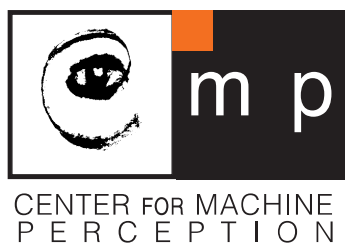
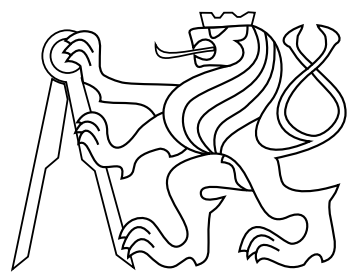
Stromy

Jan Kybic

http://cmp.felk.cvut.cz/~kybic
kybic@fel.cvut.cz

2016–2019





Stromy

Binární vyhledávací stromy

Množiny a mapy

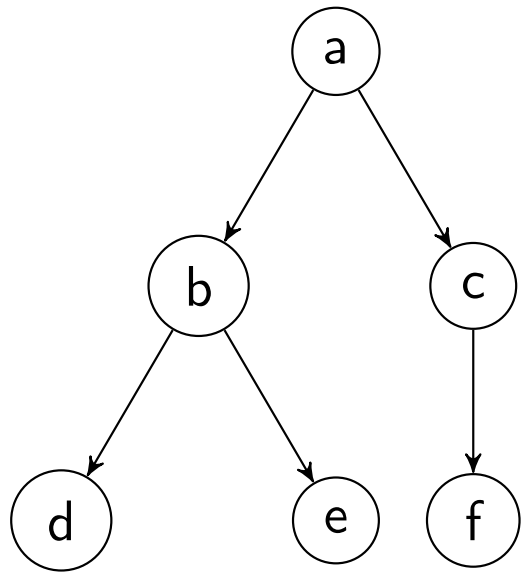
Strom
(Tree)

Strom

- skládá se s *uzlů (nodes)* spojených *hranami (edges)*.
- je souvislý a acyklický

Kořenový strom

- orientovaný graf, má jeden význačný uzel = *kořen (root)*
- z kořene vede do každého jiného uzlu právě jedna orientovaná cesta
- do kořene nevstupuje žádná hrana, do každého jiného uzlu vstupuje právě jedna hrana



Vlastnosti stromů

- každé dva uzly jsou spojeny právě jednou neorientovanou cestou
- počet hran = počet uzlů - 1
- pokud jednu hranu vyjmeme, graf bude nesouvislý
- pokud jednu hranu přidáme, graf bude obsahovat cyklus (kružnici)

Názvosloví

- kořen, list, vnitřní uzel, rodič, (pravý/levý) potomek (syn), sourozenci, stupeň uzlu, hloubka (výška)

Poziční strom

- potomci jsou označeni čísly (=levý/pravý)
- některý potomek může chybět

Binární strom

- poziční strom
- každý uzel má nanejvýš dva potomky.

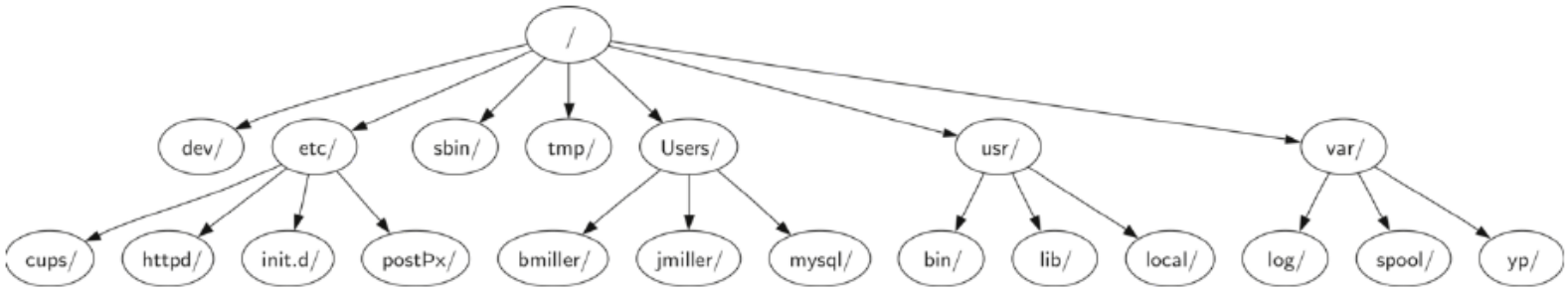
Úplný binární strom s n uzly

- každý uzel kromě listů má právě dva potomky
- Počet uzlů v hloubce i je 2^i
- $n = \sum_{i=0}^{h} 2^i = 2^{h+1} - 1$
- Všechny listy mají hloubku $h = \log_2(n+1) - 1$
- Počet listů je $(n+1)/2$, počet vnitřních uzlů je $(n-1)/2$.

Pro každý binární strom s n uzly a hloubkou h

$$\log_2(n+1) - 1 \leq h \leq n - 1$$

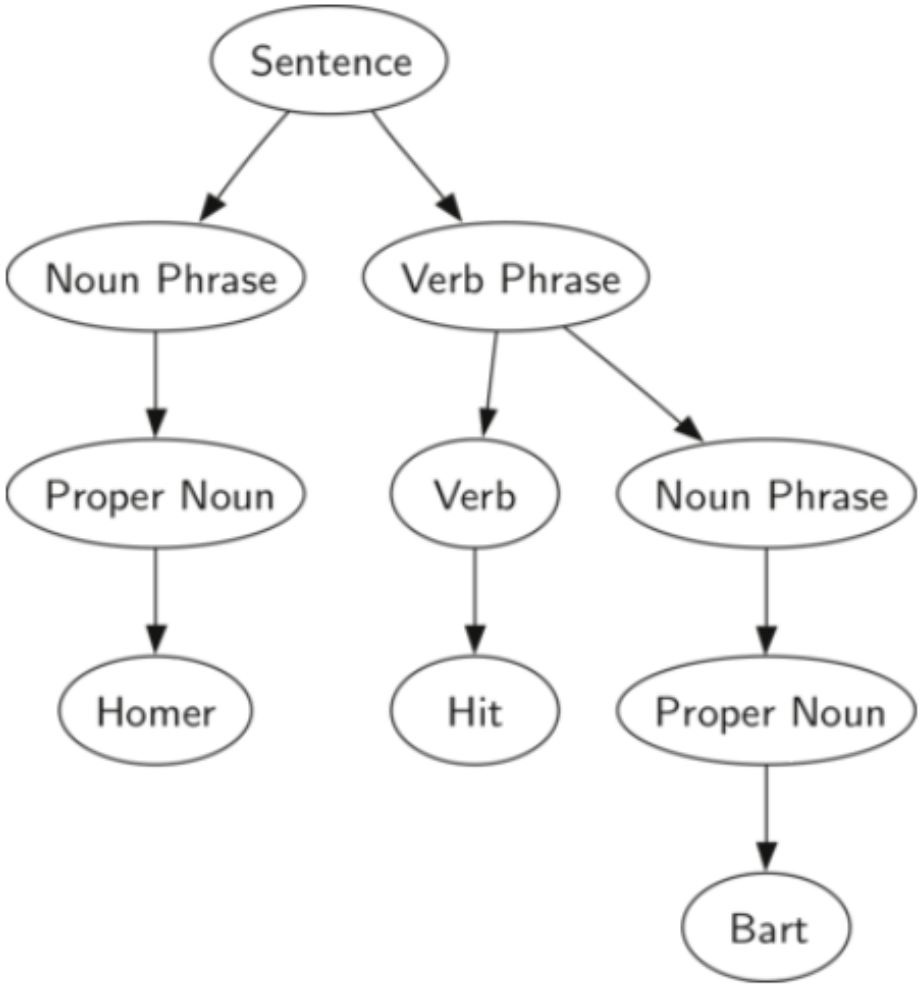
Příklady stromů



Unixová struktura adresářů

Images courtesy of Brad Miller, David Ranum.

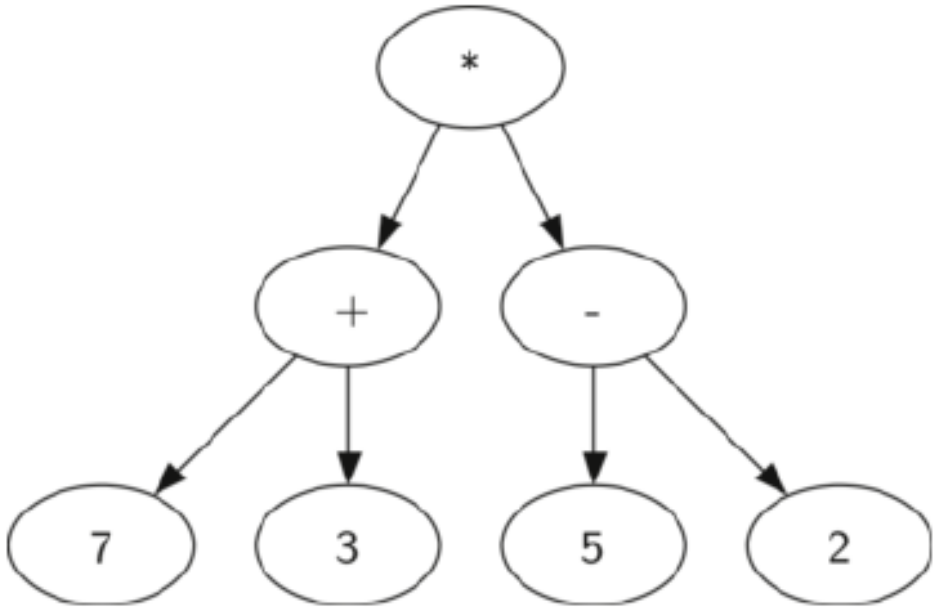
Příklady stromů



Gramatická struktura věty.

Images courtesy of Brad Miller, David Ranum.

Příklady stromů



Struktura aritmetického výrazu $(7 + 3) * (5 - 2)$

Images courtesy of Brad Miller, David Ranum.

Reprezentace stromu — záznam

```
class BinaryTree:
  def __init__(self, data,left=None,right=None):
    self.data = data
    self.left = left
    self.right = right
```

Reprezentace stromu — záznam

```
class BinaryTree:
  def __init__(self, data,left=None,right=None):
    self.data = data
    self.left = left
    self.right = right
```

Reprezentace výrazu $(7 + 3) * (5 - 2)$:

```
t=BinaryTree('*',
    BinaryTree('+',BinaryTree(7),BinaryTree(3)),
    BinaryTree('-',BinaryTree(5),BinaryTree(2)))
```

Reprezentace stromu — záznam

```
class BinaryTree:
  def __init__(self, data,left=None,right=None):
    self.data = data
    self.left = left
    self.right = right
```

Reprezentace výrazu $(7+3)*(5-2)$:

```
t=BinaryTree('*',
    BinaryTree('+',BinaryTree(7),BinaryTree(3)),
    BinaryTree('-',BinaryTree(5),BinaryTree(2)))
```

V této reprezentaci *strom*=*kořen*. Prázdný strom = `None`.

Reprezentace stromu — záznam

```
class BinaryTree:
  def __init__(self, data,left=None,right=None):
    self.data = data
    self.left = left
    self.right = right
```

Reprezentace výrazu $(7+3)*(5-2)$:

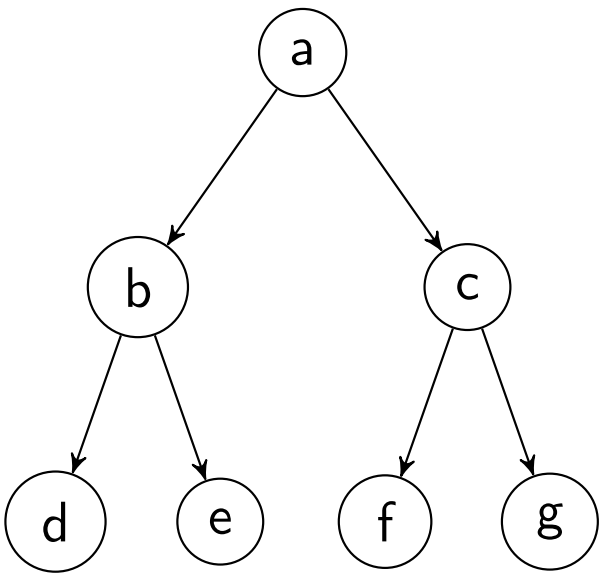
```
t=BinaryTree('*',
    BinaryTree('+',BinaryTree(7),BinaryTree(3)),
    BinaryTree('-',BinaryTree(5),BinaryTree(2)))
```

V této reprezentaci *strom=kořen*. Prázdný strom = `None`.

Implicitní *(default)* parametry mohou a nemusí být zadány.

Procházení stromu

- *preorder* — nejdřív aktuální uzel, pak oba podstromy *(prefixová notace)* `abdecfg`
- *inorder* — levý podstrom, pak aktuální uzel, pak pravý podstrom *(infixová notace)* `dbeafcg`
- *postorder* — nejdřív oba podstromy, pak aktuální uzel *(postfixová notace)* `debfgca`



Procházení stromu — implementace

```
def to_string_preorder(tree):
  return ( str(tree.data)+" "+
           to_string_preorder(tree.left) +
           to_string_preorder(tree.right)
           if tree else "" )
```

Procházení stromu — implementace

```
def to_string_preorder(tree):
  return ( str(tree.data)+" "+
           to_string_preorder(tree.left) +
           to_string_preorder(tree.right)
           if tree else "" )


print(to_string_preorder(t))
```

```
* + 7 3 - 5 2
```

Procházení stromu — implementace

```
def to_string_preorder(tree):
  return ( str(tree.data)+" "+
           to_string_preorder(tree.left) +
           to_string_preorder(tree.right)
           if tree else "" )


print(to_string_preorder(t))
```

```
* + 7 3 - 5 2
```

```
def to_string_postorder(tree):
  return ( to_string_postorder(tree.left) +
           to_string_postorder(tree.right) + " " +
           str(tree.data)
           if tree else "" )
```

Procházení stromu — implementace

```
def to_string_preorder(tree):
  return ( str(tree.data)+" "+
           to_string_preorder(tree.left) +
           to_string_preorder(tree.right)
           if tree else "" )


print(to_string_preorder(t))
```

```
* + 7 3 - 5 2
```

```
def to_string_postorder(tree):
  return ( to_string_postorder(tree.left) +
           to_string_postorder(tree.right) + " " +
           str(tree.data)
           if tree else "" )


print(to_string_postorder(t))
```

```
 7 3 + 5 2 - *
```

Procházení stromu — implementace (2)

```
def to_string_inorder(tree):
  if not tree:           # prázdný strom
    return ""
  if tree.left:   # binární operátor
    return ( "(" + to_string_inorder(tree.left)
             + str(tree.data)
             +  to_string_inorder(tree.right) + ")" )
  return str(tree.data) # jen jedno číslo
```

Procházení stromu — implementace (2)

```
def to_string_inorder(tree):
  if not tree:           # prázdný strom
    return ""
  if tree.left:  # binární operátor
    return ( "(" + to_string_inorder(tree.left)
             + str(tree.data)
             +  to_string_inorder(tree.right) + ")" )
  return str(tree.data) # jen jedno číslo


print(to_string_inorder(t))
```

```
((7+3)*(5-2))
```

Vyhodnocení výrazu

```
def evaluate(tree):
  """ Vyhodnotí aritmetický výraz zadaný stromem """
  if tree.data=='+':
    return evaluate(tree.left) + evaluate(tree.right)
  if tree.data=='-':
    return evaluate(tree.left) - evaluate(tree.right)
  if tree.data=='*':
    return evaluate(tree.left) * evaluate(tree.right)
  if tree.data=='/':
    return evaluate(tree.left) / evaluate(tree.right)
  return tree.data # jen jedno číslo


print(evaluate(t))
```

```
30
```

Soubor `binary tree`

Stromy

Binární vyhledávací stromy

Množiny a mapy

Binární vyhledávací stromy — motivace
(Binary search trees)

Aktualizovatelná datová struktura pro rychlé vyhledávání *porovnatelných* dat.

- *Setříděné pole* — vkládání $O(n)$, vyhledávání $O(\log n)$
- *Spojový seznam* — vkládání $O(1)$, vyhledávání $O(n)$
- *Vyhledávací strom* — vkládání $O(\log n)$, vyhledávání $O(\log n)$

Množina

Podporované operace

- `add(key)` — vložení prvku
- `delete(key)` — odstranění prvku
- `contains(key)` — obsahuje množina daný prvek?

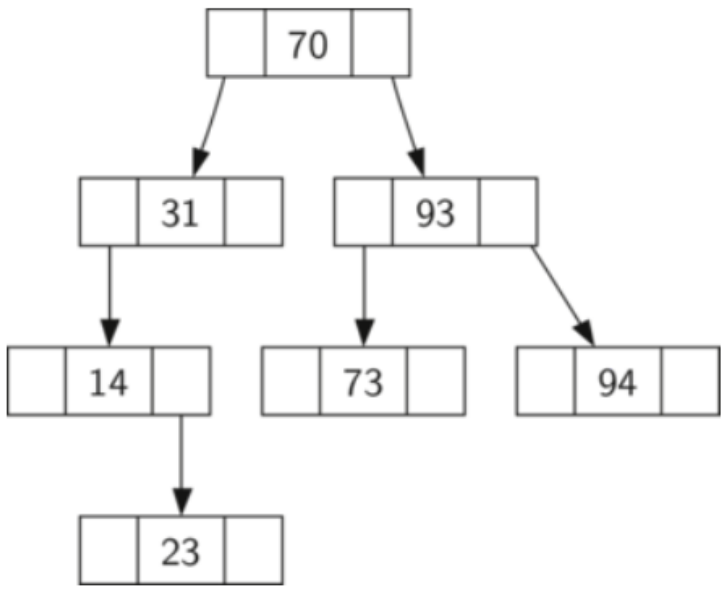
Pomocné funkce: `size` / `len`

Rychlé operace (složitost $O(\log n)$ nebo lepší)

Binární vyhledávací strom

Vlastnosti

- každý uzel obsahuje *klíč*
- klíč v uzlu není menší, než všechny klíče v jeho levém podstromu
- klíč v uzlu není větší, než všechny klíče v jeho pravém podstromu



Reprezentace vyhledávacího stromu

```
class BinarySearchTree:
  def __init__(self,key,left=None,right=None):
    self.key = key
    self.left = left
    self.right = right
```

Strom = uzel. Prázdný strom reprezentujeme jako `None`.

Soubor `binary_search_tree.py`

Vyhledávání ve stromu

```
def contains(tree,key):
    """ Je prvek 'key' ve stromu? """
    if tree:                # je strom neprázdný?
      if tree.key==key: # je to hledaný klíč?
        return True
      if tree.key>key:
        return contains(tree.left,key)
      else:
        return contains(tree.right,key)
    return False
```

Vytvoření ve stromu

Hledání ve stromu je ekvivalentní binárnímu vyhledávání. Sestrojíme strom ze setříděného pole.

```
def from_array(a):
  """ Build a tree (containing only keys) from an array """
  def build(a):
    if len(a)==0:
      return None
    if len(a)==1:
      return BinarySearchTree(a[0])
    m=len(a)//2
    return BinarySearchTree(a[m],left=build(a[:m]),
                                 right=build(a[m+1:]))
  a=sorted(a)
  return build(a)
```

Vytisknutí stromu

```
def print_tree(tree,level=0,prefix=""):
  if tree:
    print(" "*(4*level)+prefix+str(tree.key))
    if tree.left:
      print_tree(tree.left,level=level+1,prefix="L:")
    if tree.right:
      print_tree(tree.right,level=level+1,prefix="R:")
```

Vyhledávácí strom — příklad

```
import binary_search_tree as bst
t=bst.from_array([21, 16, 19, 87, 34, 92, 66])
bst.print_tree(t)
```

```
34
    L:19
        L:16
        R:21
    R:87
        L:66
        R:92
```

Vkládání do stromu

```
def add(tree,key):
  """ Vloží 'key' do stromu a vrátí nový kořen """
  if tree is None:
    return BinarySearchTree(key)
  if key<tree.key:
    tree.left=add(tree.left,key)
  elif  key>tree.key:
    tree.right=add(tree.right,key)
  return tree # hodnota již ve stromu je
```

Vkládání do stromu — příklad

```
print_tree(t)

34
    L:19
        L:16
        R:21
    R:87
        L:66
        R:92
```

```
t=add(t,41)
t=add(t,16)
print_tree(t)

34
    L:19
        L:16
        R:21
    R:87
        L:66
            L:41
        R:92
```

Příklad: strom jako množina

Vypiš všechny možné součty hodů na dvou kostkách.

```
s=None
for i in range(1000):
  s=add(s,random.randrange(1,7)+random.randrange(1,7))
print_tree(s)
```

```
10
    L:2
        R:8
            L:6
                L:4
                    L:3
                    R:5
                R:7
            R:9
    R:12
        L:11
```

Převod na pole

Projde uzly stromu podle velikosti a uloží do pole.

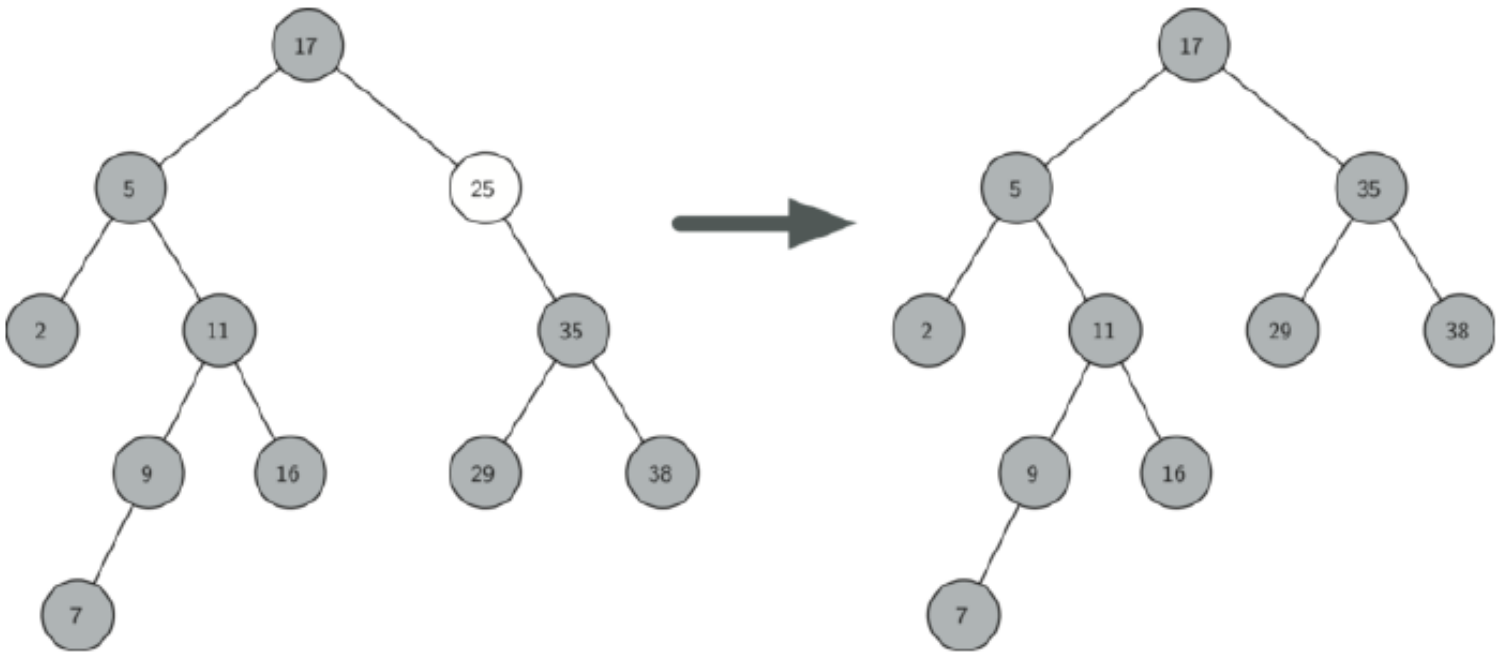
```
def to_array(tree):
  a=[]
  def insert_inorder(t):
    nonlocal a
    if t:
      insert_inorder(t.left)
      a+=[t.key]
      insert_inorder(t.right)
  insert_inorder(tree)
  return a

print(to_array(s))
```

```
[2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]
```

`nonlocal` — přístup k proměnné vnější funkce *(jen Python 3)*

Soubor `binary_search_tree.py`

Odstranění prvku ze stromu



Odstranění prvku — implementace

```
def delete(tree, key):
  """ Smaže 'key' za stromu 'tree' a vrátí nový kořen. """
  if tree is not None:
    if key < tree.key: # najdi uzel 'key'
      tree.left = delete(tree.left, key)
    elif key > tree.key:
      tree.right = delete(tree.right, key)
    else: # uzel nalezen, má syny?
      if tree.left is None:
        return tree.right # jen pravý syn nebo nic
      elif tree.right is None:
        return tree.left # jen levý syn nebo nic
      else: # nahradíme uzel maximem levého podstromu
        w = rightmost_node(tree.left)
        tree.key = w.key
        tree.left = delete(tree.left, w.key)
  return tree
```

Odstranění prvku (2)

```
def rightmost_node(tree):
  while tree.right:
    tree=tree.right
  return tree
```

Odstranění prvku — příklad

```
t=from_array([21, 16, 19, 87, 34, 92, 66])

print_tree(t)
```

```
34
    L:19
        L:16
        R:21
    R:87
        L:66
        R:92
```

```
t=delete(t,87)
print_tree(t)
```

```
34
    L:19
        L:16
        R:21
    R:66
        R:92
```

Množinový rozdíl
(set difference)

Find elements in array `y` but not in array `x`.

```
def set_difference(x,y):
  t=None
  for i in y:
    t=add(t,i)
  for j in x:
    t=delete(t,j)
  return to_array(t)
```

Množinový rozdíl — příklad

$x = [\ 186, 344, 36, 640, 617, 484, 267, 949, 890, 522, 567, 114, 841, 580, 262, 159, 979, 156, 830, 822, 598, 48, 400, 879, 418, 837, 44, 495, 304, 499, 12, 938, 276, 923, 510, 89, 764, 803, 193, 861, 970, 197, 583, 290, 215, 931, 781, 57, 527, 511, 230, 255, 610, 275, 658, 843, 366, 541, 149, 397, 272, 149, 759, 619, 822, 27, 573, 38, 475, 340, 294, 11, 197, 17, 904, 578, 838, 817, 591, 122, 516, 638, 160, 401, 232, 183, 136, 517, 108, 442, 173, 693, 240, 338, 268, 231, 177, 521, 959, 56\ ]$

$y = [\ 400, 215, 197, 837, 693, 516, 156, 890, 268, 272, 108, 56, 11, 583, 658, 517, 159, 186, 27, 160, 817, 57, 499, 366, 541, 638, 573, 938, 149, 822, 275, 578, 442, 173, 511, 12, 830, 617, 495, 923, 183, 177, 904, 484, 344, 931, 230, 304, 338, 240, 418, 262, 48, 580, 149, 122, 781, 38, 567, 764, 959, 276, 193, 522, 81, 44, 970, 340, 294, 979, 475, 759, 290, 401, 822, 843, 838, 231, 510, 255, 949, 36, 267, 114, 397, 610, 527, 879, 841, 619, 591, 89, 197, 232, 598, 861, 521, 17, 136, 640, 803\ ]$

Množinový rozdíl — příklad

$x =$ [576, 178, 273, 457, 724, 449, 739, 671, 0, 215, 972, 430, 153, 186, 249, 958, 930, 139, 926, 440, 57, 899, 928, 739, 84, 161, 166, 247, 751, 257, 740, 95, 852, 930, 750, 132, 189, 25, 394, 752, 398, 310, 729, 712, 293, 56, 500, 468, 424, 151, 926, 439, 61, 673, 891, 616, 9, 937, 31, 854, 920, 196, 924, 951, 870, 762, 932, 887, 692, 515, 964, 381, 27, 784, 770, 664, 394, 687, 496, 338, 239, 163, 832, 342, 964, 539, 825, 991, 307, 95, 175, 863, 188, 572, 761, 310, 906, 309, 843, 584]

$y =$ [27, 825, 951, 751, 257, 342, 457, 153, 572, 56, 539, 500, 95, 762, 61, 151, 249, 9, 309, 0, 924, 887, 293, 273, 163, 664, 852, 899, 424, 671, 186, 449, 739, 430, 964, 930, 576, 85, 178, 991, 338, 958, 496, 870, 394, 239, 863, 932, 215, 307, 398, 189, 920, 750, 712, 740, 381, 692, 166, 752, 161, 25, 616, 926, 31, 964, 928, 440, 891, 139, 926, 310, 687, 132, 468, 310, 729, 739, 832, 584, 906, 57, 95, 770, 854, 515, 439, 937, 972, 247, 784, 394, 843, 175, 761, 930, 188, 196, 84, 724, 673]

```
print(set_difference(x,y))
```

```
[85]
```

Složitost $O(n \log n)$ místo $O(n^2)$.

Složitost

- Vkládání, vyhledávání i mazání = 1-2 průchody stromem = $O(h)$
- **Dokonale vyvážený strom** = rozdíl počtu uzlů podstromů stejného rodiče se liší nejvýše o 1.

Složitost

- Vkládání, vyhledávání i mazání = 1-2 průchody stromem = $O(h)$
- **Dokonale vyvážený strom** = rozdíl počtu uzlů podstromů stejného rodiče se liší nejvýše o 1.
- $\rightarrow$ **hloubka** $h = O(\log n)$
- $\rightarrow$ složitost vkládání, vyhledávání i mazání $O(\log n)$

Složitost

- Vkládání, vyhledávání i mazání = 1-2 průchody stromem = $O(h)$
- **Dokonale vyvážený strom** = rozdíl počtu uzlů podstromů stejného rodiče se liší nejvýše o 1.
- $\rightarrow$ **hloubka** $h = O(\log n)$
- $\rightarrow$ složitost vkládání, vyhledávání i mazání $O(\log n)$

Nejhorší případ

- Degenerovaný strom s hloubkou $n - 1$
- Složitost vkládání, vyhledávání i mazání $O(n)$

Vyvažování stromu

- Dokonalé vyvážení je obtížné, stačí hloubka $h = O(\log n)$
- Náhodná data
- Omezení na tvar stromu
  - AVL stromy (Adelson-Velsky a Landis)
  - red-black trees (červeno-černé stromy)
  - 2-3 stromy...
- Při přidávání/odebírání elementů strom *vyvažujeme*
- Vyvažování má složitost $O(\log n)$

AVL stromy
(Adelson-Velsky a Landis)

- Rozdíl hloubek podstromů stejného rodiče se liší nejvýše o 1.
- Pro AVL strom platí

$$h < c \log_2(n+2) + b$$

kde $c = \log_2^{-1} \phi \approx 1.44$, $b \approx -1.328$ a $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ (zlatý řez)

- Pro každý binární strom

$$h \geq \log_2(n+1) - 1$$

AVL stromy
(Adelson-Velsky a Landis)

- Rozdíl hloubek podstromů stejného rodiče se liší nejvýše o 1.
- Pro AVL strom platí

$$h < c \log_2(n+2) + b$$

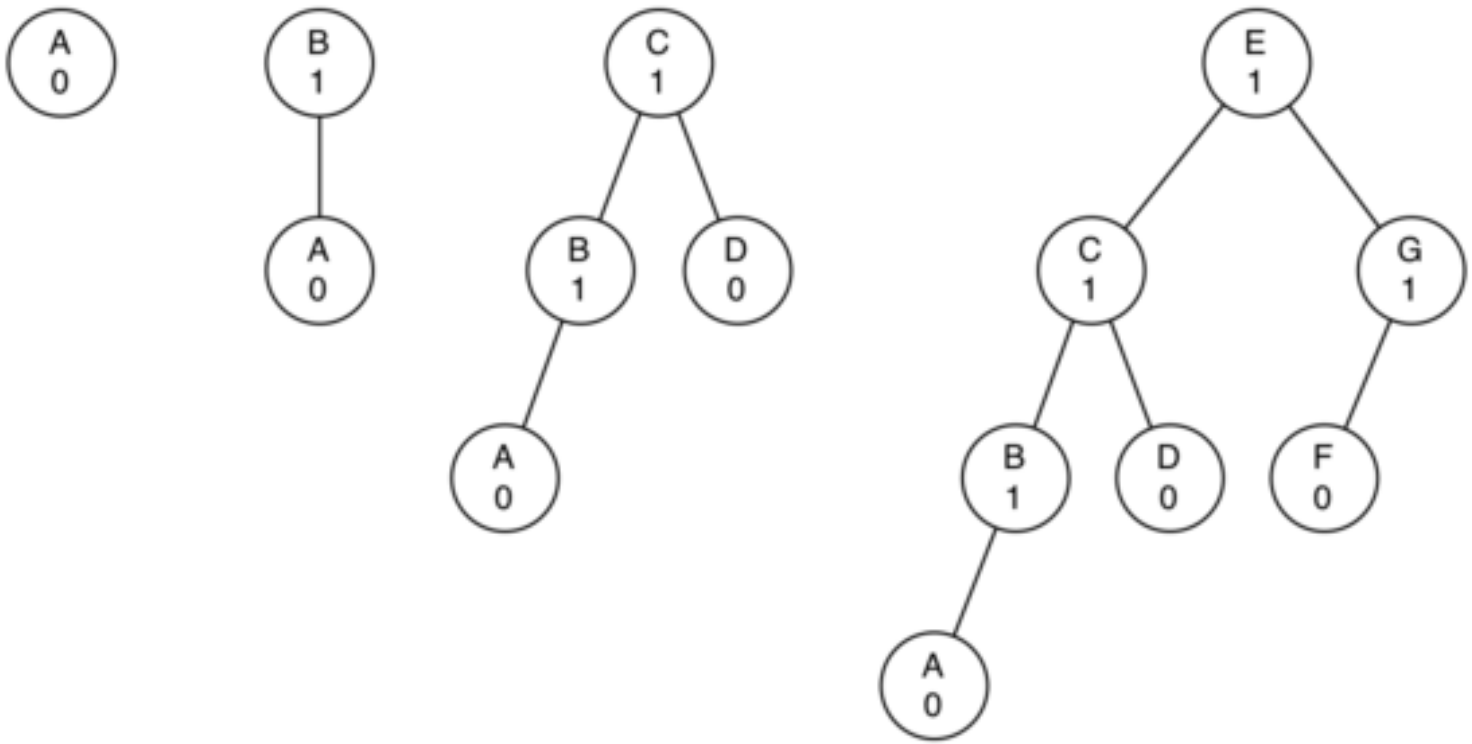
kde $c = \log_2^{-1} \phi \approx 1.44$, $b \approx -1.328$ a $\phi = \frac{1}{2}(\sqrt{5}+1) \approx 1.618$ (zlatý řez)

- Pro každý binární strom

$$h \geq \log_2(n+1) - 1$$

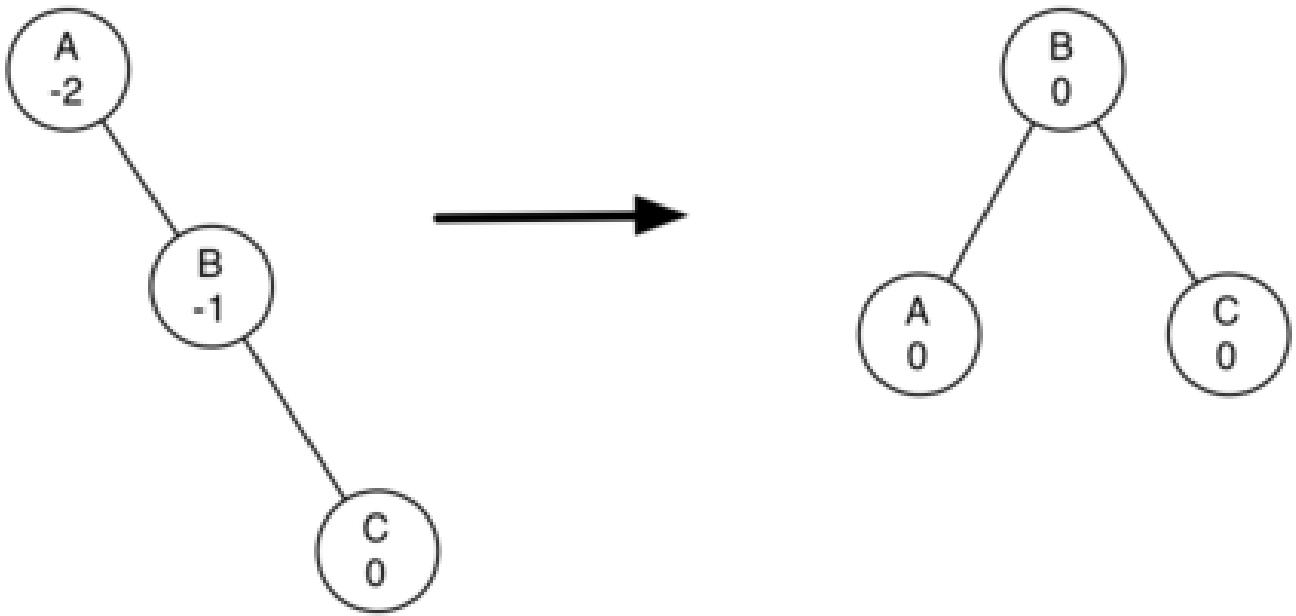
- V každém uzlu si pamatujeme $h(l) - h(r) \in \{-1, 0, 1\}$

Maximálně nevyvážené AVL stromy



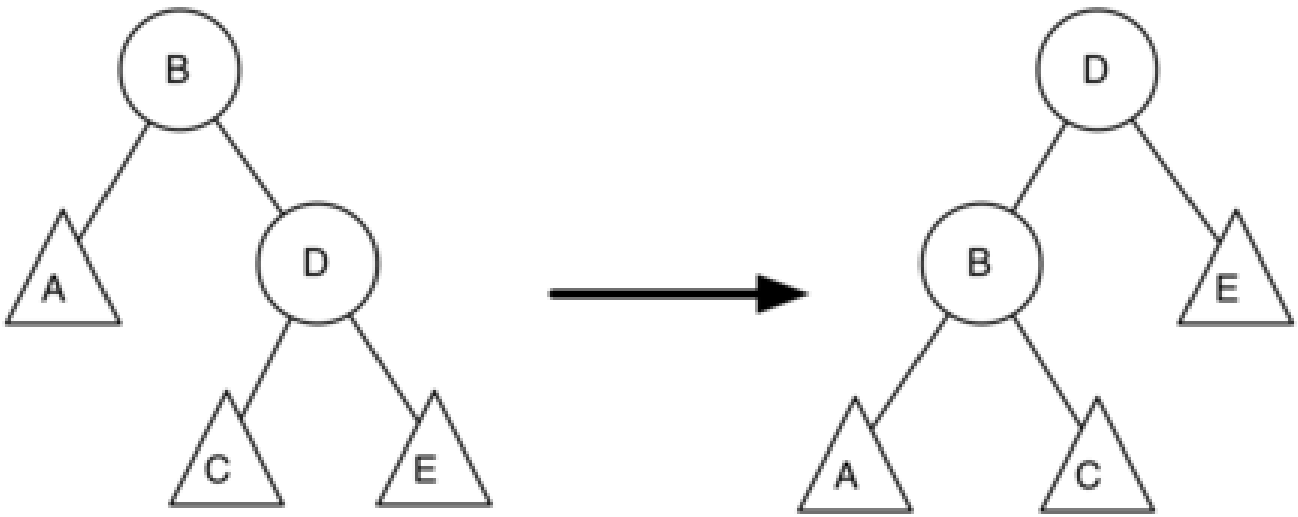
Rotace

Nevyváženost odstraníme **rotací**



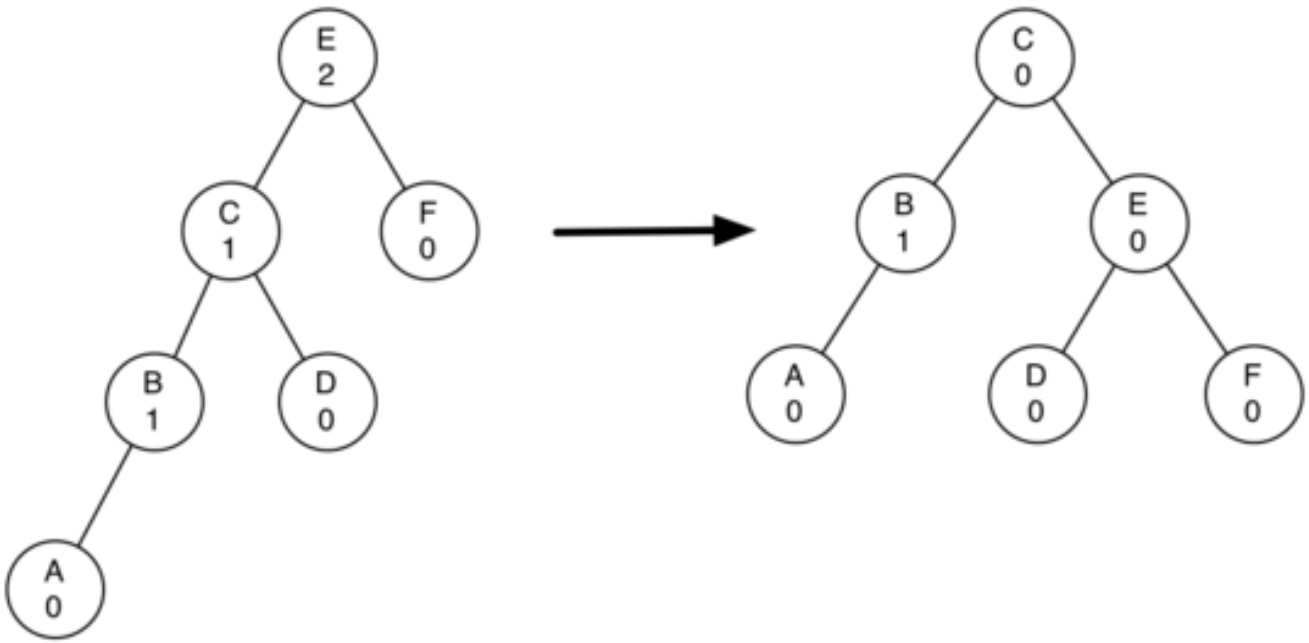
Rotace

Nevyváženost odstraníme **rotací**



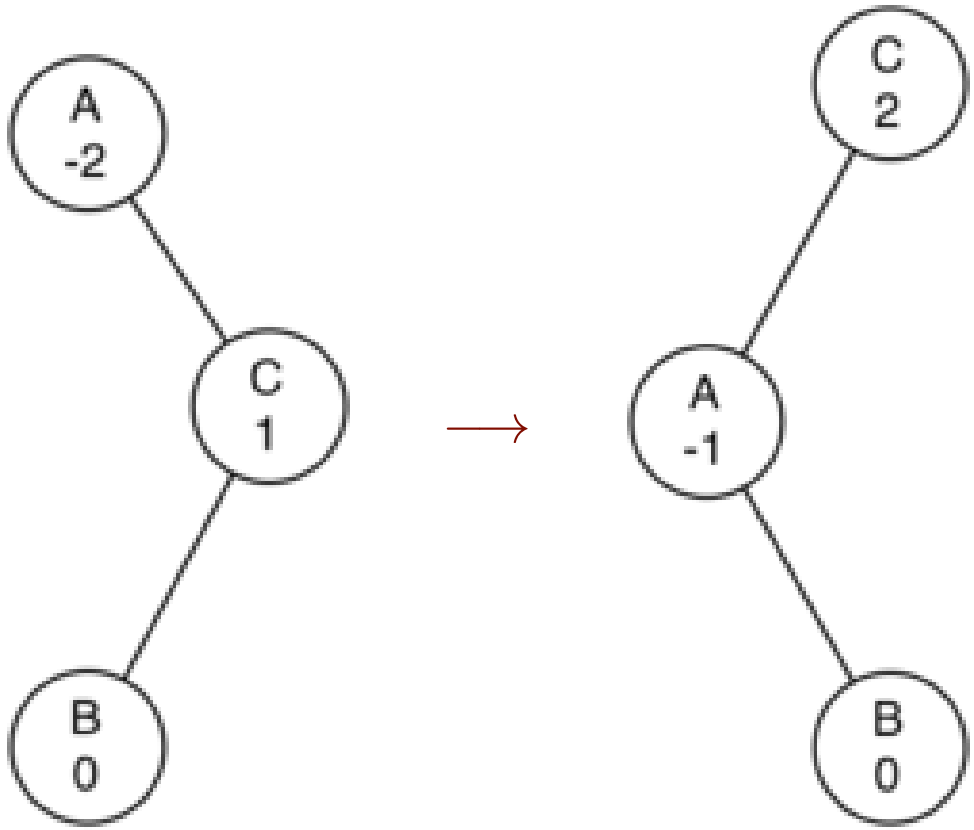
Rotace

Nevyváženost odstraníme **rotací**



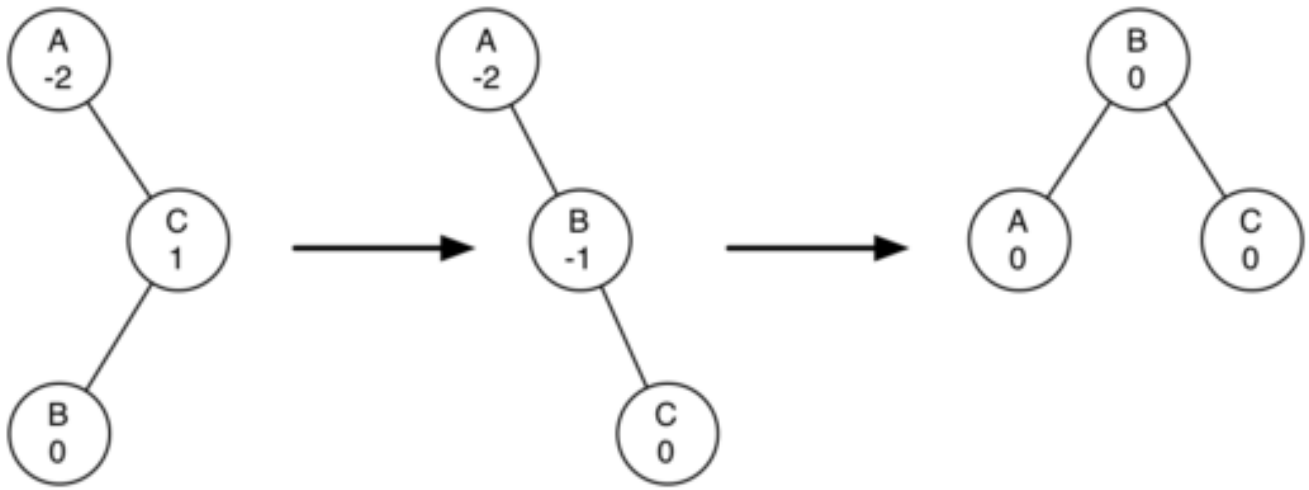
Rotace

Problém



Rotace

Dvojitá rotace



Implementace např. *Problem Solving with Algorithms and Data Structures*
https://interactivepython.org/runestone/static/pythonds/Trees/AVLTreeImplementation.html

Stromy

Binární vyhledávací stromy

Množiny a mapy

Množina

Podporované operace

- `add(key)` — vložení prvku
- `delete(key)` — odstranění prvku
- `contains(key)` — obsahuje množina daný prvek?

Pomocné funkce: `size` / `len`

Rychlé operace (složitost $O(\log n)$ nebo lepší)

Asociativní mapa
Associative map

Funkce `klíč` → `hodnota` (`key` → `value`)

Podporované operace

- `put(key,value)` — vložení položky
- `delete(key)` — odstranění prvku
- `contains(key)` — obsahuje mapa daný prvek?
- `get(key)` → `value` — nalezení/vyzvednutí hodnoty

Pomocné funkce: `size` / `len`

Rychlé operace (složitost $O(\log n)$ nebo lepší)

Asociativní mapa
Associative map

Funkce `klíč` → `hodnota` (`key` → `value`)

Podporované operace

- `put(key,value)` — vložení položky
- `delete(key)` — odstranění prvku
- `contains(key)` — obsahuje mapa daný prvek?
- `get(key)` → `value` — nalezení/vyzvednutí hodnoty

Pomocné funkce: `size` / `len`

Rychlé operace (složitost $O(\log n)$ nebo lepší)

Množina je speciální případ mapy.

Reprezentace

```
class BinarySearchTree:
  def __init__(self, key,value=None,left=None,right=None):
    self.key = key
    self.value = value
    self.left = left
    self.right = right
```

Soubor `binary_search_tree.py`

Vyhledávání v mapě

```
def get(tree,key):
    """ Vrátí 'value' prvku s klíčem 'key', jinak None """
    if tree:                  # je strom neprázdný?
      if tree.key==key: # je to hledaný klíč?
        return tree.value
      if tree.key>key:
        return get(tree.left,key)
      else:
        return get(tree.right,key)
    return None
```

Vkládání do mapy

```
def put(tree,key,value):
  """ Vloží pár 'key'->'value', vrátí nový kořen """
  if tree is None:
    return BinarySearchTree(key,value=value)
  if key<tree.key:
    tree.left=put(tree.left,key,value)
  elif  key>tree.key:
    tree.right=put(tree.right,key,value)
  else:
    tree.value=value # klíč již ve stromu je
  return tree
```

Mapa — příklad
tabulka symbolů

```
t=None
t=put(t,'pi',    3.14159)
t=put(t,'e',     2.71828)
t=put(t,'sqrt2', 1.41421)
t=put(t,'golden',1.61803)
print_tree(t)
```

```
pi -> 3.14159
    L:e -> 2.71828
        R:golden -> 1.61803
    R:sqrt2 -> 1.41421
```

```
print(get(t,'pi'))
```

```
3.14159
```

```
print(get(t,'e'))
```

```
2.71828
```

```
print(get(t,'gamma'))
```

```
None
```

Implementace funguje i pro řetězcové klíče.

Vyhledávací stromy

- Datová struktura pro porovnatelné klíče
- Může reprezentovat množinu i mapu.
- Základní operace (vkládání, hledání, mazání) mají složitost $O(\log n)$.
- Vyšší režie (oproti např. poli)
- Stromů je mnoho typů
  - B-stromy
  - k-d stromy, R-stromy
  - prefixové stromy
  - *ropes*...

Náměty na domácí práci

- Reprezentujte strom pomocí dvou tříd, aby nebylo potřeba vracet nový kořen.
- Doplňte detekci chyb.
- Zefektivněte implementaci, aby nedocházelo k neustálému přepisování odkazů.
- Zrychlete operaci `delete` pro případ mazání uzlu se dvěma syny.
- Implementujte Eratosthenovo síto pomocí množin.
- Implementujte AVL strom.
- Ověřte experimentálně časovou složitost operací se stromem.
- Přepište algoritmy bez použití rekurze.
- Najděte množinový rozdíl dvou polí pomocí třídění.