Vyhledávací stromy

Binární vyhledávání popsané v předchozí části má velmi příznivou časovou složitost. Pokud bychom měli množinu prvků, v níž velmi často a intenzívně hledáme, pak by se zřejmě vyplatilo je na začátku setřídit. Problém ovšem nastane, když tato množina se v průběhu času mění, tj. jsou k ní přidávány nové prvky nebo z ní naopak některé prvky jsou odebírány. Už jsme uvedli, že vkládání prvků doprostřed pole nebo jejich odebírání zprostředka pole je poměrně neefektivní operace, neboť je spojena s přesuny poměrné značné části prvků v poli. Pro takovéto případy je výhodnější použít vyhledávací stromy.

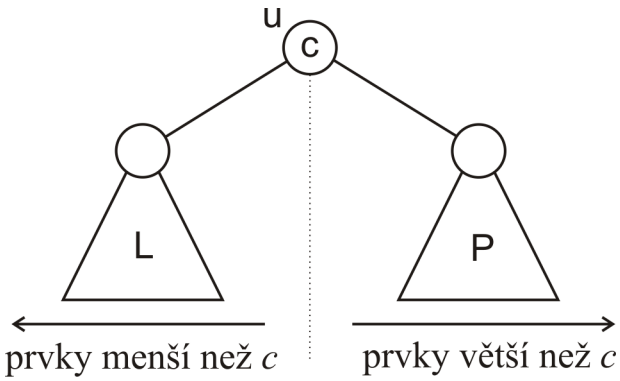
Vyhledávací stromy jsou velmi významnou a širokou skupinou vyhledávacích metod. Lze je rozdělit na

- Binární vyhledávací stromy
- Vícecestné vyhledávací stromy

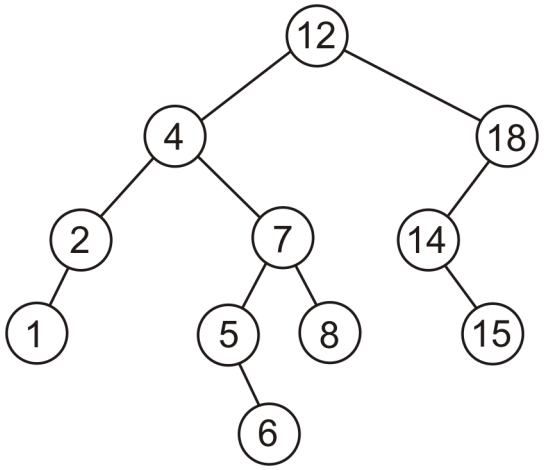
Binární vyhledávací stromy

Jsou binární stromy s vlastnostmi:

- V každém uzlu stromu je uložen jeden datový prvek.
- Pro každý uzel *u* a prvek v něm uložený *c* platí, že prvky uložené v levém podstromu uzlu *u* (má-li uzel *u* levý podstrom) jsou menší než prvek *c* a prvky uložené v pravém podstromu uzlu *u* (má-li uzel *u* pravý podstrom) jsou větší než prvek *c*.



Příklad.



Vyhledání prvku

1. Počáteční krok

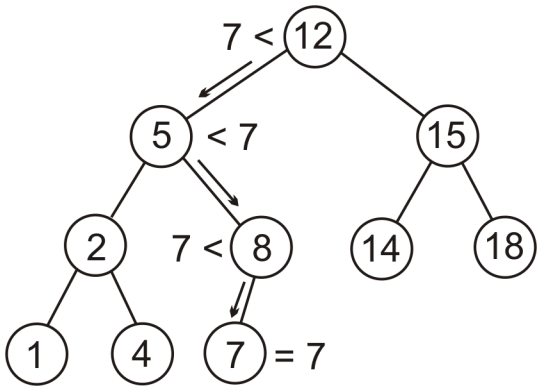
   Uzel, který je v daném okamžiku vyhledávání aktuální, budeme označovat *u*. Na začátku jím bude kořen stromu.
   Hledaná hodnota nechť je *x*.

2. Průběžný krok

   Vezmeme prvek obsažený v aktuálním uzlu *u*, označme ho *c*, a provedeme jeho srovnání s hledanou hodnotou *x*:

   - Nejprve srovnáme, zda je $x < c$:
     Pokud ano, pak je nutné v hledání pokračovat v levém podstromu. Jako nový aktuální uzel *u* položíme levého následníka současného aktuálního uzlu a znovu provedeme krok 2.
     Pokud současný aktuální uzel levého následníka nemá, vyhledávání končí - hledaný prvek není ve stromu obsažen.
   - Pokud není $x < c$, srovnáme, zda je $x > c$:
     Pokud ano, je nutné v hledání pokračovat v pravém podstromu. Jako nový aktuální uzel *u* položíme pravého následníka současného aktuálního uzlu a opět provedeme krok 2.
     Pokud uzel pravého následníka nemá, vyhledávání končí, hledaný prvek není ve stromu obsažen.
   - Pokud není $x > c$, zbývá už jen případ, že platí $x = c$, čímž jsme u konce hledání, neboť prvek *c* obsažený v současném aktuálním uzlu *u* je tím hledaným prvkem.

Příklad. V následujícím stromu máme vyhledat číslo 7.



Časová složitost: $\Theta(h)$, kde *h* je výška vyhledávacího stromu.

Pseudokód:

```
Search(T, x)
  u ← T.root
  while u ≠ NIL
    if x < u.item
      u ← u.left
    else
      if x > u.item
        u ← u.right
      else
        return u
  return NIL
```

Zdrojový kód:

```
Node *Search(Node *root, Item x)
{
  Node *u=root;
  while (u != NULL)
  { if (x < u->item) u=u->left;
    else if (x > u->item) u=u->right;
    else return u;
  }
  return NULL;
}
```

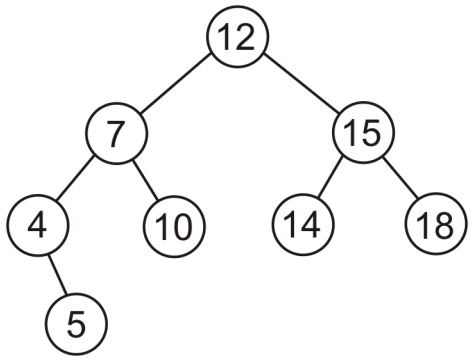
Přidání prvku

Operace přidání prvku do binárního vyhledávacího stromu znamená na příslušném místě přidat do stromu uzel, do kterého nový prvek vložíme.

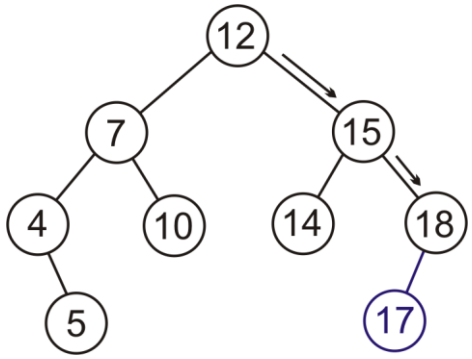
Označme přidávaný prvek x. Provedeme jeho vyhledání ve stromu. Použijeme k tomu již popsaný algoritmus vyhledávání. Ten může skončit třemi způsoby:

- Prvek x byl ve stromu nalezen. Tím přidávání končí, neboť prvek x už je ve stromu obsažen a u vyhledávacích stromů se nepředpokládá vícenásobný výskyt stejného prvku.
- Vyhledávání skončilo v uzlu u s prvkem c, přičemž $x < c$ a přitom uzel u už nemá levého následníka. V tom případě přidáme ke stromu nový uzel jako levého následníka uzlu u a do něho nový prvek x vložíme.
- Vyhledávání skončilo v uzlu u s prvkem c, přičemž $x > c$ a přitom uzel u už nemá pravého následníka. V tom případě přidáme ke stromu pravého následníka uzlu u, do kterého nový prvek x vložíme.

Příklad. Do binárního vyhledávacího stromu



máme přidat prvek 17. Následující obrázek ukazuje postup.



Časová složitost: $\Theta(h)$, kde h je výška vyhledávacího stromu.

Pseudokód:

```
NewNode(x)
  u ← new Node
  u.item ← x
  u.left ← u.right ← NIL
```

```
    return u

  Insert(T, x)
    if T.root = NIL
      T.root ← NewNode(x)
      return true
    u ← T.root
    while true
      if x < u.item
        if u.left = NIL
          u.left ← NewNode(x)
          return true
        u ← u.left
      else
        if x > u.item
          if u.right = NIL
            u.right ← NewNode(x)
            return true
          u ← u.right
        else
          return false
```

Zdrojový kód:

```
bool Insert(Node **root, Item x)
{
  Node **u=root;
  while (*u != NULL)
  { if (x < (*u)->item) u=&(*u)->left;
    else if (x > (*u)->item) u=&(*u)->right;
    else return false;
  }
```

```
    { Node *v= new Node;
      v->item = x;
      v->left = v->right = NULL;
      *u = v;
    }
    return true;
}
```

Odebrání prvku

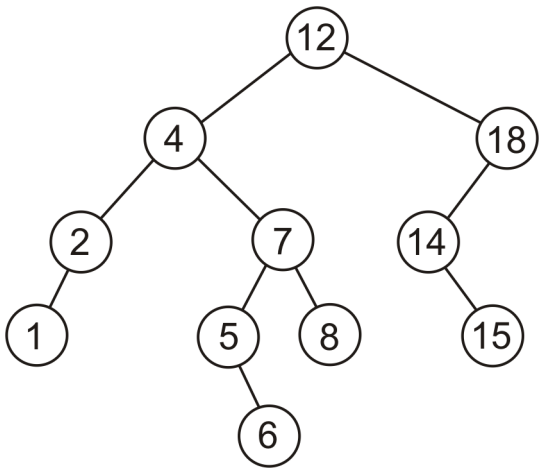
Operace odebrání prvku z binárního stromu znamená na příslušném místě zrušení uzlu ve stromu.

Označme odebíraný prvek *x*.

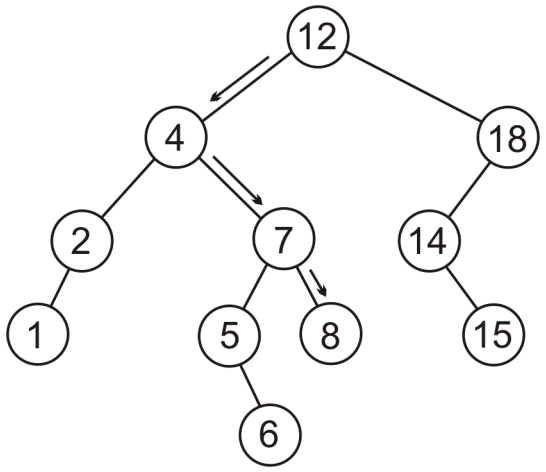
Vyhledáme prvek *x* ve stromu. Vyhledání může skončit třemi způsoby:

- Prvek *x* nebyl ve stromu nalezen – není co odebrat.
- Prvek byl nalezen v uzlu *v*, který má nejvýše jednoho následníka.Tento uzel zrušíme.
- Prvek byl nalezen v uzlu *v*, který má dva následníky. V tomto případě do uzlu *v* přesuneme buďto nejpravější (největší) prvek z jeho levého podstromu anebo nejlevější (nejmenší) prvek z jeho pravého podstromu a uzel, z kterého byl prvek přesunut, zrušíme.

Příklad. Ze stromu

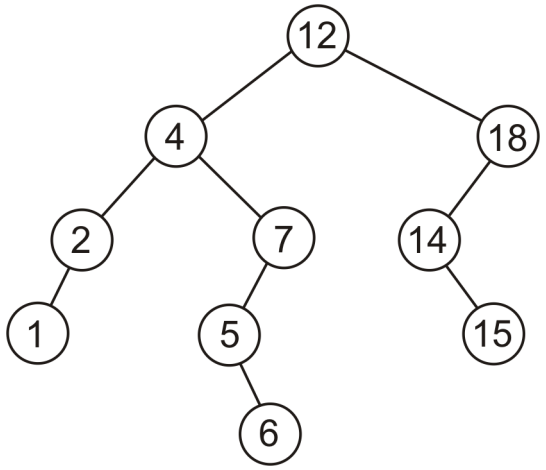


máme odebrat prvek 8.
Nejprve prvek 8 ve stromu vyhledáme

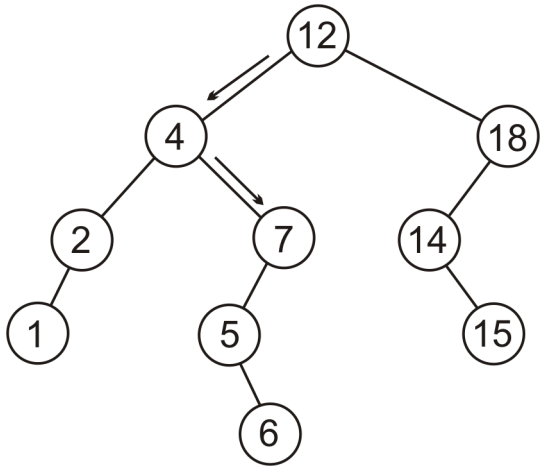


.

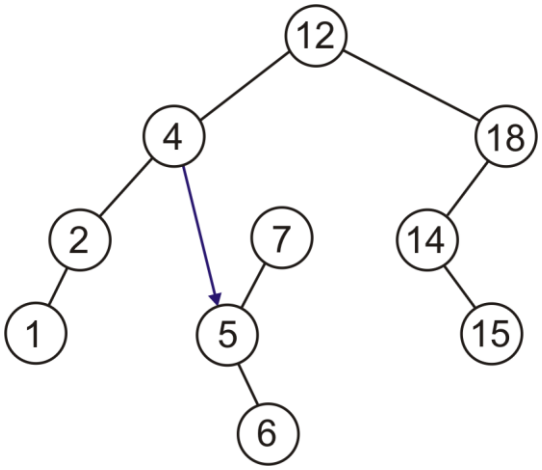
Protože prvek 8 je v uzlu, který nemá více než jednoho následníka, uzel s tímto prvkem můžeme zrušit.



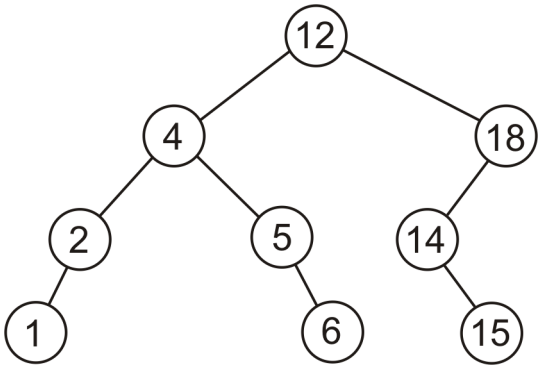
Dále máme odebrat prvek 7. Nejprve ho ve stromu vyhledáme.



Protože prvek 7 je v uzlu, který nemá více než jednoho následníka, uzel s tímto prvkem můžeme zrušit. Nejprve ale přesměrujeme odkaz v předchůdci rušeného uzlu na následníka rušeného uzlu.

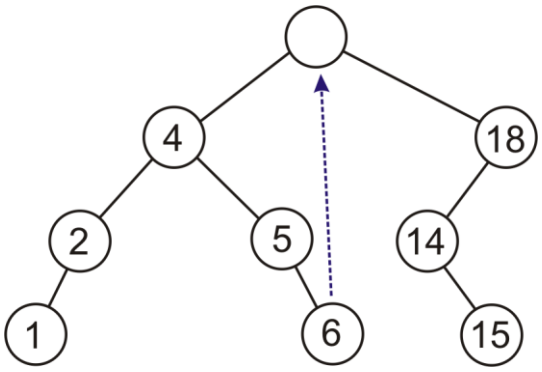


A uzel s prvkem 7 zrušíme.

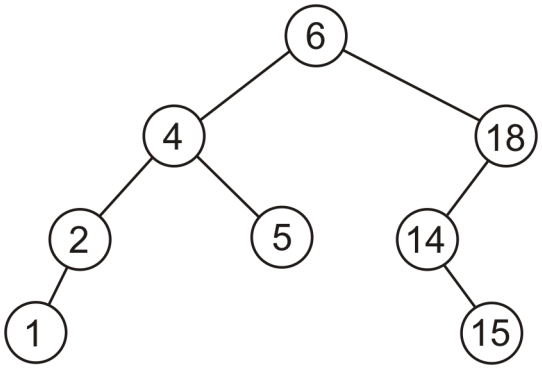


Nyní máme odebrat prvek 12. Jeho vyhledání je zde rychlé, neboť prvek je v kořenu. Uzel s tímto prvkem má dva následníky, nemůžeme ho přímo odebrat.

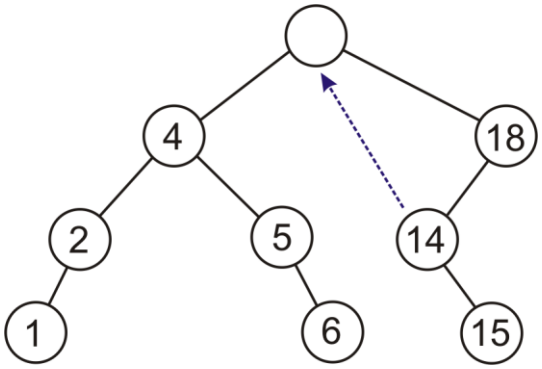
První možnost je nahradit odebíraný prvek 12 nejpravějším prvkem z levého podstromu, což je prvek 6.



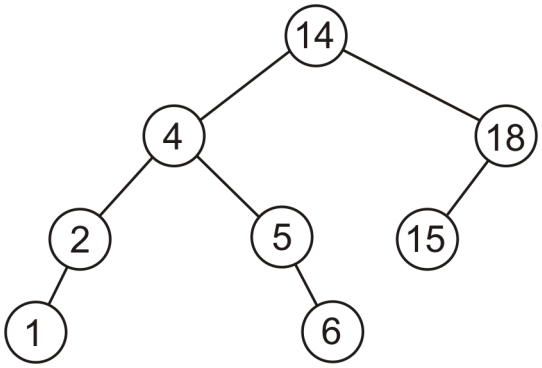
A uzel, ve kterém byl prvek 6, zrušit.



Druhá možnost je nahradit odebíraný prvek 12 nejlevějším prvkem z pravého podstromu, což je prvek 14.



A uzel, ve kterém byl prvek 14, zrušit.



Časová složitost: $\Theta(h)$, kde h je výška vyhledávacího stromu.

Pseudokód:

```
Delete(T, x)
  u ← T.root
  if u = NIL
    return false
  if u.item = x
    T.root ← DeleteNode(u)
    return true
  while true
    if x < u.item
      if u.left = NIL
        return false
      if u.left.item = x
        u.left ← DeleteNode(u.left)
        return true
      u ← u.left
```

```
    else
      if u.right = NIL
        return false
      if u.right.item = x
        u.right ← DeleteNode(u.right)
        return true
      u ← u.right
DeleteNode(u)
  if u.left = NIL
    return u.right
  if u.right = NIL
    return u.left
  v ← u.right
  if v.left = NIL
    u.item ← v.item
    u.right ← v.right
    return u
  w ← v.left
  while w.left ≠ NIL
    v ← w
    w ← w.left
  u.item ← w.item
  v.left ← w.right
  return u
```

Zdrojový kód:

```
bool Delete(Node **root, Item x)
{
  Node **u=root;
  while (*u != NULL)
  { if (x < (*u)->item) { u=&(*u)->left; continue; }
    if (x == (*u)->item) break;
    u=&(*u)->right;
  }
  if (*u==NULL) return false;
```

```
  if ((*u)->left!=NULL && (*u)->right!=NULL)
  { Node **v=&(*u)->right;
    while ((*v)->left!=NULL) v=&(*v)->left;
    (*u)->item=(*v)->item;
    u=v;
  }
  { Node *v=*u;
    *u= v->left ? v->left : v->right;
    delete v;
  }
}
```