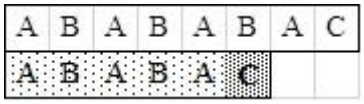
32. A Knuth-Morris-Pratt algoritmus

A „*nyers erőt*” használó *egyszerű mintaillesztés* műveletigénye legrosszabb esetben *m***n*-es volt. A *Knuth-Morris-Pratt algoritmus* (*KMP*-vel rövidítjük) egyike azon mintaillesztő eljárásoknak, amelyek ügyes észrevételek és mélyebb megfontolások alapján *hatékonyabb* módon oldják meg az stringkeresés feladatát.

32.1. Az algoritmus elve

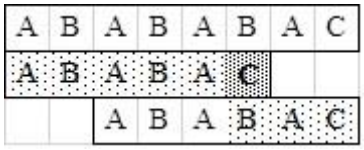
Amikor az egyszerű mintaillesztés során az illeszkedés elromlott, a mintát *egy pozícióval* eltoltuk és az elejétől újra kezdtük a minta és a lefedett szövegrész összehasonlítását. Nem biztos azonban, hogy a már megvizsgált szövegrész minden karakterén újra át kell haladni. Amennyiben az illeszkedés elromlik, akkor egy „hibás kezdetünk” van, de ez a kezdet ismert, mivel az elromlás előtti karakterig *egyezett* a mintával. Ezt az információt használjuk fel arra, hogy *elkerüljük* az állandó *visszalépést* a szövegben a minta kezdetére. Tekintsük a 32.1. ábrán látható illeszkedési feladatot.



32.1. ábra. Példa illeszkedésvizsgálatra (*KMP*)

A példában a minta 6. pozíciójánál romlik el az illeszkedés, hiszen a minta első 5 pozíciója illeszkedett. Kérdés, hogy *hová pozícionálhatjuk* a mintát a szövegben, és honnan vizsgáljuk tovább az illeszkedést, hogy a minta előfordulását megtaláljuk (ha létezik, át ne "ugorjuk") és az eddig megszerzett 5 illeszkedő karakternyi információt felhasználjuk.

Látható, hogy a minta illeszkedő részének ($M[1..5]$) van olyan *valódi kezdőszelete* (valódi prefixe), amely *egyezik* ezen illeszkedő rész egy *valódi végszeletével* (valódi szuffixével), hiszen $M[1..3] = M[3..5]$ ('ABA' = 'ABA'). A vizsgálatot ezért úgy is folytathatjuk, hogy a kezdőszelete „rátoljuk” a vele megegyező végszeletre, ahogyan a 32.2. ábrán látható.

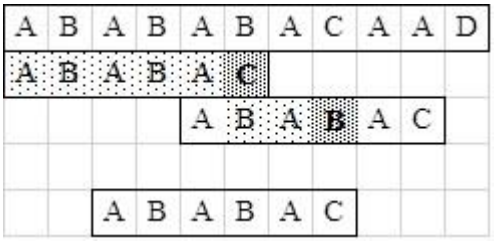


32.2. ábra. A minta megfelelő eltolása (*KMP*)

(A továbbiakban inkább a karakterisztikusabb *pefix* és *szuffix* kifejezéseket használjuk a leírásban.) Egy *prefix* vagy *szuffix valódi*, ha hossza legalább 1, és kisebb, mint annak a sorozatnak a hossza, amelynek a prefixe vagy szuffixe.

Amennyiben a mintával akkorát ugrunk, hogy a minta eleje az említett szuffixnél kezdődjön, azaz az prefix a vele egyező szuffixszel kerüljön fedésbe, a *prefixet* már *nem kell újra vizsgálni*, mivel az azonos a szuffixel, ami megegyezik a szöveg lefedett részével, mivel az részsorozata az eredetileg illeszkedő $M[1..5] = S[k+1..k+5]$ szövegrésznek. Ezek után az illeszkedés vizsgálatot a szöveg "elromlott" $S[k+6]$ karakterével, és az említett *prefix utáni első karakterrel* lehet tovább folytatni.

Mi a teendő *több* ilyen egyező prefix és szuffix pár esetén? A példában is találhatunk egy másik párost, az $M[1..1] = M[5..5]$ ('A'='A'). Ha annak megfelelően pozícionáljuk a mintát, ahogyan a 32.3. ábra is mutatja, majd a következő karaktertől kezdünk összehasonlítani, azt tapasztaljuk, hogy nem illeszkedik a minta a szövegre, mert "átugrottunk" egy illeszkedést.



32.3. ábra. Nem megfelelő eltolás több egyező prefix és suffix pár esetén

Tehát a *legkisebb* olyan ugrást kell választanunk, ahol a minta $M[1..5]$ részsorozatának egy prefixe illeszkedik a részsorozat egy szufixére. Akkor "ugrunk" a legkisebbet, ha a *legnagyobb* ilyen prefixet választjuk.

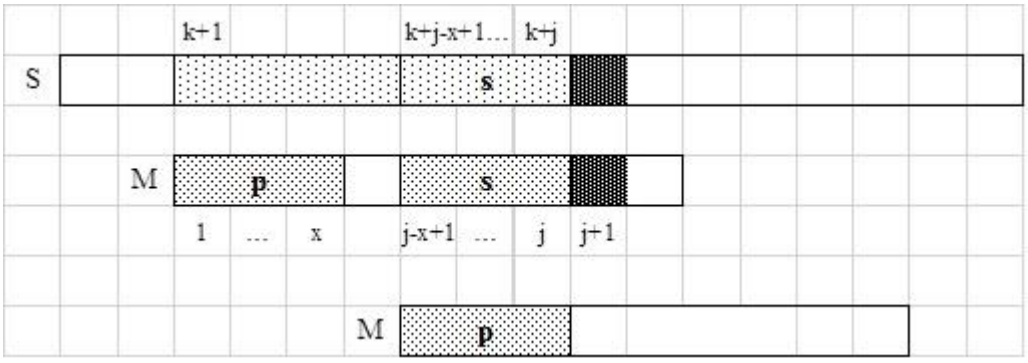
32.2. Az algoritmus helyessége

Ahhoz, hogy az algoritmus helyességét belássuk, a következő kérdéseket kell tisztáznunk. Tegyük fel, hogy a minta $M[1..j]$ részsorozata illeszkedett a szöveg $S[k+1..k+j]$ részsorozatára és az illeszkedés a következő pozíción romlott el, azaz

$$M[1..j] = S[k+1..k+j] \text{ és } M[j+1] \neq S[k+j+1]$$

1. Ha létezik $M[1..j]$ részsorozatnak olyan valódi prefixe ($p = M[1..x]$) és szuffixe ($s = M[j-x+1..j]$), hogy $p = s$, akkor valóban állítható-e, hogy az ugrás után biztosan *nem kell újra vizsgálni* az $M[1..x]$ és az általa lefedett $S[k+j-x+1..k+j]$ szövegrészt?

   Biztosan nem kell, mivel $p = s$, azaz $M[1..x] = M[j-x+1..j]$, továbbá az illeszkedés az $M[j+1]$ pozíción romlott el, tehát $M[1..j] = S[k+1..k+j]$, és ezek tetszőleges, jelenleg fedésben lévő részsorozatai is azonosak, azaz $S[k+j-x+1..k+j] = M[j-x+1..j] \Rightarrow M[1..x] = S[k+j-x+1..k+j]$. Érvelésünket alátámasztja a 32.4. ábra.



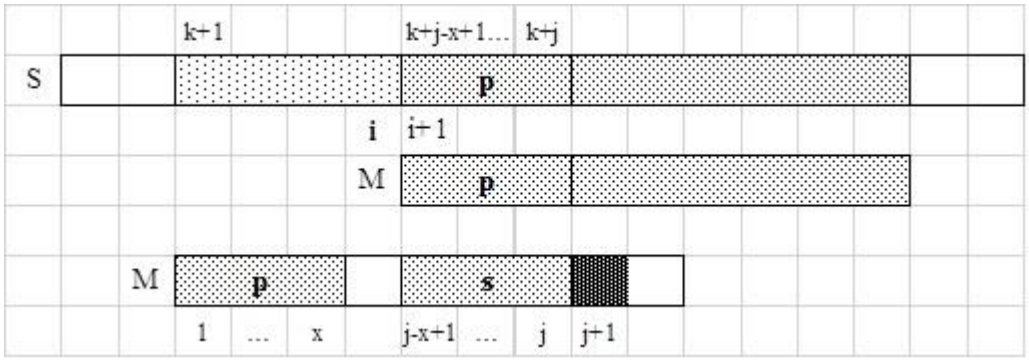
32.4. ábra. Példa egyező valódi prefix-szuffix párosra

2. Mit tegyünk, ha nincs ilyen egymással megegyező valódi prefix-szuffix páros?

   Mivel $M[1..j]$ illeszkedett és $M[j+1] \neq S[k+j+1]$, $\forall i\,(k < i < k+j)$ eltolásra a minta biztosan nem fog illeszkedni. Ahogyan a 32.5. ábrán is látszik, ahhoz hogy ilyen i eltolással illeszkedjen, az kellene, hogy legyen legalább 1 hosszú valódi, egymással azonos prefix-szuffix páros ($p = s$), mert az $M[1..x] = S[k+j-x+1..k+j]$ részsorozatoknak illeszkedniük kell (ekkor $i = k+j-x$) ahhoz, hogy teljes illeszkedés lehessen. Ebből pedig következik, hogy $M[1..x] = M[j-x+1..j]$, mivel $M[1..j] = S[k+1..k+j]$.

(Beláttuk tehát, hogy az $M[1..j] = S[k+1..k+j]$ feltétel esetén, az i $(k < i < k+j)$ érvényes eltolás szükséges feltételét is.)

Tehát a mintával "átugorhatjuk" a már vizsgált $S[k+1..k+j]$ részt, és az illesztést a minta elejétől és a szöveg $S[k+j+1]$ pozíciójától újra kezdhetjük. Ezt a konklúziót a 32.5. ábra is alátámasztja.



32.5. ábra. Példa nem egyező valódi prefix-szuffix párra

3. Mit tegyünk, ha több ilyen egymással megegyező, valódi prefix-szuffix páros is van?

   Ha több ilyen prefix-szuffix páros is van, akkor a leghosszabbat kell venni, mert ekkor "ugrunk" a legkisebbet. Ilyenkor nem fordulhat elő, hogy átugrunk egy előfordulást.

Definiáljuk a *next* függvényt, amely megadja a minta egyes kezdőrészleteire a *leghosszabb* egymással egyező prefix-szuffix párok hosszát. Ezt felhasználva meg tudjuk adni a mintával való "ugrás" mértékét.

$$\forall j \in [1..m-1]: next(j) := \max\{h \in [0..j-1]\}, \text{ ahol } M[1..h] = M[j-h+1..j]$$

A *next* függvénnyel kapcsolatban a következő megjegyzéseket tesszük.

- A *next* értelmezési tartományát elegendő $(m-1)$-ig definiálni, mert ha $(j=m)$-ig illeszkedik a minta, akkor találtunk egy érvényes eltolást, tehát készen vagyunk, és nem kell a mintával tovább lépkednünk.
- A $h = 0$ legkisebb értékét, akkor veszi fel a függvény, ha *nincs* a minta $M[1..j]$ kezdőszeletében egymással megegyező, valódi prefix-szuffix páros. Továbbá, ha létezik ilyen prefix-szuffix páros, az attól lesz valódi, hogy a hosszát $(j-1)$-gyel felülről korlátozzuk.
- A *next* függvény *csak a mintától* függ, így értékeit a minta ismeretében a keresés előtt kiszámíthatjuk, és eltárolhatjuk egy $next[1..m-1]$ vektorban.

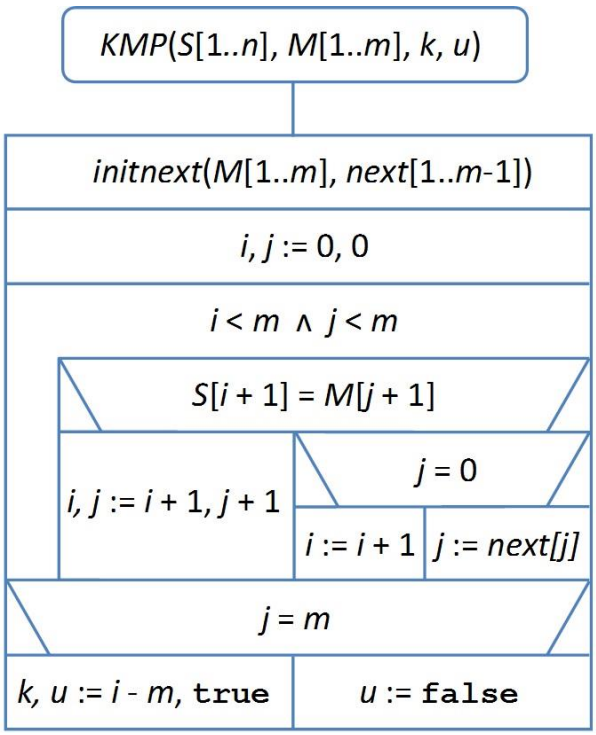
32.3. A *KMP* algoritmus

A minta elejétől kezdjük összehasonlítani a szöveg és a minta egymással fedésben lévő karaktereit. Amennyiben a szöveg és minta karakterei azonosak, akkor a szövegben és a mintában egyaránt eggyel továbblépünk. Azonban, ha a karakterek különböznek, a következőket tesszük.

- Ha a minta elején állunk ($j = 0$ esetén): a szöveg következő pozíciójától ($S[k+j+2]$) és a minta elejétől kezdve újra kezdjük az illeszkedés vizsgálatot, mivel a *next* függvény a valódi prefix-szuffix hosszát adja meg, de az 1 hosszú sorozatnak nincs valódi prefixe vagy szuffixe ($next[1] = 0$).

- Ha nem a minta elején állunk ($j > 0$ esetén), akkor a *next* függvényben rögzített eltolást hajtjuk végre: azt mondjuk, hogy eddig j hosszon illeszkedett a minta, továbbiakban $next[j]$ hosszon illeszkedik. Az összehasonlítást a minta $M[next[j]+1]$ karakterétől és a szöveg $S[k+j+1]$ karakterétől folytatjuk, azaz a szövegben onnan, ahol az illeszkedés elromlott.

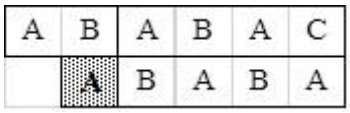
Mivel a szövegben legfeljebb 1 hosszú lépésekkel haladunk végig, az egyszerűség kedvéért a k eltolásnak megfelelő változó helyett használjunk egy i változót, amellyel a szövegben szekvenciálisan haladunk ($i = k + j$), majd az algoritmus végén beállítjuk a k változó értékét. A *KMP* algoritmus a 32.6. ábrán látható.



32.6. ábra. A *KMP* algoritmus

Az *initnext* eljárás során töltjük fel a *next* vektort. A feltöltés ötlete: a *minta elcsúsztatott keresése önmagán* (*KMP* algoritmussal), miközben feljegyezzük a legnagyobb illeszkedő részek hosszát.

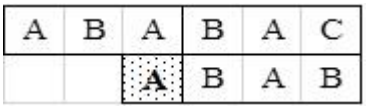
Nézzük meg egy példán a feltöltés menetét. Legyen a minta *M* = 'ABABAC'. Már korábban láttuk, hogy $next[1] = 0$. Ezután a $next[2]$ értékét szeretnénk meghatározni. Ekkor az $M[1..2]$ kezdőrészletnek keressük a legnagyobb egymással megegyező, valódi prefix-szuffix párját. A legnagyobb ilyen valódi prefix-szuffix 1 hosszúságú lehet. Tehát az a kérdés, hogy az $M[1] = M[2]$ egyenlőség teljesül-e? Ehhez a mintát csúsztassuk el eggyel, és a fedésben lévő karaktereket vizsgáljuk (lásd: 32.7. ábra):



32.7. ábra. A *next* vektor kiszámítása (1)

Látható, hogy a két karakter nem azonos, így $next[2] = 0$.

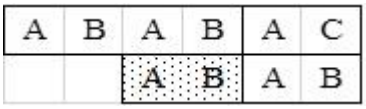
Most a *next*[3] meghatározása következik. Ekkor az $M[1..3]$ kezdőrészletnek keressük a legnagyobb egymással megegyező, valódi prefix-szuffix párját. A legnagyobb ilyen valódi prefix-szuffix 2 hosszú lehet. Azaz $M[1..2] = M[2..3]$ egyenlőség teljesül-e? Azonban ez nem teljesülhet, mivel már $M[1] = M[2]$ sem teljesült. Ezt nem is vizsgáljuk, mivel már az előző menetben sem volt egyezés. Helyette a mintát eggyel jobbra csúsztatjuk, és az $M[1] = M[3]$ egyenlőséget vizsgáljuk (lásd: 32.8. ábra):



32.8. ábra. A *next* vektor kiszámítása (2)

A vizsgált egyenlőség fennáll, ezért feljegyezzük, hogy $next[3] = 1$.

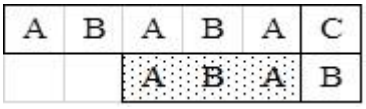
Ezután a $next[4]$ kiszámítása a cél. Ekkor az $M[1..4]$ kezdőrészletnek keressük a legnagyobb egymással megegyező, valódi prefix-szuffix párját. A legnagyobb ilyen valódi prefix-szuffix 3 hosszú lehetne, de $M[1] = M[2]$ egyenlőséget már korábban is megvizsgáltuk és nem teljesült, így ez nem jöhet szóba. Azonban, az előző menetben $M[1] = M[3]$ teljesült, így az ennek megfelelő elcsúsztatott pozíciót megtartva vizsgáljunk tovább, mert további karakter egyezés esetén ez lehetne a leghosszabb prefix-szuffix pár (lásd: 32.9. ábra):



32.9. ábra. A *next* vektor kiszámítása (3)

Valóban az $M[2] = M[4]$ teljesül, így feljegyezzük $next[4] = 2$ értéket.

A $next[5]$ meghatározásához, az előző menethez hasonlóan a mintát nem csúcstatjuk el, hanem a következő karaktert vizsgáljuk (lásd: 32.10. ábra):



32.10. ábra. A *next* vektor kiszámítása (4)

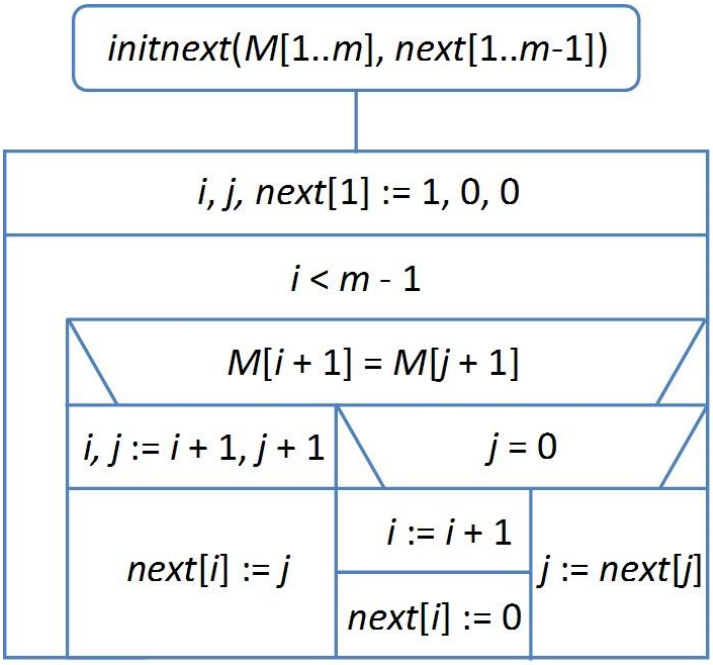
Azt látjuk, hogy $M[3] = M[5]$, így feljegyezzük $next[5] = 3$.

Összefoglaljuk egy ábrán a *next* függvény kiszámítását (lásd: 32.11. ábra):

j	next[j]	A	B	A	B	A	C
1	0						
2	0		A	B	A	B	A
3	1			A	B	A	B
4	2			A	B	A	B
5	3			A	B	A	B

32.11. ábra. A *next* vektor kiszámítása (összefoglalás)

A *next* vektor kitöltésének részletes végigkövetése után már nem nehéz felírni az *initnext* eljárást, amely 32.12. ábrán látható. Ezzel teljessé vált a 32.6. ábrán megadott *KMP* mintaillesztő algoritmus, ugyanis az inicializáló eljárását is megalkottuk.



32.12. ábra. Az initnext algoritmus (*KMP*)

Az *initnext* eljárás különlegessége az, hogy a *KMP* mintaillesztés inicializálására szolgál, de a *next* vektor kitöltésére is lényegében a *KMP* algoritmust használjuk. Ezt az teszi lehetővé, hogy a kitöltés éppen olyan mértékben halad előre a *next* vektoron, mint ami a számítás továbblépéséhez szükséges, amivel egy saját belső inicializáló eljárást valósítunk meg.

A *KMP* algoritmus *műveletigényének* megállapításához vegyük figyelemben, hogy inicializáló tevékenység, az *initnext* eljárás lépésszáma $\Theta(m)$. Tegyük fel, hogy $m << n$; ekkor a keresés műveletigénye legjobb és legrosszabb esetben is egyaránt $\Theta(n)$. A *KMP* algoritmust ezért *stabil* eljárásnak mondhatjuk.

Mivel a *KMP* algoritmus működése során a szövegben csak legfeljebb egy pozícióval történő előre lépést teszünk (nincs visszalépés), így az algoritmus *puffer* használata nélkül is átírható *szekvenciális sorozat*, illetve *fájl* formában adott szövegre.