Stromy

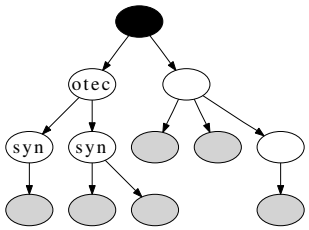
Dátová štruktúra strom je v podstate zovšeobecnením spájaného zoznamu. Rozdiel je v tom, že pri každom prvku môžeme mať ukazovatele na viacero ďalších prvkov.

Presnejšie to môžeme formulovať takto: Strom sa skladá z niekoľkých vrcholov. Jeden z týchto vrcholov voláme koreň. Podobne ako pri spájanom zozname, každý vrchol je záznam, ktorý obsahuje jeden prvok a prípadne niekoľko ukazovateľov na iné vrcholy. Pritom požadujeme, aby platilo:

- Na koreň neukazuje žiaden vrchol.
- Na každý vrchol okrem koreňa ukazuje práve jeden iný vrchol.
- Každý vrchol je dosiahnuteľný z koreňa po ukazovateľoch.

Vrcholy, na ktoré ukazuje vrchol v, voláme jeho *synmi*. A naopak, vrchol v voláme *otcom* týchto vrcholov. Vrcholy, ktoré neukazujú na žiaden iný vrchol, voláme listami.

Príklad grafického znázornenia stromu nájdete na obrázku 1.1.



Obr. 1.1: Strom. Čierny vrchol je koreň, sivé sú listy.

Hĺbka vrcholu je jeho vzdialenosť od koreňa. Hĺbka stromu je maximum z hĺbok jeho vrcholov. Strom z obrázku 1.1 má teda hĺbku 3.

Poznámka k implementácii: V niektorých situáciách môže byť vhodné pridať k nami popísanej implementácií stromu ešte pre každý prvok jeden ukazovateľ na jeho otca.

Cvičenie 1.0.1 *Akú najmenšiu a akú najväčšiu hĺbku môže mať N-vrcholový strom, v ktorom platí, že ak vrchol nie je list, tak má aspoň d synov?*

1.1 Binárne vyhľadávacie stromy

Jedným špeciálnym typom stromov sú *binárne stromy*. V binárnom strome platí, že každý vrchol má najviac dvoch synov. Navyše budeme rozlišovať medzi ľavým a pravým synom.

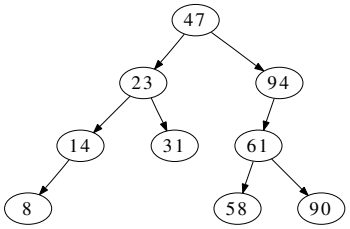
My budeme chcieť použiť takýto strom na ukladanie prvkov. Najjednoduchší spôsob je samozrejme do každého vrcholu umiestniť jeden prvok. Ak by sme to ale robili ako sa nám zachce, nič by sme nezískali oproti spájaným zoznamom – napríklad na to, aby sme zistili, či náš strom obsahuje daný prvok, by sme ho museli celý prejsť. Aby sme niečo získali, budeme musieť prvky v strome rozmiestniť nejako systematicky.

Predpokladajme, že máme vo vrcholoch binárneho stromu navzájom rôzne prvky, ktoré sa dajú usporiadať. Potom takýto binárny strom voláme *vyhľadávací*, ak v každom vrchole v platí: Prvky, ktoré sú v pravom synovi v a v jeho podstrome (vo vrcholoch pod ním) sú väčšie ako prvok vo v. A naopak, prvky, ktoré sú v ľavom synovi v a vo vrcholoch pod ním sú menšie ako prvok vo v.

Príklad vyhľadávacieho stromu nájdete na obrázku 1.2.

1.2 Prechádzanie binárneho stromu a jeho zápisy

FIXME: preorder, inorder, postorder, utriedené čísla v lineárnom čase



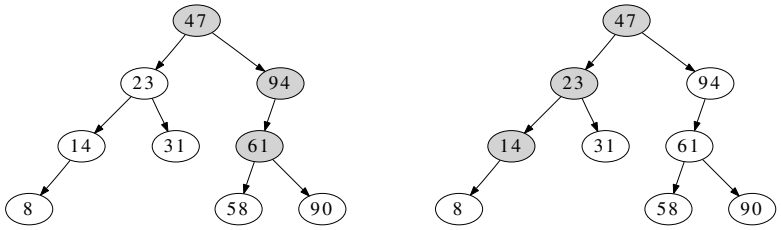
Obr. 1.2: Binárny vyhľadávací strom.

1.3 Operácie s binárnym vyhľadávacím stromom

Ak máme binárny vyhľadávací strom s hĺbkou h, tak vieme v čase $O(h)$ zistiť, či sa v ňom daný prvok x nachádza alebo nie. (Slovo "vyhľadávací" v názve nášho stromu pochádza práve z tejto jeho vlastnosti – vieme v ňom efektívne vyhľadávať.)

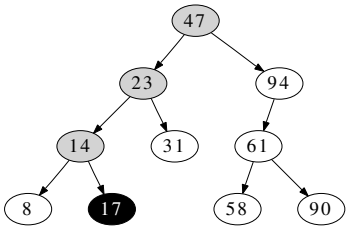
Začneme tým, že x porovnáme s prvkom v koreni. Ak nastala rovnosť, môžeme dať odpoveď "áno" a skončiť. Ak je prvok x od koreňa menší, vieme, že ak x je v našom strome, musí byť v ľavom podstrome. No a v opačnom prípade musí x byť v pravom podstrome.

Tento postup môžeme opakovať, až kým buď x nenájdeme, alebo nenastane situácia, kedy zistíme, že podstrom, ktorý by mal obsahovať x, je prázdny.



Obr. 1.3: Vrcholy navštívené pri hľadaní 61 (obrázok vľavo) a 17 (vpravo).

Vkladanie nového prvku x do binárneho vyhľadávacieho stromu je tiež jednoduché a rovnako efektívne. Práve popísaným spôsobom buď zistíme, že x už v našom strome je, alebo nájdeme miesto, kde v strome patrí x, ale tam nič nie je. Ak nastal druhý prípad, jednoducho pridáme na danom mieste do stromu nový list.



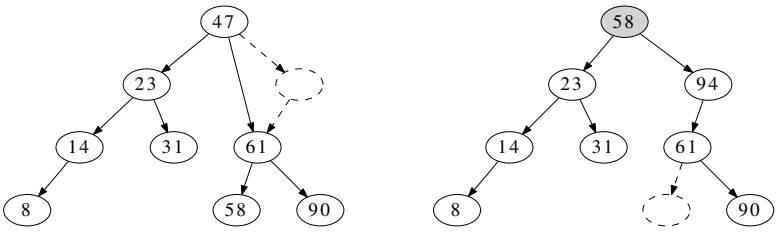
Obr. 1.4: Vloženie nového prvku 17.

Vymazanie prvku x zo stromu bude mierne komplikovanejšie. Rozlíšime preto tri prípady:

1. Ak je prvok x v liste: Odstránime zo stromu príslušný list.
2. Ak má prvok x len jedného syna s: Odstránime x zo stromu. Prvok s bude odteraz synom otca prvku x. (Alebo ak bol doteraz x koreňom, bude odteraz koreňom s.)

3. Ak má prvok x aj ľavého syna l aj pravého syna p: Nájdeme v podstrome s koreňom p prvok q s najmenšou hodnotou. (Ten nájdeme jednoducho tak, že z p ideme doľava, kým sa dá.) Tento prvok zo stromu zmažeme použitím pravidla 2. Následne prvok x nahradíme prvkom q.

Prípady 2 a 3 nájdete znázornené na obrázku 1.5.



Obr. 1.5: Strom z obr. 1.2 po zmazaní 94 (vľavo), resp. 47 (vpravo).

Vyvažovanie

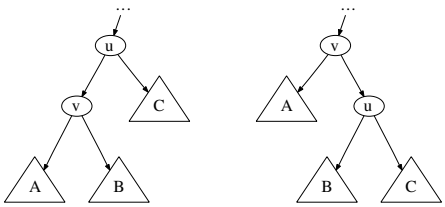
V predchádzajúcom texte sme ukázali, že keď máme binárny vyhľadávací strom hĺbky h, vieme v čase lineárnom od h pridať nový prvok, vymazať existujúci prvok aj overiť, či náš strom daný prvok obsahuje.

V priemernom prípade budú prvky v strome rozmiestnené približne rovnomerne, a teda očakávaná hĺbka binárneho vyhľadávacieho stromu s N vrcholmi bude $O(\log N)$.

Avšak v najhoršom možnom prípade sa nám môže stať, že náš strom "zdegeneruje" až do podoby spájaného zoznamu. Vyskúšajte si napríklad vyššie popísaným postupom postupne vložiť do stromu čísla od 1 po 100. Takýchto patologických prípadov by sme sa radi zbavili.

Dôležité je uvedomiť si, že množinou prvkov nie je tvar binárneho vyhľadávacieho stromu ani zďaleka určený. Ak nám teda niekedy vznikne príliš hlboký a "málo košatý" strom, môžeme sa pokúsiť upraviť ho do krajšej podoby.

Základným nástrojom na úpravu binárnych vyhľadávacích stromov je *rotácia*. Rotácia vyzerá tak, že si zvolíme niektorý vrchol iný od koreňa, vymeníme ho s jeho otcom a "prevešiame" ich podstromy tak, aby sme opäť dostali binárny vyhľadávací strom.



Obr. 1.6: Schematicky znázornená rotácia určená vrcholom v. Naľavo je strom pred rotáciou, napravo po nej. A, B a C sú ľubovoľné podstromy, rotáciou sa nemenia.

Na obrázku 1.6 je znázornená rotácia určená vrcholom v. Všimnite si, že inorder zápis oboch stromov je rovnaký, a teda ak bol pôvodný strom vyhľadávací, bude aj strom po rotácii vyhľadávací. Tiež si všimnite, že všetkým vrcholom v podstrome A sa rotáciou zmenšila hĺbka. (A naopak, vrcholom v C sa hĺbka zväčšila.)

Cvičenie 1.3.1 *Ako bude vyzerať strom, ktorý dostaneme zo stromu na obrázku 1.6 vpravo rotáciou určenou vrcholom u?*

Cvičenie 1.3.2 *Dokážte alebo vyvráťte tvrdenie: Ak dva binárne vyhľadávacie stromy obsahujú tú istú množinu prvkov, tak sa určite dá z jedného konečnou postupnosťou rotácií vyrobiť druhý.*

Druhy vyvažovaných stromov

Historicky prvým vyvažovaným stromom bol AVL strom[1].

AVL je binárny vyhľadávací strom založený na nasledujúcej myšlienke. *Vyváženosť* vrcholu nech je hĺbka jeho pravého podstromu mínus hĺbka jeho ľavého podstromu. V AVL strome si v každom vrchole budeme (okrem prvku a ukazovateľov) pamätať jeho vyváženosť. Vrchol voláme *vyvážený*, ak je jeho vyváženosť -1, 0 alebo 1. Budeme chcieť dosiahnuť, aby všetky vrcholy nášho stromu boli vyvážené.

Totiž vieme dokázať,[2] že ak máme binárny strom s N vrcholmi, ktoré sú všetky vyvážené, tak tento strom má hĺbku $\Theta(\log N)$.

Operácie s AVL stromom sa príliš nelíšia od operácií s klasickým binárnym vyhľadávacím stromom. Vyhľadávanie vyzerá presne rovnako. Pri vkladaní vložíme nový prvok ako list (postupom pre klasický strom). Tým sa nám ale mohla porušiť vyváženosť vrcholov na ceste z koreňa do práve vloženého listu. Preto sa budeme postupne vracať z listu do koreňa, a vždy, keď narazíme na nevyvážený vrchol, pomocou vhodných rotácií to napravíme. Analogicky funguje aj vymazávanie – vymažeme prvok ako z klasického stromu a následne po ceste z miesta, kde sme vymazali vrchol, do koreňa naprávame vyváženosť vrcholov.

Presný popis operácií s AVL stromom nájdete napr. v [**?**] alebo [**?**], prípadne na Wikipédii.

[1] Nazvaný podľa mien autorov: Adeľson-Veľskij a Landis

[2] Dá sa napríklad matematickou indukciou dokázať, že keď máme binárny strom hĺbky h so samými vyváženými vrcholmi, tak má najmenej $F_{h+3} - 1$ vrcholov, pričom F_i je i-te Fibonacciho číslo. Tie sú definované rekurentným predpisom $F_0 = 0$, $F_1 = 1$ a $F_{i+2} = F_{i+1} + F_i$ pre $i \geq 0$. Platí, že F_i je asymptoticky rovné Φ^i pre $\Phi = (1 + \sqrt{5})/2 \doteq 1.618$. A teda strom s N vyváženými vrcholmi môže mať hĺbku nanajvýš rádovo rovnú $\log_\Phi N$.