# ZŁOŻONOŚĆ OBLICZENIOWA

Złożoność obliczeniowa to ilość zasobów komputerowych koniecznych do wykonania programu realizującego algorytm. Przedstawiamy ją jako funkcję pewnego parametru, określającego rozmiar rozwiązywanego zadania. Wyróżniamy złożoność pamięciową i czasową.

Złożoność pamięciowa to ilość pamięci wykorzystanej w celu realizacji algorytmu, wyrażana w liczbie bajtów lub liczbie zmiennych typów elementarnych. Złożoność czasowa jest to czas wykonania algorytmu. Złożoności czasowej nie wyrażamy w standardowych jednostkach czasu, gdyż w zależności od tego, jaką maszyną dysponujemy, algorytm będzie wykonywał się na różnych maszynach w różnym czasie. Dlatego złożoność obliczeniową czasową wyrażamy w ilości operacji dominujących.

* Złożoność optymistyczna określa zużycie zasobów dla najkorzystniejszego zestawu danych.
* Złożoność średnia określa zużycie zasobów dla typowych (losowych) danych.
* Złożoność pesymistyczna określa zużycie zasobów dla najbardziej niekorzystnego zestawu danych.

Złożoności wielomianowe:

* Liniowa
* Kwadratowa
* Sześcienna
* Logarytmiczna
* Quasiliniowa
* NP-zupełna
* Wykładnicza
* Silnie NP-zupełna
* NP-trudna

Notacje złożoności obliczeniowej:

* Notacja O (“duże O”)
* Notacja Θ
* Notacja Ω

Mówimy, że (funkcja złożoności obliczeniowej jest rzędu funkcji ,   
jeśli potrafimy znaleźć takie oraz takie , iż dla każdego prawdziwa jest nierówność:

Mówimy, że jeśli istnieją stałe dodatnie , i   
takie , że dla każdego prawdziwa jest nierówność:

Mówimy, że jeśli istnieją stałe dodatnie i takie, że dla każdego prawdziwa jest nierówność:

# ABSTRAKCYJNE TYPY DANYCH

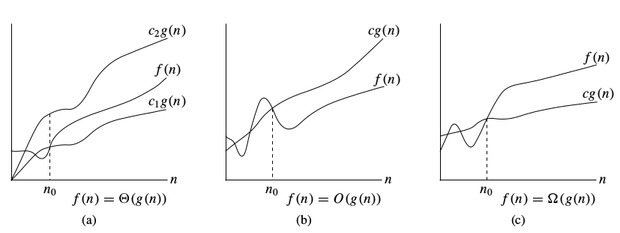
Abstrakcyjny typ danych to konstrukcja języka programowania, w której definiujemy typ oraz operacje na nim w taki sposób, że inne byty w programie nie mogą manipulować danymi inaczej, niż za pomocą zdefiniowanych przez nas operacji. Abstrakcyjne typy danych są osiągalne dzięki enkapsulacji, czyli ukrywaniu przed użytkownikiem szczegółów implementacji.

Lista – struktura danych służąca do reprezentacji danych sekwencyjnych; składa się z ciągu elementów tego samego typu. Dostęp do elementów listy jest sekwencyjny.

Implementacja list jest często zapewniana przez odpowiedni typ w języku programowania, np. w Javie. Lista może być także zaimplementowana w formie tablicy o zmiennej długości.

Do zastosowań listy w ujęciu abstrakcyjnym należą implementacje kolejki, stosu, często również są używane zamiennie ze zbiorami (*sets*) z uwagi na łatwiejszą implementację.

Operacje na listach:

* get() – zwróć dowolny element listy
* insert() – dodaj element do listy (w dowolnym miejscu)
* remove()
* size()
* isEmpty()

Operacje dostępu do danego element listy (jak również jego wstawienie lub usunięcie) mają liniową złożoność czasową O(n). Dostęp do pierwszego/ostatniego elementu listy, w przypadku zdefiniowania takiej metody, ma stałą złożoność czasową O(1).

Kolejka jest strukturą charakteryzującą się dostępem FIFO – *First In First Out*, analogicznie do kolejki sklepowej. Elementy kolejki zachowują swoją kolejność.

Podstawowymi (i często jedynymi) operacjami na kolejkach są:

* enqueue() – „zakolejkuj”, dodaj element na koniec kolejki
* dequeue() – „odkolejkuj”, usuń element z początku kolejki

Operacje te mają złożoność czasową stałą O(1).

Kolejki typowo implementowane są za pomocą list wiązanych (ang. *linked list*) lub buforów kołowych (ang. *circular buffer*).

Kolejka priorytetowa – kolejka działająca na bazie zwyczajnej kolejki, ale z uwzględnieniem priorytetów elementów, czyli wartości liczbowych przypisanych elementom. Elementy z najwyższym priorytetem odkolejkowywane są w pierwszej kolejności.

Kolejki priorytetowe są zwykle implementowane za pomocą kopców (ang. *heap*), czyli struktury będącej kompletnym drzewem binarnym (gdy wszystkie jego poziomy z wyjątkiem ostatniego są zapełnione). Podobnie jak kopce, kolejka priorytetowa może być typu max lub min (max – dla każdego węzła niebędącego korzeniem wartość tego węzła jest mniejsza lub równa wartości rodzica).

Podstawowe operacje na kolejce priorytetowej:

* buildHeap() – zbudowanie kopca
* insert() – wstaw element
* remove() – usuń element

Kolejka priorytetowa może być zaimplementowana w sposób prymitywny za pomocą nieuporządkowanej listy – wtedy operacja wstawienia elementu będzie miała złożoność O(1), ale z uwagi na konieczność każdorazowego przeszukiwania całej listy w celu wyciagnięcia elementu operacja wyciągnięcia ma złożoność O(n).

W implementacji na bazie kopca złożoność wstawienia i wyciągnięcia elementu z kolejki priorytetowej jest logarytmiczna O(log n). Jednak w tym przypadku może istnieć konieczność początkowego zbudowania kopca, które ma złożoność czasową liniową O(n).

Kolejka priorytetowa może intuicyjnie służyć np. do zarządzania przepustowością w celu priorytetyzowania ruchu sieciowego. Kolejki priorytetowe są również używane w tzw. symulatorach zdarzeń dyskretnych.

# PROGRAMOWANIE OBIEKTOWE

Dziedziczenieto mechanizm współdzielenia funkcjonalności między klasami. Klasa może dziedziczyć po innej klasie, co oznacza, że oprócz swoich własnych atrybutów oraz zachowań, uzyskuje także te pochodzące z klasy, z której dziedziczy. Dziedziczenie umożliwia zdefiniowanie hierarchii klas. Klasa bazowa (ang. *base class*) bądź inaczej nadklasa (ang. *superclass*) definiuje składowe wspólne dla wszystkich wariantów. Klasy pochodne (ang. *derived class*) bądź inaczej podklasy (ang. *subclass*) definiują pozostałe składowe, które występują tylko w poszczególnych wariantach. Związek między nadklasą i podklasami nazywany jest uogólnieniem (ang. *generalization*), bądź związkiem uogólnienie-uszczegółowienie.

Enkapsulacja (hermetyzacja, kapsułkowanie) - hermetyzacja polega na ukrywaniu pewnych danych składowych lub metod obiektów danej klasy tak, aby były one dostępne tylko metodom wewnętrznym danej klasy lub funkcjom zaprzyjaźnionym. Enkapsulacja ma 3 podstawowe zalety: wyodrębnia interfejs i jasno określa odpowiedzialność klasy (która klasa co robi), uodparnia model na błędy (np. wynikające z przypisywania nieprawidłowych wartości polom danego obiektu przez inne obiekty) oraz lepiej oddaje rzeczywistość. Enkapsulację możemy osiągnąć (np. w Javie) stosując modyfikatory dostępu. Lista modyfikatorów dostępu występujących w Javie: public (pozwala na dostęp do danego elementu ze wszystkich klas), protected (dostęp z klasy, w której zawarta jest deklaracja, klas dziedziczących oraz z klas z tego samego pakietu), private (dostęp do danego elementu ograniczony tylko do klasy, w którym jest zdefiniowany), package (dostęp ze wszystkich klas pakietu) oraz brak modyfikatora (równoważny z modyfikatorem package). Prawa dostępu dotyczą klas, a nie obiektów. W C++ nie ma organizacji klas w pakiety i wyróżnia się tylko 3 modyfikatory dostępu: public, protected oraz private.

Polimorfizm - możliwość stworzenia obiektu, który może mieć więcej niż jedną formę. W praktyce oznacza to, że może on mieć metodę, która dla tego samego typu obiektów mają różne definicje. Przykładowo mamy wiele klas (dziedziczących z jednej klasy abstrakcyjnej) dla różnych rodzajów figur geometrycznych a dla każdej z tych figur możemy policzyć jej pole. Polimorfizm pozwala nam w każdej z tych klas zaimplementować metodę wirtualną o takiej samej nazwie np. calcArea(). W celu zastosowania polimorfizmu używa się przedefiniowania (nadpisywania) metod.

Polimorfizm statyczny – na poziomie kompilacji (np. przy przeciążeniu operatorów), polimorfizm dynamiczny – w czasie wykonania (np. przy użyciu metod wirtualnych).

Abstrakcja - polega na ukrywaniu lub pomijaniu mało istotnych informacji a skupieniu się na wydobyciu informacji, które są niezmienne i wspólne dla pewnej grupy obiektów. Pojęciem abstrakcyjnym jest np. środek transportu. Środkiem transportu może być zarówno samolot jak i samochód. Zarówno samolot jak i samochód porusza się z pewną prędkością, a więc prędkość jest wspólnym parametrem. Idąc dalej samochód może być osobowy lub ciężarowy. Każdy z nich posiada cechy wspólne jak np. spalanie czy waga.

# MODELE OBLICZEŃ

Automat skończony deterministyczny , to dowolna taka piątka , w której:

* to skończony zbiór stanów,
* to skończony alfabet wejściowy,
* to funkcja przejścia - to stan, do którego przechodzimy ze stanu q na skutek wczytania znaku a,
* to stan początkowy,
* to zbiór stanów akceptujących.

Automat skończony wczytuje dane słowo znak po znaku i po wczytaniu całego słowa udziela odpowiedzi, czy słowo to należy do ustalonego języka. Przejścia między stanami zachodzą w automacie skończonym na skutek wczytywania kolejnych znaków analizowanego słowa. Automaty skończone odpowiadają klasie gramatyk liniowych wg. hierarchii Chomsky’ego.

W automacie deterministycznym, dla każdego znaku jaki może się pojawić na wejściu i dla każdego stanu automatu ze stanu tego wychodzi dokładnie jedno przejście odpowiadające danemu znakowi – w każdej sytuacji działanie automatu musi być określone jednoznacznie.

Automat skończony ma skończoną liczbę stanów. Oznacza to, że dysponuje on stałą pamięcią dodatkową, tzn. wielkość pamięci dodatkowej nie zależy od długości wczytywanego słowa i jest ograniczona przez stałą.

Automat skończony niedeterministyczny , to dowolna taka piątka , w której:

* to skończony zbiór stanów,
* to skończony alfabet wejściowy,
* to funkcja przejścia - to stan, do którego przechodzimy ze stanu q na skutek wczytania znaku a,
* to zbiór stanów początkowych,
* to zbiór stanów akceptujących.

Automat niedeterministyczny jest automatem, którego działania nie da się w pełni przewidzieć. Będąc w tym samym stanie i wczytując ten sam znak może wykonać różne przejścia. Przejść może być dowolnie wiele, a nawet może ich nie być wcale. Automat taki może również mieć wiele stanów początkowych.

Automat ze stosem (stosowy) to dowolna taka szóstka , w której:

* to skończony zbiór stanów,
* to skończony alfabet wejściowy,
* to skończony alfabet stosowy,
* to skończona relacja przejścia – określa ona dla danego aktualnego stanu, znaku na wierzchołku stosu i znaku czekającego na wczytanie, do jakiego stanu należy przejść i co należy włożyć na stos (uprzednio zdjąwszy element z wierzchołka stosu)
* to stan początkowy,
* to symbol początkowy na stosie

Automat stosowy, podobnie jak automat skończony, ma skończony zbiór stanów i stan początkowy. Dodatkowo jest on wyposażony w stos, na który może wkładać elementy, podglądać element znajdujący się na wierzchu i zdejmować elementy. Przejścia automatu stosowego są trochę bardziej skomplikowane niż przejścia w automacie skończonym, gdyż oprócz wczytywania znaków z wejścia i zmiany stanów obejmują również operacje na stosie. Automat taki może być niedeterministyczny i może zawierać   
ε-przejścia, tzn. wykonaniu przejścia nie musi towarzyszyć wczytanie znaku z wejścia. Automaty stosowe są pozbawione stanów akceptujących – automat będzie sygnalizował akceptację wejścia opróżniając stos.

Automat stosowy w jednym kroku obliczeń wykonuje następujące czynności:

1. Podgląda znak czekający na wczytanie na wejściu.
2. Podgląda element na wierzchołku stosu.
3. Na podstawie tych informacji wybiera jedno z przejść do wykonania.
4. Jeśli to nie jest ε-przejście, to wczytywany jest jeden znak z wejścia.
5. Zdejmowany jest element z wierzchołku stosu.
6. Pewna liczba określonych elementów może być włożona na stos.
7. Zgodnie z przejściem zmieniany jest stan.

# SCHEMAT BUDOWY PROCESORA

Procesor, także CPU (*Central Processing Unit*) – to główny element komputera, urządzenie cyfrowe sekwencyjne, którego zadaniem jest wykonywanie rozkazów i sterowanie pracą wszystkich pozostałych bloków systemu (m.in. pamięci i układów wejścia-wyjścia). Procesor przetwarza informacje, wykonując na niej elementarne operacje zwane instrukcjami maszynowymi (bądź rozkazami). Ciąg takich instrukcji realizujący konkretne zadanie przetwarzania informacji nazywamy programem.

Wszystkie mikroprocesory zawierają podobne elementy:

1. układ sterowania i synchronizacji, który kontroluje pracę procesora i wytwarza sygnały potrzebne do sterowania niektórymi elementami komputera
2. arytmometr, czyli układ, który wykonuje operacje arytmetyczne i logiczne
3. rejestry, tj. układy pamięci
4. wbudowana pamięć podręczna cache
5. koprocesor matematyczny
6. wewnętrzne szyny łączące elementy procesora

**ALU** - układ ten jest często nazywany układem wykonawczym procesora, ponieważ wykonuje on większość rozkazów zlecanych procesorowi. Blok ALU jest układem, który realizuje różne funkcje w zależności od zaprogramowanej operacji, tj. rozkazu umieszczonego w programie. Argumentami tych rozkazów są słowa binarne. Rozkazy te mogą dotyczyć operacji arytmetycznych (dodawanie i odejmowanie) lub logicznych (sumowanie, mnożenie itp.).

**Rejestry** - rejestry są szybkimi obszarami składowania danych wewnątrz procesora. Dostęp do nich jest o wiele szybszy niż do pamięci operacyjnej, a zatem operacje wykorzystujące tylko zawartość rejestrów są wykonywane szybciej niż odwołujące się do pamięci operacyjnej. Rejestry procesora mogą pełnić różne role w programach. Akumulatorem nazywamy rejestr, który może być użyty jako argument źródła i równocześnie przeznaczenia dla operacji arytmetycznej lub logicznej. Rejestr służący do uzyskania adresu danej umieszczonej w pamięci nazywa się ogólnie rejestrem adresowym. Rejestr może również służyć do odliczania iteracji pętli. Rejestr przewidziany do takiego zastosowania nazywa się licznikiem pętli.

Oznaczenia:  
A – akumulatory  
B, C, D, E, H, L - rejestry robocze SP - wskaźnik stosu  
F - rejestr flagowy  
PC - licznik rozkazów

**Zegar** – układ elektroniczny generujący (zwykle) sygnał prostokątny przyjmujący wartości logiczne 0 i 1 w ustalonych odstępach czasu (zwanych okresem). Częstotliwość zegara nazywana jest taktowaniem. Układy takie jak np. mikroprocesory, kości pamięci, wykonują jedną podstawową, jednostkową operację za każdym razem, gdy dotrze do nich impuls taktujący. Tak więc procesor taktowany częstotliwością   
1 GHz wykonuje 1 miliard podstawowych operacji w ciągu sekundy.

# UKŁADY SEKWENCYJNE I KOMBINACYJNE

Układem kombinacyjnym nazywamy układ cyfrowy, w którym sygnały wyjściowe są jednoznacznie określone przez aktualne wartości sygnałów wejściowych. Układ kombinacyjny o wejściach i wyjsciach jest w pełni opisany przez podanie funkcji przełączającej , gdzie jest zbiorem dopuszczalnych stanów wejść, a jest zbiorem możliwych stanów wyjść.

System funkcjonalnie pełny to zestaw typów bramek, z których można zbudować dowolny układ kombinacyjny.

Przykłady najważniejszych systemów bramek funkcjonalnie pełnych:

* AND, OR, NOT
* NAND
* NOR.

Przykłady układów kombinacyjnych: sumator, multiplekser, demultiplekser.

# RELACYJNY MODEL DANYCH

W modelu relacyjnym, baza danych jest zbiorem relacji. Każda relacja posiada swój tzw. schemat, który składa się z listy atrybutów. Schemat relacji R jest często oznaczany jako , gdzie   
A1, A2, ..., An oznaczają atrybuty. Liczbę atrybutów składających się na schemat relacji R nazywamy stopniem relacji. Relacja jest zbiorem krotek, które są listami wartości. Intuicyjnie, relacja ma postać klasycznej tabeli z kolumnami i wierszami. Kolumny odpowiadają atrybutom relacji, a wiersze odpowiadają krotkom.

Kluczem K schematu relacji R nazywamy taki **superklucz**, że usunięcie atrybutu A z K spowoduje, że K’ = K – A nie jest już superkluczem. Superklucz to każdy podzbiór atrybutów relacji, dla których wartość krotek się nie powtarza. Klucz jest minimalnym superkluczem zachowującym własność unikalności krotek relacji. Jeden z kluczy może być wyróżniony jako tzw. klucz podstawowy.

Klucz podstawowy relacji – atrybut, którego wartość jednoznacznie identyfikuje krotkę, jego wartość jest unikalna w obrębie całej relacji i jest niepusta.

Klucz obcy relacji - atrybut lub zbiór atrybutów, który wskazuje na klucz podstawowy innej relacji. Służy do łączenia relacji – reprezentowania powiązań między danymi.

Podstawowe operacje algebry relacji:

* Selekcja (SELECT)
* Projekcja (PROJECT)
* Połączenie (JOIN)
* Operacje na zbiorach:
  + Suma (UNION)
  + Część wspólna (INTERSECTION)
  + Różnica (MINUS / DIFFERENCE)

Normalizacja relacjito proces, podczas którego schematy relacji posiadające pewne niepożądane cechy są dekomponowane na mniejsze schematy relacji o pożądanych właściwościach. Proces normalizacji musi posiadać własności:

* Własność zachowania atrybutów – żaden atrybut nie zostanie zagubiony w czasie procesu normalizacji
* Własność zachowania informacji – dekompozycja nie prowadzi do utraty żadnej informacji
* Własność zachowania zależności – wszystkie zależności funkcyjne są reprezentowane w pojedynczych schematach relacji

Proces normalizacji schematu relacji polega na sprawdzeniu czy dany schemat jest w odpowiedniej postaci normalnej, jeżeli nie wówczas następuje dekompozycja schematu relacji na mniejsze schematy relacji. Ponownie, weryfikowana jest postać normalna otrzymanych schematów relacji. Jeżeli nie spełniają one zadanej postaci normalnej to proces dekompozycji jest kontynuowany, dopóki otrzymane schematy relacji nie będą w odpowiedniej postaci normalnej.

Pierwsza postać normalna (1NF) – schemat relacji R znajduje się w pierwszej postaci normalnej, jeśli wszystkie wartości atrybutów są atomowe (niepodzielne).

Druga postać normalna (2NF) – dana relacja r o schemacie R jest w drugiej postaci normalnej (2NF), jeżeli żaden atrybut wtórny tej relacji nie jest częściowo funkcyjnie zależny od żadnego z kluczy relacji r.

Dana relacja r o schemacie R jest w trzeciej postaci normalnej (3NF), jeżeli dla każdej zależności funkcyjnej X → A w R spełniony jest jeden z następujących warunków:

* X jest superkluczem schematu R lub
* A jest atrybutem podstawowym schematu R.

[*Relacja jest w trzeciej postaci normalnej wtedy i tylko wtedy, gdy jest w II postaci normalnej i żaden atrybut niekluczowy nie jest zależny funkcyjnie od innych atrybutów niekluczowych.*]

Dana relacja r o schemacie R jest w postaci normalnej Boyce’a-Codd’a (BCNF), jeżeli dla każdej zależności funkcyjnej X → A w R spełniony jest następujący warunek: X jest superkluczem schematu R.

[*W tej postaci zależności funkcyjne muszą mieć następującą postać: jeżeli X → A i atrybut A nie jest zawarty w X, to X jest kluczem lub zawiera klucz.*]

# TRANSAKCJE W BAZACH DANYCH

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 i akceptację (zatwierdzenie), lub wycofanie transakcji. Transakcje umożliwiają współbieżny dostęp do zawartości bazy danych, dostarczając niezbędnych mechanizmów synchronizacji. Transakcja to zbiór operacji, które mogą być wykonane jedynie wszystkie lub żadna; jej istotą jest integrowanie kilku operacji w jedną niepodzielną całość.

Własności transakcji**:**

* Atomowość
* Spójność
* Izolacja
* Trwałość

Każda realizowana transakcja posiada zbiór ściśle określonych stanów i zbiór ściśle określonych przejść z jednego stanu do drugiego. Stany te są następujące:

* active: transakcja jest aktywna, jest w czasie realizowania swoich operacji
* partially committed: transakcja jest częściowo zatwierdzona
* committed: transakcja została zatwierdzona
* failed: transakcja została wycofana
* terminated: transakcja zakończyła się zatwierdzeniem lub wycofaniem

|  |  |  |  |
| --- | --- | --- | --- |
| Poziom izolacji | Anomalie | | |
| Brudny odczyt | Niepowtarzalny odczyt | Fantomy |
| READ UNCOMMITTED | możliwy | możliwy | możliwy |
| READ COMMITED | brak | możliwy | możliwy |
| REPEATABLE READ | brak | brak | możliwy |
| SERIALIZABLE | brak | brak | brak |

Algorytmy zarządzające współbieżnym wykonywaniem transakcji:

* algorytmy blokowania
* algorytmy znaczników czasowych
* algorytmy optymistyczne – walidacja poprawności uszeregowania

Algorytmy blokowania – blokada jest zmienną skojarzoną z każdą daną w bazie danych, określającą dostępność danej ze względu na możliwość wykonania na niej określonych operacji

Algorytmy znaczników czasowych – algorytmy porządkowania transakcji wg. etykiet czasowych. Uszeregowanie transakcji będących w konflikcie wynika z porządku znaczników czasowych przydzielanych transakcjom w momencie ich inicjacji. Porządek jest więc predefiniowany, a nie ustalany dynamicznie, jak w przypadku algorytmów blokowania. Algorytm znaczników czasowych, w swojej podstawowej wersji, jest wolny od zakleszczeń.

Algorytmy optymistyczne- podstawowym założeniem metod optymistycznych jest założenie, że konflikty pomiędzy transakcjami występują stosunkowo rzadko, stąd algorytm powinien być maksymalnie mało restryktywny, jeżeli chodzi o ograniczenie dostępu do danych dla współbieżnie wykonywanych transakcji. Wykonywanie transakcji w metodzie optymistycznej przebiega w trzech fazach: odczytu, walidacji, zapisu. Transakcje, które nie przejdą walidacji uszeregowalności zostają wycofane i restartowane.

# MODEL ISO OSI

Model odniesienia ISO/OSI przedstawia proces komunikacji w postaci siedmiu warstw. Każda warstwa odpowiada konkretnemu fragmentowi procesu komunikacji, który sam w sobie stanowi zamkniętą całość. Dla każdej warstwy zdefiniowano interfejsy do warstw sąsiednich. Przy użyciu tego modelu można wyjaśnić, w jaki sposób pakiet przechodzi przez różne warstwy do innego urządzenia w sieci, nawet jeśli nadawca i odbiorca dysponują różnymi typami medium sieciowego.

Warstwy modelu ISO OSI:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 7 | Warstwa aplikacji |  | HTTP, HTTPS, IMAP, POP3, SSH, DNS | ↓ |
| 6 | Warstwa prezentacji |  | SSL, TLS |
| 5 | Warstwa sesji |  | L2TP, SAP, NetBIOS |
| 4 | Warstwa transportowa | Segment | UDP, TCP |
| 3 | Warstwa sieci | Pakiet | IPv4, IPv6, Apple Talk |
| 2 | Warstwa łącza danych | Ramka | Frame Relay, ARP |
| 1 | Warstwa fizyczna | Bit | Ethernet, ADSL, RS-232 |

Wędrówce danych między warstwami modelu odniesienia towarzyszy proces enkapsulacji (opakowania) jeżeli dane przekazywane są w dół stosu oraz proces dekapsulacji (rozpakowania), gdy dane przekazywane są w kierunku przeciwnym.

# ADRESACJA IP

IPv4

Adres w wersji 4 składa się z 32 bitów, podzielony jest na 4 oktety. Każdy z oktetów daje 256 kombinacji, w wyniku czego adres zapisujemy jako cztery liczby posiadające wartość od 0 do 255 oddzielone kropkami.

W przypadku adresacji IP adres składa się z części bitów przeznaczonych na identyfikację sieci, do której został przypisany dany interfejs hosta oraz z pozostałej liczby bitów przeznaczonych na adresację hosta w danej sieci.

W przypadku IPv4 część adresu przeznaczona na identyfikator sieci jest zależna od długości maski sieciowej. Maska ta służy do wyznaczania adresu sieciowego, który jest (musi być) taki sam dla wszystkich interfejsów znajdujących się w tej samej podsieci. Maska, podobnie jak adres IPv4, składa się z 32 bitów. Bity na najbardziej znaczących pozycjach powinny być ustawione na 1. Liczba tych jedynek pogrupowanych w oktety decyduje o tym, ile bitów z adresu będzie odpowiadało za identyfikację sieci.

Wyróżniamy podział na adresację klasową i bezklasową. W przypadku notacji klasowej numery IP jak i maski mają ściśle określone zakresy. W przypadku adresacji bezklasowej dozwolonym numerom IPv4 mogą być przypisane dowolne (dozwolone) maski.

Adresacja klasowa: podział na poszczególne klasy adresów wynikał z próby optymalnego (na ówczesne czasy) przydzielania adresów. Stąd wyłoniono 5 klas adresów: A, B, C, D, E.

Zasady adresowania IPv4:

* pierwszą liczbą identyfikatora nie może być 127 (zarezerwowane dla pętli zwrotnej)
* identyfikator hosta nie może składać się z samych 225 ani samych 0
* identyfikator hosta musi być unikalny w sieci

Protokół IPv4 został zaprojektowany na początku lat 80-tych XXw. W tamtym czasie spełniał on w wystarczającym stopniu wymagania co do liczby adresów niezbędnych do obsłużenia połączonych w sieci urządzeń. Jednak wraz z rozwojem sieci komputerowych wzrasta zapotrzebowanie na adresy IP.

IPv6

Jedną z najważniejszych i bardzo istotną zmianą IPv6 w stosunku do IPv4 jest przeznaczenie większej liczby bitów na określenie adresu. W przypadku adresów IPv4 tylko 32 bity były przeznaczone na adres. Protokół IPv6 na adres przeznacza 128 bitów. Adres IPv6 zapisywany jest postaci heksadecymalnej. Preferowany jest zapis, w którym co 16 bitów (4 cyfry heksadecymalne) wstawiany jest separator w postaci dwukropka. Przykładem takiego adresu może być: 0432:5678:abcd:00ef:0000:0000:1234:4321.

* Zwiększenie długości adresu do 128 bitów
* Uproszczenie nagłówka protokołu
* Wprowadzenie wsparcia dla klas usług, uwierzytelniania oraz spójności danych
* Brak kompatybilności z IPv4 (Ze względu na fakt, że spora część ruchu odbywa się w dalszym ciągu w oparciu o IPv4 pakiety IPv6 są tunelowane wewnątrz IPv4)
* Brak fragmentacji na routerach
* Zakres adresu (obszar widoczności) jest ograniczony przez odpowiedni prefiks
* 3 typy adresów IPv6:
  + Unicast – pojedynczy interfejs, pakiety są kierowane tylko do odbiorcy
  + Multicast – grupa interfejsów, pakiety kierowane są do grupy odbiorców
  + Anycast – grupa interfejsów, pakiety dostarczane są tylko do najbliższego węzła

# JĘZYK UML

Język UML (*Unified Modelling Language*) – system wizualizacji, specyfikowania oraz dokumentowania składników systemów informatycznych.

Elementami UML są:

* Notacja: elementy graficzne, składnia języka
* Metamodel: semantyka, definicje pojęć języka i powiązania między nimi

Diagramy struktur:

* Diagram klas
* Diagram obiektów
* Diagram komponentów
* Diagram wdrożenia
* Diagram struktur złożonych (UML 2.0)
* Diagram pakietów (UML 2.0)
* Diagram profili (UML 2.2)

Diagramy zachowań:

* Diagram czynności
* Diagram przypadków użycia
* Diagram maszyny stanów
* Diagram interakcji (komunikacji, sekwencji, czasowy, przeglądu interakcji)

Diagram przypadków użycia (ang. *use case diagram*) przedstawia usługi, które system świadczy aktorom, lecz bez wskazywania konkretnych rozwiązań technicznych. Diagram przypadków użycia wymienia użytkowników systemu i jednostki zewnętrzne, przedstawia funkcje dostępne dla użytkowników oraz określa powiązania i zależności między nimi.

* Aktor – zbiór ról odgrywanych przez użytkowników systemu
* Przypadek użycia – specyfikacja ciągu akcji i ich wariantów, które system może wykonać
* Związek (asocjacja, uogólnienie, rozszerzenie, zawieranie) – semantyczne powiązanie pomiędzy elementami modelu

Diagram sekwencji opisuje zależności przy przesyłaniu komunikatów pomiędzy obiektami, czyli zależności w przepływie sterowania pomiędzy obiektami. Diagram sekwencji przedstawia zależności czasowe pomiędzy obiektami. Metody realizujące sterowanie są rozproszone w wielu klasach, co powoduje trudności ze zrozumieniem ich wzajemnej zależności i interakcji. Diagramy sekwencji są tworzone, aby pomóc je zrozumieć. Na diagramie sekwencji przedstawione jest zazwyczaj zachowanie systemu dotyczące jednego przypadku użycia.

Diagram sekwencji składa się z pionowych linii życia poszczególnych obiektów uczestniczących w interakcji oraz wymienianych pomiędzy nimi komunikatów, reprezentowanych przez poziome strzałki opisane nazwą komunikatu.

* Wywołanie synchroniczne – wypełniona strzałka
* Wywołanie asynchroniczne – ‘pusta’ strzałka
* Powrót z wywołania – strzałka przerywana

# KOMPILACJA, KONSOLIDACJA, BIBLIOTEKI

Kompilacjato proces automatycznego tłumaczenia kodu napisanego w jednym języku (źródłowym) na równoważny kod w innym języku (wynikowym). Proces ten jest przeprowadzany przez program zwany kompilatorem. Najczęściej kompilacja dotyczy tłumaczenia na język maszynowy.

Każda kompilacja jest procesem dwuetapowym:

1. Analiza – program źródłowy rozkładany jest na części składowe i generowana jest jego reprezentacja pośrednia. Ważnym elementem tego etapu kompilacji jest wykrywanie i zgłaszanie użytkownikowi informacji o ewentualnych błędach w programie źródłowym. Etap analizy obejmuje:
   1. Analizę leksykalną
   2. Analizę składniową
   3. Analizę semantyczną
2. Synteza – na podstawie reprezentacji pośredniej generowany jest program wynikowy. Etap syntezy obejmuje:
   1. Generowanie kodu pośredniego
   2. Optymalizację kodu
   3. Generowanie kodu wynikowego

Konsolidacjato proces, w którym łączy się wynikowe moduły przemieszczalne, powstałe na etapie kompilacji. Wyróżniamy konsolidację statyczną (na etapie tworzenia programu wykonywalnego) i dynamiczną (w czasie ładowania lub wykonania).

Konsolidacja statyczna – odniesienia do innych modułów zamieniane są na adresy w czasie tworzenia programu wykonywalnego.

Konsolidacja dynamiczna:

* W czasie ładowania – następuje załadowanie modułów bibliotecznych i związanie odpowiednich adresów
* W czasie wykonania – załadowanie modułów bibliotecznych i związanie adresów następuje dopiero przy odwołaniu się do nich w czasie wykonywania programu

Biblioteki statyczne – w całości włączane do kodu wynikowego kompilowanych z nimi programów, zwiększając tym samym objętość plików wynikowych.

Biblioteki dynamiczne – nazywane również bibliotekami współdzielonymi lub obiektami współdzielonymi, zapisywane są w plikach wykonywalnych. Biblioteki dynamiczne nie muszą być łączone z programem podczas kompilacji. Program może sam je załadować podczas wykonania. Takie rozwiązanie umożliwia tworzenie tzw. wtyczek (ang. plug-ins).

# ALGORYTM RSA

Podpis cyfrowy – przekształcenie kryptograficzne danych umożliwiające odbiorcy danych sprawdzenie autentyczności i integralności danych oraz zapewniające nadawcy ochronę przed sfałszowaniem danych przez odbiorcę.

Do podpisu cyfrowego potrzebny jest klucz asymetryczny (tzw. para kluczy publiczny/prywatny). Klucz prywatny jest znany tylko użytkownikowi i jego poufność jest jednym z najważniejszych elementów bezpieczeństwa podpisu. Klucz publiczny jest jawny, z jego użyciem zostaje utworzony podpis cyfrowy.

Szyfrowanie asymetryczne – nazywane również szyframi z kluczem publicznym i prywatnym. W swoim działaniu wykorzystują dwa klucze, jeden do szyfrowania wiadomości, a drugi do deszyfrowania.

Klucz publiczny jest znany powszechnie i może być używany przez wszystkich zainteresowanych do szyfrowania dowolnie wybranych danych. Idea szyfrów asymetrycznych polega na tym, że tylko posiadacz drugiego klucza z pary - klucza prywatnego (który nie jest publicznie znany), może rozszyfrować takie dane. Podobnie, dane zaszyfrowane za pomocą klucza prywatnego, mogą być odszyfrowane tylko przy użyciu odpowiadającego mu klucza publicznego.

RSA jest obecnie jednym z najpopularniejszych algorytmów szyfrowania asymetrycznego. Nazwa pochodzi od pierwszych liter nazwisk jego twórców - Rona Rivesta, Adi Shamira i Leonarda Adlemana. Bezpieczeństwo szyfrowania RSA opiera się na zagadnieniach faktoryzacji dużych liczb złożonych.

Algorytm RSA składa się z trzech podstawowych kroków:

1. Generacja klucza publicznego i tajnego. Klucz publiczny jest przekazywany wszystkim zainteresowanym i umożliwia zaszyfrowanie danych. Klucz tajny umożliwia rozszyfrowanie danych zakodowanych kluczem publicznym. Jest trzymany w ścisłej tajemnicy.
2. Użytkownik po otrzymaniu klucza publicznego, np. poprzez sieć Internet, koduje za jego pomocą swoje dane i przesyła je w postaci szyfru RSA do adresata dysponującego kluczem tajnym, np. do banku, firmy komercyjnej, tajnych służb. Klucz publiczny nie musi być chroniony, ponieważ nie umożliwia on rozszyfrowania informacji – proces szyfrowania nie jest odwracalny przy pomocy tego klucza. Zatem nie ma potrzeby jego ochrony i może on być powierzany wszystkim zainteresowanym bez ryzyka złamania kodu.
3. Adresat po otrzymaniu zaszyfrowanej wiadomości rozszyfrowuje ją za pomocą klucza tajnego.

Generowanie kluczy RSA:

1. Wybieramy dwie duże liczby pierwsze: ,
2. Obliczamy ich iloczyn
3. Obliczamy wartość funkcji Eulera
4. Wybieramy losowo liczbę , względnie pierwszą z wartością funkcji Eulera.
5. Korzystając z rozszerzonego algorytmu Euklidesa znajdujemy liczbę taką, że:

Liczby i są także względnie pierwsze.

Szyfrowanie dla bloku tekstu jawnego i zaszyfrowanego ma następującą postać:

Liczby stanowią klucz publiczny, który ujawniamy, zaś liczby stanowią klucz prywatny.

Certyfikat cyfrowy - cyfrowy dokument, który pozwala zweryfikować, czy podpis elektroniczny należy do konkretnej osoby oraz potwierdza tożsamość instytucji lub osoby używającej określonego podpisu elektronicznego. Certyfikaty znajdują zastosowanie wszędzie tam, gdzie niezbędne jest potwierdzenie tożsamości instytucji lub osoby, a więc na przykład internetowe oddziały banków czy sklepy internetowe przyjmujące płatności kartami kredytowymi. Certyfikat pozwala sprawdzić, czy transakcję dokonujemy na właściwej stronie WWW i czy jest ona obsługiwana przez właściwą firmę. Zwiększa to wiarygodność operacji finansowych dokonywanych w Internecie. Certyfikaty cyfrowe wykorzystuje się także w poczcie e-mail. Dołączane do wiadomości e-mail pozwalają sprawdzić, czy wiadomość nie jest fałszywa lub zmodyfikowana. Certyfikaty cyfrowe wydawane są przez specjalne urzędy certyfikacji.

# METODY PROGRAMOWANIA

Metoda „dziel i zwyciężaj” – polega na podziale rozwiązywanego problemu na mniejsze podproblemy, które rozwiązujemy rekursywnie, a następnie scalamy, aby utworzyć rozwiązanie do początkowego problemu.

Fazy metody „dziel i zwyciężaj”:

1. Dziel – podział na mniejsze problemy, łatwiejsze w rozwiązaniu
2. Zwyciężaj – rozwiąż podproblemy, ewentualnie używając rekursji
3. Łącz – połącz rozwiązania poszczególnych podproblemów w rozwiązanie oryginalnego problemu

Typowym przykładem zastosowania metody „dziel i zwyciężaj” jest sortowanie przez scalanie.

Zalety:

* Proste rozwiązanie skomplikowanych problemów
* Dostosowane do obliczeń na wielu wątkach
* Wydajne, gdy podproblemy nie powtarzają się

Wady

* Narzut pamięciowy wywołany rekursją
* Niewydajne, gdy podproblemy powtarzają się i są rozwiązywane ciągle na nowo

Metoda zachłanna – algorytm na każdym kroku podejmuje decyzję lokalnie optymalną, tj. najbardziej opłacalną w danym momencie, nie uwzględniając dalszych skutków.

Algorytmy zachłanne zwykle nie znajdują rozwiązania optymalnego globalnie, ale są proste w implementacji i mogą przybliżyć rozwiązanie optymalne globalnie w zadowalającym stopniu.

Algorytmy zachłanne nie są dobrym wyborem przy rozwiązywaniu problemów optymalizacyjnych, w których kolejność podejmowanych wyborów ma znaczenie. Istnieją również problemy, dla których algorytm zachłanny nie tylko nie znajdzie rozwiązania optymalnego globalnie, ale znajdzie rozwiązanie najgorsze z możliwych. Przykładem takiego problemu jest problem komiwojażera.

Typowym przykładem algorytmu zachłannego jest algorytm Dijkstry służący do znajdowania najkrótszej ścieżki w grafie.

Programowanie dynamiczne – jeśli problem można rozbić na podproblemy i liczba wszystkich potencjalnych podproblemów jest wielomianowa to zamiast korzystać z rekursji możemy obliczyć wartości wszystkich podproblemów stosując kolejność od „najmniejszych” do „największych”. Programowanie dynamiczne ma zastosowanie, gdy podproblemy na siebie „zachodzą”, tzn. zawierają takie same podproblemy (własność optymalnej podstruktury).

Algorytmy programowania dynamicznego mają zastosowanie np. w problemach plecakowych (*Knapsack problem*) lub problemie komiwojażera.

# WZORCE PROJEKTOWE

Wzorzec projektowy to uniwersalne, sprawdzone w praktyce rozwiązanie często pojawiających się, powtarzalnych problemów projektowych. okazuje powiązania i zależności pomiędzy klasami oraz obiektami i ułatwia tworzenie, modyfikację oraz pielęgnację kodu źródłowego. Wzorce projektowe stosowane są w projektach wykorzystujących programowanie obiektowe. Zamiast skupiać się na funkcjonowaniu poszczególnych elementów, wzorce projektowe stanowią abstrakcyjny opis zależności pomiędzy klasami.

Podział wzorców projektowych:

* wzorce kreacyjne – pozwalają ująć w abstrakcyjnej formie proces tworzenia egzemplarzy klas, pomagają zachować niezależność systemu od sposobu tworzenia, składania i reprezentowania obiektów. Klasowe wzorce konstrukcyjne są oparte na dziedziczeniu i służą do modyfikowania klas, których egzemplarze są tworzone. W obiektowych wzorcach konstrukcyjnych tworzenie egzemplarzy jest delegowane do innego obiektu.

Wzorce konstrukcyjne kapsułkują informacje o tym, z których klas konkretnych korzysta system oraz ukrywają proces tworzenia i składania egzemplarzy tych klas. System zna tylko interfejsy obiektów zdefiniowane w klasach abstrakcyjnych.

Przykłady: Singleton, Metoda wytwórcza, Prototyp

* wzorce strukturalne dotyczą składania klas i obiektów w większe struktury.

Przykłady: Kompozyt, Pyłek, Adapter

* wzorce behawioralne – opis algorytmów, podział odpowiedzialności pomiędzy obiekty oraz charakterystyka interakcji pomiędzy nimi

Przykłady:

* SINGLETON – wzorzec konstrukcyjny – gwarantuje, że klasa będzie miała tylko jeden egzemplarz i zapewnia globalny dostęp do niego.

Singleton definiuje operację Instance umożliwiającą klientom dostęp do niepowtarzalnego egzemplarza klasy. Instance to operacja statyczna. Singleton może odpowiadać za tworzenie własnego niepowtarzalnego egzemplarza. Klienty mogą uzyskać dostęp do egzemplarza klasy Singleton wyłącznie poprzez operację Instance z tej klasy.

We wzorcu Singleton jedyny egzemplarz jest zwykłym egzemplarzem klasy, jednak jest ona napisana tak, aby można było utworzyć tylko ten egzemplarz. Standardowe rozwiązanie polega na ukryciu operacji tworzącej egzemplarz w operacji statycznej, która gwarantuje, że może powstać tylko jeden egzemplarz danej klasy. Ta operacja ma dostęp do zmiennej przechowującej ów egzemplarz, a zanim zwróci jej wartość, upewnia się, że zmienna została zainicjowana za pomocą niepowtarzalnego egzemplarza. To podejście gwarantuje, że singleton zostanie utworzony i zainicjowany przed jego pierwszym użyciem.

* ADAPTER – wzorzec strukturalny – przekształca interfejs klasy na inny, oczekiwany przez klienta. Adapter umożliwia współdziałanie klasom, które z uwagi na niezgodne interfejsy standardowo nie mogą współpracować ze sobą.

Wzorzec Adapter składa się z 4 podstawowych elementów (klas):

* Target – element docelowy – definiuje specyficzny dla dziedziny interfejs używany przez klienta
* Client – wspódziała z obiektami zgodnymi z interfejsem klasy Target
* Adaptee – element dostosowywany (adaptowany) – definiuje istniejący interfejs, który trzeba dostosować
* Adapter – dostosowuje interfejs klasy Adaptee do interfejsu klasy Target
* KOMPOZYT – wzorzec strukturalny – składa obiekty w struktury drzewiaste odzwierciedlające hierarchię typu całość-część. Wzorzec ten umożliwia klientom traktowanie poszczególnych obiektów i ich złożeń w taki sam sposób.

Kluczowym elementem wzorca Kompozyt jest klasa abstrakcyjna reprezentująca zarówno typy proste, jak i zawierające kontenery.

Elementami wzorca Kompozyt są:

* Component – obejmuje deklarację interfejsu składanych obiektów, implementację domyślnego zachowania na potrzeby interfejsu wspólnego dla wszystkich klas, interfejsu umożliwiającego dostęp do komponentów podrzędnych i zarządzania nimi
* Leaf – reprezentuje liście w złożeniu (obiekty bez elementów podrzędnych) i ich zachowanie
* Composite – definiuje zachowanie komponentów mających elementy podrzędne, przechowuje elementy podrzędne
* Client – manipuluje obiektami w złożeniu poprzez interfejs klasy Component
* PAMIĄTKA – wzorzec operacyjny – bez naruszania kapsułkowania rejestruje i zapisuje w zewnętrznej jednostce wewnętrzny stan obiektu, co umożliwia późniejsze przywrócenie obiektu do zapamiętanego stanu.

Elementami wzorca Pamiątka są:

* Memento – pamiątka – przechowuje wewnętrzny stan obiektu Originator. Pamiątka może obejmować dowolną określana przez źródło część jego wewnętrznego stanu. Dostęp do wewnętrznego stanu pamiątki ma tylko źródło, które ją utworzyło.
* Originator – źródło – tworzy pamiątkę obejmującą zapis wewnętrznego stanu źródła, korzysta z pamiątki do przywrócenia swojego stanu
* Caretaker – zarządca – odpowiada za zarządzanie pamiątką, nigdy nie manipuluje jej zawartością ani jej nie sprawdza