PT

1 Úvod do technologie programování a programovacích stylů, OO návrh, základní UML diagramy, psaní programů v Javě

- **Úvod**
  Programování je proces vytváření software s požadovanými vlastnostmi a funkcemi. Software se vytváří jako program v programovacím jazyce. Dříve se využívalo *procedurálních* programovacích jazyků (Fortran, ANSI C), dnes moderní vyšší programovací jazyky založené na *objektových principech* (Java, C#, C++, Object Pascal). Nejprve rovnou kódování spojené s testováním, později (zejména po příchodu OO jazyků) proces rozdělen na více fází – *analýza* (shromažďování požadavků na software), *návrh* (vytvoření modelu SW podle požadavků), *kódování* (implementace modelu v programovcím jazyce) a *ladění* (hledání a odstraňování chyb). Pro rozsáhlé projekty často používána hotová řešení, např. Rational Unified Process (RUP).

- **Styly programování**
  - vodopád ... fáze procesu oddělené, nepočítá se s rozšiřováním a úpravami, nejstarší
  - evoluční ... začíná se s jednoduchou verzí, požadavky na další vlastnosti až během vývoje
  - formální ... přesná specifikace (matematická) transformována na SW ⇒ odpovídá specifikaci
  - komponentový ... integrace hotových komponent (třeba i koupených jinde)
  - iterativní ... od nuly, v iteracích se přidávají funkce, SW neustále ve funkčním stavu

- **Cíle OO návrhu**
  - robustnost ... schopnost zpracovat i nestandarní vstupy (blbuvzdornost)
  - přizpůsobivost ... počítat se snadným portováním na jiné platformy
  - znovupoužitelnost ... stavět program z nezávislých komponent použitelných i později

- **Principy OO návrhu**
  - abstrakce ... rozdělit SW na základní části a ty implementovat zvlášť (např. ADT)
  - zapouzdření ... ukrýt implementační detaily a poskytnout univerzální rozhraní
  - modularita ... skládat SW z modulů, které mohou být znovupoužitelné
  - dědičnost ... vytvořit generické třídy a od nich odvozovat specializované
  - polymorfismus ... stejné metody pro různé objekty

- **Základní UML diagramy**
  - atributy a metody ... `/` pouze pro čtení, `+` veřejný, `#` chráněný, `-` soukromý, `in` & `out` argumenty
  - třída ... název tučně, pod ním atributy, pod nimi metody
  - objekt ... název:třída
  - dědičnost ... jednoduchá, vícenásobná, dělení potomků – úplné, neúplné, překrytí
  - asociace ... 0..1, 1..1, 0..*, 1..*
  - kompozice, agregace (asociace celku a částí)

- **Psaní programů v Javě**
  - Návrh ... cílem vytvořit třídy – popsat jejich atributy a metody, např. CRC karty (Class-Responsibility-Collaborator) nebo UML (Unified Modeling Language)
  - Kódování ... na základě návrhu implementovány třídy a jejich metody
  - Ladění ... kontrolní výpisy, sledování obsahu proměnných apod.
  - Testování ... kontrola běhu programu s různými vstupy, zejména nestandardními, pro složitější SW – specializované nástroje (JUnit apod.)
  - Dokumentace ... dodržovat zavedené principy během kódování (štábní kultura), dokumentační komentáře – nástroj Javadoc

2 ADT zásobník, fronta, seznam, řada, vektor a jejich implementace

- **Zásobník** (LIFO)
  Použití k ukládání návratových adres podprogramů, undo, historie web browseru, zpracování postfixových notací. Metody `push` (vložení), `pop` (výběr), `top` (přečtení posledního prvku), `size`, `isEmpty`. Implementace polem – přímá nebo zpětná (index vrcholu roste resp. klesá), spojovým seznamem.

- **Fronta** (FIFO)
  Použití v systémech hromadné obsluhy, tisková fronta, BFS. Metody `enqueue` (vložení), `dequeue` (výběr), `front` (přečtení prvního prvku), `size`, `isEmpty`. Implementace polem (cyklický buffer), spojovým seznamem.

- **Obousměrná fronta**
  Použití jako fronta i zásobník zároveň, k odvození jiných ADT. Metody `insertFirst`, `insertLast`, `removeFirst`, `removeLast`, `size`, `isEmpty`, `first`, `last`. Implementace obousměrným spojovým seznamem.

- **Vektor**
  Funguje jako pole, ale nemá pevnou délku. Metody `elemAtRank`, `replaceAtRank`, `insertAtRank`, `removeAtRank`. Implementace polem proměnné délky (při překročení překopírování do většího pole).

- **Seznam**
  Základ pro jiné ADT. Metody `first` (první prvek), `last` (poslední prvek), `before` (předchozí prvek), `after` (následující prvek), `isFirst`, `isLast`, `replaceElement`, `insertFirst`, `insertLast`, `insertBefore`, `insertAfter`, `remove`. Implementace obousměrným spojovým seznamem.

- **Prioritní fronta**
  Použití v systémech hromadné obsluhy s vlivem priority, přidělování CPU procesům. Metody `enqueue`, `dequeue`, `front`, `size`, `isEmpty`. Implementace nejlépe haldou, lze ale i polem nebo spojovým seznamem.

3 Stromové struktury a jejich implementace

Strom je ADT uchovávající prvky v hierarchické struktuře předchůdce–následovník. Použití v reprezentaci znalostí a stavového prostoru, popis scény při zpracování obrazu, v databázových systémech, systémech souborů, pro rozhodovací stromy, komprese dat, raytracing.

Obecné stromy – uzly mohou mít libovolný počet následovníků $\Rightarrow$ implementace pomocí ukazatelů, ve vrcholech spojový seznam následovníků. N-ární stromy – uzel nejvýše n následovníků, implementace ukazateli, ve vrcholech pole následovníků. Binární stromy – implementace ukazateli a výčtem následovníků, případně polem.

Binární strom, kde nalevo od každého prvku jsou k prvky s nižším klíčem a napravo s vyšším (příp. obráceně), je binární vyhledávací strom. Kořenem obyčejného BVS je první vložený prvek, další prvky se vkládají nalevo nebo napravo od něj. Při postupném vkládání seřazené posloupnosti ale takový strom degraduje na lineární seznam, což zvyšuje složitost vyhledávání. Proto snaha o vyvážení stromu.

- **AVL** (Adelson-Velsky, Landis)
  BVS zachovávající pravidlo, že podstromy každého vrcholu se mohou výškou lišit nejvýše o 1. Vyvážení kontrolováno po každém vložení a výběru. Vrchol obsahuje navíc položku informující o rozdílu výšek obou jeho podstromů. Vložení stejné jako u BVS, po vložení přepočet rozdílů výšek a kontrola vyváženosti postupně od místa vložení k vrcholu. Při nevyváženosti v některém vrcholu rotace v tomto vrcholu – jednoduchá (nevyváženost způsobuje levá větev levého podstromu nebo pravá větev pravého) nebo dvojitá (nevyváženost způsobuje pravá větev levého podstromu nebo levá větev pravého). Odebrání stejné jako u BVS – nahrazení prvku symetrickým následovníkem, po odstranění prvku přepočty rozdílů výšek kontrola vyváženosti směrem ke kořeni (od původní pozice symetrického následovníka – prvky prohozeny a zde odstraňován).

- **Red-Black**
  Další modifikace BVS. Vrcholy obsahují navíc položku s informací o barvě (červená nebo černá). Pravidla: kořen je vždy černý, cesty od kořene do každého listu obsahují stejný počet černých vrcholů (*černá výška* stromu), červený vrchol má pouze černé následovníky. Vkládaný vrchol je na začátku vždy červený, postupně od jeho pozice ke kořeni kontrola pravidel a případná náprava přebarvením vrcholů nebo rotací. Rotace jsou stejné jako u AVL stromu. Odebrání – nahrazení symetrickým následovníkem, ten ale obarven barvou odebíraného vrcholu, poté od místa odebrání kontrola pravidel směrem ke kořeni.

- **B-strom**
  B-strom řádu m má v každém vrcholu nejvýše m následovníků. Pravidla: klíčů ve vrcholu o 1 méně než následovníků, všechny listy stejně hluboko, všechny vrcholy nejvýše m následovníků, všechny vrcholy kromě

kořene alespoň $[\frac{m}{2}]$ následovníků, kořen buď list nebo alespoň 2 následovníky. Ve vrcholech *klíče seřazené*, vrcholy řazeny jako u BVS (menší klíče vlevo, větší vpravo). Při vkládání se přidávají klíče do listů, pokud je v listu překročen maximální počet (přetečení stránky), je list rozdělen uprostřed a prostřední klíč vložen do rodiče (zde opět kontrola překročení počtu). Odebereme-li klíč z listu a ten má potom příliš málo klíčů (podtečení stránky), může si vypůjčit od sousedů (klíč z rodiče do vrcholu, klíč ze souseda do rodiče), pokud sousedi mají minimální počet, dojde ke sloučení vrcholu s jedním sousedem a klíčem z rodiče. Při odebírání z vnitřních vrcholů je nutné místo vždy zaplnit – klíč od potomka. Nemají-li potomci dost klíčů, sloučí se dva potomci do jednoho. Implementace ukazateli s následovníky seznamem nebo polem, případně pomocí dvourozměrných polí (jedno pro klíče, jedno pro indexy následovníků, index kořenu).

4 Skip-list – použití a implementace

Spojová datová struktura vycházející ze seznamu, kde jsou některé vrcholy spojené nejen s bezprostředně následujícími, ale i vzdálenějšími. Ideálně i ukazatelů každých 2^{i+1} prvků (např. 1. prvek 4 ukazatele na 2., 3. a 5., 2. prvek pouze na 3., 3. prvek na 4. a 5., 4. prvek na 5. a 5. opět 4 ukazatele atd.), pak má vyhledávání složitost O($log_2 n$), ovšem je nutné při vkládání a výběru restrukturalizovat konec seznamu. Proto prvky vkládány s náhodným počtem ukazatelů podle uvedeného pravděpodobnostního rozložení. Je dobrou a snadno implementovatelnou náhradou za vyhledávací stromy, operace vložení a zejména výběru jsou mnohdy rychlejší než u stromů.

Obvykle se implementuje s hlavičkou, která má maximální počet ukazatelů (maximální úroveň). Maximální úroveň je definována jako $log_2 n$, kde n je předpokládaný maximální počet prvků. Hlavička prázdného skip-listu má všechny ukazatele `null`.

Při hledání se procházejí pouze prvky v nejvyšší úrovni, dokud není příští klíč větší. Pak se sestoupí o úroveň níže a pokračuje se stejným způsobem (zpřesňování). Pokud nebyl nalezen v úrovni 1, seznam prvek neobsahuje. Před vložením prvku je vyhledávacím algoritmem nalezeno místo vložení. Pokud je to nutné, je zvýšena hodnota maximální úrovně a vypočítána náhodně úroveň prvku (počet ukazatelů). Prvek je potom zařazen na své místo. Výběr se provede pouhým vyjmutím prvku ze seznamu a případně snížením maximální úrovně.

5 Tabulky s rozptýlenými položkami, vyhledávání v tabulkách

Rozsah klíče mnohonásobně větší než rozsah tabulky ($N >> p$). Určení pozice v tabulce výpočtem z hondoty klíče – *rozptylovací (hash) funkce* ($A_k = h(k)$). Někdy pro různá k stejné A_k = *synonymické položky*, potřeba najít způsob jejich uložení. Hash funkce musí: být definovaná pro všechny klíče, být vyčíslitelná v přijatelném čase, vytvářet minimum kolizí a rozdělovat klíče do rozsahu tabulky rovnoměrně. Způsoby realizace hash funkce – A_k je: částí k, částí výsledku operace nad k, zbytkem po dělení rozsahem p, zbytkem po dělení prvočíslem nejblíže menším p, váhovým součtem částí k.

- **otevřené rozptýlení**
  Při kolizi je vypočten jiný index v tabulce. Výpočet nového indexu buď pevně definován (např. přičtení konstanty k hashi a nové hashování), nebo index zvolen jinak (např. další volný nebo kolidující prvky zařazeny až po skončení vkládání) a uložen do kolizní položky = *vnitřní řetězení*.

- **uzavřené rozptýlení**
  Pro kolizní položky vytvořena další tabulka nebo vyhrazena část stávající. Druhý prvek se stejným indexem uložen do ní a do původního prvku uložen odkaz na něj = *vnější řetězení*.

- **rozptýlené indexy**
  V každé položce tabulky spojový seznam a do něj ukládány všechny prvky s daným indexem. Prvky lze zařazovat na konec, případně řadit podle klíčů nebo jiných kritérií. Oblíbená metoda, paměťově efektivní.

Vyhledávání v přímo adresovaných tabulkách procházením prvek po prvku, pokud jsou klíče seřazeny, lze použít binární nebo Fibbonacciho vyhledávání. V rozptylových tabulkách vygenerován hash z hledaného klíče a kontrolován prvek na daném indexu. Buď se tam nachází hledaný prvek, nebo je třeba následovat odkaz na zřetězenou položku. V případě použití spojových seznamů k ukládání synonym se prochází spojový seznam prvek po prvku. Při hledání podle sekundárního klíče je možné tabulku překopírovat do jiné, kde bude klíčem právě tento atribut, a zde vyhledávat. Vyhledávání podle více klíčů lze realizovat vícenásobným přístupem (vytvořením vyhledávacího stromu).

6 Algoritmy zpracování textů – operace s řetězci, porovnání se vzorem

Hledáme vzorový řetězec P (délky m) uvnitř textu T (délky n). Využití v textových editorech, fulltextovém vyhledávání, webové vyhledávače, analýza obrazů, strukturní rozpoznávání.

- **Hrubá síla** (brute force)
  Pro každou pozici v textu zjišťujeme, zda na ní nezačíná hledaný vzor. Složitost v nejhorším případě $O(mn)$, v běžném textu $O(m+n)$. Je rychlý pro velké abecedy (málo slov podobných vzoru), pro malé neefektivní.

- **Boyer-Moore** Prohledává se od posledního znaku P. Pokud se znak na příslušné pozici v T vyskytuje i v P před aktuální pozicí, vzor se posune doprava, aby shodné znaky byly zarovnány. Vyskytuje-li se znak v P pouze za aktuální pozicí, posune se vzor o 1 znak doprava. Když se ve vzoru znak vůbec nevyskytuje, je vzor posunut tak, aby jeho první znak byl zarovnán se znakem za aktuální pozicí v textu (tedy o m znaků doprava). Po každém posunu se pozice porovnávání přesouvá opět na konec P a odpovídající znak T. Je potřeba předzpracovat abecedu a každému znaku přiřadit buď index posledního výskytu v P nebo -1, nevyskytuje-li se v P, abychom věděli, o kolik se musí posouvat – funkce `Last`. Složitost v nejhorším případě $O(mn+A)$ (A je velikost abecedy), je vhodný pro velké abecedy a je pro ně rychlejší než brute force.

- **KMP** (Knuth-Morris-Pratt)
  Prochází text i vzor zleva doprava a snaží se posouvat vzor co nejefektivněji při neshodě. Při neshodě na j-tém znaku P změní j na délku nejdelšího prefixu $P[0..j-1]$, který je zároveň suffixem $P[1..j-1]$ – chybová funkce `F`. Postup jako brute force, při neshodě je nová pozice v P zjištěna voláním `F` s aktuální pozicí. Optimální složitost $O(m+n)$.

- **Rabin-Karp**
  Vypočten kontrolní součet vzorového řetězce (číslo) a všech podřetězců délky m v textu. Porovnávány kontrolní součty, při shodě brute force. Při optimálním nastavení lze dosáhnout složitosti $O(m+n)$.

7 Komprese dat, rozdělení kompresních metod, princip kompresních metod

Cílem komprese je snížit objem dat při zachování dostatku informací pro obnovení do přijatelné podoby. Kvalitu určuje poměr mezi zkomprimovanou a původní velikostí, dále rychlost komprese a symetrie algoritmu (tj. zda dekomprese má stejnou složitost jako komprese). Dělení na ztrátovou a bezztrátovou. Metody: *jednoduché* (odstranění opakujících se posloupností, např. RLE), *statistické* (četnost výskytu znaků v souboru, Huffmanovo nebo aritmetické kódování), *slovníkové* (vytvoření slovníku všech posloupností, LZW), *transformační* (ortogonální nebo jiné – FFT, Laplace – transformace, ztrátové, např. JPEG, waveletová, fraktálová komprese).

- **LZW** (Lempel-Ziv-Welch)
  Vyhledává posloupnosti znaků a ukládá je pod jednoznakové kódy. Kódované znaky musí mít délku větší, než znaky původní (např. 12b pro 8b). Během zpracovávání souboru se vytváří slovník, jehož prvních 2^k položek (k je počet bitů původních znaků) jsou jednotlivé původní znaky a za nimi jsou položky představující nalezené posloupnosti.

- **Huffmanovo kódování**
  Zjišťuje četnosti znaků v souboru a podle nich staví binární strom. Vždy je vložen nejčetnější znak do potomka vrcholu a druhý potomek je podstrom vytvořený ze zbytku znaků. Podle pravidla, že přechod do levého potomka přidává ke kódu nulu a do pravého jedničku (nebo obráceně) jsou pak kódovány jednotlivé znaky (prefixový kód). Strom je nutné uložit k datům, aby je bylo možné dekomprimovat.

- **Aritmetické kódování**
  Podle četností v souboru přiřazeny znakům části intervalu. Začíná se intervalem $\{0,1\}$, vybere se část intervalu podle prvního znaku, z té se opět vybere část podle druhého znaku atd. až se dojde na konec souboru. Získáme interval, ze kterého vybereme číslo a tím reprezentujeme celý soubor. Při dekompresi zjišťujeme, který znak patří k části intervalu, v níž číslo leží – získáme první znak a pokračujeme zvoleným intervalem pro další znak.

- **JPEG komprese** (Joint Photographic Experts Group)
  Ztrátová komprese obrazu. Obraz je nejprve převeden do barevného prostoru *YCbCr* (jasová a dvě barevné složky), může ale zůstat i v prostoru sRGB. Následně je možné převzorkovat barevné složky do nižšího rozlišení (oko rozeznává více detailů v jasové oblasti než v barevné) a to buď na polovinu v obou rozměrech (4:2:0), na polovinu ve vodorovném (4:2:2) nebo nechat v původním rozlišení (4:4:4). Takto upravený obraz je rozdělen na čtverce *8x8 bodů*, které jsou kódovány samostatně, nekompletní čtverce na koncích obrazu jsou

doplněny pomocnou výplní. Kódování probíhá zvlášť pro každý kanál. Obsah čtverce je vyjádřen ve frekvenční oblasti pomocí *diskrétní kosinové transformace* (DCT) a ponechány pouze koeficienty pro 8 nejnižších frekvencí ve vodorovném a 8 ve svislém směru. Těchto 64 koeficientů je ukládáno do matice, přičemž je každý nejprve vydělen konstantou z odpovídajícího místa *kvantizační matice* a zaokrouhlen na celé číslo. Tím se zejména koeficienty pro vyšší frekvence zredukují na nulu a zde dochází k největším ztrátám obrazových dat. Následně jsou čísla z matice čtením diagonálně tam a zpátky (cik cak) uložena do posloupnosti (linearizace), která je RLE kódováním zkomprimována = *entropní kódování* a uložena do souboru. Kvantizační matice určuje míru ztráty informací a míru výsledné velikosti souboru. Dekódování probíhá opačně - dekomprese RLE, naplnění do matice, vynásobení kvantizačními koeficienty, inverzní DCT, převod do RGB. Podobný princip využívá *waveletová* komprese (JPEG2000), místo DCT je ale použita lineární (waveletová) transformace a dosahuje lepších kompresních poměrů.

- **Fraktálová komprese**
  Podobně jako u JPEG jde o ztrátovou kompresi obrazu. Využívá *afinních transformací* (posunu, otočení, zrcadlení, změny velikosti) obrazu. Obraz je rozdělen na matici bloků (*range bloky*) a pro ty jsou pak zpětně v obraze hledány dvojnásobně velké bloky, které se jim podobají (*domain bloky*), a jsou zmenšovány na velikost range bloků. Následně je pro každý blok prohledán seznam jeho doménových bloků a vybrán ten, který je po transformaci nejvíce shodný s range blokem. Uloží se domain blok a použitá transformace a lze pokračovat dalším range blokem. Vyhledávání domain bloků v obraze lze provádět několika způsoby (hrubou silou, v omezené oblasti, spirálovitě od range bloku, kategorizací). Fraktálové algoritmy se využívají s dobrými výsledky také při zvětšování rozlišení obrazů.