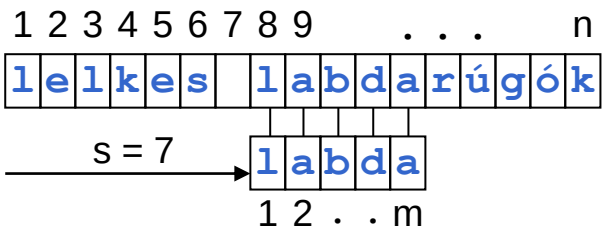
Mintaillesztő algoritmusok

- **Feladat:** egy adott P minta összes előfordulásának megtalálása egy adott T szövegben. P és T karakterei egyaránt a véges elemű Σ ábécé jelei.

- **Input:**
  - T szöveg, pl.: T= ”`lelkes labdarúgók`”
    - n = hossz(T) = 17
  - P minta, pl.: P= ”`labda`”
    - m = hossz(P) = 5

- **Output:**
  - s eltolásértékek, melyeknél a P minta a T szövegben megtalálható: T[s+1..s+m] = P[1..m] 0 =< s <= n-m;



Forrás: 1. *Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)*
2. *Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999*

Vizsgált algoritmus típusok

- **Egyszerű mintaillesztő algoritmus**
- **Rabin-Karp mintaillesztő algoritmus**
- **Knuth-Morris-Pratt mintaillesztő algoritmus**

Egyszerű mintaillesztő algoritmus

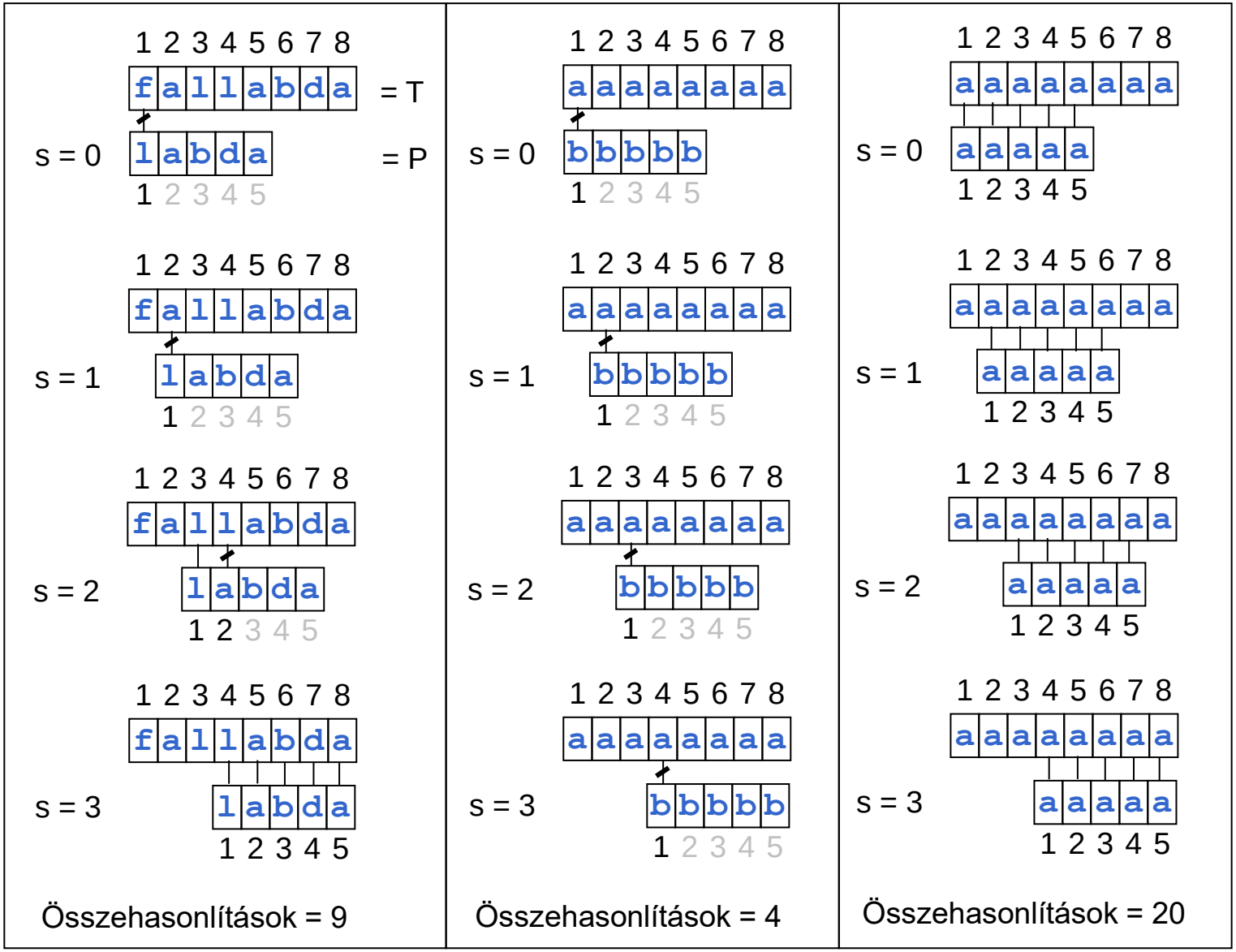
- Minden s eltolást megvizsgál a 0<= s <=n-m intervallumban.

- Egyszerű_mintaillesztő(T, P)

```
    n ← hossz(T)
    m ← hossz(P)
    for s ← 0 to n-m do
        j ← 1
        while j<=m and T[ s + j ] = P[ j ] do
            j ← j + 1
        if j = m+1 print s
```

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Példák az Egyszerű mintaillesztő algoritmus működésére



Az Egyszerű mintaillesztő algoritmus elemzése

- **Legrosszabb eset** (legtöbb összehasonlítás)
  - For ciklus: (n-m+1)
  - While ciklus: (m) elem-összehasonlítás
  - Összes elem-összehasonlítás: (n-m+1)*m
  - Elem-összehasonlítás arányos futási idő $\Theta((n-m+1)*m)$

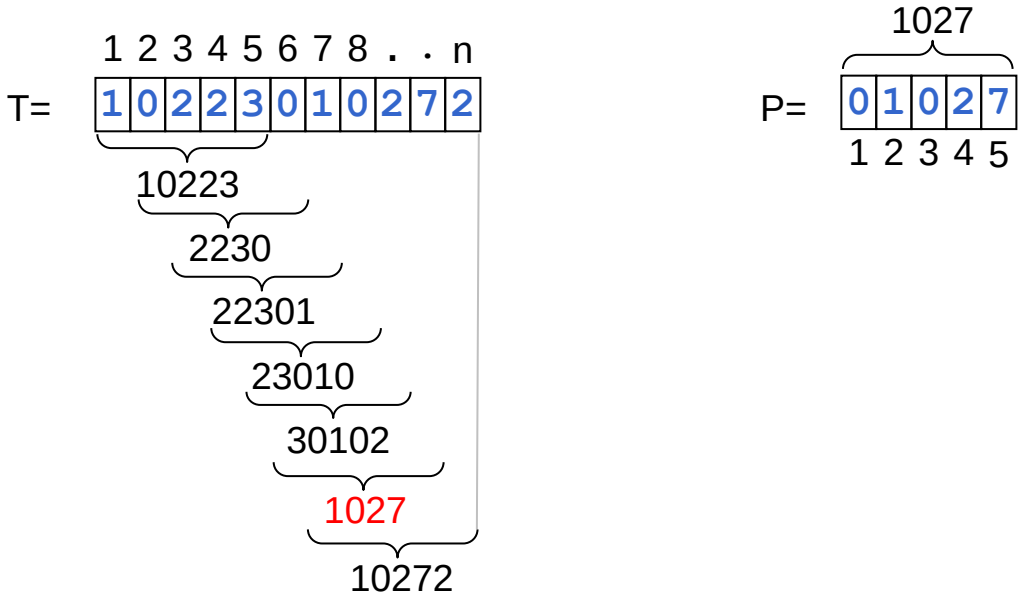
- **Legjobb eset** (legkevesebb összehasonlítás)
  - For ciklus: (n-m+1)
  - While ciklus: (1) elem-összehasonlítás
  - Összes elem-összehasonlítás: (n-m+1)*1
  - Elem-összehasonlítás arányos futási idő $\Theta((n-m+1))$

- **Átlagos eset**
  - A szöveg és a minta karakterei egyenletes véletlen eloszlásúak
  - $O(n-m)$, elég hatékony Lásd 2.

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Rabin-Karp mintaillesztő algoritmus

- **Alapgondolat:** ha a szöveg és a minta is csak számokból állna, valamint a minta számjegykaraktereit egyetlen többjegyű számnak tekinthetnénk, hasonlóan a szöveg mintahossznyi részeit is egy-egy számértékkel reprezentálhatnánk, akkor a mintakeresés n-m+1 lépésben elvégezhető lenne.



Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Rabin-Karp mintaillesztő algoritmus ..

- **Jelölések:**
  - **Az ábécé** $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$, $|\Sigma| = 10$
  - A minta számjegykaraktereit számértékre átszámító függvény:
    f(P) a Horner séma alkalmazásával O(m) időben kiszámítható,
    f(P)= P[m] +10*(P[m-1] + 10*(P[m-2] +…+ 10*(P[2] + 10*(P[1]))…))
    Pl.: f(”01027”)= 7+10*(2+10*(0+10*(1+10*(0)))) = 1027
  - Ha adott, vagy az előző módszerrel számítva rendelkezésre áll a T szöveg s eltolású, mintahosszúságú részének számértéke f_s, akkor f_{s+1} O(1) idő alatt számítható:
    $f_{s+1}=10*(f_s - 10^{m-1} * T[s+1]) + T[s+m+1]$

    Pl.: T=”10$\overbrace{\underbrace{2230}_{f_{s+1}}1}^{f_s}$8272; s=2; f_s= 22301; f_{s+1}= 10*(22301-10^4*2)+8= 23018

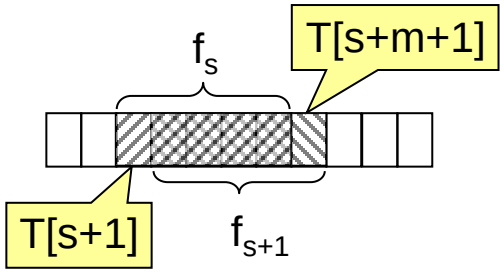
    A teljes T szöveg értékekre való átalakításának ideje tehát O(m+n-m)
    Mivel a keresés O(n-m+1) idejű, a teljes algoritmus ideje
    = O(m+ m+n-m + n-m+1) = O(2n+1)

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Rabin-Karp mintaillesztő algoritmus ..

- **Az algoritmus**

```
Rabin_Karp(T, P)
  fp ← f(P)
  fs ← f(T[1..m])
  for s ← 0 to n – m do
     if fp = fs print s
     if s<n-m then
        fs =10*(fs – 10^(m-1) * T[ s+1 ]) + T[ s+m+1 ]
```



- **Probléma:** ha m elég nagy, az f() függvényértékek ábrázolása külön algoritmust igényel, f() kiértékelése nem O(1) idejű.

- **Megoldás:** az f() értékek helyett számítsuk és használjuk azok q-ra vett modulóját, q célszerűen olyan prímszám, hogy $|\Sigma|*q$ még éppen ábrázolható.
  Ha fp **mod** q ≠ fs **mod** q $\Rightarrow$ fp ≠ fs de fp **mod** q = fs **mod** q $\not\Rightarrow$ fp = fs .
  Pl.: 65 **mod** 11 = 10 ≠ 89 **mod** 11 = 1 $\Rightarrow$ 65 ≠ 89
  de 65 **mod** 11 = 10 = 87 **mod** 11 = 10 $\not\Rightarrow$ 65 = 87, azaz a maradékegyezések további vizsgálatot kívánnak.

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Meggondolások

- **Felhasznált modulo azonosságok**
  - (a+b) **mod** q = (a **mod** q + b **mod** q) **mod** q
  - (a* b) **mod** q = (a **mod** q)*(b **mod** q) **mod** q

- **Előfeldolgozás**
  - d = |Σ|, ‘számrendszer alapszáma’ = ábécé elemszáma, q >d
  - fpm = (P[m] + d*(P[m-1] + d*(P[m-2]+ …
    …+ d*(P[2] + d*P[1])…))) **mod** q =
    (d*((d*… ((d*((d*0+P[1]) **mod** q)+P[2]) **mod** q)+…
    …+ P[m-1]) **mod** q)+P[m]) **mod** q
  - Hasonlóan számítandó fsm a T[1..m] szövegrészletből.
  - Pl.: P=”1234”; d=10; q=11; 1234 **mod** 11 = 2 =
    (10*((10*((10*(1 **mod** 11)+2) **mod** 11)+3) **mod** 11)+4) **mod** 11 = 2

- **Keresés**, T[s+1] kiléptetése, T[s+m+1] beléptetése
  - fsm_{s+1}=(d*(fsm_s – h* T[s+1]) + T[s+m+1]) **mod** q,
    ahol a h= d^{m-1} **mod** q számítható az előfeldolgozásban.

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

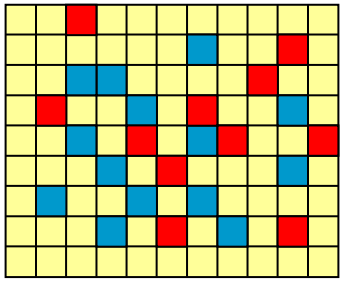
A módosított Rabin-Karp mintaillesztő algoritmus

```
Rabin_Karp_Modulo(T, P, d, q)
    m ←hossz(P)
    n ← hossz(T)
    h ← d^(m-1) mod q      // valójában (d mod q) szorzása önmagával ciklusban
    fpm ← 0                // a minta értékének modulója, inicializálás
    fsm ← 0                // a szöveg s eltolású része értékének modulója, inic.
    for i ← 1 to m do
        fpm ← (d*fpm+P[ i ]) mod q
        fsm ← (d*fsm+T[ i ]) mod q
    for s ← 0 to n-m do            //  keresés
        if fpm=fsm then            // csak lehetőség, ellenőrizni kell
            j ← 1                  // szerencsére csak ritkán kell ellenőrizni
            while j<=m and T[ s + j ] = P[ j ] do
                j ← j + 1
            if j = m+1 print s
        if s < n-m then
            fsm ← (d*(fsm – h* T[ s+1 ]) + T[ s+m+1 ]) mod q      // léptetés
```

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

A módosított Rabin-Karp mintaillesztő algoritmus időkomplexitása

- **Legrosszabb eset**
  - Csak egyféle karakter a mintában és a szövegben
  - Futási idő $\Theta((n-m+1)*m)$, minden eltolást végigellenőriz
- **Általános eset**
  - Általános esetben az érvényes eltolások száma kicsi
  - Futási idő $O(n) + O(m*(v+n/q))$, ahol v az érvényes eltolások száma
  - Ha $q >= n$, ez az idő $O(n)$
- **Megjegyzés**
  - Az algoritmus könnyen kiterjeszthető kétdimenziós mintakeresésre

T=  P=

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

Knuth-Morris-Pratt mintaillesztő

- Cél: a szöveg minden egyes karakterét **csak egyszer** felhasználni összehasonlításban
- Az Egyszerű mintaillesztő **elfelejti** a külső ciklus előző lépésében talált részleges illeszkedés során birtokába került információkat:

Pl.: T="**kirándulnak a kirándulók a hegyekbe**"
P="**kiránduló**"
Mivel csak egy 'k' van P-ben, az első eltérést adó pozícióba ugorhatunk az eltolással.

T="**blablablablalakban**"
P="**blablalak**"
Mekkora a legnagyobb ugrás P eltolásában, mely nem hagy ki esetleges egyezést? Toljuk rá hátulról a P mintát a T text feltárt részére, hogy a legnagyobb egyezést kapjuk:
T="**blablablablalakban**"
P="**blablalak**"

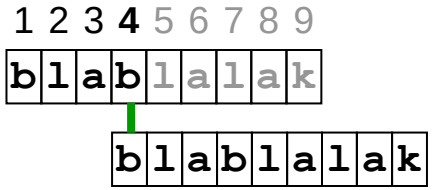
- Ezek, a mintának önmaga kezdeteire hátulról való rátolhatóságára vonatkozó információk az előkészítési fázisban előállíthatók!

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

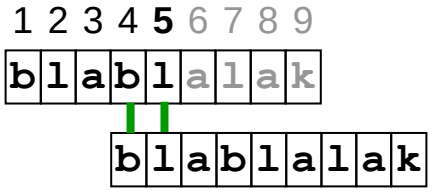
A π prefix függvény

- A π(q) prefix függvény megadja, hogy a P minta q hosszúságú első részére a mintát hátulról nem teljesen rátolva az legfeljebb hány karakterrel hozható fedésbe.
  π(1) legyen 0.
- **Példa:**
  P="**blablalak**"

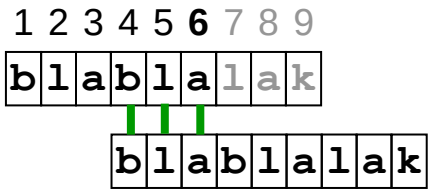
π(1) = 0
π(2) = 0
π(3) = 0
π(4) = 1



π(5) = 2



π(6) = 3



π(7) = 0

π(8) = 0

π(9) = 0

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

A π prefix függvény algoritmusa

- A prefix függvény értékei a P minta ismeretében az előkészítő fázisban meghatározhatók

- Prefix_függvény_számítás(P)

```
m ← hossz(P)
π[1] ← 0
k ← 0
for q ← 2 to m do
    while k>0 and P[k+1] ≠ P[q] do
        k ← π[k]
    if P[k+1] = P[q] then
        k ← k+1
    π[q] ← k
return π
```

- A Prefix függvény számítás futási ideje O(m).

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999

A Knuth-Morris-Pratt mintaillesztő algoritmusa

- A prefix függvény értékeinek ismeretében az algoritmus átugorhatja azon eltolások vizsgálatát, melyek eleve érvénytelenek.

- KMP_mintaillesztő(P,T)

```
    m ← hossz(P)
    n ← hossz(T)
    π ← Prefix_függvény_számítás(P)
    q ← 0
    for i ← 1 to n do
        while q > 0 and P[q+1] ≠T[ i ] do
            q ← π[ q ]          // ugrás π függvényében
        if P[q+1] = T[ i ] then
            q ← q+1             // lépegetés, amíg egyezik
            if q = m then
                print i-m+1      // talált ennél az eltolásnál
                q ← π[ q ]       // ugrás π függvényében
```

- A KMP mintaillesztő futási ideje O(m+n), igen kedvező.

Forrás: 1. Simonas Šaltenis: Advanced Algorithm Design and Analysis (Lecture 3)
2. Cormen-Leiserson-Rivest: Algoritmusok Műszaki Könyvkiadó, 1999