

# Softwarové technologie

Logický program .....	7
Predikáty .....	7
Syntaxe Prologu .....	8
Databáze .....	11
Databázový systém (DBS).....	11
Historický vývoj a modely dat .....	11
Systémy pro správu souborů .....	11
Hierarchické DBS .....	12
Síťové DBS .....	12
Relační DBS .....	12
Objektově relační DBS .....	12
Objektové DBS .....	13
Relační algebra .....	13
Základní operátory relační algebry.....	13
SQL .....	14
Základní příkazy .....	14
Parametry příkazu SELECT .....	14
Agregační funkce.....	14
Pohledy.....	14
Uložená procedura .....	14
Trigger .....	15
Transakce .....	15
Privilegia .....	15
Konceptuální modelování .....	16
Základní pojmy .....	16
Konceptuální modelování .....	16
ER model.....	16
Relační model .....	17
Transformace ER modelu do relačního modelu.....	17
Měřítko, zda je DBMS relační.....	18
Databázová relace .....	18
Normální formy relací .....	19
Znalosti.....	20

Schéma pro reprezentaci znalostí.....	20
Deklarativní reprezentace .....	20
Asociativní reprezentace poznatků .....	20
Procedurální reprezentace poznatků .....	20
Rámcová reprezentace poznatků .....	21
Životní cyklus znalostní aplikace.....	21
Tvorba znalostní aplikace .....	22
Expert a znalostní inženýr .....	22
Expert .....	22
Znalostní inženýr .....	22
Získávání znalostí a jejich uchovávání .....	23
Získávání znalostí .....	23
Uchovávání znalostí .....	23
Ontologické inženýrství.....	24
Formální ontologie .....	24
Komunikace člověk – člověk.....	24
Komunikace člověk – počítač .....	24
Komunikace počítač – počítač .....	25
Základní prvky ontologie.....	25
Typy ontologií.....	25
Dle expresivity jazyka a formálnosti .....	25
Dle specifičnosti .....	26
Reprezentace formálních ontologií .....	26
RDF .....	26
RDFS .....	27
OWL .....	27
Návrhové vzory .....	28
Rozklad hodnot .....	28
Pseudojedinec .....	28
N-ární vztah .....	28
Část – Celek.....	29
Normalizace ontologie .....	29
Primitivní třída .....	29
Přídavná třída .....	29
Definovaná třída .....	29
Odvozování.....	30

Nástroje .....	30
Aplikace ontologií .....	31
Sémantický web .....	32
Základní pojmy .....	32
Metadata .....	32
RDFS, OWL .....	33
Dotazování na sémantický web .....	33
Odvozování .....	34
Námětové mapy .....	34
Standard Topic Maps .....	34
Stavební prvky .....	35
Tvorba námětové mapy .....	35
Syntaxe .....	36
Dotazování na námětové mapy .....	36
Využití námětových map .....	37
Objektové modelování a programování .....	38
Vznik .....	38
Základní pojmy .....	38
Modelování .....	39
UML .....	41
Modelování tříd .....	42
Modelování objektových interakcí .....	42
Softwarový proces .....	43
Událostmi řízené programování .....	43
Architektura MVC .....	43
Obecný princip MVC .....	43
Práce s kolekcemi .....	45
Řídící algoritmy .....	45
Bubble sort .....	45
Insertion sort .....	45
Merge sort .....	45
Quick sort .....	45
Selection sort .....	45
Základní pojmy .....	46
Kontejnery (v Javě obvykle nazývané "kolekce") .....	46
Pole .....	46

Kolekce .....	46
Iterátor .....	46
Porovnatelnost .....	46
Kolekce .....	46
Seznamy .....	47
Množiny .....	47
Mapy .....	48
Problematika perzistovaného ukládání dat ve vybraném programovacím jazyce .....	49
Trvalá data .....	49
Vstupy a výstupy .....	49
Vstupní streamy .....	49
Výstupní streamy .....	49
Vstupní proudy .....	50
Výstupní proudy .....	50
Databáze .....	51
JDBC API .....	51
Objektově relační mapování .....	52
Objektové databáze .....	53
Java serializace .....	53
Webové aplikace .....	54
Principy .....	54
Technologie webových aplikací .....	54
PHP (personal home page – hypertext preprocessor) .....	54
JSP .....	55
ASP.NET .....	55
HTML .....	56
CSS .....	56
Třívrstvá architektura .....	56
Testování a vývoj .....	57
Zabezpečení .....	57
Validace (a escapování) .....	57
Cross-site request forgery (CSRF) .....	58
Šifrování přenášených dat .....	58
Ověření uživatele .....	58
Kontrola cookies .....	58
Základní algoritmy a principy počítačové grafiky .....	59

Reprezentace obrazu .....	59
Rasterizace úsečky .....	59
Metody vyplnění oblasti .....	60
Zobrazovací řetězec .....	60
Určení viditelnosti .....	61
Určení viditelnosti .....	61
OpenGL.....	62
Základní zpracování obrazu.....	64
Snímání.....	64
Předzpracování.....	64
Segmentace .....	64
Klasifikace.....	65
Formáty pro ukládání rastrového obrazu .....	66
GIF (Graphics Interchange Format) .....	66
PNG (Portable Network Graphics) .....	66
JPEG File Interchange Format.....	67
TIFF (Tag Image File Format) .....	67
Komprese .....	67
Bezztrátová komprese .....	67
Ztrátová komprese.....	67
Barevné modely.....	68
Aditivní barevný model .....	68
Subtraktivní barevný model.....	68
HSV .....	68
Algoritmy pracující s grafy .....	69
Základní pojmy .....	69
Využití grafů – stromů .....	69
Nalezení minimální kostry .....	70
Obecný algoritmus .....	70
Jarníkův algoritmus.....	71
Kruskalův algoritmus .....	71
Využití grafů při řešení problémů – úloh .....	72
Algoritmy prohledávání .....	72
Zásobník .....	73
Fronta (FIFO, "first in first out") .....	73
Ilustrace algoritmu prohledávání do hloubky .....	73

Ilustrace algoritmu prohledávání do šířky .....	74
---	----

# Logický program

- Jazyk pro programování symbolických výpočtů
- Založen na predikátové logice
  - $\forall x(\text{má\_réd}(x, \text{pivo}) \Rightarrow \text{má\_réd}(\text{Honza}, x))$
  - Prolog:  $\text{má\_réd}(\text{Honza}, X) \text{ :- } \text{má\_réd}(X, \text{pivo})$
- Název odvozen z PROgramování v LOGice
- Jeho úspěch pro vznik nové disciplíny matematické informatiky => logického programování
  - Specifické oblasti použití
  - Umělá inteligence
  - Znalostní inženýrství
  - Databázové a expertní systémy
  - Podpora specializovaných činností, např. projektování (CAD), výuce (CAI) aj.
- Porovnání s konvenčními jazyky
  - Jde o neprocedurální jazyk
  - Lze se více soustředit na popis vlastností relací – tedy co se má vypočítat
  - Bez nutnosti řešit, jak se to má udělat
  - Deklarativní
    - Při psaní programu deklarujeme fakta a pravidla pro popis vlastností a vztahů
  - Konverzační
    - Uživatel klade dotazy
    - Prolog odpovídá
  - Interaktivní
    - Pokud uživatel dotazy neklade, Prolog nepracuje a čeká
- V Prologu nejsou příkazy pro řízení běhu výpočtu ani příkazy pro řízení toku dat
- Chybějí prostředky pro programování cyklů, větvení, ...
- Nepoužívá se přiřazovací příkaz
- Rozdílná úloha proměnných
  - Proměnná v Prologu označuje po dobu výpočtu objekt, který vyhovuje určitým podmínkám
  - Jeho vymezení se při výpočtu upřesňuje
- Průběh výpočtu v Prologu řízen jeho interpretem na základě znění programu
- Programátor může chod výpočtu ovlivnit řídicími příkazy v mnohem menší míře než u jiných jazyků
- Původně navržen jako specializovaný na symbolické výpočty, moderní implementace směřují k obecnějšímu použití

## Predikáty

- Vyjadřují fakta, která mají význam vztahů mezi objekty
- Píše se stejně jako funktor
- 2 základní režimy
  - Konzultační režim
    - Slouží k samotnému vytváření programu

- Dotazovací režim
  - Kladou se otázky pomocí predikátů
- Otázka vždy začíná otazníkem
- Další výsledek se prolog pokusí najít po zadání středníku, stisknutím samotného enteru prolog ukončí dotazování

## Syntaxe Prologu

- Prolog se skládá z termů
- Jako termy se označují konstanty, proměnné, struktury
- Konstanty
  - Integer
    - Celá čísla
    - Rozsah záleží na implementaci
    - Většina moderních implementací i s reálnými
  - Atomy
    - Sekvence znaků začínající malým písmenem
    - Nebo řetězec uzavřený v apostrofech
    - Speciální atomy: ? – :- ! .
- Proměnné
  - Sekvence znaků začínající velkými písmeny nebo podtržítkem
  - Samotné podtržítko = anonymní proměnná
    - Její hodnota není podstatná
    - Nebude dále využívána
- Struktury
  - Objekt skládající se z jiných objektů
  - Komponenty lze spojovat pomocí funktorů
- Term je zapsán jako sekvence znaků
  - Malá a velká písmena
  - Číslice
  - Speciální znaky
- Vestavěné predikáty
  - Procedury nabízené danou implementací Prologu
  - Velké množství predikátů nejen pro výpis na obrazovku a také získávání dat od uživatele, ale také na klasifikaci a konverzi termů, groupování a ladící příkazy
  - Jsou považovány za termy
- Konstanty a proměnné se označují jako atomické termy
  - Z atomických termů se dají vytvářet složitější výrazy = struktury
- Programování v Prologu spočívá
  - V deklarování faktů o objektech a relacích mezi nimi
    - Fakta
      - Nepodmíněné výrazy, vyjadřují skutečnost o relaci
      - Jde o vztahy, které mají jméno, za kterým následuje v kulatých závorkách výčet objektů, kterých se vztah týká
      - Každý fakt je ukončen tečkou
      - matka(Matka, Dite).
  - Definování pravidel
    - Podmíněné výrazy



- Dvě části, oddělené :-
  - V levé části je hlava pravidla, v pravé tělo pravidla
    - sourozenci(X,Y) :- matka(M,X), matka(M,Y)
- Kladení a zodpovídání dotazů
  - Začíná dvojicí znaků ?-, končí tečkou
  - Dotaz, který neobsahuje proměnné se nazývá základní otázka
  - Výsledkem je odpověď
    - Výpis hodnot proměnných, které jsou součástí otázky yes/no
  - No může znamenat
    - Zápornou odpověď
    - Neexistující odpověď
- Seznamy
  - Prakticky jedinou předdefinovanou datovou strukturou
  - Tvořen posloupností prvků
    - Oddělených čárkami
    - A uzavřených do hranatých závorek
  - Vytváření
    - Funktor .
  - Seznam se skládá z hlavy a těla
    - Výjimkou je prázdný seznam
  - Syntaktická pomůcka
    - Svislítkem lze oddělit hlavu od těla
    - [ H | T ]
  - Operace se seznamy
    - Append – spojování
    - Member – zjistí, zda je prvek členem
    - Length – délka seznamu
    - Delete – odstraní prvek ze seznamu
  - Seznam je rekurzivně definovaná struktura
    - I procedury s ním pracující jsou rekurzivní
- Databáze
  - Můžeme s výhodou použít tam, kde potřebujeme aktivně přistupovat k bázi znalostí
  - Největší předností je možnost měnit program během jeho běhu
- Řízení databáze
  - Assert, asserta, assertz – přidá do DB
  - Retract, retractall – odstraní z DB
  - Listing – výpis klauzulí v DB
  - Clause – hledá klauzule
- Řízení průchodu programem
  - ! (řez)
    - Představuje klíčové rozhodnutí
    - Překročí-li jej, už se nevrací
    - Červený
      - Pokud jej z programu smažu, bude program pracovat chybně
    - Zelený

- Pokud jej smažu, může dojít ke snížení efektivity, ale funkce se nezmění
- Fail
  - Nikdy nesplněný predikát
  - Nelze se přes něj dostat
- Repeat
  - Při průchodu programem blokuje návrat
  - Nekonečný cyklus
    - Repeat
    - Write('Pracuj'), nl,
    - Fail.

# Databáze

- Báze dat
- Všechna potřebná data dané organizace

## Databázový systém (DBS)

- Specializovaný SW pro efektivní práci s daty
- DBS umožňuje shromažďovat různé informace
- Tyto informace ukládá a udržuje v platném stavu na centrálním místě
- Tvořen několika částmi
  - DBS = SŘBD + DBA + DB
  - SŘBD
    - Systém řízení báze dat
    - Program, jehož úkolem je pracovat s uloženými informacemi
    - Organizuje je a udržuje v platném stavu
  - DBA
    - Databázová aplikace
    - Program pro manipulaci s daty prostřednictvím SŘBD
  - DB
    - Databáze, množina informací
- Jednotlivé části DBS se mohou provozovat na jednom PC nebo na různých
- Historicky byly často SŘBD a DBA spojeny do jednoho programu
- V dnešní době nejpoužívanější způsoby zpracování dat databázové systémy klient/server
  - SŘBD a DBA od sebe odděleny
- Hlavní funkce
  - Definice DB
  - Efektivní manipulace DB
  - Souběžný přístup
  - Ochrana dat
  - Zotavení z chyb

## Historický vývoj a modely dat

- Databázový model je popisem fungování DB
- Hlavním úkolem je popsat, jakým způsobem jsou data zpřístupňována uživatelům
- Uživatelé se data jeví jiným způsobem, než jak jsou ve skutečnosti uložena
- Využívá se abstrakce

## Systémy pro správu souborů

- Nejjednodušší databázový model
- Jako jediný přímo popisuje způsob uložení dat na disku
- Všechny položky ukládány sekvenčně do jednoho souboru
- Pokud je třeba najít informaci, nutno prohledat celý soubor od začátku
- Výhodou je jednoduchost
- Nevýhody
  - Mezi uloženými údaji nejsou vztahy
  - Problémy s integritou – žádná kontrola správnosti ukládaných dat

- Pomalé vyhledávání informace
- Obtížné změny struktury

## Hierarchické DBS

- Organizuje data do stromové struktury
- Vždy jeden kořenový uzel, jehož vlastník je SŘBD
- Z kořene vedou ukazatele k uzlům 1. úrovně v nichž začíná vlastní DB
- Každý uzel první úrovně může mít několik synovských uzlů
- Jednotlivé položky jsou umístěny na různých větvích
- Na fyzické struktuře na disku nezáleží, tento úkol řeší SŘBD
- Z diagramu je obtížné určit, z jakých položek se skládá záznam
- Velmi obtížná změna struktury DB
  - V tomto případě je nutné vytvořit novou strukturu a překopírovat do ní údaje
- Umožňuje definovat vztahy 1:N
- Efektivnější vyhledávání informace
  - Prohledávají se příslušné části stromu
- Neumožňuje vztahy M:N

## Síťové DBS

- Popisuje DB, v nichž existují vztahy M:N
- Je založena na ukazatelích, které vyjadřují vztahy mezi jednotlivými položkami
- Vzájemné vztahy mezi množinami mohou být velmi komplikované
- Obdobné nevýhody se změnou struktury jako u Hierarchické DBS

## Relační DBS

- Z roku 1969, kdy E. F. Codd popsal model založený na matematickém pojmu relace
- Nevyužívají se rodičovské vztahy mezi položkami
- Data jsou organizována do uspořádaných n-tic
- Jednotlivé n-tice představují řádky tabulky
- Každý záznam je řádkem tabulky a každá položka sloupce
- Při definování schématu je využíván proces normalizace, při němž dochází k dekompozici dat do podmnožin pro jednotlivé tabulky
- Výhody
  - Realizuje vztahy M:N
  - Snadná změna schématu
  - Změna struktury spočívá v přidání/odebrání sloupce
  - Zachování integrity
  - Pokud je SŘBD skutečně relační musí znemožnit přístup k datům jinými kanály než prostřednictvím sebe sama
  - Programy pracují s daty, aniž by znaly jejich umístění v DB
  - K veškeré manipulaci s daty využití služeb SŘBD

## Objektově relační DBS

- Pokus integrovat objekty a relace do jednoho systému
- Tři možné přístupy
  - Objektově relační datové manažery manipulující objekty do relačních tabulek

- Relační obálky tvořené knihovnamí, které se přilinkují k relační DB. Relační obálka detekuje změny objektů a promítá je pomocí SQL do relační DB a změny v relační DB promítá do objektů
- Objektově relační DB ukládající data pomocí relací i jako objekty

## Objektové DBS

- Systém vybudovaný pomocí objektově orientovaných metod
- Každá komponenta zapouzdřuje data a funkce
- Komponenty dědí atributy a chování z jiných komponent a navzájem komunikují pomocí zpráv
- První pokusy vycházely z existence objektově orientovaných programovacích jazyků

## Relační algebra

### Základní operátory relační algebry

- Kartézský součin
- Sjednocení
- Průnik
- Rozdíl
- Projekce
- Selekce
- Spojení
  - Přirozené spojení
  - Levé polo-spojení
  - Pravé polo-spojení

#### Projekce

- Tabulky definované podmnožinou atributů představuje vypuštění některých sloupců neobsažených v požadované množině atributů
- Součástí projekce je i případné vypouštění duplicitních řádek ve výsledné tabulce
- Tabulku lze také rozšířit o sloupec vzniklý nějakou operací nad hodnotami uložených atributů

#### Selekce

- Tabulky definované podmnožinou definičního oboru relace představuje průnik relace novou podmnožinou

#### Spojení

- Vytvoření nové spojené tabulky na základě shodných hodnot atributů v obou tabulkách
- Podle způsobu porovnávání hodnot ve spojovaných sloupcích se rozlišuje na
  - Rovnost
  - Nerovnost – nemá praktický význam
  - Vnější
- Pokud výsledný operace obsahuje všechny sloupce z první i z druhé tabulky, vyjma sloupce, který byl využit pro spojení, pak toto spojení je označováno jako přirozené

## SQL

- Strukturovaný dotazovací jazyk, který byl určen jako standard pro komunikaci s relačními databázemi
- Nezávislý na datech
- Neprocedurální jazyk
  - Popíšeme, jaká data chceme najít, ne jak se to má provést
- 2 hlavní komponenty
  - DDL – definuje strukturu tabulek
  - DML – vyhledává a manipuluje s daty

### Základní příkazy

- SELECT
- INSERT
- UPDATE
- DELETE

### Parametry příkazu SELECT

- FROM
- WHERE
- GROUP BY
- HAVING
- ORDER BY
- DISTINCT

### Agregační funkce

- COUNT
- SUM
- AVG
- MIN
- MAX

### Pohledy

- Objekt, který uživateli poskytuje data ve stejné podobě jako tabulka
- Obsahuje předpis, jakým způsobem mají být data získána z tabulek a jiných pohledů
- Virtuální relace, který se vytvoří v okamžiku vyžádání
- Výhody
  - Soustřeďují potřebná data pro uživatele
  - Skrývají složitost podkladových dat
  - Zjednodušují správu uživatelských oprávnění
  - Definují uspořádání dat pro export do jiných aplikací

### Uložená procedura

- Databázový objekt, který neobsahuje data, ale část programu, který se nad daty v databázi má vykonat

## Trigger

- Definuje činnosti, které se mají provést v případě definované události nad tabulkou
- Například při smazání nebo vložení dat

## Transakce

- Skupina, příkazů, která se navenek tváří jako jeden příkaz
- 3 příkazy
  - BEGIN
  - ROLLBACK
  - COMMIT

## Privilegia

- Akce, které uživatel může realizovat nad tabulkou nebo pohledem
- Mohou být omezena na vyjmenované databáze, tabulky nebo sloupce
- Vlastník databáze uděluje ostatním potřebná privilegia pomocí GRANT
- Odebírá pak pomocí REVOKE

# Konceptuální modelování

## Základní pojmy

- Podle ANSI/SPARC architektury lze rozdělit na tři úrovně návrhu
  - Externí hladina
    - Odpovídá pohledu jednotlivých skupin uživatelů
  - Konceptuální hladina
    - Celkový logický pohled na data
    - Její vytvoření = první krok databázového návrhu
  - Interní hladina
    - Popis dat na nižší úrovni
    - Interface pro operační systém

## Konceptuální modelování

- Nejčastěji se používá ER model a různé formy jeho rozšíření
- Návrh struktury dat na konceptuální úrovni
- Analýza požadavků a jejich zobrazení pomocí grafických prostředků
- Základní prvky
  - Entity
    - Objekty reálného světa
    - Schopny samostatné existence
    - Odlišný od ostatních objektů
  - Vztahy
  - Atributy
    - Vlastnost entity
    - Doména atributu – přípustné hodnoty
    - Jednoduchý atribut – dále nedělitelný
    - Složený atribut – dělitelný, př. Adresa
- Jeden z možných postupů je entitně relační modelování – ER
  - Různé notace
    - UML Class Diagram

## ER model

- Nástroj na popis konceptuálního schématu reality bez ohledu na to, jak bude daný model implementován v konkrétním DBMS
- Pracuje s
  - Typem entit
    - Množina objektů stejného typu
    - Objekt reálného světa, který je schopen nezávislé existence
  - Atributy
    - Vlastnosti entit
  - Vztahy
    - Propojení mezi entitami
  - Kardinalita vztahů
    - Vyjadřuje kolik entit jednoho typu může být ve vztahu s kolika entitami z druhého typu entit



- IS-A hierarchie
  - $A \text{ is a } B$ , když typ entit B je zobecněním typu entit A, každý typ entit A je speciálním případem B
- Klíči
  - Kandidátní klíč
    - Minimální množina atributů pro jednoznačnou identifikaci
  - Primární klíč
    - Kandidátní klíč, který byl vybrán k té identifikaci
- Rozšířený ER model = EER
  - Zavádí supertřidu / podtřidu
  - Existuje dědičnost vztahů
  - Pro grafické znázornění lze použít digram tříd z UML
  - Kromě prostých vztahů také kompozice a generalizace

## Relační model

- Základní struktura relace
  - Definována jako podmnožina kartézského součinu N množin
- Cílem zavedení byla
  - Disciplína v manipulaci s daty
  - Zabezpečení nezávislosti dat
  - Vyšší produktivita programování
- Databáze
  - Kolekce DB relací
    - Jsou reprezentované tabulkami
      - Všechny informace v tabulkách
  - Oproti matematické definici relace je databázová relace
    - Vybavena pomocnou tabulkou
    - Jménem relace, jmény atributů a definicí domén
    - Hodnoty jsou dále nedělitelné
  - Vlastnosti databázové tabulky
    - Jednoznačné jméno tabulky a jednotlivých sloupců
    - Všechny hodnoty v jednom sloupci jsou stejného typu
    - Na pořadí sloupců a řádků nezáleží
    - Nemá duplicitní řádky
    - Každá tabulka má primární klíč

## Transformace ER modelu do relačního modelu

- Zjednodušeně
  - ER model – s obrázky
  - Relační model – textově
- Typ entity = tabulka
  - Její sloupce odpovídají atributům
- Asociace
  - Tabulka, která obsahuje primární klíče účastníků vztahu
  - U 1:1 a 1:N není nutná nová tabulka
  - U 1:N stačí zahrnout cizí klíč na stranu N
  - 1:1 sloučit tabulky do jedné

- Kompozice
  - Do nové tabulky zahrnout klíč rodičovské tabulky

## Měřítko, zda je DBMS relační

### 1. Základní pravidla

- RDBMS musí být schopný manipulovat s daty pomocí operací relační algebry
- Má-li RDBMS jazyk nižší úrovně, tato nižší úroveň nemůže porušit pravidla integrity vyjádřené na vyšší úrovni jazyka
- Přístup do DB řídí RDBMS tak, aby integrita nemohla být porušena bez vědomí admina

### 2. Pravidla týkající se struktury dat

- RDBMS by měl podporovat
  - Relace, domény, primární klíče, cizí klíče
- Všechny informace jsou reprezentovány explicitně na logické úrovni
- Je-li pohled teoreticky upravitelný, měl by být i fakticky upravitelný

### 3. Pravidla týkající se integrity dat

- Jsou podporovány hodnoty NULL na reprezentaci chybějící informace
- Integritní omezení musí být definována v DDL jazyku a musí být uložitelné v systémovém katalogu
  - Centralizovaná kontrola integrity

### 4. Pravidla týkající se modifikace dat

- Ideální DBMS by měl podporovat 18 možností na manipulaci s daty
- Garantovaný přístup ke každé atomické hodnotě na základě jména tabulky, hodnoty primárního klíče a názvu sloupce
- Obsáhlý jazyk na manipulaci s daty, který by měl umožňovat
  - Definici dat
  - Definici pohledů
  - Modifikace dat
  - Integritní omezení
  - Autorizaci
- Transakce

### 5. Pravidla o nezávislosti dat

- Nezávislost dat od aplikace, která data používá
- Uživatelé i programátoři jsou izolováni od organizace dat na nižší úrovni

## Databázová relace

- Reprezentována tabulkou, která má následující vlastnosti:
  - Každá tabulka má jednoznačné jméno
  - Každý sloupec v tabulce má jednoznačné jméno
  - Všechny hodnoty daného sloupce musí být stejného typu
  - Nezáleží na pořadí sloupců
  - Nezáleží na pořadí řádků
  - Tabulka nemůže mít duplicitní řádky
  - Hodnoty jsou atomické
  - Každé tabulky musí mít primární klíče

- DB musí splňovat následující integritní omezení
  - Integritu entit
    - V databázové relaci hodnota primárního klíče nesmí být NULL
  - Referenční integritu
    - Obsahuje-li databázová relace cizí klíč, tak každá jeho hodnota musí být obsažena v rodičovské tabulce, anebo musí mít hodnotu NULL

## Normální formy relací

- K odstranění opakujících se skupin a nadbytečných informací slouží normalizace
- Sledují se 2 cíle
  - Zabránit ztrátě užitečné informace při zrušení řádku tabulky
  - Vyhnout se redundanci
- Normalizace probíhá ve 3 stupních
  - Odstranění nadbytečnosti
  - Mít zjednodušenou kontrolu integrity
  - Odolné vůči anomáliím při údržbě dat
- Proces normalizace tabulek
  - Formální technika pro návrh relačních DB tabulek
  - Postupná dekompozice původních tabulek na základě analýzy závislosti atributů
- 1 NF
  - Všechny hodnoty jsou atomické
- 2 NF
  - Splňuje požadavky 1. NF
  - Každý neklíčový atribut je plně závislý na primárním klíči
- 3 NF
  - Splňuje požadavky 2. NF
  - Všechny neklíčové atributy jsou vzájemně nezávislé
- Funkční závislost
  - Je daná relace  $R(A,B)$ , kde A, B mohou být složené atributy
  - B je funkčně závislý na A, když pro hodnotu A je jednoznačně daná hodnota B
  - B je plně funkčně závislý na A, je-li funkčně závislý na A a není funkčně závislý na žádné podmnožině A

# Znalosti

## Schéma pro reprezentaci znalostí

### Deklarativní reprezentace

- Hlavní oblasti aplikace jsou produkční systémy
- Důraz je kladen na vyjádřitelnost
- Produkční pravidlo reprezentuje část poznatku ve formě – jestliže platí, tak platí
- Báze faktů
  - Obsahuje informace o aktuální situaci ve stavovém prostoru řešení problému
  - Informace vychází z dat, které byly odvozeny výpočtem, měřením, nebo nebyly zadány uživatelem
- Báze pravidel
  - Obsahuje pravidla, která byla zadána znalostním inženýrem ve spolupráci s expertem
  - Pravidlo funguje na principu – předpoklad => důsledek
- Inferenční mechanismus
  - Jde o soubor programů zajišťující vlastní výpočty. Jeho úkolem je napodobit pochody b myslí experta
  - Jde o soubor programů zajišťující vlastní výpočty. Jeho úkolem je napodobit pochody b myslí experta
- Způsoby řízení
  - Zpětný režim (zpětné řetězení) - Postup od cíle (cílů) směrem k počátečnímu stavu.
  - Přímý režim (dopředné řetězení) - Aplikace produkčních pravidel od počátečního stavu k některému ze stavů cílových

### Asociativní reprezentace poznatků

- Je kladen důraz na to, aby se poznatky dostaly vzájemně do souvislosti
- Generují se tzv. asociativní (sémantické) sítě, které umožňují graficky reprezentovat relace mezi jednotlivými objekty reality
- Tvorba relací
  - Z vět v češtině lze pomocí predikátové logiky vyjádřit poznatky takto:
  - věta: Rostliny a živočichové jsou živé organismy.
  - $JE(x, \text{živočich}) \Rightarrow JE(x, \text{živý organismus})$
  - $JE(x, \text{rostlina}) \Rightarrow JE(x, \text{živý organismus})$

### Procedurální reprezentace poznatků

- Vyplývá z použitelnosti
- Postupy používání poznatků budou zachycené v podobě procedur
- Procedura je volaná cílem – zadám fakta a požaduji nová fakta (vede k deklarativnímu programování)
- Definuje se: „Co musím udělat, abych s danými předpoklady zjistil, zda daný objekt 'něco' je.“
- Nedeklaruje se, co se má vypočítat, ale jak toho dosáhnout

- Má-li procedura vstupy k dispozici, pak rozhodne. Nemá-li je, pak se ze vstupů formulují podcíle a systém pokračuje v řešení

## Rámcová reprezentace poznatků

- Rámec je speciální údajovou strukturou na reprezentaci stereotypních situací
- Dříve, než přijdeme do styku s nějakou konkrétní situací, máme už s ní spojeny nějaké očekávání (defaults) a vědomosti
- Rámec si můžeme představit jako formulář, tedy soubor rubrik, které jsou vyplněny položkami rozličných typů
- Údaje v rubrikách mohou být opět názvy jiných rámců, čím se jednotlivé rámce mohou spojovat do hierarchických sítí

## Životní cyklus znalostní aplikace

- Disciplinovaný postup, pořadí, segmentace na proveditelné aktivity nebo fáze
- Dobrá dokumentace projektu (možné změny, modifikace)
- Koordinace projektu, aby byl dokončen včas
- Průběžná kontrola v každé fázi cyklu

Fáze	Klíčové otázky	Výstup
Vyhodnotit existující infrastrukturu	Jaký je problém? Je tvorba systému opodstatněná? Je vytvoření systému proveditelné?	Vyjádření cílů Výkonnostní kritéria Strategický plán
Vytvořit tým pro znalostní management	Kdo bude členem týmu? Jak bude tým fungovat?	Standardizovaná procedura pro vytvoření systému
Zachycení znalostí	Jaké a čím znalosti budou zachyceny? Jak se bude při zachycování znalostí postupovat?	Získané znalosti
Tvorba ZA	Jak budou znalosti reprezentovány?	Návrh ZA HW/SW implementační detaily Plán testů Bezpečnostní, kontrolní a operační procedury
Verifikace a validace ZA	Jak spolehlivý je systém?	Kontrola funkčnosti systému experty Návody, tutoriály
Nasazení ZA	Co je právě v organizaci řešeno? Jak jednoduché je systém používat?	Uživatelsky přívětivý systém Výukový program

Řízení změn a motivačních programů	Poskytuje systém požadovaná řešení?	Spokojení uživatelé
Post-evaluace	Měl by systém být modifikován?	Spolehlivý a aktualizovaný systém

## Tvorba znalostní aplikace

1. Tvorba znalostního týmu
  - Minimální složení
    - Doménový expert (primární, sekundární)
    - Znalostní inženýr
  - Další možné osoby
    - Konzultant
    - Propagátor projektu („champion“)
    - Experti pro doplňující tematické okruhy
    - Uživatelé (beta testing)
2. Zachycení znalostí
  - Analýza textů – cíl: získat základní povědomí o problémové doméně, šetřit čas experta, neobtěžovat jej vysvětlováním základů
  - Konzultace s expertem/experty – Cíl: Získat znalosti experta v použitelné – strojově zpracovatelné – podobě
  - Tvorba základního popisu znalostní domény – Cíl: Vytvoření základního konceptuálního popisu znalostní domény. Jedná se o přípravný krok při tvorbě ZA.
3. Tvorba ZA
  - Tvorba znalostní aplikace základní úrovně – Cíl: Vytvoření funkčního prototypu ontologie, ujasnění vzájemných souvislostí mezi třídami
  - Vytvoření plnohodnotné ontologie – Cíl: Vytvoření ucelené, logicky správné ontologie, která rozsahem odpovídá zaměření projektu
4. Verifikace a validace ZA
  - Testování znalostní aplikace – Cíl: Ověřit funkčnost aplikace a její využitelnost v praxi
5. Nasazení ZA v organizaci – Cíl: Zajistit využívání ZA uživateli a ověřit si další správnost fungování ZA v organizaci.

## Expert a znalostní inženýr

### Expert

- Specialista v určité problémové oblasti
- Efektivní v řešení problémů spadajících do jeho oboru
- Nezbytný při vývoji každé rozsáhlejší znalostní aplikace
  - Získávání znalostí
  - Verifikace znalostní aplikace
  - Ručí za správnost (odvozování, struktura apod.)

### Znalostní inženýr

- Úkolem znalostního je tvorba znalostního systému

- Má-li znalostní inženýr navrhnout znalostní systém musí
  - Získat znalosti od odborníka(ů) z předmětné oblasti,
  - Zaznamenat znalosti ve vhodném tvaru, tj. zkonstruovat model znalostí,
  - Vložit znalosti do počítačového programu, který znalostní systém reprezentuje,
  - Ověřit programový systém, tj. ujistit se, že skutečně obsahuje vložené znalosti a reaguje stejným způsobem, jako by reagoval člověk a že reakce je adekvátní vloženým znalostem

## Získávání znalostí a jejich uchovávání

### Získávání znalostí

- Interview – nejčastěji používané
- Pozorování (on-site observation) - znalostní inženýr pozoruje experta při práci
- Brainstorming – v rámci diskuse jsou kreativním myšlením vytvořeny řešení problému, všechny jsou posuzovány rovnocenně
- Protokolární analýza – Experti jsou dotazováni na postup řešení – kroky jsou zapsány
- Konsenzuální rozhodování – jasná shoda na nejlepším řešení problému, obvykle po brainstormingu
- Repertory grid – Doménový expert klasifikuje a kategorizuje problémovou doménu, vytváří vlastní model. Cílem je najít skryté principy, které experti při uvažování aplikují
- Technika číselných skupin (Nominal Group Technique NGT) – ohodnocení řešení se sečte, nejlépe hodnocené je vybráno jako konečné
- Metoda Delphi – Skupinové rozhodnutí by mělo konvergovat k optimálnímu nebo správnému řešení
- Blackboarding – Skupina expertů spolupracuje na řešení problému, který je uveden na tabuli. Tabule slouží jako jejich pracovní prostor. Obsahuje popis problému a vstupní data. Každý expert může k řešení přispět vyřešením části problému na tabuli. Na tabuli jsou postupně přidávána dílčí řešení, až je problém vyřešen celý

### Uchovávání znalostí

- Znalosti by měly být odděleny od lidí. Díky tomu mají všichni k informacím přístup a firma nepřichází o znalosti v případě odchodu zaměstnance
- Za tímto účelem jsou vytvářeny znalostní systémy, nebo jsou alespoň znalosti uchovávány na jednom místě s různými úrovněmi přístupu

# Ontologické inženýrství

## Formální ontologie

- Popis sdílení dat nějaké domény
- Reprezentuje množinu tříd v doméně a vztahy mezi nimi
- Reprezentuje znalost nebo její část
- Etymologie
  - Z řeckého ontos (bytí, to, co jest) a logos (slovo, řeč, význam) \* Původně podoblast filosofie, která se zabývá bytím a podmínkami existence věcí okolo nás
  - \* Oblast zájmu - co existuje nebo by mohlo existovat v našem světě, jsoucno a jeho podstata.
- V kontextu informatiky (Sowa):
  - „Předmětem ontologie je studium kategorií věcí, které existují nebo mohou existovat v určité doméně. Výsledek tohoto studia, nazývaný ontologie, je katalog věcí, jejichž existenci předpokládáme v dané doméně D, z perspektivy osoby používající jazyk L, aby mluvila o D.“
  - (Wikipedie) "Je to formální a deklarativní reprezentace, která obsahuje definici pojmů a tezaurus (=definici vztahů mezi jednotlivými pojmy). Ontologie je slovníkem, který slouží k uchovávání a předávání znalostí týkající se určité problematiky."
- Gruber:
  - „Ontologie je explicitní specifikace konceptualizace.“ (explicitní = znalost je jednoduše dostupná, vyjádřitelná; konceptualizace = systém pojmů modelující určitou část světa)
- Řeší, jak znalosti \* Získávat \* Sdílet \* Uchovávat (reprezentovat, modelovat) \* Zpracovávat a používat \* Předávat \* Vytvářet

## Komunikace člověk – člověk

- Stačí jednoznačná, neformální ontologie
- Komunikace mezi znalostním inženýrem a expertem
- Usnadňuje učení, vzájemné pochopení
- Podpora sběru požadavků

## Komunikace člověk – počítač

- Znovupoužitelnost
  - Formální model je možno používat opakovaně
    - Důležité entity, atributy, procesy a vztahy jsou již namodelované
  - Lze vytvářet různé aplikace, uživatelsky přizpůsobené obsahem i pojetím
    - Nad stejnou ontologií
- Vyhledávání
  - Základ indexu pro úložiště informací
- Spolehlivost
  - Možnost automatické kontroly konzistence
  - Formální modely umožňují strojové zpracování, vyhodnocení, kontrolu



## Komunikace počítač – počítač

- Podpora komunikace pomocí jednotného slovníku
- Zajišťuje jednoznačné reference na pojem/objekt
- Ontologie je součástí formátu výměny dat

## Základní prvky ontologie

- Jedinec
  - Základní stavební objekt datového modelu ontologie
- Třída
  - Množina jedinců určitého typu
  - Podmnožinou je podtřída
  - Může obsahovat jedince i podtřídy
- Atribut
  - Popisuje vlastnost, charakteristiku či parametry jedince
  - Každý atribut obsahuje přinejmenším název a hodnotu
- Vazba
  - Jednosměrné, nebo obousměrné spojení dvou jedinců
  - Typicky atribut jehož hodnota je jiný objekt v ontologii
  - Nejdůležitější typ vazeb je zařazení
  - Taxonomie vytváří neúplné uspořádání
  - Meronymy relationship (part-of) vztahy ukazují jak je vytvořen kompletní celek pomocí jednotlivých objektů

## Typy ontologií

### Dle expresivity jazyka a formálnosti

#### *Informační*

- Kresby, náčrtky, diagramy, schémata
- Pro organizaci myšlenek, podpora v rozhodování a učení
- Myšlenková mapa – poskytnutí informací o určitém předmětu zájmu, který je v centru
- Pojmová mapa – není centrální prvek, více vztahů mezi koncepty, pro celkový pohled na doménu

#### *Lingvistické/terminologické*

- Glosáře, slovníky, taxonomie, tezaury
- Reprezentace a definice slovníku – seznamu slov používaného daným jazykem
- Glosář
  - Vysvětluje ne zcela jasné pojmy
  - Využití přirozeného jazyka
- Taxonomie
  - Kolekce pojmů organizovaných do hierarchie
  - Nejčastější aplikace – strukturované slovníky pro klasifikaci zdrojů
- Tezaurus
  - Nabízí uživateli seznam synonym, antonym pro vyjádření významu

## Softwarové

- Poskytují konceptuální schémata využitelné při analýzách a návrzích softwarových systémů
- Pohled na strukturu systému, nikoliv jeho funkcionalitu
- Např UML/AML

## Formální

- Reprezentovány formálními jazyky s přesně definovanou sémantikou
- Nejvhodnější pro strojové zpracování znalostí
  - Mají vysokou úroveň formalizace
- Rámcové
  - Rámec jako struktura dat, která obsahuje informace o jedinci
  - Lze je strukturovat do podoby sítě
  - Inspirace při objektově orientovaném modelování
  - Syntéza deklarativního a procedurálního přístupu k reprezentaci dat
- Sémantické sítě
  - Založené na reprezentaci pomocí gramů
  - Uzel = koncept
  - Hrana = relace mezi uzly
  - Chybí jim matematická formalizace
  - Aplikace
    - Zpracování přirozeného jazyka
    - Znalostní a expertní systémy
    - Neformální způsob reprezentace znalostí
- Konceptuální grafy
  - Logický formalismus vycházející ze sémantických sítí
  - Strukturovaný do podoby orientovaného grafu
  - Struktura vhodná pro člověka, ale i pro strojové zpracování

## Dle specifičnosti

### Generické

- Doménově nezávislé
- Zachycení obecných zákonitostí platných napříč problémovými oblastmi
- Využitelnost
  - Výchozí bod pro tvorbu nových ontologií
  - Reprezentace znalostí běžného uvažování
  - Definice návrhových vzorů pro běžně se vyskytující situace

### Doménové

- Zaměřené na konkrétní věcnou oblast nebo její dílčí část
- UMLS, FMA, FOAF ontologie

## Reprezentace formálních ontologií

- Ontologie reprezentovány formálními, semiformálními nebo neformálními jazyky

## RDF

- Reprezentace metadat o webových zdrojích

- Dekomponuje znalosti do malých fragmentů – podmět, přísudek, předmět
- Pro jednoznačnou identifikaci věcí používá URI reference nebo konstantní hodnoty (literál)
- Pouze abstraktní přepis pro popis metadat/ontologie

## RDFS

- Nadstavba RDF pro tvorbu ontologií, která vznikla rozšířením syntaxe a sémantiky
- Třída je jakýkoliv zdroj, která má specifickou vlastnost (rdf:type), jejíž hodnotou je právě rdfs:Class
- Definuje pojem třídy
  - Definuje jak vyjádřit, že zdroj patří do jedné nebo několika tříd
- Oproti RDF poskytuje
  - Specifikace definičního oboru a oboru hodnot
  - Specifikace datových typů
  - Reprezentace tříd a jejich hierarchií
  - Reprezentace hierarchie vlastností
  - Tvorba komentářů
- Nelze vyjádřit
  - Ekvivalenci tříd
  - Inverzní vlastnosti
  - Funkcionalitu vlastnosti nebo její tranzitivitu
  - Omezení kardinality vlastností
  - Negaci tvrzení
  - Disjunktnost tříd, ale pouze vztah je podtřídou
  - Třidu použitím logických operátoru AND, OR, NOT na třídy

## OWL

- Jazyk pro tvorbu ontologií využitelných v prostředí sémantického webu
- Založený na RDF(S) a deskripční logice
- Typy vlastností
  - Inverzí – reprezentace dvousměrných vztahů
  - tranzitivní – pokud A je ve vztahu X k B a B je ve vztahu X k C pak A je ve vztahu X k C
  - symetrická – pokud A je ve vztahu X k B pak B je ve vztahu X k A (např. „sousedíS“)
  - reflexivní – pokud platí A je ve vztahu X k A
  - datotypová vlastnost – přiřazuje jedincům primitivní hodnoty (integer, float, string, boolean, ...)
- Nové vlastnosti v OWL2
  - Ireflexivní – A není ve vztahu X k A, tj. Žádný jedinec nemá vztah sám k sobě
  - Asymetrická – Jestliže existuje vztah mezi jedincem A a jedincem B, pak nepřipouštíme vztah jedince B s jedincem A
- Vymezení třídy za účelem určení jejího významu v porovnání s jinými třídami
- Omezení se vztahují na hodnotu vlastnosti
  - someValuesFrom (some) – alespoň jedna hodnota vlastnosti musí být uvedeného typu, ale mohou existovat i další hodnoty
  - AllValuesFrom (only) – všechny hodnoty vlastnosti musí být uvedeného typu
  - hasValue

- minCardinality, maxCardinality, cardinality...
- Nutné podmínky
  - Jestliže něco patří do třídy X, pak splňuje podmínky
- Nutné a dostačující podmínky
  - Jestliže něco... a zároveň...
- Anonymní třída
  - Bezejmenná, vymezená definováním logických výrazů
  - Třída není v hierarchii tříd explicitně vyjádřena
  - Členy jsou jedinci, kteří splňují určitou logickou definici
- Primitivní třída
- Definovaná třída
  - CWA – close world assumption – není v OWL
    - Nějaké tvrzení o světě může být pravdivé, nepravdivé nebo nerozhodnutelné
    - v uzavřeném světě nepředpokládáme, že budou zjišťovány nové skutečnosti, které by mohli změnit pohled na svět
  - OWA – open world assumption – je v OWL
    - V otevřeném světě předpokládáme, že časem bude možné přidat další informace k aktuálnímu zkoumané domény
    - Předpoklad otevřeného světa: „Jestliže tvrzení X (obžalovaný je nevinný) není pravdivé, pak tvrzení X může být jak nepravdivé (tudíž obžalovaný je vinný) tak nerozhodnutelné.“ \* „Jestliže se neprokáže nevina, pak nelze usuzovat, že obžalovaný je vinný.“

## Návrhové vzory

- Návrhový vzor je konstrukcí, která pomáhá s implementací často se opakujících netriviálních problémů

## Rozklad hodnot

- Vyjádření určité vlastnosti používané pro popisy nebo definice tříd
- Kvůli OWA musíme dále rozklad hodnot uzavřít a výslovně zadat, že výčet hodnot je úplný = axiom pokrytí

## Pseudojedinec

- Pokud chceme při modelování ontologie zůstat na úrovni tříd
- Platí
  - Pseudojedinec je třída, která nemá žádné podtřídy
  - Pseudojedinec je podtřídou té třídy, do které by měl nahrazovaný jedinec patřit
  - Do třídy pseudojedinec patří jediný, a to právě ten nahrazovaný jedinec
  - V ontologii výslovně reprezentujeme i to, že se jedná o třídu typu pseudojedinec
  - Pseudojedince použijeme ve všech výrazech, kde bychom měli použít nahrazovaného jedince
- Lze použít i v případech, kdy si nejsme jisti, zda daný jedinec již může jedincem být

## N-ární vztah

- Binární relace x N-ární relace
- Pokud je nutné reprezentovat další vlastnosti u určitého vztahu

- Primitivní třídy: Zvíře, Povaha
- Přídavné třídy: TypyPovahy
- Vztahy: Zvíře – máPovahu – Povaha

## Část – Celek

- Snižuje složitost
- Místo mnoha objektů se zabýváme jedním
- Modelování
  - Stavebnicové díly
    - Reprezentujeme objekty, mezi kterými existuje vztah
  - Vztah je částí mezi stavebnicovými díly
    - Reprezentujeme skutečnost, že stavebnicový díl se skládá z jiných stavebnicových dílů, jako tranzitivní vztah
  - Objekty s lokalizací na stavebnicových dílech
    - Reprezentujeme objekty, které mimo jiné chceme lokalizovat na stavebnicové díly
  - Vztah "je lokalizován" mezi stavebnicovým dílem (SD) a objektem s lokalizací na stavebním díle
    - Reprezentujeme skutečnost, že jeden objekt (OSLO) je lokalizován na jiném objektu (SD) jako vztah hasLocation.
  - Propojení objektů s lokalizací (OSLO) se stavebnicovými díly (SD) aneb tvorba definovaných tříd
    - Definice podtříd objektů s lokalizací
  - Klasifikace klasifikátorem

## Normalizace ontologie

- Pro udržitelnost ontologie
- Ontologii vytváříme jako stromovou strukturu nikoliv jako graf
- Na nejvyšší úrovni pouze 3 třídy

## Primitivní třída

- Nevyžaduje další definici, jednoznačný název
- Pro modelování konceptu, který v přirozeném jazyce vyjadřujeme podstatným jménem
- Modelujeme jako neúplnou třídu, tj. třídu pouze popisujeme pomocí omezení (existenčních a/nebo univerzálních) v bloku NECESSARY

## Přídavná třída

- Slouží k upřesnění jiných pojmů. Např.: červená, mladý, náročné, dlouho, levně.
- Obvykle představuje koncept, který v přirozeném jazyce vyjadřujeme přídavným jménem (odpověď na otázku Jaký?) nebo příslovcem (odpověď na otázku Jak?)
- Přídavnou třídu modelujeme nějakou dimenzí dané třídy
- Modelujeme s použitím návrhového vzoru "Rozklad třídy na podtřídy"

## Definovaná třída

- K modelování definovatelného konceptu, tj. takový koncept, k jehož vymezení použijeme jiné koncepty

- Např.: stařec (člověk, který je starý), student (člověk, který studuje), matka (člověk ženského pohlaví, který má alespoň jednoho potomka)
- Definovanou třídu modelujeme pomocí omezení (existenčních, univerzálních) v bloku NECESSARY AND SUFFICIENT, případně dále popisujeme pomocí omezení v bloku NECESSARY

## Odvozování

- Odvozovací stroj = program, který je schopen z ontologie odvodit informace/znalosti, které v ní nejsou explicitně vyjádřeny
- Pomocí nich může uživateli pomoci navrhnout a udržet bezchybné ontologie, které budou smysluplné, korektní, minimálně redundantní a bohatě axiomatizované
- Mezi standardní činnosti odvozovacího stroje patří:
  - Kontrola konzistentnosti ontologie
  - Klasifikace tříd
  - Klasifikace jedinců
  - Kontrola konzistence ontologie: brání disjunktnosti tříd
  - Klasifikátor tříd: je třída A podtřídou třídy B?
    - Klasifikátor (reasoner) slouží k provádění tzv. testu subsumpce = podle podmínky uvedených u definovaných a popsanych tříd testuje, jestli jedna třída nemůže být podtřídou jiné = vytváří odvozenou (inferred) hierarchii. Zviditelňuje skryté znalosti.
    - Klasifikátor umí též ontologii zkontrolovat, jestli si některé její části navzájem logicky neodporují (neporušují pravidla disjunkce, kardinality, špatný Df či Hf u vlastností...)

## Nástroje

- JENA
  - Zdarma šířená, open-source platforma pro vývoj aplikací SW využívajících jazyk Java
  - komponenty:
    - API pro RDF, OWL
    - Čtení/zápis RDF v notaci RDF/XML, N3, N-triples
    - Úložiště pro modely RDF
    - Engine pro dotazování se pomocí SPARQL
    - Pravidlový engine
  - Vývoj nejčastěji v prostředí Eclipse
- SESAME
  - Open-source Java framework pro ukládání a dotazování se na RDF
  - Komponenty:
    - RDF model: definuje rozhraní a možnosti implementace RDF struktur
    - API: metody pro práci s RDF daty (+ odvozování s RDFS)
    - HTTP server: Java servlety implementující protokol pro přístup k repozitáři Sesamu přes HTTP
- Ontopia Navigator Framework
  - Framework založený na platformě J2EE
  - Využitelnost: vývoj webových stránek zapsaných v JSP (JavaServer Pages) s využitím ontologie vytvořené dle standardu Topic Maps

- Protégé
  - Bezplatný open source
  - Poskytuje grafické uživatelské rozhraní pro definici ontologií
  - Lze z něj exportovat do všech hlavních jazyků
  - Klasifikátor Pellet

## Aplikace ontologií

- Znalostní management ve firmách
  - Pro efektivní fungování organizace je třeba, aby se informace a znalosti (jak interního, tak externího původu) neztrácely, a včas se dostávaly k těm pracovníkům, kteří je mohou využít
  - Lze zabezpečit konzistenci znalostní, jednak usnadnit jejich vyhledání.
- Elektronické obchodování typu B-to-C i B-to-B
  - Vyhledání požadovaného produktu zákazníkem či rychlé vyhledání potenciálního partnera, ale perspektivně také automatizace procesu sjednání obchodních podmínek.
- Zpracování přirozeného jazyka
  - Terminologické ontologie mohou napomáhat např. při překladu nebo automatické sumarizaci textů.
- Inteligentní integrace informací
  - Zastřešení datových schémat distribuovaných zdrojů (strukturovaných nebo semi-strukturovaných databází, případně „tabulárních“ webových stránek) na vysoké úrovni abstrakce
- Pojmové vyhledávání informací jako vylepšení stávajících internetových vyhledávačů
- Sémantické webové portály konstruované polo-automaticky na základě metadat od poskytovatelů informace
- Agentové systémy
- Wikidata
- Google Knowledge Graph
- MusicBrainz
- Inteligentní výukové systémy
- Společné rysy aplikací:
  - Využívání strukturovaných dat – jednotný způsob vyjadřování se o "věcech", formalizace tvrzení o "věcech" pro strojové zpracování, usnadnění chápání o "věcech" pro stroje-> tj. Zejména pro webové vyhledávače
  - Kombinace dat z více zdrojů - sdílení informací a znalostí, vyjádření souhlasu s obsahem zdroje

# Sémantický web

## Základní pojmy

- Metadata: klíčový aspekt sémantického webu
- Obsah sémantického webu = data + metadata
  - Smysl: obohacení dat na webu o významově bohaté popisy (výroky) pro dosažení automatizovaného zpracování webových dokumentů webovými agenty
  - Přejít od webu informačního k webu znalostnímu
- Cíle sémantického webu:
  - Dělat za uživatele triviality
  - Dodávat informacím na webu jednoznačně reprezentovaný význam, zachycený v jazyku přístupnějším pro automatické zpracování
  - Samostatně nalézat relevantní zdroje informací, zpracovávat je a odvodit ze stávajících znalostí znalosti nové
  - Podpořit sdílení informací a znalostí s ostatními
- Web 1.0
  - První implementace webu, Read-only web
  - Statické webové stránky bez možnosti interakce
- Web 2.0
  - Read-write web (load, download)
  - Zaměřen na tvorbu obsahu, komunitu, spolupráci
  - Interakce s jinými uživateli na webu, sdílení obsahu
  - Typicky blogy, wikipedia, RSS, XML, online webové služby (Gmail, eBay), tagging („folksonomie“), Google, Flickr, YouTube
- Web 3.0
  - „The portable personal web“, „Semantic Web“, „read-write execute web“
  - Tvorba vysoce kvalitního obsahu a služeb s využitím technologií Webu 2.0
  - Využití prvků umělé (výpočetní) inteligence
  - Nabídka různých pohledů na stejná data
  - Zaměření se na kontext a personalizaci – obecněji na jedince
  - Příklady aplikací: iGoogle, Google Maps, Facebook, Swoogle, FOAF
  - Nástrojem pro vytvoření sémantického webu jsou propojená data
  - Propojená data jsou aplikována pomocí RDF, RDFS, OWL
  - Google Knowledge Graph - Znalostní báze, kterou reprezentuje google vyhledávaná data
  - MUSICBarinz - otevřená hudební encyklopedie

## Metadata

- Data o datech
- Obsahově nezávislá metadata
  - Poslední datum uložení dokumentu, místo uložení, přístup typu uživatele k dokumentu, ...
- Obsahově závislá metadata
  - Velikost dokumentu, barva obrázků, autor, počet stran, jazyk dokumentu, ...
- a) Strukturovaná metadata:



- Metadata spojená se strukturováním dokumentu (kapitoly a odstavce knihy, položky objednávky, ...)
- b) Doménově specifická metadata:
  - Metadata související s aplikační doménou
- Mikroformáty - Metadata vytvářené v HTML pomocí atributu class
  - Několik typů např. XFN pro reprezentaci osob a vztahů mezi nimi, hCard (online vizitka), hCalendar (záznamy v kalendáři)...
  - Nevýhodou je přílišné užívání atributu class a naopak výhodou je široká podpora a relativní jednoduchost implementace
- RDFa – Pro metadata využívá slovníky, využívá atributy typu typeOf, property atd..
  - Výhody
  - Možnost kombinace jednotlivých slovníků
  - Strojově zpracovatelné
  - Snadná prezentace
- Nevýhody
  - Podpora pouze HTML5
  - Nůstnost znát slovníky
- HTML5 mikrodata
  - Rozšíření HTML5
  - Jedná se o podmnožinu RDFa
  - Data jsou strukturována jako vlastnost-hodnota
  - Využívá slovník Schema.org
  - Nevýhodou je horší kombinace více slovníků a problematická konverze do RDF
  - Výhodou je nižší složitost oproti RDFa
- JSON-LD
  - Javascriptový zápis objektů
  - Nenachází se přímo v HTML kódu a je nezávislý na platformě a technologii
  - Zápis dat formou klíč-hodnota
  - Google doporučuje využívání pro rich snippets

## RDFS, OWL

- Viz. [RDFS](#), [OWL](#)

## Dotazování na sémantický web

- SPARQL: SPARQL Protocol and RDF Query Language
  - Specifikace W3C (Recommendation - 2008) definující syntaxi a sémantiku dotazovacího jazyka pro RDF
  - Jazyk deklarativní
  - Vychází z SQL, podobná syntaxe, některé funkce jsou totožné - COUNT, ORDER BY...
  - Založený na použití RDF modelu
  - Po stránce syntaxe podobný jazyku SQL
  - Použití:
    - Dotazování se na RDF graf formou porovnávání se vzorem (pattern matching)
  - Předchůdci:
    - rdfDB, RDQL a SeRQL

- Klady:
  - Dotazování se na data primárně zapsaná v RDF
  - Možnost dotazování se na data, která nejsou zapsána v RDF (relační databáze, data v XML)
  - Ve fázi výzkumu jeho využití pro RDFS a OWL
- Zápory:
  - Zatím nedostatečné nasazení jazyka: datových uložišť pro možnost kladení dotazů pomálu (x SQL, x XPath)
  - Nemožnost kladení dotazů na tranzitivní vztahy nebo hierarchické struktury, které se v RDF(S) grafu mohou objevit
  - „nedospělý“ jazyk, stále ve vývoji
- Xpath
  - Umožňuje adresaci konkrétních XML elementů pomocí předků, potomků, sourozenců apod.
- Xquery
  - Jazyk pro provádění dotazů na strukturovanými i nestrukturovanými daty ve formě XML
- SQWRL
  - Pro OWL
  - Podobný SQL, pravidlový jazyk (if – then)

## Odvozování

- Například Pellet
- Umožňuje identifikovat vztahy mezi jednotlivými třídami a rozšířit a případně lépe propojit hierarchii tříd
- Výhodou je kontrola konzistence ontologie
- Menší požadavky na tvůrce ontologie
- Nalezne všechny souvislosti mezi jednotlivými daty
- RDFS umožňuje pouze omezené odvozování (třída-nadtřída, vlastnost-podvlastnost)
- OWL umožňuje zařazovat jednotlivé třídy do zdánlivě nesouvisejících jiných definovaných tříd

## Námětové mapy

### Standard Topic Maps

- Mezinárodní standard ISO/IEC 13250 pro reprezentaci
- Informací a znalostí, pro zlepšení nalezitelnosti informací v prostředí webu
- Založen na formálním modelu, který v sobě zahrnuje rysy taxonomie, indexu, tezauru a také ontologie
- Využívají dvouvrstvý model dat
  - Znalostní vrstva – náměty ztělesňující konkrétní či abstraktní subjekty a asociace
  - Informační vrstva – reprezentuje obsah, informační zdroje (dokumenty, audio, video,...). Obvykle sada zdrojů, u kterých chceme využitím map námětů zlepšit možnosti navigace a vyhledávání
- Struktura:
  - 1) Přehled a základní koncepty
    - Představení základních konceptů námět. map

- 2) Datový model
  - Poskytnutí základů pro implementaci námět. map
  - Definice struktur a jejich omezení
  - Základ pro nově vznikající standardy (TMCL, TMQL)
- 3) XML Syntaxe
  - XTM syntaxe založená na XML (standard ISO)
  - Dokument definuje konstrukce, které lze pro tvorbu námětových map s XTM využít
- 4) Kanonizace
  - Zajištění jednotného způsobu reprezentace námětových map dle předem definovaných pravidel
- 5) Referenční model
  - Abstraktnější pohled na námětovou mapu než nabízí datový model
  - Námětová mapa je reprezentována pomocí uzlů (náměty) a hran (asociace)
- 6) Kompaktní syntaxe
  - CTM (Compact Topic Maps Syntax)
  - Odlehčenější textová syntaxe oproti XTM (doplněk k XTM)
  - Základ pro nově nastupující standardy (TMQL, TMCL)
- 7) Grafická notace
  - Vizualní ztvárnění námětových map, doplněk k XTM a CTM
  - Možnost tvořit mapu pomocí grafického vyjádření

## Stavební prvky

- Typ námětu (třída)
  - Např. Jazyk, Opera, Skladatel
  - Skupina prvků s určitými vlastnosti
- Typ asociace
  - Např. Složen, Narozen
  - Obecný předpis vztahu mezi typy námětů
- Typ výskytu – obecný předpis mezi námětem a informací
  - Interní – v rámci znalostní vrstvy (literály)
  - Externí – do vrstvy informační
- Typ role – způsob účasti aktéra v asociaci
- Námět (instance) – konkrétní prvek se vztahy k dalším námětům
- Asociace – vyjadřuje konkrétní vztah mezi náměty

## Tvorba námětové mapy

- Editory: Ontopia, Wandora, CmapTools, TM4L
- Ontopia
  - Zamýšlen pro výukové účely – pro porozumění přístupu Topic Maps
  - Komponenty:
    - Ontopia Topic Maps Engine: management námětových map
    - Ontopoly: editor námětových map
  - Omnigator:
    - Prohlížeč námětových map schopný interpretovat jakýkoliv dokument vytvořený dle standardu Topic Maps a generuje HTML

- Klient-server architektura postavená na http protokolu
- Vizigator: vizualizace námětových map
- Ontopia Navigator Framework (ONF): komponenta pro tvorbu webových aplikací s využitím dotazovacího jazyka tolog, jazyka JSP a specifických elementů ONF
- Ontopia Full-text Search: implementace vyhledávací funkcionality pro webovou aplikaci
- Ontopia Web Editor Framework: komponenta pro tvorbu editoru námětových map
- Wandora
  - Umožňuje:
    - Extrakce metadat ze souborů jpg, mp3, pdf, emailů, html stránek a dokumentů ve formátu doc
    - Využití syntaxí a jazyků: XML, XTM, LTM, N3 a RDF(S)
    - Realizace konverze SQL databáze do podoby námětové mapy
    - Publikace námětové mapy do formy webových stránek
    - Využití importu dokumentů XTM, LTM, RDF, ...
    - Funkcionalita rozšiřitelná pomocí pluginů
    - Vizualizace námětové mapy
- CMapTools

## Syntaxe

- XTM (XML Topic Maps)
  - Značkovací jazyk, který vznikl v roce 2000 na základě jazyka XML (eXtensible Markup Language) a za jeho vývojem stojí organizace TopicMaps.org
- CTM (Compact topic maps)
  - Syntax pro textový zápis. Odlehčenější verze XTM
- CXTM (Canonicalization XTM)
  - Formát sloužící pouze pro export do binárního kódu.
- TMQL (Topic Maps Query Language)
  - Standardizovaný dotazovací jazyk používaný pro extrakci dat z map námětů
- TMCL (Topic Maps Constraint Language)
  - Jazyk sloužící k popisu omezení v mapách námětů

## Dotazování na námětové mapy

- AsTMa?
- TMQL a Toma
- Tolog
  - Jeden z nejucelenějších dotazovacích jazyků pro námětové mapy
  - Použití:
    - Prostředí Ontopia
    - Projekt TM4J (Topic Maps for Java)
  - Tolog čerpá z logického programovacího jazyka Prolog (Programming in Logic) a z jeho podmnožiny Datalogu (dotazovací jazyk používaný v relačních databázích spolu s jazykem SQL (jazyk pro manipulaci s daty v databázích))
  - Základní konstrukcí je predikát s parametry
  - Predikát zachycuje vztahy mezi objekty nebo vztahy mezi vlastnostmi objektů
- Predikáty:

- Vestavěné (built-in):
  - Sada predikátů, které nabízí sám Tolog k používání
  - Predikáty použitelné k porovnávání hodnot :  $\geq$ ,  $\leq$ ,  $=$ ,  $<$ ,  $>$ ,  $\neq$  (nerovnost), ...
  - Specifické predikáty: instance-of, topic, occurrence, ...
  - Predikáty pro práci s řetězci: length, contains, substring, ...
- Uživatelské: predikáty, které vytvoří sám uživatel
  - Např. pracujePro, studuje, programujeV, ...
  - Pravidla

## Využití námětových map

- Organizace velkých objemů informací pomocí ontologií
- Zachycení „paměti organizace“, tacitních znalostí
- Reprezentace složitých procesů a pravidel
- E-learning zaměřený na osvojování si konceptů
- Správa distribuovanýchází znalostí a informací
- Sloučení informací a znalostí

# Objektové modelování a programování

## Vznik

- Vznikl postupným vývojem ze strukturovaných přístupů
- Původní sekvenční dávkové zpracování bylo postupně nahrazováno interaktivní komunikací s nutností reakce na události
  - Vzniklo událostmi řízené programování
- Také rozsah systémů se výrazně zvyšoval
  - Bylo nutné snižovat náklady na výrobu SW skládáním z již existujících modulů / komponent
- Základní myšlenkou je integrace algoritmů s datovou částí do jediné uzavřené entity (objektu)
  - Skrývá svoji vnitřní strukturu
  - Komunikuje s okolím prostřednictvím rozhraní (v reakci na události)
- Systém tvoří objekty, které komunikují prostřednictvím rozhraní

## Základní pojmy

- Proč používat objekty
  - Jsou nám bližší
  - Přehlednější kódy
  - Zajímavé nástroje a jazyky
  - Znovupoužitelné
  - Snadná správa programů
- Co v současnosti umí?
  - Mají chování
  - Mají vlastnosti
  - Umí dědit od předků
  - Umí se přizpůsobovat
  - Umí komunikovat
  - Mohou se dorozumívat na dálku
  - Mohou se stěhovat
  - Přežívají i mimo operační paměť (persistence)
- Existují dva základní přístupy k programování
  - Strukturované
    - Oddělení dat a procesů
    - Rozdělení algoritmu na dílčí úlohy, které se posléze spojí
    - Každý systém do značné míry unikátní
    - Programování relativně pracné, nákladné
  - Objektové
    - Integrace dat a procesů do jediné struktury (= objektu)
    - Celek se pak skládá ze samostatně existujících objektů, které mezi sebou komunikují
    - Komunikace pomocí zpráv
    - Objekty jsou uzavřené, zapouzdřené
    - Mezi základní charakteristiky tohoto typu programování patří
      - Vysoká znovupoužitelnost kódu

- Snadnější údržba a rozvoj
  - Relativně nižší náklady na vývoj v porovnání se strukturovaným programováním
- V objektovém modelování je základním prvkem objekt
  - Vychází z reality
  - Nabízí svému okolí sadu služeb
  - Ve formě rozhraní
  - S tímto rozhraním pracuje jeho okolí
  - Okolí nezná vnitřní strukturu – zapouzdření
- Mají svůj životní cyklus
  - Vznikají, „žijí“ (mění stavy, komunikují...) a zanikají
- Základní vlastnosti objektů
  - Abstrakce
    - Jen podstatné rysy a vlastnosti
  - Synergie
  - Hierarchie
  - Identita
    - Každý je unikátní
  - Odpovědnost
- Třída je šablonou pro konkrétní objekty
- Každý objekt si s sebou nese vlastnosti (informace o sobě, atributy) a vykonává nad sebou akce (metody)
  - Atributy
    - Nositelé informací o objektu
    - Jejich hodnoty představují stav objektu
    - Obvykle tvořeny primitivními datovými typy
    - Ostatním objektům by měli být skryté – zapouzdření
      - Měli by přístupné pouze pomocí metod (getter/setter)
  - Metody
    - Funkce, které může objekt vykonávat nad sebou samým (nad svými atributy)
    - Tyto funkce jsou spouštěny zprávami
    - Mohou mít vstupní a výstupní parametry
    - Veřejné metody tvoří rozhraní
  - Třídy objektů
    - Abstraktní popis skupiny objektů se stejnými charakteristikami, chováním a interakcemi
    - Jsou abstrakcemi důležitých vlastností objektů
    - Definují třídy operací vykonávaných instancemi
    - Objekty znají své třídy, podle kterých vznikly
  - Instance třídy (objekt)
    - Konkrétní výskyt třídy v IS nebo reálném světě, má konkrétní hodnoty atributů

## Modelování

- Vychází z faktu, že budovaný IS je obrazem (modelem) vnějšího světa
- Odráží tedy realitu a struktura systému vychází z reality

- Nelze modelovat přesně celou realitu
  - => Abstrakce
    - Odstranění nepodstatných prvků
    - Lze tak modelovat i velmi rozsáhlé systémy
- Díky modelování
  - Je usnadněná komunikace se zákazníkem
  - Lépe se udržuje přehled o aktuálním stavu projektu
  - Snáze se vytváří dokumentace atd.
- Hlavní principy modelování
  - Abstrakce
    - To, co pro mě není důležité, odstíním
  - Formalizace
    - Je přesně dané, co se, jak vyjadřuje
    - Usnadnění komunikace v týmu i se zákazníkem
  - Jednoznačnost
    - Vyplývá z formalizace; každý prvek lze jednoznačně identifikovat a popsat
  - Snížení nadbytečností
  - Princip tří architektur
    - Postupná tvorba tří typů architektur
    - Každá má jinou míru abstrakce, logiku a hloubku popisu
    - Konceptuální úroveň
      - Popisuje obsah, ne formu (CO?)
    - Technologická úroveň
      - Popisuje technologii, která bude použita (JAK?)
    - Fyzická úroveň
      - Popisuje detaily implementace (ČÍM?)
- Základní principy OOP a modelování
  - Zapouzdření
    - Atributy jsou přístupné pouze prostřednictvím metod
    - Umožňuje znovupoužitelnost
    - Umožňuje komponentní vytváření systémů
    - Usnadňuje údržbu a rozšiřování objektu, zajišťuje integritu objektu
    - Několik úrovní zapouzdření (private, protected)
  - Polymorfismus
    - Umožňuje k různým objektům přistupovat stejným způsobem
    - Metody různých objektů mohou být pojmenovány stejně, ale jejich realizace je různá
    - Zjednodušuje složité systémy
  - Dědičnost
    - Mechanismus, který umožňuje převzít vlastnosti třídy předka do potomka
    - Nové třídy jsou budovány z již existujících
      - Rozšířením (přidám nové atributy a metody)
      - Modifikací (změním význam stávajících atributů a způsob vykonání stávajících metod)
    - Dědičnost lze zobrazit v hierarchii = struktura tříd
    - Hierarchie od obecného ke speciálnímu => od generalizace ke specializaci



- Specializace
    - Shora - dolů (top down)
    - Začínáme obecným a specializujeme (zpřesňujeme) vlastnosti podtříd
  - Generalizace
    - Zdola - nahoru (bottom up)
    - Hledáme společné znaky několika tříd a zobecňujeme jejich vlastnosti
- Asociace (vztah) (kardinalita)
  - Vazby mezi objekty, umožňují objektům mezi sebou komunikovat
  - Mezi objekty různých tříd i mezi objekty stejné třídy
  - 1:1, 1:n, n:n, 0..1:10
- Agregace (skládání)
  - Objekty se skládají z dílčích podobjektů
  - Objekt může být částí většího celku
- Kompozice (neviditelné části)
  - Objekty se skládají z dílčích podobjektů, které bez těchto nadřazených objektů jinak nevznikají
  - Objekty vznikají a zanikají se svým rodičem
- Vazba
  - Program volá metodu objektu, jehož typ není předem znám a bude dán až za běhu programu
  - Včasná vazba
    - Statické metody, objekt, jehož metodu voláme je znám před spuštěním (early binding)
  - Pozdní vazba
    - Objekt je znám až za běhu programu (late binding)

## UML

- Unified Modeling Language
- Nástroj, jazyk objektového modelování
  - Průmyslový jazyk určený pro specifikaci, vizualizaci, vytváření a dokumentaci artefaktů softwarových systémů
- Specifikuje sadu diagramů a jejich notací
  - Pro jednotlivé fáze analýzy a návrhu
- Čtyři základní kategorie modelování
  - Modelování aplikačních procesů
  - Modelování tříd a objektů
  - Modelování komponent
  - Modelování rozdělení a nasazení komponent
  - Modelování typových úloh
- První fáze návrhu = specifikace požadavků zákazníka
- Rozdělení požadavků
  - Funkční
    - Co má budoucí systém dělat
- Ne-funkční
  - Další požadavky (např. na použité technologie, formu dokumentace atd.)

- Dále je nutné poznat firemní procesy
- Na model požadavků (a na firemní procesy – BPM) navazuje model typových úloh (use case model)
  - Základní nástroj specifikace funkčních požadavků
  - Popisuje základní funkcionalitu systému (k čemu jej uživatel používá)
  - Součásti diagramu
    - Aktér (třída uživatelů, role)
    - Typová úloha (základní funkční jednotka aplikace)
    - Interní model – vnitřní stavba typové úlohy, pro aktéra neviditelná
    - Verbální nebo strukturovaný popis
    - Vztahy mezi typovými úlohami – viz include a extend

## Modelování tříd

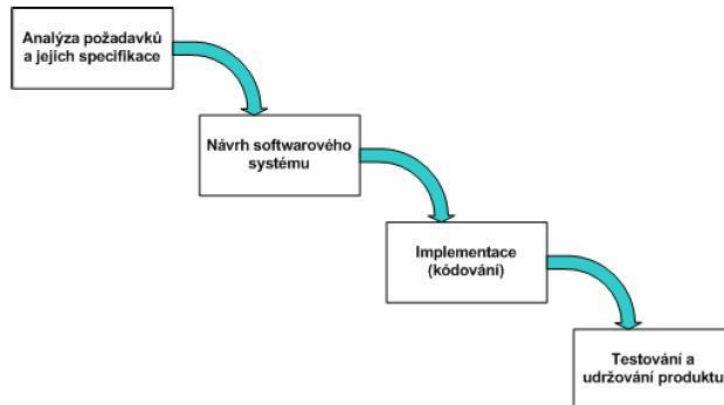
- Úrovně modelování
  - Modelování uživatelského rozhraní
    - Třídy vychází z komunikace aktéra a typové úlohy
    - Jejich účel je přijímat události a data od aktérů a předávat je aplikační logice
    - Mají málo výkonného kódu
  - Modelování objektů pro uložení dat
    - Popisují data a informace
    - Minimum výkonného kódu, dlouhodobější charakter
  - Modelování aplikační logiky
    - Prostředník mezi rozhraním a daty
    - Realizuje služby systému
    - Skládají se z pomocných a řídicích tříd
    - Velká znovupoužitelnost
- Diagram tříd (class diagram)
  - Vychází ze základní funkcionality popisované v Use Case Diagramu
  - Má popsat základní stavební prvky systému (tedy třídy) a jejich vzájemné vztahy
  - Součásti
    - Třídy objektů
    - Atributy
    - Metody
    - Asociace (prostá asociace, generalizace – specializace, agregace – kompozice...)

## Modelování objektových interakcí

- Nástroj pro realizaci typové úlohy; popis spolupráce jednotlivých tříd
- Dva základní grafy, vzájemně izomorfní
  - Sekvenční diagram (Object sequence diagram)
    - Časově orientovaný
    - Zobrazuje posloupnost zasílání zpráv v čase
    - Přiřazen k jedné typové úloze
  - Diagram objektové spolupráce (Object collaboration diagram)
    - Je více zaměřen na strukturu spolupráce, vztahy mezi objekty

## Softwarový proces

- Postup činností nutných k vytvoření softwarového produktu



Obr. 2.1: Vodopádový model softwarového procesu

## Událostmi řízené programování

- Event-driven programming
- Událost (Event) vzniká buď jako výsledek interakce GUI s uživatelem nebo jako důsledek změny vnitřního stavu aplikace či OS
- Obsluhou události
  - Nazýváme úsek kódu, který je při vzniku události automaticky vyvolán
  - Provádí činnost k události připojenou (někdy také ohlasová metoda události či Event Handler)
- Příklady typů událostí
- Klik/DvojKlik, Zaměření/Ztráta zaměření, Změna stavu komponenty, Stisk, uvolnění klávesy, Stisknutí, uvolnění tl. myši, Posun myši

## Architektura MVC

- Rozděluje do tří nezávislých komponent
  - Datový model aplikace
  - Uživatelské rozhraní
  - Řídící logiku
- Modifikace některé z nich má minimální vliv na ostatní
- Vytváření aplikací s využitím architektury MVC vyžaduje vytvoření tří komponent
  - Model (model)
    - Což je doménově specifická reprezentace informací, s nimiž aplikace pracuje
  - View (pohled)
    - Který převádí data reprezentovaná modelem do podoby vhodné k interaktivní prezentaci uživateli
  - Controller (řadič)
    - Který reaguje na události (typicky pocházející od uživatele) a zajišťuje změny v modelu nebo v pohledu

## Obecný princip MVC

1. Uživatel provede nějakou akci v uživatelském rozhraní (např. stiskne tlačítko)

2. Řadič obdrží oznámení o této akci z objektu uživatelského rozhraní
3. Řadič přistoupí k modelu a v případě potřeby ho zaktualizuje na základě provedené uživatelské akce (např. zaktualizuje nákupní košík uživatele)
4. Model je pouze jiný název pro doménovou vrstvu
  - a. Doménová logika zpracuje změněná data (např. přepočítá celkovou cenu, daně a expediční poplatky pro položky v košíku)
  - b. Některé aplikace užívají mechanismus pro perzistentní uložení dat (např. databázi)
  - c. To je však otázka vztahu mezi doménovou a datovou vrstvou, která není architekturou MVC pokryta
5. Komponenta pohled použije zaktualizovaný model pro zobrazení zaktualizovaných dat uživateli (např. vypíše obsah košíku)
  - a. Komponenta pohled získává data přímo z modelu, zatímco model nepotřebuje žádné informace o komponentě View (je na ní nezávislý)
  - b. Nicméně je možné použít návrhový vzor pozorovatel, umožňující modelu informovat jakoukoliv komponentu o případných změnách dat
  - c. V tom případě se komponenta view zaregistruje u modelu jako příjemce těchto informací
  - d. Je důležité podotknout, že řadič nepředává doménové objekty (model) komponentě pohledu, nicméně jí může poslat příkaz, aby svůj obsah podle modelu zaktualizovala
6. Samotnému konečnému zobrazení výsledku uživateli ještě může u web-aplikací předcházet odpověď ze serveru na klienta, aby si ihned vyžádal obnovení stránky (client side redirect, životnost 0, takže okamžitý):
  - a. Tím je zaručeno, že při obnovení stránky uživatelem (refresh) nevyvolá na serveru požadovanou akci opakovaně, ale že se jedná pouze o obnovení pohledu, nyní už bez požadavku na změnu dat (modelu)
  - b. Účelem je změna URL a dat http requestu, aby poslední v řadě již nebyl "server-side data-affecting" (ovlivňující model), ale pouze "read-only" (pouhé zobrazení). Celý tento client-refresh (změna URL) se děje automaticky a bez povšimnutí uživatelem
7. Uživatelské rozhraní čeká na další akci uživatele, která celý cyklus zahájí znovu

# Práce s kolekcemi

## Řídící algoritmy

- Stabilita algoritmu
  - Stabilitní znamená, že nedojde v průběhu k prohození prvků se stejnou hodnotou. Užitečné, jestliže dochází k postupnému řazení podle dvou (a více) parametrů. Řadíme-li např. osoby dle křestního jména a poté dle příjmení, pak výsledek stabilního algoritmu odpovídá očekávání (první je Karel Novák, následuje Václav Novák). Pokud by algoritmus nebyl stabilní, tak tento postup nebude fungovat, protože druhé řazení by mohlo zpřeházet výsledek prvního (Václav Novák by mohl být před Karlem Novákem).

## Bubble sort

- Postupně prochází ( $n-1$  průchodů) celou kolekci a porovnává (a popř. prohazuje) sousedící dvojice)
- $O(n^2)$ , stabilní

## Insertion sort

- Vždy vezme jeden prvek a ten je přidán na odpovídající místo v již seřazeném seznamu (na začátku v něm může být i jeden prvek)
- Stabilní, obecně  $O(n^2)$  složitost, ale výhodné pro téměř seřazené a malé kolekce

## Merge sort

- Princip, kdy se dvě kolekce spojují v jednu. Porovná se vždy nejvyšší prvky s každé a ten je pak vložen do té finální
- Řazenou kolekci je nejdříve nutné rozdělit (na jednotkové kolekce), které se pak slučují
- $O(n \cdot \log(n))$ , stabilní
- Využit v Javě pro řazení objektů (`Collections.sort`) neboť je stabilní a poskytuje danou složitost bez ohledu na vstup

## Quick sort

- Dojde k volbě pivotu, kolekce je poté rozdělena na prvky menší a větší než pivot. Na těchto částech je pak rekurzivně provedeno to samé.
- Důležitá je metoda volba pivotu:
  - Volba fixního prvku (první či poslední) – problém u částečně seřazených kolekcí – složitost až  $n^2$
  - Medián prvního, prostředního a posledního prvku
  - Náhodný prvek
- Při ideálním pivotu:  $O(n \cdot \log(n))$ , při špatném pivotu:  $O(n^2)$
- Nestabilní → defaultně využit v Javě pouze pro primitivní typy (`Arrays.sort`) a ne pro obecné objekty (`Collections.sort`), i kvůli tomu, že složitost může vystoupat až k  $n^2$

## Selection sort

- Vždy dojde k nalezení nejmenšího prvku, který je přesunut na začátek a ve zbylých prvcích se provede to samé
- Složitost  $O(n^2)$ , ale výhodou je jeho konstantní paměťová složitost

## Základní pojmy

### Kontejnery (v Javě obvykle nazývané "kolekce")

- Datové objekty, které nám umožňují snadno a efektivně spravovat variabilní hromadná data (i další objekty). Podle způsobů uložení dat a práce s nimi rozlišujeme různé druhy kontejnerů. Základními operacemi dle typu kontejneru je iterace, porovnávání, přidávání a mazání.

### Pole

- Nejzákladnější struktura vhodná zejména pro primitivní hodnoty, které se nemusí nahrazovat objektem (známe předem počet objektů ve skupině). Nevýhodou je jeho dynamické plnění (nutné definovat jeho rozsah – časová náročnost). Pole mohou být vícerozměrné.

### Kolekce

- Struktura vhodná zejména pro práci s objekty, kdy se „nemusíme“ starat o velikost kolekce. Nové objekty lze vkládat bez nutnosti realokace. Vhodné při vytváření předem neznámého počtu objektů.
- The Collection Framework (CF) – systém rozhraní a tříd, které bezprostředně souvisí s kolekcemi. Naprostou většinu z nich najdeme v balíku `java.util`

### Iterátor

- Prostředek zajišťující sekvenční přístup k datům, pracující krokově, v každém dalším kroku poskytne přístup k dalšímu prvku. Rychlé právě v případě sekvenčního procházení. Zajišťuje možnost procházení prvků bez znalosti jejich implementace.
- Nezáleží na pořadí, nebo je pořadí dané vlastnostmi kolekce
- Metoda `iterator()` daného kontejneru vrací instanci iterátoru
- Metody `next()`, `hasNext()`, `remove()`

### Porovnatelnost

- Důležitá vlastnost, kterou potřebujeme pro uložení do některých kontejnerů. Datové objekty mohou mít porovnatelnost implementovanou prakticky libovolným způsobem, třída implementuje rozhraní `Comparable` (obsahuje jedinou metodu `compareTo()`).

### Kolekce

- V Javě existují dvě samostatné koncepce kontejnerů; od JVM 1.5 generika
  - Kolekce
    - Skupina samostatných prvků
    - Rozhraní `List` (seznam) – musí prvky udržovat v určitém pořadí
    - Rozhraní `SET` (množina) – nesmí obsahovat duplicitní položky
  - Mapa
    - Skupina dvojic objektů klíč – hodnota
    - V podstatě „miniaturní databáze“
    - Někdy označovány jako ASOCIATIVNÍ POLE
    - Stejně jako pole lze rozšiřovat do více rozměrů (mapa, jejímiž hodnotami jsou také mapy)

## Seznamy

- Obecné rozhraní LIST
  - Prvek může být v kontejneru víckrát
  - Má jednoznačnou polohu (index)
  - Metody pro seřazení seznamu – SORT(LIST), promíchání seznamu – SHUFFLE(LIST), obrácení pořadí – REVERSE(LIST), hledání binárním dělením, kopírování seznamu, konverze kolekcí na pole atd.
  - Dvě implementace
    - ArrayList
      - Seznam implementovaný pomocí pole
      - Rychlý přímý přístup k prvkům, vkládání a odebírání je pomalé
    - LinkedList
      - Vkládání a odstraňování prvků je optimalizované, stejně tak sekvenční procházení
      - Přímý přístup k prvkům je relativně pomalý
      - Obsahuje metody navíc, které nejsou součástí rozhraní
- Vector
  - Nejzákladnější implementace = „pole“ s proměnnou velikostí
  - Prakticky shodné s třídou ArrayList, ale je vláknově bezpečná a má některé metody navíc (např. hledání od určitého indexu), lze mu stanovit přírůstek kapacity

## Množiny

- Rozhraní SET
  - Každý vložený prvek musí být jedinečný (jinak kontejner zabrání vložení)
  - Objekty vložené do kontejneru musí překrývat metodu Equals (umožní určit jedinečnost)
  - Není určeno žádné konkrétní pořadí
  - Implementace
    - HashSet
      - Množina s hešovací tabulkou
      - V situaci, kdy je třeba uložit obrovské pole nějakých řetězců a jednotlivé řetězce velmi rychle vyhledávat
      - Sekvenční prohledávání díky své velké časové náročnosti není vhodné
      - Jednotlivé řetězce se rozdělí do skupin, které si jsou podobné
      - K tomu jsou hešovací funkce, která dokážou jednotlivé řetězce rozptýlit do oněch políček
      - Objekty musí definovat metodu HASHCODE()
    - TreeSet
      - Setříděný kontejner typu množina, není moc rychlý
      - Rozhraní SORTEDSET
      - Metody FIRST() - vrátí nejmenší prvek, LAST() - největší prvek, SUBSET(FROMELEMENT, TOELEMENT) – podmnožina od určeného do druhého určeného prvku, HEADSET(TOELEMENT) –

podmnožina menších prvků než toElement,  
TAILSET(FROMELEMENT) – podmnožina větších prvků

- LinkedHashSet
  - Implementováno jako HashSet s LinkedListem
  - Seřazeno podle pořadí vložení
  - Prostředník mezi HashSet a TreeSet – podobná rychlost jako HashSet ale je seřazený

## Mapy

- Rozhraní MAP
- Udržuje dvojice klíč-hodnota
- Objekty nevyhledávám podle čísla, ale pomocí jiného objektu
- Metoda PUT(KLÍČ, HODNOTA) vloží hodnotu pod daný klíč
- Metoda GET(KLÍČ) – vrátí objekt odpovídající danému klíči
- Implementace
  - HashMap
    - Založeno na hešovací tabulce
    - Rychlé vkládání a vyhledávání položek
  - TreeMap
    - Oproti HashMap přináší seřazení klíčů
    - Výsledky se zobrazují setříděny podle pořadí vložení
  - LinkedHashMap
    - Lze seřadit i podle klíče
  - Hashtable
    - Starší implementace, je synchronizovaná



# Problematika perzistovaného ukládání dat ve vybraném programovacím jazyce

## Trvalá data

- Data, která jsou uložena v trvalé paměti (většinou je to disk)
- Data mohou být uložena v jednoduchém plochém souboru, nebo mohou být uložena v databázi
- Podle přístupu k datům můžeme persistentní data rozdělit do těchto kategorií:
  - Částečně perzistentní
    - Datové úložiště obsahuje jen aktuální verzi dat
    - Typický příklad: data v textovém editoru, data jsou držena jen v aktuální verzi
  - Perzistentní
    - Datové úložiště obsahuje aktuální verzi dat a verzi dat před poslední změnou
    - Typický příklad: Transakční zpracování v relačních databázích
      - Databáze si uchovává na začátku každé transakce aktuální stav, v případě že během transakce dojde k chybě je použita funkce rollback, která vrátí DB do původního stavu, před začátkem transakce
  - Plně perzistentní
    - Datové úložiště obsahuje aktuální verzi dat+všechny historické verze dat
- Persistence objektů (serializace)
- Persistence do DB
- ORM (Hibernate)

## Vstupy a výstupy

- Java řeší řadou tříd a metod
  - Uložené v balíku java.io
  - Tato knihovna založena na mechanismu tzv. vstupních a výstupních streamů (proudů)
- V zásadě čtyři základní třídy pro tvorbu streamů

## Vstupní streamy

- Čtení po bytech (binární)
  - Abstraktní třída InputStream
- Čtení po znacích
  - Abstraktní třída Reader

## Výstupní streamy

- Výstup po bytech (binární)
  - Abstraktní třída OutputStream
- Výstup po znacích
  - Abstraktní třída Writer
- Každá z abstraktních tříd má specifického potomka

- Abstraktní třídu, která je předkem filtrů
    - Třídy, které přidávají další funkce jednotlivým potomkům
- Vedle toho jsou v balíku
  - Třída File (práce se soubory a adresáři)
  - Třída StreamTokenizer
    - Rozdělení vstupního proudu na části

## Vstupní proudy

- Třídy založené na třídách InputStream nebo Reader
- Tyto třídy lze rozdělit do čtyř skupin
  - Třídy pro vytvoření vstupního proudu
  - Třídy pro rozšíření vlastností vstupního proudu
  - Třídy pro vytvoření readeru
  - Třídy pro rozšíření vlastností readeru
- Třída Reader by se měla používat vždy, když je potřeba číst text
  - Garantuje správnou obsluhu znakových sad
  - A převod textu do vnitřního kódování Javy (což je Unicode)
  - Čtení z textového souboru pak může probíhat následovně
    - `FileReader soubor = new FileReader(„soubor.txt“);`
    - Takto bych mohl číst text po znacích metodou `soubor.read();`
  - Pokud chci číst po řádcích, „obalím“ třídu `FileReader` filtrem `BufferedReader`
    - `BufferedReader filtr = new BufferedReader( soubor );`
    - Jehož parametrem je již vytvořený `FileReader`
    - A pak můžu číst po řádcích metodou `filtr.readLine();`
    - Obě tyto třídy disponují metodou `close()` pro zavření souboru
- Třída `InputStream` se používá pro binární vstup
  - Otevření streamu obdobné, jako u readeru, tedy
  - `FileInputStream soubor = new FileInputStream(„soubor.dat“)`
  - Pro čtení po bytu metoda `read(); readBoolean(); readChar();` atd.
  - Je-li třeba lepší čtení, lze tento stream „obalit“ filtrem `BufferedInputStream`
    - Bufferovaný vstupní stream
- `ObjectInputStream`
  - Umožní číst celé objekty, kontejnery atd.

## Výstupní proudy

- Třídy založené na třídách `OutputStream` a `Writer`
- I tyto třídy lze rozdělit do čtyř skupin
  - Třídy pro vytvoření výstupního streamu
  - Třídy pro rozšíření funkčnosti výstupního streamu
  - Třídy pro vytvoření writeru
  - Třídy pro rozšíření funkčnosti writeru
- Třída `Writer` pro zápis textových souborů
  - Zápis po znacích
  - `FileWriter cilovySoubor = new FileWriter(„soubor.txt“);`
  - `CilovySoubor.write();`
- Lepší je zápis po řádcích, proto je vhodné použít filtr `PrintWriter`
  - `PrintWriter filtrovanyVystup = new PrintWriter(cilovySoubor);`

- filtrovanyVystup.println();
- Opět metoda close() pro uzavření souboru
- Třída OutputStream pro zápis binárních dat
  - Pro zápis po bytech stačí otevřít FileOutputStream
    - FileOutputStream soubor = new FileOutputStream(„soubor.dat“);
  - Pro zápis celých objektů
  - ObjectOutputStream cil = new ObjectOutputStream(soubor);
  - cil.writeObject(...);

## Databáze

- Pro přístup se používá JDBC rozhraní (použití ovladače pro překlad požadavků do nativního volání daného úložiště – databáze, Excel soubory, Access atp.)
- JDBC specifikace rozpoznává čtyři typy JDBC ovladačů:
  - 1. JDBC-ODBC bridge
    - Ovladač přistupuje k databázi přes nativní ovladač ODBC, který musí být na daném stroji nainstalován
  - 2. Native-API driver
    - Ovladač přistupuje k databázi přes služby nativního ovladače dané databáze (využívá jeho metody, které jsou vytvořeny v C, C++), musí být na daném stroji nainstalován.
    - Jistá podoba s typem 1.
  - 3. Network-Protocol driver
    - Založen čistě na Javě a JDBC, který konvertuje svoji komunikaci do síťového protokolu a spojuje se s centrálním serverem (Network Server), který poskytuje připojení k databázi (obvykle s „poolem“ připojení). Tento server konvertuje síťový protokol, kterým komunikuje s klienty, do databázově specifického protokolu, jemuž již databáze rozumí. Takový model je vysoce efektivní, a to jak s ohledem na možnost „poolingu“ připojení a tím i zrychlení dotazování a práce s databází, tak i možnost připojení k sadě heterogenních databázových systémů.
  - 4. Native-Protocol driver
    - Tento ovladač komunikuje s databázovým serverem přímo přes síťový protokol

## JDBC API

- Používá pro práci s DB několik základních tříd a rozhraní, jsou to:
- DriverManager
  - Třída, která má na starost nahrávání a registraci dostupných ovladačů. Je to mezivrstva mezi JDBC ovladačem a programovým kódem.
- Connection
  - Toto rozhraní reprezentuje připojení k DB. Je to centrální rozhraní pro správu databázového spojení. Umožňuje přípravu a vykonání db dotazů, transakční zpracování.
- Statement
  - Rozhraní, které je zodpovědné za posílání SQL příkazů databázi. Rozhraní implementuje tyto metody:

- `executeQuery(sql)`: provádí SQL dotaz, který zpravidla vrací `ResultSet` což je seznam databázových vět
- `executeUpdate(sql)`: používá se v případě, že chceme modifikovat data v DB (INSERT, UPDATE, DELETE).
- **ResultSet**
  - Rozhraní představuje výsledky databázového dotazu(tabulku). Zpravidla je to několik datových vět. Metody tohoto rozhraní můžeme rozdělit do těchto kategorií:
    - Navigační metody: Tyto metody umožňují pohyb v množině výsledků pomocí funkcí `first()`, `next()`, `last()`, `previous()`
    - `getter`: `ResultSet` rozhraní poskytuje metody (`getBoolean`, `getString`, atd.), které vrací hodnoty jednotlivých sloupců v aktuální řádce
    - Modifikační metody: Tyto metody umožňují modifikovat (měnit, vkládat, rušit ..) data v `ResultSet` a následně provést modifikaci v DB. Jsou to např. `insertRow()`, `updateString`, atd

```

Class.forName("org.hsqldb.jdbcDriver");           1: Zavedení ovladače
String url = "jdbc:hsqldb:data/tutorial";          2: Vytvoření spojení s DB
conn = DriverManager.getConnection(url, "sa", "");  3: Přihlášení k DB
st = conn.createStatement();                       4: Vytvoření Statement pro
dotazování
String sqlQuery = "SELECT uid, name, duration from EVENTS";  5: Vytváření
dotazu.
ResultSet rs = stmt.executeQuery(sqlQuery);         6: Zpracování ResultSet.
rs.first();
while (rs.next()) {
    String name = rs.getString(2);
    Timestamp hireDate = rs.getTimestamp(5);
    System.out.println("Name: " + name + " Hire Date: " + hireDate);
}

```

- **Nevýhody JDBC**
  - Problém relační model + objektový přístup: Rozhraní `ResultSet` vrací data jako tabulku, ne jako objekty (OOP v Javě). Vhodnější řešení by bylo vrácení řádků dat jako objekty, které reprezentují jednotlivé entity v tabulkách (člověk, faktura). Protože `ResultSet` toto neumožňuje, programátor je velmi často nucen si takové třídy vytvořit a zde dochází k obrovské ztrátě času.
  - Vyjímky: Při každé operaci je potřeba odchytat mnoho vyjímek což značně znepřehledňuje kód.

## Objektově relační mapování

- Objektově relační mapování slouží k tomu, aby bylo možné snadno používat relační databáze v prostředí objektově orientovaných programovacích jazyků - Javě. Vzhledem k tomu, že objektově orientovaný návrh dat není jednoznačně převoditelný na relační databáze a opačně, používají se různé formy mapování.
- Mapování má za účel načítat data z relační databáze a naplnit jimi příslušné datové položky objektů, případně naopak datové položky objektů ukládat do databáze. Snahou ORM je co nejlepším využitím obou zmíněných technologií – tj. objekty by měly reprezentovat objekty reálného světa, jak to požadují principy OOP, na straně databáze

bychom zase měli využít všech možností relačních databází – indexy, pohledy, primární klíče, atd. V současnosti je snad nejpoužívanějším nástrojem pro ORM produkt od firmy JBOSS Hibernate

## Objektové databáze

- Objektové databáze umožňují skladování dat s libovolnou strukturou. Programátor či kdokoliv jiný se při použití objektové databáze nemusí vůbec starat o mapování objektových struktur do dvourozměrné tabulky.
- Každý persistentní objekt v objektové databázi má svoji vlastní identitu, je tedy jednoznačně odlišitelný od libovolného jiného objektu nezávisle na hodnotách svých vlastností – odpadají tedy problémy s tvorbou primárních klíčů. Kvalitní objektová databáze podporuje všechny vlastnosti nutné pro práci s objekty – zapouzdření, jednoznačnou identifikaci, reference mezi objekty a další (dědičnost, polymorfismus atp.)
- Např.: db4o, Ozone, ObjectDB

## Java serializace

- Objekt může být převeden na proud bytů (včetně objektových dat a informací o typu objektu a objektů, které obsahuje)
- Serializovaný objekt může být pak zapsán nebo přečten z bytového proudu ObjectInput/OutputStream pomocí metod writeObject a readObject a tak být zapsán do souboru či předán jiné aplikaci
- Pro serializaci objektu je nutné:
  - Aby implementoval rozhraní Serializable – není nutné implementovat žádné metody jak bude serializace probíhat neboť je využíváno reflexe a Java se o serializaci postará sama
  - Všechna pole objektu musí také implementovat Serializable
- Pozor na cyklické odkazy, neboť každé pole objektu je serializováno zvlášť a může dojít k zacyklení
- Je vhodné, aby třída implementující Serializable měla definované pole long serialVersionUID, které zajišťuje, že při deserializaci je zvolena správná třída objektu. (Má význam pokud se serializované objekty předávají mezi více aplikacemi či jejich verzemi.) V případě, že toto pole není uvedeno, je vytvořeno na základě různých vlastností třídy, ale může se lišit podle kompilátoru a byla by vyhozena výjimka InvalidClassException
- Existuje více možností jak serializovat např. Google Protocol Buffers či Androidí rozhraní Parcelable

# Webové aplikace

## Principy

- Webová aplikace
  - Aplikace poskytovaná uživatelům z webového serveru přes počítačovou síť Internet, nebo její vnitropodnikovou obdobu (intranet)
  - Populární především pro všudypřítomnost webového prohlížeče jako klienta
  - Schopnost aktualizovat a spravovat webové aplikace bez nutnosti šířit a instalovat software na potenciálně tisíce uživatelských počítačů
    - Hlavní důvod oblíbenosti
    - Tenký klient
- Principy
  - Založena na systému komunikace klienta (prohlížeč klientského počítače) a aplikace
  - Běžící na webovém aplikačním serveru
  - Klient vysílá požadavek (HTTP REQUEST)
  - Server zpracuje adekvátním způsobem a posílá nazpátek odpověď (HTTP RESPONSE)
  - Zpracování požadavku na serveru má na starosti mechanismus, který je příslušným způsobem vyhodnotí a přepošle dále do aplikace
    - Zasílá nazpátek odpovědi klientovi tak, že generuje výsledné webové stránky
    - Takový mechanismus se v jazyce Java nazývá SERVLET
  - Aplikace je většinou business logika společně s datovou částí
    - Vyznačuje se znovupoužitelností například i v desktopových aplikacích
  - Webový aplikační server je kontejner, umožňující spouštět skripty příslušného jazyka
  - Mezi takové kontejnery můžeme řadit
    - Apache Tomcat, Jetty (oboje pro J2EE)
    - ASP.NET
    - PHP4, PHP5...
- využití
  - Webové aplikace se používají především všude tam, kde není efektivní nebo technicky možné nainstalovat desktop aplikaci
  - Rozvíjí se využití i v oblastech statického webu
  - Webové aplikace se používají i jako konfigurační nástroje s příjemným GUI, internetové obchody, aukční servery, diskusní fóra, sociální sítě, vnitropodnikové systémy a mnoho dalšího

## Technologie webových aplikací

### PHP (personal home page – hypertext preprocessor)

- Skriptovací programovací jazyk určený především pro programování dynamických internetových stránek a webových aplikací
- Skriptovací jazyk je vyšší programovací jazyk. Typicky není potřeba definovat proměnné, dynamická kontrola typů, zotavení z chyb, které neústí v ukončení skriptu...

- PHP skripty vykonávány na straně webového serveru a k uživateli je následně přenesen výsledek jejich činnosti. Syntaxe jazyk je inspirována jazyky Pearl, Java, Pascal. PHP je nezávislý na platformě, to znamená, že skripty se dají přenášet mezi různými operačními systémy bez jakýkoliv úprav.
- PHP podporuje mnoho knihoven pro různé účely – například zpracování textu či grafiky, práci se soubory, přístup k většině databázových systémů (MySQL, Oracle, PostgreSQL...) a podporuje celou řadu protokolů – http, smtp, ftp, imap...
- Jedná se o nejrozšířenější programovací jazyk pro web. Oblíbeným se stal díky jednoduchosti použití a bohaté zásobě funkcí.
- Pro weby je nejčastěji používán na serverech s operačním systémem Linux, databázovým serverem MySQL a webovým serverem Apache
- Významné projekty implementované v PHP:
  - phpBB, Wordpress, Taxy!, phpMyAdmin, Joomla, Meebo, PhpRS, PrestaShop
  - Frameworky: Zend, Nette
- Výhody PHP:
  - Specializované na webové stránky
  - Podpora databázových systémů
  - Multiplatformost
  - PHP je standart 7 obrovská podpora na hostingových serverech
  - Slušná dokumentace
- Nevýhody PHP:
  - Nekonzistentní pojmenování funkcí, nejednotné názvosloví
  - Ve standartní distribuci chybí debugovací nástroj
  - Při zpracovávání požadavku neudrží kontext aplikace, vytváří jej vždy znovu a oslabuje výkon

## JSP

- Zkratka JSP znamená Java Server Pages. Díky platformě Java je lze použít na všech podporovaných platformách (Linux, Windows, ...)
- Rozšíření JSF, více funkcí, podpora MVC atd...

```
<HTML<HEAD><TITLE>JSP stránka</TITLE></HEAD>
<BODY>
Dnes je <%= new java.util.Date() %>.
</BODY></HTML>
```

## ASP.NET

- Microsoft
- Součást .NET frameworku pro tvorbu webových aplikací a služeb.
- Nástupce ASP
- Konkurent JSP – java server pages
- Aplikace v ASP.net jsou rychlejší – jsou překompilovány do jednoho, či několika málo dll souborů. Na rozdíl od mnoha skriptovacích jazyků, kde jsou stránky při každém přístupu znovu a znovu parsovány.
- Ulehčuje programátorům přechod od programování klasických aplikací pro Windows do prostředí webu
- Hlavní rozdíl mezi ASP a ASP.NET je ten, že kód na stránce ASP.NET je kompilovaný. Vznikne tak kód, který je samozřejmě o mnoho rychlejší. Klasické ASP stránky pomocí

skriptů na straně serveru přímo generují HTML stránky, které se pošlou klientovi. Stránky se interpretují od začátku do konce bez možnosti ošetření vzniklých stavů a událostí. ASP.NET používají podobnou technologii oken, dialogů a formulářů jako běžné Windows aplikace. K jednotlivým vizuálním prvkům se vážou procedury pro ošetření stavů a událostí.

- Výhody:
  - Rychlý běh aplikace díky kompilovanému kódu
  - Většina chyb je zachycena už při vývoji
  - Bohatý výběr ovládacích prvků, knihoven a tříd zrychluje vývoj aplikací
  - Schopnost cachovat celou stránku – zvyšuje výkon serveru
- ASP.NET MVC umožňuje snadné využití MVC architektury
- NHIBERNATE umožňuje využívat objektově-relační mapování

## HTML

- Značkový jazyk
- Slouží ke statickému definování prvků, které se mají na webové stránce zobrazit
- Dnes velice populární HTML5 – oproti HTML4 nová struktura (footer, article atd..), podpora multimediálního obsahu
- Do HTML se běžně vkládá JavaScript
- JavaScript je programovací jazyk, který běží na straně klienta

## CSS

- Stylovací jazyk
- Přiřazuje jednotlivé vlastnosti konkrétním HTML třídám, prvkem atd...
- Dnes populární preprocesory, například SASS, LESS
- Preprocesory rozšiřují práci s CSS o funkce, proměnné, snadné vnořování stylů atd.

## Třívrstvá architektura

- Model třívrstvé architektury je pokračovatelem, z dnešního pohledu již zastaralé, dvouvrstvé architektury
- Každá vrstva poskytuje navenek určité rozhraní, přes které s ní může druhá vrstva komunikovat
- Není tedy žádný problém např. změnit poskytovatele datového úložiště
- Na rozdíl od dvouvrstvé architektury, klientovi není dovoleno přímo komunikovat s datovou vrstvou
- Použití třívrstvé architektury neznamená, že pro každou vrstvu musí být vyhrazen samostatný počítač
  - Není totiž vůbec výjimkou, kdy jsou všechny tři vrstvy provozovány na jednom počítači.
- Rozlišuje tyto vrstvy
  - Prezentační vrstva
    - Obsahuje funkce uživatelského rozhraní
    - Obvykle existuje několik prezentačních vrstev pro různé druhy zařízení, platformy a prostředí
  - Aplikační vrstva
    - Tvoří prostředníka mezi vrstvou prezentační a vrstvou datovou
    - Obsahuje tzv. business logiku aplikace



- V této vrstvě dochází k transformaci dat mezi vstupně / výstupními požadavky a datovou vrstvou
- Datová vrstva
  - Obsahuje funkce pro přístup k informacím v datovém úložišti
- Ve složitějších aplikacích je možné definovat i více než tři vrstvy
  - Vždy je ale od sebe odstíněna prezenční vrstva od vrstvy datové
  - Příkladem může být např. vrstva pro kontrolu přístupových práv a zabezpečení, vrstva pro správu systémových prostředků apod.
  - Na výše zmíněné vrstvy se dá ale také nahlížet jako na podvrstvy aplikační vrstvy z třívrstvé architektury, ale také jako na samostatné vrstvy
- S tím souvisí MVC
  - Architektura, která se skládá ze tří propojených částí
  - Model – data, většinou navázaná na databázi pomocí objektově-relačního mapování
  - View – pohled šablona statického kódu, do které jsou promítány konkrétní data
  - Controller – Propojuje Data a View – vybírá a posílá data do View, reaguje na uživatelské interakce

## Testování a vývoj

- Testování je důležité pro levnější a efektivnější vývoj
- Validace – Vytvořili jsme správnou aplikaci?
- Verifikace – Funguje aplikace správně?
- 1. Funkční testování – kontrola odkazů, formulářů, výskytu prvků
- 2. Testování použitelnosti – Interakce uživatele s prostředím, testování obsahu
- 3. Testování serverového rozhraní – kontrola komunikace mezi rozhraními, reakce na přerušení komunikace
- 4. Testování kompatibility – Z hlediska OS a prohlížeče
- 5. Výkonnostní testování
- 6. bezpečnostní testování – Zkouška neplatných přihlášení, detekce virů...
- Selenium IDE – nástroj od Firefoxu pro testování webových aplikací
- Spring poskytuje rozšířenou podporu testování
- Knihovna JUnit
- 
- Nástroje pro vývoj, testování a návrh
- Návrh – UML – viz otázka modelování
- Vývoj a testování - ASP.NET MS Visual Studio
- PHP, Java – Idea, Eclipse, NetBeans

## Zabezpečení

### Validace (a escapování)

- Ochrana před vložením nepřípustných dat
- Script (XSS – Cross-site scripting) / SQL Injection
- Lze ji provádět na
  - Úrovni klienta (JavaScript) – lze obejít
  - Úrovni serveru (PHP, Java Servlet) – měla by jít ruku v ruce s validací na klientovi

- Úrovní databáze (SQL omezení) – omezená uživatelská práva (pouze DML atd.)
- Ideálně v kombinaci především prvních dvou technik
- Ošetřit přípustné hodnoty se dají prostřednictvím regulárních výrazů

## Cross-site request forgery (CSRF)

- Využívá, že formulář může být odeslán odkudkoliv. Pokud je uživatel autorizován a např. v novém tabu otevře podvodnou stránku, ta může odesílat formuláře s jakýmkoliv daty do aplikace, kde je uživatel autorizován
- Řešeno HTTP hlavičkovým polem „referer“, který obsahoval zdroj dat, ale problémy s ad-blockem či různými pluginy na ochranu soukromí
- Nyní řešeno generovaným tokenem na jedno použití, který je skrytým polem formuláře a je odeslán společně s daty. V případě, že token neodpovídá, nejsou data z formuláře zpracována.

## Šifrování přenášených dat

- Zvýšení bezpečnosti pomocí šifrování přenášených dat (a využití certifikátů pro ověření identity) je jednou ze základních vlastností bezpečné aplikace
- Online bankovníctví, business komunikace, VPN přístupy...

## Ověření uživatele

- Přihlášení uživatele k aplikaci pomocí jména a hesla
- Informace o autentizaci uživatele je uložena do cookies, takže je uživatel přihlášen po celou dobu platnosti cookie v rámci přechodů mezi stránkami
- V db neukládání přístupových údajů v plain-textu a zahashované se solí

## Kontrola cookies

- Používání cookies (mimo potřeby autentizace) zpohodlňuje práci s aplikací, ale může být potencionálním rizikem
- Proto je vhodné je buď vůbec nepoužívat, nebo ověřovat obsah cookies před jejich použitím
- Např. ASP.NET šifruje autentizační cookie, aby se zabránilo manipulacím

# Základní algoritmy a principy počítačové grafiky

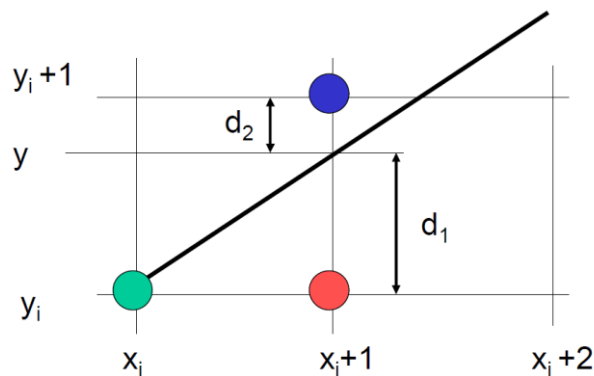
## Reprezentace obrazu

- Vektorová grafika
  - Pro technické výkresy, mapy, písmo
  - Obraz rozdělen na matematicky definované části – křivky, lomené čáry a jimi ohraničené stejnorodé plochy. Každé z těchto částí obrazu se přiřadí kódovaná informace o tom, jak má být zobrazena, jakou barvou, jak silnou čárou, zda jednoduchou či nějakou přerušovanou, tečkovanou, dvojitou, fousatou, čím má být uzavřená plocha vybarvena, zda jednoduše nebo nějakým vzorkem.
  - Jsou uloženy pouze souřadnice bodů, ze kterých prvek vychází, parametry matematické funkce, která určuje jeho průběh a údaje o způsobu zobrazení (barvě, síle a typu čáry, vzorku). Zobrazovací či tiskový program si vždy znovu z uložených dat tento grafický prvek vypočítá a zadaným způsobem zobrazí.
- Rastrová grafika (bitmapy)
  - Je popsán každý bod plochy svými parametry
  - Plocha je složena z bodů ("pixelů"), odtud bitová mapa
  - Problémem je velikost (absolutně závislá na velikosti obrázku) a fakt, že zvětšováním detailů obrázku se docílí pouze zhoršení kvality a zvětšování oněch bodů tak, jak ukazuje obrázek.

## Rasterizace úsečky

- Triviální algoritmus
  - Postupné procházení, pro každé x se vypočítá hodnota y, ta se zaokrouhlí a pixel se vykreslí
  - Problémy:
    - Násobení a sčítání v plovoucí čárce
- Midpoint
  - Rekurzivní dělení křivky na poloviny
  - Jednoduchá implementace pro všechny kvadranty
- DDA (Digital Differential Analyser)
  - Pro všechna x, se y získává přičtením k ( $k = (y_2 - y_1) / (x_2 - x_1)$ ) a pak se zaokrouhlí
  - Sčítání v plovoucí řadové čárce
- Kvadrantní DDA
  - V jednom kroku se postupuje o 1 pixel pouze ve směru x nebo y (ne v diagonálním)
  - Na základě chyby err se určí, kterým směrem jsme se vzdálili od úsečky
  - $err > 0$  – jsme nad, jdeme doprava
  - $err < 0$  – jsme pod, jdeme nahoru
  - Pro sklon blížíci se horizontálnímu udělá „schod“ hned na začátku místo uprostřed
- Bresenhamův algoritmus
  - Funguje na principu hledání nejbližších bodů, které leží k úsečce
  - $y = k(x_i + 1) + q$
  - $k = (y_2 - y_1) / (x_2 - x_1)$
  - $d1 = y - y_i$

- $d_2 = y_i + 1 - y$
- $dd = d_1 - d_2$      $dd > 0$  (horní bod);  $dd < 0$  (dolní bod)



## Metody vyplnění oblasti

- Seed-fill
  - Hranice není geometricky jasně definována, všechny informace o vyplňované oblasti se získávají čtením z obrazové, tedy rastrové paměti. Od semínka se postupně rozšiřuje prohledávání obrazové paměti a nalezeným vnitřním bodům se nastaví nová barva
  - Lze buďto rekurzivně do všech směrů ( $\rightarrow$  mnohonásobné testování) nebo po řádcích
- Scan-line
  - Každý řádek se projede a hledají se průsečíky s hranicí vyplňování
  - Průsečíky se seřadí a vyplní se oblast mezi lichým a sudým

## Zobrazovací řetězec

- Soubor činností postupně zpracovávající model v objektovém prostoru a vytvářející odpovídající obraz v obrazovém prostoru
- Transformace
  - Přechází se do homogenizovaných souřadnic = přidání čtvrté jednotkové souřadnice pro snadnější počítání – umožňuje vyjádření nejčastěji používaných lineárních transformací pomocí jediné matice
  - Tvaru tělesa – měřítko, deformace (vzhledem ke středu tělesa)
  - Pozice tělesa – posunutí, rotace (vzhledem k počátku)
  - Pohledu – umístění pozorovatele (pohledová matice) – poloha, směr pohledu (vektor  $v$ ), natočení kamery (up vektor  $u$ )
  - Zobrazovacího objemu – projekce, perspektiva
- Ořezání v homogenních souřadnicích
  - Aby se nezpracovávala část scény za pozorovatelem
  - Nutno ořezat trojúhelníky podle  $z$  (ss pozorovatele) po projekci uloženo v  $w$
- Dehomogenizace
  - Vydělení  $x', y', z'$  hodnotou  $w'$  vzniká dehomogenizovaný vektor  $x, y, z$
- Ořezání zobrazovacím objemem
  - Hodnoty v zobrazovacím objemu  $x < -1; 1$ ,  $y < -1; 1$ ,  $z < 0; 1$
  - Pokud alespoň část trojúhelníku zasahuje  $\rightarrow$  trojúhelník se kreslí a ořezává se až při rasterizaci
- Transformace do okna

- Přejít z  $[-1;1]$  do souřadnic celé obrazovky
  - Otočení y
  - Posun na střed
  - Zvětšení na velikost okna
- $x' = 0.5 * (w-1)(x+1)$
- $y' = 0.5 * (h-1)(1-y)$
- Rasterizace, viditelnost, obarvení, osvětlení, textury, stíny

## Určení viditelnosti


- Malířův algoritmus
  - Princip překreslování vzdálenějších ploch plochami v popředí
  - Setřídění objektů od nejvzdálenějších k nejbližším podle maximální z souřadnice
  - Nelze použít pro protínající se plochy
  - Možnost zacyklení, pokud se plochy vzájemně částečně překrývají
- Dělení obrazovky (Warnock subdivision)
  - Rozdělení obrazovky na díly, v případě, že do dílu zasahuje více překrývajících se objektů dále se rozdělí, klidně až na úroveň pixelů
- Vrhání paprsku (Ray casting)
  - Hledání nejbližšího objektu, který je zasažen paprskem vrženým od pozorovatele průmětnou
  - Rozšířením je sledování paprsku (Ray tracing)
    - Sledování nekončí po nalezení nejbližšího objektu, ale pokračuje dál (rozděluje se na přímo odražený, lomený a stínový – ke každému zdroji světla)
    - Pro realistické zobrazení scény
- Plovoucí horizont
  - Pouze pro graf funkce dvou proměnných
  - Je uchováván horní a dolní horizont
- Z-Buffer (depth buffer)
  - Dvojměrné pole o stejných rozměrech jako zobrazovací okno, pro každý pixel je v poli udržována hloubka (vzdálenost od pozorovatele) nejbližšího zobrazovaného bodu
  - Při vykreslování každého objektu je porovnávána tato hodnota pro každý vykreslovaný bod, pokud je bod blíže než hodnota v bufferu, je změněna hodnota v bufferu a pixel je vybarven
  - Výhody:
    - Vysoká rychlost
    - Snadná implementace
    - Správné řešení najde v každém případě hardwarová implementace
  - Nevýhody:
    - Vykreslování i neviditelných pixelů
    - Dříve: velká paměťová náročnost

## Určení viditelnosti

- Ploché stínování (flat shading)

- Určí se hodnota světlosti pro jeden celý polygon výpočtem z hodnot vrcholů polygonu. Každý polygon je pak nasvětlen po celé své ploše stejně a je tudíž snadno viditelný z jakých polygonů je objekt složený.
- Gourardovo stínování
  - Nepřesnosti u lesklých materiálů, rychlé a využívané u reálného vykreslování
  - Technika prolínání barev v polygonu (blending)
  - Pro každý vrchol je vypočtena průměrná normála podle normálových vektorů přiléhajících polygonů. Dále je podle osvětlení a těchto normál určena barva a intenzita pro každý vrchol. Finální osvětlení celého polygonu se získá výpočtem z hodnot na jednotlivých vrcholech
- Phongovo stínování
  - Pro každý pixel zvlášť → hladké přechody
  - Světlo na povrchu tělesa rozkládá do tří světelných složek:
    - Ambientní složky (ambient light)
      - Rozptýlené světlo v celé scéně
      - Nezávisí na úhlu pohledu ani směru a velikosti osvětlení
      - $I_A = I \cdot k_A$        $I$  – intenzita okolního světla;  $k_A$  – koeficient odrazu (0;1)
  - Difúzní složky (diffuse light)
    - Rozptýlené světlo materiálem (charakteristické pro matná tělesa), nezávislé na úhlu pohledu
    - $I_D = I_i \cdot k_D \cdot \cos(A)$
    - $I_i$  – intenzita světelného zdroje
    - $k_D$  – koeficient odrazu (0;1)
    - $\cos(A)$  – L,N skalární součin normovaných vektorů = úhel dopadu
  - Speculární (specular light)
    - Zrcadlová složka (odlesky)
    - Závisí na úhlu mezi odraženým vektorem a vektorem pozorovatele
    - $I_S = I_i \cdot k_S \cdot \cosh(B)$        $h$  – ostrost odrazu (jak malá plocha se bude lesknout)  $\geq 1$

## OpenGL

- Multiplatformní standart pro tvorbu počítačové grafiky, používá se při tvorbě počítačových her, programů, virtuální reality.
- Rozhraní na architekturu klient-server  program zadává dotazy a grafický adaptér je zpracovává
- Veškerá činnost OpenGL se řídí vydáváním příkazů pomocí volání procedur
- Primitiva jsou definována pomocí vrcholů
- Přiřazení barev, tloušťky, texturovací souřadnice, POV, osvětlení...
- Implementace OpenGL existují pro prakticky všechny počítačové platformy, na kterých je možno vykreslovat grafiku. Kromě implementací vestavěných v grafickém hardware (na grafické kartě) existují také softwarové implementace, které umožňují používat OpenGL i na hardwaru, který ho sám o sobě nepodporuje (ale obvykle nabízejí nižší výkon). Příkladem takové implementace je open source knihovna Mesa 3D, která ovšem z licenčních důvodů nemůže být označena jako implementace OpenGL, ale pouze jako implementace API, které je „velmi blízké“ OpenGL.

- Základní funkcí OpenGL je vykreslování do obrazového rámce (framebufferu). Umožňuje vykreslování různých základních primitiv (bodů, úseček, mnohoúhelníků a obdélníků pixelů) v několika různých režimech. Veškerá činnost OpenGL se řídí vydáváním příkazů pomocí volání funkcí a procedur (kterých OpenGL definuje cca 250). V OpenGL se nepoužívá objektově orientované programování. Jednotlivá primitiva jsou definována pomocí vrcholů – každý z nich definuje bod, koncový bod hrany nebo vrchol mnohoúhelníku. Každý vrchol má přiřazena data (obsahující souřadnice umístění bodu, barvy, normály a texturovací souřadnice).
- Soubor knihoven OpenGL byl navržen firmou Silicon Graphics (SGI) jako aplikační programové rozhraní ke grafickým akceleratorům, respektive k celým grafickým subsystémům
- Služby OpenGL jsou přístupné z většiny běžných programovacích jazyků a jejich vývojových prostředí – především C/C++, Javy, Pascalu/Delphi, Visual Basicu nebo Fortranu. Teoreticky je možné použít OpenGL na libovolný jazyk, ze strany jazyka je pouze potřeba podpora v dodatečných knihovnách. Nejčastější je použití OpenGL v jazycích C a C++, které je popisováno už samotným OpenGL standardem
- GLUT OpenGL Utility Toolkit – doplněk ke grafické knihovně OpenGL

# Základní zpracování obrazu

## Snímání

- Nejen optické snímání, ale i snímání rtg nebo ultrazvukem
- První krok při zpracování obrazu – obraz je nutné dostat do formy, kterou lze zpracovávat počítačem
- Převod optické veličiny na elektrický signál
- Vstupní informací při snímání nemusí být jen jas skeneru, ale například také intenzita rentgenového záření, ultrazvuk nebo tepelné záření
- Vzorkování – rozlišení (hustota vzorkování a rozmístění vzorků)
- Kvantová – počet snímaných úrovní, barevná hloubka

## Předzpracování

- Obraz může být následkem snímání zkreslen. Existuje ovšem velké množství metod, které usnadňují další analýzu obrazu, identifikaci objektů apod.
- Základní rozdělení metod předzpracování obrazu:
  - Jasové transformace
    - Práh – co je nad prahem → bílá, co je pod prahem → černá
    - Inverze – otočení
    - Okénko – práh s postupným náběhem
    - Nepravé barvy
  - Geometrické transformace
  - Filtrace a ostření
    - Průměrování – hodnota jasu je průměrem hodnot okolních
    - Konvoluční matice
    - Filtrace pomocí mediánu – dobře odstraňuje šum salt&pepper
    - Order static filtrace – rozšíření mediánového filtru, setřídění podle hodnot jasu, výběr min a max
    - Frekvenční filtry
      - Horní/dolní propust – co je pod/nad hranicí je ignorováno
      - Pásmová propust – propouští pouze vybrané složky prostorových frekvencí
    - Gradientní operátory
      - Sobelův / Robinsonův / Krischův / Laplaceův operátor
  - Úprava histogramu
    - Vyrovnání (korekce pře a pod exponování)
    - Vyhlazení – v případě, že je hodně lokálních minim a maxim vyhlazení váženým průměrem okolí
- Operaci na úrovni pixelů

## Segmentace

- Jeden z nejtěžších kroků při zpracování obrazu
- Analýza vedoucí k nalezení objektů v obraze. Objekty jsou části obrazu zajímavé pro další zpracování.



- Cílem segmentace je rozdělení obrazu do částí odpovídajících předmětům a oblastem reálného světa
  - Kompletní segmentace
    - Vyšší úroveň zpracování, založena na znalostech řešeného problému
    - Všechny segmenty jednoznačně korespondují s objekty reálného světa
  - Částečná segmentace
    - Založena na principu homogenity obrazových vlastností (např. jas, barva) uvnitř segmentu
    - Všechny vytvořené segmenty nemusí přímo souhlasit s reálnými objekty
- Typickým cílem segmentace obrazu je identifikace popředí a určení oblastí v obraze odpovídajícím významnému prvku zachycené scény
- Výsledky segmentace jsou využitelné například v počítačovém vidění, zpracování lékařských obrazových dat (Medical Imaging) nebo při analýze obrazů získaných při dálkovém průzkumu Země.
- Prahování (angl. Thresholding)
  - Je nejjednodušší metoda segmentace obrazu založená na hodnocení jasu každého pixelu. Jejím principem je nalezení takové hodnoty (prahu) v histogramu, pro kterou bude platit, že všechny hodnoty jasu nižší, než práh odpovídají pozadí, zatímco všechny hodnoty vyšší, než práh odpovídají popředí
  - Často je pro každou část obrazu využit jiný práh
  - Metody určení prahu
    - Pevné, známé hodnoty – např. medicínské scanery (CT, MRI, ...)
    - Procentní prahování – z histogramu určíme hodnotu prahu podle procentuálního pokrytí
    - Analýza histogramu – hodnoty mezi vrcholy určuje hodnotu prahu (např. minimum)

## Klasifikace

- Objektů nalezených v obraze do předem známých tříd
- Popis objektů obrazu
  - Kvantitativně – číselné charakteristiky
  - Kvalitativně – relace mezi objekty, reprezentace grafem
- Algoritmus barvení – postupné procházení binárního obrazu po řádcích a každému nenulovému elementu se přiřadí hodnota → každá nová barevná oblast má novou hodnotu
- Skalární popisy oblastí – na základě těchto popisů se vytváří etalony, podle kterých se v obraze hledají objekty
  - Velikost – počet obrazových elementů, lze uvažovat skutečná velikost elementů (plocha)
  - Eulerovo číslo  $E$  – rozdíl počtu souvislých částí  $S$  a počtu děr  $A$ ;  $E = S - A$
  - Projekce, výška šířka
  - Podlouhlost – poměr výšky a šířky opsaného obdélníka
  - Pravoúhlost – poměr velikosti oblasti a plochy opsaného obdélníka ve směru  $k$
  - Nekompaktnost – poměr kvadrátu délky hranice a velikosti
  - Těžiště objektu
  - Výstřednost – poměr délek nejdelších na sebe kolmých tětiv

## Formáty pro ukládání rastrového obrazu

- Parametry:
  - Formát uložení barev – omezená množina barev, barevná paleta, true color, odstíny šedi, B/W
  - Komprese – ztrátová, bezztrátová
  - Kritéria komprese – kompresní poměr, kvalita rekonstrukce, cena za kompresi (paměťová a časová náročnost algoritmu – různé pro kompresi a dekompresi).
  - Rozklad obrazu – prokládání, skládání obrazu
  - Negrafické informace – závislost na HW (přenositelnost), popis obrazu, řízení způsobu vykreslování, způsob vytvoření (EXIF)

### GIF (Graphics Interchange Format)

- LZW bezztrátová komprese (12-ti bitová)
- Grafický formát určený pro rastrovou grafiku
- Vhodný pro uložení tzv. pérovek (nápis, plánky, loga)
- Maximální počet současně použitých barev palety je 256 (8 bitů), v případě animace pak umožňuje využít odlišné palety 256 barev pro každý snímek. Toto omezení nemá formát PNG, který se hodí ke stejným účelům jako GIF a nabízí pro většinu obrazů výrazně lepší kompresi. Formát PNG však neumožňuje animace (ty umožňuje až APNG a MNG).
- Verze 87a – prokládání obrazu v pruzích, umístění bloků na pozadí.
- Verze 89a – doprovodný text vypisovaný na obrazovku, řízení grafiky (průhlednost, animace)
- Firma Unisys vydala později patent na LZW kompresy a tím pohřbila GIF formát

### PNG (Portable Network Graphics)

- Primárně určen pro internet.
- Reakce na licencování LZW algoritmu a GIF formátu. Bezztrátová komprese, nezávislá na platformě. Barvy RGB, index do palety.
- Nemá omezení v paletě barev – na rozdíl od gifu nabízí barvy 24bitové hloubky
- Alfa kanál – průhlednost
- Vlastnosti srovnatelné s GIF: bezztrátová komprese LZ77 + Huffmanovo – lepší než GIF
- Prokládání (progresivní mód), průhlednost, textová informace v obraze
- Nové vlastnosti: True Color paleta, grayscale paleta, alfa kanál – průhlednost, gamma korekce (věrné zobrazení), detekce poškozených dat při přenosu, rychlejší zobrazení náhledu v progresivním průchodu, různé komprese a kompresní filtry.
- Skládá se ze shluků (chunk). Povinné chunks – hlavička (délka, typ, CRC), paleta, obrazová data, zakončení. Pomocné chunks – průhlednost, gamma korekce, barevný rozsah, sRGB prostor, ICC profil, text (iso-8859-1), barva pozadí, histogram, pozice obrázku na stránce...
- Lepší pro obrázky obsahující text, čárovou grafiku, čisté barevné plochy a ostré rozhraní barev
- Fotky by se měly před finální úpravou ukádat do png, po finální úpravě do jpg pro minimální ztráty
- Může obsahovat metadata

## JPEG File Interchange Format

- Pro ukládání obrázků ve fotorealistické kvalitě – je ztrátový, ale stále zachovává dobrou kvalitu obrazu
- Ideální pro publikování obrázků a fotografií na webu
- Nevhodný pro kresbu, křivkové obrázky, texty v obrázku
- Postup komprese: transformace předlohy do optimálního barevného prostředí (YCBCR), redukce barev (vzorkování komponent průměrnými skupinami pixelů), zavedení DCT (diskrétní kosinová transformace) na blok pixelů 8x8, kvantizace DCT koeficientů, cik-cak výběr výsledných koeficientů a jejich kódování Huffmanovým kódováním.
- Podporuje metadata EXIF, které mohou obsahovat informace o fotografii: značka a model fotoaparátu, datum a čas pořízení snímku, nastavení fotoaparátu – clona, blesk, citlivost, náhled, GPS, komentáře a info o autorovi
- JPEG2000 – kompresní algoritmus založený na vlnkové transformaci (bez čtverců 8x8). Lepší, rychlejší a kvalitnější komprese. Větší odolnost vůči chybám v datovém toku. Možnost zpracovávat obrázky větší než 64tisx64tis pixelů. Kvalitní zpracování počítačově generované grafiky s ostrými přechody. Možnost využití různých barevných módů. Progresivní přenos. Ukládání metadat.

## TIFF (Tag Image File Format)

- Vysoká univerzálnost (různé kódování bitů, barevné modely, druhy komprese – RLE, LZW, JPEG).
- Různé kategorie datových položek – velikost, způsoby reprezentace, pomocné – jméno autora, datum, vlastní data – pruhy, interpretace – šedotónový, RGB, binární maska
- Použití v DTP, zpracování obrazu

## Komprese

### Bezztrátová komprese

- Zachovává věrnost obrazu (medicína, geografie), má nižší kompresní poměr (většinou 3:1, ale záleží na povaze dat), odstraňuje redundance dat.
- RLE (Run Length Encoding)
  - Pixely stejné barvy efektivně komprimuje jako četnost znaku a jeho hodnotu
  - Aplikace algoritmu:
    - Horizontální, vertikální, úhlopříčkové
    - Binární, bytová, pixelová, řádková úroveň

### LZW (Lempel, Ziv, Welch)

- Substituční algoritmus, slovníková metoda komprese (slovník vytvářen při kódování i dekódování – není nutno ho přenášet)

### CCITT (Huffmanovo kódování)

- Statistická metoda komprese – zjišťuje četnost výskytu hodnot, použití prefixového kódu, každá hodnota je reprezentována unikátní bitovou kombinací

### Ztrátová komprese

- Vysoká komprese (až 50:1). Dochází k redukci barev a ztráty přesnosti geometrických tvarů.

- Princip – transformace obrazu do prostoru atributů (DCT, FFT, ...). Fraktálová komprese (soběpodobnost, transformace)
- Redukce počtu barevných odstínů
- Diskrétní kosinová transformace (transformace obrazového prostoru do prostoru kosinových funkcí)
- PCX

## Barevné modely

- Člověk dokáže rozlišit více než  $4 \times 10^5$  barev a jejich odstínů. Existují tzv. komplementární barvy, jejichž složením vznikne bílé světlo.
- Barevné modely určují, ze kterých základních barev se budou ostatní skládat, jaký bude poměr jednotlivých základních barev a jakým způsobem se budou základní barvy míchat.

## Aditivní barevný model

- Barvy se přidávají do černé
- Typický aditivní model je RGB – červená, zelená, modrá
- Používaný v monitorech a projektorech
- Před tiskem RGB obrázku je tedy nutné ho převést do barevného prostoru (režimu) CMYK
- Čím více barev se přidá, tím více se blíží bílé
- Aditivní barevné prostředí nepotřebuje vnější světlo

## Subtraktivní barevný model

- Základní barvy se odčítají od bílého světla, čím větší odečet, tím více se blíží černé
- Typický subtraktivní model je CMY (cyan, magenta, yellow)
- Subtraktivní prostředí je prostředí, které odráží světlo, a proto potřebuje vnější zdroj světla. Jakákoliv předloha zobrazená na papíře je příkladem subtraktivního modelu.
- Tento model používají například tiskárny.
- V tiskárnách se také používá model CMY (K) – „K“ značí „black“ – pro úsporu barevných tiskových kazet při černém tisku.

## HSV

- HSV – Hue (odstín), Saturation (sytylost), Value (intenzita (jas))
- Odpovídá více lidskému vnímání barev
- Jde o jeden z modelů orientovaných na uživatele, které používají intuitivní fyzikální veličiny — odstín (základní spektrální barva), sytylost (poměr čisté barvy a bílé), intenzita (jas)
- Odstín – základní spektrální barva
- Sytylost – množství bílé v barvě – 0 % bílé = 100% sytylost. 50 % bílé v červené barvě => růžová
- Intenzita – stupeň zářivosti barvy – kolik světla odráží. Snížení jasu si lze představit přidáním černé barvy do základní spektrální barvy.

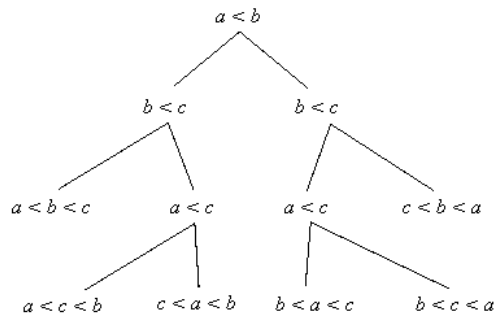
# Algoritmy pracující s grafy

## Základní pojmy

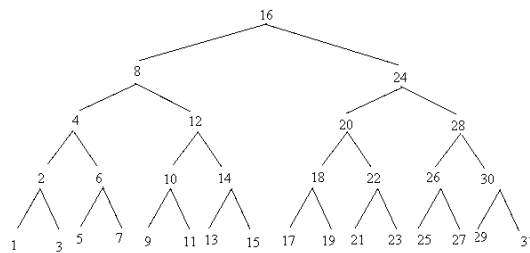
- Slovo graf se používá v různých významech
- V matematice také existuje pojem grafu, který nemá takměř nic společné se známými grafy funkcí, až na název, který je odvozený z řeckého slova "grafein", co znamená "psát"
- Pojem "graf" vykrystalizoval jako matematický objekt (univerzální zobrazovací technika), který se skládá z prvků dvojitého druhu – z vrcholů a hran
- Obyčejný graf  $G$  rozumíme uspořádanou dvojicí  $(V,E)$ , tzn.  $G = (V,E)$ ,
  - $V$  je libovolná množina vrcholů
  - $E$  je podmnožina hran
- Graf, který neobsahuje kružnici, nazýváme „lesem“
- Souvislý graf, který neobsahuje kružnici, nazýváme „stromem“
- Strom, který obsahuje jen jeden vrchol, nazýváme stromem triviálním
- Strom je komponenta lesa
- Vrchol stromu stupně jedna nazýváme „listem stromu“
- Vrchol stromu stupně většího než jedna nazýváme „vnitřním vrcholem“ stromu
- Sled v grafu  $G$  je konečná posloupnost  $S=(v_0,e_1,v_1,e_2,\dots,v_{n-1},e_n,v_n)$ , v které se střídají vrcholy a hrany, a která se začíná a končí ve vrchole, přičemž pro všechny  $i=1,2,3,\dots,n$  je  $e_i(v_{i-1}, v_i)$ 
  - V sledu se hrany a vrcholy mohou opakovat
  - Číslo  $n$  nazýváme délkou sledu  $S$
  - Sled nazýváme uzavřený, když jeho začátek a konec jsou totožné
    - V opačném případě ho nazýváme otevřený
- Tah je také sled, v kterém se žádná hrana nevyskytuje dvakrát
  - Otevřený tah, v kterém jsou všechny vrcholy navzájem různé, se nazývá cesta
  - Počet hran v cestě se nazývá délka cesty
- Strom je souvislý graf neobsahující kružnici (acyklický souvislý graf)
- Každý netriviální strom obsahuje list
- Odebereme-li ze stromu list, dostaneme opět strom
- Přidáme-li k vrcholu stromu list, dostaneme zase strom

## Využití grafů – stromů

- Stromy, i když mají jednoduchou strukturu, jsou velmi často používané
- V současnosti jsou používané velmi běžně například v informatice
- Typickým použitím jsou úkoly na třídění, vyhledávání a rozhodování
- Příklad
  - Nechť např. jméno  $X$  je slovo složené z písmen  $x_1x_2x_3\dots x_T$
  - Postupujeme tak, že nejdříve najdeme jména začínající na  $x_1$ , pak mezi nimi, kde se nachází druhé písmeno  $x_2$ , atd.
  - Využijeme toho, že účastníci telefonního seznamu jsou vedeni v seznamu podle abecedy
  - Toto uspořádání je dané lexikografickým uspořádáním slov, založeném na přirozeném uspořádání písmen  $a, b, c, \dots$
- Třídění



- Vyhledávání



- Graf  $G$  je souvislý právě tehdy, když obsahuje kostru
  - Kostra grafu  $G=(V,E)$
  - Je strom  $T=(V,E')$ ,
  - Acyklický a souvislý faktor grafu  $G$
  - Je minimální souvislý podgraf grafu  $G$  obsahující všechny jeho vrcholy
  - Je to strom rozpínající se do všech vrcholů daného grafu
    - Graf je souvislý právě tehdy, obsahuje-li alespoň jednu kostru
  - Obsahuje-li souvislý graf právě jednu kružnici délky  $k$ , pak obsahuje  $k$  koster
  - Obsahuje-li souvislý graf jednu kostru, pak je stromem. Počet koster úplného grafu  $K_n$  je  $n-2$
- Borůvka
  - Hrany grafu jsou ohodnoceny číslem rovnajícím se velikosti nákladů na výstavbu trasy
  - Hledá se podgraf  $T$  (multi)grafu  $G$ , který neobsahuje kružnici, je souvislý a součet ohodnocení hran patřících podgrafu je nejmenší možný
  - $T$  má tedy vlastnosti minimální kostry

## Nalezení minimální kostry

### Obecný algoritmus

- Všechny metody popsány jako proces barvení hran grafu, na počátku je každá hrana neobarvená
- Pravidlo řezu
  - Zvolíme lib. řez  $R_x(G)$ , který neobsahuje modrou hranu
  - Mezi neobarvenými hranami řezu vybereme tu, která má minimální ohodnocení a obarvíme ji modře
- Pravidlo kružnice
  - Zvolíme libovolnou kružnici, která neobsahuje žádnou červenou hranu

- Mezi neobarvenými hranami kružnice vybereme tu, která má maximální ohodnocení a obarvíme ji červeně.
  - V každém kroku algoritmu, dokud v grafu  $G$  existuje neobarvená hrana, použijeme jedno z předchozích dvou pravidel
- Modré hrany tvoří minimální kostru grafu  $G$

## Jarníkův algoritmus

- V každém kroku přidává k modrému stromu  $T$ , který na počátku obsahuje pouze vrchol  $v$ , hranu s minimálním ohodnocením, jejíž jeden vrchol leží ve stromu  $T$  a druhý v něm neleží
- V každém kroku, dokud strom  $T$  neobsahuje  $(n-1)$  hran, je k němu přidána jedna hrana (list), o kterou se tento strom zvětší
- Znamená to, že je vybírán takový řez, jehož množinu  $X$  tvoří množina vrcholů modrého stromu  $T$

## Kruskalův algoritmus

- Postupně zkoumá hranu po hraně (hrana je neobarvená, do modrého stromu ještě nezařazená) a rozhoduje o jejím zařazení do minimální kostry daného grafu
- Rozhoduje, zda ji obarvit modře či nikoli
- Modře obarví pouze tu hranu, která spojuje dva různé modré stromy, a tak nevytvoří s modře obarvenými hranami kružnici
- Vzhledem k počátečnímu uspořádání hran zařazuje do modrého lesa ty hrany, které mají co možná nejmenší ohodnocení
- Tímto způsobem nakonec vytvoří minimální kostru daného grafu
- v tomto algoritmu je v každém kroku, ve kterém zkoumaná hrana má své koncové vrcholy v navzájem různých modrých stromech, vybírán řez s množinou vrcholů  $X$  rovnou množině vrcholů jednoho (libovolného) z těchto dvou stromů
- Kruskalův i Jarníkův algoritmus v každém kroku, v kterém dojde k obarvení hrany modře, spojuje právě dva nejbližší modré stromy v jeden modrý strom
- U algoritmu KA jsou tyto dva stromy určeny vkládanou hranou (v pořadí daném ohodnocením hran)
- U Jarníkova algoritmu je spojován stále se zvětšující jeden modrý strom obsahující počáteční vrchol  $v$  s nejbližším triviálním modrým stromem, vrcholem dosud nezařazeným do stromu  $T$
- Kdežto podle Borůvkova algoritmu
  - Dochází v každém kroku ke spojení všech navzájem si nejbližších modrých stromů
  - Ke spojení každého modrého stromu  $T^*$  s tím modrým stromem, ve kterém leží druhý koncový vrchol hrany, mající nejmenší ohodnocení ze všech hran, které se stromu  $T^*$  dotýkají
  - V algoritmu BA ve vztahu k algoritmu OA je v každém kroku zvoleno hned několik řezů, pro každý zkoumaný modrý strom vždy právě jeden
- Graf  $G = (V, E)$  nazveme bipartitní
  - Jestliže vrcholy množiny  $V$  můžeme rozdělit do dvou množin  $V_1$  a  $V_2$  tak, že každá hrana grafu  $G$  má jeden koncový vrchol v množině  $V_1$  a druhý v množině  $V_2$

- Je bipartitní, když neobsahuje kružnici liché délky

## Využití grafů při řešení problémů – úloh

- Vytvoření vnitřního strojového modelu prostředí (světa)
- Každému modelu odpovídá stav prostředí
- Množina všech stavů - stavový prostor
- Lze reprezentovat orientovaným grafem
- Stav – reprezentován uzlem grafu
- Přejít mezi stavy – reprezentován orientovanou hranou
- Řešení problému
  - Hledání přijatelné cesty v orientovaném grafu mezi uzlem specifikovaného počátečního stavu a uzlem specifikovaného cílového stavu
- Hledání ve stavovém prostoru
  - Specifikovat množinu pravidel opisujících přípustné akce -> operátory
    - Čili výpočetní prostředky, které umožňují transformaci jedné struktury symbolů (reprezentující každý objekt prostoru) na jinou tak, abychom postupně mohli prohlédnout celý prostor
    - Efektivní řídicí strategie, podle které se provádějí transformace tak, aby byl hledaný objekt nalezen co nejrychleji
- Řídicí strategie
  - Systematické prohledávání stavového prostoru může být velice neefektivní
  - Prohledávací algoritmy
    - Neinformované
      - Do šířky
      - Do hloubky
    - Informované
      - Dle charakteru úlohy je v nich možné definovat hodnotící funkci  $f$ , která pro každý uzel stromu řešení určí jeho ohodnocení
      - Hodnoty hodnotící funkce se pak používají k výběru uzlu pro expanzi, budou vždy expandovány “nejperspektivnější” uzly a zabrání se prohledávání cest, které nevedou k cíli

## Algoritmy prohledávání

- Řeší problémy související s
  - Průchodem
  - Prohlédnutím
  - Vyhledáním
  - Kontrolou apod. jistých míst.
- Lze tyto algoritmy sestavit pomocí jednoduché modifikace Jarníkova algoritmu
- Nejprve nás bude zajímat systematický průchod a zpracování (tj. provedení požadovaných operací) všech vrcholů daného grafu
  - V každém kroku Jarníkova algoritmu zvětšujeme modrý strom o právě jedinou hranu a jediný vrchol
  - Toto přidávání můžeme vnímat jako systematické zpracování jednotlivých vrcholů grafu, ovšem v pořadí, které je závislé na ohodnocení hran grafu
  - Představme si tedy činnost uvedeného algoritmu na grafu, jehož všechny hrany jsou ohodnoceny stejným číslem, např. jedničkou



- V tom případě je to totéž, jako kdybychom pracovali s grafem neohodnoceným, přičemž v každém kroku můžeme volit libovolnou z hran modrého stromu se dotýkajících
- Abychom výběr hrany (a tím i vrcholu), o kterou modrý strom rozšíříme, upřesnili, stanovíme ještě jistá pravidla
- Pravidla, která nám umožní formulovat dva nejčastěji používané algoritmy prohledávání grafu, a to prohledávání do hloubky a prohledávání do šířky
- Pravidlem pro prohledávání do hloubky (resp. do šířky)
  - Bude, že modrý strom v každém kroku zvětšíme o hranu, která se jej dotýká ve vrcholu, který byl zařazen do modrého stromu jako poslední (resp. jako první) ze všech, které jsou v daném kroku koncovými vrcholy hran dotýkajících se modrého stromu
  - K tomu, abychom takový vrchol jednoduše určili, využijeme datových struktur zásobník pro prohledávání do hloubky a frontu pro prohledávání do šířky

## Zásobník

- Název datové struktury vhodný pro ukládání záznamů
- Provádíme operaci
  - Vložení nového záznamu za záznam ležící na posledním místě v zásobníku
  - Určení záznamu ležícího na posledním místě v zásobníku
  - Odebrání záznamu ležícího na posledním místě v zásobníku
- Pro zásobník se též běžně používá označení LIFO, odvozené ze slov "last in first out"
  - Naznačují provádění zmíněných operací

## Fronta (FIFO, "first in first out")

- Název datové struktury vhodný pro ukládání záznamů, s kterou provádíme operaci
  - Vložení nového záznamu za záznam ležící na posledním místě ve frontě
  - Určení záznamu ležícího na prvním místě ve frontě
  - Odebrání záznamu ležícího na prvním místě ve frontě

## Ilustrace algoritmu prohledávání do hloubky

- Začneme-li prohledávání do hloubky ve vrcholu c a použijeme-li při výběru hrany {x, y} určené k obarvení modře lexikografické pravidlo (tj. pokud existuje více hran dotýkajících se modrého stromu ve vrcholu x, volíme vrchol y vzhledem k abecednímu pořadí koncových vrcholů těchto hran), budou vrcholy zpracovány v pořadí c, a, d, b, e, g, f,
- Přičemž bude konstruován modrý strom
- Tzv. strom prohledávání do hloubky
- Jak naznačuje název tohoto způsobu prohledávání, jeho základní strategie spočívá ve snaze postupovat co „nejhlouběji do grafu, jak je to jen možné
- Algoritmus prohledávání do hloubky (angl. depth-first search, zkrácene DFS) probírá hrany vycházející z posledně nalezeného uzlu v, který má ještě nějaké neprobrané hrany
- Když probere všechny jeho hrany, vrátí se zpátky k uzlu, z něhož se do uzlu v dostal, a z něho pokračuje po další dosud neprobrané hraně
- Takovým způsobem se postupuje tak dlouho, dokud se neobjeví všechny uzly dosažitelné z prvního výchozího uzlu

- Pokud zbývá nějaký neobjevený uzel, zvolí se jako další výchozí uzel, a prohledávání do hloubky pak pokračuje z něho
- Tento postup se opakuje tak dlouho, až jsou objeveny všechny uzly
- Podobně jako při hledání do šířky uchovávají se i při hledání do hloubky o každém uzlu určité pomocné údaje, namísto fronty otevřených uzlu je zde ale (implicitní) zásobník zajišťující implementaci rekurze
- Při objevení nového uzlu  $v$  jako následníka uzlu  $u$  se stejně jako u hledání do šířky přiřadí  $p[v] := u$ .
- Vzhledem k opakované volbě výchozího uzlu z dosud neobjevených uzlu je po skončení prohledávání do hloubky prostřednictvím odkazu  $p[v]$  definován podle prohledávaného grafu tvořený obecně několika stromy
- Tento  $p$ -podle budeme nazývat DF-lesem a každou jeho komponentu DF-stromem
- Algoritmus opět rozděluje uzly do tří skupin
  - Nové (dosud neobjevené)
  - Otevřené
  - Uzavřené
- Otevřeným se uzel stane po svém prvním objevení, uzavřený je poté, co byl kompletně prozkoumán jeho seznam sousedu
- Vedle toho se každému uzlu  $u$  přidělují dvě časové značky uchovávané v poli  $d[u]$  a  $f[u]$ :
  - Značka  $d[u]$  odpovídá okamžiku objevení uzlu  $u$  (t.j. okamžiku jeho otevření)
- Značka  $f[u]$  odpovídá jeho uzavření (tj. okamžiku skončení průchodu jeho seznamu sousedu)
- Tyto značky vypovídají nejen o průběhu prohledávání, ale odrážejí i radu vlastností grafu, a tak se často používají v dalších grafových algoritmech
- Hodnotami značek jsou přirozená čísla od 1 do  $2|U|$ , neboť pro každý uzel nastává právě jednou jeho objevení, a právě jednou se uzavře
- Po přidělení určité hodnoty se značkovací čítač zvýší o 1, a tato hodnota bude přidělena příště
- Z tohoto způsobu přidělování značek plyne, že platí  $d[u] < f[u]$  pro všechny uzly  $u \in U$ . (4.20)

## Ilustrace algoritmu prohledávání do šířky

- Začneme-li prohledávání do šířky ve vrcholu  $c$  a respektujeme-li již zmíněné lexikografické pravidlo, budou vrcholy zpracovány v pořadí  $c, a, d, f, e, b, g$ ,
- Přičemž bude konstruován modrý strom
- Tzv. strom prohledávání do šířky
- Představuje základní variantu postupu prohledávání
- Z jeho hlavních myšlenek vycházejí další důležité grafové algoritmy
- Např. Dijkstruv algoritmus hledání nejkratší cesty nebo Jarníkův na hledání minimální kostry grafu
- Pro zadaný graf  $G = (H, U)$  a vyznačený uzel  $s$  se prostřednictvím hledání do šířky systematicky prověřují hrany grafu  $G$  s cílem „nalézt každý uzel, který je dosažitelný z uzlu  $s$ “
- Prohledáním do šířky se současně stanoví vzdálenost od  $s$  ke každému dosažitelnému uzlu
- Jako vedlejší produkt se tak získá strom hledání do šířky obsahující všechny z  $s$  dosažitelné uzly

- Pro každý uzel v obsažený v tomto stromu – nadále jej pro stručnost budeme označovat jako BF-strom – je cesta z kořene s do uzlu v zároveň nejkratší cestou z s do v grafu G
  - BF-strom je tedy současně stromem nejkratších cest z uzlu s do všech dosažitelných uzlu
  - V popsané podobně funguje algoritmus prohledávání do šířky pro neorientované i pro orientované grafy
  - Název tohoto způsobu prohledávání vyjadřuje uniformitu postupu, při kterém se hranice mezi objevenými a (dosud) neobjevenými uzly rovnoměrně posouvá na celé své šířce
  - To má za následek, že algoritmus objeví všechny uzly ležící ve vzdálenosti k od uzlu s dříve, než první uzel ve vzdálenosti k + 1
  - V průběhu prohledávání je každý algoritmem zpracováváný uzel označen nejprve jako FRESH, potom jako OPEN, a nakonec jako CLOSED
  - Na začátku jsou všechny uzly nové (FRESH)
  - Jakmile algoritmus poprvé narazí při prohledávání na určitý uzel, označí jej jako otevřený (OPEN)
  - Jako uzavřený (CLOSED) pak označí
  - Každý uzel, který už nepatří do hranice mezi objevenými a neobjevenými uzly
  - Tedy uzel, který už zaručeně nemá žádného nového souseda
- 
- Z uvedených ilustrací je patrné, že pro tentýž graf jsme dostali při prohledávání do hloubky jiný modrý strom než ten, který vznikl v průběhu prohledávání do šířky
    - Oba modré stromy mají, vzhledem k hranám do modrého stromu nezařazených, své specifické vlastnosti, jichž je s výhodou využíváno při formulaci dalších algoritmů
    - Např. strom prohledávání do šířky využijeme ke zjištění, zda je graf bipartitní
    - Nebo tento typ prohledávání využijeme pro nalezení cesty s minimálním počtem hran z jednoho vrcholu do druhého