Mérnöki modellalkotás
Az elmélettől a gyakorlatig

IP forgalomtovábbítási táblák tömörítése

Összefoglalás

Rétvári Gábor

Bad programmers worry about the code. Good programmers worry about data structures and their relationships.

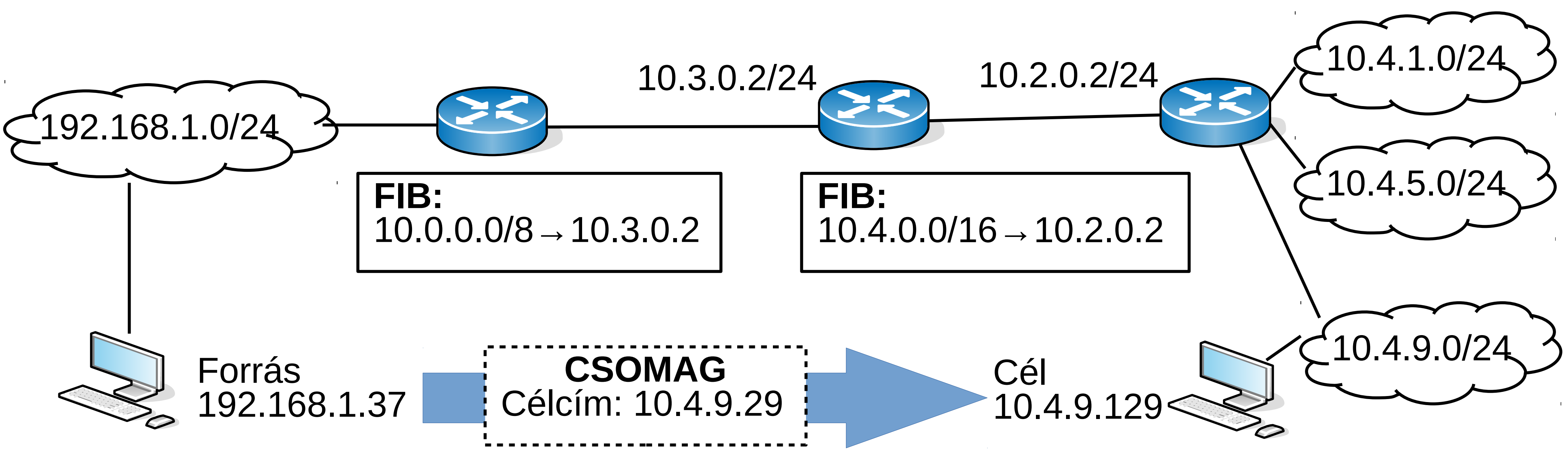
Linus Torvalds

Az előadáson elsajátítható tudás

- A korábbi előadásokon elhangzottak összefoglalója: IP forgalomtovábbítás, LPM és a prefix fák, preorder/postorder, prefix fák ekvivalenciája, normalizálás, ORTC
- Kitekintés: prefix fák optimális szinttömörítése
- FIB tömörítés a gyakorlatban

IPv4 forgalomtovábbítás

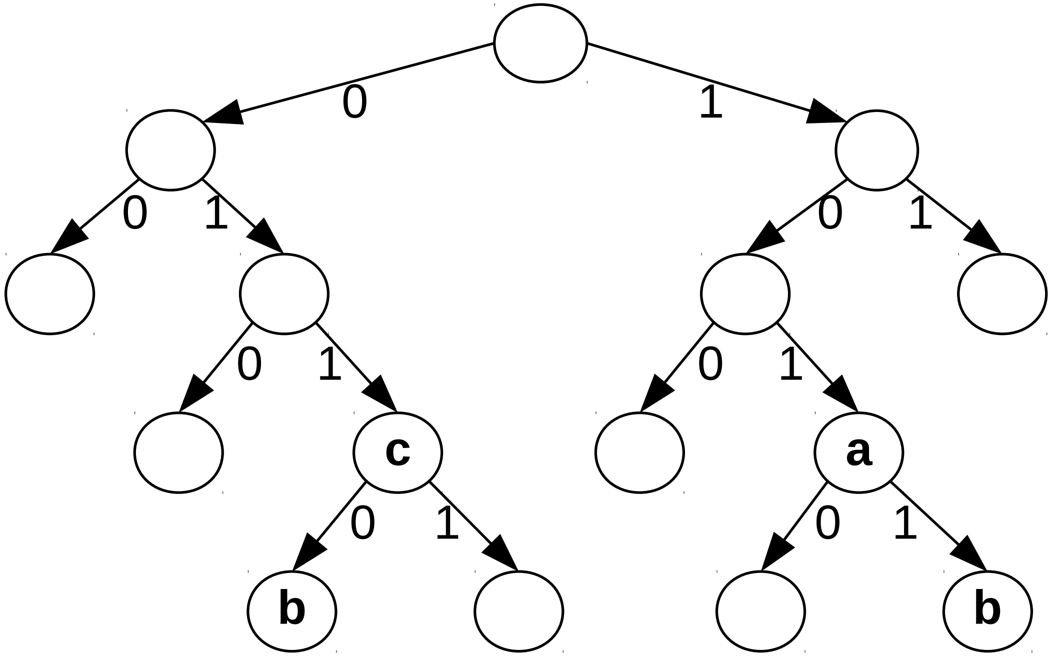
- Minden csomag tartalmazza a cél IP címét
- Keresés a forgalomtovábbítási táblában (FIB)
- Eredmény: az útvonalon következő router (next-hop) IP címe



A legspecifikusabb prefix

- **Longest Prefix Match (LPM):** ha egy IP címre több bejegyzés illeszkedik, akkor a legtöbb biten (leghosszabb prefixen) illeszkedő a preferált
- Táblázat: LPM komplexitása *O(n)*, *n* bejegyzésre
- **Bináris prefix fa:** *O(log n)* futási időben LPM

IP prefix	Prefix	NH
160.0.0.0/3	101	a
96.0.0.0/4	0110	b
96.0.0.0/3	011	c
176.0.0.0/4	1011	b

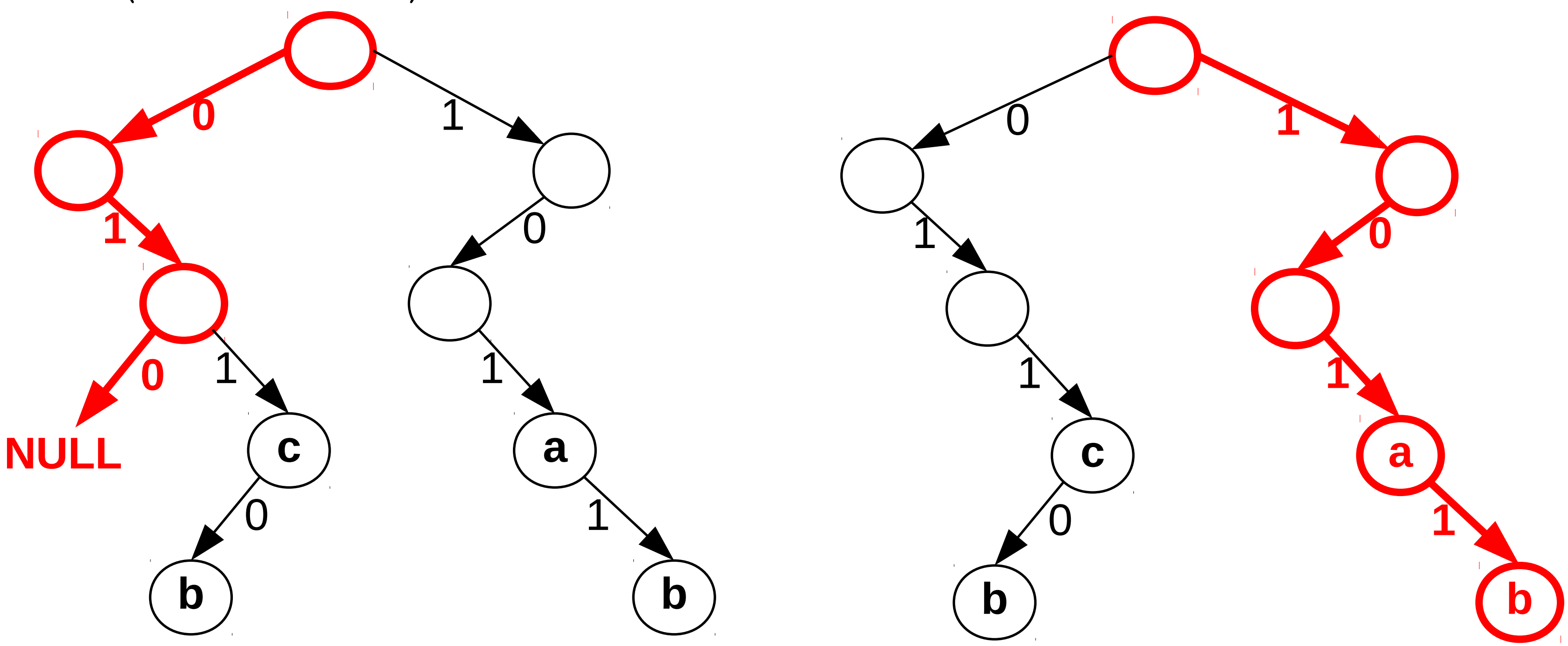


A prefix fa: keresés

- A keresett IP cím következő bitjének megfelelően a `0` vagy `1` élcímkéjű élen lépünk tovább
- Tároljuk a legutoljára olvasott címkét

LPM(69.12.75.54) =
LPM(01000...) → NULL

LPM(178.4.66.19) =
LPM(1011...) → b



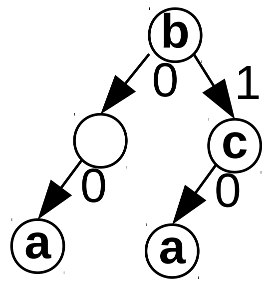
Prefix fák ekvivalenciája

- FIBek leírása nem egyedi: **FIB aggregáció**

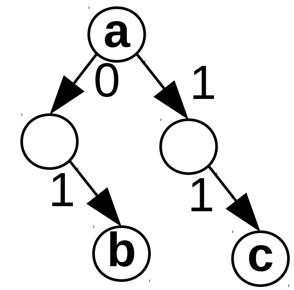
IP prefix	Prefix	Címke
0.0.0.0/0	-	b
128.0.0.0/1	1	c
0.0.0.0/2	00	a
128.0.0.0/2	10	a

$\equiv$

IP prefix	Prefix	Címke
0.0.0.0/0	-	a
64.0.0.0/1	01	b
192.0.0.0/2	11	c



$\equiv$



- Két FIB ekvivalens, ha minden 32 bites α IPv4 címre az LPM eredménye megegyezik

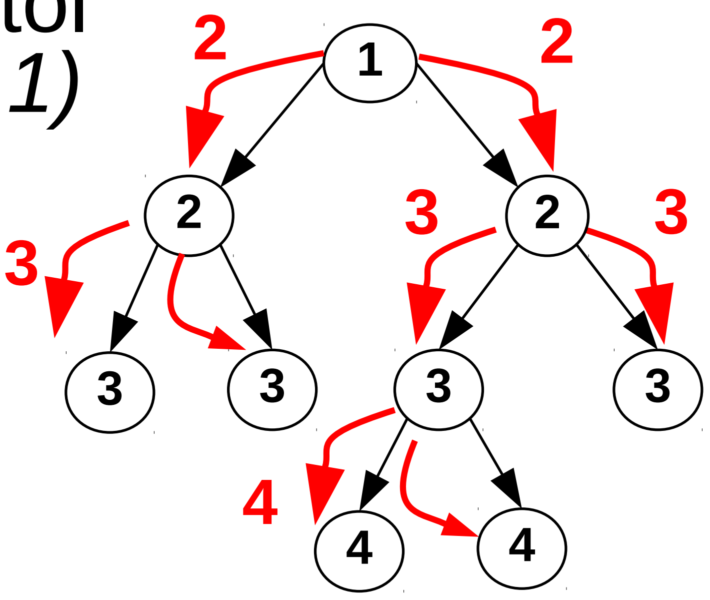
$$\boxed{FIB_1 \equiv FIB_2, \text{ ha } \forall\alpha: \; LPM(FIB_1, \alpha) = LPM(FIB_2, \alpha)}$$

Fabejárások: preorder

- *preorder(F, f, i):* alkalmazzuk *f*-et *F* gyökerére majd a bal és jobb oldali részfákra rekurzívan

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

- Írjuk minden pontba a gyökértől vett távolságot: *preorder(F, f, 1)* ahol az *f* függvény:




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

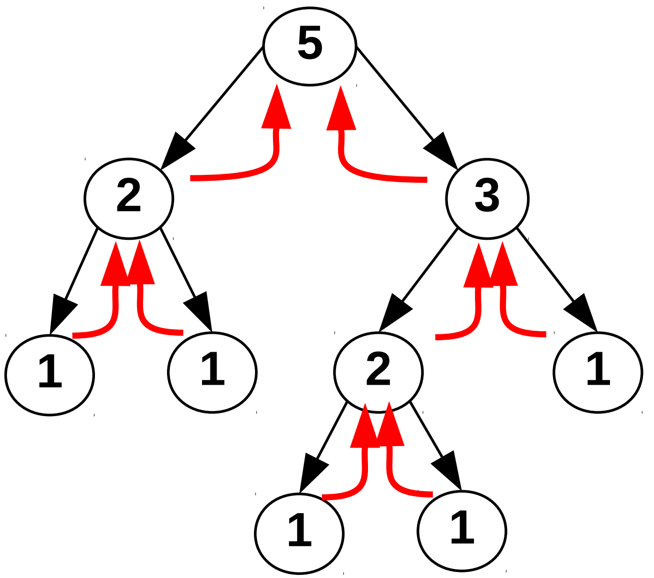
Fabejárások: postorder

- *postorder(F, f): f* függvényt előbb alkalmazzuk a részfákon rekurzívan és csak ezután a gyökéren

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

- Írjuk be minden pontba a részfa **leveleinek** számát

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



Normalizálás

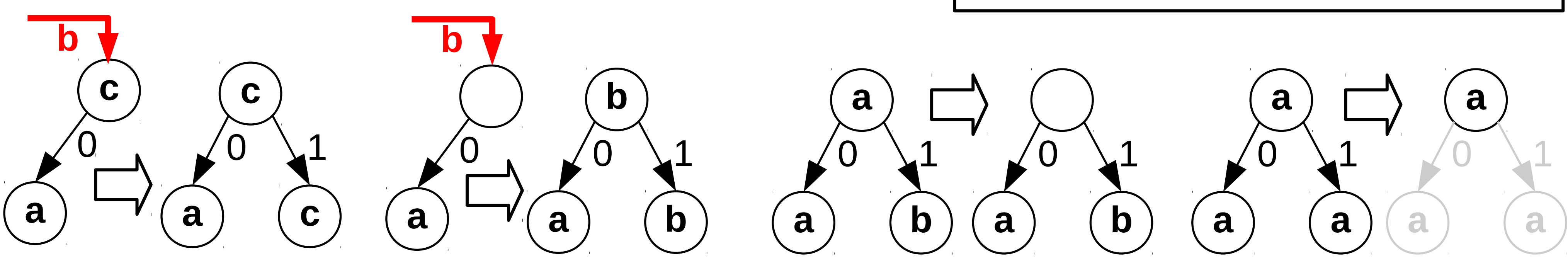
- FIB ekvivalens transzformációja egyedi alakba
- Egymás utáni preorder és postorder bejárással

1. preorder(F, f, d_0)

```
f(F, i):
  if F is interior:
    if ∄ left(F):  add_node(left(F))
    if ∄ right(F): add_node(right(F))
  if (label(F) == ∅):
    label(F) ← i
  return  label(F)
```

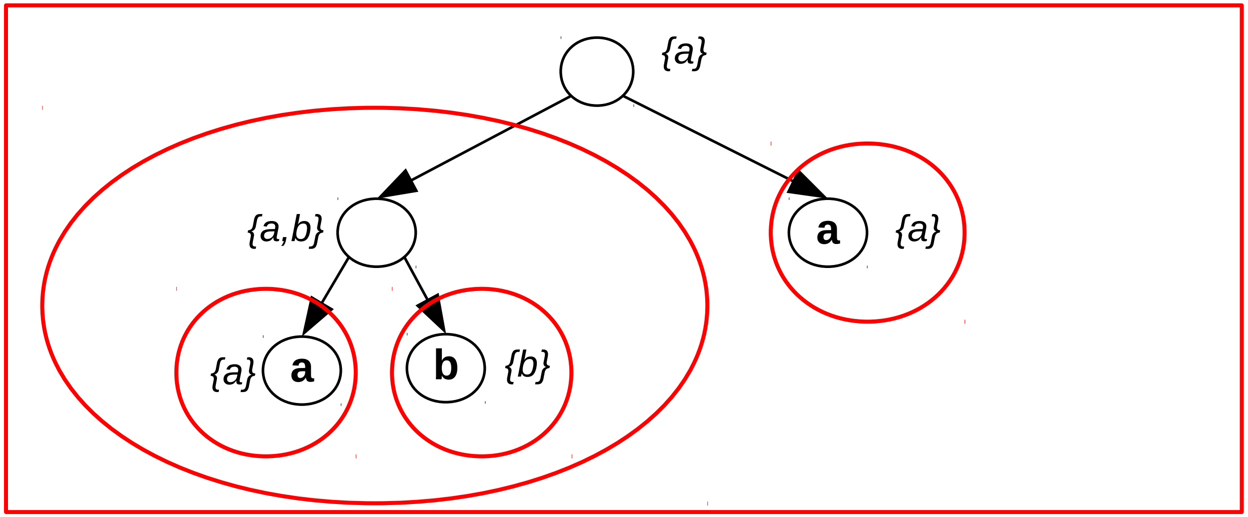
2. postorder(F, g)

```
g(F, i):
  if F is leaf: return
  if label(left(F)) ==
    label(right(F)) ==
    label(F):
      remove_node(left(F))
      remove_node(right(F))
      return
  label(F) = ∅
```



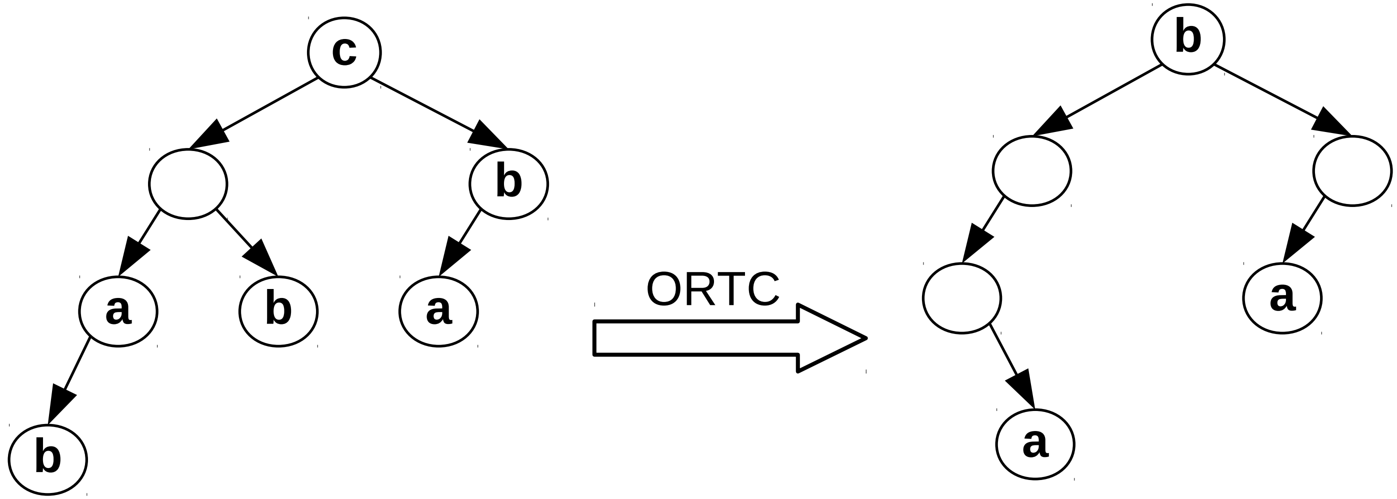
Dinamikus programozás

- Általános problémamegoldási stratégia
  - a problémát felosztjuk egymásba ágyazódó részproblémákra
  - a „legszűkebb” részproblémákra megadjuk a megoldásokat
  - innen indulva rekurzívan felírjuk a megoldást



FIB optimális tömörítése: ORTC

- **ORTC:** dinamikus programozási algoritmus a legkevesebb bejegyzést tartalmazó FIB előállítására (legkevesebb címkét tartalmazó fa)

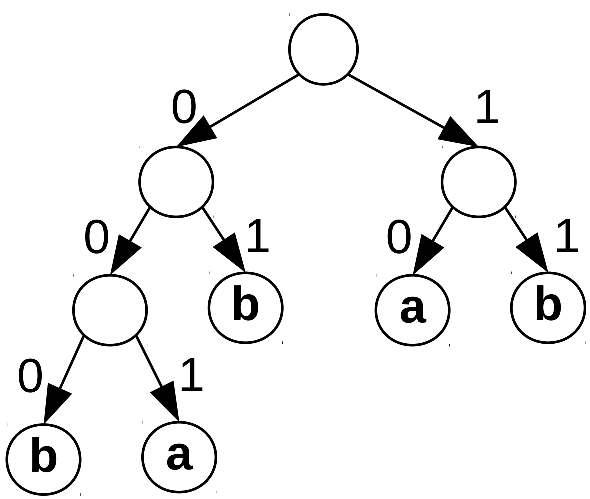


IP prefix	Prefix	Címke
0.0.0.0/0	-	c
128.0.0.0/1	1	b
0.0.0.0/2	00	a
64.0.0.0/2	01	b
128.0.0.0/2	10	a
0.0.0.0/3	000	b

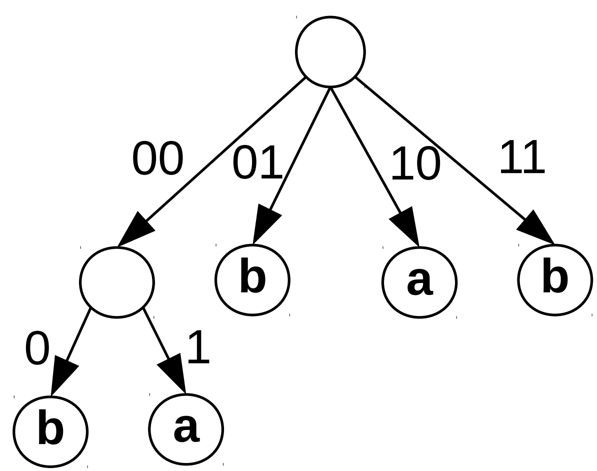
IP prefix	Prefix	Címke
0.0.0.0/0	-	b
128.0.0.0/2	10	a
32.0.0.0/3	001	a

Kitekintés: Szinttömörített prefix fák

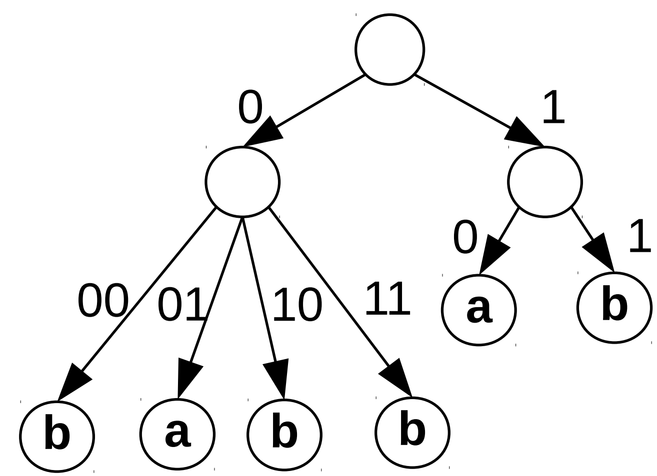
- Eddig bináris prefix fákkal dolgoztunk: minden belső pontnak 2 gyermeke van a fában
- **Szinttömörített fa:** minden belső pontnak 2^k gyermeke van valamely $k > 0$ egész számra



Bináris prefix fa
(normalizált)



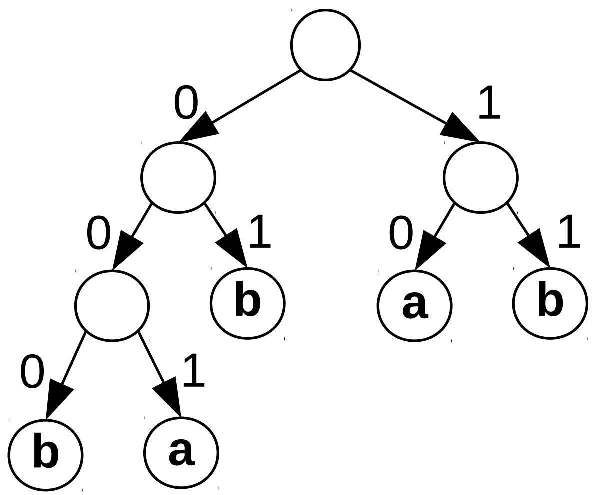
Szinttömötített prefix fa
(normalizált)



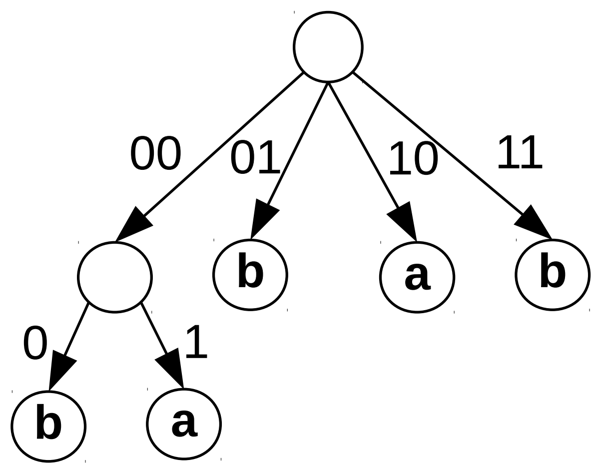
Alternatív
szinttömötített prefix fa
(normalizált)

Kitekintés: Szinttömörített prefix fák

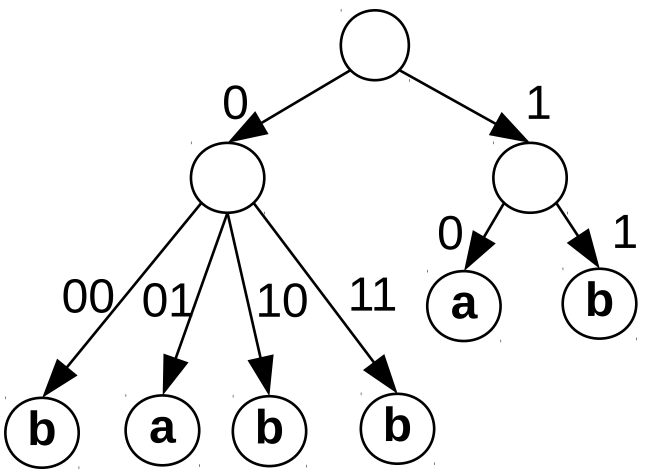
- LPM alapvetően ugyanaz, mint a bináris fán, de ha egy pontnak *2^k* gyermeke van, akkor egyszerre *k* bitet olvasunk a keresett IP címből
- *$2=2^1$* gyermek esetén *1* bitet, *$4=2^2$* esetén kettőt



Bináris prefix fa
(normalizált)



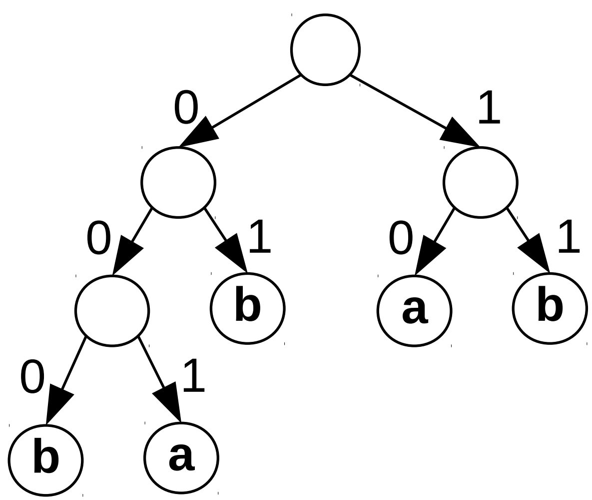
Szinttömötített prefix fa
(normalizált)



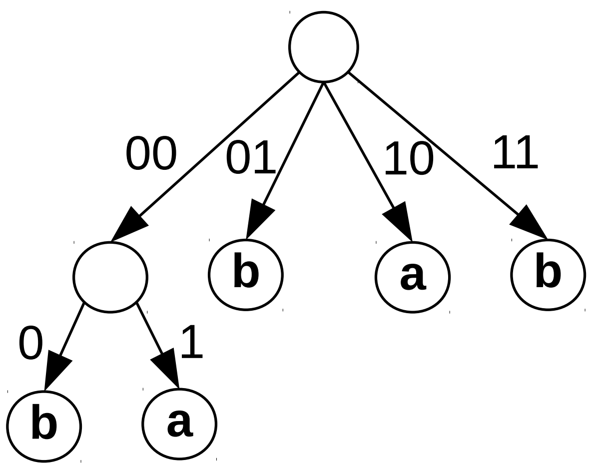
Alternatív
szinttömötített prefix fa
(normalizált)

Kitekintés: Szinttömörített prefix fák

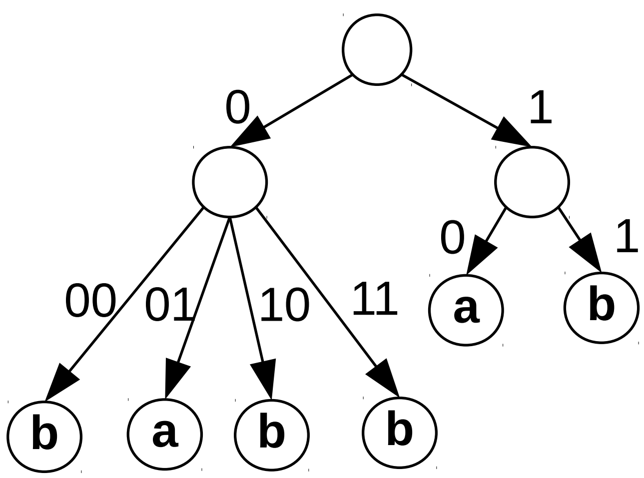
- **Előny:** kevesebb pointer = kisebb tárolási méret + gyorsabb LPM (kevesebb szintet kell bejárni)
- **Hátrány:** a pontokban tárolni kell a gyermekek 2^k számát, pontosabban k-t (ez az ún. **stride**)



Bináris prefix fa
(normalizált)



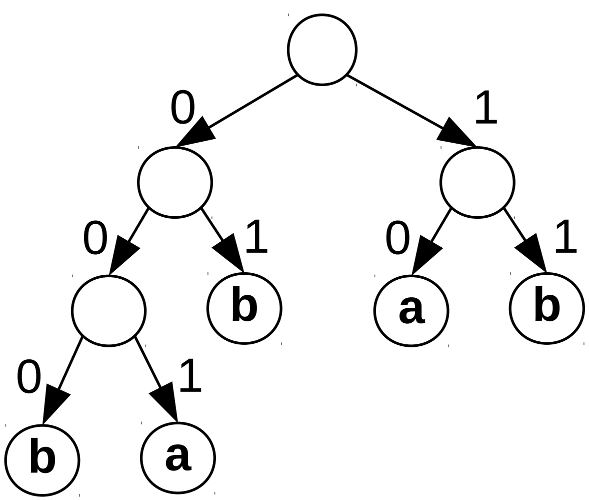
Szinttömötített prefix fa
(normalizált)



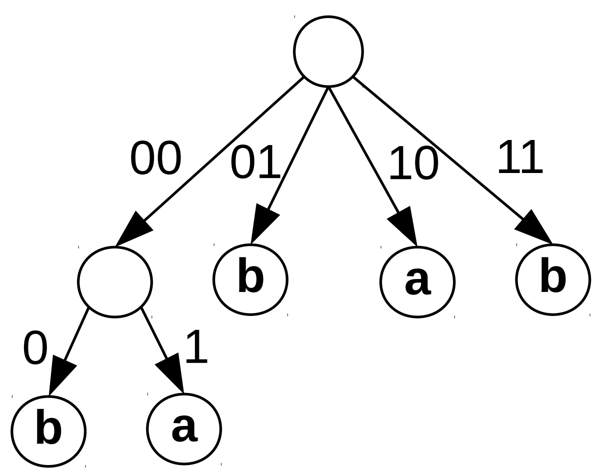
Alternatív
szinttömötített prefix fa
(normalizált)

Kitekintés: Szinttömörített prefix fák

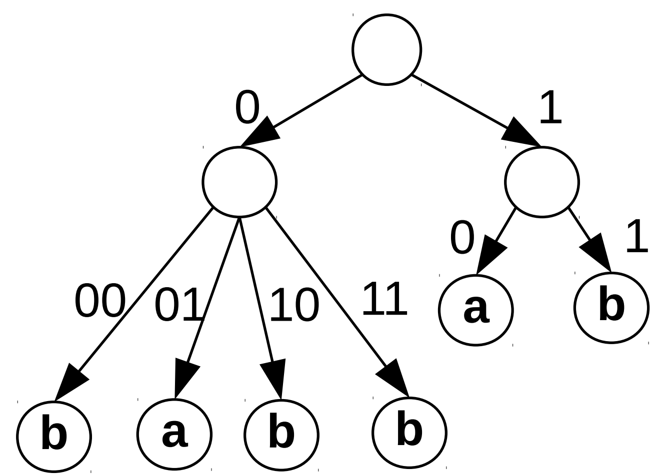
- Nem mindegy, hogy a fa egyes pontjait mekkora *stride*-ra írjuk ki → **optimalizálás**
- Az alábbi példán az első szinttömörített fában csak 6 pointer van, a másodikban 8



Bináris prefix fa
(normalizált)



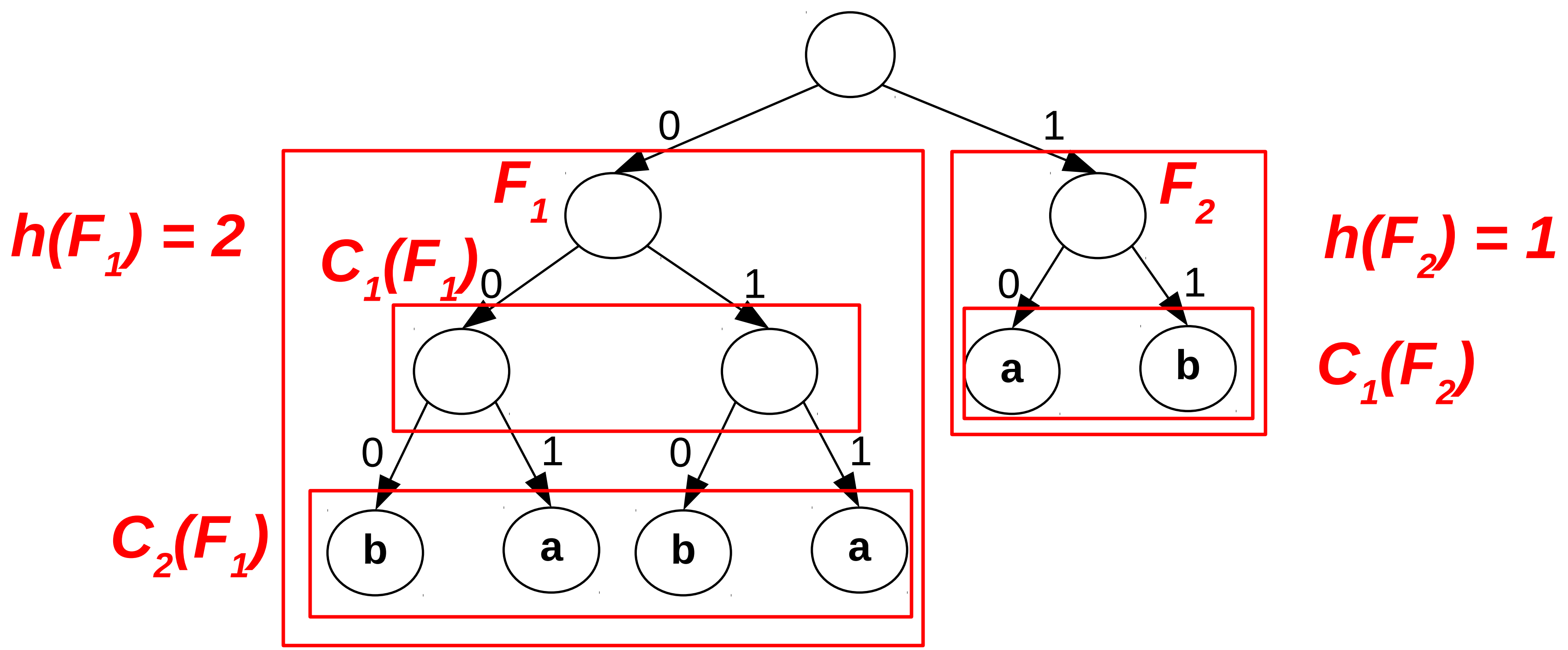
Szinttömötített prefix fa
(normalizált)



Alternatív
szinttömötített prefix fa
(normalizált)

Kitekintés: Szinttömörített prefix fák

- Induljunk ki a normalizált bináris prefix fából
- Legyen *h(F)* egy *F* részfa **mélysége** és legyen $C_i(F)$ az *F* részfában az ***i-edik* szinten levő gyermekek halmaza** (*root(F)*-től számítva)



Kitekintés: Szinttömörített prefix fák

- **Feladat:** határozzuk meg minden *F* részfára az optimális *stride* *n(F)* értékét úgy, hogy a lehető legkevesebb élet tartalmazó szinttömörített prefix fát kapjuk
- **Dinamikus program:** a részproblémák ismét a részfák, önmagukban optimálisan megoldhatók!

$$n(F) = \min_{k=1..h(F)} \left(2^k + \sum_{G \in C_k(F)} n(G)\right) \qquad \forall F \text{ belső pont}$$

$$n(F) = 0 \qquad \forall F \text{ levélpont}$$

- Ez alapján az optimális szinttömörítő algoritmus már könnyen megírható (vizsgafeladat)

FIB tömörítés: összefoglaló

- A FIB tömörítés ma is aktív kutatás tárgya, lásd

  G. Rétvári, J. Tapolcai, A. Kőrösi, A. Majdán, and Z. Heszberger. ***Compressing IP forwarding tables: towards entropy bounds and beyond.*** *In ACM SIGCOMM 2013, pages 111--122, 2013.*

- A **tömörített FIB adatstruktúrák a gyakorlatban elterjedten használtak**:
  - a Linux kernel `fib_trie` adatstruktúrája egy szint- (és útvonal)tömörített prefix fa
  - network processzorok mindegyike támogatja
  - Intel DPDK
- A szinttömörítés esetleg kombinálható lenne az ORTC-vel (PhD téma, valaki?)