Algoritmusok és adatszerkezetek II. előadásjegyzet:

Mintaillesztés, Tömörítés

Ásványi Tibor – asvanyi@inf.elte.hu

2023. augusztus 21.

Tartalomjegyzék

Hivatkozások

[1] Ásványi Tibor, Algoritmusok és adatszerkezetek II.
Útmutatások a tanuláshoz, jelölések, tematika,
fák, gráfok,
mintaillesztés, tömörítés
http://aszt.inf.elte.hu/~asvanyi/ad/ad2jegyzet/

[2] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.,
angolul: Introduction to Algorithms (Fourth Edititon), chapter 32 (String Matching) *The MIT Press*, 2022.
magyarul: Új Algoritmusok, 32. fejezet *Scolar Kiadó*, Budapest, 2003. ISBN 963 9193 90 9

[3] Fekete István, Algoritmusok jegyzet
http://ifekete.web.elte.hu/

[4] Rónyai Lajos – Ivanyos Gábor – Szabó Réka, Algoritmusok, *TypoTEX Kiadó*, 1999. ISBN 963 9132 16 0
https://www.tankonyvtar.hu/hu/tartalom/tamop425/2011-0001-526_ronyai_algoritmusok/adatok.html

[5] Weiss, Mark Allen, Data Structures and Algorithm Analysis, *Addison-Wesley*, 1995, 1997, 2007, 2012, 2013.

[6] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet (2022)
http://aszt.inf.elte.hu/~asvanyi/ad/ad1jegyzet/ad1jegyzet.pdf

1. Mintaillesztés (String-matching [2])

1.1. Jelölések. *(Notations)*
$\mathbb{N} = \{\, i \in \mathbb{Z} \mid i \geq 0 \,\}$ $\qquad$ $i\,..\,k = \{\, j \in \mathbb{N} \mid i \leq j \leq k \,\}$
$[i\,..\,k) = \{\, j \in \mathbb{N} \mid i \leq j < k \,\}$ $\qquad$ $(i\,..\,k) = \{\, j \in \mathbb{N} \mid i < j < k \,\}$
$\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_d\}$ *az ábécé (alphabet), ahol* $d \in \mathbb{N} \wedge d > 0$
$T : \Sigma[n]$ *a szöveg (text), ami betűk egy sztringje* 0*-tól* $n-1$*-ig indexelve.*
$T[i\,..\,j) = T[i\,..\,j-1]$
$P : \Sigma[m]$ *a minta (pattern), ahol* $0 < m \leq n$.

A továbbiakban feltesszük, hogy $T : \Sigma[n]$ és $P : \Sigma[m]$, valamint a hosszuk, azaz m és n változatlanok, ahol $1 \leq m \leq n$ (a minta nem üres, és legfeljebb olyan hosszú, mint a szöveg).

A $T : \Sigma[n]$ szövegben keressük a $P : \Sigma[m]$ minta előfordulásait (occurrences). Más szavakkal, a T szövegben P minta azon eltolásait keressük, amelyekre $T[s\,..\,s+m) = P[0\,..\,m)$. Ezekre az eltolásokra $s \in 0\,..\,n-m$ nyilvánvalóan teljesül.

1.2. Definíció.
$s \in 0\,..\,n-m$ *a* P *minta* lehetséges eltolása (possible shift) *a* T *szövegen.*
$s \in 0\,..\,n-m$ érvényes eltolás (valid shift), **ha** $T[s\,..\,s+m) = P[0\,..\,m)$.
Különben $s \in 0\,..\,n-m$ érvénytelen eltolás (invalid shift).

1.3. Probléma. *(Mintaillesztés.) Az érvényes eltolások* V *halmazát szeretnénk meghatározni. Ehhez eltolások egy* S *halmazába gyűjtjük a mintaillesztő keresések futása során talált érvényes eltolásokat. A problémát megoldó algoritmusok közös utófeltétele* $S = V$*, ahol*

$$V = \{\, s \in 0\,..\,n-m \mid T[s\,..\,s+m) = P[0\,..\,m) \,\}.$$

1.1. Egyszerű mintaillesztés (Naive string-matching)

Vegyük bevezetésként a következő példát! A $P[0\,..\,4) = BABA$ mintát keressük a $T[0\,..\,11) = ABABBABABAB$ szövegben. ($\underline{B}$: B-t sikeresen illesztette a szöveg megfelelő betűjére; $\not{B}$: sikertelenül illesztette.)

Ennél az algoritmusnál sorban megvizsgáljuk a lehetséges eltolásokat, és kiszűrjük közülük az érvényeseket. Úgymond nyers erővel (brute-force) oldjuk meg a problémát.

(The naive string-matching [the Brute-Force] algorithm checks each possible shift in order and collects the valid shifts with maximal (i.e. worst-case) time complexity $\Theta((n-m+1) * m)$, which is $\Theta(n^2)$ if $m = \lfloor n/2 \rfloor$. [2])

$i=$	0	1	2	3	4	5	6	7	8	9	10
$T[i]=$	A	B	A	B	B	A	B	A	B	A	B
	~~B~~	A	B	A							
		$\underline{B}$	$\underline{A}$	$\underline{B}$	~~A~~						
			~~B~~	A	B	A					
				$\underline{B}$	~~A~~	B	A				
$s=4$					$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$			
						~~B~~	A	B	A		
$s=6$							$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$	
								~~B~~	A	B	A

$$S = \{\, 4; 6 \,\} = V$$

BruteForce($T : \Sigma[n]$; $P : \Sigma[m]$; $S : \mathbb{N}\{\}$)

$S := \{\}$

$s := 0$ **to** $n - m$

match(T, s, P) // $T[s\,..\,s{+}m) = P[0\,..\,m)$

$S := S \cup \{s\}$ | SKIP

match($T : \Sigma[]$; $s : \mathbb{N}$; $P : \Sigma[m]$) : $\mathbb{B}$

// check, whether $T[s\,..\,s{+}m) = P[0\,..\,m)$

$j := 0$

$j < m \wedge T[s + j] = P[j]$

$j{+}{+}$

return $j \geq m$

A $T[s\,..\,s{+}m) = P[0\,..\,m)$ egyenlőségvizsgálatra: $MT_e(m) \in \Theta(m)$
$mT_e(m) \in \Theta(1)$
Innét a teljes algoritmusra: $MT_{\mathrm{BF}}(n, m) \in \Theta((n - m + 1) * m)$
$mT_{\mathrm{BF}}(n, m) \in \Theta(n - m + 1)$

Ha egy feladatosztályon valamely $0 < k < 1$ konstansra $m \leq k * n$, akkor
$1 * n \geq n - m + 1 > n - k * n = (1 - k) * n$, ahol $1 > 1 - k > 0$ konstans.
Innét a $\Theta(\cdot)$ függvényosztályok definíciója szerint
$n - m + 1 \in \Theta(n)$ és $(n - m + 1) * m \in \Theta(n * m)$.
Ebből pedig az $\cdot \in \Theta(\cdot)$ reláció tranzitivitása miatt:

1.4. Következmény. *Ha egy feladatosztályon a k és az m konstansokra*
$0 < k < 1 \wedge m \leq k * n \Rightarrow mT_{BF}(n, m) \in \Theta(n) \wedge MT_{BF}(n, m) \in \Theta(n * m)$

Másrészt, amennyiben egy feladatosztályon m az n-hez képest nem is elhanyagolható, azaz valamely $\varepsilon > 0$ konstansra $m \geq \varepsilon * n$, akkor
$\varepsilon * n * n \leq n * m \leq n * n$. Következésképp $n * m \in \Theta(n^2)$.
Ebből az 1.4 következménnyel adódik az alábbi eredmény:

1.5. Következmény. *Ha egy feladatosztályon az ε és a k konstansokra* $0 < \varepsilon \leq k < 1 \wedge \varepsilon * n \leq m \leq k * n \Rightarrow MT_{BF}(n, m) \in \Theta(n^2)$

1.2. Quicksearch

A gyorsabb keresés érdekében ennél és a következő (KMP) algoritmusnál általában egynél nagyobb lépésekben növeljük a $P[0\,..\,m)$ minta eltolását a $T[0\,..\,n)$ szöveghez képest úgy, hogy biztosan ne ugorjunk át egyetlen érvényes eltolást sem. Mindkét algoritmus, a tényleges mintaillesztés előtt egy előkészítő fázist hajt végre, ami nem függ a szövegtől, csak a mintától.

A Quicksearch-nél ebben az előkészítő fázisban az ábécé σ elemeihez $shift(\sigma) \in 1\,..\,m{+}1$ címkéket társítunk, ahol $P[0\,..\,m)$ a keresett minta.

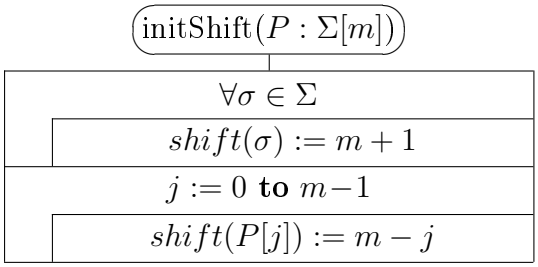
Tegyük fel most, hogy $\sigma = T[s + m]$. Ekkor a $shift(\sigma)$ érték megmondja, hogy a $T[s\,..\,s{+}m) = P[0\,..\,m)$ összehasonlítás után legalább mennyivel kell (jobbra) eltolni a P mintát a szövegen ahhoz, hogy a $T[s{+}m]$ alapján legyen esély a mintának a megfelelő szövegrészhez való illeszkedésére.
– $\sigma \in P[0\,..\,m)$ esetén a $shift(\sigma) \in 1\,..\,m$ érték azt mondja meg, hogy legalább mennyivel kell tovább tolni a P mintát ahhoz, hogy a $T[s + m]$ betűhöz kerülő karaktere maga is σ legyen. Világos, hogy a σ legjobboldali P-beli előfordulásához tartozik a legkisebb ilyen eltolás.
– $\sigma \notin P[0\,..\,m)$ esetén $shift(\sigma) = m + 1$ lesz, azaz a minta *átugorja* a $T[s + m]$ karaktert.

Arra az esetre, amikor az ábécé $\Sigma = \{\text{A,B,C,D}\}$, a minta pedig $P[0\,..\,4)$=CADA, az alábbi félig absztrakt példákban xxxx mutatja a CADA mintával az eltolás előtt összehasonlított szövegrészt, maga a CADA pedig a minta eltolás utáni helyzetét. (Ezután természetesen újabb összehasonlítás kezdődik a szöveg megfelelő része és a minta között stb.)

```
Szöveg: ...xxxxA......xxxxB......xxxxC......xxxxD...
 Minta:     CADA           CADA       CADA      CADA
```

A megfelelő $shift$ értékek értékeket a következő táblázat mutatja.

σ	A	B	C	D
$shift(\sigma)$	1	5	4	2



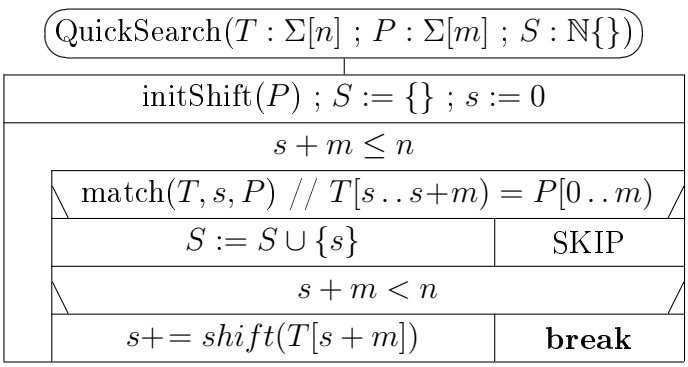
Az ábécé méretét konstansnak tekintve $T_{\text{initShift}}(m) \in \Theta(m)$ adódik.

A $P[0\,..\,4)$=CADA mintával az initShift() majd a Quicksearch() működése:

σ	A	B	C	D
initial $shift(\sigma)$	5	5	5	5
C			4	
A	3			
D				2
A	1			
final $shift(\sigma)$	1	5	4	2

$i=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
$T[i]=$	A	D	A	B	A	B	C	A	D	A	B	C	A	B	A	D	A	C	A	D	A	D	A
	$\not{C}$	A	D	A																			
		$\not{C}$	A	D	A																		
$s=6$							$\underline{C}$	$\underline{A}$	$\underline{D}$	$\underline{A}$													
												$\underline{C}$	$\underline{A}$	$\not{D}$	A								
														$\not{C}$	A	D	A						
$s=17$																		$\underline{C}$	$\underline{A}$	$\underline{D}$	$\underline{A}$		
																				$\not{C}$	A	D	A

$$S = \{\,6; 17\,\} = V$$



$mT(n,m) \in \Theta\left(\frac{n}{m+1} + m\right)$ (pl. ha $T[0\,..\,n)$ és $P[0\,..\,m)$ diszjunktak)
$MT(n,m) \in \Theta((n-m+2)*m)$ (pl. ha $T = AA\ldots A$ és $P = A\ldots A$)

A minimális műveletigény nagyságrenddel jobb, mint az egyszerű mintaillesztésnél, a tényleges (nem az aszimptotikus) maximális futási idő viszont még egy kicsit rosszabb is. Szerencsére, a tapasztalatok szerint az átlagos futási idő inkább a legjobb esethez áll közel, mintsem a legrosszabbhoz. Időkritikus alkalmazásokban viszont egy stabilabb futási idejű, minden esetben hatékony algoritmusra lenne szükségünk.

1.3. Mintaillesztés lineáris időben (Knuth-Morris-Pratt, azaz KMP algoritmus)

A KMP algoritmus futási ideje minden esetben $\Theta(n)$, a legjobb és a legrosszabb eset között körülbelül kétszeres szorzóval, ami nem csak hatékony működést, de nagyon stabil futási időt is jelent. Ellentétben a korábban ismertetett megoldásokkal, sohasem lép vissza a T szövegen, így a KMP algoritmus könnyen implementálható szekvenciális fájlokra.

A következő speciális jelöléseket fogjuk használni.

1.6. Jelölések.

- *Ha x és y két sztring, akkor $x+y$ a konkatenáltjuk. Pl. $x = AB$ és $y = BA$ esetén $x+y = ABBA$ és $y+x = BAAB$.*
- *ε az üres sztring. Nyilván tetszőleges x sztringre $x+\varepsilon = x = \varepsilon + x$.*
- *Ha x és y két sztring, akkor $x \sqsubseteq y$ (x az y* prefixe*) azt jelenti, hogy $\exists z$ sztring, amire $x+z = y$. ($\varepsilon, A, AB, ABB, ABBA \sqsubseteq ABBA$)*
- *Ha x és y két sztring, akkor $x \sqsubset y$ (x az y* valódi prefixe [proper prefix]*) azt jelenti, hogy $x \sqsubseteq y \wedge x \neq y$. ($\varepsilon, A, AB, ABB \sqsubset ABBA$)*
- *Ha x és y két sztring, akkor $x \sqsupseteq y$ (x az y* szuffixe*) azt jelenti, hogy $\exists z$ sztring, amire $z+x = y$. ($ABBA, BBA, BA, A, \varepsilon \sqsupseteq ABBA$)*
- *Ha x és y két sztring, akkor $x \sqsupset y$ (x az y* valódi szuffixe [proper suffix]*) azt jelenti, hogy $x \sqsupseteq y \wedge x \neq y$. ($BBA, BA, A, \varepsilon \sqsupset ABBA$)*
- *$P_{:j} = P[0\,..\,j) = P[0\,..\,j-1]$ (csak ebben az alfejezetben) $P_{:j}$ a P sztring j hosszúságú prefixe, azaz kezdőszelete. $P_{:0} = \varepsilon$ az üres prefixe. Hasonlóan $T_{:i} = T[0\,..\,i)$.*
  [*Eszerint minden sztringnek* szuffixe *az üres sztring, azaz $P_{:0} \sqsupseteq P_{:j}$*

$(j \in 0\,..\,m)$, *és minden nemüres sztringnek* valódi szuffixe *az üres sztring, azaz* $P_{:0} \sqsupset P_{:j} \quad (j \in 1\,..\,m)$.]

A sztringek alábbi, nyilvánvaló tulajdonságai hasznosak lesznek. [2].

1.7. Lemma. *A szuffixum reláció tranzitivitása (Transitivity of the suffix relation)* $\quad x \sqsupset y \wedge y \sqsupseteq z \Rightarrow x \sqsupset z.$

1.8. Lemma. *Átfedő szuffix (Overlapping-suffix) lemma*
Tegyük fel, hogy x, y és z olyan sztringek, amelyekre $x \sqsupseteq z$ *és* $y \sqsupseteq z$.
$|x| \leq |y| \Rightarrow x \sqsupseteq y. \quad |x| < |y| \Rightarrow x \sqsupset y. \quad |x| = |y| \Rightarrow x = y.$

1.9. Lemma. *Szuffix kiterjesztési (Suffix-extension) lemma*
$P_{:j} \sqsupseteq T_{:i} \wedge P[j] = T[i] \iff P_{:j+1} \sqsupseteq T_{:i+1}.$
$P_{:i} \sqsupset P_{:j} \wedge P[i] = P[j] \iff P_{:i+1} \sqsupset P_{:j+1}.$

Bevezetés a KMP algoritmushoz: Tekintsük a következő példát!

1.10. Példa. *Feltesszük, hogy van egy hosszabb T szövegünk, de most ennek csak a* $T[i-5\,..\,i+2] = BABABABB$ *részét vesszük figyelembe. A minta a* $P_{:6} = BABABB$. *Az aktuális eltolás* $i-5$. *A sikeresen illesztett betűket aláhúztuk. A sikertelenül illesztett karaktert pedig áthúztuk.*

...	T_{i-5}	T_{i-4}	T_{i-3}	T_{i-2}	T_{i-1}	T_i	T_{i+1}	T_{i+2}
...	B	A	B	A	B	A	B	B
$P_{:6} =$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\not{B}$		
			B	A	B	$\underline{A}$	$\underline{B}$	$\underline{B}$

A táblázat harmadik sorában, sikeresen illesztettük a minta $P_{:5}$ *kezdőszeletét a* $T[i-5\,..\,i)$ *szövegrészhez, de* $P[5] \neq T[i]$. *Következésképpen* $i-5$ *egy érvénytelen eltolás.*

Most a legkisebb további eltolást *keressük úgy, hogy a minta* $P_{:5}$ *kezdőszeletének az a* $P_{:k}$ $(0 \leq k < 5)$ *prefixe, ami még a szöveg* $T[i-k\,..\,i)$ *része alatt van, illeszkedjen arra, azaz* $P_{:k} = T[i-k\,..\,i)$ *teljesüljön. (Lásd a fenti táblázat utolsó sorát!) Ez a* $P_{:k}$*-t meghatározó* legkisebb további eltolás *legfeljebb 5, mert* $P_{:0} \sqsupset T_{:i}$. *Ezzel a* $P_{:k}$*-t meghatározó eltolással biztosan nem ugrunk át egyetlen érvényes eltolást sem. A mi esetünkben két betűvel kell tovább tolni a mintát és* $k = 3$. *Ezután azt találjuk, hogy* $P[3] = T[i]$, $P[4] = [i+1]$ *és* $P[5] = T[i+2]$. *Következőleg* $i-3$ *egy érvényes eltolás. Bármely ennél nagyobb eltolás átugorta volna az* $i-3$ *érvényes eltolást.*

Az előbbi példa felveti a kérdést, hogyan lehetne a k értékét hatékonyan meghatározni. Bár a példa szerint továbbra is $j = 5$, a következő gondolatmenet tetszőleges $P_{:m}$ minta és $T_{:n}$ szöveg esetében alkalmazható, ha $j \in 1\,..\,m$, ahol $i-j$ az aktuális eltolás, $P_{:j} = T[i-j\,..\,i)$ azaz $P_{:j} \sqsupseteq T_{:i}$, valamint $(P[j] \neq T[i] \vee j = m)$.

Világos, hogy nagyobb *további eltoláshoz* kisebb k érték, míg kisebb *további eltoláshoz* nagyobb k érték tartozik, mivel a *további eltolás* $+\ k = j$. Ezenkívül k a *legkisebb további eltoláshoz* tartozik, amelyre $P_{:k} \sqsupset T_{:i}$, ahol ez a *további eltolás* legfeljebb j, ui. $P_{:0} \sqsupset T_{:i}$. Ezért k a legnagyobb h, amire $P_{:h} \sqsupset T_{:i} \wedge 0 \leq h < j$ teljesül. Továbbá $P_{:h} \sqsupset T_{:i}$ ekvivalens a $P_{:h} \sqsupset P_{:j}$ relációval, mert $P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq h < j$. (Más szavakkal, $P[0\,..\,h) = T[i-h\,..\,i)$ ekvivalens a $P[0\,..\,h) = P[j-h\,..\,j)$ relációval, mert $P[0\,..\,j) = T[i-j\,..\,i) \wedge 0 \leq h < j$.)

Következésképpen k csak $P_{:j}$-től függ. Mivel $P_{:m}$ fix, k csak j-től függ. Más szavakkal, a leghosszabb $P_{:h}$-t keressük, amire $P_{:h} \sqsubset P_{:j} \wedge P_{:h} \sqsupset P_{:j}$. Ennek hosszát a π prefix függvény határozza meg.

1.11. Definíció. $\pi(j) = \max\{\, h \in 0\,..\,j-1 \mid P_{:h} \sqsupset P_{:j} \,\} \quad (j \in 1\,..\,m)$

Ennek a függvénynek a hatékony kiszámítását később, 1.3.2-ben részletezzük. Előbb azonban megnézzük a KMP algoritmus még egy, konkrét esetét, majd 1.3.1-ben formálisan is megadjuk és elemezzük a fő eljárás struktogramját. A π prefix függvény következő tulajdonságai megkönnyítik a számolást.

1.12. Tulajdonság. $j \in 1\,..\,m \Rightarrow \pi(j) \in 0\,..\,j-1$

Bizonyítás. A 1.11. definíció szerint $\pi(j)$ a $0\,..\,j-1$ egy részhalmazának a maximuma. □

1.13. Lemma. $j \in [1\,..\,m) \Rightarrow \pi(j+1) \leq \pi(j)+1$

Bizonyítás.
– Ha $\pi(j+1) = 0 \Rightarrow \pi(j+1) = 0 \leq 0+1 \leq \pi(j)+1$ mivel $\pi(j) \geq 0$ az 1.12. tulajdonság szerint.
– Ha $\pi(j+1) > 0 \Rightarrow$ a prefix függvény definíciója (1.11) szerint
$P_{:(\pi(j+1)-1)+1} = P_{:\pi(j+1)} \sqsupset P_{:j+1} \Rightarrow$ az 1.9. lemmával $P_{:\pi(j+1)-1} \sqsupset P_{:j} \Rightarrow$ újra az 1.11. definícióval $\pi(j+1)-1 \leq \pi(j) \Rightarrow \pi(j+1) \leq \pi(j)+1$. □

1.14. Példa. *Kiszámítjuk a π függvényt az 1.11. definíció, az 1.12. tulajdonság és az 1.13. lemma szerint a $P_{:8} = P[0\,..\,8) = BABABBAB$ mintára.*

$P[j-1] =$	B	A	B	A	B	B	A	B
$j =$	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*
$\pi(j) =$	*0*	*0*	*1*	*2*	*3*	*1*	*2*	*3*

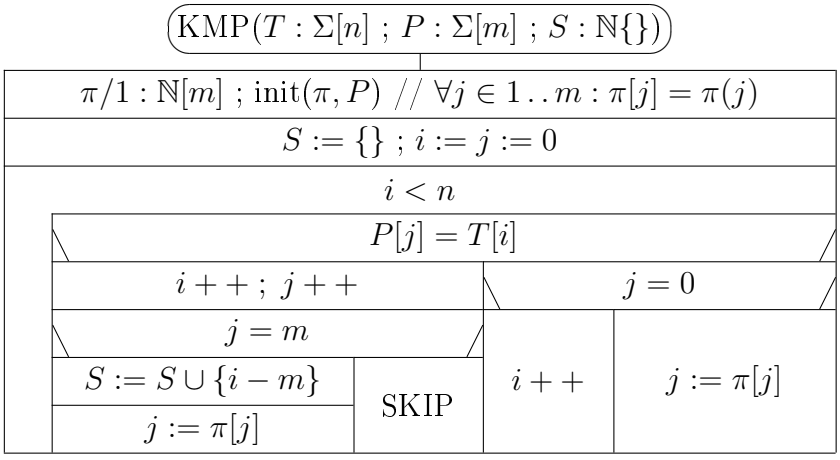
1.15. Példa. *Alkalmazzuk az eddig tanultakat a következő esetben! A* $P_{:8} = P[0\,..\,8) = BABABBAB$ *minta előfordulásait keressük a* $T_{:18} = T[0\,..\,18) = ABABABABBABABABBAB$ *szövegben. (A minta elején a jelöletlen betűkről „illesztés nélkül is tudja" az algoritmus, hogy illeszkednek a szöveg megfelelő karakterére.* $\underline{B}$*: B-t sikeresen illesztette a szöveg megfelelő betűjére;* $\not{B}$*: sikertelenül illesztette.)*

$i=$	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$T[i]=$	A	B	A	B	A	B	A	B	B	A	B	A	B	A	B	B	A	B
	$\not{B}$																	
		$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\not{B}$											
$s=3$				B	A	B	$\underline{A}$	$\underline{B}$	$\underline{B}$	$\underline{A}$	$\underline{B}$							
									B	A	B	$\underline{A}$	$\underline{B}$	$\not{B}$				
$s=10$											B	A	B	$\underline{A}$	$\underline{B}$	$\underline{B}$	$\underline{A}$	$\underline{B}$
																B	A	B

$$S = \{\,3; 10\,\} = V$$

1.3.1. A KMP algoritmus fő eljárása

Az 1.10. példával kapcsolatos megfontolások alapján megadjuk a KMP algoritmus fő eljárásának struktogramját.



A 1.3.2 alfejezetben fogjuk látni, hogy az init(π, P) eljáráshívás a π prefix-függvény értékeit összegyűjti a π tömbbe, $\Theta(m)$ műveletigénnyel. Az init(π, P) eljáráshívás utófeltétele az, hogy $(\forall j \in 1\,..\,m : \pi[j] = \pi(j))$.

A fő eljárás elemzése a ciklusa 1.17. invariánsán alapszik, ahol $i-j$ a P minta aktuális eltolása a T szövegen. Az invariáns helyességének bizonyításához hasznos lesz a következő lemma.

1.16. Lemma. $j \in 1\,..\,m \wedge P_{:j} \sqsupseteq T_{:i} \Rightarrow$
nincs érvényes eltolás az $(i{-}j\,..\,i{-}\pi(j))$ *intervallumban.*

Bizonyítás. Tegyük fel indirekt módon, hogy $k \in (\pi(j)\,..\,j)$ és $i{-}k$ érvényes eltolás! Ez azt jelenti, hogy $T[i{-}k\,..\,i{-}k{+}m) = P[0\,..\,m)$. Nyilván $k < j \leq m$. Innét $k < m$, amiből $i{-}k{+}m > i$ adódik. Ezért $T[i{-}k\,..\,i) = P[0\,..\,k)$, vagy másképp írva $P_{:k} \sqsupseteq T_{:i}$. Továbbá $P_{:j} \sqsupseteq T_{:i} \wedge k < j$. Következésképpen $P_{:k} \sqsupset P_{:j}$ az Átfedő szuffix lemma (1.8) miatt, de $k > \pi(j)$, amiből viszont $P_{:k} \not\sqsupset P_{:j}$ következik, ui. $P_{:\pi(j)}$ a $P_{:j}$ leghosszabb valódi prefixe, ami egyben szuffixe is, a π prefix függvény definíciója (1.11) alapján. □

1.17. Tétel. *Az* (Inv) *állítás a* $\mathrm{KMP}(T,P,S)$ *eljárás ciklusának invariánsa.*
(Inv) $\quad P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i \leq n \wedge j < m \ \wedge S = V \cap [0\,..\,i{-}j).$

Bizonyítás. Kezdetben $i = j = 0$ miatt (Inv) a következő nyilvánvaló állításnak felel meg.
$P_{:0} \sqsupseteq T_{:0} \wedge 0 \leq 0 \leq 0 \leq n \wedge 0 < m \ \wedge S = V \cap [0\,..\,0) = V \cap \{\} = \{\}.$

A továbbiakban igazoljuk, hogy a ciklusmag végrehajtása (mind a négy programágon) tartja (Inv)-et. Állításainkhoz mindig hozzáértjük $\mathrm{init}(\pi,P)$ utófeltételét. Feltéve, hogy $i < n$, akkor belépünk a ciklusba, és
(Inv1) $\quad P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i < n \wedge j < m \ \wedge S = V \cap [0\,..\,i{-}j)$ $\quad$ teljesül.

- Ha $P[j] = T[i]$, akkor a Szuffix kiterjesztési lemma (1.9) szerint $P_{:j+1} \sqsupseteq T_{:i+1}$, majd i és j növelése után
  (Inv2) $\quad P_{:j} \sqsupseteq T_{:i} \wedge 0 < j \leq i \leq n \wedge j \leq m \ \wedge S = V \cap [0\,..\,i{-}j).$

  1. Amennyiben $j = m$, akkor $P_{:m} \sqsupseteq T_{:i}$, vagy másképpen írva $P[0\,..\,m) = T[i{-}m\,..\,i)$. Tehát $i{-}m$ érvényes eltolás, amit S-hez hozzávéve
     (Inv3) $\quad P_{:j} \sqsupseteq T_{:i} \wedge 0 < j \leq i \leq n \wedge j \leq m \ \wedge S = V \cap [0\,..\,i{-}j].$
     Mivel $j \in 1\,..\,m \wedge P_{:j} \sqsupseteq T_{:i}$, az 1.16. lemma szerint nincs érvényes eltolás az $(i{-}j\,..\,i{-}\pi(j))$ intervallumban. Ezért
     $P_{:j} \sqsupseteq T_{:i} \wedge 0 < j \leq i \leq n \wedge j \leq m \ \wedge S = V \cap [0\,..\,i{-}\pi(j)).$
     Figyelembe véve, hogy $P_{:\pi(j)} \sqsupset P_{:j} \sqsupseteq T_{:i}$, a szuffixum reláció tranzitivitása (1.7. lemma) és $\pi[j] = \pi(j)$ alapján:
     $P_{:\pi[j]} \sqsupset T_{:i} \wedge 0 < j \leq i \leq n \wedge j \leq m \ \wedge S = V \cap [0\,..\,i{-}\pi[j]).$
     Tekintettel arra, hogy $\pi[j] = \pi(j) \in [0\,..\,j)$, a $j := \pi[j]$ értékadás után már $0 \leq j < i \wedge j < m$ teljesül. Innét
     $P_{:j} \sqsupset T_{:i} \wedge 0 \leq j < i \leq n \wedge j < m \ \wedge S = V \cap [0\,..\,i{-}j)$ adódik, amiből az első programág végén közvetlenül következik (Inv).
  2. Amennyiben $j \neq m$, akkor viszont (Inv2) szerint $j < m$, és így a második programág végén is teljesül (Inv).

- Ha $P[j] \neq T[i]$, akkor a Szuffix kiterjesztési lemma (1.9) szerint $P_{:j+1} \not\sqsupseteq T_{:i+1}$, vagy ezt másképpen írva $P[0\,..\,j] \neq T[i-j\,..\,i]$. (Inv1)-ből $j < m$ figyelembevételével ebből $P[0\,..\,m) \neq T[i-j\,..\,i-j+m)$ következik. Tehát az $i-j$ érvénytelen eltolás. Ebből az (Inv1), azaz a $P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i < n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j)$ tulajdonsággal (Inv4) $P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i < n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j]$ adódik.

  3. Ha $j = 0$, akkor (Inv4) figyelembevételével a $P_{:0} \sqsupseteq T_{:i} \wedge 0 = j \leq i < n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j]$ állítást kapjuk, amiből i növelése után $P_{:0} \sqsupseteq T_{:i} \wedge 0 = j \leq i \leq n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j)$ következik, tehát a harmadik programág végén is teljesül (Inv) $P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i \leq n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j)$.
  4. Ha viszont $j \neq 0$, akkor (Inv4) figyelembevételével $P_{:j} \sqsupseteq T_{:i} \wedge 0 < j \leq i < n \wedge j < m \ \wedge S = V \cap [0\,..\,i-j]$ teljesül. Ennek közvetlen következménye (Inv3) $P_{:j} \sqsupseteq T_{:i} \wedge 0 < j \leq i \leq n \wedge j \leq m \ \wedge S = V \cap [0\,..\,i-j]$. Az első programág vizsgálatánál már láttuk, hogy (Inv3) esetén a $j := \pi[j]$ értékadást végrehajtva teljesül (Inv), tehát az utolsó programág végén is igaz.

□

A KMP(T, P, S) eljárás parciális helyessége:

1.18. Tétel. *Ha a KMP algoritmus megáll, akkor megoldja az 1.3. (mintaillesztési) problémát, azaz $S = V$ teljesül, amikor a* KMP(T, P, S) *eljárás befejeződik.*

Bizonyítás. A KMP(T, P, S) eljárás ciklusának (Inv) invariánsa (1.17), azaz $P_{:j} \sqsupseteq T_{:i} \wedge 0 \leq j \leq i \leq n \wedge j < m \wedge S = V \cap [0\,..\,i-j)$, és a ciklusfeltétel tagadása, vagyis $i \geq n$ szerint $i = n \wedge j < m \wedge S = V \cap [0\,..\,n-j)$ teljesül, mikor a ciklus befejeződik. Ezenkívül, $j < m \Rightarrow n-j > n-m \Rightarrow [0\,..\,n-j) \supseteq 0\,..\,n-m \supseteq \{\, s \in 0\,..\,n-m \mid T[s\,..\,s+m) = P[0\,..\,m) \,\} = V \Rightarrow [0\,..\,n-j) \supseteq V$. Ezért $S = V \cap [0\,..\,n-j) = V$. Következőleg, $S = V$ teljesül, amikor a KMP(T, P, S) eljárás befejeződik. □

A KMP(T, P, S) eljárás megállása és hatékonysága: A terminálás igazolásához az 1.3.2. alfejezetben be fogjuk látni, hogy az init(π, P) eljárás $\Theta(m)$ idő alatt lefut, most pedig azt, hogy a KMP(T, P, S) eljárás ciklusa legfeljebb $2n$ iterációt hajt végre.

A KMP(T, P, S) eljárás lineáris, pontosabban $\Theta(n)$ műveletigényének igazolásához elég belátni, hogy a fő ciklusának a futási ideje $\Theta(n)$, ui. $m \in 1\,..\,n$ miatt ezen az init(π, P) eljárás (1.3.2) és egyéb inicializálások $\Theta(m)$ műveletigénye aszimptotikus nagyságrendben már nem módosít. A fő ciklus $\Theta(n)$ műveletigényének igazolásához pedig elegendő azt belátni, hogy az legalább n és legfeljebb $2n$ iterációt hajt végre. A ciklus (Inv) invariánsa (1.17) szerint $i \in 0\,..\,n \wedge j \in 0\,..\,(m{-}1) \wedge j \leq i$.

- Mivel az első iteráció előtt $i = 0$, és mindegyik iteráció legfeljebb eggyel növeli az i változót, továbbá a ciklusfeltétel $i < n$, azért legalább n iteráció szükséges ahhoz, hogy $i = n$ legyen, és az eljárás befejeződjék.
- Tekintsük a $2i{-}j$ kifejezést! Az $i \in 0\,..\,n \wedge j \in 0\,..\,(m{-}1) \wedge j \leq i$ invariáns tulajdonság miatt $2i{-}j \in 0\,..\,2n$.

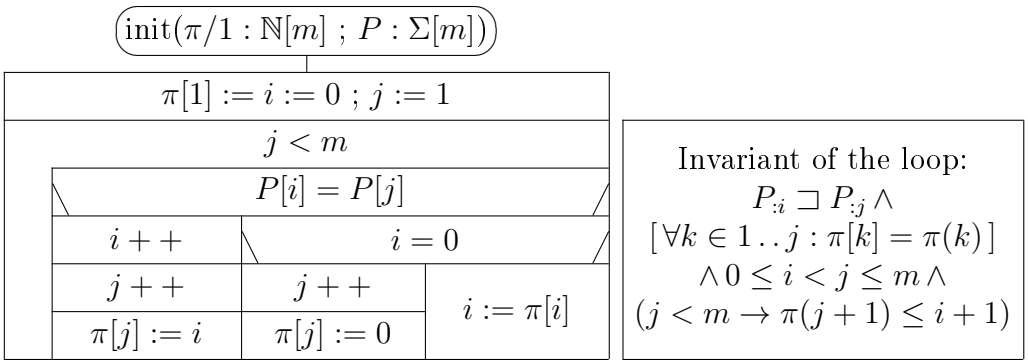
  Az első iteráció előtt $2i{-}j = 0$, ami mindegyik iterációval (mind a négy programágon) szigorúan monoton nő, és végig $2i{-}j \leq 2n$, így legfeljebb $2n$ iteráció után megáll a ciklus.

1.3.2. A π prefix tömb inicializálása

A következő, a π prefix *függvényre* vonatkozó lemma hasznos lesz.

1.19. Lemma. $P_{:i} \sqsupset P_{:j} \wedge 0 < i < j < m \wedge \pi(j{+}1) \leq i \Rightarrow \pi(j{+}1) \leq \pi(i){+}1$

Bizonyítás. Ha $\pi(j{+}1) = 0$, akkor $\pi(j{+}1) < 0{+}1 \leq \pi(i){+}1$, mert definíció szerint $\pi(i) \geq 0$. Másrészt, amennyiben $\pi(j+1) > 0$, $k := \pi(j+1) - 1$. Így $k \geq 0$ és $k + 1 = \pi(j + 1)$. A π függvény definíciója szerint $P_{:k+1} \sqsupset P_{:j+1}$. Következésképpen $P_{:k} \sqsupset P_{:j}$. Figyelembe véve, hogy $P_{:i} \sqsupset P_{:j}$ és $k < i$, a $P_{:k} \sqsupset P_{:i}$ állítást kapjuk. Következőleg $k \leq \pi(i)$. Innét pedig $\pi(j + 1) = k + 1 \leq \pi(i) + 1$ adódik. □



Az init(π, P) eljárás elemzése a kódja mellett, és az alábbi, 1.20. tételben található ciklusinvariánsán alapszik.

1.20. Tétel. *Az* (inv) *állítás az* $\mathrm{init}(\pi, P)$ *eljárás ciklusának invariánsa.*

(inv) $\quad P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge$
$\quad 0 \le i < j \le m \wedge (\,j < m \to \pi(j+1) \le i+1\,).$

Bizonyítás.Közvetlenül az első ciklusiteráció előtt a $\pi[1] := i := 0$; $j := 1$ inicializálások miatt (inv) a következő állításnak felel meg: $P_{:0} \sqsupset P_{:1} \wedge (\,\forall k \in 1\,..\,1 : \pi[k] = \pi(k) = \pi(1) = 0\,) \wedge 0 \le 0 < 1 \le m \wedge (\,1 < m \to \pi(2) \le 1\,)$. Ez utóbbi tényezői igazak, sorban, mert a $P_{:0}$ üres sztring bármely nem nemüres sztringnek valódi szuffixe, továbbá az 1.12. tulajdonság szerint $\pi(1) = 0$, valamint a P minta m mérete nem nulla, és végül a 1.13. lemma alapján $\pi(1+1) \le \pi(1) + 1 = 1$, feltéve, hogy $m > 1$.

Belátjuk még, hogy a ciklusiterációk tartják az (inv) tulajdonságot, azaz, ha tetszőleges ciklusiteráció előtt (inv) és a ciklusfeltétel igaz, akkor a ciklusmag bármelyik ágának a végére jutva ugyancsak igaz lesz. Amikor belépünk a ciklusmagba, akkor (inv) és a ciklusfeltétel alapján (inv1) teljesül:

(inv1) $\quad P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge$
$\quad 0 \le i < j < m \wedge \pi(j+1) \le i+1.$

1. Ha $P[i] = P[j]$, akkor a az 1.9. lemma és (inv1)-ből $P_{:i} \sqsupset P_{:j}$ alapján $P_{:i+1} \sqsupset P_{:j+1}$. Innét a π prefix függvény definíciója (1.11) szerint $\pi(j+1) \ge i+1$, (inv1) alapján pedig $\pi(j+1) \le i+1$. Következésképpen $\pi(j+1) = i+1$. Az $i{+}{+}; j{+}{+}; \pi[j] := i$ értékadások után pedig $P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge 0 < i < j \le m \wedge \pi(j) = i$. Amennyiben $j < m$, az 1.13. lemma és $\pi(j) = i$ szerint $\pi(j+1) \le \pi(j) + 1 = i + 1$. A ciklusmag első ágának végén tehát teljesül az (inv) állítás.

2-3. Ha $P[i] \neq P[j]$, akkor az 1.9. lemma alapján $P_{:i+1} \not\sqsupset P_{:j+1}$. Innét a π prefix függvény definíciója (1.11) szerint $\pi(j+1) \neq i+1$, (inv1) alapján pedig $\pi(j+1) \le i+1$, tehát $\pi(j+1) \le i$.
Ezt (inv1)-gyel összevetve, a belső elágazás előtt (inv2) teljesül:

(inv2) $\quad P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge$
$\quad 0 \le i < j < m \wedge \pi(j+1) \le i.$

   2. Amennyiben $i = 0$, akkor (inv2)-ből a $\pi(j+1) \le i$ állítást figyelembe véve $\pi(j+1) = 0$ adódik, mivel a π függvény nemnegatív. Ezt (inv2)-vel összevetve, a $j{+}{+};\ \pi[j] := 0$ értékadások után $P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge 0 = i < j \le m \wedge \pi(j) = i$. Amennyiben $j < m$, az 1.13. lemma és $\pi(j) = i$ szerint $\pi(j+1) \le \pi(j) + 1 = i + 1$. Tehát a ciklusmag második ágának végén is teljesül az (inv) állítás.

3. Amennyiben $i \neq 0$, akkor (inv2) szerint (inv3) teljesül:
   (inv3) $\quad P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge$
   $0 < i < j < m \wedge \pi(j+1) \leq i.$
   Mivel $i > 0$, az 1.19. lemma feltételei teljesülnek. Tehát $\pi(j+1) \leq \pi(i) + 1$. Másrészt, (inv3) 2. és 3. tényezője figyelembevételével $\pi[i] = \pi(i)$. Ebből π prefix függvény definíciójával (1.11) $P_{:\pi[i]} \sqsupset P_{:i}$ adódik. Ebből az (inv3) a $P_{:i} \sqsupset P_{:j}$ állításával együtt, az 1.7. tranzitivitási lemma alapján $P_{:\pi[i]} \sqsupset P_{:j}$.
   Ezekből (inv3) alapján, az $i := \pi[i]$ értékadás után
   $P_{:i} \sqsupset P_{:j} \wedge (\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,) \wedge$
   $0 \leq i < j < m \wedge \pi(j+1) \leq i + 1.$
   Tehát a ciklusmag utoló ágának a végén is teljesül az (inv) állítás.

□

Az init(π, P) eljárás parciális helyessége:

1.21. Következmény. *Ha az* init(π, P) *eljárás megáll, akkor a visszatérésekor teljesül az utófeltétele:*

$$\forall k \in 1\,..\,m : \pi[k] = \pi(k)$$

Bizonyítás. Az *init*(π, P) eljárásnak az 1.20. tételből ismerős (inv) invariánsa és a ciklusfeltétel tagadása, azaz $j \geq m$ alapján $j = m$. Figyelembe véve még az (inv) invariáns $(\,\forall k \in 1\,..\,j : \pi[k] = \pi(k)\,)$ tényezőjét, az init(π, P) eljárás fenti utófeltétele azonnal adódik. □

Az init(π, P) eljárás megállása és hatékonysága: A megálláshoz az alábbiakban igazoljuk, hogy az eljárás ciklusa legfeljebb $2m-2$ iterációt hajt végre.

Mivel a ciklus minden egyes iterációja $\Theta(1)$ műveletigényű, a lineáris, pontosabban $\Theta(m)$ műveletigény igazolásához elég belátni, hogy az init(π, P) eljárás ciklusa legalább $m-1$ és legfeljebb $2m-2$ iterációt hajt végre. A ciklus (inv) invariánsa szerint $0 \leq i < j \leq m$.

- Mivel az első iteráció előtt $j = 1$, továbbá mindegyik iteráció legfeljebb eggyel növeli a j változót, és a ciklusfeltétel $j < m$, legalább $m-1$ iteráció szükséges ahhoz, hogy $j = m$ legyen, és az eljárás befejeződjék.

- Tekintsük a $2j-i$ kifejezést! $0 \leq i < j \leq m$ miatt $2j-i \in 2\,..\,2m$; ui. $j-i \geq 1 \wedge j \geq 1 \Rightarrow 2j-i \geq 2$; továbbá $j \leq m \Rightarrow 2j \leq 2m \Rightarrow 2j-i \leq 2m$, mivel $i \geq 0$.

Az első iteráció előtt $2j-i = 2$, ami mindegyik iterációval (mind a három programágon) szigorúan monoton nő, és végig $2j-i \leq 2m$, így legfeljebb $2m-2$ iteráció után megáll a ciklus.

1.3.3. A KMP algoritmus összegzése

A fentiekben egy *viszonylag* egyszerű és rövid bizonyítást sikerült adnunk a lineáris műveletigényű KMP algoritmus helyességére és hatékonyságára. (Érdemes összehasonlítani a [2]-ben olvasható bizonyítással. Kikerültük a *mintaillesztő automaták* bevezetését, elemzését és szimulálását.) Ehhez

- megvizsgáltuk a kérdéses sztringek megfelelő tulajdonságait,
- meghatároztuk a KMP(T, P, S) és az init(π, P) eljárások ciklusainak megfelelő invariáns tulajdonságait, valamint
- a ciklusokhoz tartozó alkalmas terminátor kifejezéseket.

Láttuk, hogy a KMP algoritmusnál legjobb és a legrosszabb eset absztrakt műveletigénye között kétszeres szorzó van, így a futási idő is nagyon stabil, és a legrosszabb esetben is meglepően hatékony, a szöveg hosszának közelítőleg lineáris függvénye. Online alkalmazásoknál a hatékonyság (a legrosszabb esetben is) és a futási idő stabilitása együtt, határozott előny a másik két algoritmusal szemben, bár a Quicksearch várható futási ideje jobb, mint a KMP algoritmusé, így offline alkalmazásoknál ez lehet előnyösebb.

A KMP(T, P, S) eljárás i változója sohasem csökken. Mivel a $T[0\,..\,n)$ szövegre csak a $T[i]$ kifejezésen keresztül hivatkozunk, a szövegen sohasem kell visszalépni. Ezért ez a KMP algoritmus kényelmesen, hatékonyan implementálható abban az esetben is, ha a szöveg (amiben keressük a mintát) egy szekvenciális fájlban van. Mivel a Brute-force és a Quicksearch algoritmusoknál a szövegen esetleg $m-2$ karakternyit is vissza kell lépni, ezeknél – feltéve, hogy a szöveg egy szekvenciális input fájlban van – annak az utoljára beolvasott $m-1$ karakterét folyamatosan nyilván kell tartani egy átmeneti tárolóban.

1.3.4. A KMP algoritmus szemléltetése

Az init(π, P) algoritmus szemléltetése az $ABABBABA$ mintán:
(A három programág mindegyikének az elején kezdünk új sort.)

i	j	$\pi[j]$	$\overset{0}{A}$	$\overset{1}{B}$	$\overset{2}{A}$	$\overset{3}{B}$	$\overset{4}{B}$	$\overset{5}{A}$	$\overset{6}{B}$	$\overset{7}{A}$
0	1	0		$\not{A}$						
0	2	0			$\underline{A}$					
1	3	1			A	$\underline{B}$				
2	4	2			A	B	$\not{A}$			
0	4	2					$\not{A}$			
0	5	0						$\underline{A}$		
1	6	1						A	$\underline{B}$	
2	7	2						A	B	$\underline{A}$
3	8	3								

A végeredmény:

$P[j-1] =$	A	B	A	B	B	A	B	A
$j =$	1	2	3	4	5	6	7	8
$\pi[j] =$	0	0	1	2	0	1	2	3

Példa:
A $P[0\,..\,8) = ABABBABA$ mintát keressük
a $T[0\,..\,17) = ABABABBABABBABABA$ szövegben.

A mintához tartozó $\pi/1 : \mathbb{N}[8]$ tömböt fentebb már kiszámoltuk.

A keresés:

$i = 0$		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
$T[i]=$	A	B	A	B	A	B	B	A	B	A	B	B	A	B	A	B	A
	$\underline{A}$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\not{B}$												
s=2			A	B	$\underline{A}$	$\underline{B}$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$							
s=7								A	B	A	$\underline{B}$	$\underline{B}$	$\underline{A}$	$\underline{B}$	$\underline{A}$		
													A	B	A	$\underline{B}$	$\not{B}$
															A	B	$\underline{A}$

$$S = \{\, 2; 7 \,\} = V$$

2. Információtömörítés ([4] 5)

2.1. Naiv módszer

A tömörítendő szöveget karakterenként, fix hosszúságú bitsorozatokkal kódoljuk.

$\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_d \rangle$ az ábécé.

Egy-egy karakter $\lceil \lg d \rceil$ bittel kódolható, ui. $\lceil \lg d \rceil$ biten $2^{\lceil \lg d \rceil}$ különböző bináris kód ábrázolható, és $2^{\lceil \lg d \rceil} \geq d > 2^{\lceil \lg d \rceil - 1}$, azaz $\lceil \lg d \rceil$ biten ábrázolható d-féle különböző kód, de eggyel kevesebb biten már nem.

$In : \Sigma\langle\rangle$ a tömörítendő szöveg. $n = |In|$ jelöléssel $n * \lceil \lg d \rceil$ bittel kódolható.

Pl. az ABRAKADABRA szövegre $d = 5$ és $n = 11$, ahonnét a tömörített kód hossza $11 * \lceil \lg 5 \rceil = 11 * 3 = 33$ bit. (A 3-bites kódok közül tetszőleges 5 kioszthatóaz 5 betűnek.) A tömörített fájl a kódtáblázatot is tartalmazza.

A fenti ABRAKADABRA szöveg kódtáblázata lehet pl. a következő:

karakter	kód
A	000
B	001
D	010
K	011
R	100

A fenti kódtáblázattal a tömörített kód a következő lesz:
00000110000001100001000000011000000.

Ez a tömörített fájlba foglalt kódtáblázat alapján könnyedén 3 bites szakaszokra bontható és kitömöríthető. A kódtáblázat mérete miatt a gyakorlatban csak hosszabb szövegeket érdemes így tömöríteni.

2.2. Huffman-kód

A tömörítendő szöveget karakterenként, változó hosszúságú bitsorozatokkal kódoljuk. A gyakrabban előforduló karakterek kódja rövidebb, a ritkábban előfordulóké hosszabb.

Prefix-mentes kód: Egyetlen karakter kódja sem prefixe semelyik másik karakter kódjának sem.

A karakterenként kódoló tömörítések között a Huffman-kód hossza minimális. Ugyanahhoz a szöveghez többféle kódfa és hozzátartozó kódtáblázat építhető, de mindegyik segítségével az input szövegnek ugyanolyan hosszú tömörített kódját kapjuk. Betömörítés a kódtáblával, kitömörítés a kódfával. Ezért a tömörített fájl a kódfát is tartalmazza.

A **tömörítendő fájlt, illetve szöveget kétszer olvassa végig.**
- Először meghatározza a szövegben előforduló karakterek halmazát és az egyes karakterek gyakoriságát, majd ennek alapján kódfát, abból pedig kódtáblázatot épít.
- Másodszorra a kódtábla alapján kiírja az output fájlba sorban a karakterek bináris kódját.

A **kódfa** szigorúan bináris fa. Mindegyik karakterhez tartozik egy-egy levele, amit a karakteren kívül annak gyakorisága, azaz előfordulásainak száma is címkéz. A belső csúcsokat a csúcshoz tartozó részfa leveleit címkéző karakterek gyakoriságainak összegével címkézzük. (Így a kódfa gyökerét a tömörítendő szöveg hossza címkézi.)

A **kódfát** úgy**építjük** fel, hogy először egycsúcsú fák egy minimum-prioritásos sorát határozzuk meg, amelyben mindegyik karakter pontosan egy csúcsot címkéz. A csúcsot a karakteren kívül annak gyakorisága is címkézi. A minimum-prioritásos sort a benne tárolt fák gyökerét címkéző gyakoriságértékek szerint építjük fel. Ezután a következőt csináljuk ciklusban, amíg a kupac még legalább kettő fából áll.

Kiveszünk a kupacból egy olyan fát, amelyeknek gyökerét a legkisebb gyakoriság címkézi. Ezután a maradék kupacra ezt még egyszer megismételjük. Összeadjuk a két gyakoriságot. Az összeggel címkézünk egy új csúcsot, amelynek bal és jobb részfája az előbb kiválasztott két fa lesz. A bal ágat a 0, a jobb ágat az 1 címkézi. Az így képzett új fát visszatesszük a minimum-prioritásos sorba.

A fenti ciklus után a minimum-prioritásos sorban maradó egyetlen bináris fa a Huffman-féle kódfa.

A kódfából ezután kódtáblázatot készítünk. Mindegyik karakterekhez tartozó kódot úgy kapjuk meg, hogy a kódfa gyökerétől elindulva és a karakterhez tartozó levélig lefelé haladva a kódfa éleit címkéző biteket összeolvassuk. (Ezt hatékonyan kivitelezhetjük pl. a kódfa preorder bejárásával, az aktuális csúcshoz vezető bitsorozat folyamatos nyilvántartásával, és levélhez érve, a kódtáblázatba írásával.)

Befejezésül újra végigolvassuk a tömörítendő szöveget, és a kódtáblázat segítségével sorban mindegyik karakter bináris kódját a (kezdetben üres) tömörített bitsorozat végéhez fűzzük. A tömörített fájl a kódfát is tartalmazza,

így a gyakorlatban Huffman-kódolással is csak hosszabb szövegeket érdemes tömöríteni.

A **kitömörítést** is karakterenként végezzük. Mindegyik karakter kinyeréséhez a kódfa gyökerétől indulunk, majd a tömörített kód sorban olvasott bitjeinek hatására 0 esetén balra, 1 esetén jobbra lépünk lefelé a fában, mígnem levélcsúcshoz érünk. Ekkor kiírjuk a levelet címkéző karaktert, majd a Huffman-kódban a következő bittől és újra a kódfa gyökerétől folytatjuk, amíg a tömörített kódon végig nem érünk.

2.2.1. Huffman-kódolás szemléltetése

Pl. az *ABRAKADABRA* szöveget egyszer végigolvasva meghatározhatjuk milyen karakterek fordulnak elő a szövegben, és milyen gyakorisággal. Úgy képzelhetjük, hogy az alábbi táblázat az új betűkkel folyamatosan bővül, ahogy haladunk előre a szövegben.

szöveg:	A	B	R	A	K	A	D	A	B	R	A
A	1			2		3		4			5
B	-	1							2		
D	-	-	-	-	-	-	1				
K	-	-	-	-	1						
R	-	-	1							2	

A fenti számolást (betű/gyakoriság) alakban összegezve:

$$\langle (D/\mathbf{1}), (K/\mathbf{1}), [B/\mathbf{2}], \{R/\mathbf{2}\}, (A/\mathbf{5}) \rangle$$

A fenti öt kifejezést öt egycsúcsú bináris fának tekinthetjük. (A jobb olvashatóság kedvéért többféle zárójelpárt alkalmaztunk.) Mindegyik csúcs egyben levél és gyökér. A levelekhez tartozó két címkét *karakter/gyakoriság* alakban írtuk le. A tömörítés algoritmusa szerint ezeket egy minimum-prioritásos sorba tesszük. A könnyebb érthetőség kedvéért ezt a minimum-prioritásos sort a szokásos minimum-kupacos reprezentáció helyett most a fák gyökerében lévő gyakoriság-értékek (röviden *fa-gyakoriság-értékek*) szerint rendezett fa-sorozattal szemléltetjük. (Azonos gyakoriságok esetén a betűk alfabetikus sorrendje szerint rendezünk. Ez ugyan önkényes, de az algoritmus bemutatása szempontjából hasznos egyértelműsítés. A fák ágait is hasonlóképpen rendezzük sorba.)

Ezután kivesszük a két legkisebb gyakoriság-értékű fát, egy új gyökércsúcs alá tesszük őket bal- és jobboldali részfának, a új gyökércsúcsot pedig a két

fa-gyakoriság-érték összegével címkézzük. Végül visszatesszük az új fát a minimum-prioritásos sorba.

$$\langle [B/\mathbf{2}], [(D/1)\mathbf{2}(K/1)], \{R/\mathbf{2}\}, (A/\mathbf{5})\rangle$$

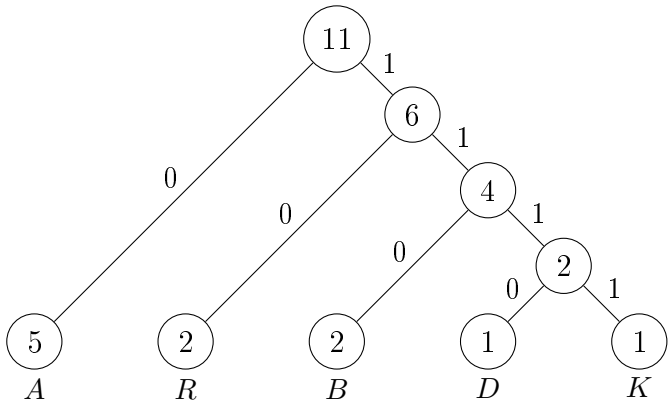
A fenti eljárást addig ismételjük, amíg már csak egy fánk marad. Ezt végül kivesszük a minimum-prioritásos sorból: ez a Huffman-féle kódfa.

$$\langle \{R/\mathbf{2}\}, \{[B/2]\mathbf{4}[(D/1)2(K/1)]\}, (A/\mathbf{5})\rangle$$

$$\langle (A/\mathbf{5}), (\{R/2\}\mathbf{6}\{[B/2]4[(D/1)2(K/1)]\})\rangle$$

$$[(A/5)\mathbf{11}(\{R/2\}6\{[B/2]4[(D/1)2(K/1)]\})]$$

A fent kapott kódfát az 1. ábrán is láthatjuk.



1. ábra. Az *ABRAKADABRA* szövegnek az alfabetikus konvencióval adódó Huffman-féle kódfája

Tekintsünk az 1. ábrán látható kódfában egy tetszőleges egyszerű, azaz körmentes utat, amely a fa gyökerétől lefelé valamelyik leveléig halad! Az út éleit címkéző biteket összeolvasva adódik a levelet címkéző karakter Huffman-kódja. Így a karaktarekre a következő kódtáblázatot kapjuk.

karakter	kód
A	0
B	110
D	1110
K	1111
R	10

A fentiek alapján az ABRAKADABRA szöveg Huffman kódja 23 bit, ami lényegesen rövidebb, mint a fenti naiv tömörítés esetén. A kódtáblázat bináris kódjait az ABRAKADABRA szöveg karakterei szerint sorban egymás után fűzve kapjuk a szöveg Huffman-kódját.

01101001111011100110100

A **kitömörítéshez** az előbbi Huffman-kód és a kódfa alapján a kezdő nulla rögtön az „A" címkéjű levélhez visz. Ezután sorban olvasva a maradékból a biteket, a 110 a B-hez visz, majd a 10 az R-hez, a 0 az A-hoz, a 1111 a K-hoz, a 0 az A-hoz, az 1110 a D-hez, a 0 az A-hoz, a 110 a B-hez, a 10 az R-hez, és végül a 0 az A-hoz. Így visszakaptuk az eredeti, tömörítetelen szöveget.

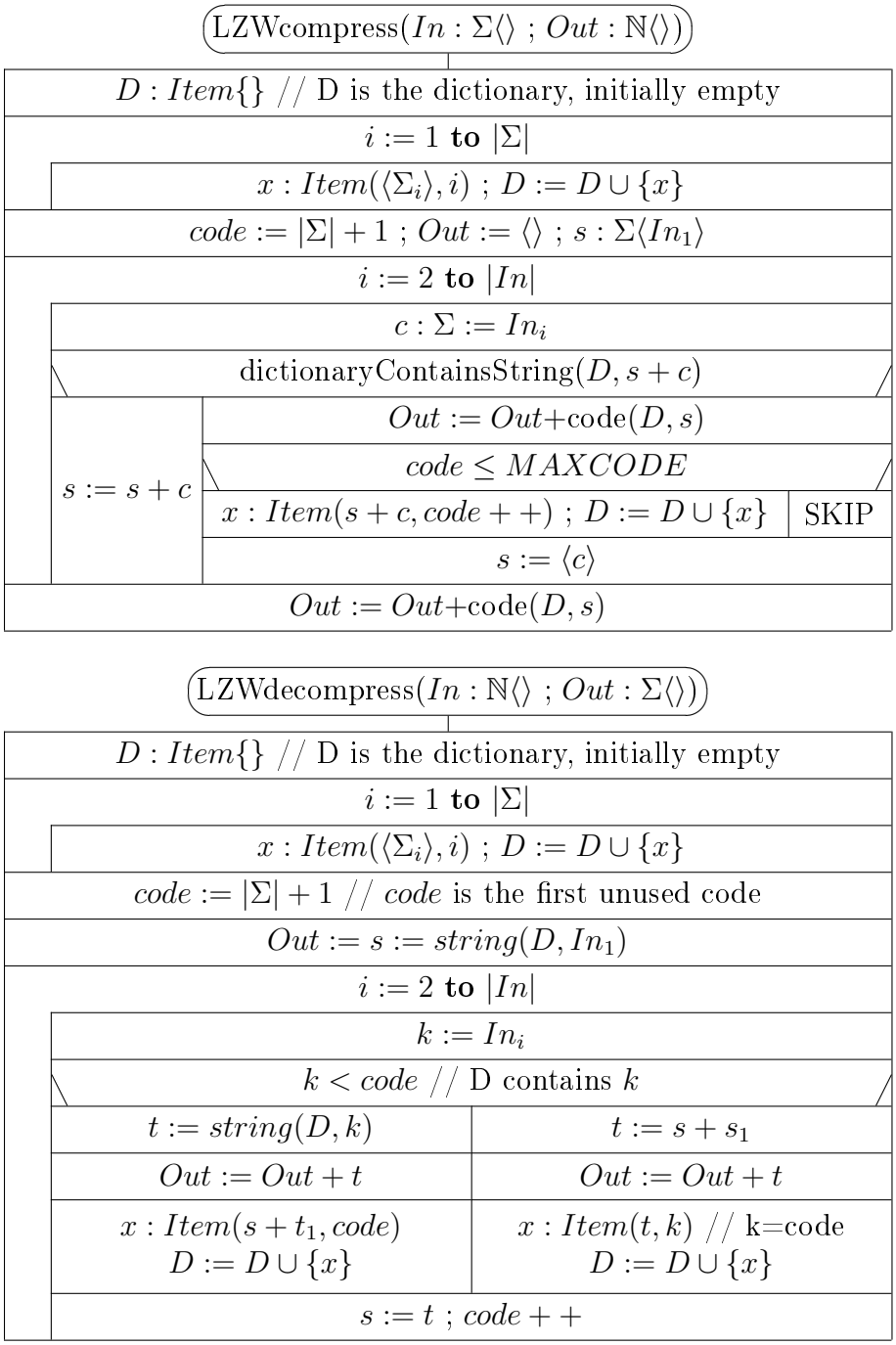
2.1. Feladat. *Próbáljuk ki, hogy ha a Huffman-kódolásban lévő indeterminizmusokat a fenti alfabetikus sorrendtől eltérően oldjuk fel, ugyanarra a tömörítendő szövegre mégis mindig ugyanolyan hosszú Huffman-kódot kapunk! (Ha például a minimum-prioritásos sorból azonos fa-gyakoriság-értekek esetén az alacsonyabb fát vesszük ki előbb – ezt az ad-hoc szabályt az alfabetikus konvenciónál erősebbnek véve –, akkor a fenti példában a kódfát felépítő ciklus második iterációjában a* $[B/\mathbf{2}]$ *és az* $\{R/\mathbf{2}\}$ *fát fogjuk összevonni.)*

2.3. Lempel–Ziv–Welch (LZW) módszer

Az input szöveget ismétlődő mintákra (sztringekre) bontja. Mindegyik mintát ugyanolyan hosszú bináris kóddal helyettesíti. Ezek a minták kódjai. A tömörített fájl a kódtáblázatot nem tartalmazza. Részletes magyarázat olvasható Ivanyos Gábor, Rónyai Lajos és Szabó Réka: *Algoritmusok* c. könyvében [4]. (Online elérhetősége az irodalomjegyzékünkben.)

Jelölések az absztrakt struktogramokhoz:
- Ha a kódok b bitesek, akkor $MAXCODE = 2^b - 1$ globális konstans a kódként használható legnagyobb számérték. Ha pl. $b = 12$, akkor $MAXCODE = 2^{12} - 1 = 4095$.
- A $\Sigma = \langle \sigma_1, \sigma_2, \ldots, \sigma_d \rangle$ sorozat tartalmazza az ábécé karaktereit.
- A tömörítésnél „*In*" a tömörítendő szöveg. „*Out*" a tömörítés eredménye: kódok sorozata. A kitömörítésnél fordítva.
- D a szótár, ami $(string, code)$ rendezett párok, azaz $Item$-ek halmaza. A szótárat a tömörített kód nem tartalmazza. Ehelyett a kitömörítés rekonstruálja az ábécé és a tömörített kód alapján.

Item
$+string : \Sigma\langle\rangle$ $+code : \mathbb{N}$
$+Item(s : \Sigma\langle\rangle \ ; \ k : \mathbb{N})\{string := s \ ; \ code := k\}$


LZWcompress($In : \Sigma\langle\rangle$; $Out : \mathbb{N}\langle\rangle$)
$D : Item\{\}$ // D is the dictionary, initially empty
$i := 1$ **to** $|\Sigma|$
$x : Item(\langle\Sigma_i\rangle, i)$; $D := D \cup \{x\}$
$code := |\Sigma| + 1$; $Out := \langle\rangle$; $s : \Sigma\langle In_1\rangle$
$i := 2$ **to** $|In|$
$c : \Sigma := In_i$
dictionaryContainsString($D, s + c$)
$s := s + c$
$Out := Out$+code(D, s)
$code \leq MAXCODE$
$x : Item(s + c, code + +)$; $D := D \cup \{x\}$
SKIP
$s := \langle c\rangle$
$Out := Out$+code(D, s)

LZWdecompress($In : \mathbb{N}\langle\rangle$; $Out : \Sigma\langle\rangle$)
$D : Item\{\}$ // D is the dictionary, initially empty
$i := 1$ **to** $|\Sigma|$
$x : Item(\langle\Sigma_i\rangle, i)$; $D := D \cup \{x\}$
$code := |\Sigma| + 1$ // $code$ is the first unused code
$Out := s := string(D, In_1)$
$i := 2$ **to** $|In|$
$k := In_i$
$k < code$ // D contains k
$t := string(D, k)$
$t := s + s_1$
$Out := Out + t$
$Out := Out + t$
$x : Item(s + t_1, code)$
$D := D \cup \{x\}$
$x : Item(t, k)$ // k=code
$D := D \cup \{x\}$
$s := t$; $code + +$