Binární vyhledávací strom

Aleksandr Shevchenko

30. dubna 2022

Vysoké učení technické v Brně
Fakulta informačních technologií

Obsah prezentace

Motivace

Proč potřebujeme řadicí algoritmy?

Pro rychlé vyhledávání a představu dat. Některé úlohy jsou nevyřešitelné bez zařazování.

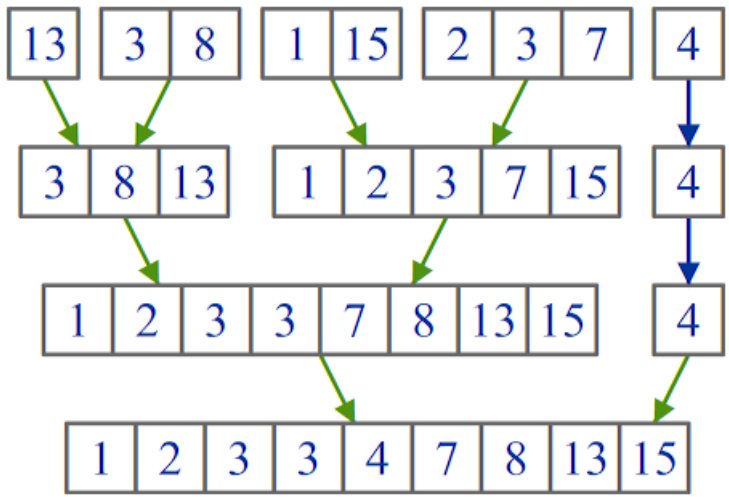
Motivace

Proč potřebujeme řadicí algoritmy?

Pro rychlé vyhledávání a představu dat. Některé úlohy jsou nevyřešitelné bez zařazování.

Některé populární algoritmy:

- Bubble sort
- Shakesort
- Quicksort
- Radix sort
- Binary search tree



Binární vyhledávací strom

Definice

Binární vyhledávací strom (Binary Search Tree – BST) je datová struktura založená na binárním stromu, umožňující rychlé vyhledávání daných hodnot.

Základní vlastnosti:

- Levý podstrom uzlu obsahuje pouze uzly s klíči menšími než klíč uzlu.
- Pravý podstrom uzlu obsahuje pouze uzly s klíči většími než klíč uzlu.
- Levý a pravý podstrom musí být také binárním vyhledávacím stromem.

Vyhledávání klíče

Kroky pro vyhledávání klíče v BST:

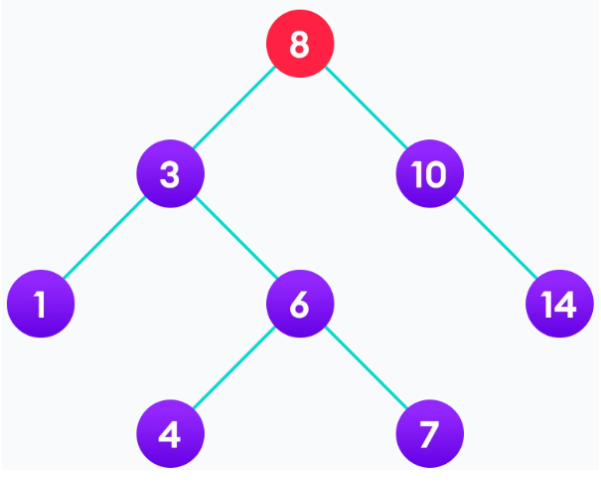
1. Ujístíme se, že daná hodnota je v povoleném rozsahu.
2. Hledáme kořenový uzel stromu.
3. Pokud daná hodnota je menší než hodnota klíče uzlu, jdeme do levého podstromu, jinak do pravého.
4. Pokud nenajdeme naší hodnotu nebo se nenarazíme na konec, opakujeme krok 3.

Zkusíme na příkladu najít 6.

Vyhledávání klíče

Kroky pro vyhledávání klíče v BST:

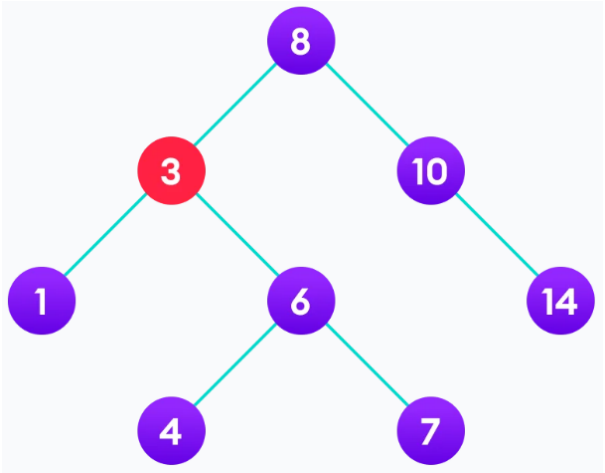
1. Ujístíme se, že daná hodnota je v povoleném rozsahu.
2. Hledáme kořenový uzel stromu.
3. Pokud daná hodnota je menší než hodnota klíče uzlu, jdeme do levého podstromu, jinak do pravého.
4. Pokud nenajdeme naší hodnotu nebo se nenarazíme na konec, opakujeme krok 3.



Vyhledávání klíče

Kroky pro vyhledávání klíče v BST:

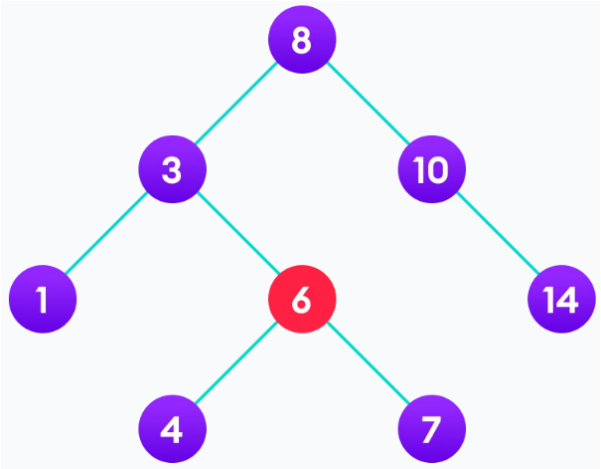
1. Ujístíme se, že daná hodnota je v povoleném rozsahu.
2. Hledáme kořenový uzel stromu.
3. Pokud daná hodnota je menší než hodnota klíče uzlu, jdeme do levého podstromu, jinak do pravého.
4. Pokud nenajdeme naší hodnotu nebo se nenarazíme na konec, opakujeme krok 3.



Vyhledávání klíče

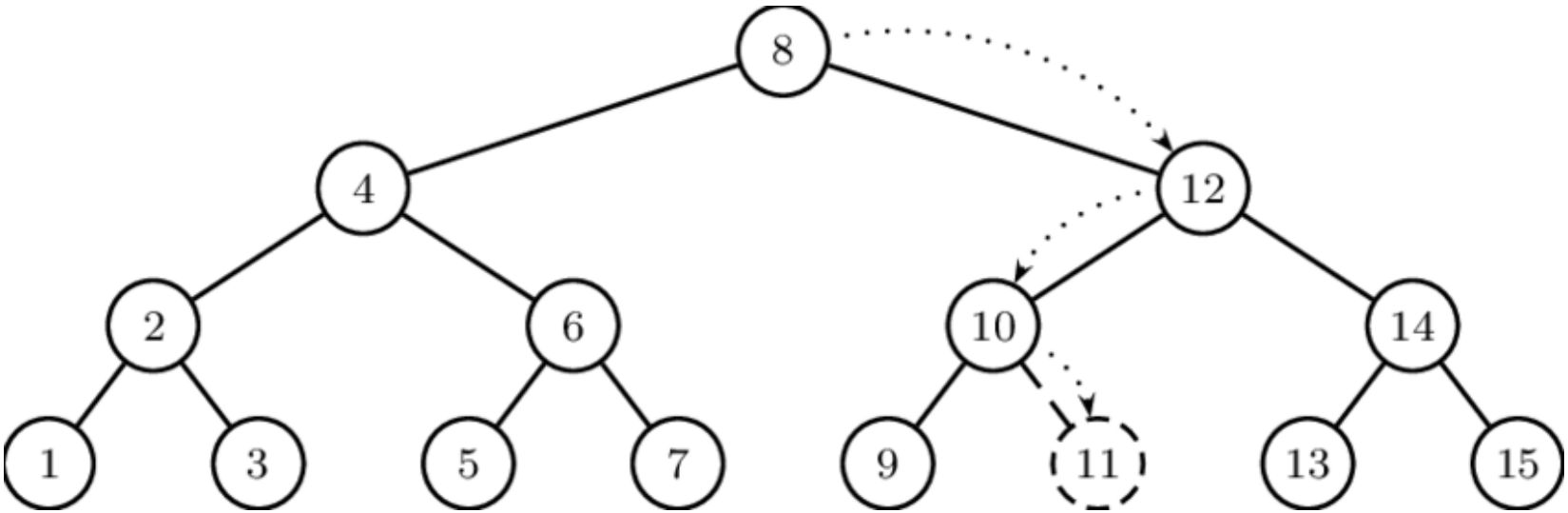
Kroky pro vyhledávání klíče v BST:

1. Ujístíme se, že daná hodnota je v povoleném rozsahu.
2. Hledáme kořenový uzel stromu.
3. Pokud daná hodnota je menší než hodnota klíče uzlu, jdeme do levého podstromu, jinak do pravého.
4. Pokud nenajdeme naší hodnotu nebo se nenarazíme na konec, opakujeme krok 3.



Vložení klíče

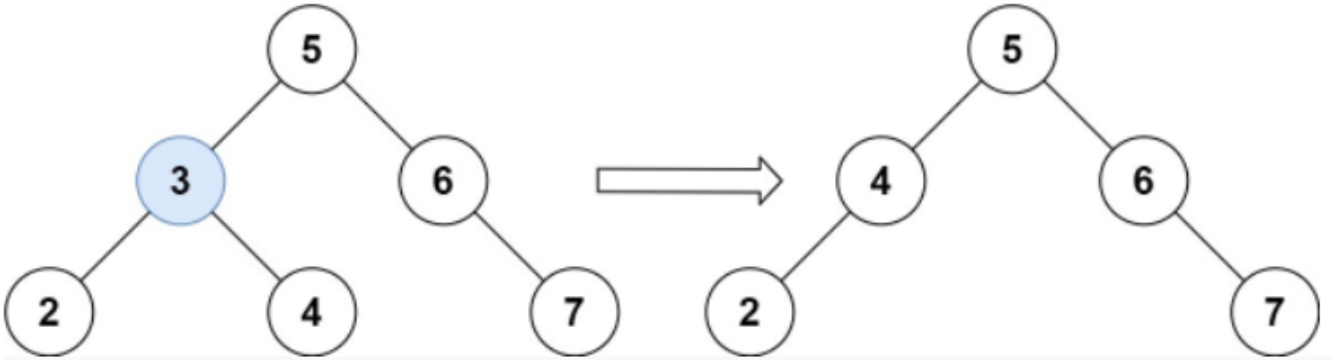
Nový klíč vždy vkládáme na listové úrovni. Hledáme ho pokud nenarazíme na situaci, když nemůžeme jít hlouběji a přidáváme k uzlu nový list.



Smazání klíče

Při smazání klíče můžeme se narazit na několik situací:

- Uzel pro smazání je **listem** – jednoduše ho mažeme ze stromu.
- Uzel pro smazání má **jedno dítě** – kopírujeme dítě do uzlu a mažeme dítě.
- Uzel pro smazání má **dvě děti** – v pravém podstromu uzlu hledáme uzel s nejmenší hodnotou, kopírujeme jeho obsah do původního uzlu a mažeme list. Pokud pravý podstrom neexistuje, děláme to samé s největší hodnotou levého podstromu.



Pseudokód BST

Algoritmus 1: Pseudokód pro vyhledávání klíče v BST

```
1:  search (value, root) {
2:  if (root == NULL) then
3:      return  NULL;
4:  else if (root.data == value) then
5:      return  root;
6:  else if (root.data > value) then
7:      return  search (value, root.left);
8:  else
9:      return  search (value, root.right);
10: end if
11: }
```

Složitost BST, závěr

Operace s BST se provádí rychleji, než s obyčejným polem. Při práci s polem potřebujeme $O(n)$ kroků, kde n je velikost pole.

Pro práci se stromem provádíme $O(h)$ operací, kde h je maximální hloubka stromu. V nejoptimálnějším případě, když hloubka všech listů je stejná, strom má $n = 2^h - 1$ vrcholů, takže $O(h) = O(\log n)$.

Musíme ale dávat pozor, aby se BST nestal polem. Naštěstí, existují metody optimalizace hloubky stromů pro jejich vybalancování.

Použité zdroje

- Algoritmy.net: Porovnání řadicích algoritmů
- Habr.com: struktury dannych: binarnye derevya
- GeeksforGeeks: Binary Search Tree
- Programiz: Binary Search Tree

Děkuji za pozornost!