Obsah

Úvod

První kapitola práce má název Úvod. Slouží k zasazení řešené problematiky do širšího kontextu a v podobě stručného obsahu jednotlivých kapitol definuje strukturu písemné práce. Šablona obsahuje formátování, podle Závazných pokynů na stránkách FEI http://www.fei.vsb.cz/cs/okruhy/studium-a-vyuka/informace-pokyny/pokyny-zpracovani-bp-dp. Rozsah úvodu BP je doporučena cca půl strany A4, u DP cca 1 A4.

1 Datové struktury

Datová struktura je množina k dat sloužící k jejich uchovávání a uspořádání. Velikost datové struktury se může měnit. Říkáme tedy, že má dynamický charakter. Datové struktury mohou být lineární (pole, zásobník, fronta, seznam) či nelineární (stromy, grafy). Výběr datových struktur je pro vývoj programů často stěžejní. Výběr struktury ovlivňuje množství použité paměti, čas potřebný k operacím prováděným s daty jakými jsou vkládání do datové struktury, jejich vyhledávání, odstranění. Různé datové struktury mají své specifické operace pro práci s daty. [1][3]

1.1 Abstraktní datové typy

Datové struktury, kterými se budeme zabývat, jsou abstraktní datové typy. Umožňují totiž sestavení programů s velkou mírou abstrakce. Jedná se o datové typy, ke kterým přistupujeme skrze jejich rozhraní. [3]

1.2 Prvky datových struktur

Jednotlivé prvky datových struktur mohou být různé. Může se jednat o jednoduché typy (primitivní datové typy) či celé třídy s komplikovanou vnitřní strukturou.

Jednotlivé prvky lze od sebe na základě nějaké jejich vlastnosti rozlišovat porovnávat a tím pádem také uspořádat. [1]

1.3 Operace nad prvky datových struktur

Operace mohou být pro různé datově struktury různé. Obecně však lze operace rozdělit na 2 typy: dotazy a modifikující operace.

Dotazy vrací nějakou informaci o datové struktuře. Nejčastější dotazovací operace jsou:

- DS.Search(k): vyhledávání prvku k v datové struktuře DS,
- DS.Min (): nalezení minimálního prvku v uspořádané datové struktuře DS,
- DS.Max (): nalezení maximálního prvku v uspořádané datové struktuře DS,

Modifikující operace mění datovou strukturu. Nejčastější modifikující operace jsou:

- DS.Insert(x): vložení prvku x do datové struktury DS,
- DS.Delete(x): odstranění prvku x z datové struktury DS.

2 Nelineární datové struktury - stromy

Strom je souvislý, acyklický, neorientovaný graf. Vrcholy takového grafu nazýváme uzly.

Kořenový strom je takový strom, který má jeden odlišný uzel. Tento uzel nazýváme kořen. Máme-li cestu mezi kořenem a libovolným jiným uzlem x, pak říkáme, že x je následovník kořene. Všechny uzly na této cestě od kořene k uzlu x jsou předchůdci uzlu x. Existuje vždy právě jedna cesta z uzlu ke každému uzlu ve stromu.

Pokud mezi uzlem u a kořenem není žádný jiný uzel, pak tento uzel nazýváme potomkem kořene. Každý uzel s výjimkou kořene má právě jeden uzel, který uzlu předchází a ten nazýváme rodič.

Uzel, který nemá žádné potomky, nazýváme list. Uzel s potomky je vnitřní uzel. Pro další uzly ve stromu používáme obdobné názvy jako v rodokmenu. Nazýváme tedy rodiče, který je rodičem uzlu x prarodičem uzlu x, ve stromu můžeme najít třeba i sourozence a strýce.

Pokud v každém uzlu musíme mít určitý počet potomků ve specifikovaném pořadí, pak tento strom nazýváme M-ární strom. Toto pravidlo však často neplatí pro kořen a vnější uzly - listy. [1][3]

2.1 Binární stromy

Binární strom je složen z uzlů majících dva potomky. Každý potomek s výjimkou uzlu binárního stromu má právě jednoho rodiče. Pro každého potomka platí pravidla binárního stromu. Každý potomek je tedy buď levým, nebo pravým podstromem binárního stromu. Krom potomků pak ještě každý uzel obsahuje data s výjimkou listů, které mohou být prázdné (pro dodržení podmínky, že každý uzel má právě 2 potomky).

Binární vyhledávací strom je takový binární strom, který má potomky setříděny podle nějakého klíče. Toto setřídění je pak stejné pro všechny uzly v daném binárním vyhledávacím stromu. Levý potomek tedy bude vždy dle tohoto porovnání menší než pravý.

Většinou nevíme dopředu nic o klíčích binárního vyhledávacího stromu, dle kterých pak budou ve stromu přibývat nové uzly a které tedy budou strom formovat. Může se tedy stát, pokud budou přibývat uzly s klíči, které jsou seřazeny vzestupně (nový uzel se vždy zařadí jako nejpravější uzel stromu) či sestupně (nový uzel se vždy zařadí jako nejlevější uzel stromu). Takový strom pak degraduje na lineární seznam a k nalezení prvku je potřeba průměrně zde $\frac{n}{2}$ porovnání. V nejhorším případě je potřeba až n porovnání.

Vyhledávání v Binárním vyhledávacím stromu probíhá zavoláním metody vyhledávání na kořen tohoto stromu. Pokud se klíč tohoto uzlu shoduje s hledaným, pak byl uzel úspěšně nalezen. Pokud je vyhledávaný klíč větší jak klíč v aktuálním uzlu, pak pokud neexistuje pravý potomek, tak skončím s tím, že uzel nebyl nalezen. Pokud pravý potomek existuje, tak zavolám rekurzivně metodu vyhledávání na pravého potomka. Analogicky pokud hledaný klíč je menší jak klíš v aktuálním uzlu a neexistuje levý potomek, pak se hledaný klíč ve stromu nenachází. Pokud levý potomek existuje, tak na něj rekurzivně zavolám metodu vyhledávání.

Vkládání do Binárního vyhledávacího stromu je obdobné jako vyhledávání. Vkládaný prvek se pokusíme vyhledat. Pokud hledání skončí úspěšně, tak se jedná o duplicitní klíč. Pokud by vyhledávání skončilo neúspěšně (tedy potomek, na kterého jsme chtěli zavolat metodu vyhledávání, neexistuje) tak právě zde vytvoříme nový uzel s klíčem, který chceme vložit a který jsme rovněž použili pro vyhledávání, které skončilo neúspěšně.

Rušení uzlu opět nejdříve zahrnuje jeho vyhledávání. Pokud prvek není nalezen, pak tato procedura končí. Pokud je prvek nalezen, pak záleží na počtu potomků další postup. Pokud rušený uzel nemá žádného potomka, pak jej lze odstranit bez jakékoli další akce. Pokud má uzel jednoho potomka, pak se rodič rušeného uzlu stane rodičem uzlu, který je potomkem rušeného uzlu. Potomek rušeného uzlu se tak stane potomkem uzlu, který je rodič rušeného uzlu. Po vytvoření této vazby je možné rušený uzel odstranit. Jestliže má rušený uzel 2 potomky, pak máme dvě možnosti. Buď nahradíme rušený uzel nejpravějším uzlem levého podstromu, nebo nejlevějším uzlem pravého podstromu. Nejpravější (respektive nejlevější) uzel nalezneme rekurzivně tak, že projdeme levý (respektive pravý) podstrom rušeného uzlu a pokud existuje pravý (levý) potomek, tak jej navštívíme a rekurzivně se opět snažíme dostat do jeho pravého (levého) potomka až do té doby, dokud takový potomek existuje. Poslední uzel - tedy ten, který již nemá pravého (levého) potomka je nejpravější (nejlevější) potomek.

2.2 Dokonale vyvážené stromy

Nejhorší případy např. pro sestavení binárního vyhledávacího stromu jsou takové, kdy vkládáme položky, které jsou již seřazené, mají velké množství duplicitních klíčů (pokud jsou ve stromu povoleny), jsou opačně seřazeny, nebo alternují klíče s velkými a malými hodnotami. V takových případech strom degraduje, stává se i pro relativně malý počet položek vysoký a vyhledávání se tak v něm stává velmi pomalé v porovnání se stromem, který obsahuje stejný počtem položek, ale má lepší strukturu - tedy není tak vysoký. [3]

Dokonale vyvážený strom je takový strom, který má počet uzlů v levém podstromu stejný jako v tom pravém, nebo se jejich počet liší maximálně o jeden. Toto pravidlo platí pro každý uzel takového stromu.

Dokonale vyvážené stromy jsou velmi výhodné pro vyhledávání, jelikož složitost vyhledávání v nejhorším případě se rovná délce nejdelší cesty ve stromu. Jelikož dokonale vyvážený strom má pro všechny listy cestu stejně dlouhou, lišící se maximálně o 1, pak je tato cesta nejkratší v porovnání s jinými stromy, jejichž uzly mají stupeň 2. Průměrný počet porovnání k nalezení uzlu v takovémto stromu je $\log n$.

Velkou nevýhodou dokonale vyvážených stromů je odebrání, nebo přidání nového uzlu. Tato akce je časově náročná, jelikož téměř vždy naruší dokonalou vyváženost stromu a vyžaduje tak jeho přestavění. Dokonale vyvážené stromy lze tedy použít v případě, že dopředu známe počet uzlu ve stromu a po jeho sestavení se pak uzel již nemění.

2.3 AVL stromy

Vyvážený strom je takový binární strom, u kterého se délka nejdelší cesty levého podstromu a délka nejdelší cesty pravého podstromu liší maximálně o 1. AVL stromy jsou vyvážené stromy a na rozdíl od dokonale vyvážených stromů není nutné při každém vkládání / rušení uzlu strom znovu konstruovat. Pokud však při vkládání strom přestane být vyvážený, pak je potřeba jej opět vyvážit.

Pro každý uzel si uchováváme informaci o jeho vyváženosti, a jestliže uzel není vyvážený, pak jej vyvážíme za pomocí rotace.

Rotace je operace, při níž dochází k výměně pozice rodiče a potomka takovým způsobem, aby byl strom opět vyvážený, a zároveň zachovává pravidlo binárního vyhledávacího stromu, že má rodič vlevo potomka s menším klíčem a vpravo potomka s klíčem větším než je klíč rodiče. Rotace rozlišujeme na jednoduché a dvojité. Jednoduchá rotace je buď pravá RR, nebo levá LL. Pravá rotace RR je operace, při níž se z rodiče stává levý potomek a současně z jeho původně pravého potomka se nově stává rodič. Levá rotace LL je pak operace, při níž se z levého potoka stává rodič a z rodiče pravý potomek. Dvojité rotace rozlišujeme dvě a to LR a RL. Při LR rotaci je nejdříve provedena levá rotace s tím, že levý potomek původního pravého potomka (nynějšího rodiče) se stane pravým potomkem původně rodiče (nynějšího levého potomka) a následuje pravá rotace rodiče nového rodiče. RL rotace je opět nejdříve pravá rotace s přesunem pravého potomka od původního potomka (nynějšího rodiče) k původnímu rodiči (nynějšímu pravému potomku) jako jeho levý potomek Následuje levá rotace rodiče od nového rodiče (původního levého potomka).

Při rušení uzlu, pokud je to potřeba a strom přestane být vyvážený, je opět potřeba strom vyvážit za pomoci zde popsaných rotací.

2.4 2-3-4 stromy

2-3-4 strom je takový strom, který obsahuje 3 typy uzlů. 2-uzel, 3-uzel a 4-uzel. Číslo uvedené u názvu jednotlivých uzlů říká, na kolik potomků daný uzel ukazuje. 2-uzel tedy ukazuje na 2 potomky, 3-uzel na 3 potomky a 4-uzel na 4. Každý z těchto uzlů má počet klíčů na kolik potomků ukazuje - 1, tedy 2-uzel je klasický uzel binárního stromu s 1 klíčem a dvěma potomky, kdežto 3-uzel obsahuje 2 klíče a 4-uzel 3.

Vkládání do takového stromu probíhá tak, že nalezneme pozici, kde by se měl uzel nacházet a pokud je zde 2-uzel, tak je do něj nový klíč přidán a stává se tak z něj 3-uzel. Obdobně je tomu u 3-uzlu, ze kterého se analogickým způsobem stane 4-uzel. V případě, že se na místě, kam chceme klíč vložit, nachází 4-uzel, pak je tento uzel rozdělen. Rozdělení uzlu je provedeno tak, že prostřední klíč je vložen do rodiče a z krajních klíčů se stávají potomci. Do jednoho z těchto potomků je pak nově vkládaný klíč vložen (v závislosti na porovnání s klíčem, který byl vložen do rodiče).

V případě, že by i rodič byl 4-uzlem, tak do něj opět nelze nový klíč vložit. Bylo by nutné provést štěpení - jak je popsáno v předchozím odstavci. Stejná situace by pak nastala i s jeho rodičem až, v nejhorším případě, s kořenem. Aby se tomuto zabránilo, tak vždy při vyhledávání místa pro vložení listu pokud se narazí na 4-uzel, tak je tento uzel rozdělen. Díky této operaci pak budou 4-uzly pouze v listech a máme jistotu, že při rozdělování 4-listu můžeme vložit prostřední hodnotu do rodiče, jelikož ten jistě 4-uzel není. Pokud by se kořen stal 4-uzlem, tak při jeho rozdělení se z prostředního klíče stane nový kořen, a ze 2 zbývajících pak jeho potomci.

Díky tomu, že strom "roste do výšky", tak se jeho výška zvětšuje pouze tehdy, když dělím kořen. 2-3-4 strom je tedy vždy dokonale vyvážený.

Vyhledávání, z důvodu vyskytujících se 3-uzlům a 4-uzlům, bude pomalejší než vyhledávání v binárním stromu.

2.5 Red-Black stromy

Algoritmus pro vkládání položek do 2-3-4 je snadný k pochopení, ale poněkud složitý na implementaci kvůli množství případů, které mohou nastat. Hlavní myšlenkou Red-Black stromů je tedy mít binární vyhledávací strom s výhodami 2-3-4 stromu.

Red-Black strom, někdy také červeno-černý strom, je částečně vyvážený binární strom s výškou $2\log(n+1)$, kde n je počet uzlů daného stromu. Částečně vyvážený je proto, že každá cesta z libovolného uzlu do listu obsahuje vždy stejný počet černých uzlů. Počet takovýchto černých uzlů nazýváme černou výškou. Nejdelší cesta je tak vždy nejvýše tak dlouhá, jak dvojnásobek nejkratší cesty ze stejného uzlu.

Každý uzel Red-Black stromu je buď černý, nebo červený, což je zaznamenáno pomocí příznaku uvnitř každého uzlu. Každý list je černý a neobsahuje žádnou hodnotu. Je reprezentován hodnotou NULL. Jestliže je některý uzel červený, pak jsou jeho potomci vždy černí. Uzel je vždy černý.

Strom se díky tomuto příznaku stává velmi podobným 2-3-4 stromu. 2 červené potomky s černým rodičem si lze představit jako 4-uzel a 3-uzel je právě jeden červený potomek (levý nebo pravý) a černý rodič. [3]

Operace vkládání a rušení uzlů mají složitost $O(\log \mathrm{n})$ a k vyváženosti jsou stejně jak u AVL stromů použity rotace. Počet těchto rotací je konstantní a tak se Red-Black stromy často využívají tam, kde se uzly rychle objevují a rychle mizí, nebo v aplikacích běžících v reálném čase.

V Red-Black stromech rozeznáváme 2 rotace - levou a pravou. Levou rotací rozumíme operaci, kdy se z rodiče stane potomek jeho původně pravého potomka rodič. Z levého potomka původního pravého potomka (nyní rodiče) se stane pravý potomek původního rodiče (nyní levého potomka). Pravou rotací pak rozumíme operaci inverzní k levé rotaci. Z rodiče se stane pravý potomek, z jeho levého potomka se stane nový rodič a z pravého potomka od tohoto původně levého potomka (nyní rodiče) se stane levý potomek původního rodiče (nynějšího pravého potomka).

Uzel, který vkládáme do tohoto stromu, je vždy červený. Vkládání probíhá na stejnou pozici jako bychom vkládali do binárního vyhledávacího stromu, tedy pokusíme se uzel vyhledat a v případě neúspěchu jej vložíme na místo, kde bychom jeho pozici předpokládali. Pokud je rodič černý uzel, pak jsou všechny podmínky pro Red-Black strom splněny. Pokud ne, pak může nastat několik dalších případů. Jestliže je rodič červený, tak dochází k porušení podmínky, že každý červený uzel musí mít 2 černé potomky. Důležitá je barva "strýce" vkládaného uzlu - tedy sourozence od červeného rodiče. Pokud je tento strýc červený, pak jen zaměníme barvu rodiče, strýce a prarodiče od vkládaného uzlu. Opět může být narušeno některé z pravidel Red-Black stromu, ale to o 2 úrovně výše, což je opět nutné opravit. Pokud však strýc není červený, ale černý, pak závisí, jestli vkládám nový prvek jako levého, nebo jako pravého potomka. Pokud je vkládán uzel jako levý potomek, pak je nutné provést pravou rotaci a obarvit původního rodiče a prarodiče (nynějšího rodiče a bratra) vkládaného uzlu. Jestliže však je strýc vkládaného uzlu černý a vkládá se nový uzel jako pravý potomek červeného uzlu, pak se provede levá rotace a původní rodič bude nyní jako nově vkládaný uzel. Tím pádem se nám problém mění na předchozí. Červený rodič, černý strýc a vlevo vložený nový prvek.

Pokud rušíme uzel, tak postupujeme nejdříve stejně jako u binárního vyhledávacího stromu. Pokud jsme smazali červený uzel, tak pravidla Red-Black stromu zůstala zachována. Pokud však rušíme uzel, který je černý a má černého potomka, tak jakmile potomek nahradí smazaného rodiče, tak jej označíme jako "dvojnásobně černý uzel". Snažíme se najít nejbližší červený uzel a dvojici červený - dvojnásobně černý uzel nahradit dvěma černými uzly. Máme 2 způsoby jak toho docílit: restrukturalizace a přebarvení. Restrukturalizace řeší problém lokálně, přebarvení šíří problém vzhůru. Pokud bratr dvojnásobně černého uzlu má červeného potomka, pak udělám restrukturalizaci. Pokud je tento potomek pravý, pak uděláme levou rotaci a přebarvíme dvojnásobně černý uzel na černý a červený uzel na černý. Pokud je onen potomek levý, tak provedeme pravou a levou rotaci a opět přebarvení oné dvojice uzlů. Pokud je bratr černý a má černého potomka, pak je třeba provést přebarvení bratra na červený uzel. Jestliže rodič byl červený, tak je odebrání uzlu hotovo. Pokud však byl otec černý, tak se z něj stává dvojnásobně černý uzel a je potřeba na něj aplikovat některé pravidlo pro opravení tohoto jevu. Pokud je pravý bratr červený, pak je třeba rodiče obarvit na červeno, červeného bratra na černo a provést restrukturalizaci pomocí levé rotace. Problém je tak oddálen o 1 krok dále od kořene.

2.6 Ternární stromy

?

2.7 B-stromy

Místo toho, abychom trvali na tom, že v každém uzlu musí být právě M klíčů, tak budeme trvat na tom, že každý uzel musí mít klíčů nejvýše M s M+1 odkazy na potomky a nejméně pak M/2 s M/2+1 odkazy. Výjimkou je pochopitelně kořen, který musí mýt alespoň jeden klíč a 2 odkazy na potomky, nejvýše pak také M klíčů s M-1 odkazy. Strom je buď prázdný, nebo obsahuje k-uzly. S k-1 klíči a k odkazy na potomky - stromy které reprezentují interval mezi klíči, mezi kterými je odkaz umístěn. Pokud se jedná o první odkaz, pak interval je menší než první klíč, pokud se jedná o nejpravější odkaz, pak odkazuje na strom, jehož interval je větší než nejpravější klíč aktuálního uzlu. Pokud je odkaz mezi dvěma klíči, pak je interval stromu, na nějž odkaz odkazuje, ohraničen právě těmito dvěma klíči. [3]

2.8 R-stromy

3 Ok

3.1 ok

	Nejhorší případ			Průměrný případ		
	Insert	Search	Select	Insert	Search hit	Search miss
Indexované pole	1	1	M	1	1	1
Setříděné pole	N	N	1	N/2	N/2	N/2
Setříděný spojový seznam	N	N	N	N/2	N/2	N/2
Nesetříděné pole	1	N	NlgN	1	N/2	N
Nesetříděný spojový seznam	1	N	NlgN	1	N/2	N
Binární vyhledávání	N	lgN	1	N/2	lgN	lgN
Binární vyhledávací strom	N	N	N	lgN	lgN	lgN
Red-black tree	lgN	lgN	lgN	lgN	lgN	lgN
Randomized tree	N*	N*	N*	lgN	lgN	lgN
Hashing	1	N*	NlgN	1	1	1

[3] str 494

Použitá literatura

[1] ŽALUD, Václav. *Moderní radioelektronika*. 1. vyd. Praha: BEN, 2000, 656 s. ISBN 80-86056-47-3.

[2] Výkony, limity ČTU a GL č. 12/R/2000. *KHnet.info* [online]. [cit. 2011-01-05]. Dostupné z: http://forum.khnet.info/viewtopic.php?f=10&t=964#p7299

[3] Sedgewick, Robert. Algorithms in C (Parts 1-4), 3. vyd. Addison-Wesley, 1998, 702 s. ISBN 0-201-31452-5.