Binární vyhledávání a BST

Karel Horák, Petr Ryšavý

23. března 2016

Katedra počítačů, FEL, ČVUT

Příklad 1

Naimplementujte binární vyhledávání. Upravte metodu `BinarySearch::binarySearch`.

Příklad 2

Myslím si číslo mezi 1 a 10000. Navrhněte algoritmus, který uhodne číslo, které si myslím. Můžete pokládat pouze otázky, na které je odpověď ano/ne. Protože jsem líný, snažte se co nejmenší počet otázek.

Kolik otázek budete muset maximálně položit, než odhalíte číslo, které si myslím.

Řešení 2

Vhodným řešením je použít binární vyhledávání. Vždy máme k dispozici nějaký interval, ze kterého je myšlené číslo. Interval rozdělíme na dvě poloviny a zeptáme se otázku, zda je myšlené číslo větší než prostřední bod intervalu. Podle odpovědi interval nahradíme jeho levou nebo pravou polovinou.

K řešení je třeba $\lceil \log_2 10000 + 1 \rceil$ otázek, což je 15 otázek.

Příklad 3

Čísla ze zadané posloupnosti postupně vkládejte do prázdného binárního vyhledávacího stromu (BVS). Jak bude vypadat takto vytvořený BVS?

Poté postupně odstraňte první tři prvky. Jak bude vypadat výsledný BVS?

1. $14, 24, 5, 13, 1, 3, 22, 10, 19, 11$
2. $10, 16, 5, 17, 4, 15, 3, 1, 23, 13, 2, 11$

Příklad 4

Mějme klíče $1, 2, 3, \ldots, n$. Číslo n je liché. Nejprve vložíme do BVS všechny sudé klíče v rostoucím pořadí a pak všechny liché klíče také v rostoucím pořadí.

Jaká bude hloubka výsledného stromu?

Změnil by se nějak tvar stromu, kdybychom lichá čísla vkládali v náhodném pořadí?

Řešení 4

Hloubka výsledného stromu bude $\frac{n-1}{2}$.[1] Pokud vložíme všechny sudé uzly, bude hloubka $\frac{n-1}{2} - 1$. Po přidání lichých uzlů se hloubka zvýší o jedna, protože n bude vždy napravo od $n-1$.

Pokud bychom vkládali lichá čísla v náhodném pořadí, strom se nezmění. Jeho tvar je již určen vloženými sudými čísly.

[1] Předpokládáme, že hloubka stromu o jednom uzlu je 0.

Příklad 5

V jakém pořadí vypíšeme prvky binárního vyhledávacího stromu, pokud ho projdeme inorder?

Řešení 5

Prvky budou seřazené podle velikosti.

Příklad 6

Předpokládejme, že binární vyhledávací strom obsahuje přirozená čísla mezi 1 a 1000. Které z následujících sekvencí navštívených uzlů nemohou nastat, pokud hledáme klíč 363?

1. $2, 252, 401, 398, 330, 363$
2. $399, 387, 219, 266, 382, 381, 278, 363$
3. $3, 923, 220, 911, 244, 898, 258, 362, 363$
4. $4, 924, 278, 347, 621, 299, 392, 358, 363$
5. $5, 925, 202, 910, 245, 363$

[`http://www.cs.princeton.edu/courses/archive/fall12/cos126/precepts/BSTex.pdf`]

Řešení 6

Možnost 4 nemůže nastat. Hodnota 299 nemůže být po 621, protože to by pak byla v levém podstromě uzlu s klíčem 347.

Příklad 7

Jaká je asymptotická složitost operací `insert`, `find` a `delete` pro

1. vyvážený binární vyhledávací strom;
2. obecný binární vyhledávací strom.

Řešení 7

V případě vyváženého binárního vyhledávacího stromu všechny operace trvají $\Theta(\log n)$.

V případě obecného binárního vyhledávacího stromu mohou trvat až $\mathcal{O}(n)$.

Příklad 8

V jakém pořadí máme vkládat $2^n - 1$ prvků do binárního vyhledávacího stromu tak, aby byl vyvážený? Formulujte nutnou a postačující podmínku, aby byl výsledný vyhledávací strom vyvážený.

Řešení 8

Existuje právě jeden vyvážený strom o daných $2^n - 1$ prvcích. Pokud ho chceme vybudovat, je nutnou a postačující podmínkou, abychom získali stejný strom, že vkládáme vždy uzel dříve než jeho potomky. Takto docílíme toho, že strom budujeme odshora.

Při vkládání je jen jedno možné místo, kam lze nový uzel vložit. Pokud vkládáme další uzel a jeho rodič a všechny uzly na cestě ke kořeni z vyváženého stromu jsou již vloženy do stromu, pak je uzel vložen na stejné místo jako ve vyváženém stromě. Protože uzel vkládáme vždy před jeho potomky, platí, že získáme vyvážený strom.

Příklad 9

Mějme binární vyhledávací strom. Jaká je asymptotická složitost operace, která spočte počet klíčů, jejichž hodnota je menší než x, pokud do uzlů nepřidáváme žádné pomocné informace?

Navrhněte efektivnější algoritmus, pokud si můžete uložit pomocné informace.

Řešení 9

Pokud nepřidáváme žádné další informace do stromu, je nutné BST procházet v pořadí inorder, dokud nenarazíme na klíč, který je alespoň x. Počet navštívených uzlů je pak požadované číslo. Složitost operace je tedy $\mathcal{O}(n)$.

Efektivnějším by mohlo být pamatovat si velikost každého podstromu. Pak by nám stačilo vyhledat prvek s hodnotou x a přičíst velikosti všech levých podstromů vrcholů cesty, které jsme nenavštívili plus počet navštívených uzlů, které měly klíče menší. Asymptotická složitost je $\mathcal{O}(n)$, ale v případě vyváženého stromu poklesne na $\Theta(\log_2 n)$.

Příklad 10

Naimplementujte vyhledání minima v binárním vyhledávacím stromu a operaci `delete`.

Programovací úlohy k procvičení

`https://open.kattis.com/problems/insert`