Tömörítés

Alapfogalmak

- **Ábécé**: $\Sigma = \{\sigma_1,\sigma_2,\ldots,\sigma_d\}$
- **Ábécé mérete**: d
- **Tömörítendő szöveg hossza**: n
- **Kódszó hossza**: $L = \lceil \log_2 d \rceil$ (kettes alapon vett logaritmus, felfelé kerekítve)
- **Kódtáblázat**: Az ábécé karaktereihez hozzárendelt, fix hosszúságú bináris kódokat tartalmazza
- **Kódfa:**
  - levelei az ábécé karaktereihez tartoznak
  - minden élhez egy bináris címkét rendelünk: bal él → 0, jobb él → 1
  - egy levélhez vezető úton az élek címkéit összeolvasva kapjuk meg az adott karakter kódszavát
  - a kódfában nincs olyan kódszó, ami egy másik kódszó előtagja (prefixmentes kód)

Naiv módszer

A naiv egy olyan veszteségmentes tömörítési eljárás, amelyben minden karaktert azonos hosszúságú, fix bitsorozattal kódolunk. A kódhossz a karakterkészlet méretének függvényében a legkisebb olyan egész szám, amelyen minden karakter külön bináris kódot kaphat.

Tömörítési eljárás

1. Meghatározzuk az ábécé elemeit: Σ = { … }
2. Meghatározzuk az ábécé méretét: d = …
3. Meghatározzuk a tömörítendő szöveg hosszát: n = …
4. Meghatározzuk kódszó hosszát: L = $\lceil \log_2 d \rceil$
5. Minden karakterhez egyedi, L bites kódszót rendelünk
6. A bemeneti szöveget karakterenként helyettesítjük a kódszavával
7. A kimeneti fájl tartalmazza:
    a. a kódtáblázatot
    b. a tömörített kódsorozatot

Kitömörítési eljárás

Minden L-bites szakaszhoz a kódtáblázat alapján hozzárendeljük a megfelelő karaktert

Példa 1

- Tömörítendő szöveg: ABRAKADABRA

1. Σ = {A, B, R, K, D}
2. d = 5
3. n = 11
4. L = $\lceil \log_2 5 \rceil = 3$
5.

Karakter	Kód
A	000
B	001
D	010
K	011
R	100

6. 000 001 100 000 011 000 010 000 001 100 000
   A B R A K A D A B R A

- Tömörített szöveg mérete: n * L = 11 · 3 = 33 bit
- Kódtábla mérete: d * 8 + d * 3 = 5 * 8 + 5 * 3 = 40 + 15 = 55
- Teljes tömörített méret: 33 + 55 = 88 bit *(plusz meta adatok)*
- Eredeti méret: 11 * 8 = 88 bit *(plusz meta adatok)*

Példa 2

- Tömörítendő szöveg: ABRAKADABRAABRAKADABRAABRAKADABRA

1. Σ = {A, B, R, K, D}
2. d = 5
3. n = 33
4. L = $\lceil \log_2 5 \rceil = 3$
5.

Karakter	Kód
A	000
B	001
D	010
K	011
R	100

- Tömörített szöveg mérete: n * L = 33 · 3 = 99 bit
- Kódtábla mérete: d * 8 + d * 3 = 5 * 8 + 5 * 3 = 40 + 15 = 55
- Teljes tömörített méret: 99 + 55 = 154 bit *(plusz meta adatok)*
- Eredeti méret: 33 * 8 = 264 bit *(plusz meta adatok)*

Előnye

- Egyszerű
- Az eredeti adat teljesen visszaállítható, semmilyen információ nem vész el
- Minden kódszó fix hosszúságú → a bitsorozatot egyszerűen szakaszokra lehet bontani
- Nem szükséges bonyolult fa vagy keresés (ellentétben pl. a Huffman-kóddal)

Hátránya

- A gyakori karakterek ugyanannyi bitet kapnak, mint a ritkán előfordulók, ezért a tömörítési arány sokkal rosszabb, mint változó hosszúságú kódoknál (pl. Huffman)
- A fájlban a kódtáblát is el kell menteni, ami rövid szövegeknél nagy többletet jelenthet.
- Ha az ábécé túl nagy (d nagy), akkor a szükséges kódszóhossz ($\lceil \log_2 d \rceil$) is megnő, és a tömörítés értelme elvész
- Egy 90%-ban „A”-ból álló szöveg ugyanannyiba kerül, mint egy teljesen vegyes szöveg (ugyanakkora ábécé mellett)

Huffman-kód

A Huffman-kódolás egy olyan veszteségmentes tömörítési eljárás, amelyben a gyakrabban előforduló karakterek rövidebb, a ritkábbak hosszabb kódszót kapnak. A kódfát a karakterek gyakorisága alapján építjük, így a teljes kódolt üzenet hossza optimálisan minimális lesz a prefixmentes kódok között.

Tömörítési eljárás

1. Határozzuk meg a karakterekhez tartozó gyakoriságokat
2. Hozzunk létre minden karakterből egy fát a gyakoriság, mint kulcs segítségével (tehát itt lesz sok-sok 1 csúcsból álló fánk)
3. Tegyük be az összes így kapott fát, egy min-prioritásos sorba (tehát itt a kis fák egy csúcsként fognak viselkedni)
4. Vegyünk ki két csúcsot (fát) a sorból
5. Készítsünk egy szülőt a két fa gyökércsúcsának, ahol a kulcs a két fa gyökércsúcsában lévő kulcs összege lesz, továbbá a bal élt címkézzük „0”-val, a jobb élt pedig „1”-el
6. Az így 2 fából és egy harmadik csúcsból alkotott fát tegyük vissza a min-prioritásos sorba
7. Ismételjük meg az egészet a 4. lépéstől, ameddig a min-prioritásos sorban legalább 2 elem van
8. Ha szerencsénk van, itt már csak egy fa szerepel a sorban, ezt nevezzük kódfának
9. A kódfában balra vagy jobbra egészen a levélig haladva ki tudjuk olvasni az adott karakterhez (ami ugye a levélben van) tartozó kódszót az élek címkéiről
10. Építsük fel a kódtáblázatot a 9. pont alapján
11. A kimeneti fájl tartalmazza:
    a. a kódtáblázatot
    b. a tömörített kódsorozatot

Kitömörítési eljárás

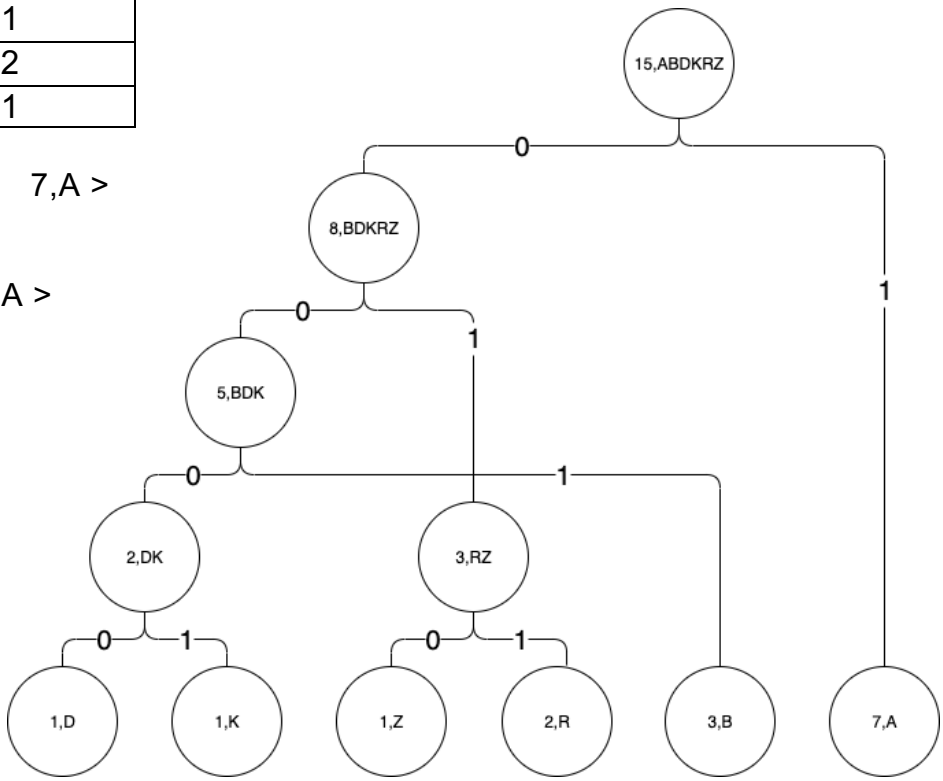
A tömörített szöveget elkezdjük olvasni, és ha találunk olyan sorozatot ami a kódtáblában szerepel, akkor az a sorozat a kódtáblában lévő karakternek fog megfelelni

Példa 1

- Tömörítendő szöveg: AZABBRAKADABRAA
- Σ = {A, Z, B, R, K, D}

Karakter	Előfordulás
A	7
B	3
D	1
K	1
R	2
Z	1

1. < 1,D 1,K 1,Z 2,R 3,B 7,A >
2. < 1,Z 2,DK 2,R 3,B 7,A >
3. < 2,DK 3,B 3RZ 7,A >
4. < 3RZ 5,BDK 7,A >
5. < 7,A 8,BDKRZ >
6. < 15,ABDKRZ >



Karakter	Kód	Előfordulás
A	1	7
B	001	3
D	0000	1
K	0001	1
R	011	2
Z	010	1

- 1 010 1 001 001 011 1 0001 1 0000 1 001 011 1 1
  A Z A B B R A K A D A B R A A

- Tömörített szöveg mérete: 7*1 + 3*3 + 1*4 + 1*4 + 2*3 + 1*3 = 33 bit
- Kódtábla mérete: 6*8 + 1 + 3 + 4 + 4 + 3 + 3 = 66 bit
- Teljes tömörített méret: 33 + 66 = 99 bit *(plusz meta adatok)*
- Eredeti méret: 15 * 8 = 120 bit *(plusz meta adatok)*

Példa 2

- Tömörítendő szöveg:
  AZABBRAKADABRAAAZABBRAKADABRAAAZABBRAKADABRAA
- Σ = {A, Z, B, R, K, D}

Karakter	Előfordulás
A	21
B	9
D	3
K	3
R	6
Z	3

7. < 3,D 3,K 3,Z 6,R 9,B 21,A >

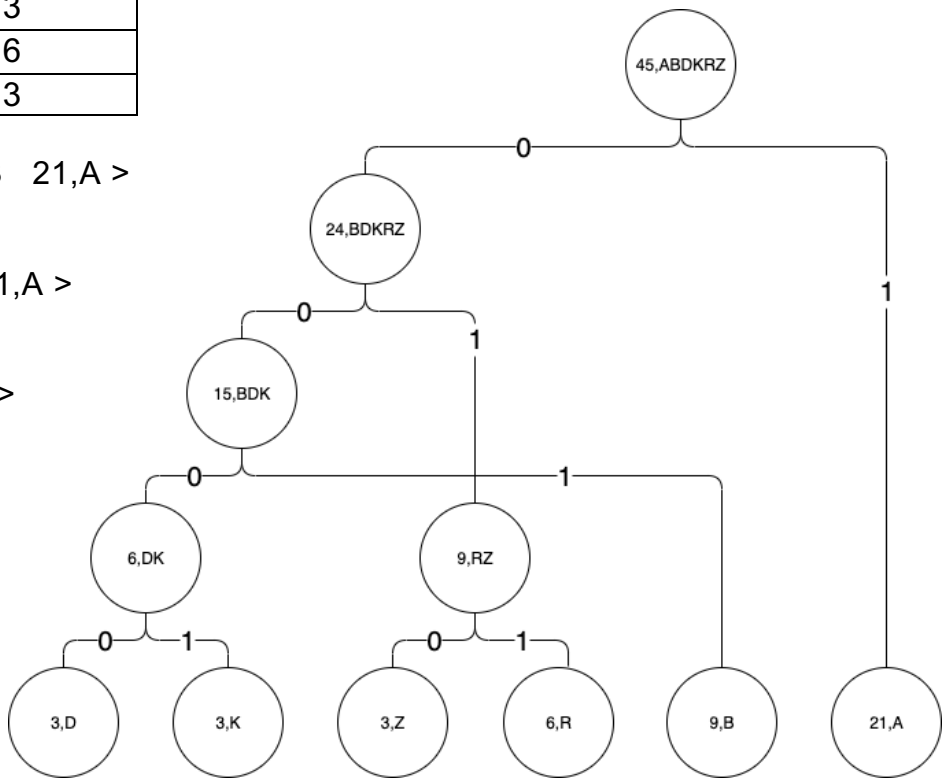
8. < 3,Z 6,DK 6,R 9,B 21,A >

9. < 6,DK 9,B 9,RZ 21,A >

10. < 9,RZ 15,BDK 21,A >

11. < 21,A 24,BDKRZ >

12. < 45,ABDKRZ >



Karakter	Kód	Előfordulás
A	1	21
B	001	9
D	0000	3
K	0001	3
R	011	6
Z	010	3

- Tömörített szöveg mérete: 21*1 + 9*3 + 3*4 + 3*4 + 6*3 + 3*3 = 99 bit
- Kódtábla mérete: 6*8 + 1 + 3 + 4 + 4 + 3 + 3 = 66 bit
- Teljes tömörített méret: 99 + 66 = 165 bit *(plusz meta adatok)*
- Eredeti méret: 45 * 8 = 360 bit *(plusz meta adatok)*

Előnye

- Mindig a lehető legrövidebb átlagos kódhosszt adja meg a karaktergyakoriságok alapján
- Az eredeti adat teljesen visszaállítható, semmilyen információ nem vész el
- Minél nagyobb az eltérés a gyakori és ritka karakterek előfordulásában, annál hatékonyabb

Hátránya

- A dekódoláshoz ismerni kell a kódfát vagy a kódtáblát, ami plusz helyet foglal
- A kódszavak változó hosszúságúak, ezért nem lehet egyszerűen szakaszokra vágni a bitsorozatot

LZW (Lempel–Ziv–Welch) tömörítés

Az LZW tömörítés egy veszteségmentes adat-tömörítési algoritmus. Az algoritmus célja, hogy ismétlődő minták és szimbólumok hatékony kódolásával csökkentse az adatmennyiséget.
Az LZW tömörítés során:

- Nincs szükség előzetes statisztikai elemzésre (ellentétben a Huffman-kódolással)
- A kódtáblázat a feldolgozás közben épül fel

Tömörítési eljárás

1. Vegyük fel a kódtáblázatba az összes karaktert
2. Olvassunk be egy karaktert
    a. Ha nem tudtunk beolvasni (elfogyott a bemenet), írjuk ki az előző „szó” kódját
    b. Végeztünk
    c. Egyébként menjünk tovább a 3. lépésre
3. Konkatenáljuk az előző és a beolvasott „karaktereket”
    a. Ha a konkatenált „szó” benne van a kódtáblázatban
        i. Akkor az előző „szó” legyen a konkatenált „szó”
    b. Ha a konkatenált „szó” nincs benne a szótárban
        i. Vegyük fel a konkatenált „szót” a kódtáblázatba
        ii. Írjuk ki az előző „szó” kódját
        iii. Az előző „szó” legyen a beolvasott „szó”
4. Vissza a 2. lépésre

```
1   inicializál()
2   előző := ””
3   beolvasott := új_karakter
4   if beolvasott = null then
5           kiír(előző)
6           exit()
7   konkatenált := előző + beolvasott
8   ismert := konkatenált ∈ kódtáblázat
9   if ismert then
10          előző := konkatenált
11          jump(line_3)
12  else
13          kódtáblázat.add(konkatenált)
14          kiír(előző)
15          előző := beolvasott
16          jump(line_3)
```

Példa

Tömörítendő szöveg: ABAB ABAA CAAA AAAA A *(space nélkül)*

karakter / szó	kód	lépés	előző	beolvasott	konkatenált	ismert	output
A	1	1	-	A	-	1	-
B	2	2	A	B	AB	0	1
C	3	3	B	A	BA	0	2
AB	4	4	A	B	AB	1	-
BA	5	5	AB	A	ABA	0	4
ABA	6	6	A	B	AB	1	-
ABAA	7	7	AB	A	ABA	1	-
AC	8	8	ABA	A	ABAA	0	6
CA	9	9	A	C	AC	0	1
AA	10	10	C	A	CA	0	3
AAA	11	11	A	A	AA	0	1
AAAA	12	12	A	A	AA	1	-
		13	AA	A	AAA	0	10
		14	A	A	AA	1	-
		15	AA	A	AAA	1	-
		16	AAA	A	AAAA	0	11
		17	A	A	AA	1	-
		18	AA	-	-	-	10

- Tömörített szöveg: 1,2,4,6,1,3,1,10,11,10
- Ellenőrzés *(nem kitömörítés)*: A,B,AB,ABA,A,C,A,AA,AAA,AA = ABAB ABAA CAAA AAAA A
- *Megjegyzés:*
    - *Az első lépés kicsit értelemszerűen eltérő*
    - *A lépés oszlop nem vonatkozik a kódtáblára*
    - *A lépésszámokat és a beolvasott karaktereket érdemes előre felvenni a táblázatba, mivel bemenettől függően ezek előre meghatározhatók (így könnyebb nem belebonyolódni)*
    - *Gyakorlásra: https://denvaar.dev/playground/lzw_encode.html*
        - *A weboldalon már van egy alapértelmezett kódtábla, amiben az ascii karakterekhez a hozzájuk tartozó integert rendeli kódként, tehát pl. az „A” kódja 65, a „B” kódja 66, a „C” kódja 67, ...*
        - *Az új „szavak” kódja 257-től kezdődik*

Kitömörítési eljárás

1. Vegyük fel a kódtáblázatba az összes karaktert
2. Olvassunk be egy kódot
    a. Ha nem tudtunk beolvasni (elfogyott a bemenet) végeztünk
    b. Egyébként menjünk tovább
    c. Ha a beolvasott kód benne van a kódtáblázatban
        i. Vegyük ki a hozzá tartozó karakterláncot
        ii. Írjuk ki a karakterláncot
        iii. Vegyük fel a kódtáblázatba: előző karakterlánc + első karaktere a jelenlegi karakterláncnak
    d. Ha a beolvasott kód NINCS benne van a kódtáblázatban *(speciális eset)*
        i. Vegyük az előző karakterlánc + előző karakterlánc első karakterének a konkatenációját
        ii. Vegyük fel a konkatenált „szót” a szótárba
        iii. Írjuk ki a konkatenált „szót”
3. Legyen az előző karakterlánc a jelenlegi karakterlánc
4. Vissza a 2. lépésre

```
1   inicializál()
2   előző := ””
3   beolvasott := új_kód
4   if beolvasott is null then
5         exit()
6   if beolvasott in kódtáblázat then
7         jelenlegi := kódtáblázat[beolvasott]
8   else
9         jelenlegi := előző + előző[0]
10  kiír(jelenlegi)
11  kódtáblázat.add(previous + jelenlegi[0])
12  előző := jelenlegi
```

Példa

Kitömörítendő szöveg: 1,2,4,6,1,3,1,10,11,10

karakter / szó	kód	lépés	beolvasott	előző	jelenlegi	új szó	output
A	1	1	1	-	A	-	A
B	2	2	2	A	B	AB	B
C	3	3	4	B	AB	BA	AB
AB	4	4	6	AB	ABA	ABA	ABA
BA	5	5	1	ABA	A	ABAA	A
ABA	6	6	3	A	C	AC	C
ABAA	7	7	1	C	A	CA	A
AC	8	8	10	A	AA	AA	AA
CA	9	9	11	AA	AAA	AAA	AAA
AA	10	10	10	AAA	AA	AAAA	AA
AAA	11						
AAAA	12						

- Output: A B AB ABA A C A AA AAA AA = ABAB ABAA CAAA AAAA A
- *Megjegyzés:*
    - *Az első lépés kicsit értelemszerűen eltérő*
    - *A lépés oszlop nem vonatkozik a kódtáblára*
    - *A lépésszámokat és a beolvasott karaktereket érdemes előre felvenni a táblázatba, mivel bemenettől függően ezek előre meghatározhatók (így könnyebb nem belebonyolódni)*
    - *Gyakorlásra:* https://denvaar.dev/playground/lzw_decode.html
        - *A weboldalon már van egy alapértelmezett kódtábla, amiben az ascii karakterekhez a hozzájuk tartozó integert rendeli kódként, tehát pl. az „A” kódja 65, a „B” kódja 66, a „C” kódja 67, ...*
        - *Az új „szavak” kódja 257-től kezdődik*

Előnye

- Nem kell előre ismerni a szöveg statisztikáját, menet közben építi a szótárat
- Általánosan jó tömörítési arány, különösen akkor, ha sok ismétlődés vagy minta van a szövegben
- Veszteségmentes tömörítés

Hátránya

- rövid vagy nagyon változatos adatoknál a tömörítés akár nagyobb is lehet, mint az eredeti
- ha nincs korlátozva, a szótár túl nagyra duzzadhat; ha korlátozva van, akkor kezelni kell a túlcsordulást

AVL fák

Az AVL fa egy speciális bináris keresőfa (Binary Search Tree, BST), amelyet önkiegyensúlyozó bináris keresőfának is nevezünk. Lényege, hogy a beszúrások és törlések után automatikusan biztosítja, hogy a fa magassága a lehető legkisebb maradjon.

Az alapelv:

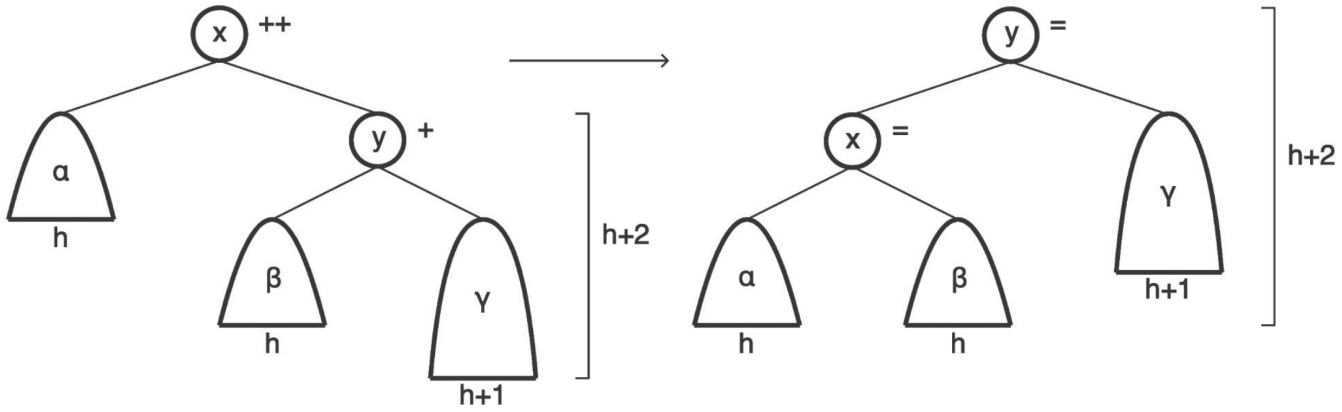
- Ha ez a különbség mindig –1, 0 vagy +1, akkor a fa kiegyensúlyozott
- Ha ennél nagyobb eltérés keletkezne, akkor a fa rotációkkal kiegyenlíti magát

Ennek köszönhetően az AVL fa garantálja, hogy:

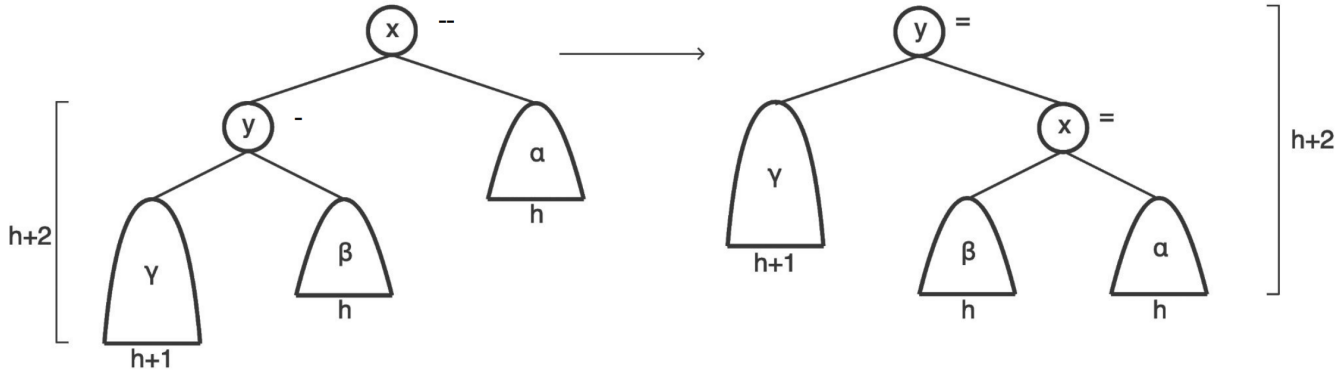
- a keresés, beszúrás és törlés műveletek O(log n) időben futnak
- a fa magassága mindig logaritmikus marad a csomópontok számához képest

Forgatások

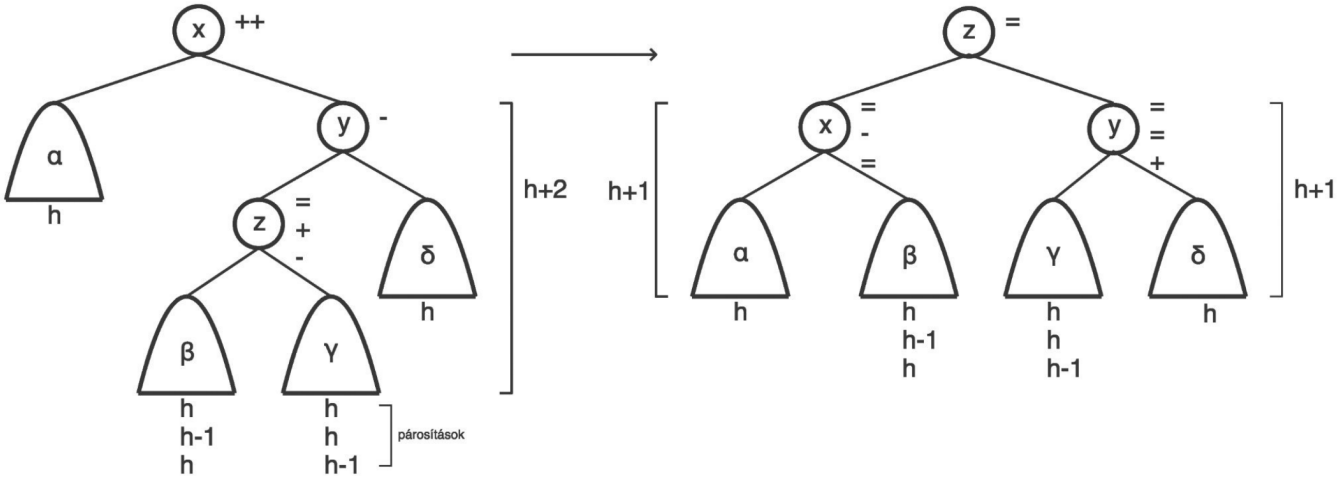
(++, +)



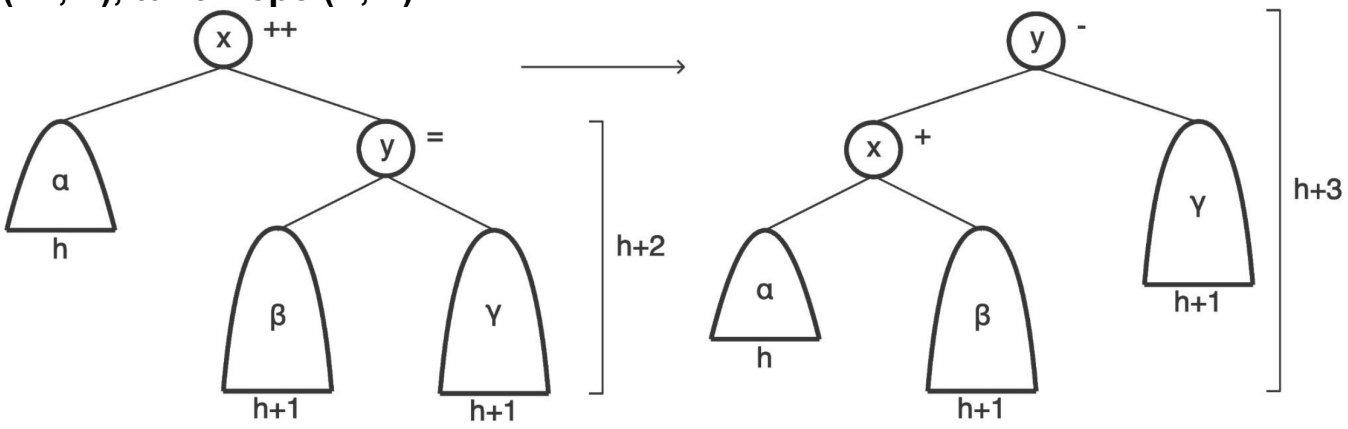
(--, -)



(++, -), tükörképe (--, +)



(++, =), tükörképe (--, =)



Beszúrás

1. Megkeressük a kulcs helyét a keresőfa szabály szerint
2. Ha ott üres részfa van, új levélként beszúrjuk
3. Felfelé haladva minden csomópontnál:
    a. frissítjük az egyensúlyt
    b. ha az új érték 0 → megállhatunk (nem nőtt a magasság)
    c. ha az új érték ±1 → tovább kell menni (a részfa magasabb lett)
    d. ha az új érték ±2 → rotációval kiegyensúlyozzuk, és kész vagyunk

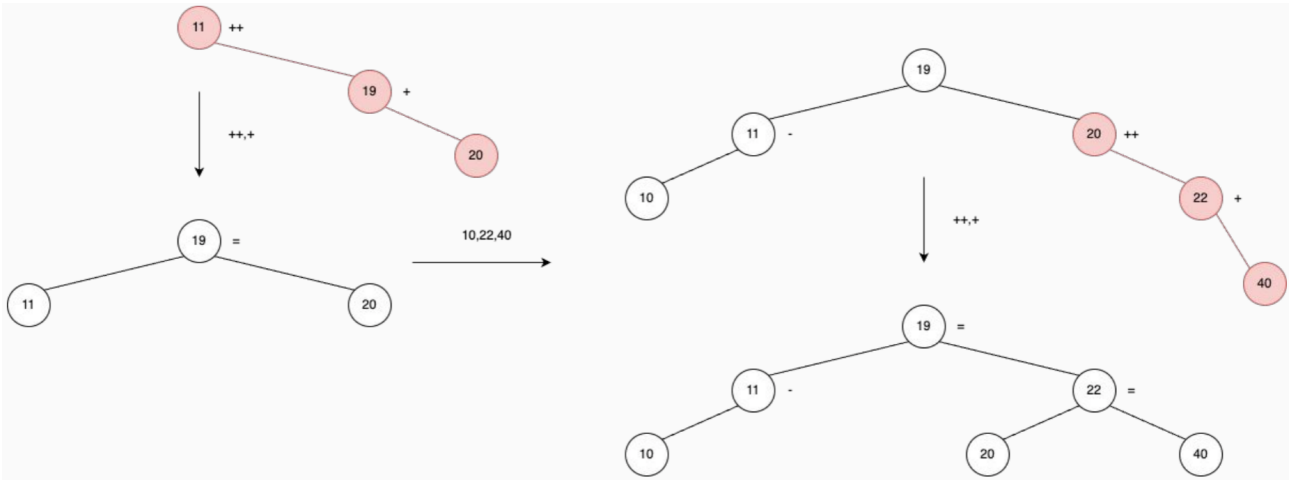
Röviden: beszúrás után alulról felfelé frissítjük a magasságokat és balance faktorokat, és ha kell, egy forgatással helyreállítjuk az egyensúlyt.

Törlés

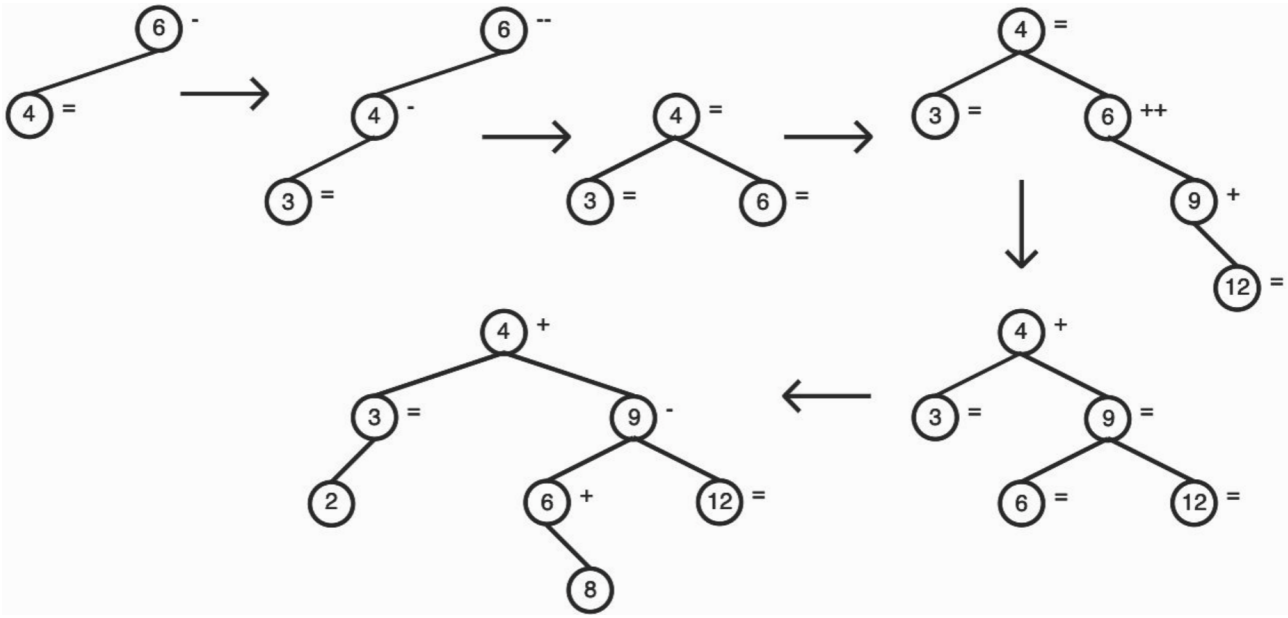
1. Megkeressük a törlendő elemet a keresőfa szabály szerint.
2. Három eset lehet:
    a. Levél vagy egygyerekes csúcs: egyszerűen eltávolítjuk, a másik részfát (ha van) láncoljuk a helyére
    b. Kétgyerekes csúcs: megkeressük a jobb részfa minimumát (vagy bal részfa maximumát), azt kivesszük, és az új helyére tesszük (Ez biztosítja, hogy a keresőfa tulajdonság megmaradjon)
3. Az eltávolítás miatt a részfa magassága csökkenhet, ezért felfelé haladva:
    a. frissítjük az egyensúlyt
    b. ha az érték ±2 → rotációval kiegyensúlyozzuk
4. a kiegyensúlyozás után is folytatjuk a szülők felé, mert törlésnél több szinten is szükség lehet rotációra (akár a gyökérig)

Példa

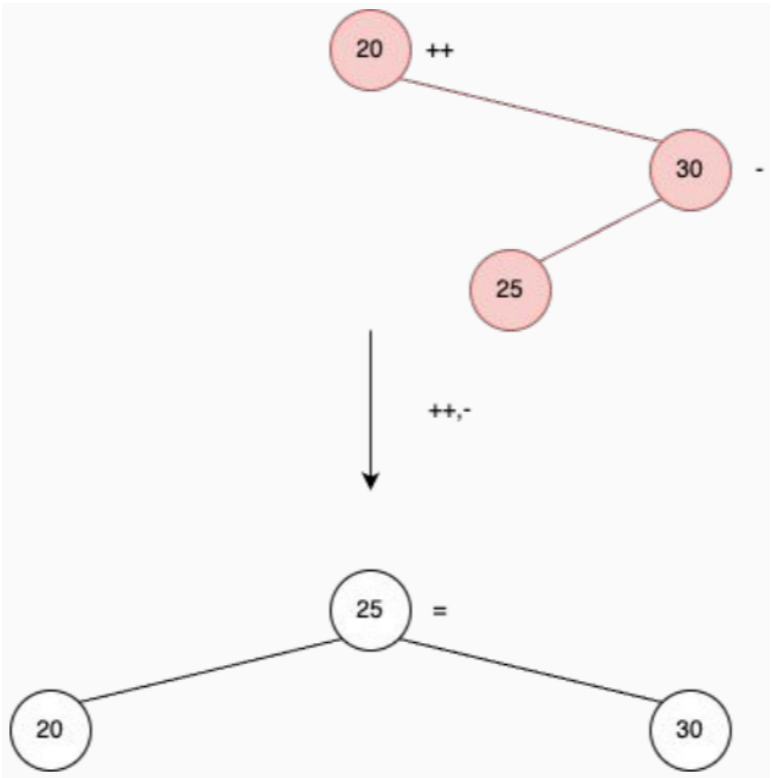
Építsünk fát a következő adatokból: 11, 19, 20, 10, 22, 40



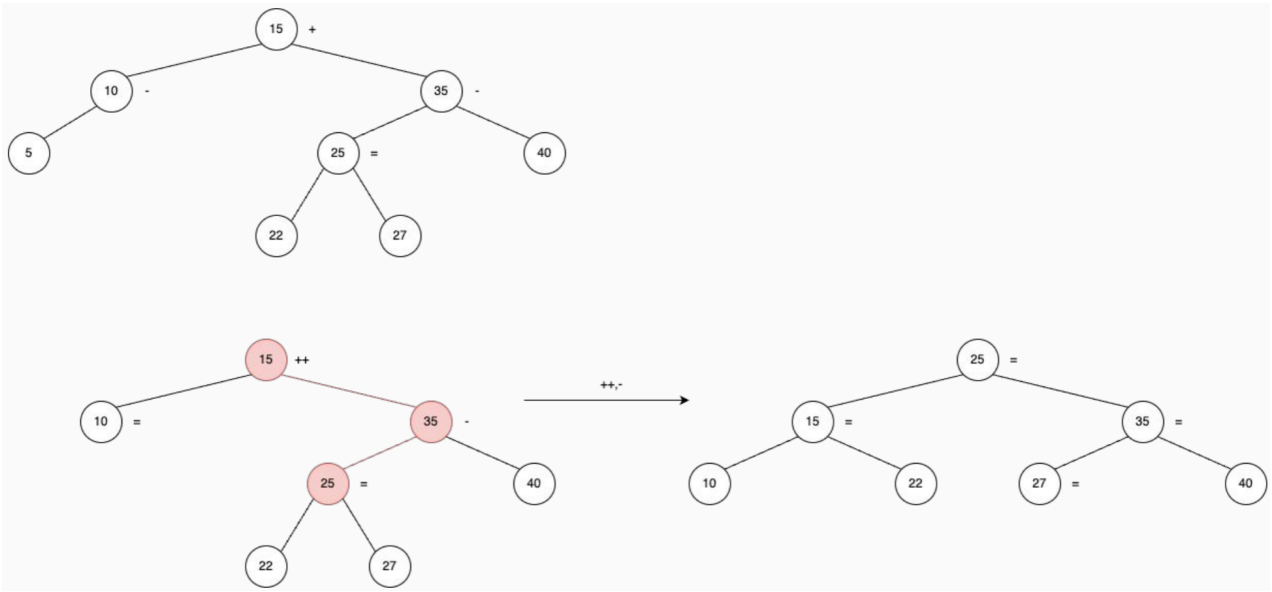
Építsünk fát a következő adatokból: 6, 4, 3, 9, 12, 2, 8



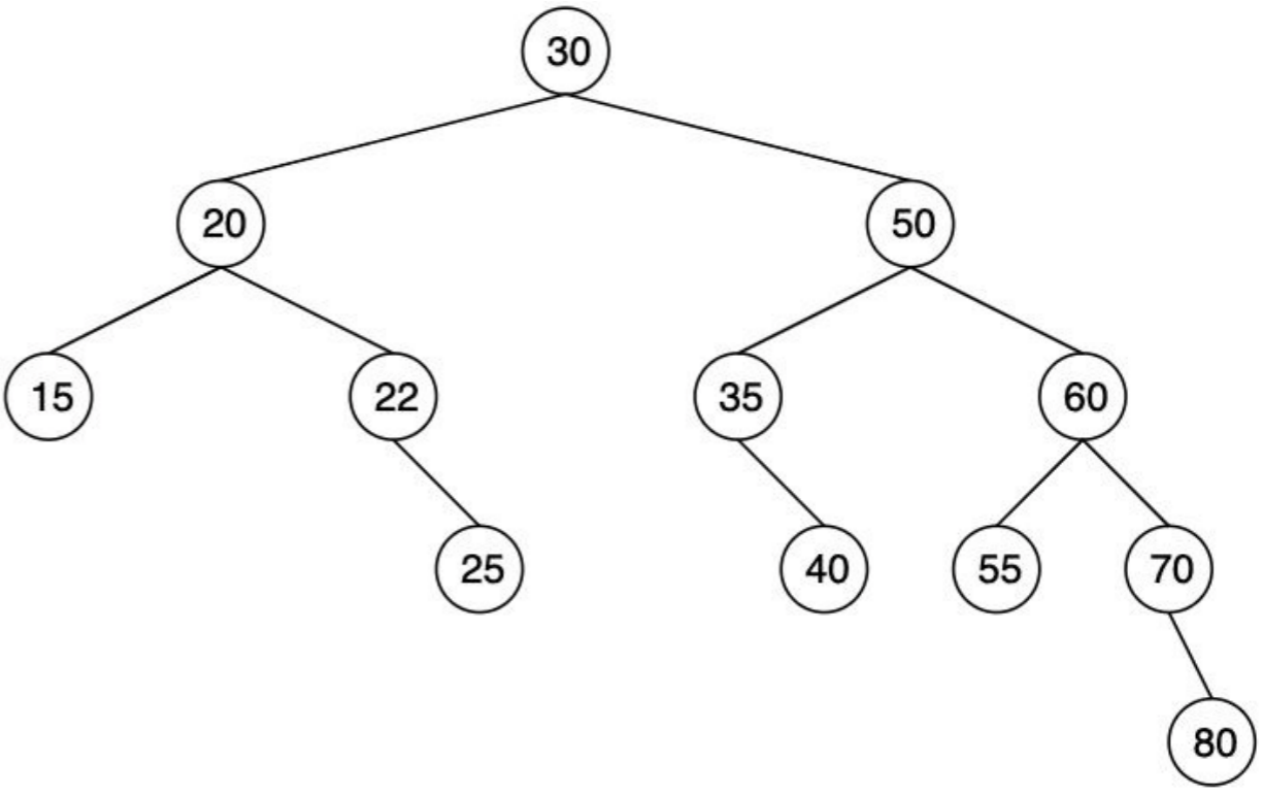
Építsünk fát a következő adatokból: 20, 30, 25

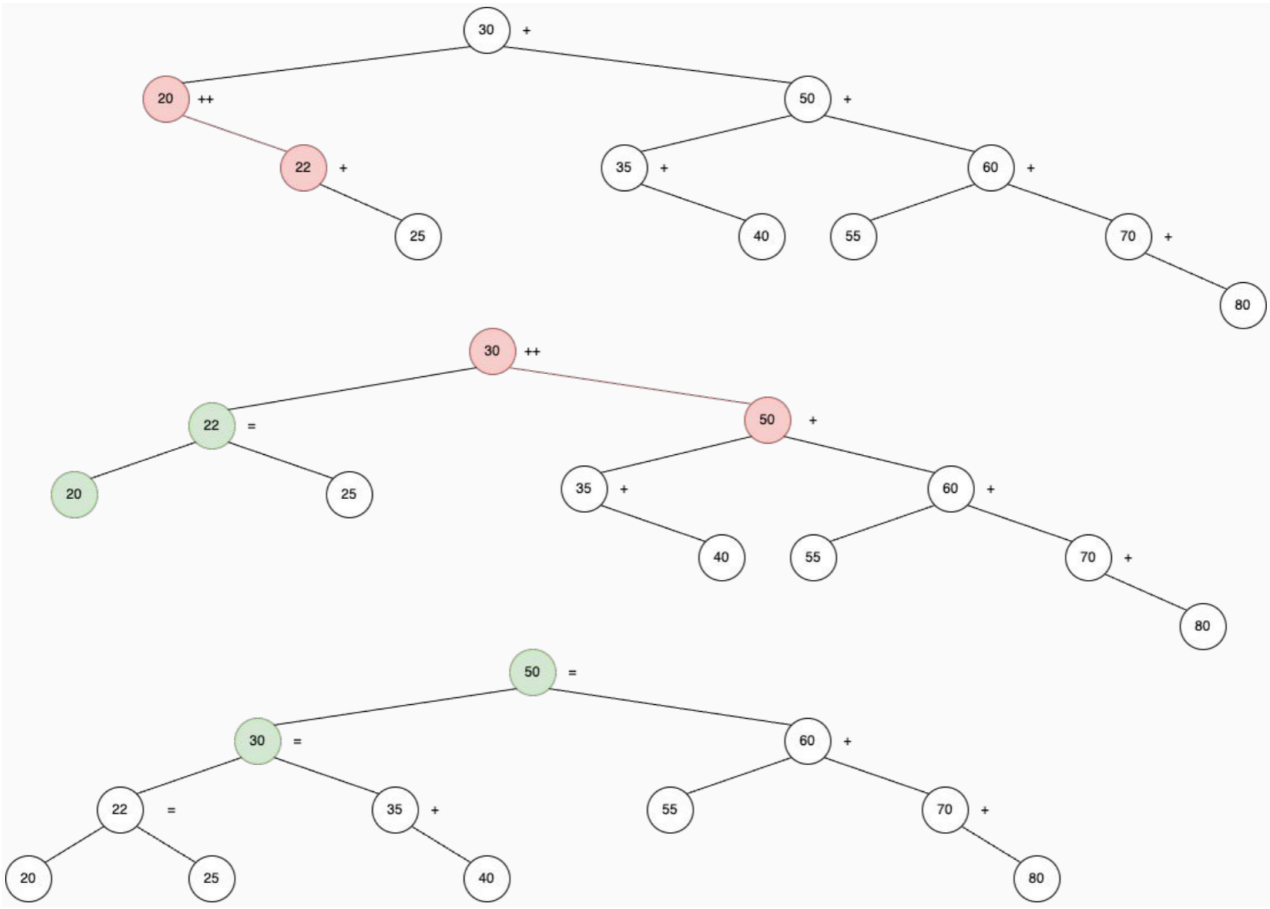


Töröljük az alábbi fából az 5-ös elemet

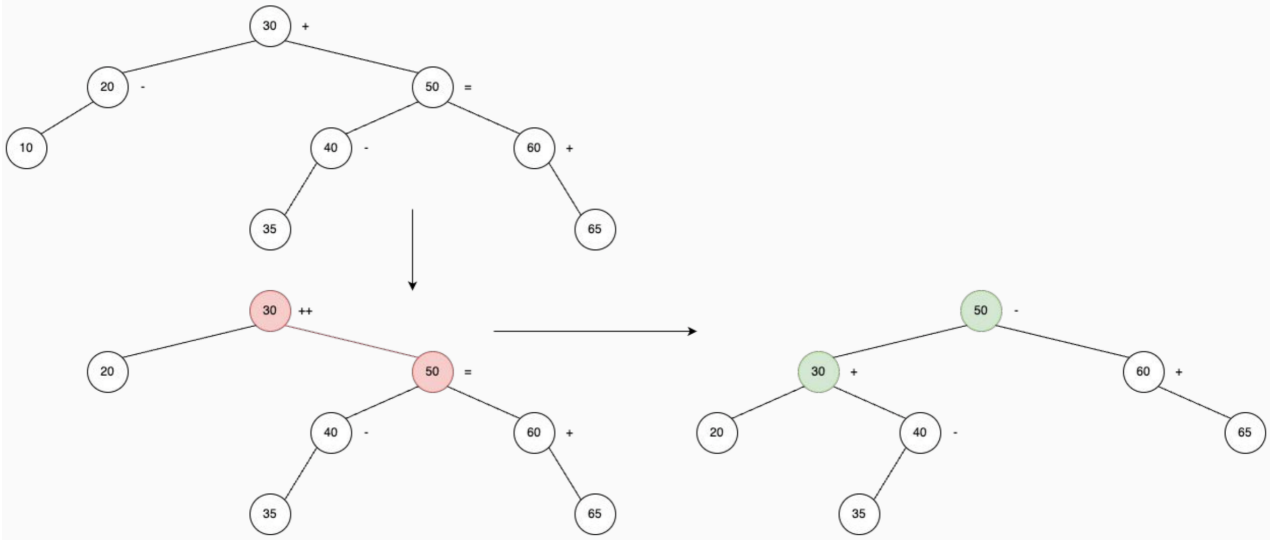


Töröljük az alábbi fából a 15-ös elemet





Töröljük az alábbi fából a 10-es elemet



Gyakorlásra: https://www.cs.usfca.edu/~galles/visualization/AVLtree.html (a jelölés picit más, de a beszúrás, törlés, és forgatások ugyanaz)

Szorgalmi feladatok

1. Tömörítsd a Naiv módszerrel az alábbi szöveget. (1 pont)
   - HUMBABUMBLAKUMPALUMPABUUUUU

   Továbbá add meg a
   - Kódtáblát
   - Kódtábla méretét
   - Tömörített szöveg méretét
   - A szöveg eredeti méretét

2. Tömörítsd a Huffman módszerrel az alábbi szöveget. (1 pont)
   - HUMBABUMBLAKUMPALUMPABUUUUU

   Továbbá add meg a
   - Kódfát
   - Kódtáblát
   - Kódtábla méretét
   - Tömörített szöveg méretét
   - A szöveg eredeti méretét

3. Miért működik úgy az LZW speciális esete ahogy? (1 pont)

4. Tömörítsd az LZW módszerrel az „XXYXXXYXXYXYZYYZZZZZXZY” szöveget. (1 pont)

5. Építs AVL fát a következő elemekből: 20, 10, 80, 25, 50, 40, 70, 60, 90, 30. Töröld a következő elemeket az előbbi fából: 10, 25, 20, 30, 70 (2 pont)