DELTA TopGun

08 - Binární vyhledávací strom

Tomáš Faltejsek, Luboš Zápotočný, Michal Havelka

2023

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

Spojový seznam - složitosti

- *index*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

Spojový seznam - složitosti

- *index* $\Theta(n)$
- *vložení na začátek*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

Spojový seznam - složitosti

- *index* $\Theta(n)$
- *vložení na začátek* $\Theta(1)$
- *vložení na konec*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

Spojový seznam - složitosti

- *index* $\Theta(n)$
- *vložení na začátek* $\Theta(1)$
- *vložení na konec* $\Theta(n)$ bez tail ukazatele
- *vložení na aktuální pozici*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

Spojový seznam - složitosti

- *index* $\Theta(n)$
- *vložení na začátek* $\Theta(1)$
- *vložení na konec* $\Theta(n)$ bez tail ukazatele
- *vložení na aktuální pozici* $\Theta(1)$
- *vyhledání prvku*

Rekapitulace – dynamické pole, spojový seznam

Dynamické pole - složitosti

- *index* $\Theta(1)$
- *vložení na začátek* $\Theta(n)$
- *vložení na konec* $\Theta(1)$ amortizováno
- *vložení na aktuální pozici* $\Theta(n)$
- *vyhledání prvku* $\Theta(n)$ neseřazené

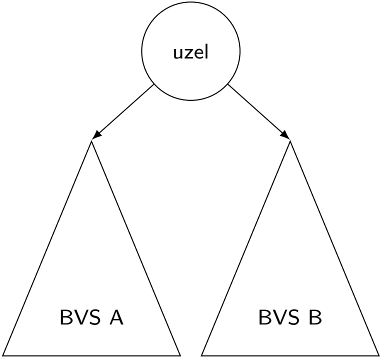
Spojový seznam - složitosti

- *index* $\Theta(n)$
- *vložení na začátek* $\Theta(1)$
- *vložení na konec* $\Theta(n)$ bez tail ukazatele
- *vložení na aktuální pozici* $\Theta(1)$
- *vyhledání prvku* $\Theta(n)$

Seřazené dynamické pole

Za předpokladu seřazeného pole můžeme zredukovat složitost operace *index* a *vyhledání prvku* na $\Theta(\log n)$. **Mutační operace ale stále mají neefektivní složitost kvůli režii spojenou s přesunem prvků. Existuje vhodnější struktura? Diskutujte.**

Binární vyhledávací strom – vlastnosti

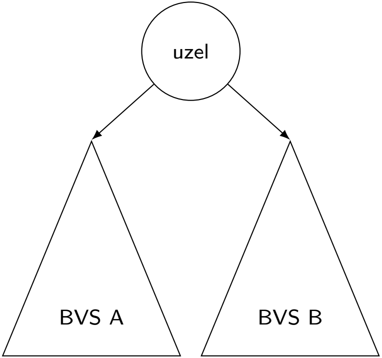


BVS = binární vyhledávací strom

Vlastnosti

- BVS je speciální případ binárního stromu

Binární vyhledávací strom – vlastnosti

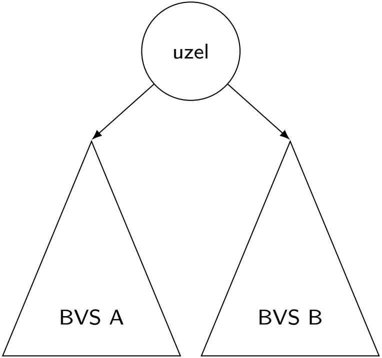


BVS = binární vyhledávací strom

Vlastnosti

- BVS je speciální případ binárního stromu
- **levý potomek** každého uzlu nabývá **menší** hodnoty

Binární vyhledávací strom – vlastnosti

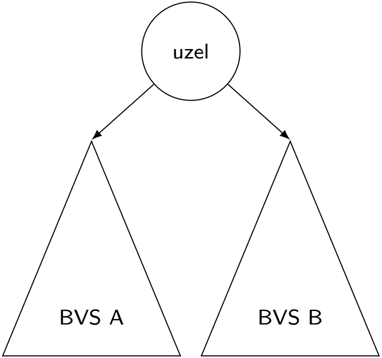


BVS = binární vyhledávací strom

Vlastnosti

- BVS je speciální případ binárního stromu
- **levý potomek** každého uzlu nabývá **menší** hodnoty
- **pravý potomek** každého uzlu nabývá **větší** hodnoty

Binární vyhledávací strom – vlastnosti

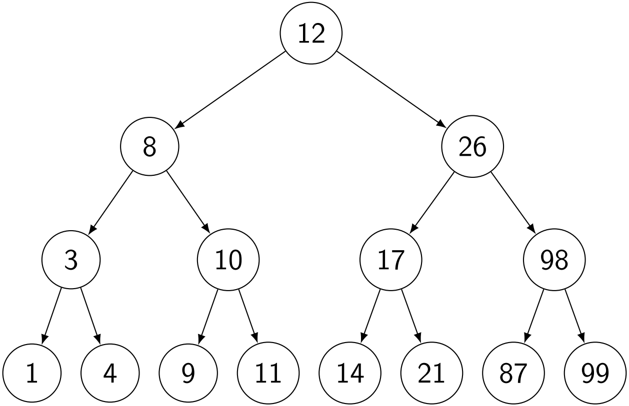


BVS = binární vyhledávací strom

Vlastnosti

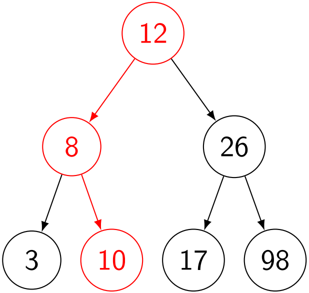
- BVS je speciální případ binárního stromu
- **levý potomek** každého uzlu nabývá **menší** hodnoty
- **pravý potomek** každého uzlu nabývá **větší** hodnoty
- *BVS A* a *BVS B* jsou rekurzivní reprezentací podstromů a nabývají vlastností binárního vyhledávacího stromu

Binární vyhledávací strom



Binární vyhledávací strom – vyhledávání

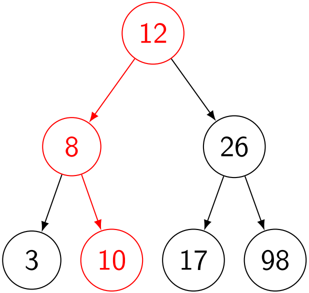
Ilustrace procedury *Search(10)*:



Složitost – vyvážený BVS *Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o polovinu. Kolik musíme v nejhorším případě provést kroků, abychom našli prvek?*

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(10)*:

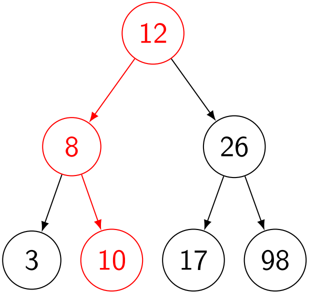


Složitost – vyvážený BVS *Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o polovinu. Kolik musíme v nejhorším případě provést kroků, abychom našli prvek?*

1. $n = 2^0 + 2^1 + \cdots + 2^{k-1}$

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(10)*:

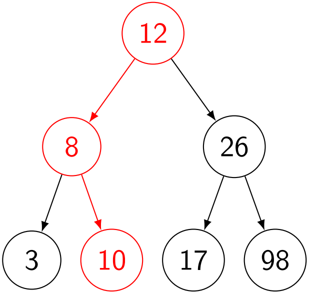


Složitost – vyvážený BVS *Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o polovinu. Kolik musíme v nejhorším případě provést kroků, abychom našli prvek?*

1. $n = 2^0 + 2^1 + \cdots + 2^{k-1}$
2. $\Rightarrow n = 2^k - 1$

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(10)*:

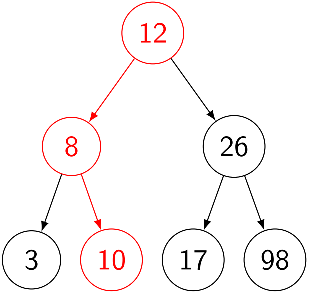


Složitost – vyvážený BVS *Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o polovinu. Kolik musíme v nejhorším případě provést kroků, abychom našli prvek?*

1. $n = 2^0 + 2^1 + \cdots + 2^{k-1}$
2. $\Rightarrow n = 2^k - 1$
3. $k = \log_2(n+1)$

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(10)*:



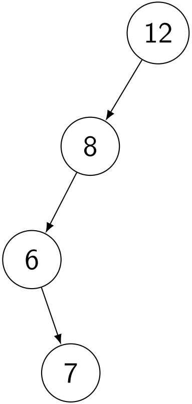
Složitost – vyvážený BVS *Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o polovinu. Kolik musíme v nejhorším případě provést kroků, abychom našli prvek?*

1. $n = 2^0 + 2^1 + \cdots + 2^{k-1}$
2. $\Rightarrow n = 2^k - 1$
3. $k = \log_2(n+1)$
4. $\Rightarrow \Theta(\log_2 n)$

kde k je hloubka stromu

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(4)*:

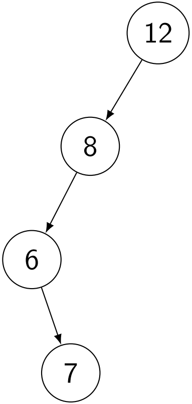


Složitost – NEvyvážený BVS

Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o 1. Kolik musíme provést kroků, abychom našli prvek?

Binární vyhledávací strom – vyhledávání

Ilustrace procedury *Search(4)*:



Složitost – NEvyvážený BVS

Při každém kroku prohledávání je prohledávací prostor (počet uzlů) redukován o 1. Kolik musíme provést kroků, abychom našli prvek?

$n \to n-1 \to n-2 \to \cdots \to 1$

$\Rightarrow \Theta(n)$

Vyvážení BVS

Vyvážení BVS

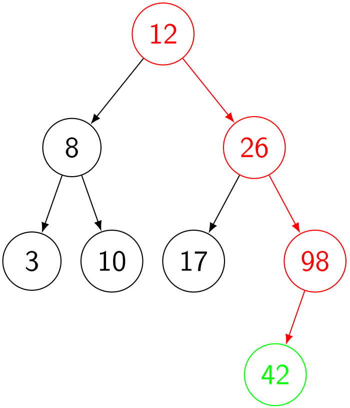
Jak vyvážit binární vyhledávací strom tak, abychom zachovali optimální vlastnosti a složitost operací $\Theta(\log_2 n)$?

Binární vyhledávací strom – vložení prvku

Ilustrace procedury *Insert(42)*:

Binární vyhledávací strom – vložení prvku

Ilustrace procedury *Insert(42)*:

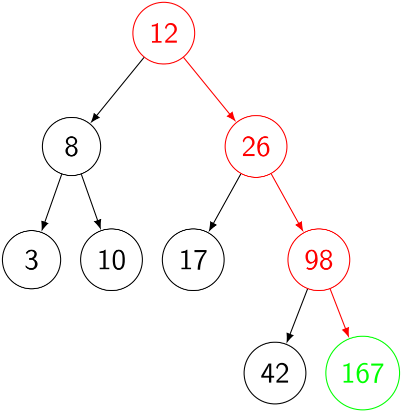


Binární vyhledávací strom – vložení prvku

Ilustrace procedury *Insert(167)*:

Binární vyhledávací strom – vložení prvku

Ilustrace procedury *Insert(167)*:

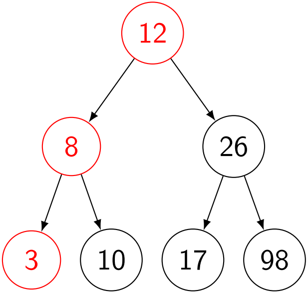


Binární vyhledávací strom – smazání prvku (list)

Ilustrace procedury *Delete(3)*:

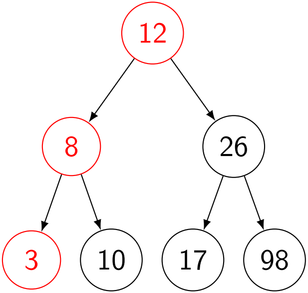
Binární vyhledávací strom – smazání prvku (list)

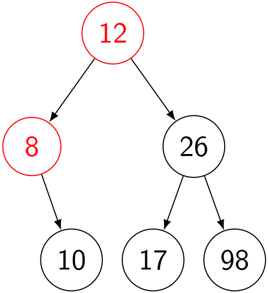
Ilustrace procedury *Delete(3)*:



Binární vyhledávací strom – smazání prvku (list)

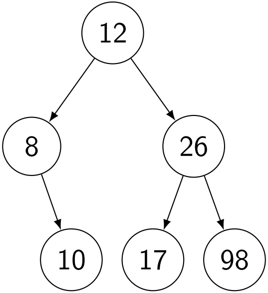
Ilustrace procedury *Delete(3)*:





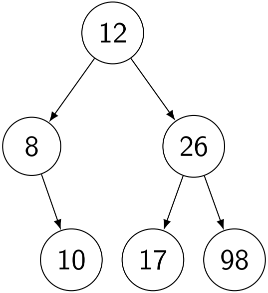
Binární vyhledávací strom – smazání prvku (jeden potomek)

Ilustrace procedury *Delete(8)*:



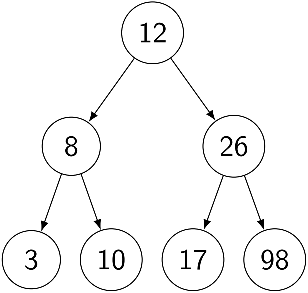
Binární vyhledávací strom – smazání prvku (jeden potomek)

Ilustrace procedury *Delete(8)*:



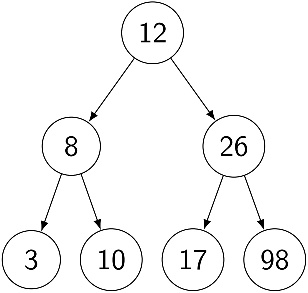
Binární vyhledávací strom – smazání prvku (dva potomci)

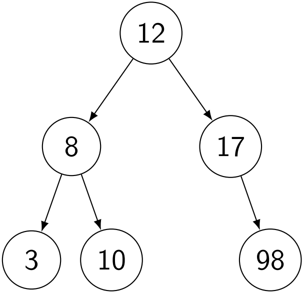
Ilustrace procedury *Delete(26)*:



Binární vyhledávací strom – smazání prvku (dva potomci)

Ilustrace procedury *Delete(26)*:





Operace smazání – pseudokód

```
TREE-DELETE(T, z)
if z.left == NIL
    TRANSPLANT(T, z, z.right)          // z has no left child
elseif z.right == NIL
    TRANSPLANT(T, z, z.left)           // z has just a left child
else y = TREE-MINIMUM(z.right)         // y is z's successor
    if y.p ≠ z                         // y lies within z's right subtree
        TRANSPLANT(T, y, y.right)      but is not the root of this subtree.
        y.right = z.right
        y.right.p = y
    TRANSPLANT(T, z, y)                // Replace z by y.
    y.left = z.left
    y.left.p = y
```

¿¿

Operace smazání – pseudokód – prohození podstromů

```
TRANSPLANT(T, u, v)
if u.p == NIL
        T.root = v
elseif u == u.p.left
        u.p.left = v
else u.p.right = v
if v ≠ NIL
        v.p = u.p
```

(Vyrovnaný) Binární vyhledávací strom – složitost operací

- *index* $\Theta(\log n)$
- *vložení na začátek* $\Theta(\log n)$
- *vložení na konec* $\Theta(\log n)$
- *vložení na prostředek* $\Theta(\log n)$
- *vyhledání prvku* $\Theta(\log n)$

Vyrovnaný vyhledávací strom

Složitosti $\Theta(\log n)$ dosáhneme pouze pokud zajistíme tzv. *vyrovnaný vyhledávací strom*. Způsoby vyrovnání binárního vyhledávacího stromu budeme detailně demonstrovat v příští přednášce.

Ukázka - Threaded Binary Tree

Alternativa rekurzivní implementace inorder průchodu, s použitím *pomocného (auxiliary)* stacku

– Demonstrace na tabuli –

Vyvážení Binárního vyhledávacího stromu - syntaktické stromy

– Bude detailně rozebráno v přednášce č. 9 –