3

Keresés sztringben csúszóablakkal

Ciuciu-Kiss Jenifer Tabita
Programtervező informatikus BSc

2018. január 22.

Az EFOP-3.6.3-VEKOP-16-2017-00002. számú projektben elvégzett szakmai feladat az Európai Unió támogatásával, az Európai Szociális Alap társfinanszírozásával valósult meg.

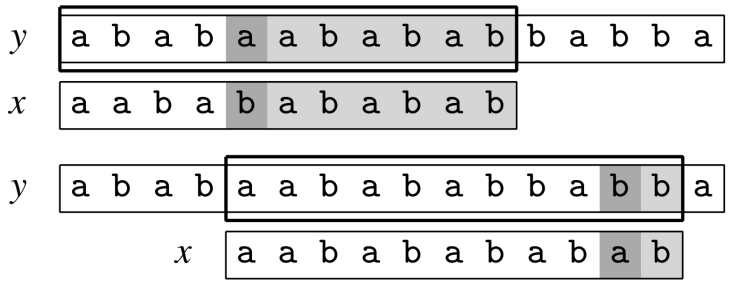
Tartalomjegyzék

1. Bevezetés

Azt vizsgáljuk, hogy hogyan kell karakterláncokat keresni egy hosszabb szövegben. Erre több megoldást is láthatunk majd, melyek egyre több memóriafelhasználással ugyan, de egyre gyorsabban müködnek.

2. Keresés egy memóri felhasználásával

Nézzük meg, hogy kell egy szövegben karakterláncot keresni hatékonyabban a naív, mindent mindennel összehasonlító algoritmusnál. A keresés során megjegyzünk legalább egy információt az előző egyezésről, jelen esetben az utolsó olyan szuffixet a sztringben ami a szövegben is előfordult. Ezt a technikát "faktor memorizációnak" nevezzük, ami kiterjeszti a prefix memorizációs implementációt, amit a szakirodalom WL-Memoryless-Suffix-Search -nek nevez. Ennek a keresésnek szüksége van egy konstans méretű extra memóriára a feldolgozandó szövegtől függően, hasonlóan az egyszerű Memoryless-Suffix-Search. A keresés működése azonban így már nem kvadratikus, hanem 2n-es, ahol n jelöli annak a szövegnek a hosszát amelyben keresünk.



1. ábra. Látható, hogy a 10-es pozícióban végzett kísérlet során felismertük az $ababab$ szuffixet. Ezután eltoljuk az ablakot 4 pozícióval, és megjegyezzük, hogy az $abababab$ szuffix a szövegben 4 periódusú. Így az y tényező $aababab$ értéke megyegyezik az x-hez igazított tényező értékével. A következő lépésben felismerjük a b utótagot, az $y[9] = a$ és az $y[13] = b$ azt mutatják, hogy a szövegnek nincs 4-es periódusa. Tehát az $abababab$ szuffixe a 4-et ismeri fel periódusként, de az $y[9] = a$ és az $y[13] = b$ azt mutatják, hogy nem lehet őket egyidejűleg igazítani. Ez ahhoz vezet, hogy az eltolás $|ababab| - |b| = 5$ hosszú lesz.

2.1. Keresési fázis

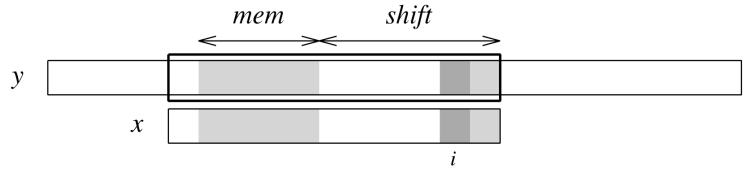
A Memoryless-Suffix-Search mindegyik eltolás után eldobja az korábbi kísérletek során összegyűjtött infomációkat. A jelenlegi algoritmus működését úgy gyorsítjuk, hogy megjegyezzük az x egy szuffixének utolsó előfordulását az y szövegben. Ennek az információnak a memorizációja több előnnyel is jár:

$\rightarrow$ lehetővé teszi a már biztosan nem jó tényezők átugrását
$\rightarrow$ lehetővé teszi az eltolások meghosszabítását

Az ebben az esszében ismertetett algoritmus nagy részben kihasználja ezeket a lehetőségeket. Az egyetlen tényező memorizálása egy pontosan adott szituációban lehetséget. Az eltolás hosszának növelését a szakirodalom turbo-shiftnek nevezi, magyarosítva turbó-eltolásnak nevezik.

A következőkben részletesebben ismertetve lesz maga az algoritmus. Az általános szituáció egy *T* kísérlet során a keresési fázisban Turbó-szuffix-keresés során bemutatásra került az 1. Az előző kísérlet legyen *T'*, a *j'* pozición , a z' szuffix lett felismerve a szövegben, és az eltolás hossza a következő lesz: $d' = bestfact(m-1-|z'|,\ y[j'-|z'|])$.

A *T* próbálkozás során a $j = j' + d'$ pozícióban egy ugrás z' elem felett az *y* sztringben lehetséges, ha a *d'* hosszú szuffix felismerhető az *y* sztringben. Ebben a esetben, felesleges össehasonlítani az z' elemet a a szövegbol a keresett karakterlánccal, mert az eltolás definíciója alapján azok biztosan meg fognak egyezni. A turbó-eltolás használható, ha a z felismert szuffix az aktuális próbálkozásban rövidebb, mint z'. Tehát a turbó-eltolás hossza: $|z'| - |z|$.



2. ábra. Az következő változók: *i*, *mem* és *shift*, az instukciók végrehajtása során a Turbó-Szuffix-Keresés nevű, algoritmus 10-es sorában lévő adatok alapján láthatóak. A világos szürke rész az egyezést jelöli, a sötét szürke rész pedig azt amelyik nem egyezik.

Abban az esetben amikor a turbó-eltolás nagyobb mint a $bestfact(m-|z|,\ y[j-|z|])$, megjegyezzük, hogy az eltolás a jelenlegi lépésben lehet nagyobb, mint $|z|$. Az adott tényező megjegyzése csak akkor lehet végleges, ha a $bestfact$ általi eltolás már ki lett számolva. Az metódus helyessége az argumentumok periodicitásán alapul.

Most következik maga a Turbó-Szuffix-Keresés algoritmusa. Az algoritmusban hivatkozunk a korábban már definiált $bestfact$ értékére, de a $bestfact$ helyett a $goodsuff$ is használható az eltolás hosszának kiszámítására.

2.2. Tétel

Az *x* és *y* sztringekre alkalmazott Turbó-Szuffix-Keresés megtalálja *x* összes előfordulását *y*-ban

```
TURBO-SUFFIX-SEARCH(x, m, y, n)
 1  shift ← 0
 2  mem ← 0
 3  j ← m − 1
 4  while j < n do
 5        i ← m − 1
 6        while i ≥ 0 and x[i] = y[j − m + 1 + i] do
 7              if i = m − shift then
 8                    i ← i − mem − 1      ▷ Jump
 9              else  i ← i − 1
10        OUTPUT-IF(i < 0)
11        if i < 0 then
12              shift ← per(x)
13              mem ← m − shift
14        else  turbo ← mem − m + 1 + i
15              if turbo ≤ best-fact(i, y[j − m + 1 + i]) then
16                    shift ← best-fact(i, y[j − m + 1 + i])
17                    mem ← min{m − shift, m − i}
18              else  shift ← max{turbo, m − 1 − i}
19                    mem ← 0
20        j ← j + shift                    ▷ Shift
```

2.2.1. Bizonyítás

A különbseg a Memoryless-Suffix-Search és a Turbó-Szuffix-Keresés között azon alapul, hogy az eltolás hosszát növeljük, és a már biztosan egyező részeket nem hasonlítjuk össze. Ez elég, hogy belássuk, hogy a 18-as sorban kiszámolt eltolás helyes. Először lássuk be azt, hogy a turbó-eltolás (turbo változóként jelölve) hossza helyes. Ekkor belátjuk, hogy az $m-1-i$ hosszú eltolás szintén helyes. Jegyezzük meg, hogy a 18. sor akkor hajtódik végre, ha $turbo > bestfact(i,\ y[j-m+1+i])$, ami implikálja azt, hogy $turbo > 1$.

A *mem* változó értéke a felismert z' szuffix hosszát jelöli az előző, *T*' lépésben. A felismert $z = x[i+1 \ldots m-1]$ szuffix hossza a *T* lépésben $m-1-i$. A $turbo$ változó hossza $|z'|-|z|$. Legyen $a = x[i]$ az a karakter, amely megelőzi a z szuffixet a szövegben, és legyen $b = y[j-m+1+i]$ az a karakter, amelyik megelőzi a megefelelő előfordulását z-nek a szövegben. Legyen $u = x[m-d \ldots i]$, (tehát van egy $z'uz \preceq_{szuff} x$). Amikor z

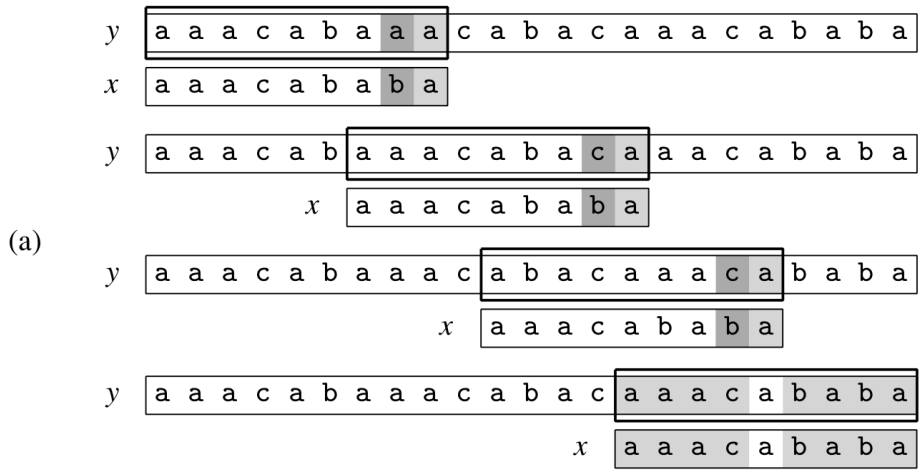
rövidebb mint z', akkor az egy szuffix a z'-ben. Ebből következi, hogy az a és b karakterek előfordulnak a $d = |uz|$ távolságra egymástól a szövegben. De a $z'uz$ szuffix a sztringben is rendelkezik a $d = |uz|$ periódussal (mert z' egy keret $z'uz$-ben), tehát az eltolások, amelyek rövidebbek, mint $|z'| - |z| = turbo$ feleslegesek, hiszen nem vezetnek karakter különbözéshez. Tehát a $turbo$ hosszó eltolás szabályos. Lásd 1.

Most lássuk be, hogy a $|z| = m - 1 - i$ hosszú eltolás szabályos. Legyen $l = bestfact(i\ b)$. A $bestfact$ definíciója alapján, $x[i - l] \neq x[i]$. Mivel a *l* egy periódusa z-nek, a $x[i - l]$ és a $x[i]$ karakterek nem lehetnek egyszerr e a z előfordulásához igazítva az y tömbben. Ebből arra következtethetünk, hogy a $|z| = m - 1 - i$ eltolás szabályos.

Két példa lesz majd látható a Turbó-Szuffix-Keresés algoritmusára, az egyik a $bestfact$-ot, a másik a $goodsuff$-et használja a **??**.

2.3. A keresési fázis futási ideje

Lássuk be, hogy a Turbó-Szuffix-Keresés lineáis futási idejű a legrosszabb esetben.



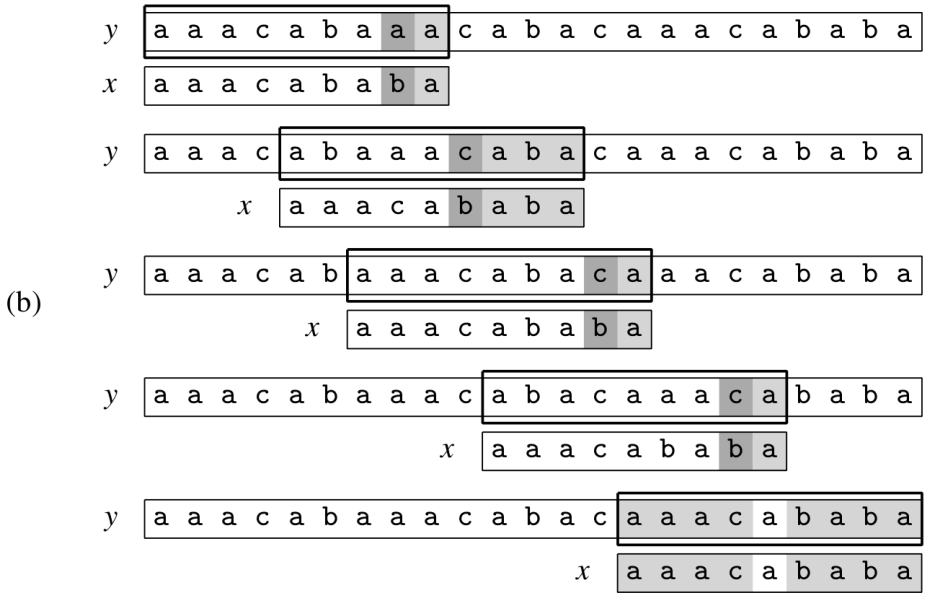
2.4. Tétel

Az m hosszú x karakterlánc összes előfordulásának keresése egy n hosszú y szövegben a Turbó-Szuffix-Keresés algoritmusával legrosszabb esetben $2n$ karakterösszehasonlítást jelent.

2.4.1. Bizonyítás

Felhasználva az előző bizonyítást, azt mondjuk, hogy a d hosszú eltolás a T-edik kísélet után rövid, ha $2d < |z| + 1$, és hosszú egyébként.

Különböztessünk meg három tipusú próbálkozást:



3. ábra. Péda a Turbó-Szuffix-Keresés két alkalmazására (a) $bestfact$-os használva, (b) $googsuff$-et használva a 18-as sorban az összehasonlításnál.

1. Olyan próbálkozás, amelyet egy ugrás követ,

2. Olyan próbálkozás, ami nem 1-es típusú és egy hosszú eltolás követi,

3. Olyan próbálkozás, ami nem 1-es típusú és egy rövid eltolás követi.

Az bizonyítás alapötlete az, hogy csökkentsük az összehasonlítások számát az eltolásokkal. Legyen a próbálkozások $cost(T)$ a következő:

$$f := \begin{cases} 1, & \text{ha a T próbálkozás 1-es típusú,} \\ 1 + |z|, & \text{ha a T próbálkozás 2-es vagy 3-as típusú.} \end{cases}$$

Abban az esetben ha 1-es típusú egyezésünk van, a $cost(T)$ függvény megadja, hogy mennyibe kerül a felesleges tesztelés. Az egyéb összehasonlítások költsége el lesz halasztva a következő összehasonlításra. Következtetésképpen az összes összehasonlítások száma a Turbó-Szuffix-Keresés során egyenlő az összehasonlítások értékeinek összegével. Tehát bebizonyítottuk, hogy $\sum_T cost(T) \leq 2 \sum_T d \leq 2n$.

Egy 1-es típusú T_0 próbálkozásra: $cost(T_0) = 1 < 2d_0$, ha $d_0 \geq 1$.

Egy 2-es típusú T_0 próbálkozásra: $cost(T_0) = |z_0| + 1 \leq 2d_0$.

Már csak a 3-as típusú próbálkozás maradt hátra. Legyen egy 3-as típusú T_0 próbálkozás a $_0$ pozíción az y szövegben. Ebben az esetben $d_0 < |z_0|$, és tudjuk, hogy $d_0 = bestfact(m - |z_0|,\ y[j_0 - |z_0|])$. Ez azt jelenti, hogy a következő T_1 próbálkozás során a j_1 és pozíción az y sztringben lehetséges egy turbó-eltolás

Nézzük a következő eseteket:

a. $|z_0| + d_0 \leq m$. Ebből következik a turbó-eltolás definíciója alapján, hogy $d_1 \geq |z_0| - |z_1|$. Tehát: $cost(T_0) = |z_0| + 1 \leq |z_1| + d_1 + 1 \leq d_0 + d_1$.

b. $|z_0| + d_0 > m$. Ebből következik a turbó-eltolás definíciója alapján, hogy $|z_1| + d_0 + d_1 \geq m$. Tehát: $cost(T) \leq m \leq 2d_0 - 1 + d_1$

Mindig feltételezhetjük azt, hogy a b. eset következik be a T_1-es próbálkozásra, hiszen egy hosszú határt kapunk értékül a $cost(T_0)$-ra.

Amikor a T_1 1-es típusú próbálkozásra tudjuk hogy $cost(T_1) = 1$ és $cost(T_0) + cost(T_1) \leq 2d_0 + d_1$. Ez pedig jobb a vártnál.

Amikor a T_1 2-es típusú próbálkozás vagy amikor $|z1| \leq d_1$ akkor $cost(T_0) + cost(T_1) \leq 2d_0 + d_1$.

Már csak az az eset maradt hátra, amikor T_1 3-as típusú próbálkozás és $|z_1| > d_1$. Ez azt jelenti, hogy a T_0 próbálkozás után $d_1 = bestfact(m - |z_1|,\ y[j_1 - |z1|])$. A T_1 próbálkozásnál használt argumentumot használjuk a következő T_2 próbálkozásnál is. Az a. eset csak a T_2 próbálkozásnál következhet be. Ez a következő eredményt adja $cost(T_1) \leq d_1 + d_2$. Végül $cost(T_0) + cost(T_1) \leq 2d_0 + 2d_1 + d_2$.

Az utolsó argumentum adja meg az indukciós bizonyítás lépését. Ha $T_0,\ T_1, \ldots,\ T_k$ mind 3-as típusú és $|z_j| > d_j$ minden $j = 0,\ 1, \ldots,\ k$, akkor

$$cost(T_0) + cost(T_1) + \cdots + cost(T_k) \leq 2d_0 + 2d_1 + \cdots + 2d_k + d_{k+1}.$$

Legyen $T_{k'}$ az első próbálkozás a T_0 próbálkozás után, tehát $|z_{k1}| \leq d_{k'}$. Ez a próbálkozás létezik, hiszen az ellentétes eset azt jelentené, hogy létezik egy végtelen ciklus az eseteknél az egyre rövidebb és rövidebb eltolások felé, ami lehetetlen. Tehát $cost(T_0) + cost(T_1) + \cdots + cost(T_{k'}) \leq 2d_0 + 2d_1 + \cdots + 2d_{k'}$ és $\sum_T cost(T) < 2\sum_T d_T \leq 2n$.

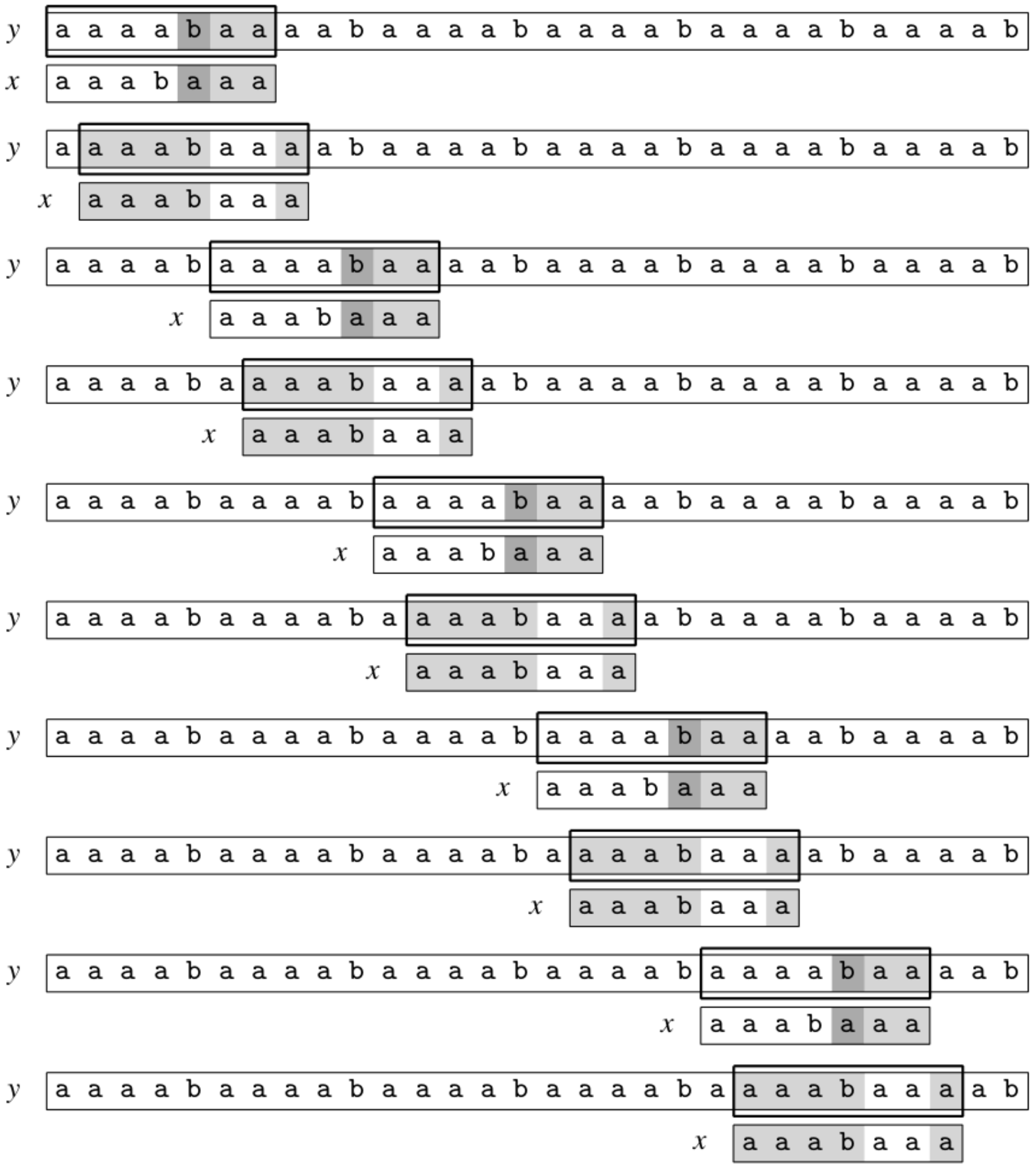
A határ a $2n$ összehasonlítás a tétel esetén majdnem optimális. Nézzük a következő példát. Legyen $x = a^k b a^k$ és $y = (a^{k+1})^l$, $(k \geq 1)$. Tudjuk hogy $m = 2k + 1$ és $n = l(k + 2)$. Az $a^{k+1}b$ ($k + 2$ hosszú) első és az utolsó előfordulásától eltekintve az y szövegben a Turbó-Szuffix-Keresés $2k + 2$ összehasonlítást hajt végre. Elsőre $k + 2$-t, utolsóra k-t. Tehát ezt kaptuk.

$$(l - 1)(2k + 2) = 2n(\tfrac{m+1}{m+3}) - m - 1$$

karakter összehasonlítás történik az algoritmus müködése közben. a 4-án látható egy példa $k = 3$ és $l = 6$ esetre.

2.5. Következmény

A Turbó-Szuffix-Keresés algoritmusa megtalálja az összes előfordulását egy adott karakterláncnak egy n hosszú szövegben **O**(n) idő alatt, konstans extra tárhely felhasználásával.



4. ábra. Példa a legrosszabb esetre a Turbó-Szuffix-Keresés használatával.

2.5.1. Bizonyítás

Egyértelműen következik az előző tételből.

3. Keresés több memória felhasználásával

Most egy még gyorsabb algoritmussal ismerkedhetunk meg. Ezt a gyorsulást úgy érhetjük el, hogy még több információt jegyzünk meg a korábbi próbálkozások során. Ehhez **O**(m) konstans méretű extra memóriára van szükség, azonban a futási idő $1.5n$-re csökken.

Ennek az algoritmusnak a neve Memória-Szuffix-Keresés. Az algoritmus megjegyzi a keresett karakterlánc összes szuffixének összes előfordulását a szövegben, és ezt az információt együtt használja a *szuffix tábláv*al, annak érdekében, hogy még inkább megnöveljük

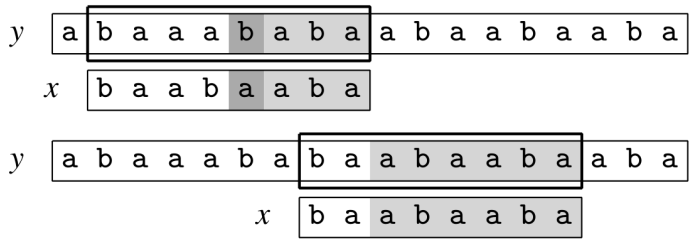
az eltolások hosszúságát. Ezek az eltolások a $bestfact$ segítségével vannak kiszámolva, de kiszámíthatóak a $goodsuffix$ felhasználásával is.

3.1. Keresési fázis

Itt kerül leírásra az algoritmus lényege. Minden próbálkozás után a j' pozíción az y szövegben, az x leghossabb szuffixének hossza felismert a j' pozícióban, $|j'|$ el van tárolva, $|z'|$ pedig a táblában van eltárolva ($S[j'] = |z'|$). A jelenlegi próbálkozás alatt a j pozíción az y szövegben, ha meg kell vizsgálnunk a $|j'|$, $j' < j$, ahol a $k = S[j']$, tudjuk hogy $y[j' - k + 1 \dots j'] \preceq_{szuff} x$. Legyen $i = m - 1 - j + j'$. Elég tudni az x leghosszabb szuffixének a hosszát, $s = szuff[i]$ az i pozíción végződve, hogy a legtöbb esetben következtetni tudjunk a próbálkozás eredményére.

3.2. Lemma

Amikor $s \leq k$ és $s = i + 1$, x egy előfordulása a jó j pozíción következik be az y szövegben. Tudjuk, hogy $S[j] = m$ és hogy a $per(x)$ hosszú eltolás szabályos.



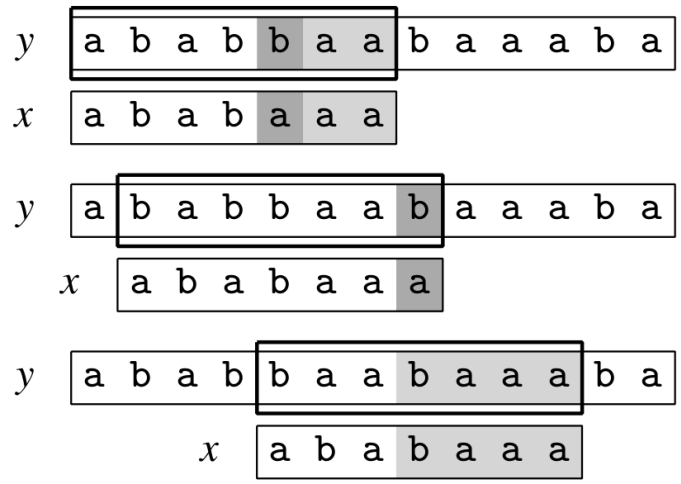
5. ábra. Egy próbálkozás alatt a 14-es pozíción észrevesszük, hogy a karakterenkénti összehasonlításkor az $abaaba$ suffix 6 hosszú a sztringben. Elérünk a 8-as pozícióig az y-ban, ahol tudjuk, hogy a leghosszabb szuffix az x-en ezen a pozíción végződik és 3 hosszú. Tudjuk, hogy az x leghosszabb szuffixe az 1-es pozíción ér véget. Az első lemma alapján tudjuk, hogy ez az x egy előfprdulását jelenti a 14-es pozíción.

3.3. Lemma

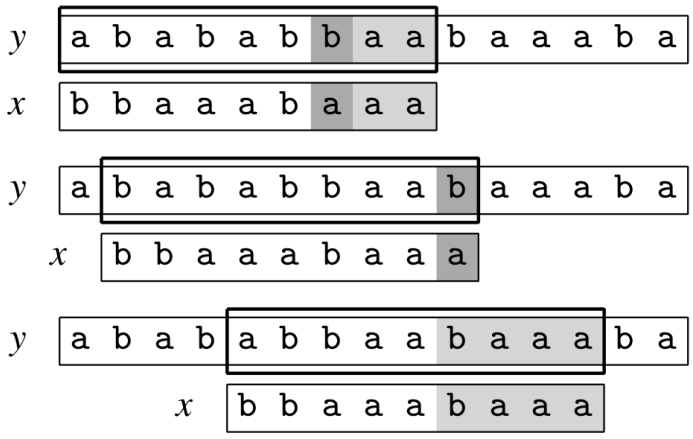
Amikor $s \leq i$ és $s < k$ akkor tudjuk hogy $S[j] = m-1-i+s$ és legyen $j' = j-m+1+i$, ekkor a $bestfact(i - s,\ y[j' - s])$ hosszú eltolás szabályos.

3.4. Lemma

Amikor $k < s$, akkor tudjuk, hogy $S[i] = m - 1 - i + k$, és legyen $j = j - m + 1 + 1$, ekkor tudjuk hogy a $bestfact(i - k,\ y[j' - k])$.



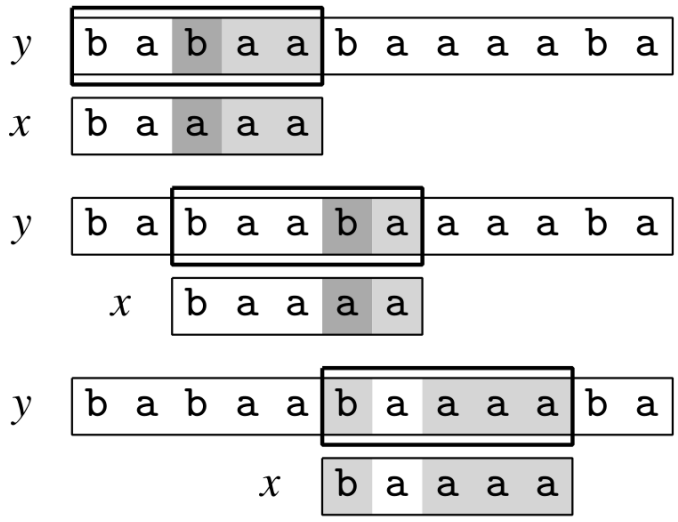
6. ábra. Amig a 10-es pozición próbálkozunk, észrevehetjük, hogy a $baaa$ a szöveg 4 hosszú szuffixe. Ekkor elmehetünk az y 6-os pozíciójára, ahol tudjuk, hogy az x lehosszabb szuffixe itt végződik és 2 hosszú. Tehát az $y[4 \dots 6]$ összehasonlítása felesleges, mert ott nem lesz egyezés.



7. ábra. Amíg a 12-es pozición próbálkozunk, észrevehetjük, hogy a szring 4 hosszú szuffixe a 8-as pozíción van, ahol tudjuk, hogy a leghosszabb szuffix ami ezen a helyen végződik az 2 hosszú. Tehát egyetlen összhasonlítás nélkül is tudjuk, hogy $x[2] = a$ és $y[6] = b$.

3.5. Lemma

Amikor $k = s$, akkor tudjuk, hogy $x[i - s + 1 \dots m - 1] = y[j' - s + 1 \dots j]$, és $S[j] = m - 1 - i + s + |lcszuff(x[0 \dots i - s],\ y[j - m + 1 \dots j' - s])|$ ami azt jelenti, hogy $j' = j - m + 1 + i$.



8. ábra. Amíg a 9-es pozíción próblkozunk, észrevehetjük, hogy a sztring 3 hosszú szuffixe a 6-os pozícióra került, ahol tudjuk, hogy a leghosszabb szuffix a sztringben ezen a pozíción végződik és 1 hosszú. Emellett tudjuk, hogy a leghosszabb szuffix a sztringben ami az 1-es pozíción végződik 1 hosszú. Tehát átugorhatjuk a $y[6]$-ot, mert összehasonlítás nélkül is tudjuk, hogy $x[0] = y[5]$.

3.6. Az algoritmus

Most pedig következzen maga az algoritmus 3.6. Jelen esetben a $bestfact$-ot és a szuffix táblát használtuk. Az x szuffixeinek előfordulásainak memorizálásához a szövegben a szuffix táblában kerül tárolásra. Ennek a táblának az értékeire kezdetben mindenhol 0 prioritást állítunk be.

3.7. Tétel

A Memória-Szuffix-Keresés megtalálja egy x karakterlánc összes előfordulását egy y szövegben.

3.7.1. Bizonyítás

Egyértelműen következik az első és az utolsó lemmából.

3.8. A keresési fázis komlexitása

Eddig folyamatosan azt viszgáltuk, hogy a mekkora a tárigénye a Memória-Szuffix-Keresésnek futás közben.

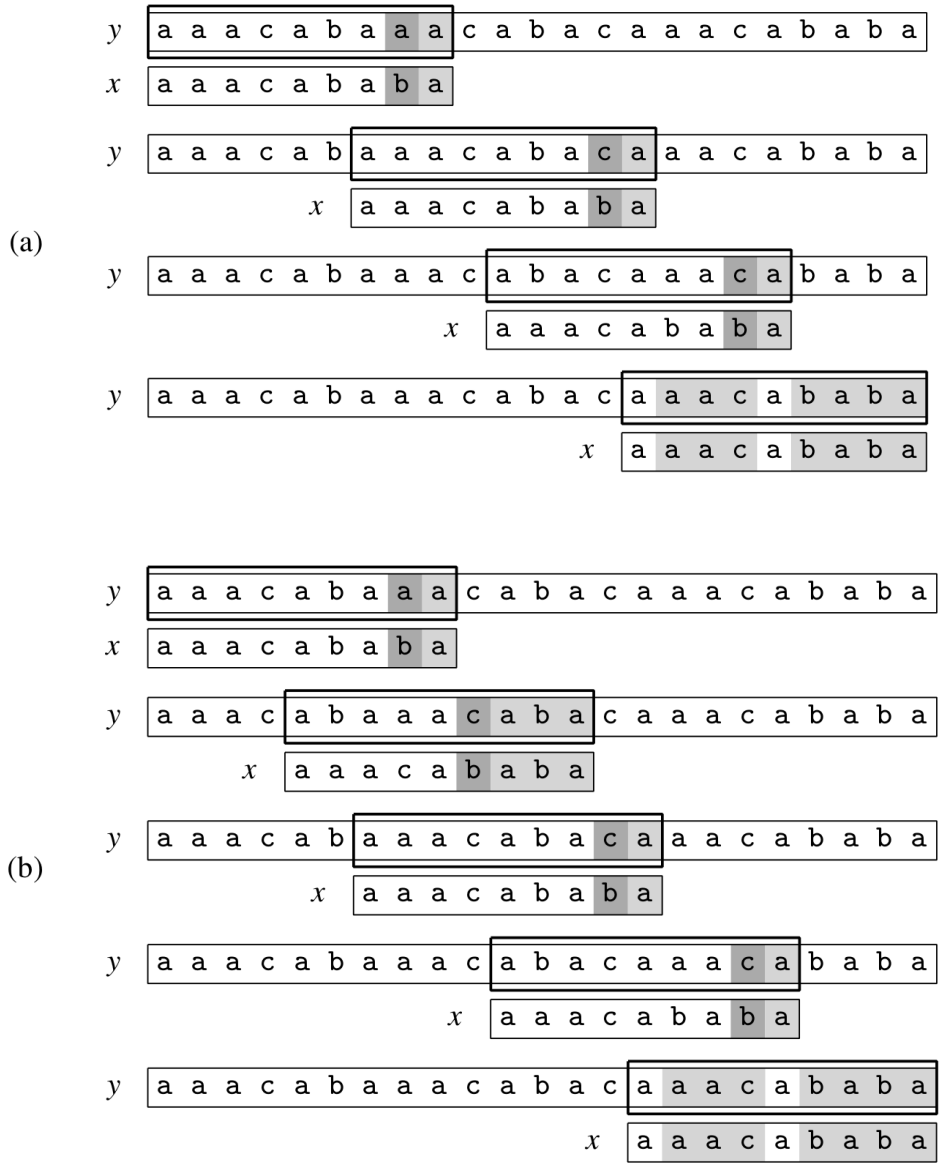
```
MEMORY-SUFFIX-SEARCH(x, m, y, n)
 1  for j ← 0 to n − 1 do
 2        S[j] ← 0
 3  j ← m − 1
 4  while j < n do
 5        i ← m − 1
 6        while i ≥ 0 do
 7              if S[j − m + 1 + i] > 0 then
 8                    k ← S[j − m + 1 + i]
 9                    s ← suff[i]
10                    if s ≠ k then
11                          i ← i − min{s, k}
12                          break
13                    else  i ← i − k ▷ Jump
14              elseif x[i] = y[j − m + 1 + i] then
15                    i ← i − 1
16              else  break
17        OUTPUT-IF(i < 0)
18        if i < 0 then
19              S[j] ← m
20              j ← j + per(x)
21        else  S[j] ← m − 1 − i
22              j ← j + best-fact(i, y[j − m + 1 + i])
```

3.9. Következmény

Egy m hosszú x karakteránc keresése egy szövegben megoldható $\mathbf{O}(m)$ extra memória felhasználásával.

3.9.1. Bizonyítás

A tárhelyet a szuffix tábla memorizálásához használjuk, hogy implementáljuk a $bestfact$ funkciót vagy a $goodsuff$ funkciót az S szuffix tábla létrehozásához. Az első 3 elem elhasznál **O**(m) memóriát. Az S táblában nem jegyezzük meg csak a $S[j-m+1 \ldots j]$, mert csak ez a része a hasznos amikor a kereső ablak a jobboldali j pozíción van. Az S



9. ábra. Két példa a Memória-Szuffix-Keresés algoritmusára. (a) $bestfact$ használatával. Ebben az esetben a 13 az összehasonlítások száma a keresett karakterlánc karakterei és a szöveg karakterei köszött. (b) a $goodsuff$ használatával. Ebben az esetben ugyanez a szám 17.

tábla menedzseléséhez úgy tekintünk rá mint egy ciklikus listára, amiben csak a hasznos elemeket tároljuk, azaz $S[l]$ elemet, tehát ez így még mindig **O**(m) tárigényű.

Most pedig lássuk be, hogy a futási ideje **O**(n). Ez egész pontosan $1.5n$ karakter-összehasonltást jelent ahhoz, hogy megtaláljuk egy x karaktelánc összes előfordulását egy y szövegben.

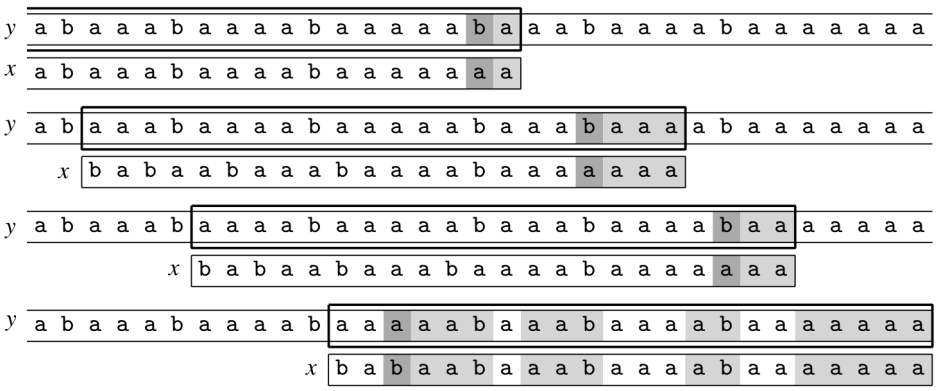
Jegyezzük meg először is, hogy a keresési fázis alatt egy karakter előfordulása az y szövegben csak egyszer lesz összehasonlítva bármivel, egy karaktert a legtöbb esetben nem hasonlítunk kétszer. Ez tehát legrosszabb esetben n darab összehasonlítást jelent.

Csak azokat a karaktereket hasonlíthajuk össze kétszer, amelyek egy előzőleg egy nem egyezést mutattak.

Nézzük a következő lemmát.

3.10. Lemma

Egy próbálkozás során a Memória-Szuffix-Keresés algoritmusában ha k pozitív osszehasonlítást már elvégeztünk, akkor a következő eltolás legaább k hosszú lesz.



10. ábra. Egy próbálkozás alatt a j pozíción a szövegben észrevehetjük, az 1 hosszú szuffixet a sztringben, ekkor pedig eltolhatjuk a 6-os pozícióba. Vegyük észre, hogy ekkor a szuffix 3 hosszú, tehát eltolhatjuk 4 pozícióval a keresőablakot. Ekkor láthatjuk, hogy itt egy 2 hosszú szuffix van, tehát eltolhatjuk az ablakot 5 pozícióval. A $j + 15$ próbálkozás során észrevehetjük, hogy a 19 hosszú szuffix egyezik, mindez annak árán, hogy a $y[j + 8]$, $y[j + 3]$, $y[j - 1]$ pozíción lévő elemeket kétszer is összehasonlítottuk. Az eltolás ami ezt a próbálkozást követi nem lehet rövidebb vagy egyenlő mint 3 a negyedik lemma alapján. Valóban a szuffix $aabaaabaaaabaaaaaaa$ a szövegben nem lehet kisebb vagy egyenlő pediódusú mint 3.

3.10.1. Bizonyítás

Legyen T egy kísérlet a Memória-Szuffix-Keresés algoritmus során. Legyen k az a arakter, amelyet más összehasonlítottunk egyszer és újrahasonltjuk. Legyen a szöveg:

$b_0v_0b_1v_1 \dots b_ku_kv_k$

a megmaradt tényező a T kísérlet során.

$\rightarrow b_0$ az a karakter amelyik nem egyezést okoz T során, $\rightarrow b_0v_0b_1v_1 \dots b_ku_kv_k$ egy szuffixe x-nek, $\rightarrow$a b_l, $(1 \leq l \leq k)$ karakterek azok a karakterek amelyek újra lettek próbálva T során és egyezést mutattak, $\rightarrow$a u_l, $(1 \leq l \leq k)$ azok a szuffixek (valószínüleg üresek), amelyek azok a karaktereket tartalmazzák amelyek össze lettek hasonlítva egyezően, és b_l az amelik nem egyezett, ezek a tényezők át lesznek ugorva a T kísérlet

során, →a v_l, $(1 \leq l \leq k)$ tényezők azok amelyek egyezően lettek összehasonlítva elsőre T során.

Definíció alapján $b_l u_l$, $(1 \leq l \leq k)$ nem szuffixei x-nek.

A bízonyítás az összegésen alapszik. Feltételezzük, hogy d eltolás alkalmazva lett csak a T kísérlet után, és ez rövidebb mint k. Ekkor legyen w az x-nek az a szuffixe amelyik d hosszú. Ekkor a definícióból következik, hogy

$v_0 b_1 u_1 v_1 b_2 u_2 v_2 \ldots b_k u_k v_k w$

egy szuffixe x-nek úgy hogy $d = |w|$.

Ekkor két különböző indexre $l' \neq l$", $u_{l'}$ és $u_{l''}$ ugyanazzal a w tényezővel vannak határolva, és ez maxium $k-1$ pozíción lehetséges. Ez azt jelenti, hogy $b_{l'} u_{l'} = b_{l''} u_{l''}$. Az eltolás, amely alkalmazva lett a két kísérletre $b_{l'}$ és $b_{l''}$. Ekkor létezik egy olyan l index, $l < k$ és $b_l u_l = b_k u_k$, ami ellentmond annak a ténynek, hogy az x karakterlánc már határolva volt a T kísérlet előtt.

Ez azt jelenti, hogy a T kísérlet során alkalmazott eltolás legalább k hosszú.

3.11. Lemma

A Memória-Szuffix-Keresés algoritmusa legfeljebb $n/2$ olyan karaktert hasonlít újra, amelyet már egyszer összehasonított.

3.11.1. Bizonyítás

Csoportosítsuk a próbálkozásokat. Két próbálkozás azonos csoportba kerül, ha van olyan karakter amelyet mindketten összehasonlítanak a szövegben. Egy p csomag amelyben k darab egyező újraösszehasonlítás volt és ebben a csomagban legyen legalább $k+1$ próbálkozás. Ezek alapján van k darab próbálkozásunk amelyek egy legalabb 1 hosszú eltolással járnak, illetva van egy eltolásunk, ahol az eltolás legalább k az 5-ös lemma alapján. Tehát az együttes eltolás összesen legalabb $2k$.

Ebből következik hogy az összes csomag együttes eltolása a Memória-Szuffix-Keresés közben nem több mint n. Tehát az újraösszehasonlítások száma nem több mint $n/2$.

3.12. Tétel

Egy n hosszó y szövegben való keresés közben a Memória-Szuffix-Keresés algoritmusa legrosszabb esetben $1.5n$ összehasonlítást tartalmaz a szöveg és a keresett karakterlánc betűi között.

3.12.1. Bizonyítás

Egyértelműen következik az előző lemmából és abból a tényből, hogy legfeljebb n egyező összehasonlítás lehet.

3.13. Következmény

A Memória-Szuffix-Keresés algoritmusa egy m hosszú karakterlánc összes előfordulását megtalálja egy m hosszú szövegben **O**(n) idő alatt és **O**(m) extra memória felhasználásával.

3.13.1. Bizonyítás

Egyértelműen következik az ezt meglőző és az azelőtti tételből.

4. Keresés szótárban

A csúszó ablak technikával megoldható, hogy keressük egy karakterlánc összes előfordulását egy k karakterláncból álló $X = x_0, x_1, \ldots, x_{k-1}$ y szövegben. Ez most legyen m' és m" rendre a leghosszabb és a legrövidebb karakter hossza X-ben.

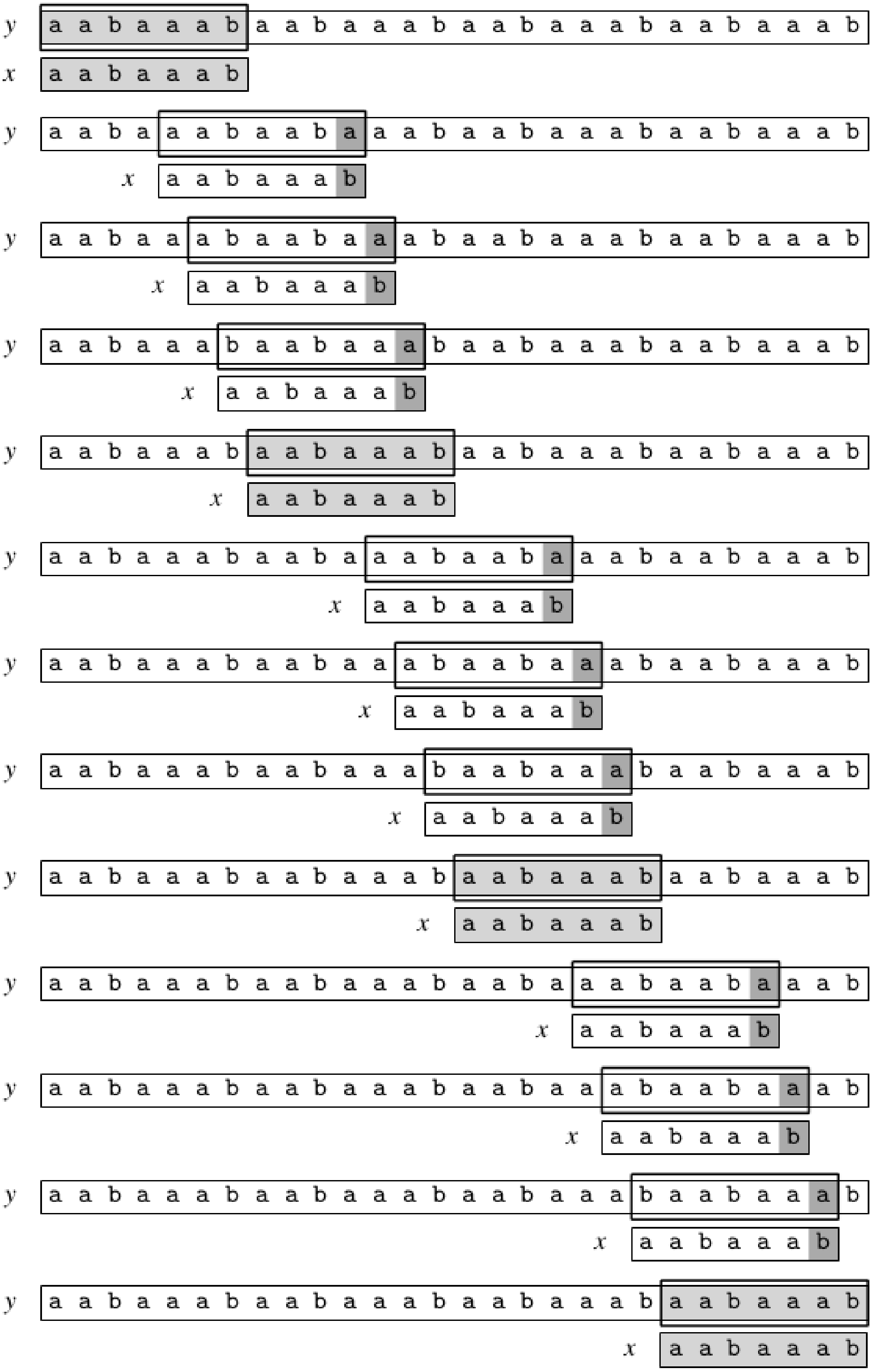
A szöveg beolvasásának hosszát az határozza meg, hogy mekkora a lehosszabb tényezője X-nek, ami egy szuffixe lesz a csúszó ablakunknak. Ilyen módon hosszabbítsa meg a szuffixet a beolvasott ablakhoz képest az előző módszer alapján. Ez lehetőséget ad arra, hogy több információt gyűjtsön a szövegről, és gyakran hosszabb eltolódáshoz vezet. Így az y szöveg beolvasáse során az algoritmus elegendő információt fog tartalmazni az keresett karakterláncok előfordulási helyeinek észleléséhez.

Az algoritmus elsődleges célja az X karakterláncainak felderítése az m" hosszúságú csúszóablakban. Az alapelv az, hogy minden próbálkozás során meghatározzuk az x karakterláncainak prefixeit, amit a csúszóablak szuffixei lesznek. Ezzel egyidejűleg figyeljük, hogy az X elemeinek előfordulásai az ablakban, és megjegyzettük azt az eltolódást aminek nem tolhatunk el hosszabban, ez legyen m'.

Most ismertessük azt a technikát amelyet ennek megvalósítására használunk. Legyen $X^{\sim}$ az a karakterláncokból álló szöveg, amelyik az X egyes karakterláncainak megfordítását tartalmazza. Legyen N egy olyan determinisztikus automata, ami felismeri az $X^{\sim}$ sztringek utótagjait. A hozzá rendelt átmeneti függvény legyen δ . Az automata a következő nyelvet ismeri fel

$$Suff(X^{\sim}) = v \in A^* : uv = x^{\sim}, u \in A^*, x \in X.$$

Vagyis N determinisztikusan felismeri a prefixeit X elemeinek. Minden terminális állapotára az automatának legyen:



11. ábra. Példa a tételre, $x = aabaaab$ és $y = (aabaaab)^4$. A keresett karakterlánk 7 hosszú, egy 28 hosszú szövegben. Az összehasonlítások száma 37. Az x utolsó 3 előfordulása esetén a Memória-Szuffix-Keresés algoritmusa 10 összehasonlítást végez.

$output[q] = i : 0 \leq i \leq k-1 \ \wedge \ \overline{\delta}(q_0, x_i^{\sim})$.

ahol δ az automata δ átmeneti függvényeinek kiterjesztése, a q_0 pedig a kezdőállapot.

Egy kísérlet a j pozíción az y szövegben az y karaktereinek elemézéséből áll a bal oldali $y[j]$-től az N automata által. Minden alkalommal, amikor a q állapot elér egy $y[j']$ karakterig, ellenőrizzük hogy az $output[q]$ üres e. Ha ez a helyzet, akkor az x_i sztring megjelenik az y-ban a j' pozíción, amikor

$i \, inoutput[q] \ \wedge \ j - j' + 1 = |x_i|$.

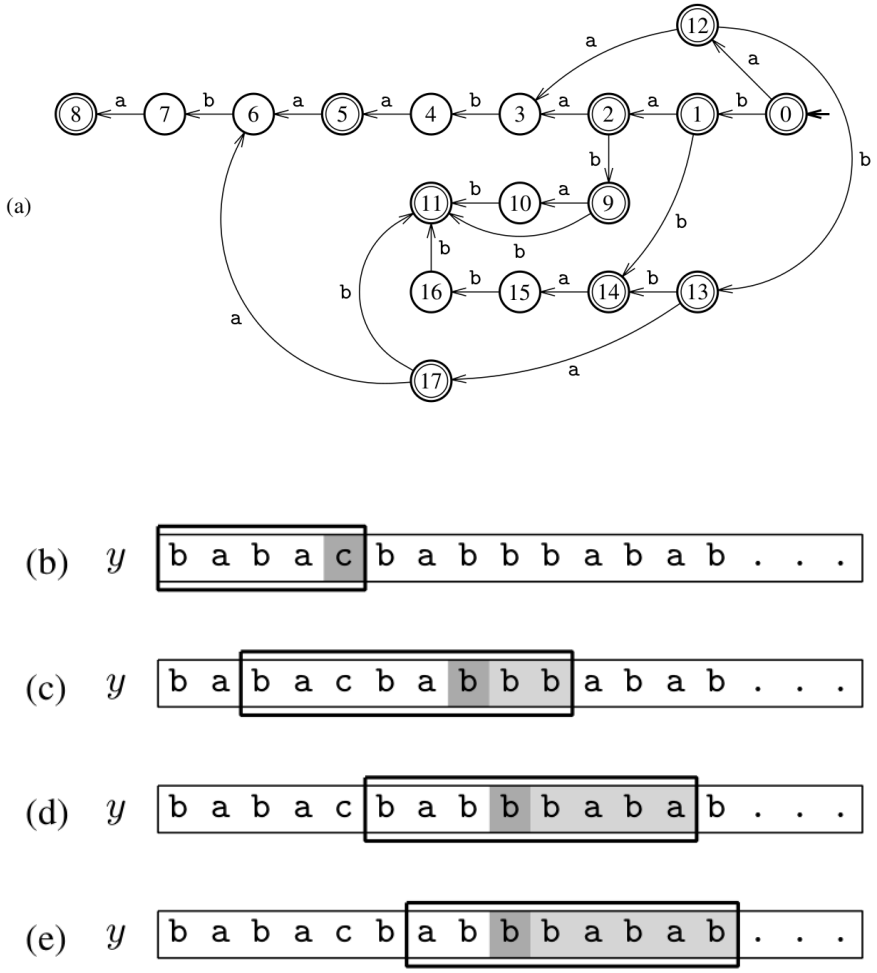
Emellett ha a q állapot terminálóállapot, akkor az eltolás hossza nem lehet nagyobb, mint $m' - (j - j' + 1)$, feltéve, hogy ez a mennyiség pozitív. Így számíthatunk az érvényes eltolások minimális d hosszúságára. Egy kísérlet akkor ér véget, amikot az aktuállis állapotból a következő karakteren keresztül nincs több átmenet.

4.1. Az algoritmus

```
MULTIPLE-SUFFIX-SEARCH(X, m', y, n)
 1  let N be a (deterministic) automaton
    accepting the suffixes of the reverse strings of X
 2  j ← m' − 1
 3  while j ≤ n do
 4        q ← initial[N]
 5        j' ← j
 6        d ← m'
 7        while j' ≥ 0 and TARGET(q, y[j']) ≠ NIL do
 8              q ← TARGET(q, y[j'])
 9              if terminal[q] then
10                    for each i ∈ output[q] do
11                          OUTPUT-IF(|x_i| = j − j' + 1)
12                    if m' − j + j' − 1 > 0 then
13                          d ← min{d, m' − j + j' − 1}
14                    else d ← 1
15              j' ← j' − 1
16        j ← j + d
```

4.2. Tétel

A Többszörös-Szuffix-Keresés algoritmus felismeri az X karakterlánchalmaz összes elemének összes előfordulását egy y szövegben.



12. ábra. A Többszörös-Szuffix-Keresés algoritmusa egy példán keresztül bemutatva.

5. Összefoglalás

Szó volt arról, hogy hogyan kell karakterláncokat keresnünk egy hosszabb szövegben. Erre több megoldást is láthattunk, melyek egyre több memóriafelhasználással ugyan, de egyre gyorsabban müködtek.

Először csak egy segédmemóriát használtunk, ennek az algoritmusnak a neve Turbó-Szuffix-Keresés. itt használtuk fel a legkevesebb extra tárhelyet, azonban ez az elgoritmus a legrosszabb esetben $2n$ karakterösszehasonlítást végez, ahol n annak a szövegnek a hossza, amelyben keresünk.

Ezt követően már több extra tárhelyet is igénybe vettünk, azonban így el tudtuk érni, hogy a legrosszabb esetben is csak $1.5n$ összehasonlítást végezzunk. Ennek az algoritmusnak a neve Memória-Szuffix-Keresés volt.

Ezt követően megnéztük, hogy hogyan kell egy több szót tartalmazó halmazban keresni egy adott karakterlánc előfordulását.

Mindegyik algoritmusra néztünk példát, elemeztük a működését.

6. Források

→ Maxime Crochemore, Christophe Hancart, Thierry Lecroq: Algorithms on strings, Cambridge University Press, Paris, 2001