16 Datové typy, seznam, zásobník, fronta, operace s nimi, jejich složitost. Vyhledávací a rozhodovací stromy (binární, AVL, B) jejich specifika a využití, efektivita operací, volba rozptylovací funkce pro specifické typy dat. (A4B36ALG)

16.1 Datové typy

Datový typ definuje v programování druh nebo význam hodnot, kterých smí nabývat proměnná (nebo konstanta). Datový typ je určen oborem hodnot a zároveň výpočetními operacemi, které lze s hodnotami tohoto typu provádět. Datový typ nemůže být určen pouze oborem hodnot, protože existují i datové typy, lišící se pouze v operacích, které je s nimi možné provádět.

Následující výpis je pro jazyk Java.

16.1.1 Primitivní datové typy

typ	popis	velikost
byte	celé číslo	8 bitů
short	celé číslo	16 bitů
int	celé číslo	32 bitů
long	celé číslo	64 bitů
float	reálné číslo	32 bitů
double	reálné číslo	64 bitů
char	znak UNICODE	16 bitů
boolean	logická hodnota	1 bit

16.1.2 Referenční datové typy

- Objekty
- Pole

Hodnota referenční proměnné je odkaz (reference) do paměti na místo, kde je objekt (nebo pole) uložen. V jiných programovacích jazycích se používají pro tento účel pointery (ukazatelů do paměti), v Javě se přímo s pamětí nepracuje, místo pointerů se používají referenční proměnné.

Typ `null` je prázdná hodnota pro referenční typy a znamená, že chybí odkaz na objekt resp. pole.

Jelikož referenční proměnná obsahuje pouze odkaz na objekt, nikoli objekt samotný, přiřazením její hodnoty jiné proměnné se přiřadí opět pouze reference na původní objekt, nikoli jeho samotná data tak, jak to bylo u primitivních datových typů.

16.2 Seznam

V Javě je seznam implementován třídou ArrayList. Ukládá veškeré prvky do běžného pole, které má neměnnou délku. V okamžiku, kdy překročíme délku tohoto pole, tak ArrayList vytvoří nové pole dvojnásobné délky, překopíruje do něj veškeré prvky a původní pole zahodí.

V případě, že je značná část pole nevyužívaná, tak ArrayList vytvoří nové menší pole, do nejž opět veškerá překopíruje a původní pole zahodí (čímž nám ušetří paměť).

Operace

- add(i, e) - vloží prvek `e` do seznamu na index `i` (náročnost O(n))
- remove(i | e) - odstraní prvek `e` nebo to, co je na indexu `i` (náročnost O(n))
- get(i) - vrátí prvek na indexu `i` (náročnost O(1))

16.3 Zásobník

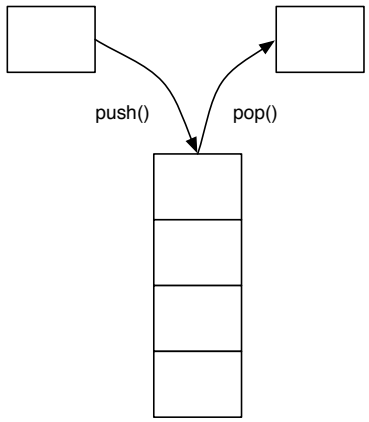

Figure 16.1: Zásobník

Zásobník (stack) je jednou ze základních datových struktur, která se využívá především pro dočasné ukládání dat v průběhu výpočtu. Zásobník data ukládá způsobem, kterému se říká **LIFO** - last in, first out - čili poslední vložený prvek jde na výstup jako první, předposlední jako druhý a tak dále.

Operace

- push - vloží prvek na vrch zásobníku (náročnost O(1))
- pop - odstraní vrchol zásobníku (náročnost O(1))
- top - dotaz na vrchol zásobníku (náročnost O(1))
- isEmpty - dotaz na prázdnost zásobníku

16.4 Fronta

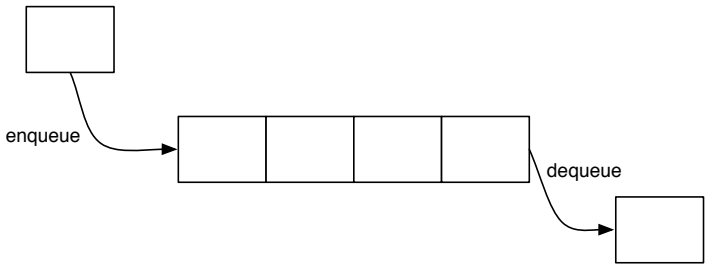

Figure 16.2: Fronta

Fronta (queue) slouží k ukládání a výběru dat takovým způsobem, aby prvek, který byl uložen jako první, byl také jako první vybrán. Tomuto principu se říká FIFO - first in, fist out.

Operace

- addLast (enqueue) – vloží prvek do fronty (náročnost O(1))
- deleteFirst (poll, dequeue) – získá a odstraní první prvek (hlavu) fronty (náročnost O(1))
- getFirst (peek) – získá první prvek fronty (náročnost O(1))
- isEmpty - dotaz na prázdnost fronty

16.5 Stromy

Jedná se o hierarchickou strukturu, kde každý otec má 0 až mnoho dětí a každé dítě právě jednoho otce takovým způsobem, že v této struktuře nejsou cykly. Uzel, který je

praotcem všech ostatních uzlů nazveme kořenem (z pohledu teorie grafů tím vytvoříme orientovaný strom). Uzel, který nemá žádné potomky nazýváme listem.

Být stromem je rekurzívní vlastnost - každý podstrom stromu S je také stromem.

16.5.1 Binární strom

Binární strom je orientovaný graf s jedním vrcholem (kořenem), z něhož existuje cesta do všech vrcholů grafu. Každý vrchol binárního stromu může mít maximálně dva orientované syny a s výjimkou kořene právě jednoho předka. Kořen předka nemá.

V praktickém programování je obvykle binární strom reprezentován dvěma způsoby:

1. pomocí dynamické struktury, kde jsou hrany reprezentovány ukazateli

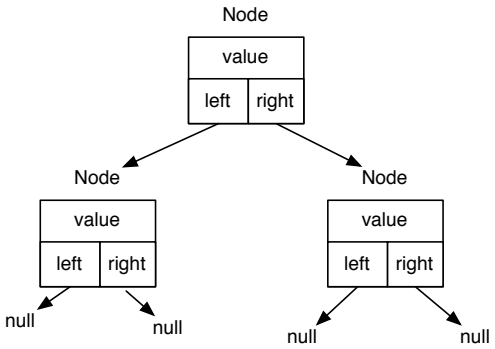

Figure 16.3: Strom pomocí dynamické struktury

2. pomocí pole, kde prvek s indexem i má následníky s indexem 2i+1 a 2i+2 (za předpokladu, že pole je indexováno od 0). Takto je například reprezentovaná halda v algoritmu heapsort (otázka 15).

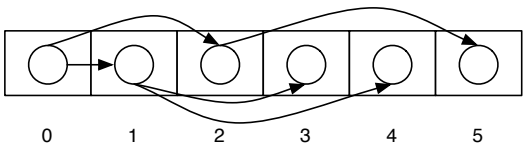

Figure 16.4: Strom pomocí pole

Za zmínku stojí speciální typ binárního stromu - **Vyvážený binární** strom, jehož hloubka listů se od sebe liší maximálně o jedna.

16.5.1.1 Binární vyhledávací strom

Grafické zpracovaní binárního vyhledávacího stromu z předmětu A4B36ALG spolecna/15/bvs.pdf

Binární vyhledávací strom (BST - z angl. Binary Search Tree) je datová struktura založená na binárním stromu, v němž jsou jednotlivé prvky (uzly) uspořádány tak, aby v tomto stromu bylo možné rychle vyhledávat danou hodnotu. To zajišťují tyto vlastnosti:

- Jedná se o binární strom, každý uzel tedy má nanejvýš dva syny — levého a pravého.
- Každému uzlu je přiřazen určitý klíč. Podle hodnot těchto klíčů jsou uzly uspořádány.
- Levý podstrom uzlu obsahuje pouze klíče menší než je klíč tohoto uzlu.
- Pravý podstrom uzlu obsahuje pouze klíče větší než je klíč tohoto uzlu.

Vyhledávání

Vyhledání konkrétní hodnoty v binárním vyhledávacím stromu typicky probíhá rekurzivně. Začíná v kořeni. V každém kroku porovná hledanou hodnotu s klíčem zkoumaného uzlu. Pokud jsou si rovny, hodnota byla nalezena. Je-li hledaná hodnota menší, pokračuje hledání v levém podstromu. Je-li větší, bude hledání pokračovat v pravém podstromu. Díky uspořádání stromu je cesta k hledané hodnotě jednoznačně určena.

Přídání uzlu

Vložení nového uzlu začíná hledáním jeho pozice ve stromu – postupuje se stejně jako při vyhledávání, jako hledaná hodnota se použije klíč vkládaného uzlu. Tato fáze může vést ke dvěma různým výsledkům:

- klíč byl nalezen, strom tedy dotyčnou hodnotu již obsahuje a není třeba ji vkládat (komplikovanější varianty připouštějící vícenásobný výskyt stejného klíče by pokračovaly dál do podstromu připouštějícího rovnost)
- algoritmus narazil na neexistující uzel, nový uzel bude vložen na toto místo, protože sem podle hodnoty svého klíče patří

Odstranění uzlu

- **Odstranění listu:** Odstranění uzlu bez potomků se jednoduše provede odstraněním uzlu ze stromu.
- **Odstranění uzlu s jedním potomkem:** Provede se nahrazením uzlu uzlem potomka.
- **Odstranění uzlu se dvěma potomky:** Nechť se odstraněný uzel nazývá N. Pak je hodnota uzlu N nahrazena nejbližší vyšší (uzel pravého podstromu, který je nejvíc vlevo (nejlevější ☺)), nebo nižší hodnotou (uzel levého podstromu, který je nevíc vpravo (nejpravější ☺)). Takový uzel má nanejvýš jednoho potomka, lze jej tedy ze stromu vyjmout podle jednoho z předchozích pravidel. Obě možnosti ilustruje následující obrázek, kdy je ze stromu odstraněn uzel s klíčem 7. V dobré implementaci je doporučeno obě varianty střídat, jinak dochází k narušení rovnováhy.

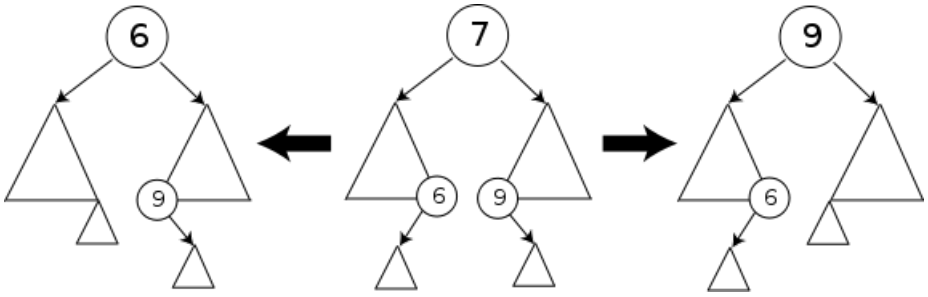

Figure 16.5: Odstranění uzlu se dvěma potomky

16.5.1.2 AVL strom

Grafické zpracovaní AVL stromu z předmětu A4B36ALG spolecna/15/avl.pdf

AVL strom je datová struktura pro uchovávání údajů a jejich vyhledávání. Pracuje v logaritmicky omezeném čase. Jedná se o **samovyvažující se binární vyhledávací strom**.

- Vrchol má maximálně dva následníky (plyne z vlastností binárního stromu, kterým AVL strom je).
- V levém podstromu vrcholu jsou pouze vrcholy s menší hodnotou klíče
- V pravém podstromu vrcholu jsou pouze vrcholy s větší hodnotou klíče
- **Délka nejdelší větve levého a pravého podstromu se liší nejvýše o 1 (vyvážení AVL stromu).**

Poslední vlastnost je demonstrována na následující dvojici stromů.

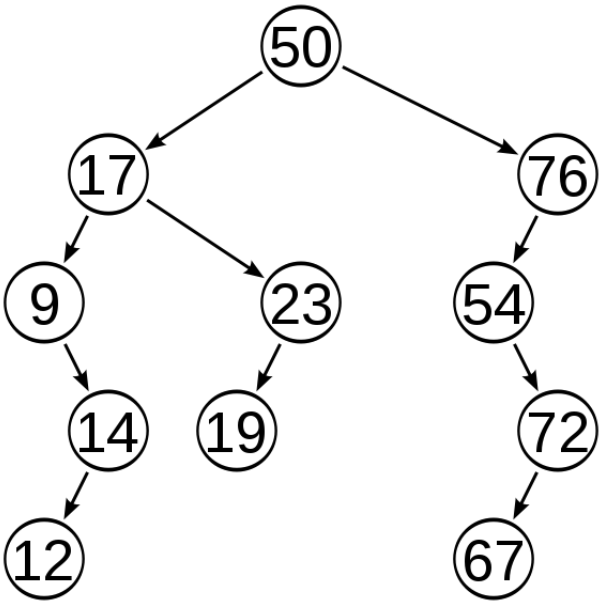

Figure 16.6: Binární vyhledávací strom

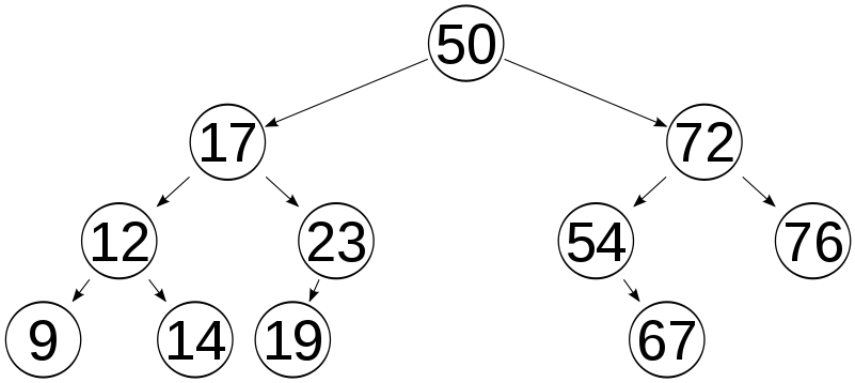

Figure 16.7: AVL strom

První strom nesplňuje vlastnosti AVL stromu v uzlech 9, 76 a 54, kde je rozdíl délky nejdelší větve levého a pravého podstromu 2, 3 a 2, tedy více než 1.

Vyhledávání

Totožné jako u vyhledávání v BVS 16.5.1.1.

Přidání uzlu

Totožné jako u přidávání v BVS 16.5.1.1. Strom se ale po přidání může stát nevyváženým, proto je potřeba provést vyvažovací rotace.

Rotace

Podrobně a obrázky popsáno v spolecna/15/avl.pdf. Postup je následující.

1. Ve směru rotace nahradíte špatně vyvážený uzel potomkem.
2. Pokud by se stalo, že by nově dosazený vrchol měl 3 potomky, tak jeho původního potomka umístíte pod právě sesazený vrchol.

Možná to bude lépe vidět z následujícího obrázku, kam jsme přidali vrchol 93. Uzel, který nesplňuje požadavky AVL je 51 (pravá větev délky 3, levá 1 - rozdíl 2). Ten v kroku `a` zaměníme za 70, tím ale dostáváme pod 70 tři potomky. Jedná se o levou rotaci (typy rotací popsány níž), proto potomka 53 v kroku `b` hodíme nalevo. Hehe, ja vím, ta vedlejší věta důsledková teď nedává moc smysl, tak čistě úvahou - potřebujeme někam udat toho prostředního potomka 53, protože každý vrchol může mít maximálně dva potomky. Doprava pod 84 ho dát nemůžeme - je totiž menší než 70, a tak nemá v pravém stromu uzlu 70 co dělat. Proto ho dáme nalevo, pod 51.

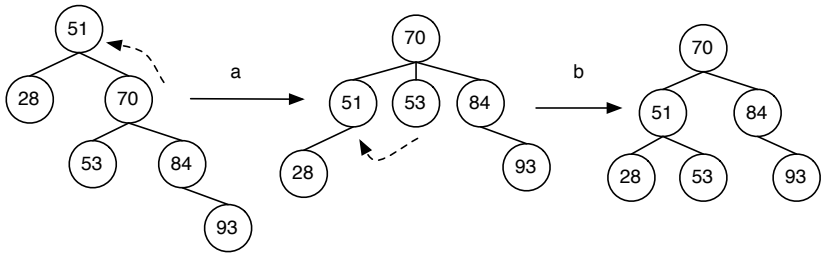

Figure 16.8: Ukázka L rotace

Na předchozím příkladu jsme ilustrovali levou rotaci. Kromě ní existuje i pravá rotace a kombinace obou. Kdy jakou použít? Od přidaného (nebo smazaného, viz dále) uzlu postupujeme směrem ke kořenu a aktualizujeme hloubky podstromů v každém navštíveném uzlu.

- Když narazíme na rozvážený uzel, do kterého jsme bezprostředně došli dvěma hranami doleva nahoru, provedeme v tomto uzlu L rotaci.

  To byl náš příklad, od přidané 93 jsme postupovali nahoru, kde jsme narazili na rozvážený uzel 51. Doskákali jsme k němu dvakrát doleva nahoru.

- Když narazíme na rozvážený uzel, do kterého jsme bezprostředně došli dvěma hranami doprava nahoru, provedeme v tomto uzlu R rotaci.

- Když narazíme na rozvážený uzel, do kterého jsme bezprostředně došli hranami doleva a pak doprava nahoru, provedeme v tomto uzlu LR rotaci (tedy nejprve levou a pak pravou).

- Když narazíme na rozvážený uzel, do kterého jsme bezprostředně došli hranami doprava a pak doleva nahoru, provedeme v tomto uzlu RL rotaci.

Po provedení jedné rotace je AVL strom opět vyvážen.

Odstranění uzlu

Opět totožné jako u odstranění v BVS 16.5.1.1. Poté postupujeme od místa smazání nahoru ke kořeni a aktualizujeme výšky podstromů v každém uzlu.

16.5.2 B-strom

Grafické zpracovaní B-stromu z předmětu A4B36ALG spolecna/15/b-tree.pdf

B-strom je druh stromu, který ale **nesouvisí** s binárními stromy. Jeho definice

- Kořen má nejméně dva potomky, pokud není listem
- Každý uzel kromě kořene a listu má nejméně $\left[\frac{n}{2}\right]$ a nejvýše n potomků.
- Každý uzel kromě kořene má nejméně $\left[\frac{n}{2}\right] - 1$ a nejvýše $n - 1$ položek.
- Všechny cesty od kořene k listům jsou stejně dlouhé

B-strom je díky těmto vlastnostem vyvážený, operace přidání, vyjmutí i vyhledávání tedy probíhají v logaritmickém čase.

Na následujícím diagramu je ukázka b-stromu řádu 5. Řád určuje, kolik může mít vrchol maximálně potomků, v tomto případě 5. Položek může být maximálně (5-1), tedy 4.

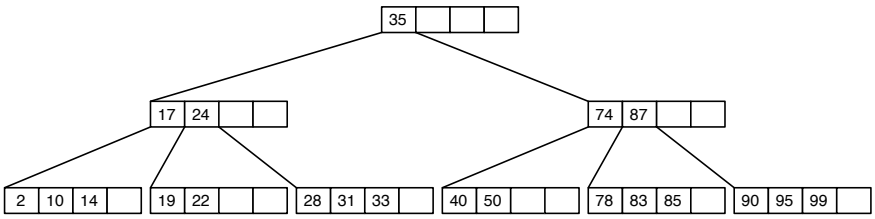

Figure 16.9: Ukázka b-stromu

Operace vyhledávání, přidávání a odstraňování uzlu b-stromu jsou graficky popsány v souboru `spolecna/15/b-tree.pdf`. Jde o to po každé operaci udržovat vlastnosti b-stromu buďto „sléváním" nebo „rozdělováním" vrcholů.

16.5.3 Srovnání stromů

	BVS s n uzly		AVL s n uzly	B-strom s n uzly
Operace	Vyvážený	Možná nevyvážený	Vyvážený	Vyvážený
Find	$O(\log(n))$	$O(n)$	$O(\log(n))$	$O(\log(n))$
Insert	$\Theta(\log(n))$	$O(n)$	$\Theta(\log(n))$	$\Theta(\log(n))$
Delete	$\Theta(\log(n))$	$O(n)$	$\Theta(\log(n))$	$\Theta(\log(n))$

Pár poznámek na závěr

- označení binární strom je nadmnožina pro binární vyhledávací strom a AVL strom

Figure 16.10: Vlastnosti stromů

Vyváženost zajišťuje AVL stromu lepší asymptotickou složitost. V tabulce srovnávající jednotlivé stromy je uveden sloupec „vyvážený binární vyhledávací strom", nicméně to, že náš BVS na vstupu bude skutečně vyvážený, nemáme zaručeno. U AVL stromu to zaručeno je.

- B-strom a AVL strom mají stejnou složitost, každý se ale hodí v jiné situaci. Využití b-stromu může být v aplikacích, kdy není celá struktura uložena v paměti RAM, ale v nějaké sekundární paměti, jako je pevný disk (například databáze). Protože přístup do tohoto typu paměti je náročný na čas (hlavně vyhledání náhodné položky), snažíme se minimalizovat počet přístupů do této paměti.

Příklad

Máme-li B-strom hloubky 2 a počet potomků každého uzlu je 1001, můžeme do něj uložit milion klíčů a ke každé položce se dostaneme maximálně po dvou diskových operacích.