AVL fa

ADATSZERKEZETEK ÉS ALGORITMUSOK
6. GYAKORLAT

AVL fa

Bináris rendezőfa.

A bal és jobb részfák magassága legfeljebb 1-gyel tér el egymástól.

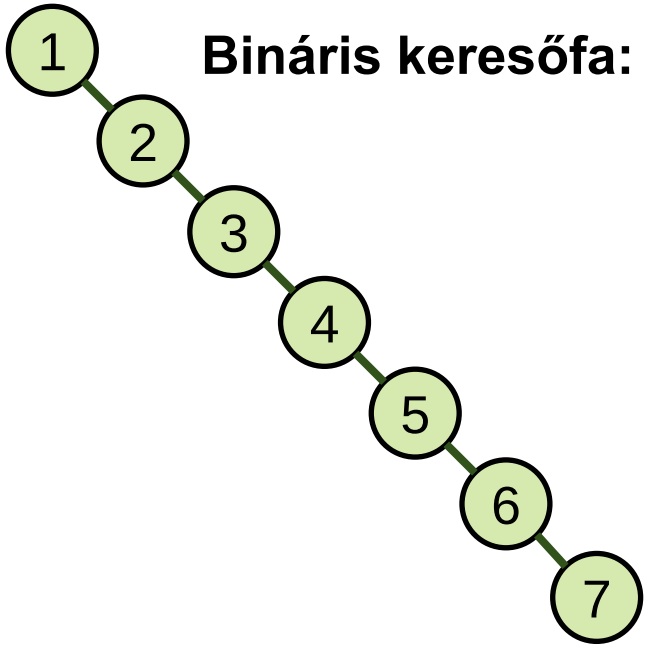
Az AVL fa minden részfája is AVL fa.

Motiváció

Szúrjuk be a következő elemeket egy fába az alábbi sorrendben:

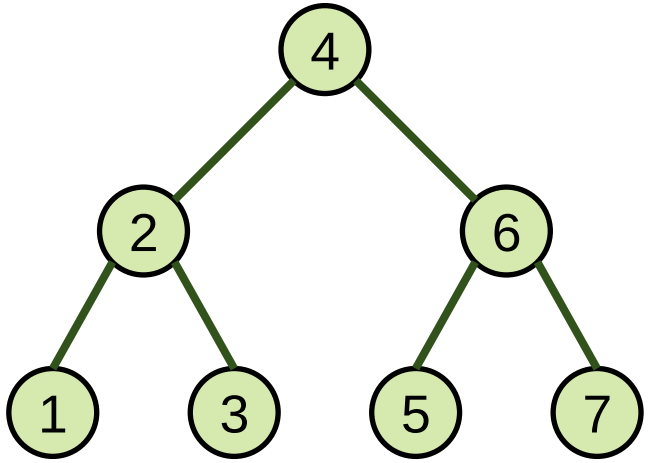
1, 2, 3, 4, 5, 6, 7

Bináris keresőfa:



A fa magassága **N**,
mintha csak egy lista lenne

AVL fa:



Garantált **1,44*log$_2$N** magasság

Bináris keresőfák teljesítménye

Beszúrt random elemek száma	Kiegyensúlyozatlan fa magassága *	AVL fa magassága *
10	6	4
100	12	8
1000	23	12
10000	37	16
100000	159	19

* egy teszt eredménye, csak a nagyságrendek szemléltetésére

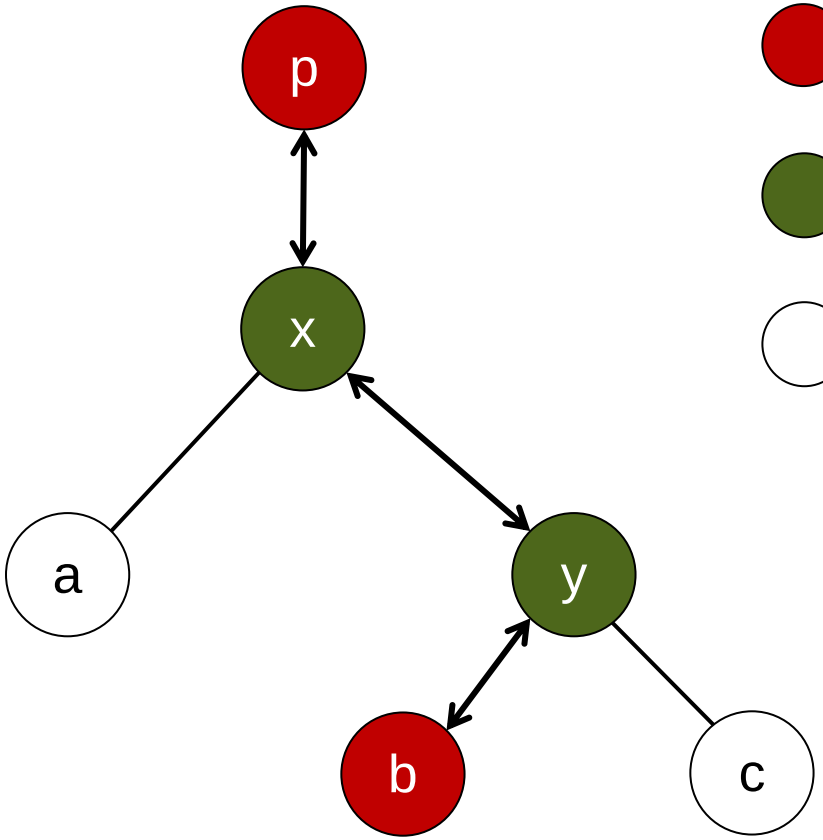
Forgatások

Bináris keresőfákban a forgatások megváltoztatják a fa alakját, a sorrendiség megőrzése mellett.

A forgatások során egy csúcspont eggyel lejjebb, és annak egyik gyereke pedig eggyel feljebb kerül a fában.

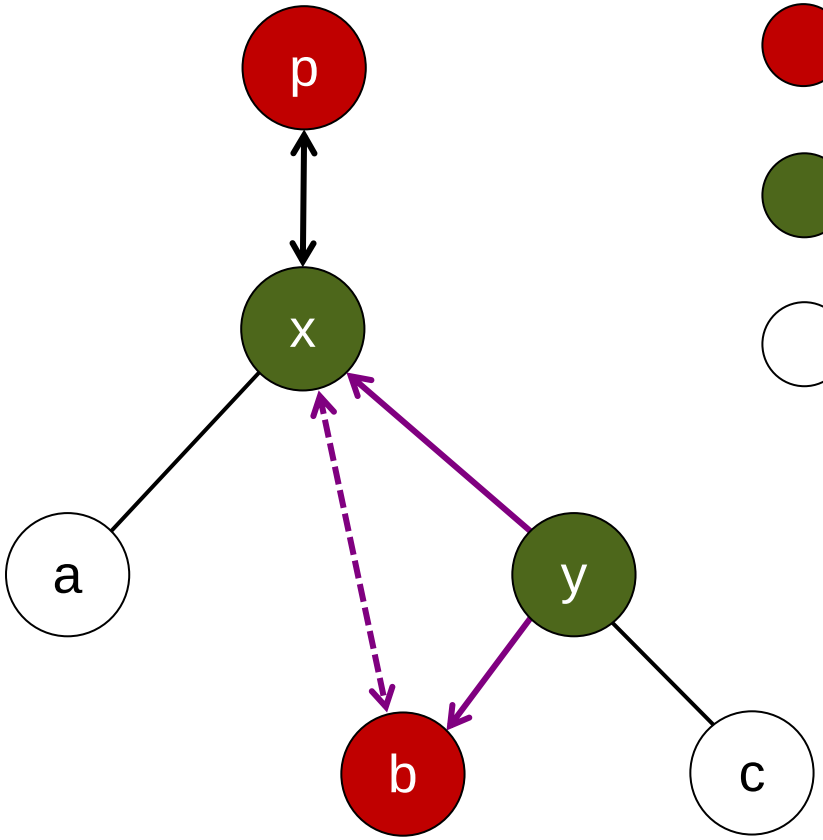
A forgatás irányát mi úgy definiáljuk, hogy merre mozdulnak el a csomópontok.

Forgatások: balra forgatás *x* körül - I



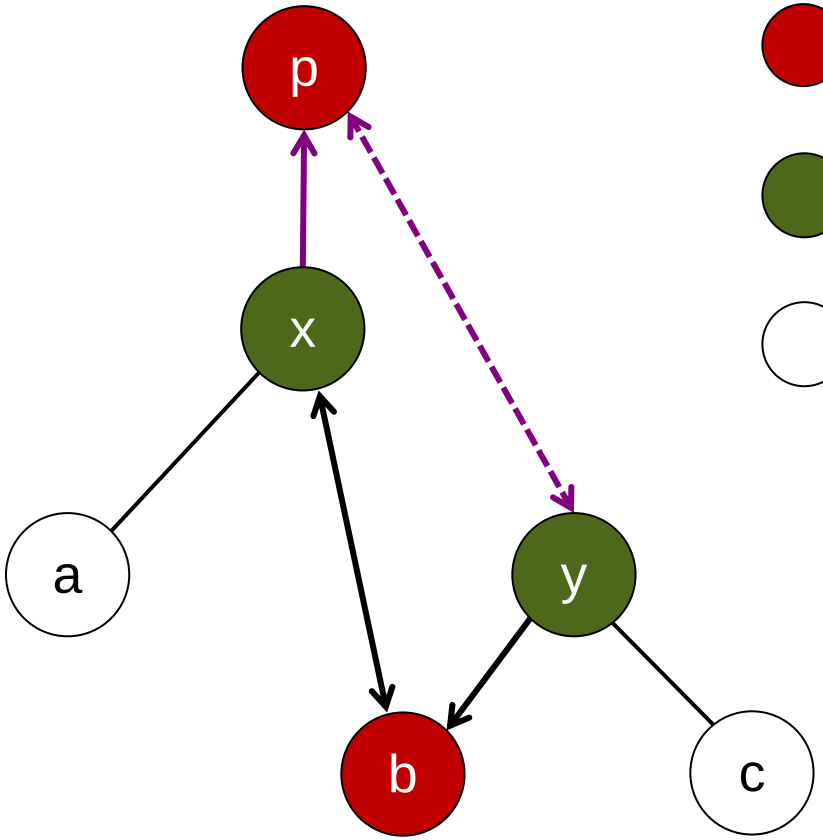
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: balra forgatás *x* körül - II



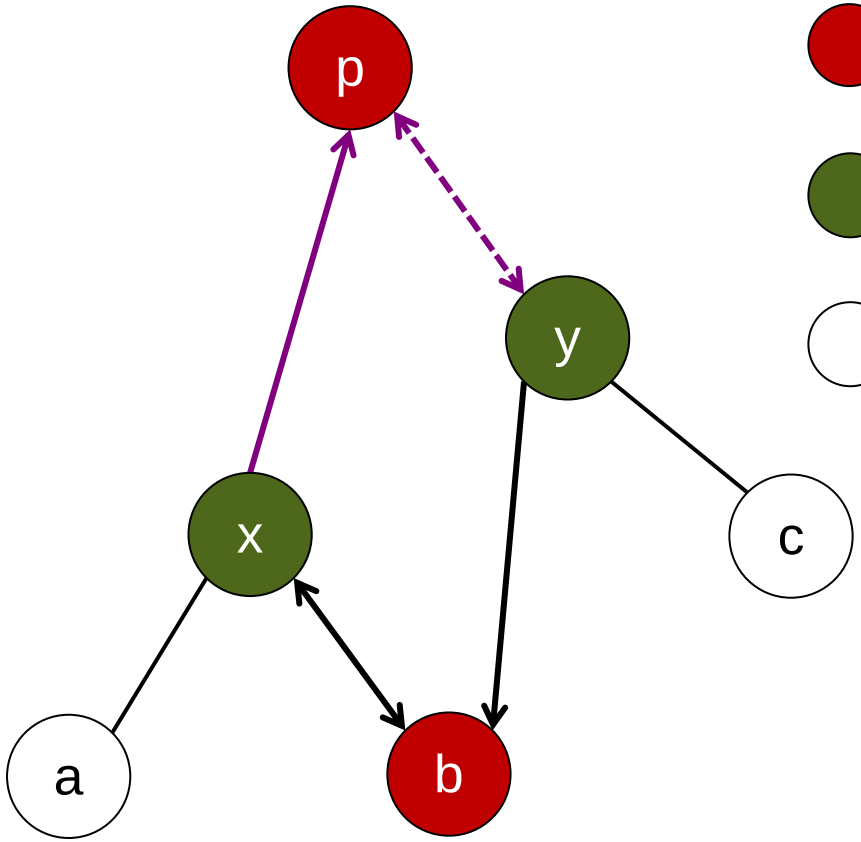
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: balra forgatás *x* körül - III



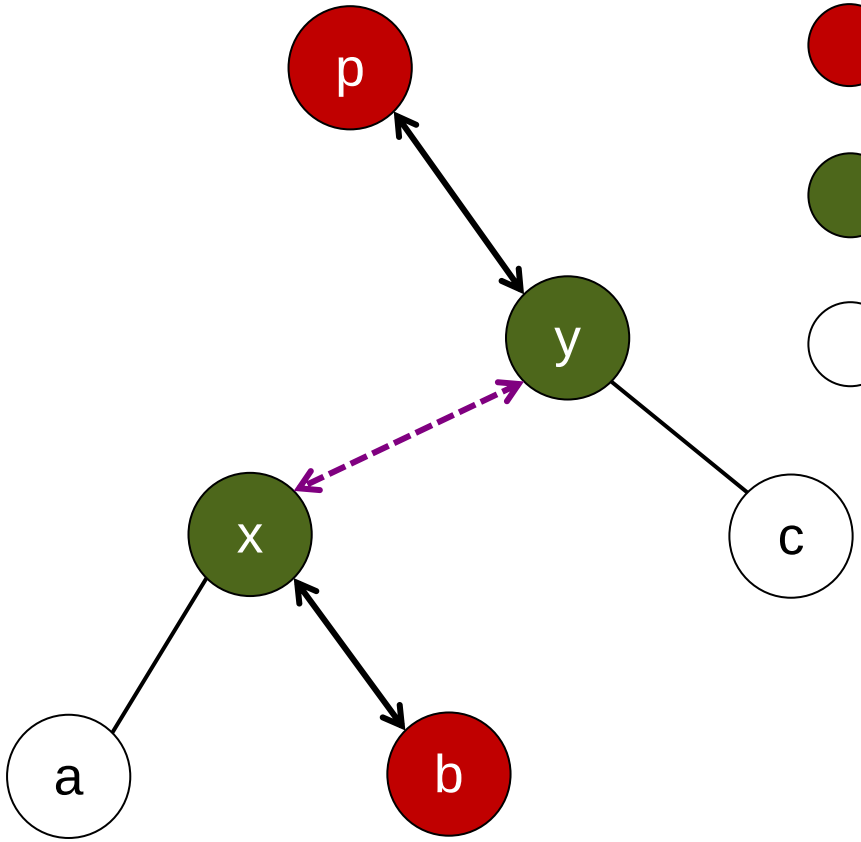
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: balra forgatás *x* körül - III



- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: balra forgatás *x* körül - IV



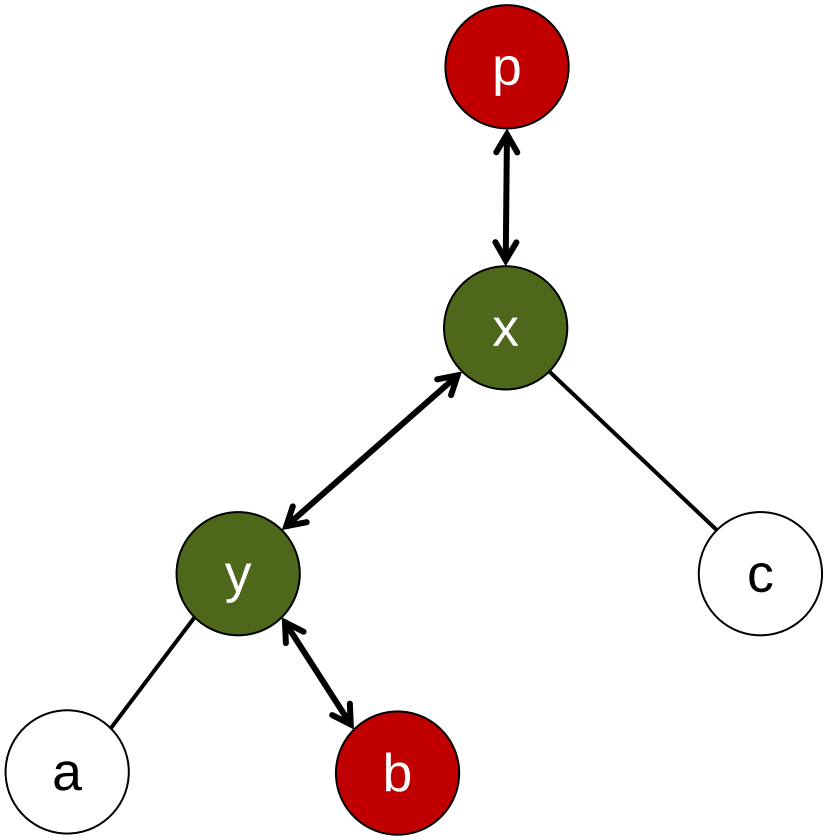
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Órai feladat 1

Nyisd meg a kiadott kódot és implementáld a balra forgatást!

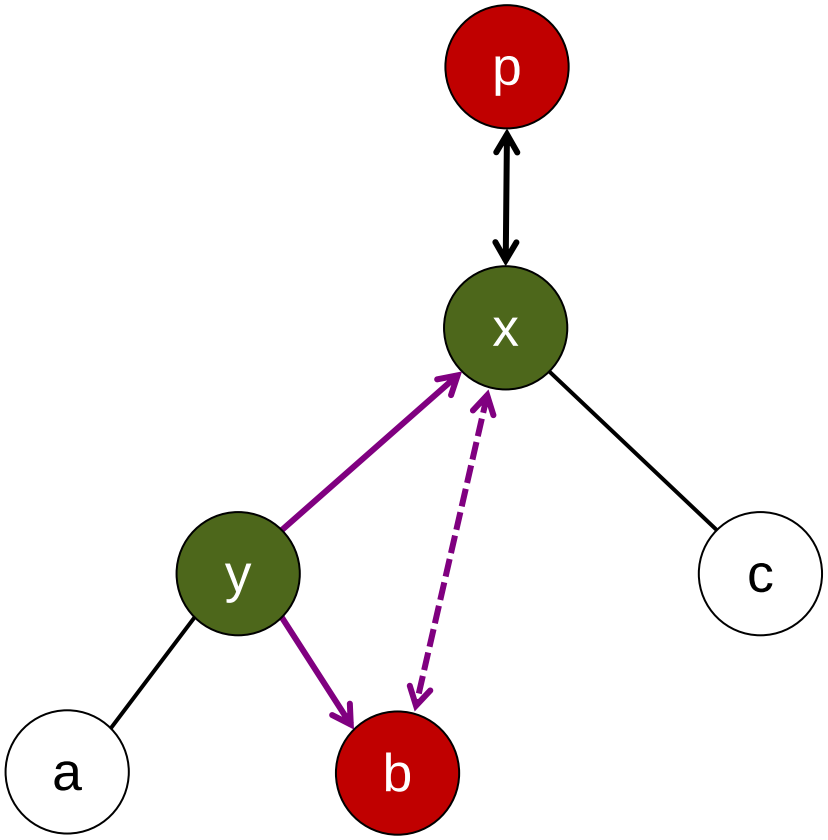
`avl_tree<T>::_rotate_left`

Forgatások: jobbra forgatás *x* körül - I



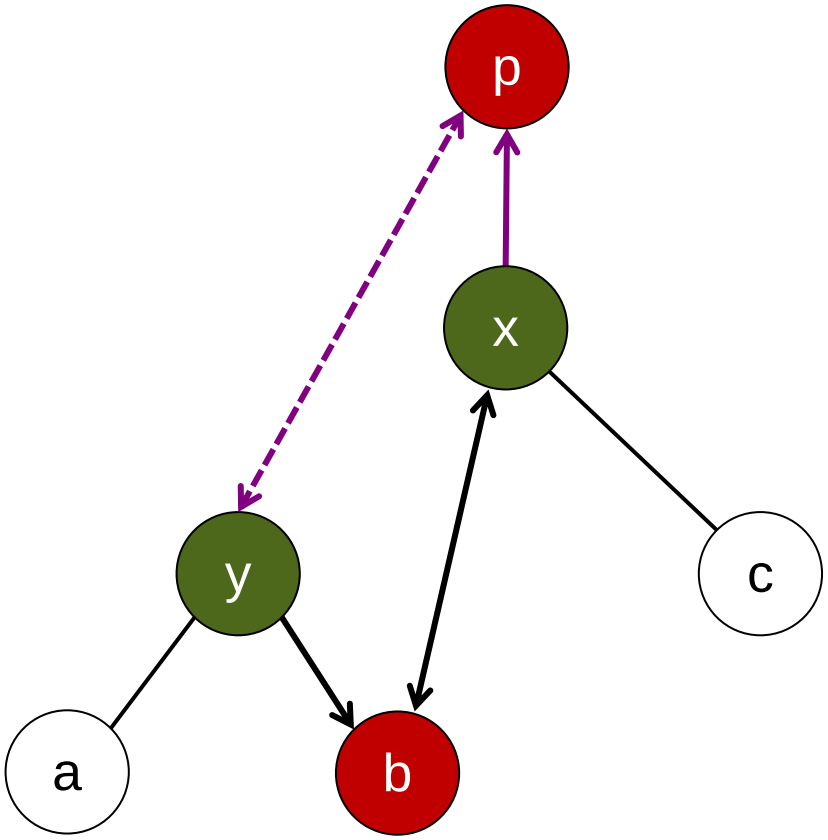
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: jobbra forgatás *x* körül - II



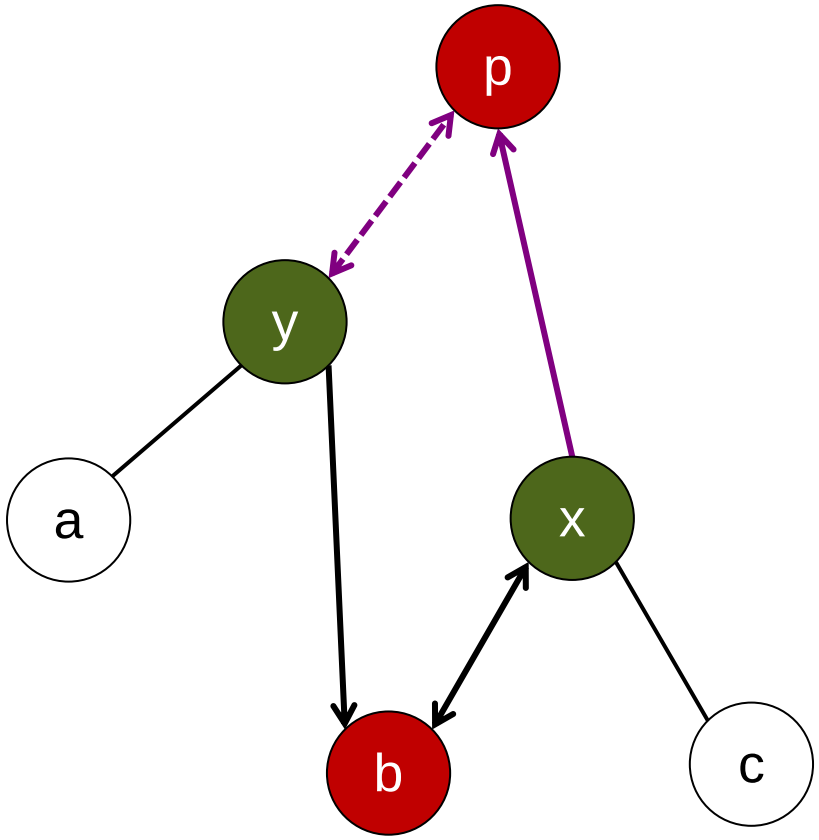
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: jobbra forgatás *x* körül - III



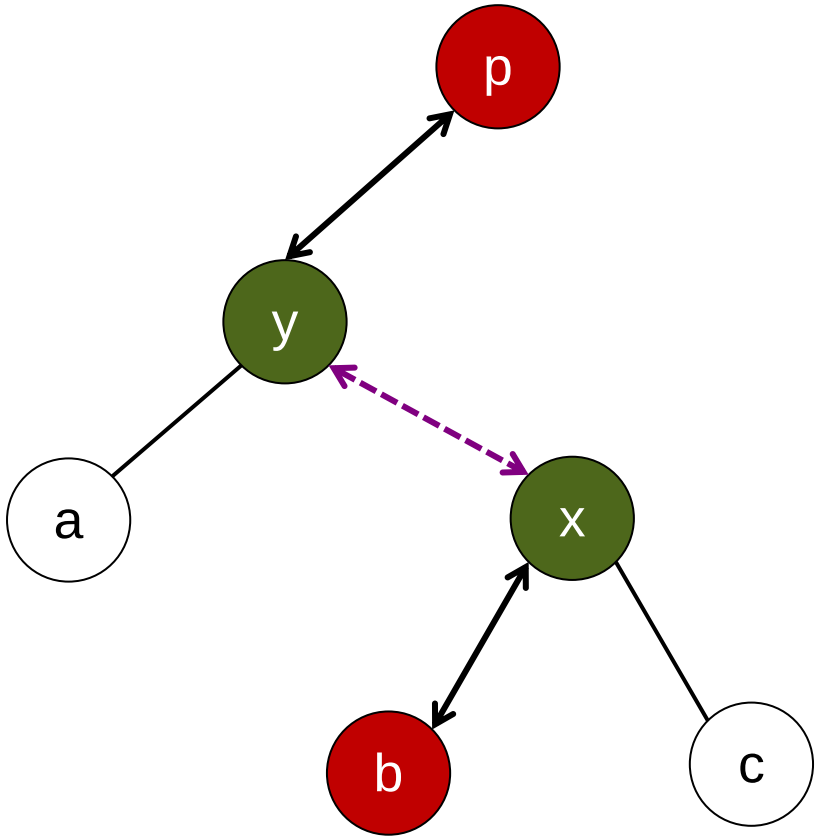
- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: jobbra forgatás *x* körül - III



- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Forgatások: jobbra forgatás *x* körül - IV



- Figyelni kell, mert lehetnek NULL pointerek is!
- Nem lehet NULL.
- Nem kell külön foglalkozni velük, nem változnak a kapcsolataik.

Órai feladat 2

Nyisd meg a kiadott kódot és implementáld a jobbra forgatást!

`avl_tree<T>::_rotate_right`

AVL fa – részfák magassága

Ahhoz, hogy a bináris keresőfa kiegyensúlyozottságát „olcsón” fenn tudjuk tartani, minden csúcsot kibővítünk egy **magasság** mezővel, melyben az adott részfa magasságát tároljuk.

Definíció szerint az üres fa magassága nulla.

```
class Node {
public:
	//...
	int height;
	void update_height();
};
```

A magasságot a forgatások után nekünk kell frissíteni az `update_height()` segédfüggvénnyel.

AVL fa – egyensúly információ

A fa kiegyensúlyozásához viszont nem kell a részfák magassága, elég, ha tudjuk a magasságok különbségét. Ezért a csúcs struktúránkat kiegészítjük egy **egyensúly** tagfüggvénnyel, amivel ezt a különbséget lehet lekérdezni.

```
class Node {
public:
      //...
      int balance_factor() const;
};
```

A `balance_factor()`-t úgy definiáljuk, hogy (jobb magasság – bal magasság), ezért egyértelműen megfeleltethető a diákban használt ++, +, 0, -, -- jelöléssel.

AVL fa – kiegyensúlyozás

Minden beszúrás és törlés művelet után meghívunk egy kiegyensúlyoz (`_rebalance()`) függvényt arra a pontra, ahol a módosítás történt.

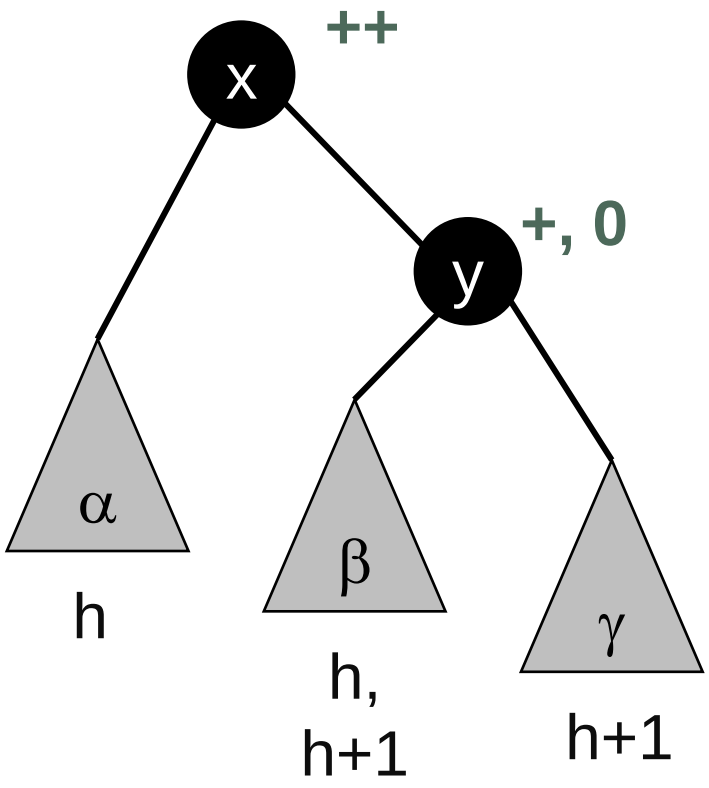
Feladata:

- Ha sérül az AVL fa tulajdonság, forgatásokkal helyreállítani. (Ezek esetei a későbbi diákon.)
- Frissíteni a csúcsokban a magasságinformációt.

Szükséges forgatások száma a legrosszabb esetben:

- Beszúrásnál konstans
- Törlésnél a fa magasságával arányos

Kiegyensúlyozás (++, +) és (++, 0)



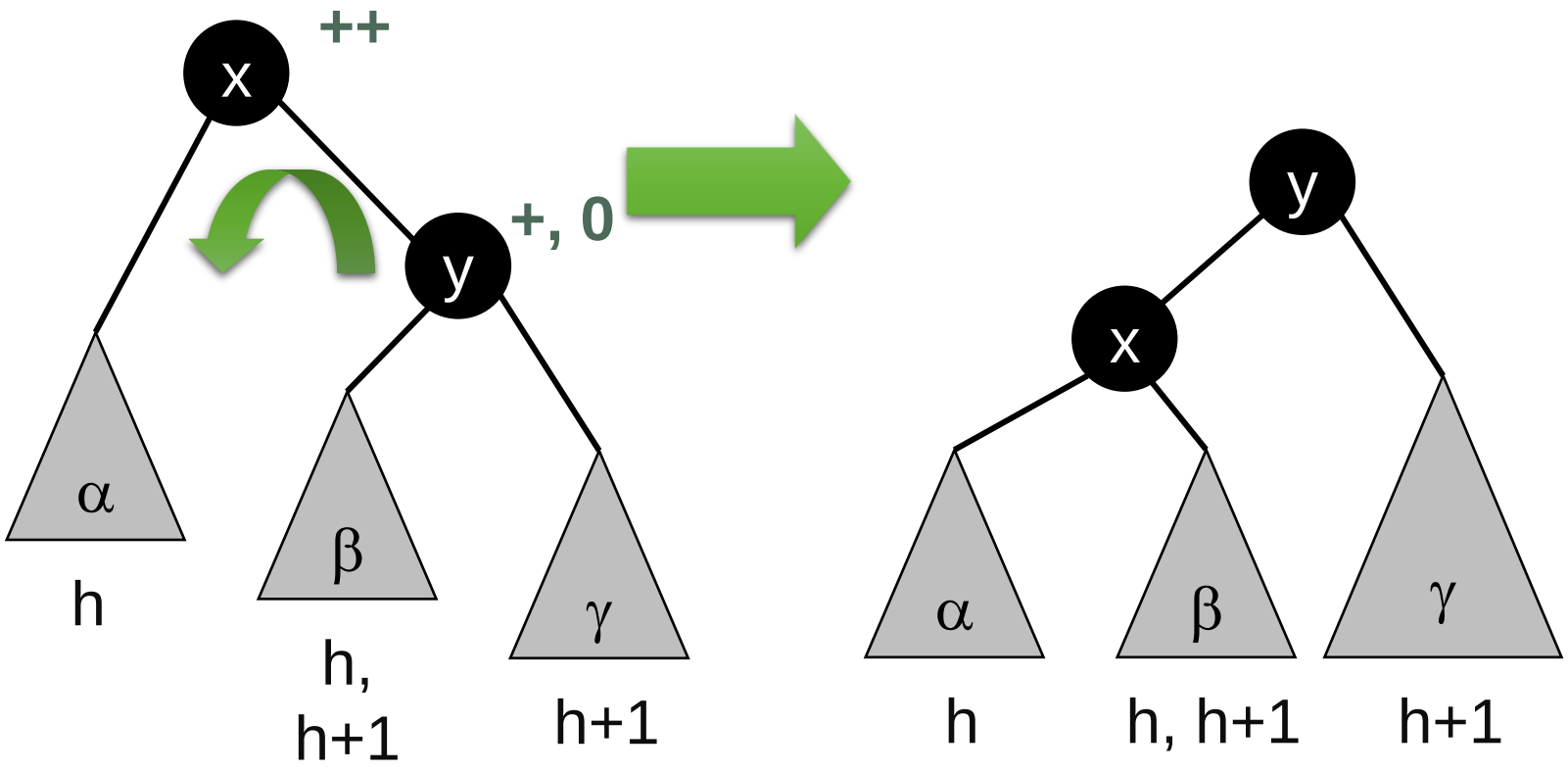
x bal oldali részfája ***h***, jobb oldali pedig ***h+2*** magas, és ez sérti az AVL fa tulajdonságot.

x körüli balra forgatással helyreáll az AVL tulajdonság

Ennek tükörképe a (--,-) és (--,0) esetek.

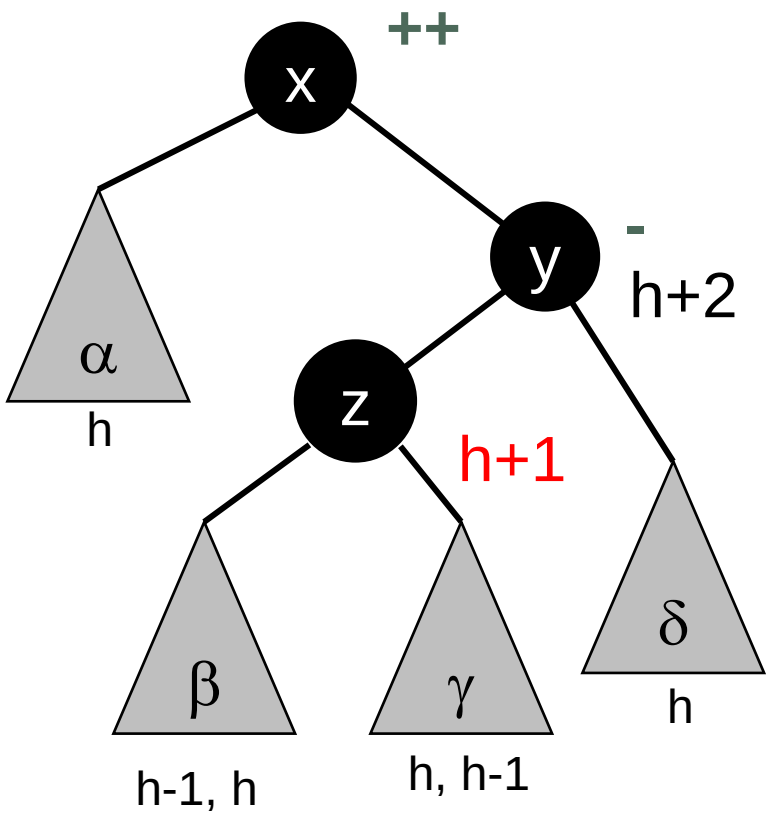
… a forgatás hatása

Ha a részfa magassága változott, úgy **y** szülőjénél folytatjuk a kiegyensúlyozást.



Ne felejtsük el forgatás után a csúcsok magasság mezőit frissíteni!

Kiegyensúlyozás (++, -)



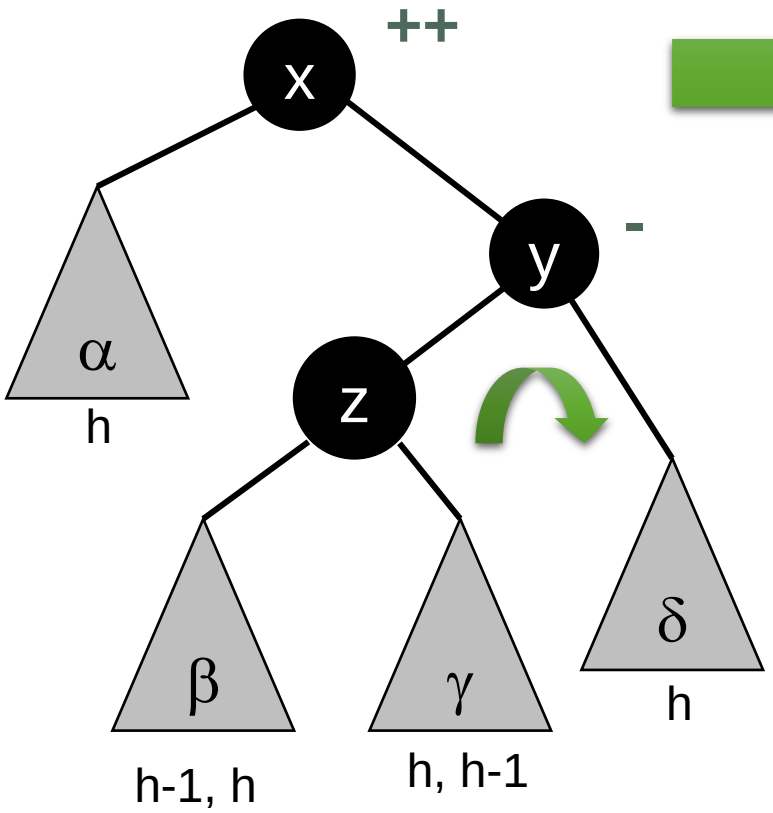
x bal oldali részfája ***h***, jobb oldali pedig ***h+2*** magas, és ez sérti az AVL fa tulajdonságot.

Most az **x** körüli balra forgatás nem oldja meg a problémát, mert a forgatás során az **y**-ról **x**-re átszálló **z** gyökerű fa a „túlsúlyos”.

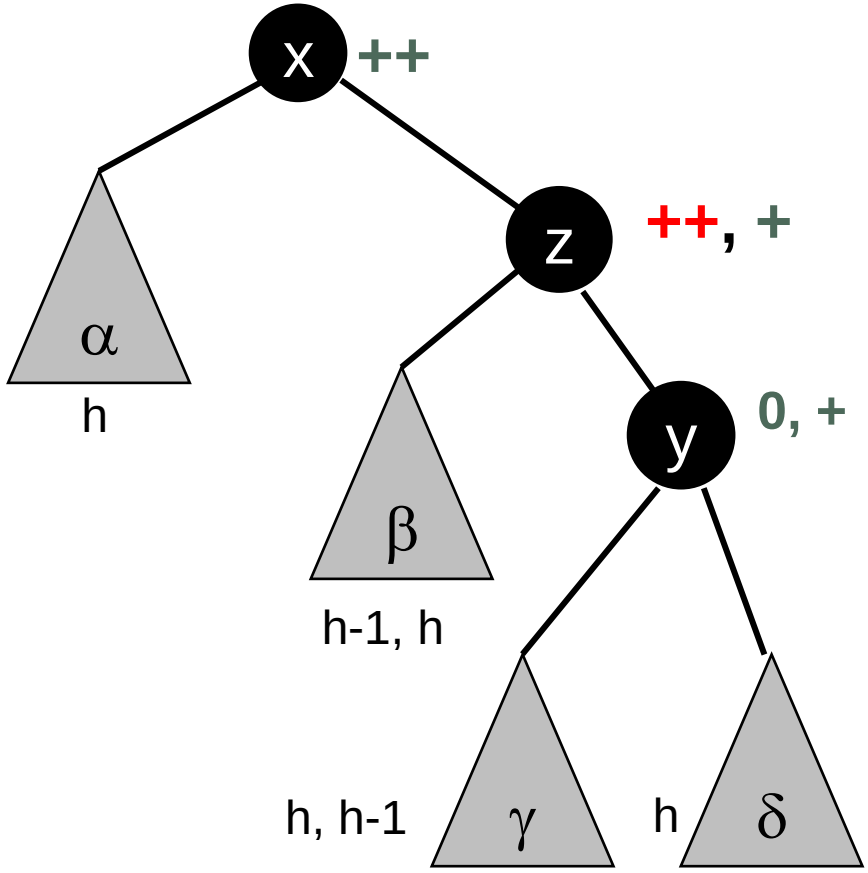
Ezért először egy **y körüli jobbra forgatás**sal szétszedjük **z** részfáit.

Ennek tükörképe a (--, +) eset.

Az első forgatás…

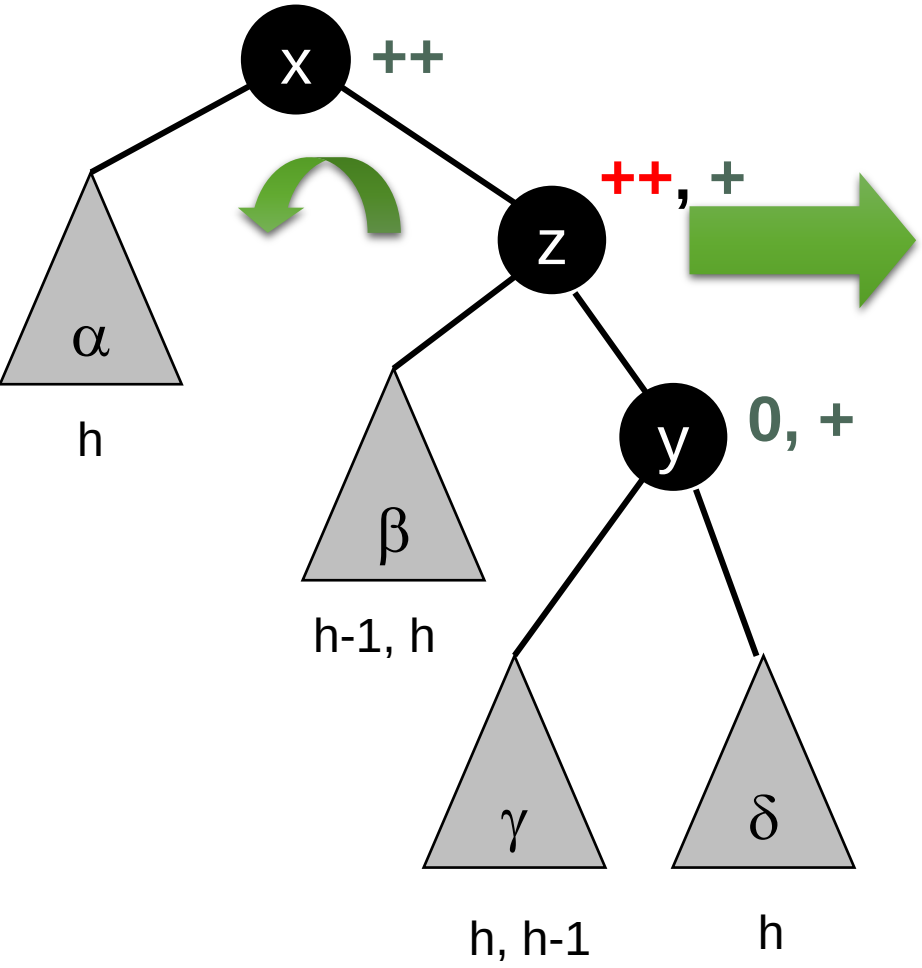


Dupla forgatás: először **y körül jobbra**…

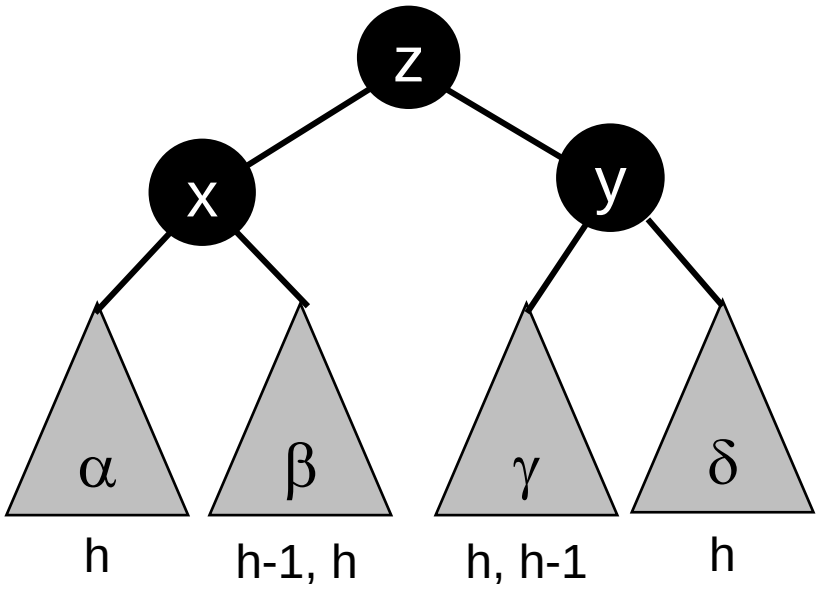


Ne felejtsük el forgatás után a csúcsok magasság mezőjét frissíteni!

… a második forgatás



… majd **x körül balra.**
Ez a második forgatás ugyanaz, mint az első esetnél.



Ezzel helyreállt az AVL fa tulajdonság ebben a részfában.

Ne felejtsük el forgatás után a csúcsok magasság mezőit frissíteni!

Kiegyensúlyozás - röviden

(++, -)	gyereket jobbra, szülőt balra forgatni
(++, 0) és (++, +)	szülőt balra forgatni
(--, +)	gyereket balra, szülőt jobbra forgatni
(--, 0) és (--, -)	szülőt jobbra forgatni

Végül, ha a részfa magassága változott, akkor folytatjuk a kiegyensúlyozást a részfa szülőjével.

Órai feladat 3

Nyisd meg a kiadott kódot és implementáld a kiegyensúlyozó függvényt!

`avl_tree<T>::_rebalance`

Több template típusparaméter (ismétlés)

Több template típusparamétert egyszerűen felsorolással csinálhatunk:

```
template<class K, class V>
class avl_tree
{ //... };

template<class K, class V>
const V & avl_tree<K, V>::iterator::value() const
{
      //...
      return n->value;
}
```

Gyakorló feladat G06F01

Alakítsátok át úgy az AVL fát, hogy kulcs-érték párokat lehessen benne tárolni. Második template paraméterként várja az érték típusát. A kulcshoz tartozó értéket a node-ban tárolja.

Példa a használatára:

```
avl_tree<int, string> szamok;
szamok.insert(6, “hat”);
avl_tree<int, string>::iterator it = szamok.find_it(6);
cout << “6 = “ << it.value();
```

Gyakorló feladat G06F01 (folytatás)

Az így megírt map segítségével

- hozzatok létre egy telefonkönyvet,
- szúrjatok bele pár név/telefonszám párost,
- keressetek rá egy-egy név telefonszámára,
- majd pedig iteráljatok végig a telefonkönyvön és írjátok ki az elemeit a konzolra.

Gyakorló feladat G06F02

AVL fákat, mint halmazokat felhasználva valósítsd meg az unió, metszet és különbség műveleteket!

Ehhez tölts fel két AVL fát véletlen számokkal, majd végezd el a műveleteket!

Művelet: Két AVL fa paraméter, és egy AVL fa visszatérési érték! (Vagyis valóban művelet!)

(megj.: Mivel két egyforma csúcs nem lehet ezért mondhatjuk, hogy halmazok!)

Gyakorló feladat G06F03

AVL fák segítségével készíts egy angol-magyar illetve egy magyar-angol szótár adatbázist.

A fordítandó szavai legyenek a csomópontok kulcsai és a hozzájuk tartozó fordítás pedig az értékek.

- Legyen lehetőség arra, hogy akár több hasonló jelentésű fordítást is felvegyünk egy adott szóhoz.

Készíts felhasználóbarát menüt, ami alkalmas a szótárak használatára:

- Tudjunk keresni egy adott szót és kiírni a hozzá tartozó fordításokat.
- Ha a keresett szó nem szerepel a fában, akkor vegyük fel legalább egy fordítással.

Gyakorló feladat G06F04

Készíts valamilyen iterátort a fához! Egyéni választásod szerint lehet külön iterátor osztály, amelyből több példány is lehet, vagy egyetlen act mutató az AVL fa tagjaként, ahogyan a láncolt listánál láttuk.

Minimális műveletkészlet az iterátornak:

- Fa legkisebb elemére állítás
- Rákövetkező elemre állítás
- Mutatott csúcs értékének lekérdezése
- Lekérdezni, hogy túlfutottunk-e az utolsó elemen

Házi feladat G06F05

Generáljatok kellően sok véletlen adatot, majd szúrjátok be egy AVL fába. (Figyeljünk arra, hogy az AVL fa kiszűri az ismétlődéseket! Ha véletlenül egy már létező elemet hoztunk létre, akkor generáljunk helyette újat!)

Készítsetek statisztikát, hogy mennyi időbe telt egy elem átlagos beszúrása, összesen mennyi idő kellett az AVL fa felépítéséhez, hány forgatásra volt szükség a kiegyensúlyozásokhoz.

Ha elkészült az AVL fa akkor olvassátok ki az elemeket sorrendben egy vektorba.

A most már rendezett elemeket ismét szúrjátok be egy új AVL fába és készítsétek el ugyan azt a statisztikát, mint első esetben.

Értékeljétek ki (txt-ben) a tapasztaltakat!