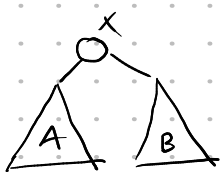
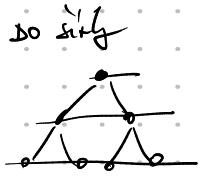
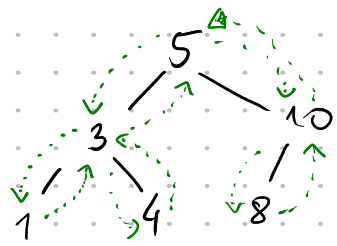
# průchod stromem do hloubky




rekurzivně projít A
navštívit B
X



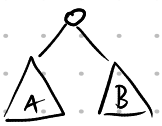




```
[in-order_walk
<= r: Node
  if (r = nil) then return
  in-order-walk(r.left)
→ visit r
  in-order-walk(r.right)
```

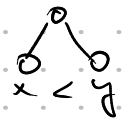
1 3 4 5 8 10

Věta: a) Navštíví každý vrchol právě jednou.
proč: Indukcí pes výšku: 0 ... ⊙

h




b) Navštíví vrcholy ve vzestupném pořadí.
Pokud existuje vrchol s klíčem menším než r.key, pak je v levém podstromu.




c) složitost: $\Theta(n)$, n je počet vrcholů, visit je v $\Theta(1)$
$\Theta(n \cdot f(n))$ visit je v $\Theta(f(n))$.

Otázka:

pole klíčů (nebo vrcholů s klíči)
klíče jsou různé

úkol: sestavte vyhledávací strom

Najděte algo a jakou může mít složitost?

A

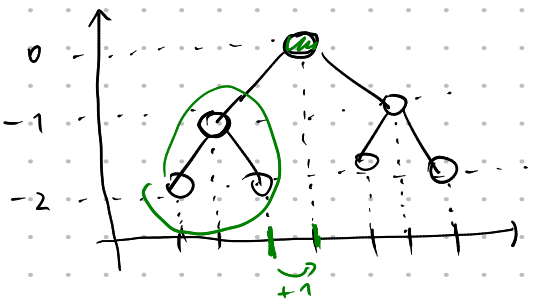
Třídící algoritmus

1. pomocí A sestavím strom
2. udělám in-order-walk na stromu z 1.
   (zapíšu klíč do pole)

$\Omega(n \log n)$ ← lower bound na třídění porovnáváním

2. má složitost $\Theta(n)$, 1. musí být $\Omega(n \log n)$.

Malování stromu (bin. strom) průchodem do hloubky (in-order-walk)




Pravidla pro kreslení:
souřadnice vrcholu v: v.x
v.y

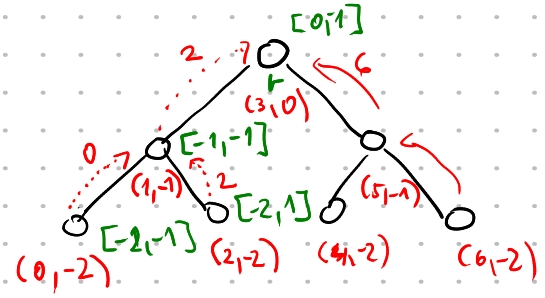
v.x. ~ pořadí vrcholu v průchodu

Po strukly pro vrchol doplníme položky pro x, y.

```
draw
← r : Node
← y : Int      --hloubka
← x : Int      ... nejpravější x-souřadnice již nakresleného vrcholu
⟶ Int          x-ová souřadnice nejpravějšího namalovaného vrcholu.
```

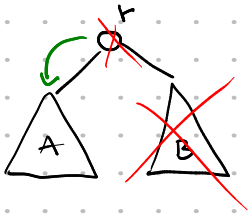
```
1. IF (r=nil) then return x
2. r.x ← draw(r.left, y-1, x) + 1
3. r.y ← y
4. return draw(r.right, y-1, r.x)
```

draw(r, 0, -1)




---

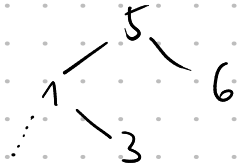
findmin




pokud je A = r.left neprázdný, existuje ve stromu vrchol s klíčem menším než r.key

Levý vrcholy vynechám.

pokud je A prázdný, je r.key nejmenší klíč.




⇒ min nemusí být list !

Složitost: Podstromy, kdy prohledávám má menší výšku než r.

$\Theta(h)$, kde h je výška stromu.

Pořádkový následník vrcholu x:

$\{ y \mid y.key > x.key \}$ (prázdná pk. $x \neq max$)

Min z této množiny je pořádkový následník x.

(vrchol s min klíčem)

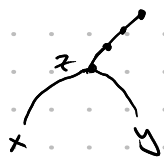
Věta: Pro každé x, y takové, že x.key > y.key platí jedna z následujících možností:

a) x je v pravém podstromu y. [y je předek x]
b) y je v levém podstromu x [x je předek y]
c) existuje vrchol z, x je v jeho pravém podstromu, y je v jeho levém podstromu.

Jak najít z bodu (c):

procházím od kořene,
jsou x, y ve stejném podstromu?
ANO: půjdu na kořen tohoto podstromu
NE: jsou v různých podstromech, podm. uspoř. ⇒ x v pravo, y vlevo.

Zastavím se? ANO: x a y mají společného předka,
A = (množina předků x ∩ množina předků y)



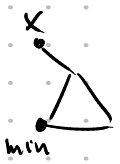

z je vrchol s největší hloubkou v A.

Věta (kde je pořádkový následník vrcholu x)

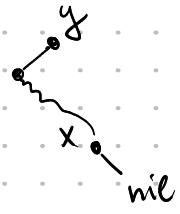
(a) pokud má x neprázdný pravý podstrom, je poř. následník vrcholu x minimem v tomto podstromu.

(b) pokud má x prázdný pravý podstrom, je poř. následník vrcholu x jeho nejhlubší předek y, takový y.left je předek x nebo přímo x.

(a)



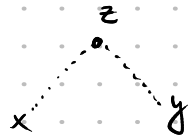

(b)



Důkaz: y .... pův. následník

pom. tvrzení: pokud y neleží v x.right, pak je předkem x.

Pokud y není předkem x, pak existuje z:



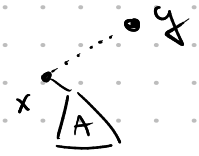

→ ale pak x.key < z.key < y.key )

to je spor tím, že y je pův. nás. x.

□

(a) Sporem. Předp. že y není v pravém podstromu.

Pak podle pom. tvrzení:




pro libovolné $w \in A$ máme

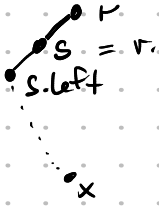
$x.key < w.key < y.key$

Spor s tím, že y je pův. následník x.

(b) r, s (hloubka s je větší než hloubka r)

r.left je předek x
s.left je předek x nebo přímo x

Ukážeme, že r nemůže být pův. následník x



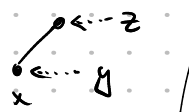

(toto je situace, kdy r je nejvíce)

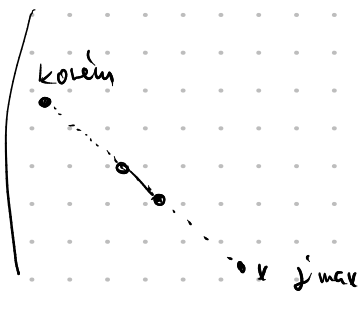
$x.key < s.key < r.key$

□

successor
← x: Node
→ Node (může být nil, pokud pův. nástl neexistuje)

```
if (x.right ≠ nil) then return find-min(x.right)
else
  y ← z
  while y.parent ≠ nil
    z ← y.parent
    if (y = z.left) then return z
    y ← z
  return nil
```




Složitost: min $\Theta(h)$, h je výška stromu
zbylé $\Theta(h)$

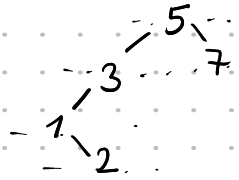
Insert – vložení vrcholu a

princip: kam vložit? Tam, kde ho najde search

1. Provedeme search a.key → skončí neúspěchem
2. Nechť z je poslední vrchol navštívený během 1.
   Vložím a jako potomka z.

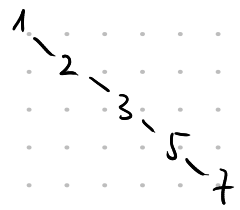
⇒ složitost je $\Theta(h)$, kde h je výška stromu.

Příklad: postupně vložíme 5, 3, 1, 7, 2,
5, 3, 7, 1, 2




postupně vložíme: 1, 2, 3, 5, 7

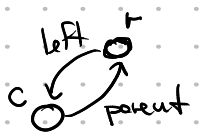
(1+2+3+···) ~ složitost je $O(n^2)$




kód

```
struct Tree
    root: Node
```

```
set-left-child
← r: Node
← c: Node (může být nil)
```




set-right-child

triviální

```
r.left ← c
if (c ≠ nil) then c.parent ← r
```

```
set-child
← r: Node } nejsou nil
← c: Node }
```

```
if (r.key > c.key) then set-left-child(r,c)
else set-right-child(r,c)
```

Insert
← t : Tree
← a : Node

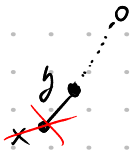
- a.left ← nil
  a.right ← nil
  IF (t.root = nil) t.root ← a
  else y ← poslední vrchol při search(t.root, a.key)
  set-child(y, a)

==

delete → odebrání vrcholu x ze stromu.

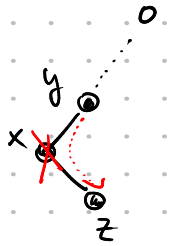
3 případy: podle toho, kolik má x potomků

→ 0 potomků → x je list



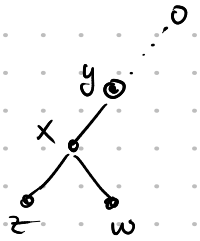

y.left = nil

→ 1 potomek

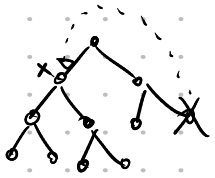



set-child(y, z)

→ 2 potomci




bez klíčů




... jdou snadno odebrat

ex. vrchol z, → nemá 2 potomky

→ lze ho dat místo x,
a neporušit uspořádání!

ANO: ~~pořádkový~~ následník x
⇒ minimum v x.left