagramów UML.

Czym jest UML

* Językiem pozwalającym tworzyć modele systemów (np. informatycznych)
* Pozwala obrazować, specyfikować, tworzyć i dokumentować elementów systemu
* Ułatwia wymianę informacji pomiędzy przyszłymi użytkownikami systemu, menadżerami, analitykami, projektantami, programistami i testerami
* Ułatwia wykorzystanie zalet programowania obiektowego
* UML jest jedynie językiem modelowania używanym w procesie analizy i projektowania systemów komputerowych

Opis wszystkich diagramów: [http://zasoby.open.agh.edu.pl/~09sbfraczek/#](http://zasoby.open.agh.edu.pl/~09sbfraczek/%23).

# 8.Scharakteryzuj model przetwarzania równoległego z pamięcią wspólną i środowisko programowania OpenMP

## Model przetwarzania równoległego z pamięcią wspólną (Parallel Random Access Machine)

Model PRAM zapewnia możliwość jednoczesnego dostępu każdego spośród n procesorów o architekturze RAM do wspólnej pamięci (pamięci współdzielonej). W modelu tym komunikacja między procesami i ich synchronizacja odbywa się z użyciem wspólnych zmiennych. W rezultacie komputery równoległe o wspólnej przestrzeni adresowej są stosunkowo łatwe do programowania, gdyż minimalizują one problemy związane z przydziałem danych do poszczególnych procesorów i dynamicznym równoważeniem ich obciążenia.

Powyższy model pociąga jednak za sobą możliwość wystąpienie konfliktów w przypadku, gdy kilka procesorów jednocześnie odwołuje się do tej samej komórki pamięci. Podczas gdy możliwość jednoczesnego wykonania operacji odczytu wydaje się koncepcyjnie spójną (nawet, jeśli jej fizyczna realizacja jest nieoczywista), to nie można tego powiedzieć o operacji zapisu. Dlatego też ocena stopnia równoległości realizacji zapisu do pamięci i/lub odczytu z pamięci pozwala wyróżnić w ramach modelu PRAM kilka klas systemów różniących się sposobem organizacji i zarządzania pamięcią. Są to:

* model EREW (ang. exclusive-read, exclusive write), w którym tylko jeden procesor posiada prawo do odczytu lub zapisywania informacji do określonej komórki pamięci;
* model CREW (ang. concurrent read, exclusive write), w którym możliwe jest jednoczesne wykonywanie operacji odczytu z pamięci przez wiele procesorów przy ograniczeniu możliwości jednoczesnego zapisu do pojedynczego procesora;
* model ERCW, w którym więcej niż jeden procesor posiada prawo do jednoczesnego wykonywanie operacji zapisu przy ograniczeniu możliwości jednoczesnego odczytu do pojedynczego procesora;
* model CRCW, w którym zarówno dla operacji zapisu, jak i odczytu możliwe jest ich jednoczesne wykonywanie przez więcej niż jeden procesor.

Do badania algorytmów równoległych i organizacji obliczeń w komputerach równoległych najczęściej stosowane są modele EREW i CRCW, podczas gdy dla celów realizacji technicznej najszerzej wykorzystuje się model EREW PRAM, co wynika z możliwości jego realizacji w postaci modelu RAM.

W przypadku maszyn, które umożliwiają równoległą (dodajmy konkurencyjną) realizację operacji zapisu, kilka procesorów może jednocześnie ubiegać się o prawo zapisu pewnego zbioru danych do pamięci. Możliwe sposoby realizacji powyższej równoległości to:

* deterministyczny, gdy preferencję przy zapisie danych oddaje się procesorowi z najwyższym priorytetem;
* grupowy (inaczej kolektywny), gdy dane niektórych procesorów grupuje się (wykorzystując np. prawo łączności operacji dodawania, mnożenia, znajdowania minimum itd.), co pozwala na redukcję liczby odwołań do pamięci dzięki przechowywaniu rezultatów pośrednich w procesorach, podczas gdy tylko rezultaty operacji grupowych przesyłane są do pamięci;
* niedeterministyczny (losowy), gdy dowolnie wybrany procesor ma możliwość dokonania natychmiastowego zapisu do pamięci, a wszystkie pozostałe próby kończą się niepowodzeniem.

## OpenMP (Open Multi-Processing)

**Cechy OpenMP:**

* Przenośność oprogramowania
* Model SPMD
* Szczegółowe wersje (*bindings*) dla różnych języków programowania
* Elementy składowe:
* dyrektywy dla kompilatorów
* funkcje biblioteczne
* zmienne środowiskowe

**Składnia dyrektyw:**

* format ( dla powiązania z językami C i C++ ):

#pragma omp *nazwa\_dyrektywy lista\_klauzul znak\_nowej\_linii*

* najważniejszymi z dyrektyw są dyrektywy podziału pracu (*work sharing constructs*), występujące w obszarze równoległym i stosowane do rozdzielenia poleceń realizowanych przez poszczególne procesory
* najważniejsze klauzule określają sposób traktowania zmiennych przez wątki w obszarze równoległym
* każda dyrektywa posiada swój własny zestaw dopuszczalnych klauzul

**Dyrektywy podziału pracy:**

* wszystkie wątki w zespole muszą realizować te same dyrektywy podziału pracy (i bariery)
* nie ma niejawnej bariery przy wejściu
* rozpoczynanie i kończenie wykonania wybranych dyrektyw może być związane z synchronizacją (realizacją bariery)
* wątki dzielą się pracą – każdy realizuje przydzielone sobie operacje na przydzielonej sobie części danych

**Podstawowe funkcje OpenMP**

* **Zamki**

Zamki są to części kodu, które mogą być wykonywane tylko przez jeden wątek na raz. Gdy jeden wątek wkracza w blok kodu zamykając zamek, każdy inny chcący wejść do tego bloku, musi poczekać, na otwarcie zamka.

**omp\_init\_lock**(omp\_lock\_t **lock**) – zainicjowanie zamka **lock omp\_destroy\_lock**(omp\_lock\_t **lock**) – zniszczenie zamka **lock**

**omp\_set\_lock**(omp\_lock\_t **lock**) – zamknięcie zamka **lock**

**omp\_set\_lock**(omp\_lock\_t **lock**) – sprawdzenie zamka **lock**, gdy zamknięty zwraca 0, gdy otwarty, zamyka go i zwraca 1

**omp\_unset\_lock**(omp\_lock\_t **lock**) – otwarcie zamka **lock**

* **Sekcja krytyczna**

Sekcja krytyczna jest tym samym, co zamek, ale jest zaimplementowana jako dyrektywa preprocesora. Najwyżej jeden wątek może w danej chwili wykonywać kod w sekcji krytycznej, pozostałe czekają aż on skończy, aby móc wejść do sekcji.

**#pragma omp critical (nazwa\_sekcji)** – następny blok kodu będzie sekcją krytyczną o nazwie **nazwa\_sekcji**

* **Równoległa pętla for**

Równoległa pętla for jest jedną z najczęściej wykorzystywanych metod wieloprocesowości w OpenMP. Dlatego ma tak wiele opcji.

**#pragma omp parallel for** [**opcje**] – po tej linii następuje pętla for ze składnią zgodną z językami C/C++, której iteracje zostaną podzielone na wątki, w zależności od wybranych opcji:

* + **schedule** – sposób dzielenia iteracji pomiędzy wątki:
    - **static** – ilość iteracji podzielona na równe części i każda z tych części przydzielona jednemu wątkowi
    - **dynamic, ilość** – każdy wątek dostaje kolejne **ilość** iteracji, gdy je wykona, dostaje następną część iteracji o wielkość **ilość**, jeśli **ilość** nie jest zdefiniowana, to jej domyślna wartość jest równa 1
  + **private(zmienna1, zmienna2, ...)** – zmienne wyliczone będą miały wykonane kopie dla każdego wątku, wątki będą korzystały z tej samej nazwy, jednak każdy z nich będzie działał na swojej kopii zmiennej (iterator pętli jest domyślnie private)
  + **firstprivate(zmienna1, zmienna2, ...)** – każdy wątek dostanie kopie wyliczonych zmiennych, po zakończeniu bloku równoległego zmiennym tym nadane zostaną wartości jak sprzed tego bloku
  + **lastprivate(zmeinna1, zmienna2, ...)** - każdy wątek dostanie kopie wyliczonych zmiennych, po zakończeniu bloku równoległego zmiennym zostaną nadane wartości z ostatniej wykonanej iteracji pętli
  + **shared(zmienna1, zmienna2, ...)** – zmienne wyliczone będą wspólne dla wszystkich wątków, należy przy tym uważać aby nie dopuścić do przekłamań w zapisywaniu/odczytywaniu danych do tych zmiennych
  + **default(shared|none)** – ustawia domyślny typ na shared lub żaden
  + **reduction(operator:zmienna)** – każdy wątek dostanie swoją część zmiennej **zmienna**, a na końcu pętli wszystkie kopie zostaną złączone operacją operator
  + **nowait** – po zakończeniu pętli wątki nie będą czekały na te, które jeszcze nie skończyły swojej pracy (brak domyślnej bariery na końcu pętli)
  + **num\_threads(ilość)** – ilość wątków użytych w pętli (metodę można użyć do dowolnego bloku równoległego, nie tylko do pętli)
* **Sekcje**

W bloku kodu będą wyszczególnione sekcje, z których każda zostanie wykonana tylko przez jeden wątek (tylko raz). Wszystkie będą wykonywane równocześnie, chyba że nie będzie wystarczającej ilości wątków, wtedy wątki które zakończyły swoje sekcje, będą rozpoczynały kolejne, które jeszcze nie zostały wykonane.

Możliwe opcje: **private, firstprivate, lastprivate, reduction, nowait**.

**#pragma omp parallel sections** [**opcje**]

{

**#pragma omp section**

{/\*blok kodu\*/}

**#pragma omp section**

{/\*blok kodu\*/}

}

* **Bariera**

Bariera jest miejscem synchronizującym wykonanie kodu przez wątki. Każdy wątek, który dotrze do bariery, musi poczekać na inne. Dopiero kiedy wszystkie wątki dojdą do tego miejsca, mogą kontynuować wykonywanie kodu.

**#pragma omp barier**

* **Pozostałe ważniejsze funkcje Openmp:**
  + int **omp\_get\_num\_threads**() - zwraca ilość wątków
  + void **omp\_set\_num\_threads**(int **ilość**) – następny blok równoległy będzie używał **ilość** wątków
  + int **omp\_get\_thread\_num**() - zwraca numer bieżącego wątku

# 9.Jak mierzymy i wyrażamy wydajność obliczeń równoległych? Czym jest skalowalność i jak ją wyrażamy?

Miary wydajności obliczeń równoległych:

* przyspieszenie obliczeń: *S(p) = Ts / T||(p)*
  + *Ts* – czas rozwiązania zadania najlepszym algorytmem sekwencyjnym na pojedynczym procesorze
  + *T||(p)* – czas rozwiązania zadania rozważanym algorytmem równoległym na *p* procesorach (w praktyce często zamiast *Ts* używa się *T||(1)* )
* efektywność zrównoleglenia: *E(p) = S(p) / p*

ideałem jest uzyskanie liniowego przyspieszenia i 100% *e*fektywności (jak musi się zachowywać narzut obliczeń, T||narz(p) = T||(p) – T||(1)/p , aby uzyskać przyspieszenie liniowe?)

Skalowalność - dobre zachowanie systemu (sprzętu, środowiska wykonania, programu) w sytuacji rosnącego obciążenia i, najczęściej, rosnących zasobów. Mówiąc o skalowalności często rozważa się wydajność jako funkcję rosnącego obciążenia i, ewentualnie, rosnących zasobów.

Przyspieszenie skalowane to przyspieszenie jakie uzyskuje się rozważając dla danej liczby procesorów *p* zadanie o rozmiarze *p-*krotnie większym od zadania rozwiązywanego na pojedynczym procesorze

* jako funkcja liczby procesorów przyspieszenie skalowane jest równe:

SS(p) = T||(1, pW0) / T||(p, pW0) = p\*T||(1, W0) / T||(p, pW0)

# 10.Wyjaśnij ideę gradientowych metod optymalizacji.

Gradientowe metody optymalizacji do znalezienia rozwiązania wykorzystują nie tylko znajomość funkcji celu, ale także ich gradientu (wektor składający się z pochodnym funkcji celu względem każdej ze zmiennych) lub wielkości z nim związanych. Z tego powodu funkcja celu musi być określona i różniczkowalna w całej przestrzeni.

W procesie poszukiwania minimum, metody optymalizacji gradientowej analizują nie tylko wartości, ale również trendy wzrostu funkcji celu (gradient wskazuje kierunek największego wzrostu funkcji celu). Wektor przeciwny do gradient wskazuje kierunek, w którym funkcja celu się zmniejsza i nazywany jest kierunkiem największego spadku. Jest on wykorzystywany przy poszukiwaniu minimum (maksimum) funkcji celu we wszystkich metodach gradientowych.

Wspólne cechy metod gradientowych:

* Ogólna strategia:
  + Określenie kierunku poszukiwania minimum, przy użyciu gradientu funkcji celu
  + Dobór kroku o odpowiedniej długości, który należy wykonać zgodnie ze znalezionym uprzednio kierunkiem szukania (metody stałokrokowe, metody zmiennokrokowe – krok dobierany tak, aby minimalizował funkcję celu)
* Poszukiwania są kończone do momentu spełnienia warunku stopu, którym może być np. odpowiednio mała długość wektora gradientu
* Musi być znany gradient funkcji celu

# 11.Wyjaśnij różnice pomiędzy deterministycznymi i niedeterministycznymi metodami optymalizacji.

Metody deterministyczne (np. gradientowe, simpleks, liniowe) nie radzą sobie dobrze w przypadku funkcji, które posiadają wiele minimów lokalnych – „grzęzną” one wtedy w którymś minimum lokalnym. Dlatego poszukuje się metod, których działanie nie ogranicza się do lokalnej analizy kształtu funkcji celu. Dzięki temu zwiększa się szanse lokalizacji minimów globalnych. Do tej grupy zaliczyć można metody niedeterministyczne, które wykorzystują w swoim założeniu czynnik losowy.

Metody niedeterministyczne:

* Monte Carlo (czynnik losowy powtarzany w n losowaniach)
* algorytmu genetyczne (kolejne losowe punkty są losową modyfikacją punktów z poprzedniej iteracji
* algorytmy ewolucyjne (podobne do genetycznych, ale: geny są liczbami rzeczywistymi a nie 0 lub 1, wymuszone odrzucenie najgorszych osobników)

# 12.Omów ideę algorytmów genetycznych.

Rozwiązanie każdego problemu może zostać przedstawione za pomocą pewnego zbioru *n* parametrów: *R = {param1, param2, ..., paramn*.

Pewne rozwiązania są lepsze od innych. Zatem każde rozwiązanie ma jakąś wartość, którą będziemy starać się maksymalizować (jeżeli wartość ta oznacza zysk) lub minimalizować (jeżeli wartość ta oznacza koszt).

Niech będzie dana początkowa pula *p* rozwiązań: *R11, R21, ..., Rp1*.

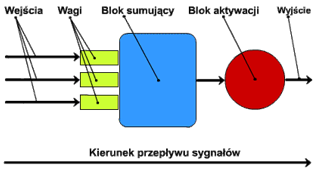
Może być ona wynikiem losowania (czyli każde rozwiązanie ma na początku losowe wartości swoich parametrów), bądź wynikiem jakiś różnych przybliżeń dla których spodziewamy się uzyskać zadowalające nas wyniki.

Każda kolejna pula rozwiązań (pokolenie) będzie powstawać poprzez wykonanie następujących kroków:

* oceń każde rozwiązanie w puli (pokoleniu)
* wybierz najlepsze rozwiązania (osobniki) i na ich podstawie wygeneruj nową pulę (pokolenie)
* wprowadź mutacje

Tworzenie nowych pokoleń następuje tak długo, aż najlepszy osobnik w danym pokoleniu będzie wystarczająco dobry, bądź zadaną liczbę razy.

# 13.Wyjaśnij zasadę budowy i działania sztucznego neuronu.



Sztuczny neuron skonstruowany został na wzór neuronu naturalnego.

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

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

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

|  |  |
| --- | --- |
|  | · skoku jednostkowego - tzw. funkcja progowa |
|  | · liniowa |
|  | · nieliniowa |

# 14.Wyjaśnij ideę uczenia sieci neuronowej metodą wstecznej propagacji błędu.

Uczenie sieci metodą wstecznej propagacji błędu umożliwia na minimalizację błędów popełnianych przez poszczególne neurony w sieci podczas nadzorowanego nauczania sieci. Polega ona na przenoszeniu błędu jaki popełniła cała sieć w kierunku od warstwy wyjściowej, do warstwy wejściowej.

Cykl uczenia tą metodą:

* wyznaczenie odpowiedzi neuronów warstwy wyjściowej oraz warstw ukrytych na zadany sygnał wejściowy
* Wyznaczenie błędu popełnionego przez neurony warstwy wyjściowej i przesłanie go w kierunku warstwy wejściowej
* Adaptacja wag

Korekta wektora wag sieci opiera się na minimalizacji funkcji miary błędu, którą określono jako sumę kwadratów błędów. Do obliczenia delty konkretnej wagi można wykorzystać regułę najszybszego spadku (pierwsza pochodna błędu średniokwadratowego po tej wadze). Następnie konkretna waga ulega korekcie poprzez tą deltę.

Więcej: <http://www.neurosoft.edu.pl/jbartman/NTI%20cwiczenie_5.pdf>.

# 15.Omów główne elementy systemu ekspertowego.

System ekspertowy składa się z następujących, niezależnych fizycznie, lecz współpracujących ze sobą, elementów:

* + - **baza wiedzy** – znajdują się w niej wszelkie informacje z zakresu wybranej dziedziny: wiedza faktograficzna (fakty), wiedza o wnioskowaniu (zbiór reguł), wiedza o sposobach rozwiązywania problemu (meta-wiedza) – musi być ona zapisana w postaci sformalizowanej, zrozumiałej dla mechanizmu wnioskującego i pozwalającej na prześledzenie sposobu dojścia systemu do rozwiązania;
    - **maszyna wnioskująca** – na podstawie zgromadzonej wiedzy wyszukuje rozwiązanie postawionego problemu – jest ona oddzielona od bazy wiedzy, dzięki czemu działa tak samo w systemach ekspertowych dla dowolnej dziedziny jak i w szkieletowych systemach ekspertowych; algorytm wyszukiwania zawiera szereg strategii przeszukiwań, heurystyk i metod wnioskowania – strategie wyznaczają kolejne kroki przeszukiwań, heurystyki pomagają zoptymalizować przestrzeń poszukiwań, a metody decydują w jaki sposób zachodzi proces myślenia (wnioskowane wstecz, w przód, czy inne);
    - **procedury objaśniania** – objaśniają strategię wnioskowania, sposób dojścia do rozwiązania i pokazują dokładniejsze dane o rozwiązaniu;
    - **interfejs** do porozumiewania się z użytkownikiem – umożliwia komunikację człowieka z systemem (pracę interaktywną) – służy do zasięgania informacji u użytkowania, przedstawiania wygenerowanego wyniku oraz udzielania wyjaśnień na temat procesu wnioskowania – konstrukcja i wygląd interfejsu zależy od języka programowania, za pomocą którego stworzono system ekspertowy oraz systemu operacyjnego, w którym on działa;
    - **moduły zdobywania i modyfikacji wiedzy** – pozyskiwanie wiedzy pozwala na automatyczne ulepszanie systemu;

Więcej: <http://aragorn.pb.bialystok.pl/~radev/ai/se/zal/ready/yes1.htm>.

# 16.Opisz algorytmy rozwiązania zagadnienia programowania liniowego.

Programowanie liniowe jest to model, w którym zarówno warunki ograniczające jak i funkcja celu maja postać liniową.

## Metoda geometryczna

Metoda geometryczna, zwana również graficzną, polega na znalezieniu rozwiązania zagadnienia programowania liniowego wśród wierzchołków wieloboku powstałego przez ograniczenia i warunki brzegowe.   
Metoda geometryczna jest stosowana głownie przy rozwiązywaniu zadań z małą ilością zmiennych decyzyjnych. Najlepsze rozwiązania otrzymujemy przy rozwiązywaniu zadań, w których występują dwie zmienne decyzyjne.

## Metoda Simpleks

Algorytm simpleks jest uniwersalna metodą rozwiązywania programów liniowych.   
Istota tego algorytmu polega na badaniu kolejnych rozwiązań bazowych (rozwiązań dopuszczalnych) programu liniowego w postaci kanonicznej w taki sposób, że :   
a) znajdujemy (dowolne) rozwiązanie bazowe programu;  
b) sprawdzamy, czy jest ono optymalne;  
c) jeżeli dane rozwiązanie nie jest optymalne, konstruujemy następne rozwiązania bazowe lepsze (lub przynajmniej nie gorsze od poprzedniego).

Postępowanie kończy się w momencie stwierdzenia, że aktualne rozwiązanie bazowe jest optymalne, tzn. nie można już go poprawić.  
Algorytm simpleks jest więc procedurą iteracyjna (etapową),a wyniki poszczególnych etapów zestawia się w kolejnych tablicach simpleks.

Ponieważ algorytm simpleks polega na badaniu rozwiązań bazowych postaci kanonicznej, przed przystąpieniem do budowy pierwszej tablicy simpleks należy zamienić wszystkie nierówności na równania, poprzez wprowadzenie pewnych nowych zmiennych. I tak dla nierówności typu "≤" do ich lewych stron dodajemy tzw. zmienne swobodne, które stanowią początkowe rozwiązania bazowe (w pierwszej tablicy simpleksowej). Natomiast w przypadku nierówności typu "≥" do ich lewych stron odejmujemy zmienne swobodne i dodajemy tzw. zmienne sztuczne. W tym przypadku zmienne sztuczne wchodzą do pierwszej bazy. Do funkcji celu, zmienne swobodne wchodzą ze współczynnikami równymi zero, natomiast zmienne sztuczne z tzw. współczynnikami M, gdzie M jest liczbą dużą (dążącą do nieskończoności).

Więcej: <http://www.ekonometria.4me.pl/programowanie.htm>.

# 17.Co rozumiesz pod pojęciem logistyka? Cele i zadania logistyki w przedsiębiorstwie.

**Logistyka** – termin opisujący proces planowania, realizowania i kontrolowania sprawnego i efektywnego ekonomicznie przepływu surowców, materiałów, wyrobów gotowych oraz odpowiedniej informacji z punktu pochodzenia do punktu konsumpcji w celu zaspokojenia wymagań klienta. Działania logistyczne mogą obejmować (choć nie muszą się do nich ograniczać): obsługę klienta, prognozowanie popytu, przepływ informacji, kontrolę zapasów, czynności manipulacyjne, realizowanie zamówień, czynności reparacyjne i zaopatrywanie w części, lokalizację zakładów produkcyjnych i składów, procesy zaopatrzeniowe, pakowanie, obsługę zwrotów, gospodarowanie odpadami, transport i składowanie.

Uproszczoną definicję przedstawia reguła "7R" w Polsce przetłumaczona na 7W:

1. right product (właściwy produkt),
2. right quantity (właściwa ilość),
3. right condition (właściwy stan),
4. right place (właściwe miejsce),
5. right time (właściwy czas),
6. right customer (właściwy klient),
7. right price (właściwa cena).

Mówi się również o formule 4C związanej z klientem (customer): jego potrzeby, koszty, wygoda zakupu, dostarczone informacje.

Zadania logistyki (są równoważne, wzajemnie powiązane i uwarunkowane)

1. Koordynacja przepływu surowców, materiałów, wyrobów gotowych do konsumentów

2. Minimalizacja kosztów tego przepływu

3. Podporządkowanie działalności logistycznej wymogom obsługi klienta

Cele logistyki – minimalizacja kosztów fizycznej dystrybucji dóbr, stosownie jednak do żądanego poziomu usług, co oznacza nadrzędność świadczenia usług nad maksymalizacją **zysków** i zwrotem nakładów kapitałowych. W odniesieniu do wszystkich przedsiębiorstw wyzwanie logistyki jest podobne i chodzi o to aby dać firmie metody i sposoby zdobycia rynku (satysfakcjonujących konsumentów) korzystne dla kosztów jak i dla dobra świadczonych usług.

# 18.Co to jest prognozowanie? Wymień i krótko opisz modele prognozowania ilościowego.

Prognozowanie to naukowa metoda przewidywania tego, w jaki sposób będą kształtowały się w przyszłości procesy lub zdarzenia. W trakcie procesu prognozowania formułuje się sąd na temat przyszłych stanów zjawisk i zdarzeń nazywany prognozą.

Na kształtowanie się procesów lub zjawisk mają wpływ różne czynniki, które można podzielić na:

* czynniki zewnętrzne (egzogeniczne), na które nie ma się wpływu, a które należy uwzględnić w trakcie prognozowania jako pewne zewnętrzne ograniczenia przebiegu zjawisk i procesów gospodarczych,
* czynniki wewnętrzne (endogeniczne), które mogą być kształtowane przez decydentów.

Prognozowanie wykorzystuje informację dotyczącą tych czynników i ich wpływu na badane zjawisko. W prognozowaniu bada się relacje między tymi czynnikami a badanym zjawiskiem oraz kształtowanie się ich w przeszłości do wnioskowania o przyszłości.

* Zmienna ciągła:
  + Modele trendu

<http://pl.wikipedia.org/wiki/Tendencja_rozwojowa>

<http://www.wkuwanko.pl/ekonomia/ekonomia-trend-liniowy-%5B5-stron%5D_22_1650.html>

* + Analiza harmoniczna

<https://sites.google.com/site/forexharmo/analiza-techniczna-1/teorie-handlowe/analiza-harmoniczna>

<http://www.polsl.pl/Wydzialy/RG/Wydawnictwa/Documents/kwartal/6_1_11.pdf>

* + Analiza regresji

<http://pl.wikipedia.org/wiki/Regresja_%28statystyka%29>

* + Metoda najmniejszych kwadratów

<http://pl.wikipedia.org/wiki/Metoda_najmniejszych_kwadrat%C3%B3w>

* Zmienna dyskretna
  + Analiza dyskryminacyjna

<http://pl.wikipedia.org/wiki/Analiza_dyskryminacyjna>

* + Modele probitowe

<http://en.wikipedia.org/wiki/Probit_model>

* + Modele logitowe

<http://en.wikipedia.org/wiki/Logistic_regression>

# 19.Co to jest relacyjna baza danych? Opisz rodzaje relacji oraz podaj przykład każdego z typów relacji. Wyjaśnij pojęcie "klucza podstawowego", jakie spełnia on funkcje, oraz podaj przykład "klucza prostego", "klucza wielokrotnego" i klucza obcego.

## Relacyjna baza danych

W najprostszym ujęciu w modelu relacyjnym dane grupowane są w relacje, które reprezentowane są przez tablice. Relacje są pewnym zbiorem rekordów o identycznej strukturze wewnętrznie powiązanych za pomocą związków zachodzących pomiędzy danymi. Relacje zgrupowane są w tzw. schematy bazy danych. Relacją może być tabela zawierająca dane teleadresowe pracowników, zaś schemat może zawierać wszystkie dane dotyczące firmy. Takie podejście w porównaniu do innych modeli danych ułatwia wprowadzanie zmian, zmniejsza możliwość pomyłek, ale dzieje się to kosztem wydajności.

## Typy relacji

* **jeden-do-jednego -** każdy rekord w tabeli A może mieć tylko jeden dopasowany rekord z tabeli B, i tak samo każdy rekord w tabeli B może mieć tylko jeden dopasowany rekord z tabeli A. Ten typ relacji spotyka się rzadko, ponieważ większość informacji powiązanych w ten sposób byłoby zawartych w jednej tabeli. Relacji jeden-do-jednego można używać do podziału tabeli z wieloma polami, do odizolowania części tabeli ze względów bezpieczeństwa, albo do przechowania informacji odnoszącej się tylko do podzbioru tabeli głównej.
* **jeden-do-wielu** - jest najbardziej powszechnym typem relacji. Rekord w tabeli A może mieć wiele dopasowanych do niego rekordów z tabeli B, ale rekord w tabeli B ma tylko jeden dopasowany rekord w tabeli A.
* **wiele-do-wielu** - rekord w tabeli A może mieć wiele dopasowanych do niego rekordów z tabeli B i tak samo rekord w tabeli B może mieć wiele dopasowanych do niego rekordów z tabeli A. Jest to możliwe tylko przez zdefiniowanie trzeciej tabeli (nazywanej tabelą łącza), której klucz podstawowy składa się z dwóch pól - kluczy obcych z tabel A i B. Relacja wiele-do-wielujest w istocie dwiema relacjami jeden-do-wielu z trzecią tabelą.

## Klucz podstawowy

Klucz podstawowy (*ang.* primary key) zwany też kluczem głównym to jedno lub więcej pól, których wartość jednoznacznie identyfikuje każdy rekord w tabeli. Taka cecha klucza nazywana jest unikatowością. Klucz podstawowy służy do powiązania rekordów w jednej tabeli z rekordami z innej tabeli. Klucz podstawowy jest nazywany kluczem obcym, jeśli odwołuje się do innej tabeli. Na przykład, w bazie pracowników kluczem podstawowym może być numer ewidencyjny pracownika.

Klucz podstawowy jednopolowy (*ang.* single primary key)

Jeśli istnieje pole zawierające dane unikatowe, jak na przykład numer katalogowy czy numer identyfikacyjny, można je zadeklarować jako klucz podstawowy. Jeśli jednak w polu tym powtarzają się wartości, klucz podstawowy nie zostanie ustawiony. Aby znaleźć rekordy zawierające te same dane, należy usunąć rekordy o powtarzających się wartościach bądź zdefiniować wielopolowy klucz. podstawowy.

Klucz podstawowy wielopolowy zwany też kluczem złożonym (*ang.*composed key)

W sytuacji, gdy żadne z pól nie gwarantuje unikatowości wartości w nim zawartych, należy rozważyć możliwość utworzenia klucza podstawowego złożonego z kilku pól.

Klucz obcy to kombinacja jednego lub więcej atrybutów tabeli, który wyraża się w dwóch lub większej liczbie relacji. Wykorzystuje się go do tworzenia relacji pomiędzy parą tabel.

Np. jeśli mamy bazę "Firma" składającą się z tabel:

* + Oddział (id\_oddziału, miejscowość, telefon, ...)
  + Pracownik (id\_pracownika, imię, nazwisko, id\_oddziału, ...)

to kolumna Pracownik.id\_oddziału mogłaby być kluczem obcym związanym z kolumną Oddział.id\_oddziału.

# 20.Co to jest zapytanie w bazie danych, opisz rodzaje zapytań oraz ich przeznaczenie. Podaj polecenia w języku SQL tworzące te zapytania.

**Zapytanie** (niekiedy zwane kwerendą, z łac. *quaerenda*) – czynność polegająca na zbieraniu lub poszukiwaniu informacji w aktach, bibliotekach, a przede wszystkim bazach danych.

Użytkownik serwera baz danych (program lub osoba) wysyła do niego zapytanie, na które serwer odpowiada przesyłając oczekiwane dane, czyli *wynik zapytania*. Zapytania mogą mieć na celu wyłącznie pobranie danych (tzw. zapytania wybierające), jak i usuwanie, dodawanie czy modyfikację danych (tzw. zapytania funkcjonalne).

**Kwerenda wybierająca** jest najczęściej używanym rodzajem kwerendy. Służy do otrzymywania danych z tabeli lub tabel i wyświetlania wyników w arkuszu danych, w którym można je następnie przeglądać. Kwerendy wybierające mogą być również używane do grupowania rekordów i obliczania sum, zliczeń, wyliczania średnich i przeprowadzania innych obliczeń.

Polecenia SQL:

SELECT dane FROM tabele WHERE warunki – pobieranie danych z tabel

INSERT INTO tabela VALUES dane – dodanie danych do tabeli

UPDATE tabela SET dane WHERE warunki – uzupełnienie/zmiana danych w tabeli

DELETE FROM tabela WHERE warunki – usuniecie rekordów z tabeli

CREATE TABLE tabela – utworzenie tabeli

DROP TABLE tabela – usuniecie tabeli

ALTER TABLE tabela ADD/REMOVE – zmiana kolumn tabeli

# 21.[puste]Podstawowe równania teorii sprężysto - plastycznych odkształceń.

Założnia:

- wartości odkształceo sprężystych i plastycznych są zbliżone

- odkształcenia objętościowe są sprężyste

- obciążenie jest monotoniczne i proste

- odkształcenia są małe

# 22.[puste]Podstawy teorii plastycznego płynięcia.

# 23.Ogólne zasady metody elementów skończonych.

Algorytm:

1. W rozpatrywanym ośrodku bierzemy pod uwagę ograniczoną ilość punktów (węzłów).
2. Wartości temperatury (lub innej funkcji) w każdym węźle definiujemy jako parametr, który musimy wyznaczyć.
3. Strefa wyznaczenia temperatury dzieli się na ograniczoną ilość pod-stref, które nazywamy elementami skończonymi.
4. Temperaturę aproksymuje się na każdym elemencie za pomocą wielomianu, który wyznaczony jest za pomocą węzłowych wartości temperatury. Wyznacza się go w taki sposób, aby zachować warunek ciągłości temperatury na granicach elementów.
5. Węzłowe wartości temperatury muszą być dobrane w taki sposób, aby zapewnić najlepsze do rzeczywistego przybliżenie pola temperatury. Taki dobór wykonywany jest za pomocą minimalizacji funkcjonału, który odpowiada różniczkowemu równaniu przewodzenia ciepła.

# 24.Przedstaw podstawowe cechy oraz zasady implementacji technologii XML. Omów kilka wybranych technologii stworzonych w oparciu o XML.

XML - eXtensible Markup Language z ang. „rozszerzalny język znaczników” to otwarty standard opracowany przez W3C (*World Wide Web Consortium*), zajmującą się systematyzowaniem standardów internetowych.

XML nie jest kolejnym językiem do przechowywania konkretnych danych, jak np. język HTML opisujący wygląd stron sieciowych. XML to język opisujący dane, czyli metajęzyk. W uproszczeniu można powiedzieć, że XML służy do tworzenia innych języków (aplikacji XML) służących do przechowywania informacji.

XML pozwala na zapis określonych danych o określonej strukturze.

W przeciwieństwie do np. HTML, XML nie ma ograniczonej liczby znaczników, pozwala przechowywać dowolne dane i to w jak najbardziej wygodny sposób, określany przez użytkownika.

Programista określa strukturę danych, która może być tabelaryczna, ale może także tworzyć drzewo. W ten sposób twórca zbiorów XML nie jest w żaden sposób ograniczony. Na tym przede wszystkim polega wyższość XML nad innymi formatami zapisu danych.

Prawidłowo wygenerowany dokument XML powinien być tworzony zgodnie z kilkoma zasadami:

powinien zawierać deklarację XML, która może posiadać atrybuty takie jak:

* version - określa wersję specyfikacji XML – wersja 1.0
* encoding – deklaruje zestaw znaków używanych w dokumencie XML, wartością domyślną jest kodowanie unicode „UTF-8”
* standalone – określa tryb dokumentu XML, może przyjmować wartość yes lub no. Jeśli ustawimy wartość na yes to będzie oznaczało, że dokument nie zawiera innych plików, które muszą zostać przetworzone wraz z dokumentem.

W XML używane są tzw. tagi. Każdy z nich jest elementem drzewa dokumenty XML i może zawierać w sobie inne tagi. Wszystkie tagi muszą być zamknięte. Muszą być odpowiednio zagnieżdżane (Tag-dziecko musi być zamknięty przed takgiem-rodzicem). XML jest czuły na wielkość liter.

Znak dwukropka jest zarezerwowany dla identyfikacji przestrzeni nazw, która jest dopisywana przed nazwą elementu. Nazwy nie mogą zaczynać się od znaków łącznika -, cyfry, kropki.

Każdy element może zawierać atrybuty, które definiuje się w znaczniku otwierającym element, ich wartości podaje się w cudzysłowach. Komentarze rozpoczyna się <!--a kończy -->.

Znaki szczególne nie mogą się pojawiać w danych, nazwach oraz atrybutach, są to m.in. < oraz &. Zamiast nich wprowadza się np. &lt; oraz &amp;.

Instrukcje przetwarzania umieszcza się w znakach <? Oraz ?>.

Technologie stworzone w oparciu o XML:

* SVG – Scalable Vector Graphics - uniwersalny format dwuwymiarowej grafiki wektorowej (statycznej i animowanej). W SVG oprócz standardowych obiektów (prostokąty, elipsy, krzywe) można opisywać efekty specjalne (filtry), maski przezroczystości, wypełnienia gradientowe itp. W SVG można też opisać sposób animacji elementów za pomocą standardowych. SVG pozwala na użycie języków skryptowych (np. JavaScript), szablonów stylów (CSS), a także na rozszerzanie funkcjonalności przez dodanie własnych elementów i właściwości przy pomocy standardowych technik XML (przestrzenie nazw). Również w drugą stronę – SVG może być użyty wewnątrz innego dokumentu (aplikacji XML).
* XAML – eXtensible Application malkup language - jest językiem opisu interfejsu użytkownika wykorzystywanym m.in. w technologii Windows Presentation Foundation. Pliki XAML zazwyczaj tworzone są ręcznie lub przy pomocy wizualnych narzędzi, takich jak Microsoft Expression Studio. Wynikowy kod XML interpretowany jest w locie przez podsystem systemu operacyjnego odpowiedzialny za wyświetlanie danych, co zastępuje GDI z poprzednich wersji Windows. Elementy XAML mapowane są do obiektów Common Language Runtime. Atrybuty mapowane są do właściwości lub zdarzeń tych obiektów. XAML został stworzony w celu zapewnienia wsparcia klasom i metodom w architekturze .NET, które zajmują się obsługą interakcji z użytkownikiem, zwłaszcza wyświetlaniem. Język ten może być wykorzystywany jako język opisu nie tylko technologii Windows Presentation Foundation, lecz także innych, jak np. Windows Workflow Foundation.
* MathML - to język będący zastosowaniem XML-a, służący do opisywania wzorów i symboli matematycznych. Specyfikacje języka MathML opracowywane są przez W3C. Aktualną wersją jest MathML 2.0. W wersji tej wprowadzono podział na znaczniki prezentacyjne, opisujące sposób renderowania wzorów, oraz znaczniki semantyczne, opisujące matematyczne znaczenie wzorów i mogące być wykorzystywane do obliczeń. Oba typy znaczników można stosować jednocześnie. MathML jest na coraz szerszą skalę wykorzystywany do prezentacji wzorów w Internecie na stronach WWW. Znaczniki MathML można umieścić zarówno w osobnym dokumencie (z rozszerzeniem .mml), jak i włączyć do dokumentu XHTML lub HTML 5.

# 25.Opisz sposób implementacji oprogramowania oparty o architekturę Model-View-Controller.

<http://pl.wikipedia.org/wiki/Model-View-Controller>

# 26.Przedstaw technologię Single Sign-On (SSO) oraz przykładowe narzędzia wykorzystywane do pojedynczego logowania.

**Pojedyncze logowanie** (ang. *single sign-on*) – możliwość jednorazowego zalogowania się do usługi sieciowej i uzyskania dostępu do wszystkich autoryzowanych zasobów zgodnych z tą usługą. Technika pojedynczego logowania jest stosowana m.in. w Windows Live, Central Autentication Service (CAS, Yale University).

W niektórych publikacjach preferowany jest termin *zredukowane logowanie*, według nich nazwa *pojedyncze logowanie* jest błędna, bo "nie da się go osiągnąć w niejednolitej strukturze IT".

W strukturze jednolitej lub gdzie scentralizowano bazę użytkowników, technika pojedynczego logowania posiada jasno widoczne zalety. Wszyscy użytkownicy w tej strukturze posiadaliby pojedyncze dane uwierzytelniające (na przykład zachowane w katalogu LDAP), co skracałoby czas poświęcany na rejestrację i logowanie dla użytkownika. Wszystkie procesy mogłyby wtedy używać wspólnej bazy służącej do uwierzytelniania i autoryzacji użytkowników. Problem pojawia się w momencie gdy nieuprawniona usługa lub osoba wykorzysta dane dostępne we wspólnym repozytorium przeciwko użytkownikowi.

Narzędzia:

* Kerberos – protokół uwierzytelnienia i autoryzacji z wykorzystaniem Centrum Dystrybucji Kluczy (implementacja w Java – General Security Service)
* Karta elektroniczna, karta chipowa - uniwersalny nośnik danych w postaci karty wykonanej z plastiku z umieszczonym na niej (lub wewnątrz niej) jednym lub kilkoma układami scalonymi, które pozwalają na ochronę procesu logowania użytkownika, kontrolę dostępu i zawartych na niej danych. Może być odczytywana za pomocą urządzeń automatycznych, np. przy zawieraniu i rozliczaniu transakcji finansowych oraz w kasach cyfrowych.
* Token haseł jednorazowych
* Integrated Windows Authentication
* SAML - (ang. Security Assertion Markup Language) - nazwa protokołu wykorzystywanego do pośredniczenia w uwierzytelnianiu i automatycznego przekazywania między systemami i aplikacjami informacji o uprawnieniach użytkowników. Protokół ten bazuje na standardzie XML.

Więcej: <http://en.wikipedia.org/wiki/Single_sign-on>

# 27.Omów podstawowe metody algorytmiczne: metodę "dziel i zwyciężaj", metodę zachłanną, programowania dynamicznego. Podaj dla każdej z nich przykłady zastosowania.

## Metoda dziel i zwyciężaj.

W strategii tej problem dzieli się rekurencyjnie na dwa lub więcej mniejszych podproblemów tego samego (lub podobnego) typu tak długo, aż fragmenty staną się wystarczająco proste do bezpośredniego rozwiązania. Z kolei rozwiązania otrzymane dla podproblemów scala się, uzyskując rozwiązanie całego zadania.

**Sortowanie quicksort**:

Algorytm działa rekurencyjnie - wybierany jest pewien element tablicy, tzw. element osiowy, po czym na początek tablicy przenoszone są wszystkie elementy mniejsze od niego, na koniec wszystkie większe, a w powstałe między tymi obszarami puste miejsce trafia wybrany element. Potem sortuje się osobno początkową i końcową część tablicy. Rekursja kończy się, gdy kolejny fragment uzyskany z podziału zawiera pojedynczy element, jako że jednoelementowa podtablica nie wymaga sortowania.

## Algorytm zachłanny.

Algorytm, który 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.

**Problem wydawania reszty**:

Przykładowo, dane są dwa rodzaje monet: 2 zł i 5 zł. Należy obliczyć ile, i jakich monet należy wydać, by reszta wynosiła 6 zł.

Gdy dobór pierwszej monety będzie zachłanny (tj. algorytm wybierze jedną "piątkę", bo 1 \* 5zł jest bliżej wyniku ostatecznego (jest lokalnie lepszym rozwiązaniem), niż 1 \* 2zł. Jednak już w następnym kroku okaże się, że droga zachłanna była w tym przypadku drogą ślepą. Postępując niezachłannie ostatecznie dochodzimy do prawidłowego i optymalnego wyniku.

## Programowanie dynamiczne.

W kontekście algorytmiki programowanie dynamiczne nie jest rozumiane jako pisanie programów komputerowych, lecz jako metodę rozwiązywania problemów za pomocą rozbicia go na mniejsze podproblemy (rozwiązania podproblemów umieszczane są w tablicy, stąd nazwa "programowanie").  
Programowanie dynamiczne stosuje się najczęściej do rozwiązywania tzw. problemów optymalizacyjnych. W takich zagadnieniach istnieje zazwyczaj więcej, niż jedno rozwiązanie. Dzięki zastosowaniu tytułowego algorytmu, można szybko wyznaczyć maksymalny bądź minimalny koszt, dzięki rozwiązywaniu mniejszych podproblemów i wykorzystywaniu tych rezultatów przy dochodzeniu do wyników coraz większych podproblemów (tzw. metoda wstępująca).

**Problem plecakowy:**

Problem plecakowy, którego rozwiązanie polega na podaniu maksymalnej wartości przedmiotów zapakowanych do plecaka, przy ograniczonej jego pojemności oraz określonej wartości i ilości zajmowanego miejsca poszczególnych przedmiotów.  
  
Dla przykładu posłużę się następującymi danymi:

Pojemność plecaka: 10

Przedmioty:

1) wartość: 3; objętość: 2

2) wartość: 5; objętość: 3

3) wartość: 1; objętość: 1

Do zapisywania danych najlepiej użyć tablicy jednowymiarowej od 1 do 10. Kolejne komórki w takiej tablicy będą oznaczały pojemności plecaka, a wartościami tych komórek będą maksymalne wartości plecaków o danej pojemności (na początku wszystkie przyjmują wartość zero, gdyż nie został jeszcze rozpatrzony żaden przedmiot).  Programowanie dynamiczne polega na stopniowym rozwiązywaniu problemu - tak więc najpierw uzyskujemy wynik optymalnego wypełnienia plecaka o pojemności 1, potem 2, itd.  
  
Tablicę wypełniamy, biorąc pod uwagę kolejne przedmioty. Tak więc rozpoczynamy od pierwszej rzeczy. Zajmuje ona dwie jednostki. Wiadomo, że nie zmieści się do plecaka o pojemności 1 - komórka numer jeden pozostaje bez zmian. Natomiast do plecaka dwujednostkowego zmieścimy jeden taki przedmiot - do trzyjednostkowego podobnie, a do torby o pojemności cztery upchniemy już dwie sztuki, itd.  
  
Teraz nieco trudniejszy etap - bierzemy pod uwagę drugi przedmiot. Wypełnianie rozpoczynamy od trzeciej komórki (gdyż do mniejszych plecaków przedmiot się nie zmieści, a to nie przyniesie żadnych zmian). Mamy w tym przypadku do wyboru upakowanie za pomocą przedmiotu nr 1 (wartość: 3) lub nr 2 (wartość: 5). Optymalniejszym wyborem jest druga możliwość, więc wartość trzeciej komórki należy ustawić na 5. Jeśli istniałaby możliwość optymalnego wypakowania przedmiotami więcej, niż jednego rodzaju, należy taki sposób zastosować (stanie się tak w piątej komórce - jeden przedmiot nr 1 i jeden nr 2). Konkretnie: do wypełnienia komórki wybieramy jedną z dwóch wartości: aktualna wartość komórki

t[i]

lub wartość komórki o numerze pomniejszonym o objętość przedmiotu plus wartość tegoż przedmiotu

t[i-obj[j]] + wart[j].

Dalej tablicę wypełniamy analogicznie. Wynikiem działania algorytmu będzie tablica, której to wartościami komórek o poszczególnych indeksach będą maksymalne wartości przedmiotów, jakie można zmieścić do plecaka o pojemności równej indeksowi tablicy. Więc dla naszych danych poprawny wynik zapisany jest w dziesiątej komórce.

# 28.Złożoność obliczeniowa algorytmu. Zdefiniuj pojęcie notacji asymptotycznej. Podaj klasy złożoności wraz z przykładami algorytmów.

## Złożoność obliczeniowa algorytmu.

**Złożoność czasowa.**

Przyjętą miarą złożoności czasowej jest liczba operacji podstawowych w zależności od rozmiaru wejścia. Pomiar rzeczywistego czasu zegarowego jest mało użyteczny ze względu na silną zależność od sposobu realizacji algorytmu, użytego kompilatora oraz maszyny na której algorytm wykonujemy. Dlatego w charakterze czasu wykonania rozpatruje się zwykle liczbę operacji podstawowych (dominujących). Operacjami podstawowymi mogą być na przykład: podstawienie, porównanie lub prosta operacja arytmetyczna.

Kolejny problem polega na tym, w jakim języku programowania formułować będziemy algorytmy oraz co można założyć o maszynie, na której algorytm ten będzie wykonywany. Istniejące komputery różnią się między sobą istotnymi (z punktu widzenia konstruowania algorytmów) parametrami, jak na przykład liczba i rozmiar rejestrów, udostępnianymi operacjami matematycznymi, a ponadto podlegają ciągłym ulepszeniom. Wobec tego algorytmy analizuje się, wykorzystując abstrakcyjne modele obliczeń. Do popularnych modeli należą maszyna RAM, maszyna Turinga i maszyna wskaźnikowa.

**Złożoność pamięciowa.**

Podobnie jak złożoność czasowa jest miarą czasu działania algorytmu, tak złożoność pamięciowa jest miarą ilości wykorzystanej pamięci. Jako tę ilość najczęściej przyjmuje się użytą pamięć maszyny abstrakcyjnej (na przykład liczbę komórek pamięci maszyny RAM) w funkcji rozmiaru wejścia. Możliwe jest również obliczanie rozmiaru potrzebnej pamięci fizycznej wyrażonej w bitach lub bajtach.

Jak wyznaczyć złożoność czasową algorytmu? Załóżmy, że program musi wykonać dla n danych 2n dodawań i 4n mnożeń, co da nam w sumie 6n operacji dominujących. Możemy więc liczbę operacji potraktować jako funkcję zmiennej n - czyli rozmiaru danych wejściowych. W tym wypadku mielibyśmy f(n)=6n. Funkcja byłaby funkcją liniową, czyli algorytm miałby **liniową złożoność obliczeniową,** ponieważ czas wykonania zależy w sposób liniowy od rozmiaru danych wejściowych n. W praktyce ważny jest tylko typ zależności. Na przykład dla funkcji f(n)=3n2+5 mamy **kwadratową złożoność obliczeniową**.

## Notacja asymptotyczna.

Asymptotyczne tempo wzrostu jest miarą określającą zachowanie wartości funkcji wraz ze wzrostem jej argumentów. Stosowane jest szczególnie często w teorii obliczeń, w celu opisu złożoności obliczeniowej, czyli zależności ilości potrzebnych zasobów (np. czasu lub pamięci) od rozmiaru danych wejściowych algorytmu. Asymptotyczne tempo wzrostu opisuje jak szybko dana funkcja rośnie lub maleje, abstrahując od konkretnej postaci tych zmian.

Więcej: <http://www.bioinformatyk.eu/index.php/matematyka/notacja-asymptotyczna.html>

## Klasy złożoności.

* O(1) – złożoność stała – funkcje hashujące
* log(n) - złożoność logarytmiczna – wyszukiwanie binarne w zbiorze uporządkowanym. Sprawdzenie środkowego elementu pozwala określić, w której części znajduje się poszukiwany element.
* n - złożoność liniowa – mnożenie wektora przez stałą
* nlog(n) - złożoność liniowo-logarytmiczna - quicksort
* n2 - złożoność kwadratowa – sortowanie bąbelkowe
* nk – złożoność wykładnicza – operacje na macierzach
* 2n - złożoność wykładnicza - Złożoność obliczeniową O(2n) posiada algorytm, w którym wykonywana jest stała liczba operacji dla każdego podzbioru n danych wejściowych. Złożoność obliczeniową O(n!) posiada algorytm, w którym wykonywana jest stała liczba operacji dla każdej permutacji n danych wejściowych. Np. wyszukiwanie największego podzbioru poprzez znalezienie wszystkich podzbiorów i wybranie tego największego.

# 29.[puste]Zdefiniuj pojęcie analizy wrażliwości. Przedstaw przykłady zastosowania.

# 30.[puste]Zagadnienie uwarunkowania problemów odwrotnych. Metody regularyzacji.