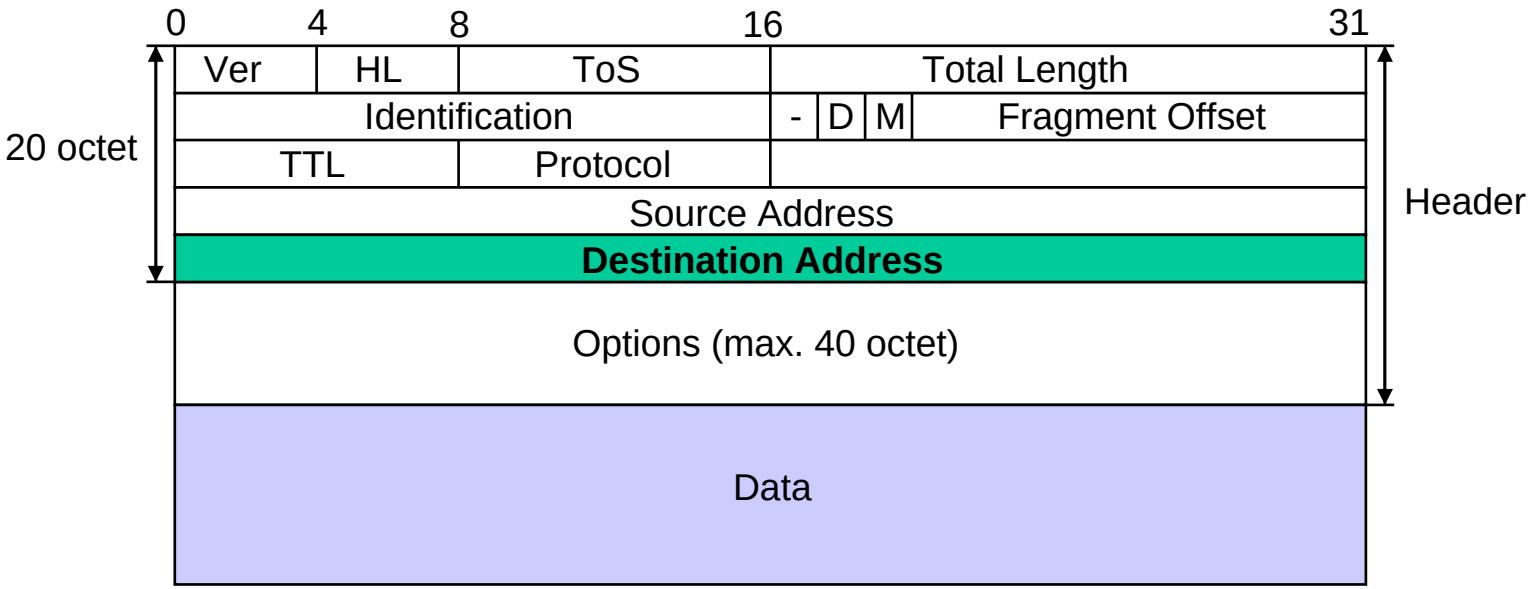
Hálózatok II
2007

3: Az IP Prefix Lookup Probléma –
A Trie adatstruktúra

Internet Protocol IP

- Az adatok a küldőtől a cél-állomásig IP-csomagokban kerülnek átvitelre
- A csomagok fejléce tartalmazza a cél IP-címét
  - IPv4: 32 Bit-címek
  - IPv6: 128 Bit-címek

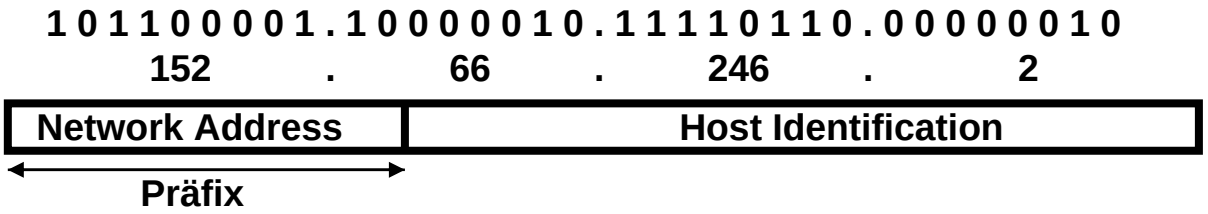


IPv4 Paket

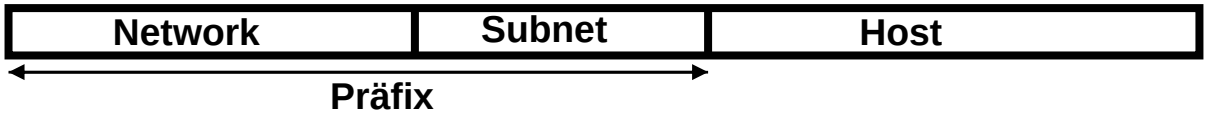
Internet Protocol IP

- Minden csomópont (router) a cél címe alapján dönt, hogy melyik szomszédos csomópontnak kell továbbítania a csomagot
  - Az ehhez szükséges információt minden csomópontban egy forwarding-tábla tárolja

Osztály alapú címzés IPv4-ben: A prefix 8, 16, vagy 24 bit hosszú lehet



Kiterjesztett hálózati cím (Extended Network Address) IPv4-ben:



- Hogyan lehet ezeket az információkat hatékonyan tárolni és megtalálni?

IP Prefix Lookup Probléma

- A címek *W* hosszúságú bináris sztringek.

Egy router forwarding-táblája *R*

- bejegyzéseket tartalmaz (*x,y*) formában
  - *x*: legfeljebb *W* hosszú bináris sztring, melynek a neve prefix.
    x lehet az üres sztring is, amit ε-nal jelölünk,
  - *y*: egy a router-ből kivezető link.
- Minden *x* bináris sztringhez *R* legfeljebb egy (*x,y*) bejegyzést tartalmaz.
- A bejegyzések száma a forwarding-táblában *N.*

Feladat:

- Egy a routerhez érkező csomaghoz, melynek cél-címe *k*,
- találjuk meg *R*-ben azt az (*x,y*) bejegyzést, melyre teljesül, hogy
  - *x* prefixe *k*-nak és
  - *x* maximális hosszúságú.
- Ezt az *x* sztringet a leghosszabb egyező prefixnek nevezzük
  (angol.: best matching prefix, röviden BMP)

IP Prefix Lookup Probléma

Példa:

- Legyen W=5.
- R bejegyzései: (0,y_1), (001,y_2), (10,y_1), (110,y_3), (111,y_2)
- Legyen a cél-cím egy csomagban k=(00100)
- A leghosszabb egyező prefix (BMP) ekkor (001,y_2)

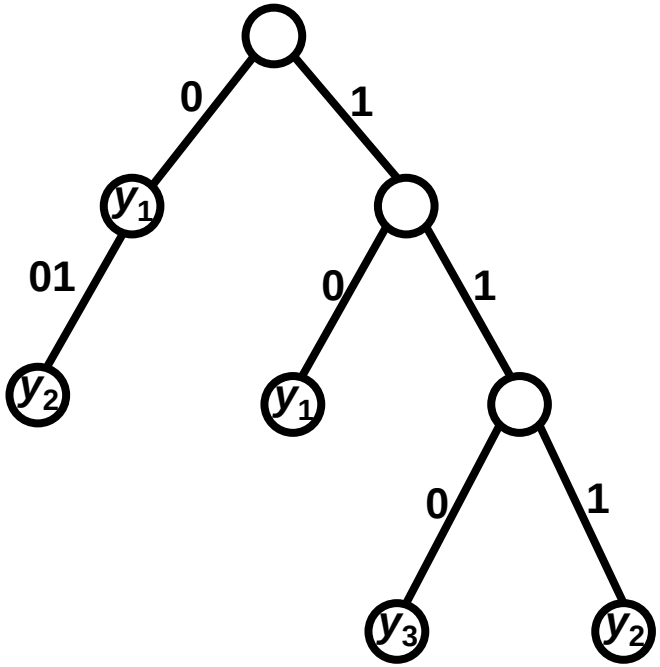
Kívánt tulajdonságok R adatstruktúrájához:

- Gyors BMP keresés (Lookup) minden csomaghoz
  - O(log W) tárhozzáférés, független N-től (a gyakorlatban 3-4 hozzáférés)
- Alacsony tárigény, azaz $O(N)$
- R aktualizálásának hatékony támogatása, azaz bejegyzések hozzáadásának, törlésének és megváltoztatásának a támogatása. Az ilyen adatstruktúrákat dinamikus adatstruktúráknak nevezzük.

IP Prefix Lookup Trie Adatstruktúrával

Egy Trie egy keresőfa, amiben

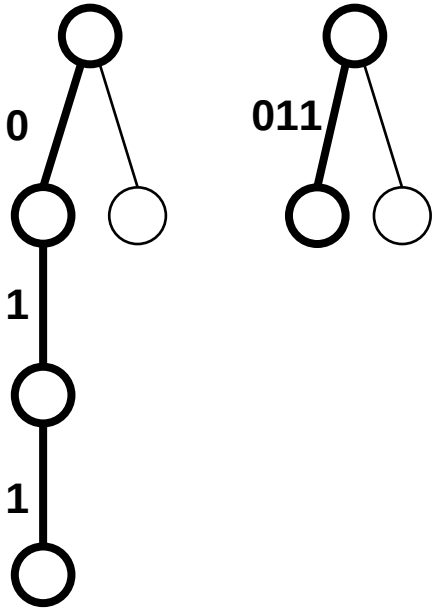
- az élek cimkézettek,
- minden *v* csomópontnál az élek cimkéi, amik *v*-t a gyermekeivel kötik össze, különböző jellel kezdődnek,
- minden *v* csomópont azt a sztringet reprezentálja, ami az élcimkék konkatenálásával az úton a gyökértől *v*-hez áll elő.



IP Prefix Lookup Trie Adatstruktúrával

Megjegyzés:

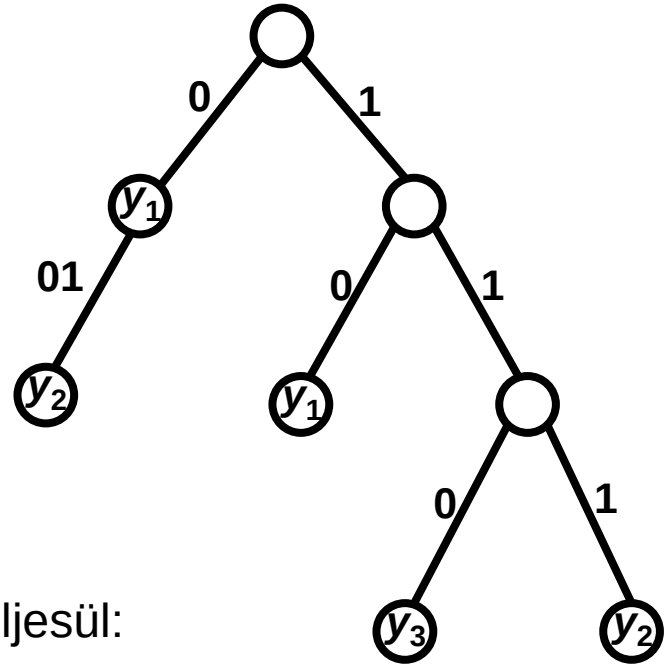
- Gyakran használnak olyan trie-t,melyben minden élcimke egyetlen jelből áll.
- Ha egy trie egy láncot tartalmaz, melyben a csomópontoknak csak egyetlen gyermeke van (és nem reprezentálnak bejegyzést a forwarding-táblában), akkor helyettesíthetjük a láncot egy egyetlen éllel, melynek cimkéje az eredeti élek cimkéinek konkatenációja.
- Ezt az operációt úttömörítésnek nevezzük. Az úttömörítés előnye: a trie csomópontok alacsonyabb száma és ezzel alacsonyabb tárigény.



IP Prefix Lookup Trie Adatstruktúrával

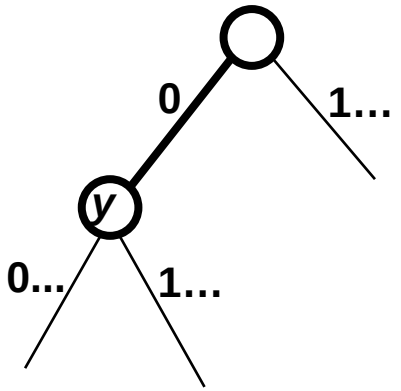
Trie mint adatstruktúra egy forwarding-tábla *R* tárolására:

- Az élcimkék bináris sztringek.
- Minden csomópontnak legfeljebb két gyermeke van.
- A cimke a baloldali gyermekhez vezető élen 0-val kezdődik, a jobboldalihoz 1-gyel.
- Minden (*x*,*y*) bejegyzéshez van a trie-ban egy csomópont, ami *x*-et reprezentálja. Ebben a csomópontban tároljuk az *y* linket.
- Minden legfeljebb *W* hosszú *x* bináris sztringhez, a trie pontosan egy csomópontot tartalmaz, amely *x*-et reprezentálja, ha a következő feltételek közül egy teljesül:
  - *x* az üres sztring, ekkor *x* a gyökérnek felel meg;
  - *R* tartalmaz egy (*x*,*y*) bejegyzést;
  - *R* tartalmaz két bejegyzést (x_a,y_a) és (x_b,y_b), úgy hogy *x*0 prefixe x_a-nak és *x*1 prefixe x_b-nek.



A Trie Konstrukciója

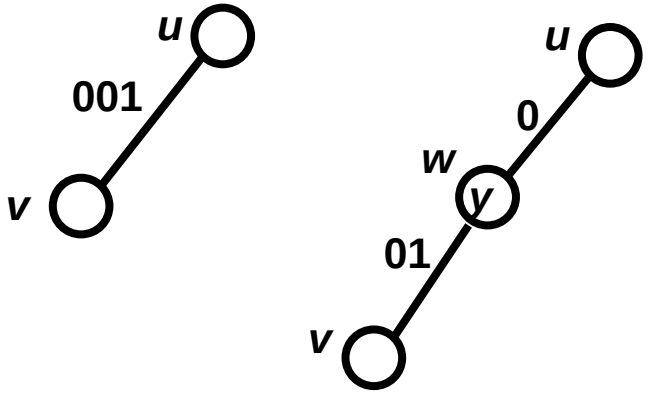
- Kezdetben a trie csak a gyökércsomópontból áll.
- Ezután *N* bejegyzést tetszőleges sorrendben befűzünk.
- Az (*x,y*) bejegyzés befűzésekor a gyökérnél indulunk és követjük az utat, amit *x* definiál:
  - Minden csomópontnál azt az élet követjük, melynek cimkéje ugyanazzal a jellel kezdődik, mint *x* fennmaradó része.
- Az út következőképp fejeződhet be:
  1. Elértük a trie egy *v* csomópontját, amikor *x*-et teljesen feldolgoztuk:
     Megjelöljük *v*-t mint csomópontot, ami *R* egy bejegyzésének felel meg és tároljuk az *y* linket *v*-ben. (Ilyen csomópontot röviden megjelöltnek nevezünk)



(*x,y*) befűzése *x*=0-val

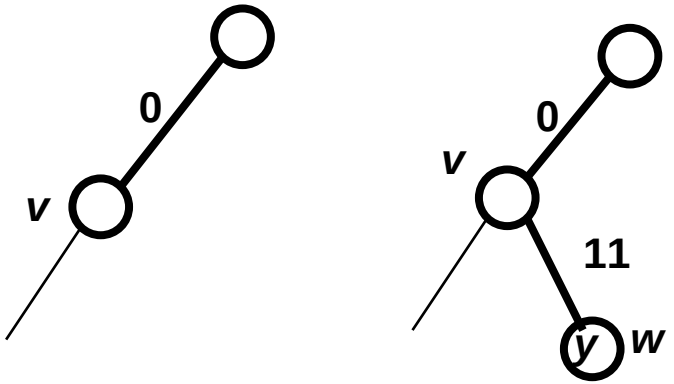
A Trie Konstrukciója

2. Az út egy (*u,v*) élen végződik, melynek cimkéje $\alpha\beta$, az α és β között, mikor *x*-et teljesen feldolgoztuk:
   Letrehozunk egy új *w* csomópontot (*u,v*) közepén és az (*u,w*) élhez az α cimkét rendeljük és a (*w,v*) élhez a β cimkét. Az *y* linket tároljuk *w*-ben és *w*-t megjelöljük.



(*x,y*) befűzése, *x*=0

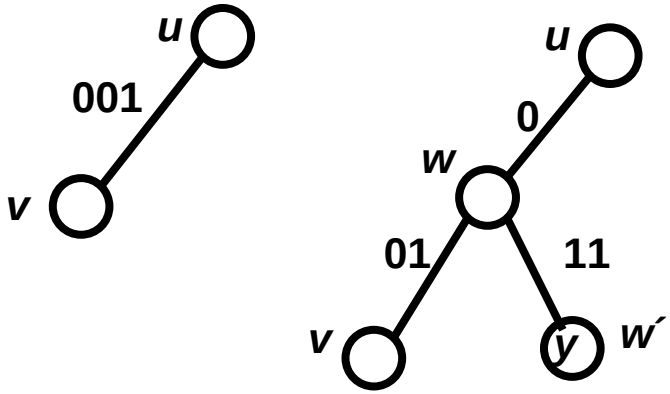
3. Az út egy *v* csomópontban végződik, mielőtt *x*-et teljesen feldolgoztuk:
   Legyen γ az *x* fennmaradó része. Létrehozunk *v*-hez egy új *w* gyermeket és az (*v,w*) élnek a γ cimkét adjuk. Az *y* linket tároljuk *w*-ben és *w*-t megjelöljük.



(*x,y*) befűzése, *x*=011

A Trie Konstrukciója

4. Az út egy (*u,v*) élen végződik, melynek cimkéje $\alpha\beta$, az α és β között, mielőtt x-et teljesen feldolgoztuk:
   Legyen γ az *x* fennmaradó része.
   A (*u,v*) élet kettéosztjuk α és β között egy új *w* csomópont befűzésével.
   Létrehozunk *w*-hez egy új *w´* gyermeket.
   Az (*u,w*) élnek az α cimkét, a (*w,v*) énak a β-t és a (*w,w´*) élnek a γ-t adjuk.
   Az *y* linket tároljuk *w´*-ben és *w´*-t megjelöljük.



(*x,y*) befűzése, *x*=011

- Minden bejegyzést be tudunk a trie-ba fűzni $O(W)$ idő alatt
- A teljes konstrukció időigénye: $O(N \cdot W)$

A Trie Tárigénye

- Minden csomópont tárigénye konstans.
- Minden él tárigénye szintén konstans.
  (Feltesszük, hogy egy *W* hoszú bináris sztring konstans sok számítógép szóban tárolható.)
- Pontosan *N* csomópont reprezentál a trie-ban egy *R*-beli bejegyzést, a trie minden más csomópontja belső csomópont, melynek pontosan két gyermeke van (máskülönben, a trie definíciója miatt, ezeket a csomópontokat nem tartalmazná a trie).
- Egy fában, melynek legfeljebb *N* levele van, legfeljebb *N*-1 belső csomópont lehet melynek (legalább) két gyereke van.
- Ezért a trie legfeljebb 2*N*-1 csomópontot és 2*N*-2 élet tartalmaz.
- Így a tárigény *O*(*N*).

Keresés a Trie-ban

- Legyen *k* a cél címe, amit keresünk.
- A gyökérben kezdünk és követjük az utat, amit *k* határoz meg. Közben mindig megjegyezzük az utolsó megjelölt csomópontot, amit már meglátogattunk.
- Ha *k*-t teljesen feldolgoztuk, vagy ha az út a trie-ban nem hosszabbítható meg *k* következő bitjével, akkor az utolsó megjelölt csomópont, amit meglátogattunk, reprezentálja a leghosszabb egyező prefixet (BMP).
- Így a BMP keresése $O(W)$ időt igényel.

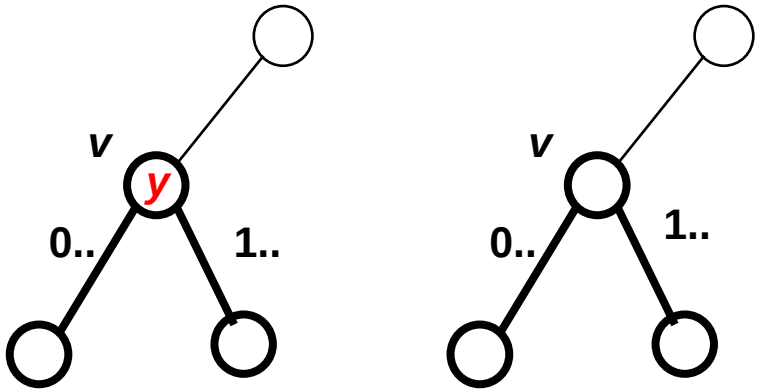
A Trie Aktualizálása

- Legyen (*x,y*) egy új bejegyzés, amit be kell fűzni.
  Ez, mint a trie konstrukciójánál, *O*(*W*) időt igényelt.

- Legyen (*x,y*) egy bejegyzés, amit törölni kell.

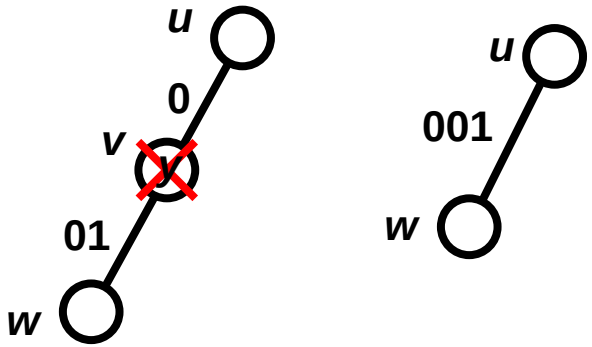
  - Megkeressük *O*(*W*) idő alatt azt a *v* csomópontot, ami (*x,y*) reprezentálja.

  - Ha *v*-nek két gyermeke van, a megjelölést töröljük, de *v* a trie-ban marad.

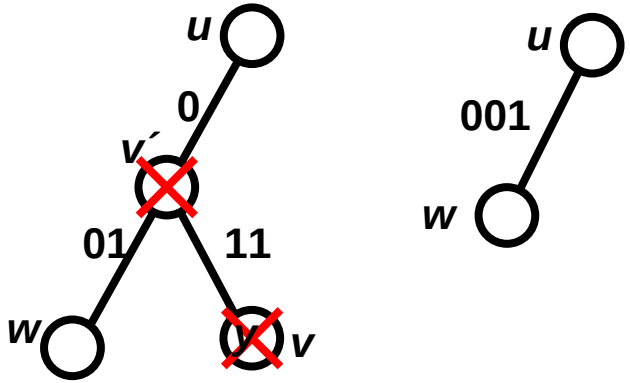


A Trie Aktualizálása

- Ha *v*-nek pontosan egy gyermeke van és *v* nem a gyökér, akkor töröljük *v*-t és az éleket, ami *v*-t a szülőjével és a gyermekével kötötte össze, egy éllé kombináljuk.



- Ha *v* egy levél, akkor töröljük *v*-t. Ha *v* szülője *v´* nem a gyökér és nem megjelölt, akkor *v´*-t töröljük és az *v´*-höz incidens két élet egy éllé kombináljuk..



A Trie Aktualizálása

- Egy bejegyzés törlése $O(W)$ időt igényel.

- Egy (x,y) bejegyzésben, ha csak a linket változtatjuk meg, akkor elég a trie-ban a bejegyzést lokalizálni és y-t megváltoztatni.
  Ez $O(W)$ időt igényel.
  (Egyébként egy bejegyzés $O(W)$ idő alatt úgy változtatható meg, hogy töröljük és újra befűzzük.)

Tétel 1: A trie adatstruktúrával a forwarding-táblát $O(N)$ tárterületen lehet tárolni. Minden keresés és aktualizálás $O(W)$ időt igényel. A trie felépítésének időigénye $O(N \cdot W)$. □