Mérnöki modellalkotás
Az elmélettől a gyakorlatig

IP forgalomtovábbítás:
Prefix fák és fabejárások

Tartalom

- IP címzés és forgalomtovábbítás
- Legspecifikusabb bejegyzés keresése (LPM)
- LPM prefix fákkal, prefix fák transzformációja és ekvivalenciája
- Fabejárások: preorder-postorder

IP címzés és forgalomtovábbítás

IPv4: címzés

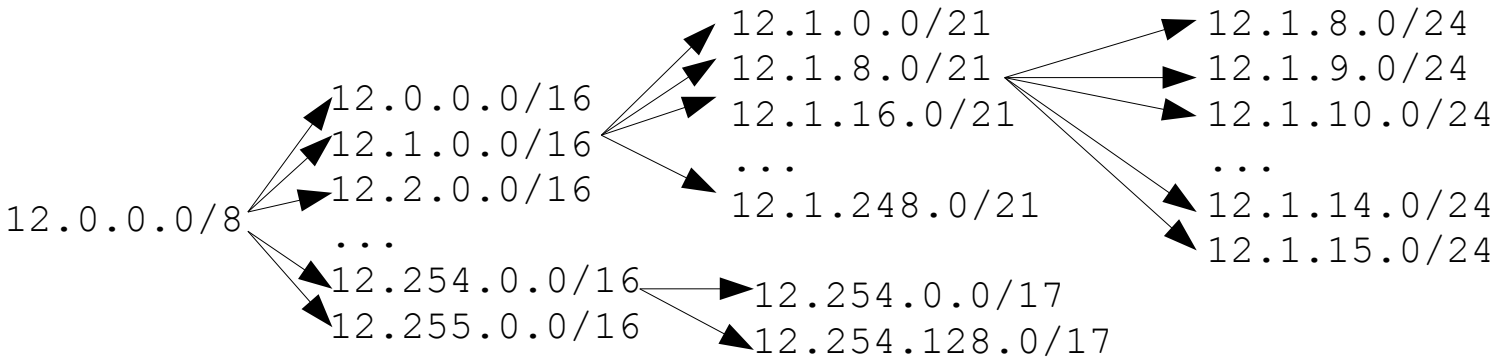
- IPv4 cím: 32 bit unsigned integer, 4294967296 (2^{32}) darab egyedi azonosító
- De pl. a `2554524783` cím nehezen olvasható
  - decimális jelölés: `2554524783`
  - bináris:`10011000 01000010 11110100 01101111`
  - „dotted decimal”: 8 bitenként feltördelve négy 1 byte-os számra: `152.66.244.111`

152	66	244	111
10011000	01000010	11110100	01101111
2554524783			

IPv4: alhálózatok

IPv4 cím = alhálózat-azonosító + hoszt-azonosító

- IP címek csoportjai **alhálózati prefixbe (subnet)** gyűjthetőek
- Az alhálózat egyetlen IP prefixen „látszik” az Internetről (aggregáció)
- Az alhálózat-azonosító hossza: **prefix hossz**



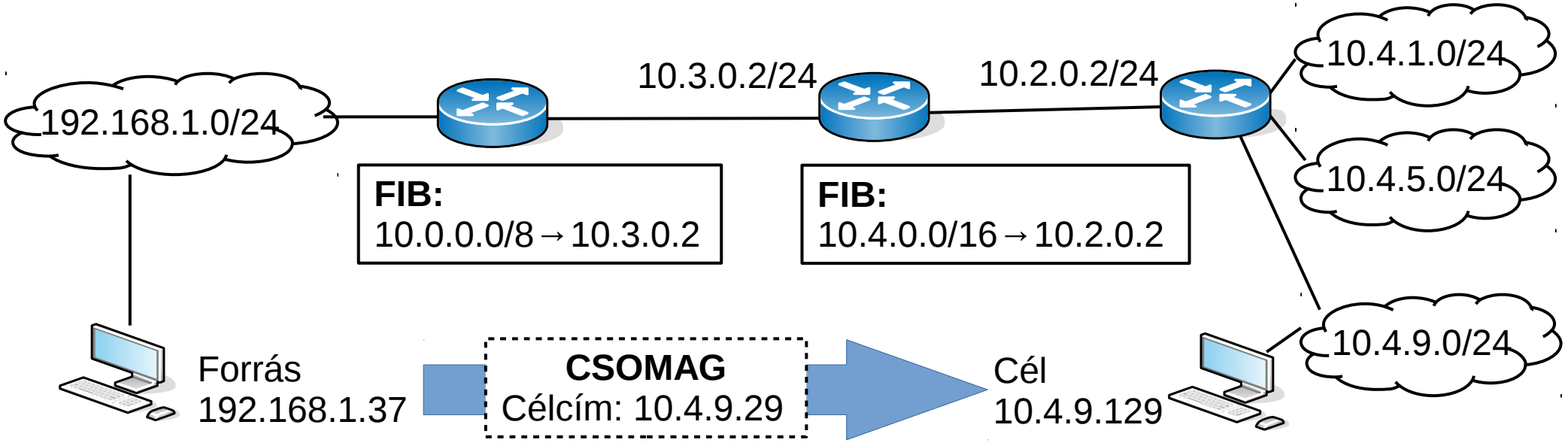
IPv4 alhálózatok: CIDR

- CIDR: rugalmasan kialakítható alhálózatok
- Variable Length Subnet Masking (VLSM)

CIDR notation	192.168.192.0/18
Prefix hossza	18 bit (az MSB-től)
bináris	11000000 10101000 11000000 00000000
Subnet mask (bináris)	11111111 11111111 11000000 00000000
Subnet mask (dotted)	255.255.192.0
Egyedi IP címek száma	$2^{32-18}=2^{14}=16384$ (valójában 2-vel kevesebb, a tartomány első és az utolsó IP címe nem használt)
Első IP cím	192.168.192.1
bináris	11000000 10101000 11000000 00000001
Utolsó IP cím	192.168.255.254
bináris	11000000 10101000 11111111 11111110

IPv4 forgalomtovábbítás

- Minden csomag tartalmazza a cél IP címét
- Forgalomtovábbítási tábla (Forwarding Information Base, FIB) minden routerben
  - minden ismert prefixre egy bejegyzés
  - az útvonalon következő router címe (next-hop)



A legspecifikusabb prefix

- Adott IP címre több FIB-bejegyzés illeszkedhet

Egy router FIB-jének részlete		
IP prefix/prefix hossz	**A prefix binárisan**	**Next-hop IP címe**
192.168.0.0/16	11000000 10101000	10.0.0.1
192.168.0.0/17	11000000 10101000 0	10.0.0.2
192.168.64.0/18	11000000 10101000 01	10.0.0.3
192.168.96.0/19	11000000 10101000 011	10.0.0.4

- **Longest Prefix Match (LPM):** preferált a legspecifikusabb bejegyzés
  - a legtöbb biten illeszkedő prefix (MSB-től szamítva)
  - a legszűkebb alhálózat, ami tartalmazza a címet

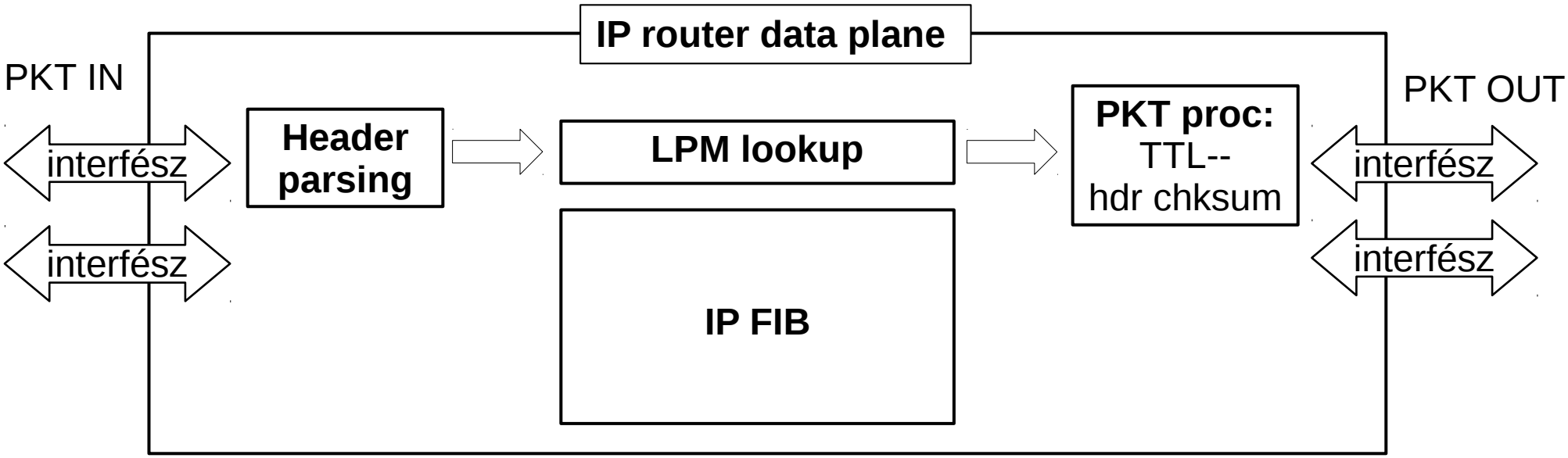
LPM: példa

Egy router FIB-jének részlete		
IP prefix/prefix hossz	**A prefix binárisan**	**Next-hop IP címe**
192.168.0.0/16	11000000 10101000	10.0.0.1
192.168.0.0/17	11000000 10101000 0	10.0.0.2
192.168.64.0/18	11000000 10101000 01	10.0.0.3
192.168.96.0/19	11000000 10101000 011	10.0.0.4

- A 192.168.1.1=x.x.000000001.000000001 címre az első két bejegyzés illik, a 3. és 4. a pirossal jelzett pozíciókban eltér: a 2. preferált
- A 192.168.95.2=x.x.010111111.000000010 címre a 3. bejegyzés, a 192.168.97.3=x.x.011000001.000000011 címre a 4. a LPM

A LPM jelentősége

- Az IP csomagtovábbítás legbonyolultabb része
- IP core router: LPM több mint 550 ezer FIB bejegyzés felett **minden csomagra** (40 Gbit/sec ~ 50 millió LPM/sec)



LPM prefix fákkal

FIB keresés: LPM

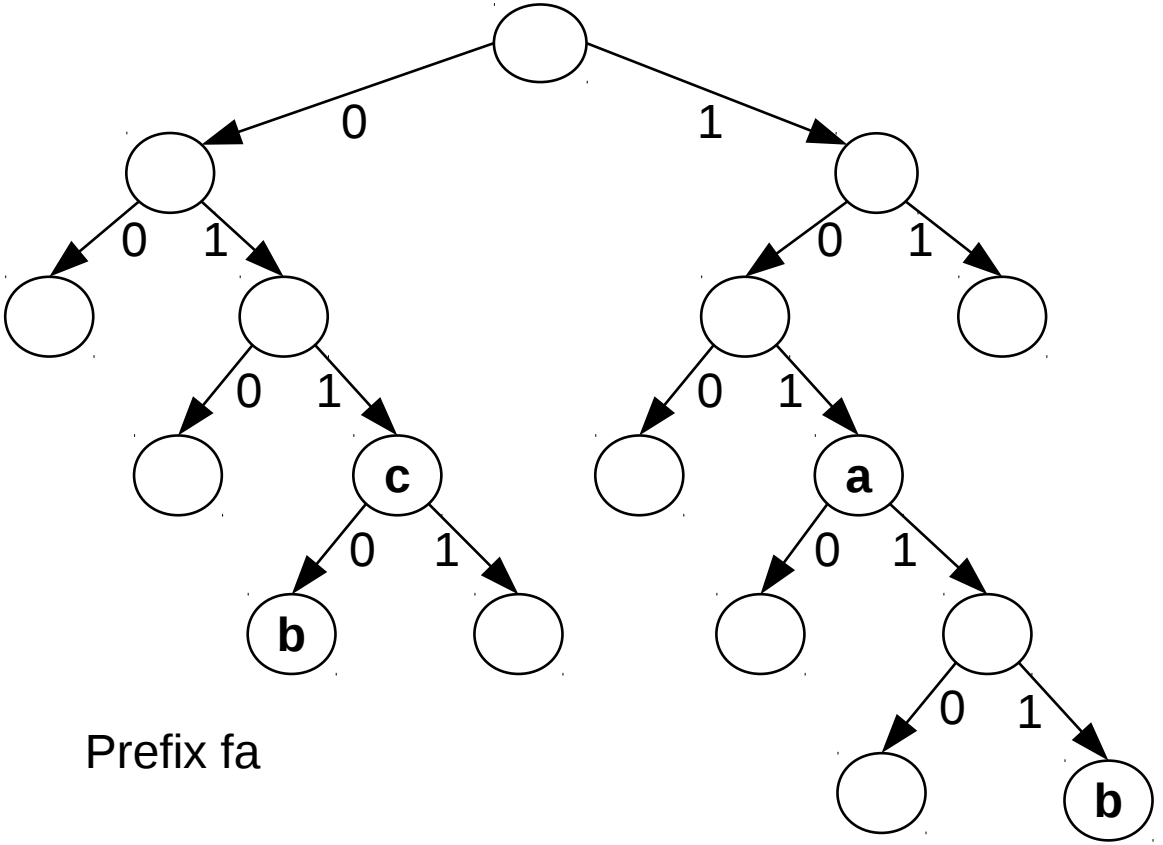
- A LPM megvalósítása nem triviális
- Naív megközelítés: végigkeresni az összes bejegyzést a FIBben és megjegyezni a legtöbb biten illeszkedőt
- *O(N)* komplexitás, ha a bejegyzések száma *N*
- A naiv módszer használhatatlan

A (bináris) prefix fa

- LPM keresésre optimalizált adatstruktúra
- A **prefix fa** a következő műveleteket támogatja:
  - **keresés:** adott mintára legtöbb biten illeszkedő prefix megtalálása és a hozzá tárolt címke kiolvasása
  - **beszúrás:** (prefix → címke) páros beillesztése
  - **törlés:** prefix és hozzá tartozó címke törlése
  - **módosítás:** prefix címkéjének módosítása

A prefix fa

- A next-hopokat azonosítsuk **egyedi címkékkel** és tároljuk külön egy **next-hop index** táblában



Prefix fa

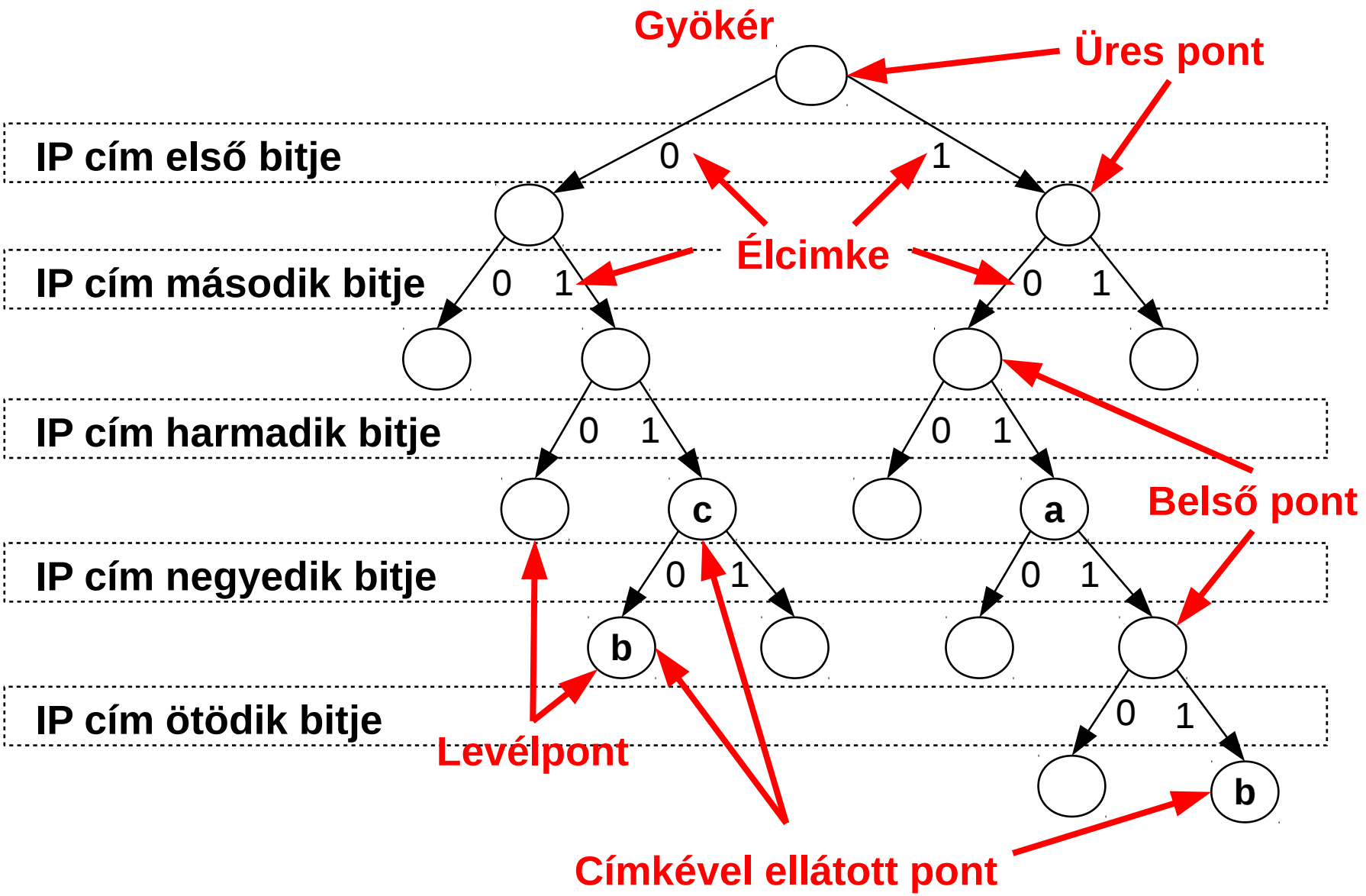
FIB

IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

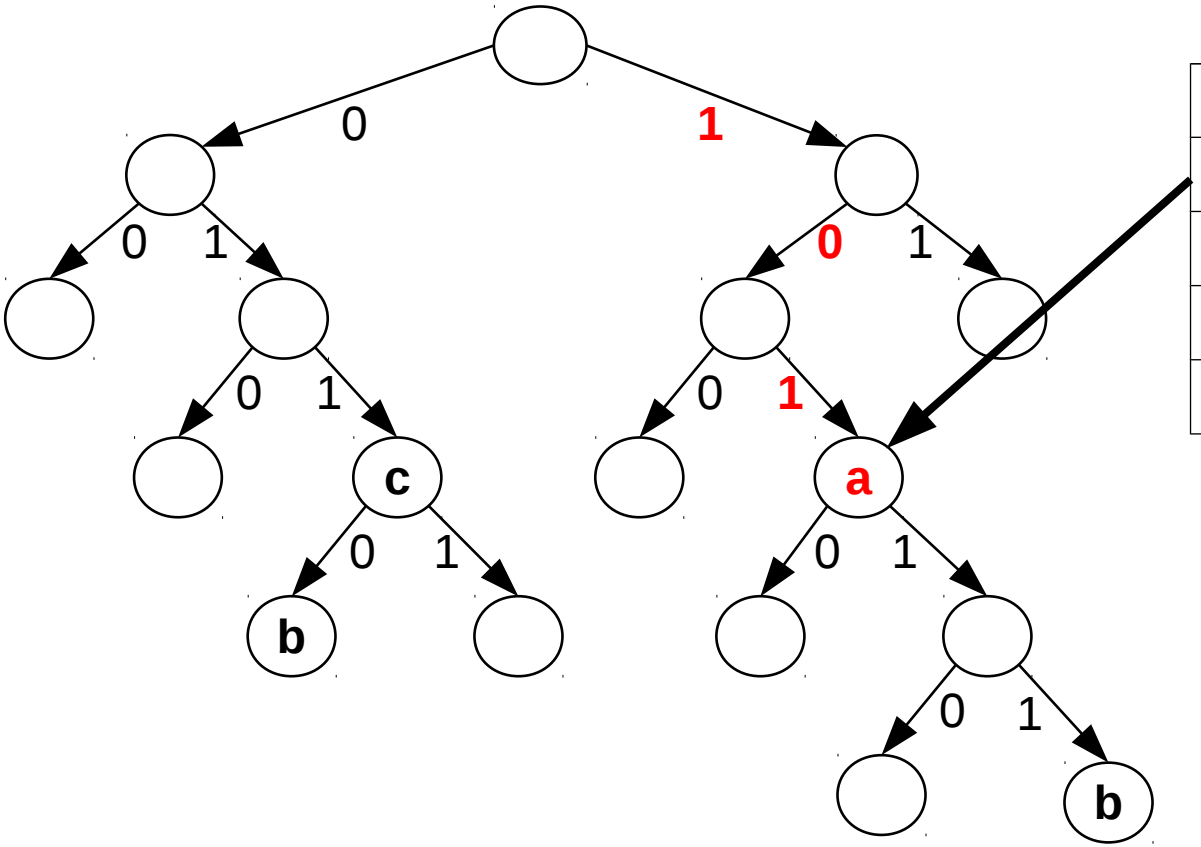
Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa


Gyökér
Üres pont
IP cím első bitje
0
1
Élcimke
IP cím második bitje
0
1
0
1
IP cím harmadik bitje
0
1
0
1
c
a
Belső pont
IP cím negyedik bitje
0
1
0
1
b
IP cím ötödik bitje
0
1
Levélpont
b
Címkével ellátott pont

A prefix fa

- **Prefix=pontba vezető út élcímkéinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük



FIB

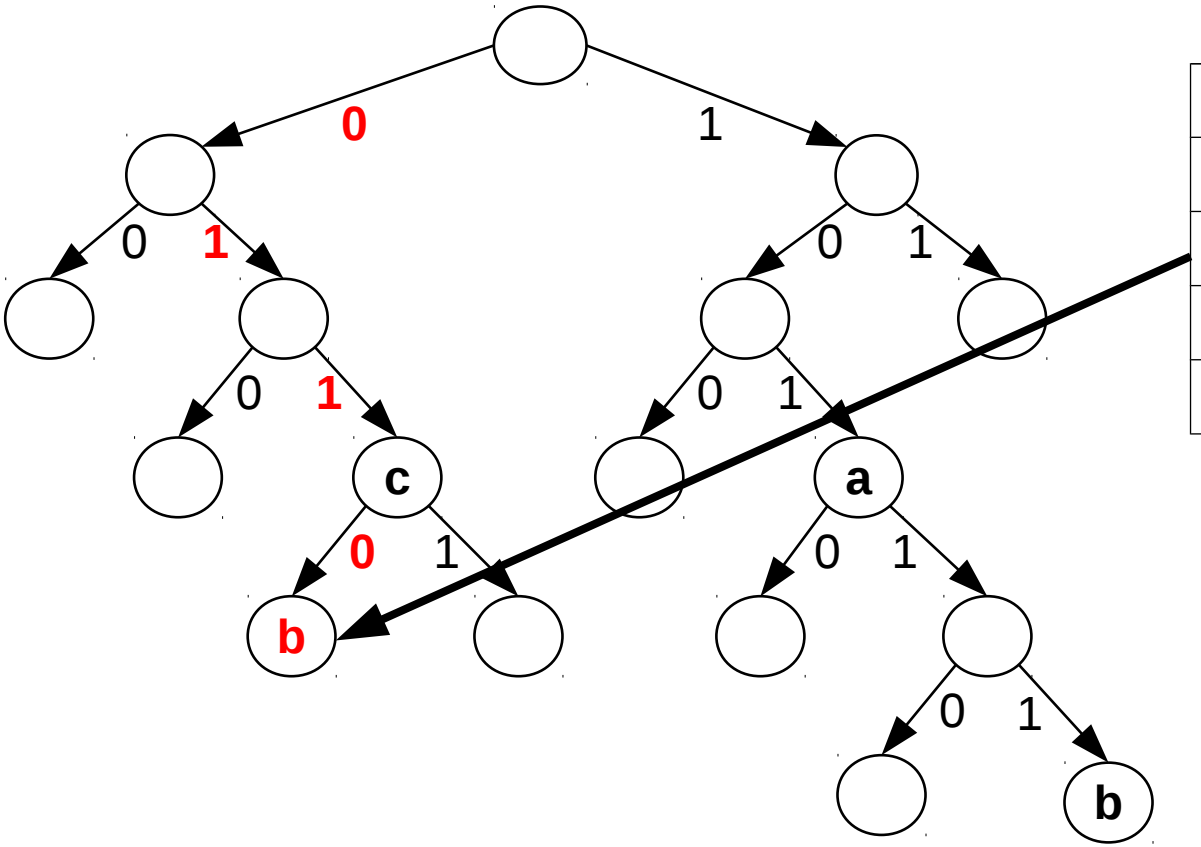
IP prefix	Prefix	Címke
160.0.0.0/3	**101**	**a**
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa

- **Prefix=pontba vezető út élcímkéinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük



FIB

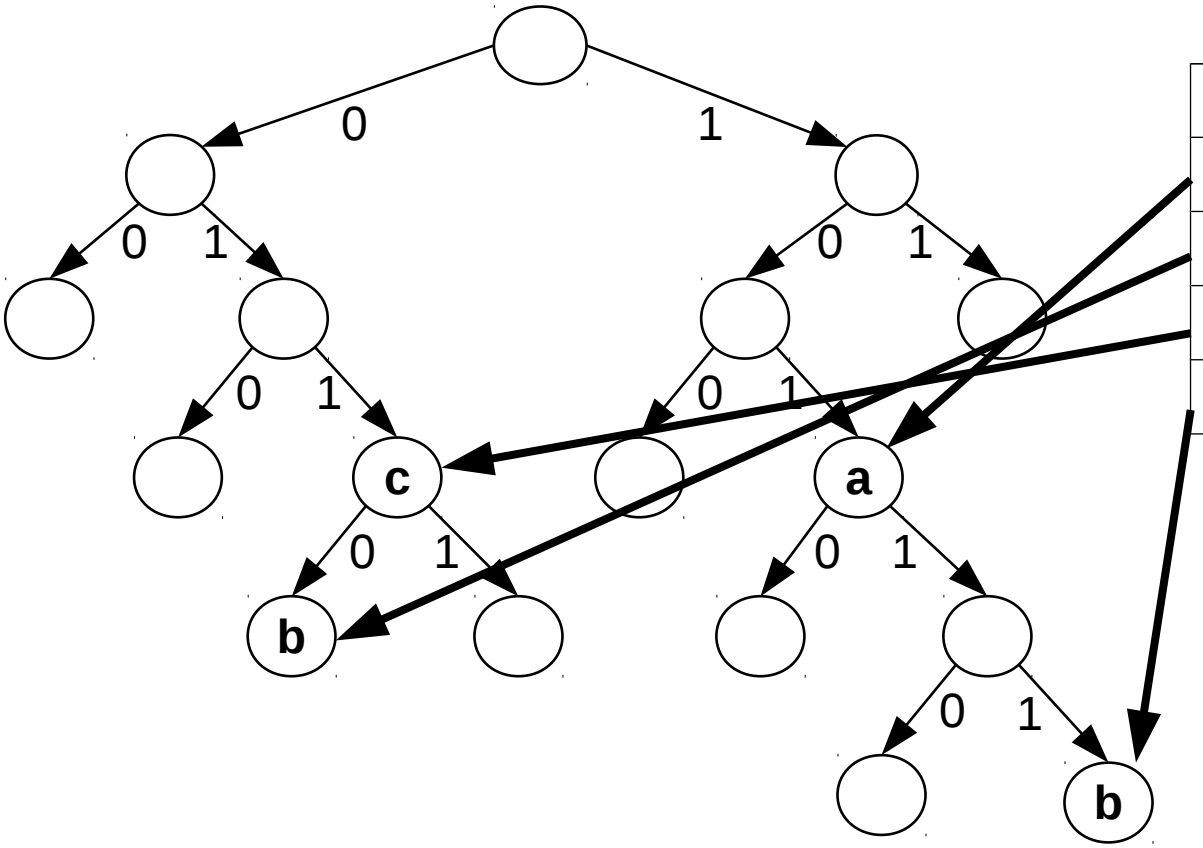
IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	**0110**	**b**
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa

- **Prefix=pontba vezető út élcímkéinek sorozata**
- A fában a prefixeknek megfelelő pontokat a prefixhez tartozó next-hop címkével jelöljük



FIB

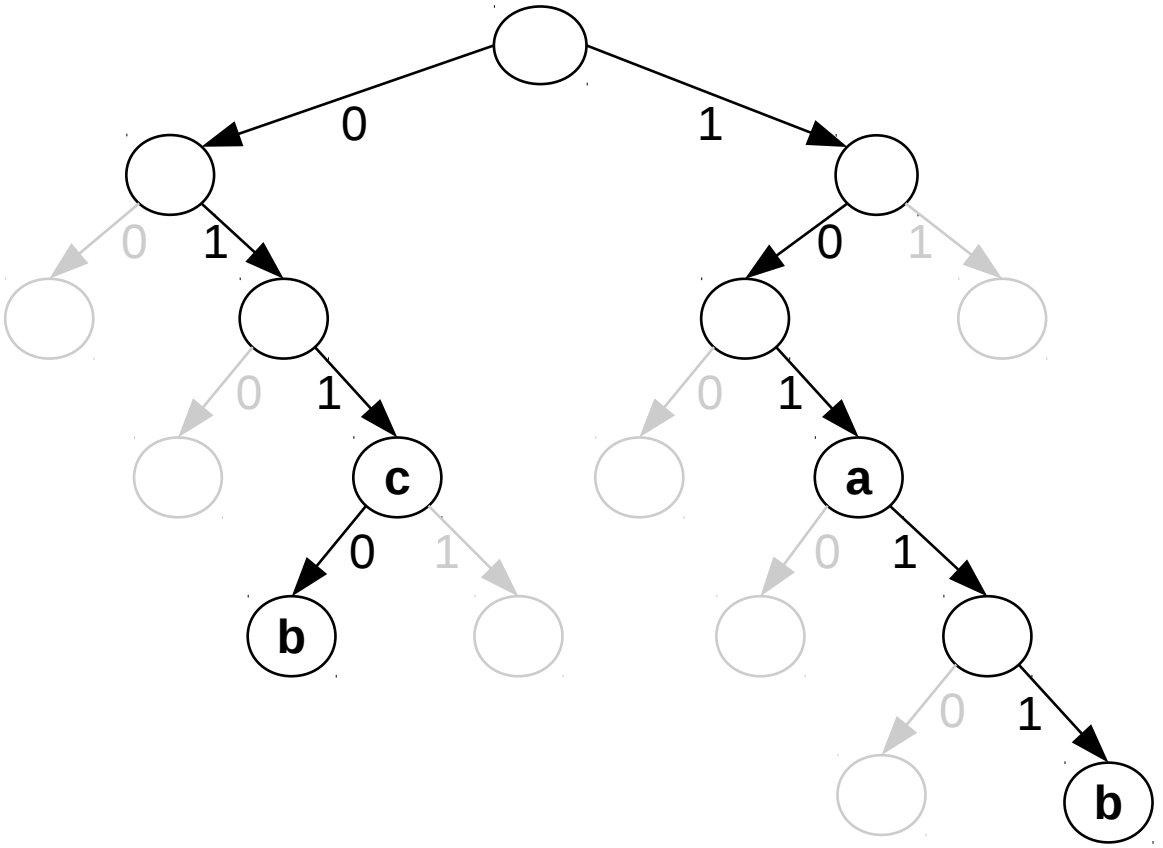
IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa

- Az üres levélpontokat elhagyhatjuk, az üres pontokba mutató pointerek: NULL
- Kisebb fa, kevesebb memória



FIB

IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

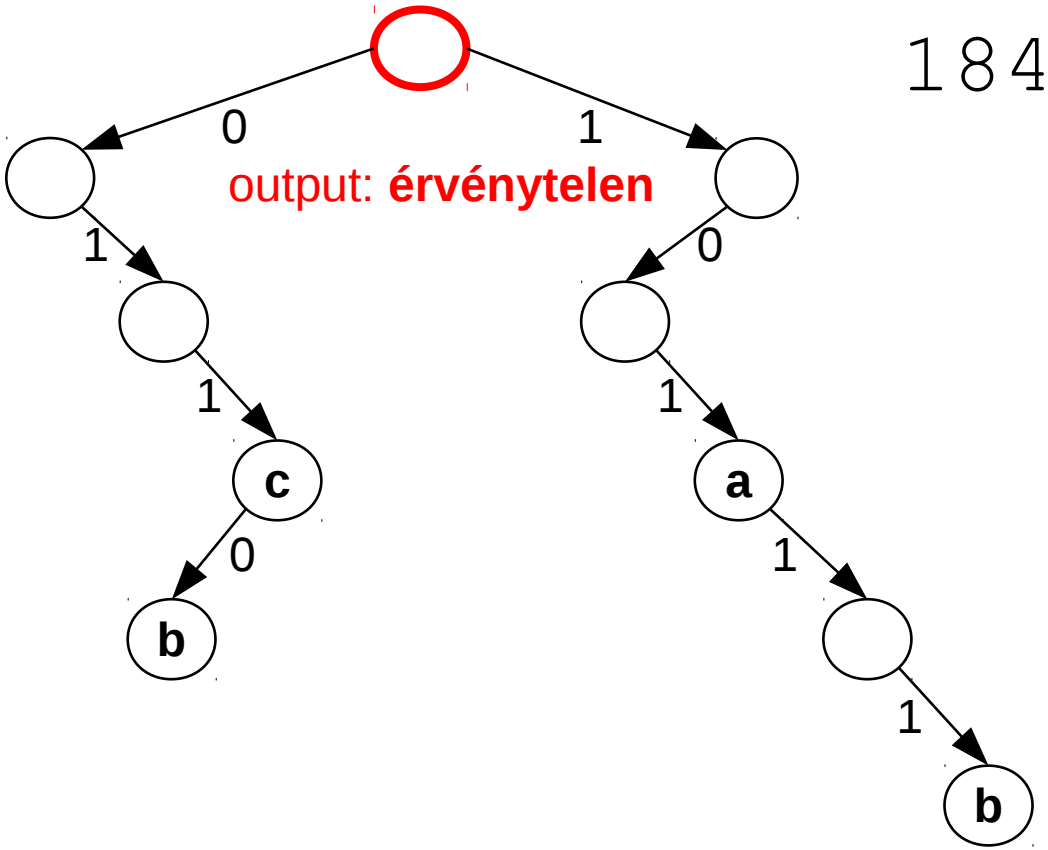
Next-hop index

Címke	Next-hop
a	10.0.0.1
b	10.0.0.2
c	10.0.0.3

A prefix fa: keresés

- Keressük a `184.1.1.1=10111...` IP címre a legspecifikusabb találatot a prefix fában
- Start a **gyökérpontból**, **`output←érvénytelen`**

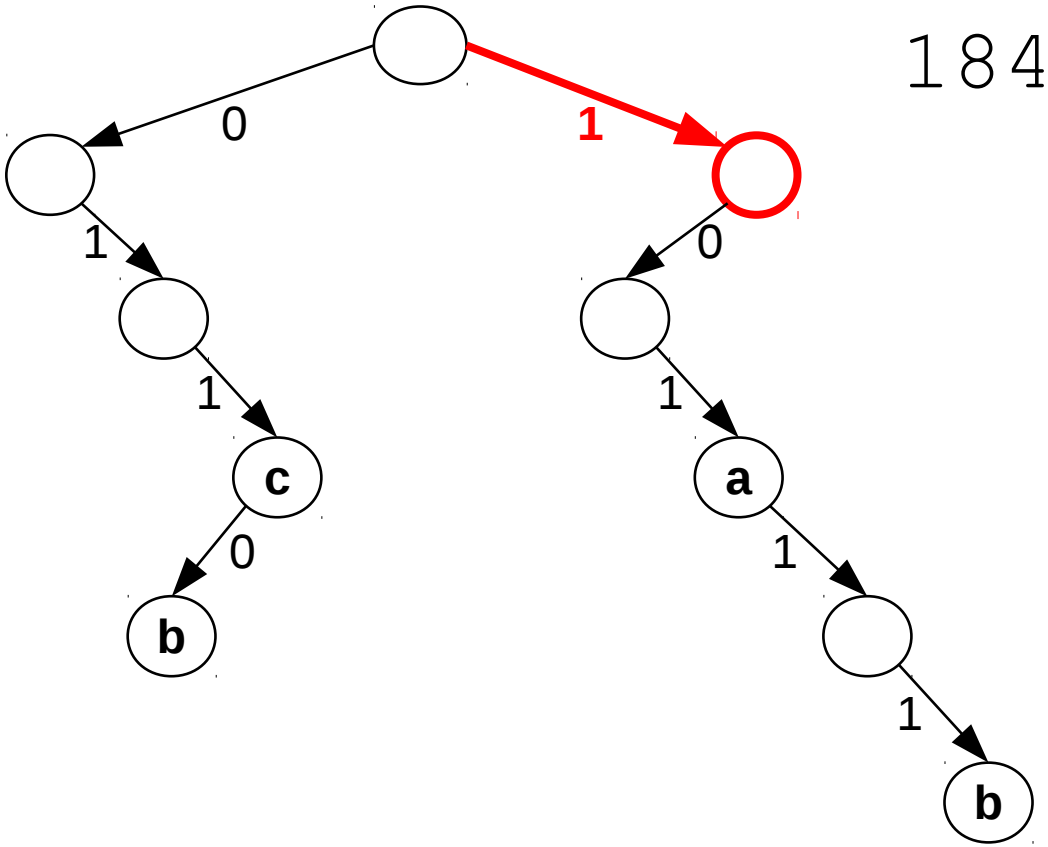
`184.1.1.1=10111...`



IP prefix	Prefix	Címke
`160.0.0.0/3`	`101`	`a`
`96.0.0.0/4`	`0110`	`b`
`96.0.0.0/3`	`011`	`c`
`184.0.0.0/5`	`10111`	`b`

A prefix fa: keresés

- A `184.1.1.1=10111...` keresési cím első bitje `1` értékű, így a gyökérből az `1`-es élcímkével ellátott élen lépünk a **következő pontba**

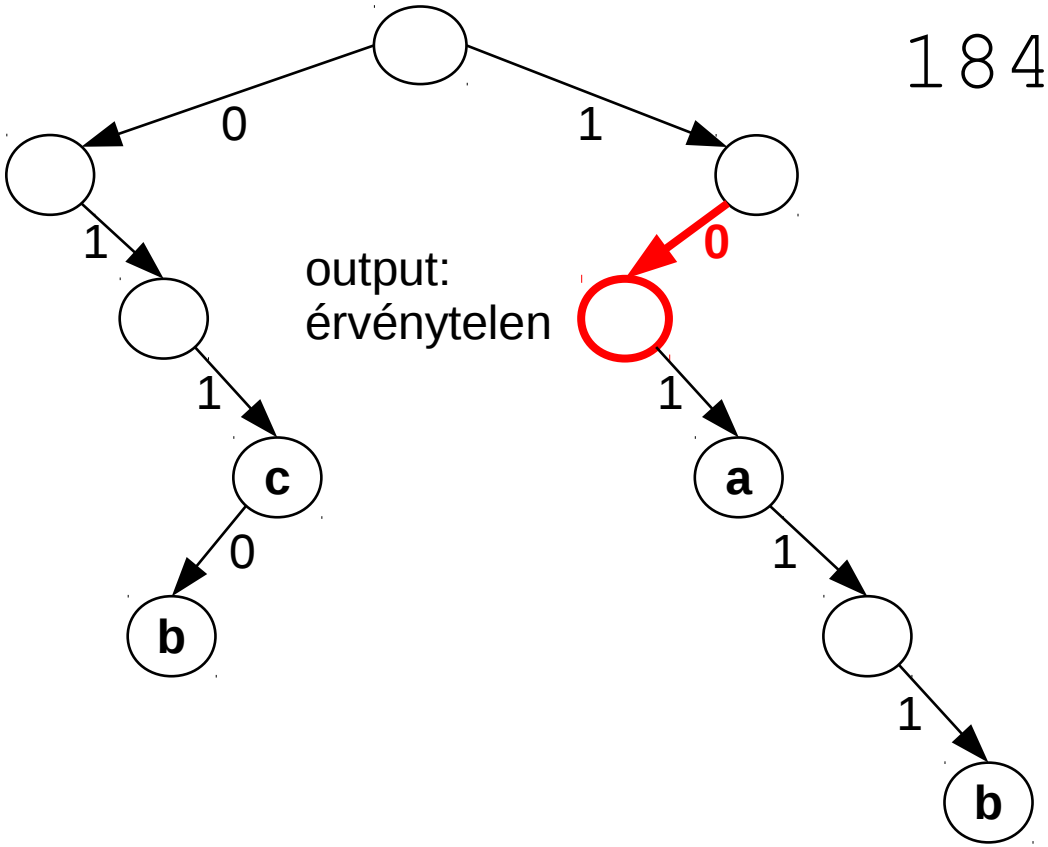


`184.1.1.1=`**`1`**`0111...`

IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

A prefix fa: keresés

- Az új pontban nincs címke, output változatlan
- A második bit `0`, így az `0` élcímkén haladunk tovább a **következő pontba**



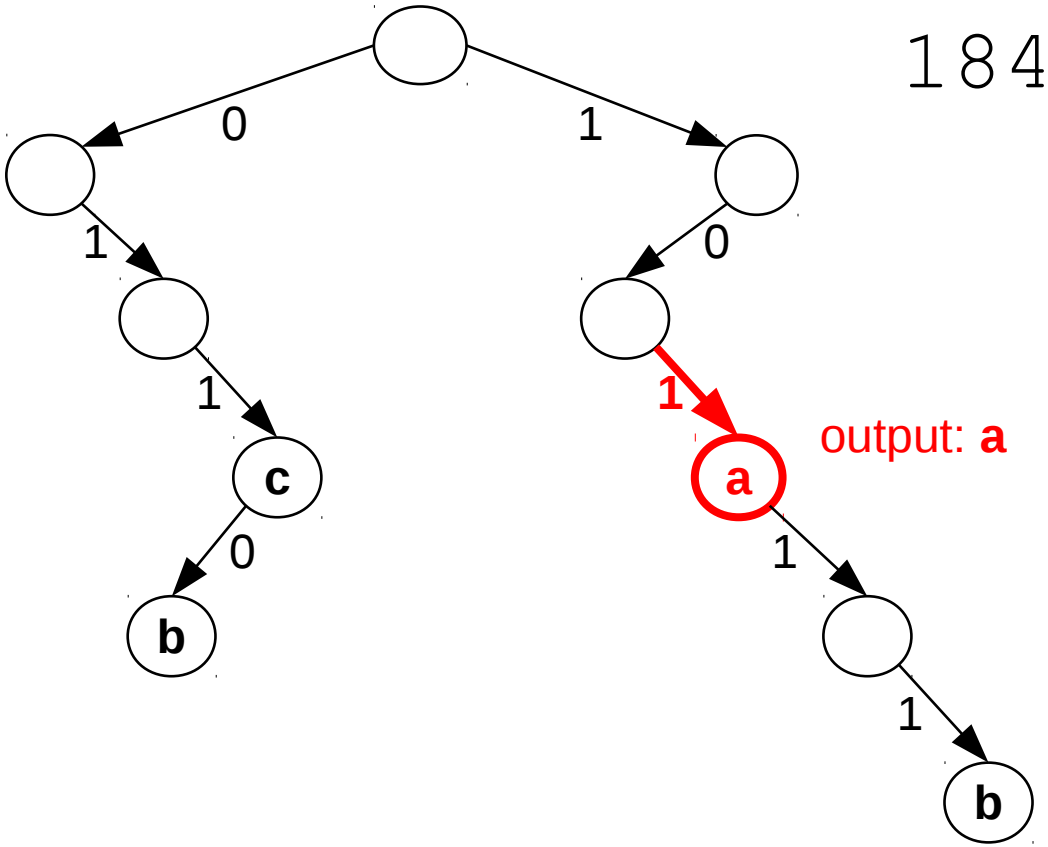
`184.1.1.1=10111...`

IP prefix	Prefix	Címke
`160.0.0.0/3`	`101`	`a`
`96.0.0.0/4`	`0110`	`b`
`96.0.0.0/3`	`011`	`c`
`184.0.0.0/5`	`10111`	`b`

A prefix fa: keresés

- A harmadik bit ismét `1`, az `1` élcímkén haladunk tob a **következő pontba**
- Az új pont címkéje **a**, ezért: **output ← a**

`184.1.1.1=10111...`

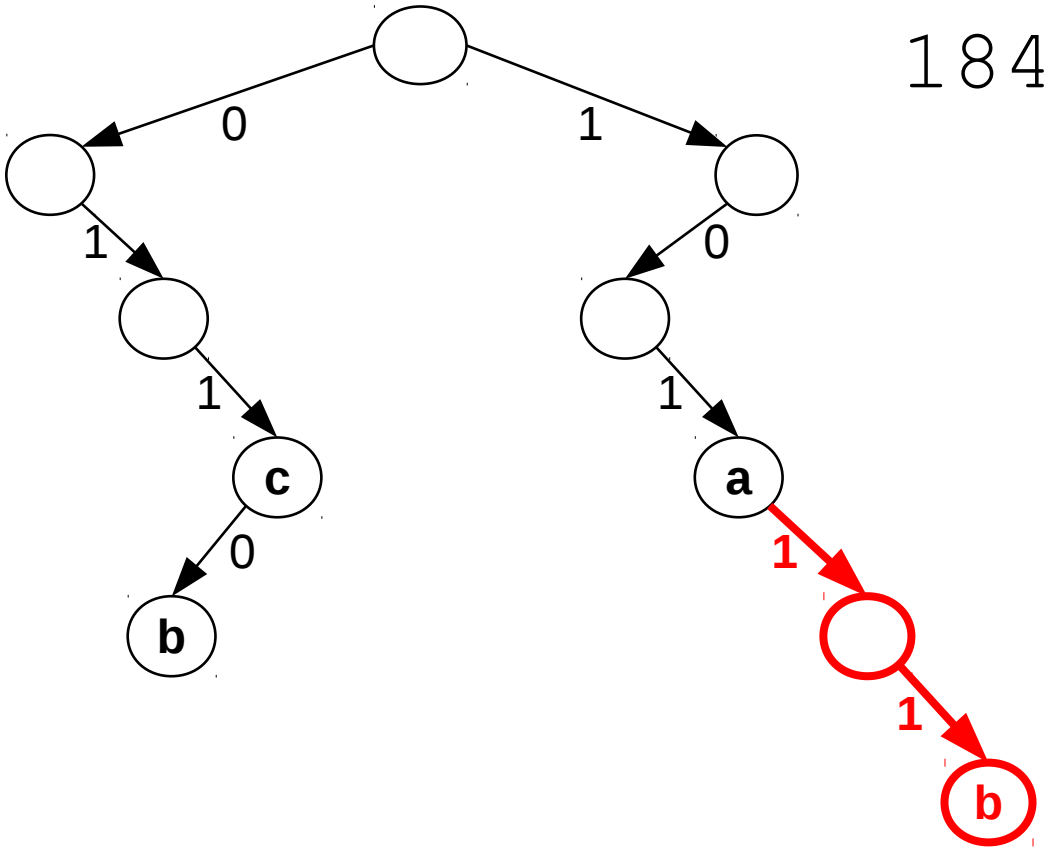


IP prefix	Prefix	Címke
160.0.0.0/3	**101**	**a**
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	10111	b

A prefix fa: keresés

- A 4. és 5. bit `11`, az `1` élcímkéken egy **b**, címkével ellátott pontba jutunk, ezért: **output ← b**
- A kapott pont levél, így **kilépés**: **output = b**

`184.1.1.1=101`**`11`**`...`



output: **b**

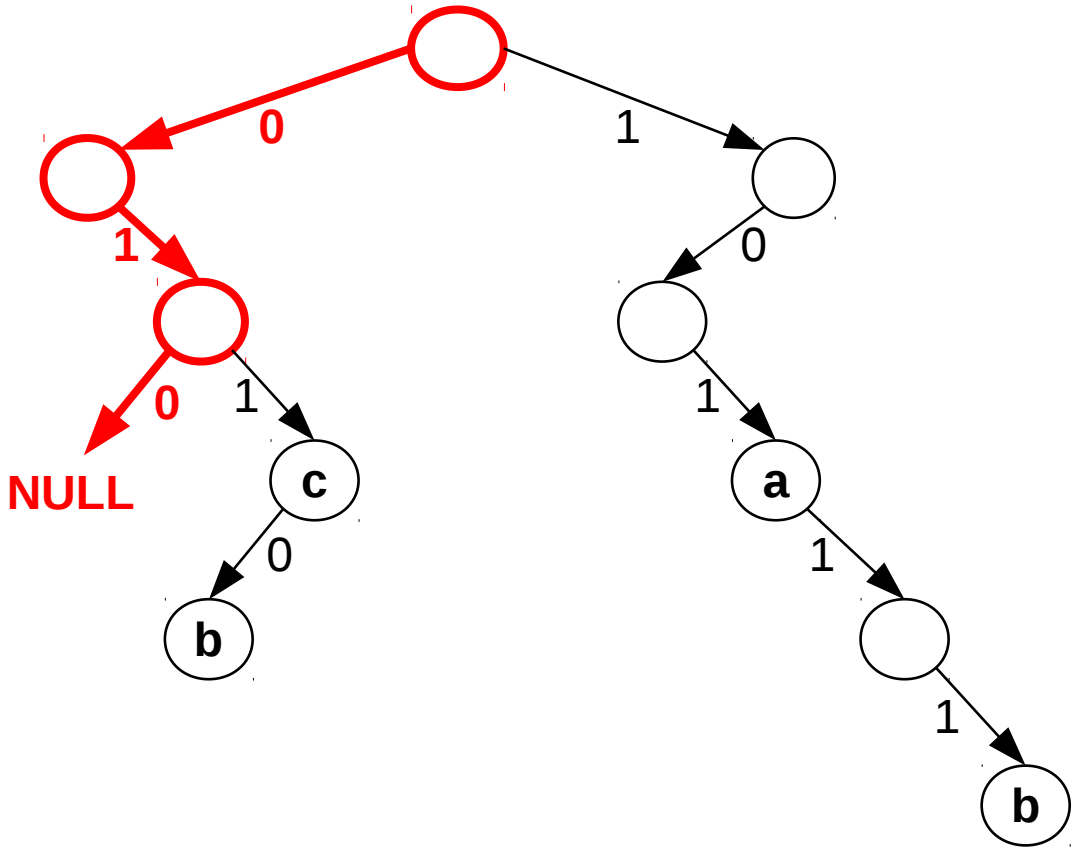
IP prefix	Prefix	Címke
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
184.0.0.0/5	**10111**	**b**

A prefix fa: keresés

- **Algoritmus:** sorban olvassuk a keresett IP cím bitjeit, és az olvasott bitnek megfelelően mindig a `0` vagy `1` élcímkével ellátott élen lépünk tovább egy következő pontba
- Az iteráció közben az output változóban tároljuk a legutoljára olvasott címkét (kezdetben: érvénytelen)
- Ha levélpontot vagy NULL pointert találunk: kilépés
- A kapott címkéhez (output) kiolvasuk a next-hop indexből a megfelelő next-hop címet és visszaadjuk
- Esetünkben az LPM eredmény: `b→10.0.0.2`

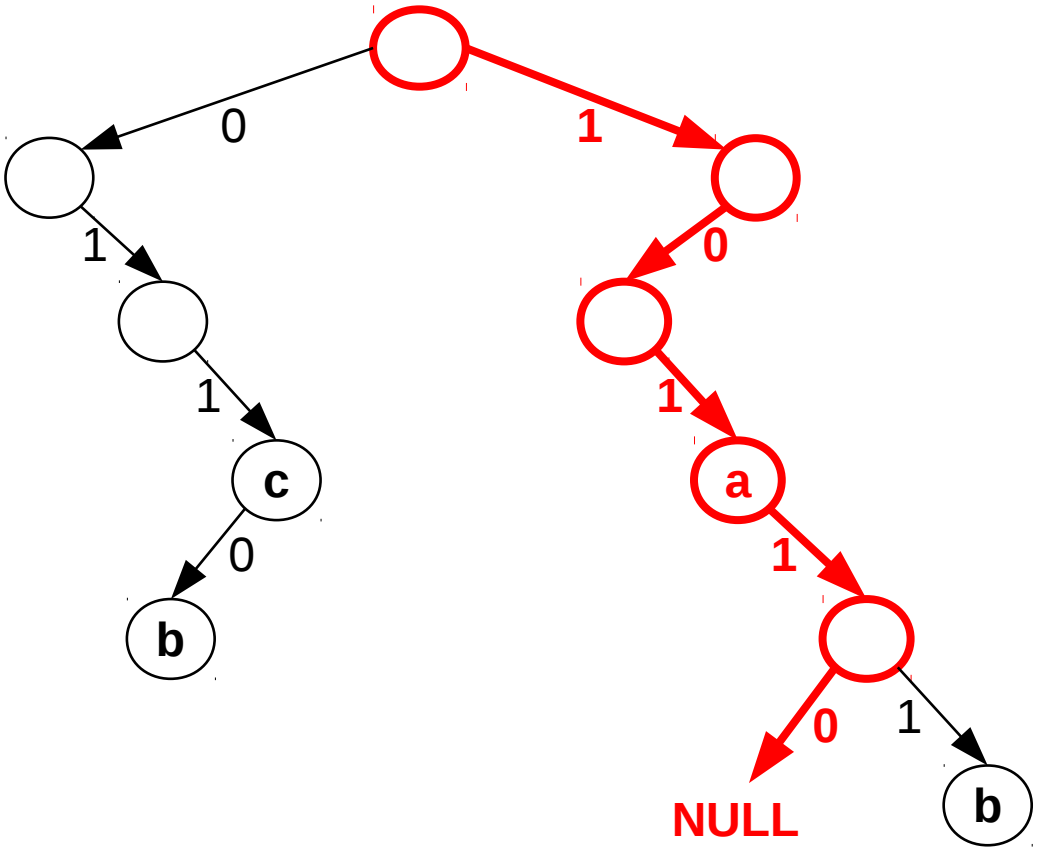
A prefix fa: keresés

- LPM a `69.12.75.54=01000...` IP címre
- Nem találkozunk érvényes címkéjű ponttal:
  `output = érvénytelen`



A prefix fa: keresés

- LPM a `178.4.66.19=10110...` IP címre
- Utolsó látott címke: **a**, **`output = a`**



A prefix fa

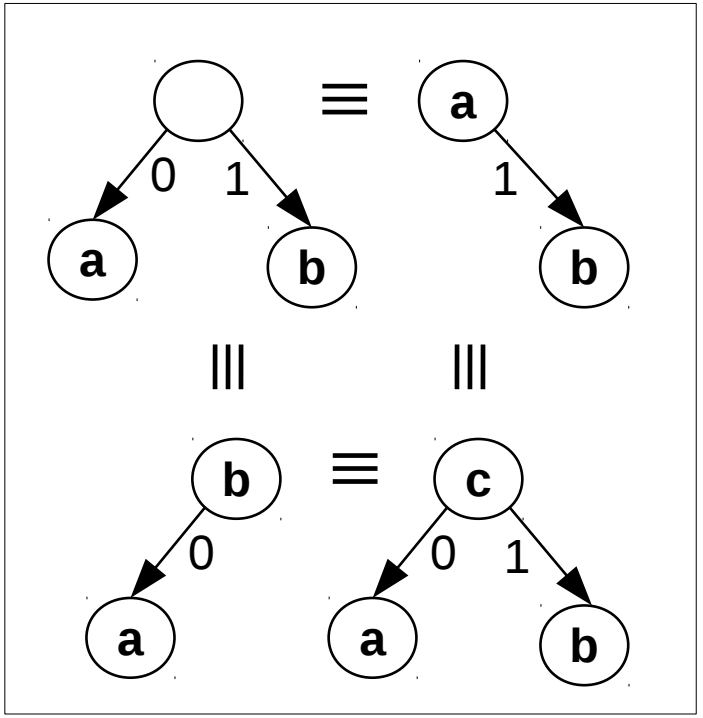
- **Tétel:** a prefix fán a LPM keresés, beillesztés, törlés és módosítás, legfeljebb *O(W)* lépésben végez, ahol *W* a címtér bitszélessége (IPv4: 32, IPv6: 128)
- A fának annyi szintje van, ahány bitből áll a cím
- Általánosságban: *N* prefix tárolása esetén a LPM keresés, beillesztés, törlés és módosítás műveletek komplexitása *O(log N)*
- A táblázatos tárolási módszerrel az összes prefixen végig kellene keresni: *O(N)* lépés

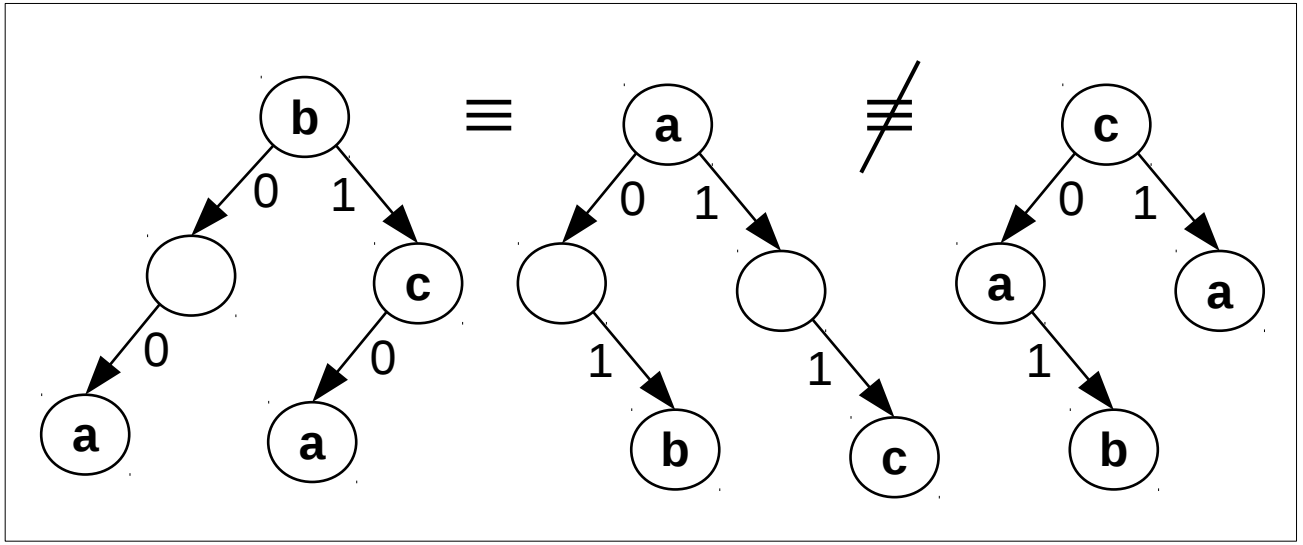
Prefix fák transzformációja

- A bináris prefix fa nem minden esetben ideális
  - túl sok pontot/pointert tartalmaz, nem fér a gyors memóriába (SRAM cache)
  - 32 RAM elérés nagy sebességnél gond lehet
- A prefix fák nem egyediek: keressünk adott FIBhez hatékonyabb prefix fákat!
- **FIB aggregáció:** a prefix fa konverziója kisebb, de az eredetivel keresés szempontjából teljesen ekvivalens formába

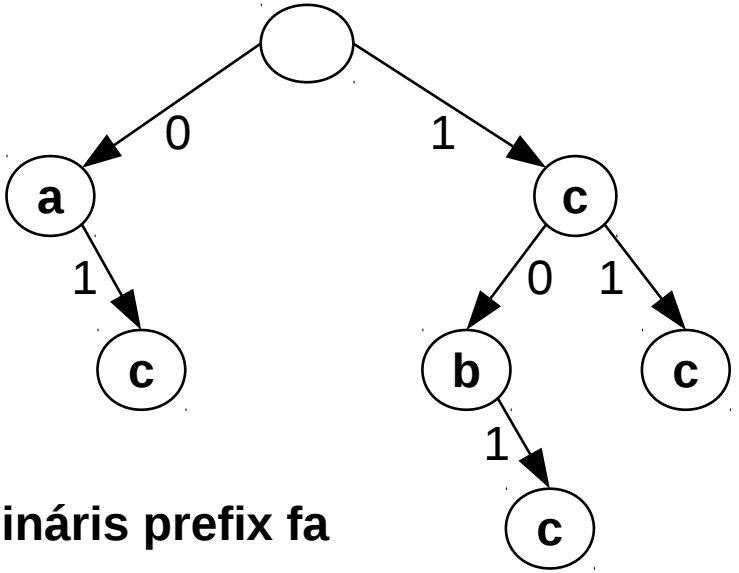
Prefix fák ekvivalenciája

- **Jelölés:** legyen *LPM(F,* α*)* = *x*, ha *F* FIBen α IP címre pontosan *x* next-hop címke a LPM
- **Def.:** $FIB_1 \equiv FIB_2$, ha minden 32 bites α IPv4 címre: $\forall\alpha$: $LPM(FIB_1, \alpha) = LPM(FIB_2, \alpha)$

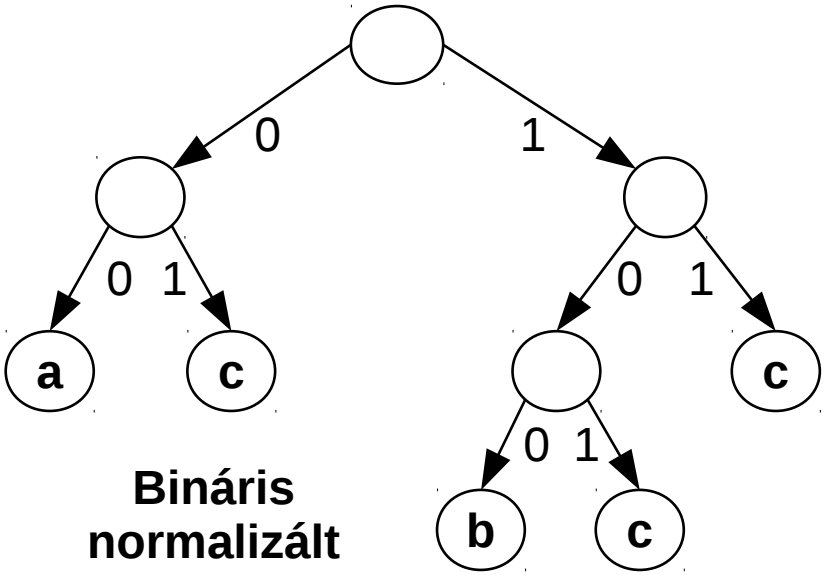




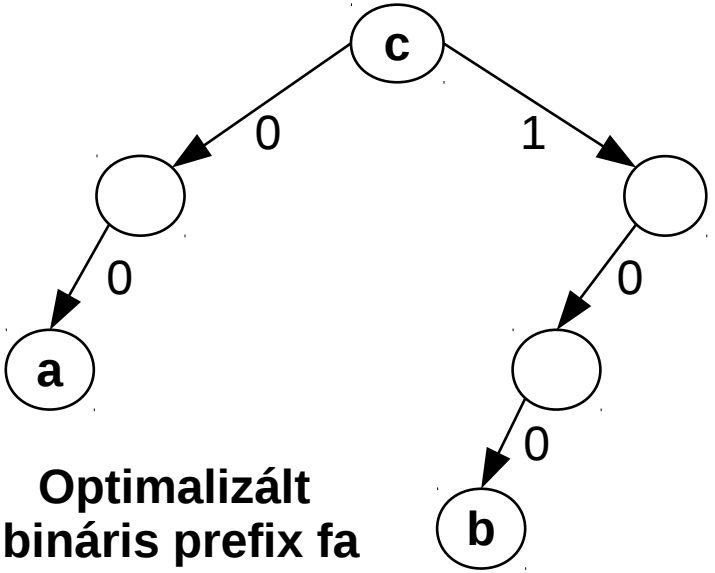
FIB aggregáció



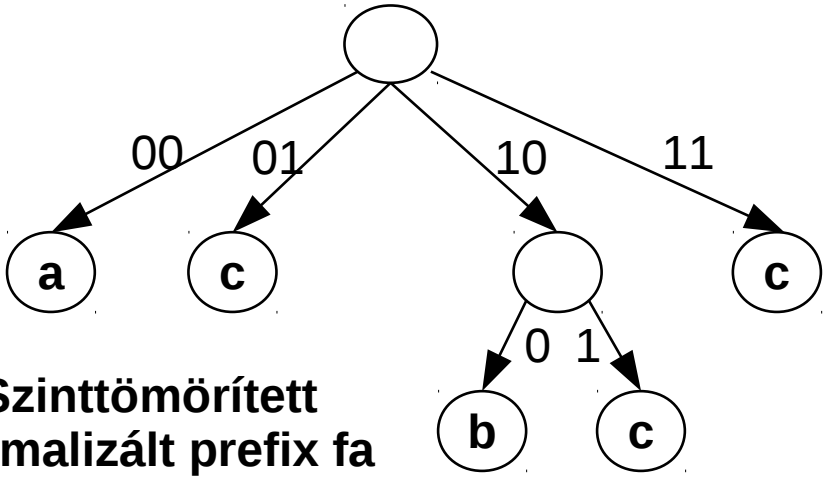
Bináris prefix fa



Bináris normalizált prefix fa



Optimalizált bináris prefix fa



Szinttömörített normalizált prefix fa

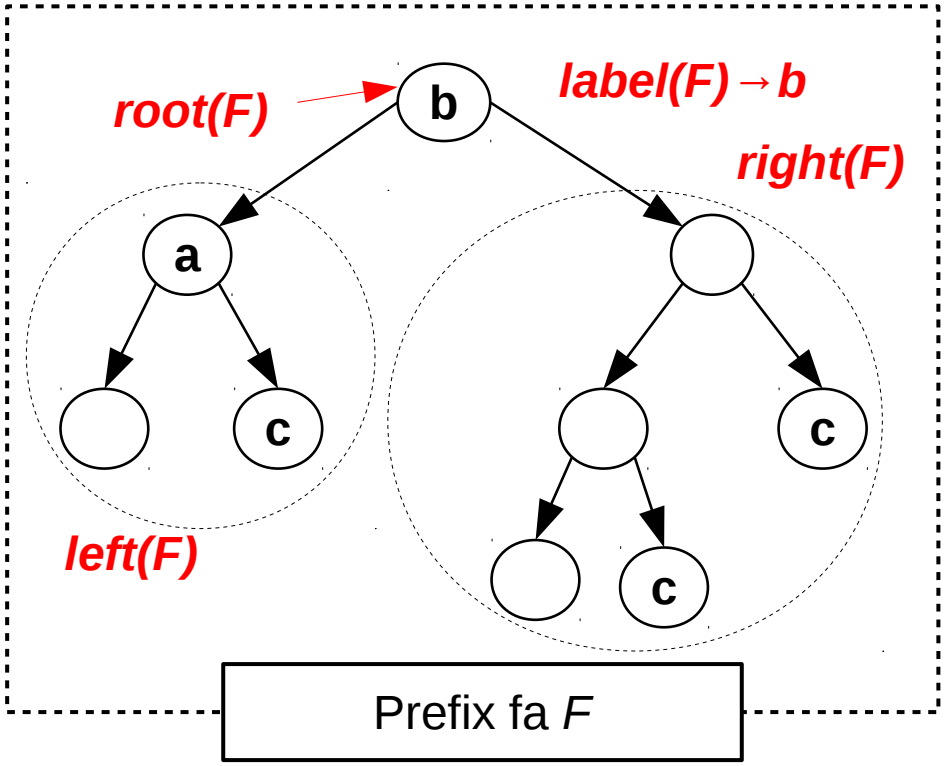
Fabejárások

Fabejárások

- A fabejárások segítségével egyszerű formában definiálhatók a fákon végzett transzformációk
- **Preorder/postorder bejárások:** *F* címkézett fa összes pontján egy *f* függvény alkalmazása
- A különbség a pontok sorrendje
- Az alábbiakban csak a legegyszerűbb esettel foglalkozunk („parentless” fák, „állapotmentes preorder”)

Jelölés

- **Egyszerűsítés:** *F* címkézett fa és gyökérpontja *p = root(F)* a jelölésben felcserélhető
- *F* pont bal/jobb oldali részfája: *left(F), right(F)*
- *F* címkéje: *label(F)*
- Az egyszerűsítés miatt tehát például *label(F) = label(root(F))*
- Most az üres pontokat is tároljuk



Fabejárások: preorder

- **Preorder bejárás:** *preorder(F, f, i)*
  - először *F* gyökerére alkalmazzuk *f*-et
  - majd a bal és jobb oldali részfákra rekurzívan
  - *f* visszatérési értéke tovább a gyermekeknek mint kezdeti érték
  - a bejárás kezdeti értékét meg kell adni!

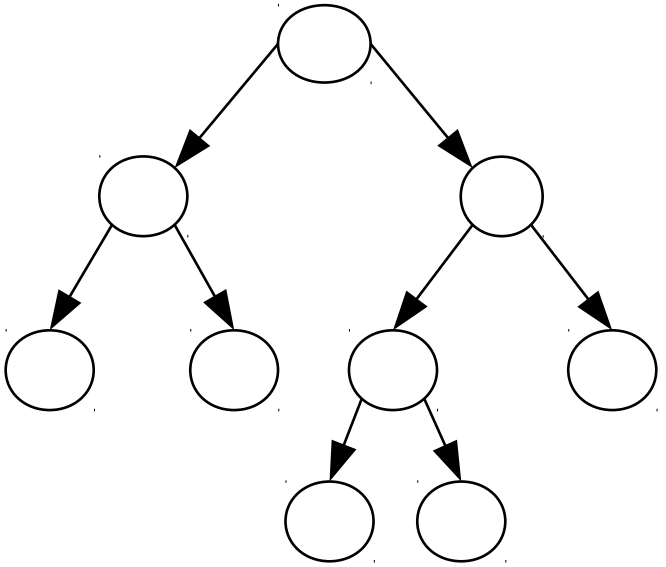
```
preorder(F, f, i):
   x ← f(F, i)
   preorder(left(F), f, x)
   preorder(right(F), f, x)
```

Preorder: példa

- Legyen *f* a következő:

```
f(F, i):
   label(F) ← i
   return (i+1)
```

- Legyen *F* prefix fa az alábbi:



- Értékeljük ki *preorder(F, f, 1)* bejárást

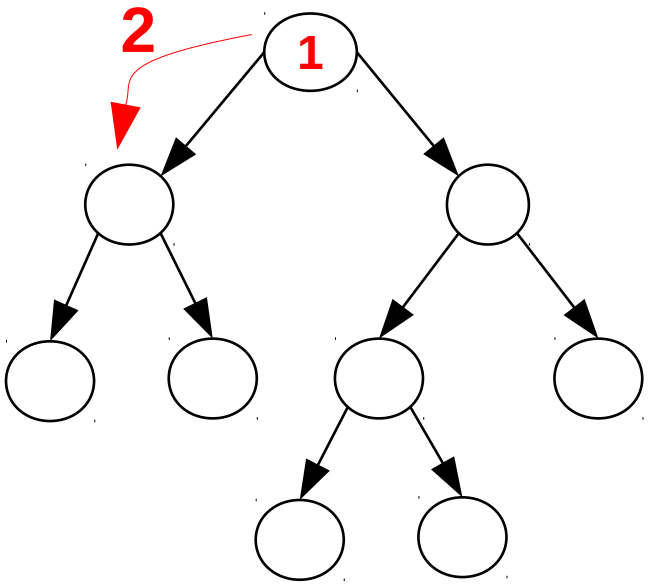
Preorder: példa

```
preorder(F, f, i):
   x ← f(F, i)
   preorder(left(F), f, x)
   preorder(right(F), f, x)
```

```
f(F, i):
   label(F) ← i
   return (i+1)
```

1) *F* gyökerének címkéje → **1**

- *f* visszatérési értéke: **2**
- *preorder(left(F), f, 2):* a visszatérési érték a bal oldali részfa kezdőértéke
- rekurzió a részfákba



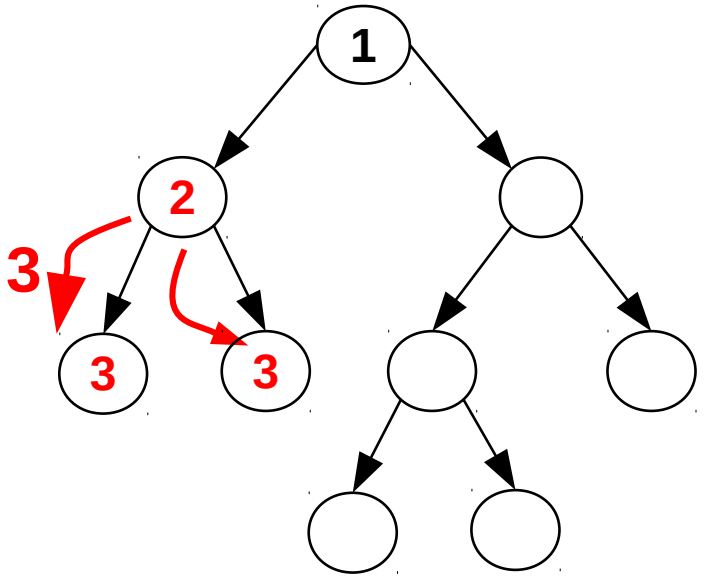
Preorder: példa

```
preorder(F, f, i):
   x ← f(F, i)
   preorder(left(F), f, x)
   preorder(right(F), f, x)
```

```
f(F, i):
   label(F) ← i
   return (i+1)
```

2) *left(F)* gyökercímkéje → **2**

- *f* visszatérési értéke: **3**
- *preorder(left(F), f, 3):* bal oldali fa címkéje → **3**
- levélpont: rekurzió leáll
- *preorder(right(F), f, 3):* jobb oldali fa címkéje → **3**



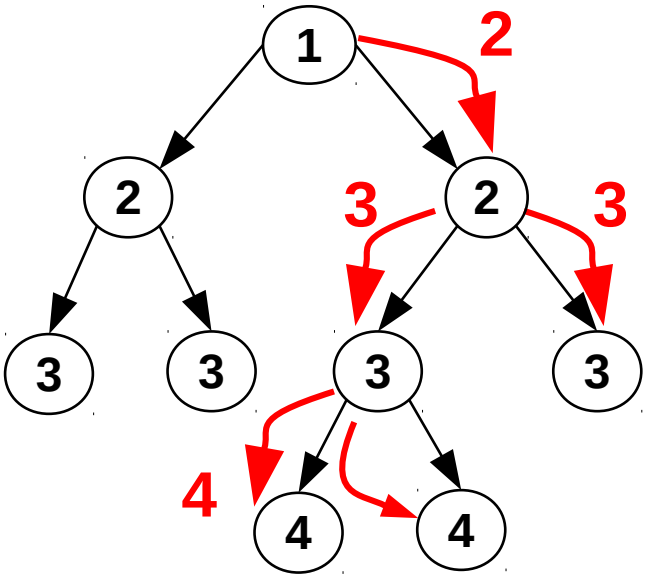
Preorder: példa

```
preorder(F, f, i):
   x ← f(F, i)
   preorder(left(F), f, x)
   preorder(right(F), f, x)
```

```
f(F, i):
   label(F) ← i
   return (i+1)
```

3) Vissza *F* gyökerére és *preorder(right(F), f, 2)*

- A preorder bejárásunk érdekes dolgot csinál
- Minden ponton beállítja a pont szintjét a fában!



Fabejárások: postorder

- A preorder ellentéte: most *f* függvényt előbb alkalmazzuk a részfákon rekurzívan és csak ezután a gyökéren
- **Postorder bejárás:** *postorder(F, f)*
  - alkalmazzuk *f*-et a bal és jobb részfákra rekurzívan, majd hívjuk f-et a gyökéren
  - most nincs visszatérési- és kezdőérték

```
postorder(F, f):
   postorder(left(F), f)
   postorder(right(F), f)
   f(F)
```

Postorder: példa

- Legyen most *f* a következő:

```
f(F):
    if (F leaf):
        label(F) ← 1
    else:
        label(F) ← label(left(F)) +
                   label(right(F)) + 1
```

- Ha nincs további részfa (*F* levélpont) a címke 1
- Ellenkező esetben a gyermekek címkéinek összege plusz 1
- Legyen *F* prefix fa ugyanaz, mint előbb
- Értékeljük ki *postorder(F, f)* bejárást

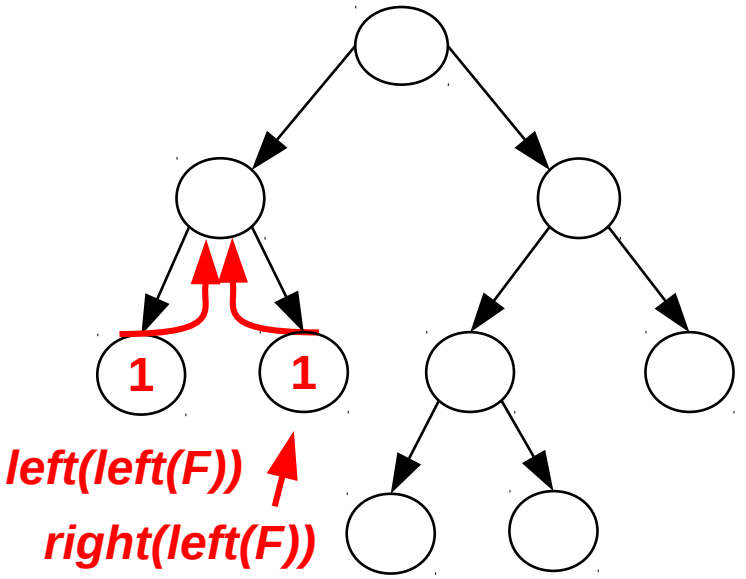
Postorder: példa

```
postorder(F, f):
   postorder(left(F), f)
   postorder(right(F), f)
   f(F)
```

```
f(F):
  if (F leaf): label(F)←1
  else:
    label(F)←label(left(F))
      + label(right(F)) + 1
```

1) *F* mindkét részfáján rekurzió

- *postorder(left(F), f):* további rekurzió a két részfán
- *f* először *left(left(F))* és *right(left(F))* levélen hívódik
- mindkét levél címkéje: **1**



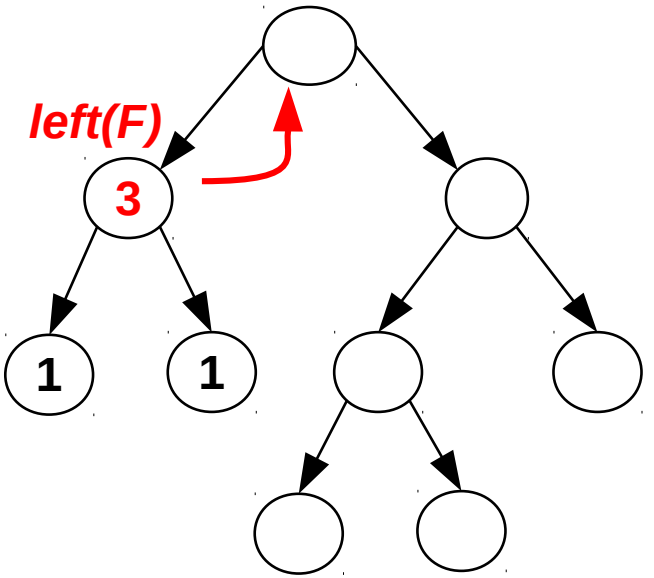
Postorder: példa

```
postorder(F, f):
   postorder(left(F), f)
   postorder(right(F), f)
   f(F)
```

```
f(F):
  if (F leaf): label(F)←1
  else:
    label(F)←label(left(F))
      + label(right(F)) + 1
```

2) Mivel a gyermekei bejárva, *f* függvény hívódik *left(F)* részfán

- az új címke: **1 + „a gyermekek címkéinek összege” = 3**



Postorder: példa

```
postorder(F, f):
   postorder(left(F), f)
   postorder(right(F), f)
   f(F)
```

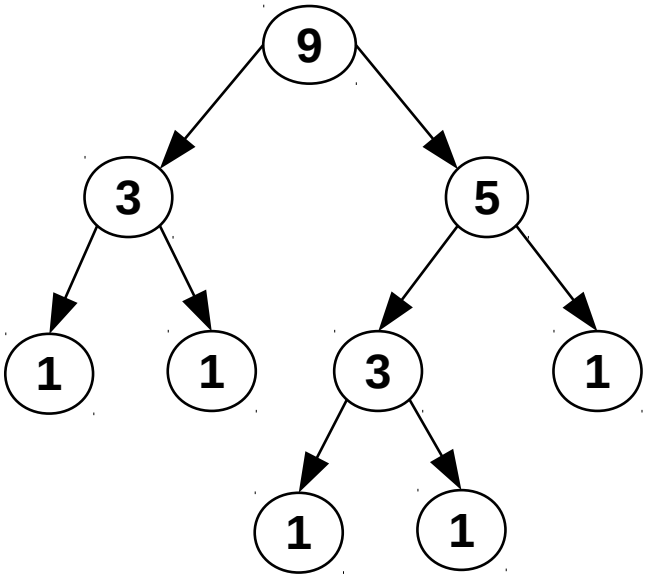
```
f(F):
  if (F leaf): label(F)←1
  else:
    label(F)←label(left(F))
      + label(right(F)) + 1
```

3) Rekurzívan bejárjuk *right(F)* részfát

4) Legutoljára a gyökérponton értékeljük ki *f* függvényt

az új címke: **1+3+5=9**

A bejárás minden pontba beírja a hozzá tartozó részfa pontjainak számát!



Érdekes fabejárások

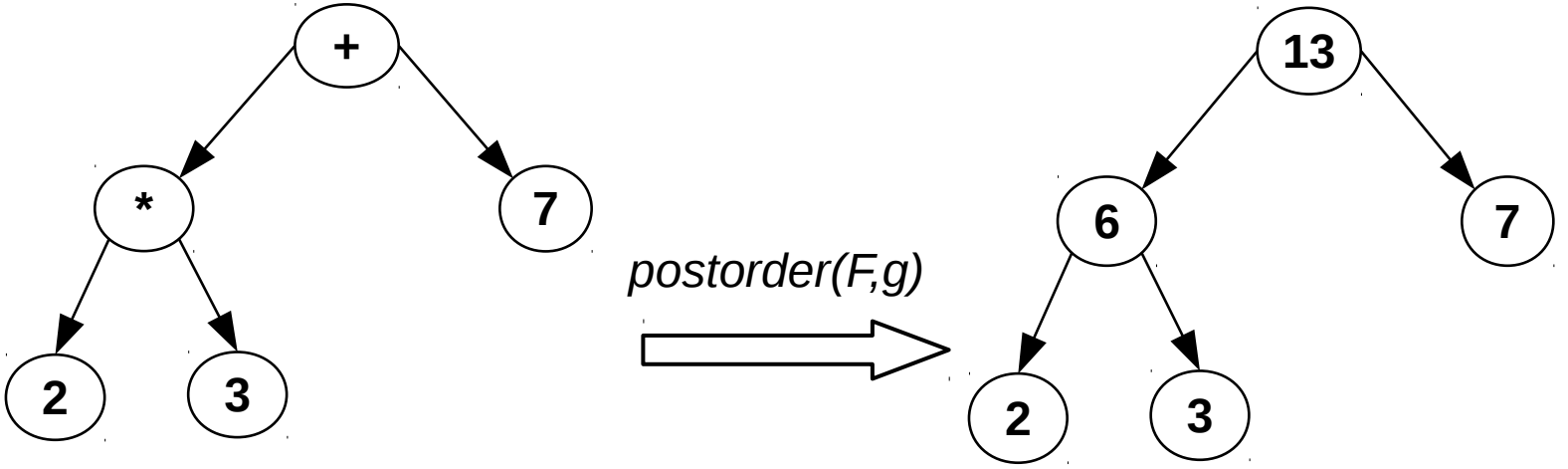
- Részfák mélysége (leghosszabb út levélig): *postorder(F, f)* az alábbi *f* függvénnyel

```
f(F): if (F leaf): label(F) ← 1
  else label(F) ← max[label(left(F)), label(right(F))] + 1
```

- Aritmetikai kifejezés kiértékelése: *postorder(F,g)*

```
g(F):
  if (F='+'): label(F)←label(left(F))+label(right(F))
  if (F='*'): label(F)←label(left(F))*label(right(F))
```

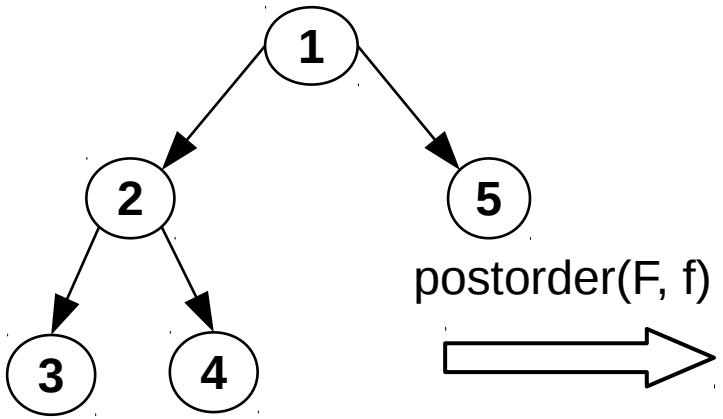
2 * 3 + 7



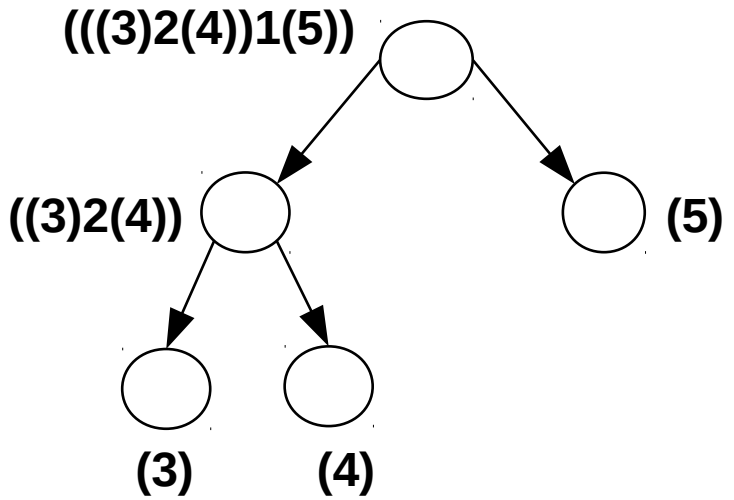
Érdekes fabejárások

- Fa visszaállítható konvertálása sztringre (szerializálás): *postorder(F, f)*

```
f(F):
   if (F is leaf):
      label(F) ←'(' . label(F) . ')'
   else: label(F) ←'(' . label(left(F)) .
                label(F) . label(right(F)) . ')'
```



postorder(F, f)



Érdekes fabejárások

- Legyenek *f* és *g* függvények az alábbiak

```
f(F):
   if (F leaf): label(F)←1
   label(F)←label(left(F))
      + label(right(F)) + 1
```

```
g(F, i):
   label(F) ← label(F) / i
   return i
```

- Mit tartalmaznak ekkor az alábbi *F''* címkéi?

```
F' ← postorder(F, f)
F'' ← preorder(F, g, label(F'))
```

- Először előállítjuk a pontokban a részfák méretét, majd leosztjuk a címkéket az egész fa méretével
- Részfák relatív mérete az egész fához képest