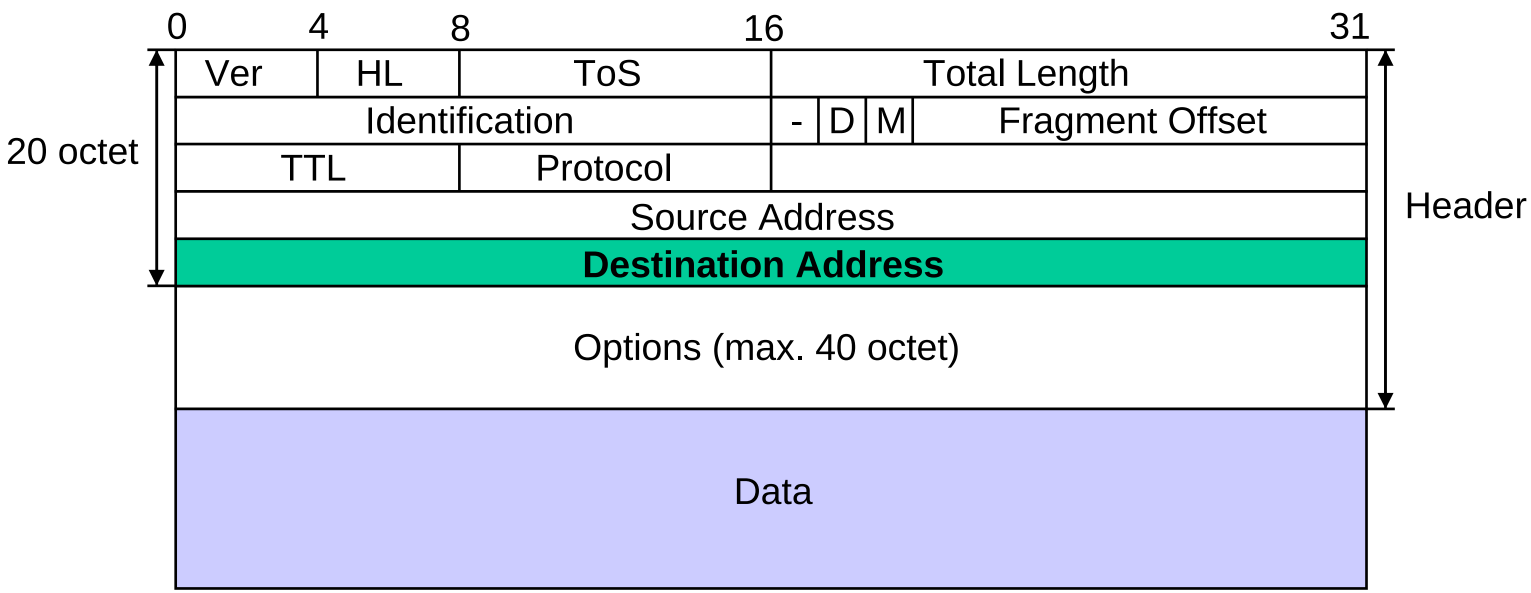
Hálózatok II
2007

4: Az IP Prefix Lookup Probléma –
Bináris keresés hosszosztályok alapján

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

1 0 1 1 0 0 0 0 1 . 1 0 0 0 0 0 1 0 . 1 1 1 1 0 1 1 0 . 0 0 0 0 0 0 1 0

152 . 66 . 246 . 2

Network Address	**Host Identification**

Präfix

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

Network	**Subnet**	**Host**

Präfix

- Hogyan lehet ezeket az információkat hatékonyan tárolni és megtalálni?

IP Prefix Lookup Probléma

- A címek *W* hosszúságú bináris sztringek.

Egy router forwarding-táblája *R*

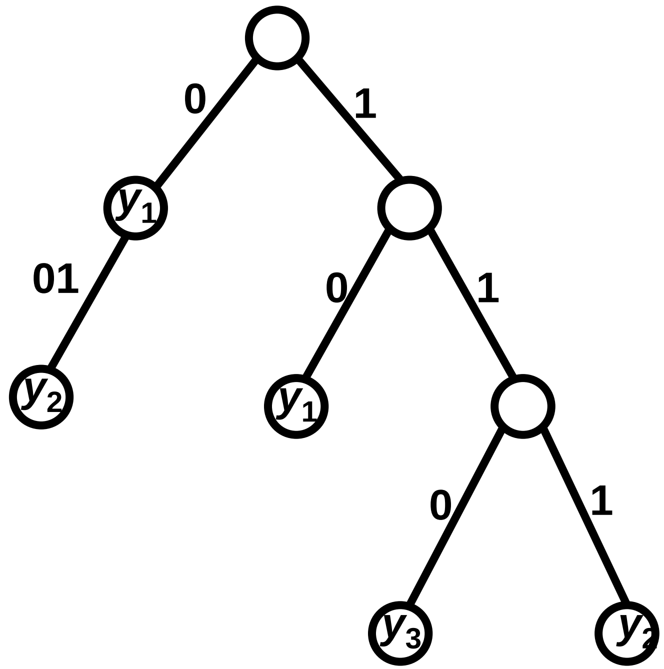
- bejegyzéseket tartalmaz (*x*,*y*) formában
  - *x*: legfeljebb *W* hosszú bináris sztring, melynek a neve prefix.
    x lehet az üres sztring is, amit ε-nal jelölünk,
  - *y*: egy a router-ből kivezető link.
- Minden *x* bináris sztringhez *R* legfeljebb egy (*x*,*y*) bejegyzést tartalmaz.
- A bejegyzések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

A Trie Adatstruktúra

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 $O(N \cdot W)$. □



IP Prefix Lookup bináris kereséssel prefixhossz alapján

[Waldvogel et al. 97]

- Alapötlet: Az IP Prefix Lookup Probléma megoldható konstans idő alatt, ha minden bejegyzés hossza megegyezik (azaz ha az *x* sztring minden bejegyzésben ugyanolyan hosszú), ha elegendő tár áll rendelkezésre.

- A forwarding-tábla bejegyzéseit a hosszuk alapján *W* osztályba soroljuk.
- Egy osztályon belül a bejegyzéseket egy olyan adatstruktúrában tároljuk, amiben a keresés *k* cím szerint konstans idő alatt lehetséges.
- Bináris kereséssel $O(\log W)$ idő alatt megkeressük azt a hosszosztályt, amely *k*-hoz a leghosszabb egyező prefixét tartalmazza.

Keresés egy hosszosztályon belül

- Notáció:
  - Egy hosszosztály L_i tárolja az R forwarding-tábla mindazon (x,y) bejegyzéseit, melyben x hossza i.
  - Legyen $N_i=|L_i|$ az ilyen bejegyzések száma.

- Van olyan adatstruktúra, amiben
  - L_i -t $O(N_i)$ tárterületen tudjuk tárolni és
  - egy adott i hosszú k bináris sztringhez $O(1)$ idő alatt el tudjuk dönteni, hogy L_i tartalmaz-e egy (k,y) bejegyzést, vagy nem.

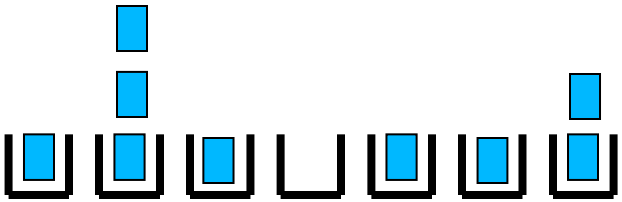
- Példa egy ilyen adatstruktúrára:
  Perfekt Hashing [Fredman, Komlós, Szemerédi 84]

Keresés egy hosszosztályon belül - Hashing

Hashing:

- A hash-táblában L_i tárolására a rendelkezésre álló indexek száma legalább $c \cdot N_i$ kell hogy legyen, ahol c>0 egy konstans.
- Egy f hash-függvény L_i minden (x,y) bejegyzéséhez hozzárendel egy $f(x)$ indexet.

- Pl. $f(x)$ = (3*x mod 23) mod 7



Keresés egy hosszosztályon belül - Hashing

Hashing:

- Egy *f* hash-függvény L_i minden (*x*,*y*) bejegyzéséhez hozzárendel egy *f*(*x*) indexet.
- Feltesszük, hogy *f*(*x*) konstans idő alatt kiszámolható.
- Ha több bejegyzés ugyanarra az indexre képeződik le, akkor kollízióról beszélünk.
- Kollíziók kezelésére egy lehetőség az összeláncolás: azaz minden *j* indexnél a prefixeket, melyek *j*-re képeződnek le, egy listába összeláncolva tároljuk.
- Ha a hash-táblában egy *k* prefixet keresünk, akkor kiszámoljuk *f*(*k*)-t és az *f*(*k*) indexnél az összeláncolt listában keresünk.
- Egy lista hosszának várható értéke *O*(1), így a keresés várhatóan *O*(1) időt igényel. (worst-case idő: $O(N_i)$)

Keresés egy hosszosztályon belül - Hashing

- Perfekt Hashing:
  - tárigénye $O(N_i)$ és
  - keresés időigénye $O(1)$ worst-case
- Alapötlet: Kétszintű hashing:
  - Az első hash-függvény f alkalmazása után kollízió léphet fel.
  - A kollíziók feloldására minden j indexnél egy második f_j hash-függvényt alkalmazunk, amely a j-re leképeződött bejegyzések számától függ, és amely címtartománya négyzetes, azaz, ha n_j a bejegyzések száma, amiket f a j -re képezett le, akkor $f_j(k) \in [1, c \cdot n_j^2]$.
  - Ha f és f_j véletlen lineáris függvények, akkor konstans valószínűséggel (pl. 1/2 valószínűséggel) nem lesz kollízió, és a szükséges tár $O(N_i)$.
  - Így a „függvények fele jó“. Ha egy véletlen funkcióról kiderül, hogy nem rendelkezik az elvárt tulajdonsággal, akkor választunk véletlenül egy másikat.

Perfekt Hashing

- Perfekt Hashing:
  - Minden hash függvényhez van worst-case input, ezért hash-függvények családját tekintjük:
    - Legyen s a hash-tábla mérete,
    - $U = \{1,\ldots,m\}$ (univerzum)
    - $S \subseteq U$, $|S| = n$, az input
    - $H_s = \{\, h : U \rightarrow \{1,\ldots,s\} \mid h(x)= (kx \bmod p) \bmod s,\ 1 \leq k < p \,\}$ ahol p prím, $p > m$.
  - Az első hash-függvény s partícióra osztja az elemeket
  - A partíciókat egy második hash függvénnyel tároljuk kollízió nélkül
  - Válasszuk h-t véletlenül H_s-ből egyenletes eloszlás szerint
  - Legyen $W_j^h = \{\, x \in S \mid h(x) = j \,\}$

Perfekt Hashing

Lemma 1: (bizonyítás nélkül) Ha h-t egyenletes eloszlás szerint véletlenül válaszjuk H_s-ből, akkor

$$E\left(\sum_{1\leq j<s}\binom{|W_j^h|}{2}\right)\leq\frac{n(n-1)}{s}$$

Ahol $E(X)$ az X véletlen változó várható értéke. □

Következésképpen:

$$\Pr\left(\sum_{1\leq j<s}\binom{|W_j^h|}{2}<\frac{2n(n-1)}{s}\right)\geq\frac{1}{2} \qquad (1)$$

Ha $s = 2(n-1)$, akkor ebből következik, hogy legalább a $h\in H_s$ függvények felére:

$$\sum_{1\leq j<s}\binom{|W_j^h|}{2}<n$$

Egy ilyen függvényt használunk S partícionálására W_j blokkokra, $1\leq j < s$.

Perfekt Hashing

Minden W_j blokkhoz, $1 \leq j < s$, válasszuk $s_j = \max\{ 1, 2|W_j|(|W_j|-1) \}$,

Ekkor (1) szerint legalább a $h_j \in H_{s_j}$ függvények felére érvényes:

$$\sum_{1 \leq j < s} \binom{|W_{j,\ell}^{h_j}|}{2} < 1$$

ahol $W_{j,\ell}^{h_j} = \{x \in W_j \mid h_j(x) = \ell\}$

Azaz legalább a $h_j \in H_{s_j}$ függvények fele injektív.

Egy ilyen függvényt használunk W_j tárolására. Így nem fordul elő kollízió.

A teljes tárígény:

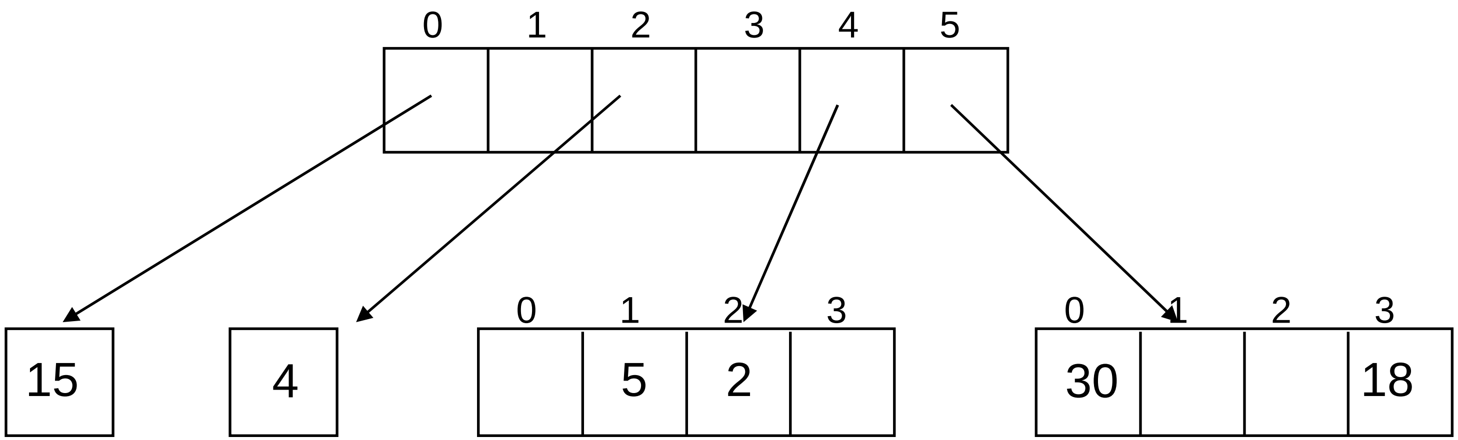
$$\sum_{1 \leq j < s} s_j \leq s + 4 \cdot \sum_{1 \leq j < s} \binom{|W_j|}{2} = O(n)$$

Perfekt Hashing Példa

m=30, p=31, s=6, S={2,4,5,15,18,30}

$h(x) = (2x \bmod 31) \bmod 6$

$h_4 = (1x \bmod 31) \bmod 4$, $h_5 = (3x \bmod 31) \bmod 4$,



Bináris keresés prefixhossz szerint

- Első ötlet:
  - Kezdjünk a „középső“ hosszosztállyal $L_{W/2}$-vel. $O(1)$ idő alatt teszteljük, hogy $L_{W/2}$ tartalmazza-e k prefixét.
  - A teszt eredményétől függően, keressünk tovább rekurzívan a $L_1,..,L_{W/2-1}$ (ha nem) vagy a $L_{W/2+1},\ldots,L_W$ (ha igen) osztályokban.
  - Ehhez $O(\log W)$ teszt szükéges és így $O(\log W)$ idő.

- Sajnos ez az ötlet így még nem működik. Pl.:
  - Legyenek 0, 1, 00, 111 a prefixek R-ben.
    Ekkor L_1={0,1}, L_2={00}, L_3={111}.
  - Keressünk a 11101 célcím szerint.
    - L_2-vel kezdünk: a prefix 11 nincs benne L_2-ben, így a rövidebb prefixek között keresnénk L_1-ben, pedig a leghosszabbb egyező prefix 111 L_3-ban van.

Bináris keresés prefixhossz szerint

- Első módosítás:
  - Legyen *x* egy *i* hosszú prefix és jelölje $x[1..j]$ az *x* első *j* jelét, $1 \leq j \leq i$.
  - Ha *x* az L_i hosszoszályban van, akkor tárolunk minden L_j hosszosztályban, $1 \leq j < i$, amelyben nincs bejegyzés $x[1..j]$-hez, egy u.n. jelzést $x[1..j]$-hez.
    - A jelzés biztosítja, hogy egy keresésnél $x[1..j]$-nél felismerjük, hogy *R*-ben van egy hosszabb bejegyzés, ami $x[1..j]$-vel kezdődik. Így a bináris keresés a helyes döntést tud hozni, miszerint a hosszabb prefixek között keres tovább.
    - Az előző példában: L_1={0,1} L_2={00,11}, L_3={111}.

Bináris keresés prefixhossz szerint

- Van még egy probléma:
  - Ha a példánkban a 11010 célcím szerint keresünk, a 11 jelzés L_2-ben ahhoz vezetne, hogy a keresés L_3-ban folytatódik.
  - Mivel L_3-ban nem találunk 110 bejegyzést, a keresés visszatérne L_2-höz.
  - Mivel 11 csak egy jelzés, nem valódi bejegyzés, a keresést L_1-nél kellene folytatni. Ez a „backtracking“ $O(\log^2 W)$ futási időt eredményezne.

Bináris keresés prefixhossz szerint

- Második módosítás:
  - Minden *x* jelzésnél tárolunk egy *x.BMP* mutatót:
  - *x.BMP* arra az *R*-beli (*x´*,*y´*) bejegyzésre mutat,
    - ahol *x´* egy prefixe *x*-nek és
    - *x´* maximális hosszúságú.
  - Ha a bináris keresés egy *x* jelzést talál meg, mint leghosszabb egyező prefix, akkor *x.BMP* megadja a valódi leghosszabb egyező prefixet.
    - A példánkban 11.*BMP* az 1-re mutat L_1-ben.
    - Ha a 11010 célcím szerint keresünk, akkor 11.*BMP* megadja a leghosszabb egyező prefixet 1-et.

- Ezen kiegészítések után a bináris keresés helyesen működik:
  - A worst-case keresési idő $O(\log W)$.
  - Ha *R*-ben kevesebb mint *W* különböző hossz fordul elő, mondjuk *r*, akkor a worst-case keresési idő $O(\log r)$.

A forwarding-tábla tárigénye

- A jelzések bevezetése nélkül $O(N)$ tár.

- Ha egy L_i-beli x prefixhez minden L_j,$1 \leq j < i$, hosszosztályban tárolunk egy jelzést, ha L_j még nem tartalmaz bejegyzést $x[1..j]$-hez, prefixenként $O(W)$ jelzésre lehet szükség.
- Így az adatstruktúra társzükségletére csak $O(W \cdot N)$ korlátot tudunk adni.
- Ha a jelzéseket csak azokban a hosszosztályokban tároljuk, ahol tényleg szükséges (amiket a bináris keresés tényleg tesztel), akkor a korlát $O(N \log W)$.

A forwarding-tábla konstrukciója

- Az adatstruktúra felépítéséhez szükséges idő lineáris a tárolt értékek (bejegyzések+jelzések) számával, feltéve, hogy a táblákat L_i-khez $O(|L_i|)$ idő alatt fel tudjuk építeni. Ez érvényes hash-táblák esetén.
  - R bejegyzéseit prefixhossz szerint növekvő sorredben dolgozzuk fel.
  - Egy i hosszú x prefixet befűzzük L_i táblájába.
  - Ezután befűzzük a jelzéseket az L_j táblákba, $j < i$, hossz szerint csökkenő sorrendben, amíg egy tábla nem tartalmaz egy igazi bejegyzést a megfelelő prefixhez.

A forwarding-tábla aktualizálása

- Egy (*x,y*) bejegyzés hozzáfűzése:
  - Először (*x,y*)-t befűzzük L_i-be, ahol *i* az *x* hossza.
  - Az L_j, $j < i$, táblákban be kell fűzni esetleg egy jelzést.
  - Minden *x´* jelzésnél a L_j, $j > i$, táblában, amihez *x* a leghosszabb egyező prefix, az *x´.BMP* mutatót (*x,y*)-ra kell állítani.
  - Az aktualizálás nagyon időigényes lehet, mivel lehetséges, hogy sok jelzést kell aktualizálni.
  - Egy bejegyzés törlésekor a szituáció hasonló.
  - Tanács: Egy darabig gyüjtsük az aktualizálásokat és utánna építsük fel újra a teljes adatstruktúrát.

Irodalom

- Marcel Waldvogel, George Varghese, Jon Turner, Bernhard Plattner: **Scalable High Speed IP Routing Lookups**. *Proc. ACM SIGCOMM '97,* 25-36, 1997.

- Michael L. Fredman, János Komlós, and Endre Szemerédi: **Storing a sparse table with O(1) worst case access time**. *Journal of the ACM*, Vol. 31(3), 538--544, 1984.

- Martin Dietzfelbinger, Anna R. Karlin, Kurt Mehlhorn, Friedhelm Meyer auf der Heide, Hans Rohnert, and Robert E. Tarjan: **Dynamic perfect hashing: upper and lower bounds**, *SIAM Journal on Computing*, Vol. 23(4), 738-761, 1994.