1. Vyhledávací stromy

V této kapitole budeme postaveni před následující problém. Je dáno nějaké univerzum prvků U a naším úkolem bude navrhnout datovou strukturu, která udržuje konečnou množinu prvků $X \subseteq U$. Pod univerzem si můžeme představit například přirozená čísla, tedy univerzum může být nekonečné na rozdíl od X.

Na našich datech budeme chtít provádět následující operace:

- *Insert* – vložit novou položku
- *Delete* – smazat položku
- *Find* – najít položku

Způsob realizace jednotlivých operací a jejich časová složitost se bude zjevně odvíjet od vlastností prvků univerza. V našem případě budeme chtít udržovat *slovník*, tedy množinu dvojic (k, v) kde $k \in U$ se nazývá klíč a v hodnota. Dále předpokládejme, že univerzum U je lineárně uspořádané (důležitá vlastnost!) a s klíči i hodnotami pracujeme v konstantním čase.

Po slovníku budeme chtít:

- *Insert(k, v)* – vložit novou hodnotu spolu s klíčem (předpokládáme, že ve struktuře dosud tento klíč není přítomen)
- *Delete(k)* – smazat položku podle klíče
- *Find(k)* – najít položku podle klíče

Tyto operace je možné realizovat nad množstvím různých datových struktur, od jednoduchých po značně složité. Věříme, že čtenáři nedělá větší problém vymyslet, jak by požadované operace realizoval například nad setříděných či nesetříděným polem, setříděným či nesetříděným spojovým seznamem. Hlavním cílem této kapitoly je popsat datovou strukturu vyhledávacího stromu, nicméně než se pustíme do bližšího vysvětlování, uvedeme ještě srovnání časových složitostí operací nad vyhledávacími stromy a nad jednoduchými datovými strukturami.

	Insert	*Delete*	*Find*
pole	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
setříděné pole	$\Theta(n)$	$\Theta(n)$	$\Theta(\log n)$
spojový seznam	$\Theta(1)$	$\Theta(n)$	$\Theta(n)$
setříděný seznam	$\Theta(n)$	$\Theta(n)$	$\Theta(n)$
vyhledávací stromy	$\Theta(\log n)$	$\Theta(\log n)$	$\Theta(\log n)$

1.1. Definice binárního vyhledávacího stromu

Zavzpomínáme nyní, jak funguje algoritmus vyhledávání prvku půlením intervalů nad setříděným polem. Zvolíme prvek uprostřed pole, porovnáme s hledaným a podle výsledku porovnání se zaměříme buďto na levý nebo na pravý interval, kde opakujeme stejný algoritmus.

Stejnou myšlenku je možné zopakovat s jiným uložením dat než je pole. Představme si, že máme binární strom s prvky uloženými ve vrcholech, jehož jednotlivé podstromy ztotožníme s intervaly, nad kterými pracuje půlící vyhledávání, a vrcholy stromu ztotožníme se zvolenými prvky pole, se kterými půlící algoritmus porovnává hledaný prvek. Jeden průběh půlícího vyhledávání potom odpovídá průchodu tímto stromem z kořene dolů do nějakého listu.

Dále si povšimněme, že pro správnou funkci vyhledávacího algoritmu nebylo potřeba volit vždy prostředky intervalů, stačí volit libovolný vnitřní prvek intervalu. Tato volba se projeví pouze na čase, v jakém bude vyhledávání probíhat. Ani příslušný ztotožněný strom tedy nemusí být stejné pravidelný jako

Tyto postřehy nyní přetavíme do přesné definice binárních vyhledávacích stromů.

Definice: Strom T nazveme *binární*, pokud je zakořeněný a každý vrchol $v \in V(T)$ má nejvýše dva syny, u nichž rozlišujeme, který je levý a který pravý.

Definice: Pro vrchol v binárního stromu T značíme:

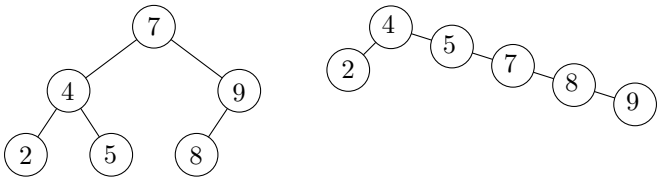
- $\ell(v)$ a $r(v)$ – levý a pravý syn vrcholu v
- $L(v)$ a $R(v)$ – levý a pravý podstrom vrcholu v
- $p(v)$ – otec vrcholu v (pokud existuje)
- $T(v)$ – příslušný podstrom s kořenem v
- $d(v)$ – hloubka stromu $T(v)$ – délka nejdelší cesty z kořene do listu

Definice: Binární strom T nazveme *binární vyhledávací strom* (BVS), pokud v každém vrcholu T je uložena dvojice (klíč, hodnota) [ztotožníme vrchol s klíčem] a pro každý vrchol $v \in V(T)$ platí:

$$(\forall u \in L(v) : u < v) \,\&\, (\forall u \in R(v) : u > v)\,.$$

Poznamenejme ještě, že právě zavedená podmínka na uspořádání prvků uvnitř binárního vyhledávacího stromu je velmi silná, neboť dává kompletní informaci o uspořádání prvků reprezentované množiny.

Pro lepší ilustraci definice BVS uveďme na obr. 1.1 několik příkladů.



Obr. 1.1: Příklady binárních vyhledávacích stromů

1.2. Operace nad BVS

Nyní popíšeme operace nad binárním vyhledávacím stromem. Operaci vyhledávání prvku jsme už nastínili v neformálním úvodu. Začneme v kořeni, klíč v něm uložený porovnáme s hledaným prvek. Buďto jsme měli štěstí a klíč našli, nebo víme, že se klíč může být jen v levém nebo naopak pravém podstromu, kde celý postup zopakujeme.

Algoritmus FIND
Vstup: kořen BVS v, vyhledávaný klíč x
Výstup: ukazatel na vrchol obsahující x (nebo $\emptyset$)

1. Pokud $v = \emptyset$, vrať $\emptyset$.
2. Pokud $v = x$, vrať v.
3. Pokud $v < x$, vrať FIND$(r(v), x)$.
4. Pokud $v > x$, vrať FIND$(\ell(v), x)$.

Vkládání nového prvku funguje velmi podobně jako vyhledávání, s tím rozdílem, že v okamžiku, kdy by vyhledávací algoritmus měl přejít do neexistujícího vrcholu, tento nový vrchol vytvoříme a vložíme do něj vkládaný prvek.

Algoritmus INSERT
Vstup: kořen BVS v, vkládaný klíč x

1. Pokud $v = x$, skonči. *(Případně nějak vyřeš duplicitu prvků.)*
2. Pokud $v < x$, $s \leftarrow r(v)$, jinak $s \leftarrow \ell(v)$.
3. Jestliže $s = \emptyset$, vytvoř nový vrchol s klíčem x, napoj ho pod $p(s)$ a skonči.
4. Jinak zavolej INSERT(s, x).

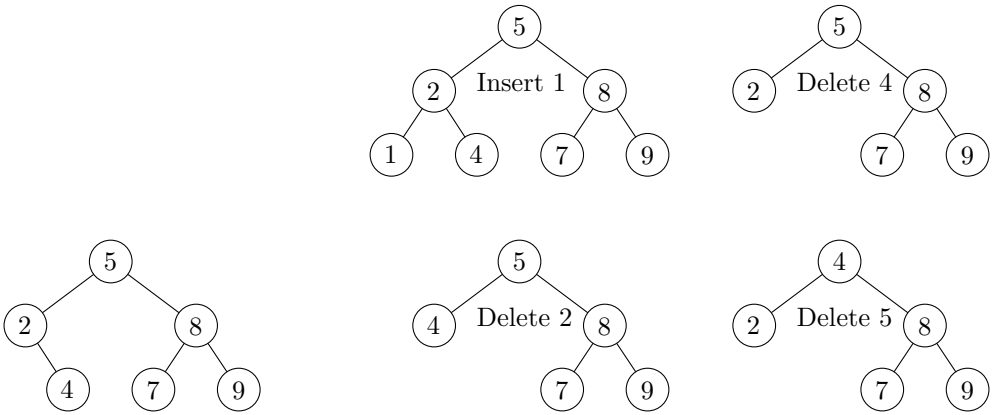
Algoritmus DELETE
Vstup: ukazatel na mazaný vrchol v

1. Pokud v je list, jednoduše list utrhni.
2. Pokud v má jednoho syna s, napojíme s pod $p(v)$ namísto v a vrchol v zrušíme.
3. Jinak má v oba syny, potom do vrcholu v vložíme minimum z $R(v)$ nacházející se ve vrcholu m. Vrchol m umíme smazat protože je to buď list nebo má jen pravého syna.

Poznámka: Pokud má vrchol v při operaci *Delete* oba syny, je vložení minima z $R(v)$ ekvivalentní s vložením maxima z $L(v)$.

Příklady operací INSERT a DELETE aplikovaných na BVS ilustrujeme na obr. 1.2.

Čtenáři ponecháváme ve cvičeni 1.2.1 k rozmyšlení, jak všechny výše uvedené operace realizovat i bez použití rekurze. To je ve většině imperativních programovacích jazyků i podstatně rychlejší a jednodušší implementace. Zároveň si však povšimněme, že rekurzivní povaha BVS i operací nad ním z něj činí přirozenou datovou strukturu k použití v čistě funkcionálních programovacích jazycích.



Obr. 1.2: Příklad operací INSERT a DELETE aplikovaných na strom vlevo

V průběhu všech tří operací se kráčí od kořene v nejhorším případě až do nejhlubšího listu. Hloubka N-prvkového stromu může být $\Theta(N)$, když strom bude (téměř) lineární spojový seznam. Takovéto degenerované stromy vzniknou snadno, například přidáváním setříděné posloupnosti. Naopak když bude strom pěkně vyváženě vystavěný, dostaneme složitost $\Theta(\log N)$. Vidíme tedy, že složitost operací stojí a padá s hloubkou stromu. Shrneme tedy složitost v následujícím tvrzení.

Důsledek: Pro N-prvkový binární vyhledávací strom T je časová složitost operací FIND, INSERT a DELETE $\Theta(\textit{hloubka } T)$.

Uvedeme nyní podmínku, která zajišťuje dobrou hloubku stromu.

Definice: Binární vyhledávací strom T nazveme *dokonale vyvážený*, pokud pro každý jeho vrchol v platí

$$||L(v)| - |P(v)|| \leq 1.$$

Dokonale vyvážený binární strom bude mít určitě logaritmickou hloubku. Jak lze takový strom konstruovat nebo dokonce průběžně udržovat při mazání a vkládání prvků? Odpověď není snadná; buďto strom zkonstruujeme naráz staticky, anebo se smíříme s tím, že operace na něm budou pomalejší než $\Theta(\log N)$. Algoritmus pro postavení dokonale vyváženého BVS ze setříděného pole ponecháme čtenáři jako cvičení.

Lemma: Buď operace vkládání nebo operace mazání v dokonale vyváženém BVS trvá $\Omega(N)$.

Důkaz: Nechť $N = 2^k - 1$ je počet vrcholů BVS T. Pak má dokonale vyvážený BVS T určený tvar a je jednoznačné, která hodnota je ve kterém vrcholu. Nechť nejmenší číslo v T je x a největší je $x + N - 1$.

Proveďme posloupnost operací

$$\mathit{Insert}(x + N), \mathit{Delete}(x), \mathit{Insert}(x + N + 1), \mathit{Delete}(x + 1), \ldots$$

Za každou dvojicí operací se aspoň polovina listů posune o patro výš. Víme, že listů ve stromě T je $(N+1)/2$. Tedy aspoň jedna z operací INSERT nebo DELETE trvá $\Omega(N)$. □

Poznamenejme ještě, že ve skutečnosti obě operace INSERT i DELETE současně musí trvat $\Omega(N)$, ale důkaz je o trochu obtížnější.

Cvičení:

1. Operace FIND, INSERT a DELETE jsme popsali rekurzivně. Navrhněte, jak je dělat bez použití rekurze.
2. Navrhněte algoritmus, který ze setříděného pole vyrobí v lineárním čase dokonale vyvážený BVS.
3. Navrhněte algoritmus, který ze setříděného pole vyrobí v lineárním čase dokonale vyvážený BVS.
4. Navrhněte algoritmus, který dostane dva BVS T_1 a T_2 sloučí jejich obsah do jediného BVS. Algoritmus by měl pracovat v čase $\mathcal{O}(|T_1| + |T_2|)$.
5. Navrhněte operaci Split, která dostane BVS T a hodnotu s, a rozdělí strom na dva BVS T_1 a T_2 takové, že hodnoty v T_1 jsou menší než s a hodnoty v T_2 jsou větší než s.

1.3. Zavedení AVL stromů

Vidíme tedy, že dokonale vyvážené stromy nejsou vhodné. Potřebovali bychom totiž, aby se strom dal také efektivně udržovat. Zavedeme proto slabší podmínku.

Definice: Binární vyhledávací strom T nazveme *hloubkově vyvážený*, pokud pro každý jeho vrchol $v \in V(T)$ platí

$$|h(L(v)) - h(P(v))| \leq 1.$$

Stromům s hloubkovým vyvážením se říká *AVL stromy*[1].

Důležitou vlastností AVL stromů je jejich logaritmická hloubka.

Tvrzení: AVL strom na N vrcholech má hloubku $\Theta(\log N)$.

Důkaz: Definujme čísla

$$A_k = \text{minimální počet vrcholů AVL stromu hloubky } k.$$

Stačí nyní ukázat, že posloupnost A_k roste exponenciálně.

Jak bude vypadat minimální AVL strom o k hladinách? S počtem hladin určitě poroste počet vrcholů, proto pro každý vrchol budeme chtít, aby se hloubky jeho synů lišily. Tedy kořen v bude mít syna a hloubky $k-1$ a syna b hloubky $k-2$

[1] Navrhli je ruští matematikové G. M. Adělson-Veľskij a E. M. Landis, podle nichž jsou také pojmenovány.

takové, že stromy $T(a)$ a $T(b)$ jsou minimální AVL stromy o počtu hladin $k-1$ a $k-2$.

Pro malá k dokážeme určit A_k ručně, pro větší k jsme právě sestrojili rekurentní vztah.

$$\begin{aligned} A_0 &= 1 \\ A_1 &= 2 \\ A_2 &= 4 \\ A_3 &= 7 \\ &\vdots \\ A_k &= 1 + A_{k-1} + A_{k-2}. \end{aligned}$$

Rekurentní vzorec jsme dostali rekurzivním stavěním stromu hloubky k: nový kořen a 2 podstromy o hloubkách $k-1$ a $k-2$.

Lze poměrně snadno dokázat, že $A_n = F_{n+2} - 1$ (kde F_n je n-té Fibonacciho číslo). My se však bez analýzy Fibonacciho posloupnosti obejdeme, protože nám stačí dokázat, že A_k rostou exponenciálně a nepotřebujeme přesný vzorec pro hodnotu A_k. Tento fakt přenecháme čtenáři do cvičení 1.4.1.

Indukcí dokážeme, že $A_k \geq 2^{\frac{k}{2}}$. První indukční krok jsme již ukázali. Pro $k \geq 2$ platí

$$A_k = 1 + A_{k-1} + A_{k-2} > 2^{\frac{k-1}{2}} + 2^{\frac{k-2}{2}} = 2^{\frac{k}{2}} \cdot (2^{-\frac{1}{2}} + 2^{-1}) \cong 2^{\frac{k}{2}} \cdot 1.21 > 2^{\frac{k}{2}}.$$

Tímto jsme dokázali, že na každé hladině je minimálně exponenciálně vrcholů, což nám zaručuje hloubku $\mathcal{O}(\log N)$. Druhou nerovnost nahlédneme zkoumáním B_k maximálního počtu vrcholů AVL stromu hloubky k. Ta je však shora omezena počtem vrcholů v dokonale vyváženém stromě, který je $\Theta(\log N)$, z čehož vyplývá výsledná hloubka AVL stromů $\Theta(\log N)$. □

1.4. Operace nad AVL stromy

Operace vyhledání prvku je naprosto stejná jako FIND v binárních stromech.

Důraz klademe na operace INSERT a DELETE, protože při nich musíme ošetřit udržení struktury AVL stromů. První nutnou podmínkou je pro návrh těchto operací je, že v každém vrcholu AVL stromu budeme udržovat informaci o vyvážení hloubky jeho podstromů. Protože definice AVL stromů povoluje v rozdíly hloubek pouze o 1, vystačíme s třemi symboly.

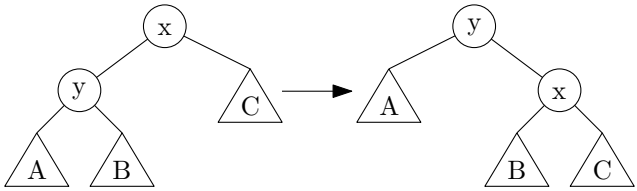
Dostaneme tedy tři typy vrcholů AVL stromu, pro které budeme používat následující značení:

- *Vrchol typu* $\oplus$, pokud je pravý podstrom hlubší
- *Vrchol typu* $\ominus$, pokud je levý podstrom hlubší a
- *Vrchol typu* $\odot$ *(nula)*, který má oba podstromy shodné hloubky.

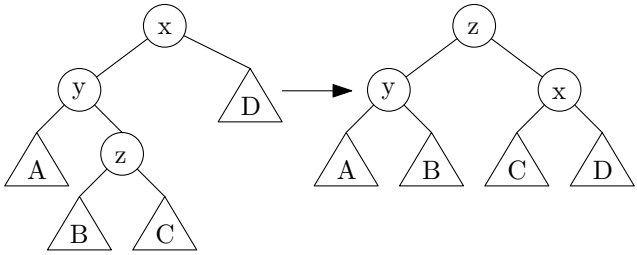
Stromové rotace

V průběhu algoritmů vkládání a mazání prvku bude nějak třeba průběžně udržovat vyváženost AVL stromů. Dosáhneme toho tzv. rotacemi. Jde v zásadě o převrácení hrany mezi původním otcem (kořenem podstromu) a nevyváženým vrcholem tak, aby byli i po přeskupení synové vzhledem k otcům správně uspořádáni.

My budeme používat dva typy rotací: *jednoduchou* a *dvojitou*. Namísto slovního popisu tentokrát čtenáře odkážeme na popis obrázkem. Jednoduchá rotace je na obr. 1.3 a dvojitá na obr. 1.4.



Obr. 1.3: Jednoduchá rotace



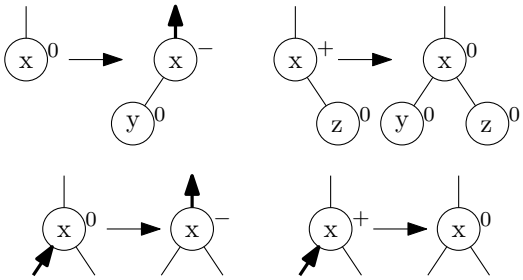
Obr. 1.4: Dvojitá rotace

Vkládání do AVL stromu

Nový prvek vložíme jako list. Nový list má vždy „znaménko“ nula $\odot$. Předpokládáme, že patří nalevo od posledního otce. Podíváme se na znaménko jeho otce:

- *měl* $\odot$ *(neměl syna)* $\rightarrow$ *teď má* $\ominus$, po struktuře stromu nahoru posíláme informaci, že se podstrom prohloubil o 1, což může mít samozřejmě vliv na znaménka vrcholů na cestě ke kořeni.
- *měl* $\oplus$ *(měl pravého syna, který je listem)* $\rightarrow$ *teď má* $\odot$, hloubka podstromu se nemění
- *měl* $\ominus$ – nenastane, protože v binární struktuře nemohou být dva leví synové

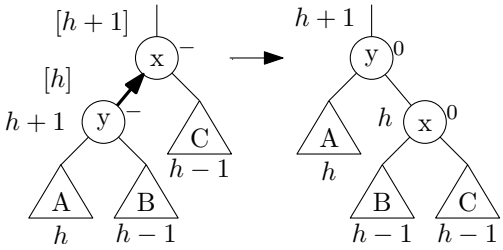
Připadne-li přidaný list napravo, řešíme zrcadlově.



Prohloubil-li se strom vložením nového listu, musíme pracovat s vyvážením:

- Informace o prohloubení přišla zleva *do vrcholu typu* $\odot \rightarrow$ mění jej na vrchol se znaménkem $\ominus$ a informace o prohloubení je třeba poslat o úroveň výš.
- Informace o prohloubení přišla zleva *do vrcholu typu* $\oplus \rightarrow$ mění jej na vrchol se znaménkem $\odot$, hloubka je vyrovnána, dál nic neposíláme.
- Informace o prohloubení přišla zleva *do vrcholu s* $\ominus \rightarrow$

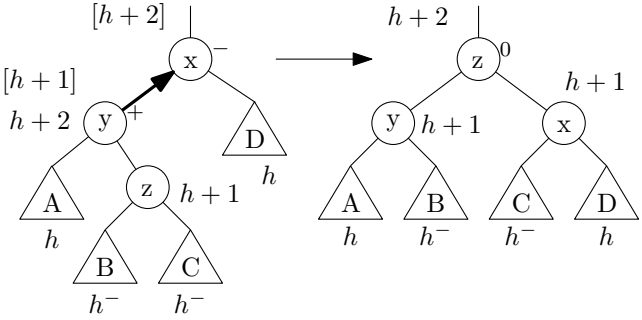
  rozebereme na tři případy podle znaménka vrcholu, ze kterého přišla informace o prohloubení:
  - Informace přišla *z vrcholu typu* $\ominus \rightarrow$ provedeme rotaci doprava tak, že novým kořenem se stane vrchol y, ze kterého přišla informace o prohloubení.



    Pozorování 1: znaménko vrcholů y a x je $\odot$

    Pozorování 2: hloubka před vkládáním byla $h + 1$ a nyní je také $h + 1$, tedy nemusíme dále posílat informaci o prohloubení a můžeme skončit
  - Informace přišla *z vrcholu typu* $\oplus$
    - uvažme ještě vrchol z jako pravého syna vrcholu y, ze kterého prišla informace o prohloubení, a jeho podstromy B a C
    - vrcholy B a C mají hloubku h nebo $h-1 \rightarrow$ označme ji tedy $h-$ (to zřejmě protože vrchol y má znaménko $\oplus$, tedy jeho pravý podstrom s kořenem z má hloubku $h + 1$)

- provedeme dvojrotaci tak, že novým kořenem se stane vrchol z



Pozorování 1: znaménko vrcholu z bude $\odot$

Pozorování 2: znaménka vrcholu x a y se dopočítají v závislosti na hloubce B a C

Pozorování 3: rozdíl hloubky pravého a levého podstromu bude u těchto vrcholů 0 nebo 1

Pozorování 4: hloubka před vkládáním byla $h + 2$ a nyní je také $h + 2$, tedy nemusíme dále posílat informaci o prohloubení a můžeme skončit

- informace přišla *z vrcholu typu* $\odot$ – to nemůže nastat, protože v tom případě by nešlo o prohloubení

Delete – odebrání vrcholu z AVL stromu Buď mažeme list nebo mažeme vrchol, který měl nějaké syny.

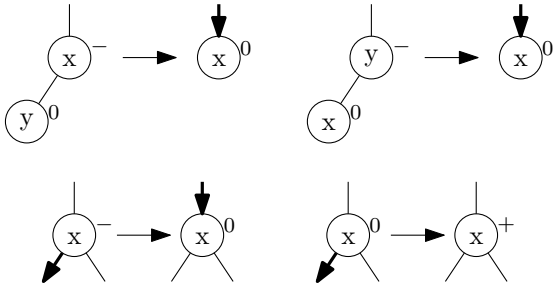
- pokud mažeme list, podíváme se na typ otce. Předpokládáme mazání levého syna.
  - byl typu $\ominus$ (neměl pravého syna) $\rightarrow$ změní se na $\odot$ (vrchol teď nemá žádné syny)
  - byl typu $\odot$ (měl oba syny) $\rightarrow$ změní se na $\oplus$

  (mažeme-li pravý list, řešíme zrcadlově)
- mažeme vrchol s jedním (levým nebo pravým) synem $\rightarrow$ syn nastupuje na místo otce a získává typ $\odot$

  V obou případech posílame informaci o změně hloubky stromu…
- mazaný vrchol měl oba syny (listy) $\rightarrow$ vybereme jednoho ze synů na místo smazaného otce. Hloubka se nemění.
- mazaný vrchol měl syny podstromy $\rightarrow$ na jeho místo vezmeme největší prvek levého podstromu (nebo nejmenší prvek pravého podstromu) a od odebraného (nahrazujícího) listu kontrolujeme vyvážení podstromu.

Úprava vyvážení stromu po odebrání listu z podstromu

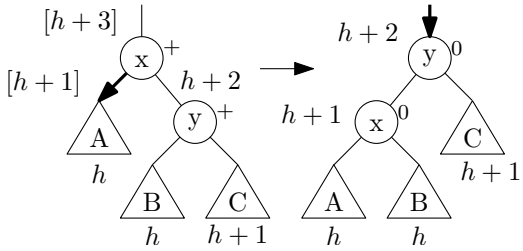
- informace o změně hloubky přišla z levého podstromu do vrcholu typu $\odot \rightarrow$ vrchol se změní na $\oplus$ a dál se hloubka nemění
- informace přišla zleva do vrcholu s $\ominus \rightarrow$ mění se na $\odot$ a posíláme informaci o změně hloubky.



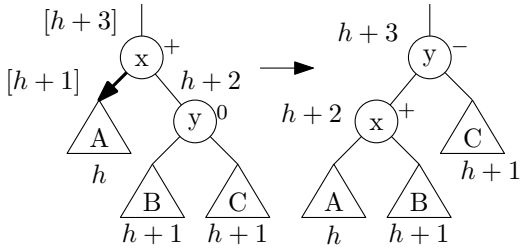
- problémová situace nastává, když informace o změně přišla zleva do vrcholu se znaménkem $\oplus$

Rozebereme na tři případy podle znaménka pravého syna nevyváženého vrcholu

- *pravý syn je typu* $\oplus \rightarrow$ provedeme rotaci vlevo, novým kořenem se stává y (pravý syn), oba vrcholy změní typ na $\odot$ a posíláme informaci o změně hloubky.

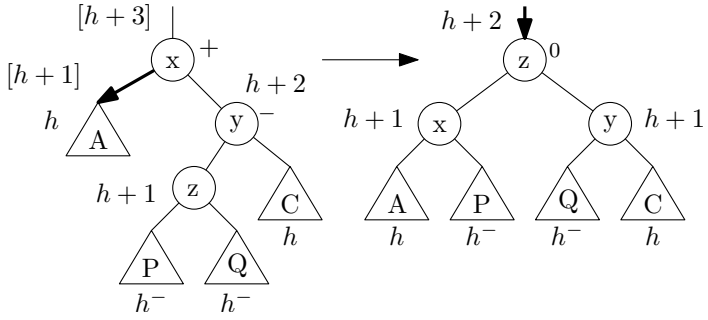


- *pravý syn je typu* $\odot \rightarrow$ provedeme opět rotaci vlevo, kořenem se stává y, následně se u y změní typ na $\ominus$, u vrcholu x se typ nemění. Hloubka stromu se nemění, tudíž není třeba posílat informaci.



- *pravý syn je typu* $\ominus \rightarrow$ v tomto případě uvažujeme ještě vrchol z jako levého syna vrcholu y, s podstromy B a C, podstromy B a C

mají hloubku h nebo $h-1$. Provedeme dvojrotaci, napřed vpravo rotujeme vrcholy z a y, potom vlevo vrcholy x a z tak, že se z stane novým kořenem, typ vecholu x bude potom $\ominus$ nebo $\odot$, typ y $\oplus$ nebo $\odot$ (podle toho, jaké znaménko měl původně vrchol z), typ z bude $\odot$ a opět posíláme informaci o změně hloubky stromu.



Cvičení:

1. Dokažte, že pro minimální velikost A_k AVL stromu hloubky k platí vztah $A_k = F_{k+2} - 1$ (kde F_n je n-té Fibonacciho číslo).