Mintaillesztés

Feladat: Egy szövegben egy minta összes előfordulását keressük.

Jelölések:

- szöveg: T [1..n]
- minta: P[1..m]
- Érvényes eltolások halmaza: S
- ábécé: Σ = {σ_1, σ_2, ..., σ_d}

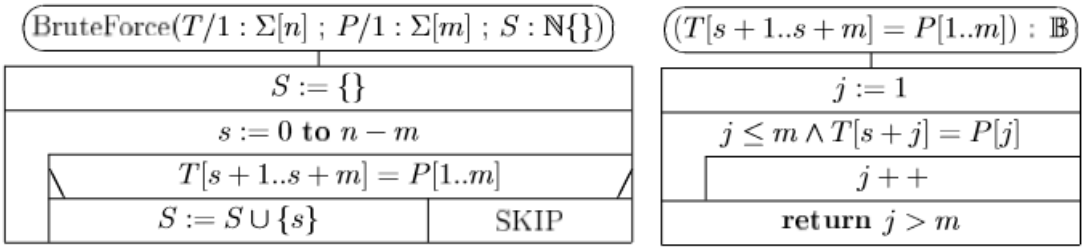
Mintaillesztő algoritmusok **összehasonlításánál** két fontos kérdéssel foglalkozunk:

- Műveletigény (hány összehasonlítást végzünk el)
- Mennyire „ugrál" a szövegben (Ez azért fontos, mert előfordulhat, hogy a mintaillesztést olyan adatszerkezetre kell megoldani (pl. szekvenciális fájl, bináris fa), ahol nem megengedett művelet az indexelés. Ekkor az olyan algoritmusoknál, ahol a szövegben ugrálunk, buffer segítségét kell igénybe venni.)

Brute-Force (BF) algoritmus:

A Brute-Force (nyers erő) algoritmus nem más, mint egy kiválogatásba ágyazott optimista lineáris keresés. Az algoritmust szemléletesen úgy lehet elképzelni, mintha a mintát tartalmazó sablont tolnánk végig a szövegen, és balról jobbra ellenőrizzük, hogy a minta karakterei egyeznek-e a lefedett szöveg karaktereivel. Amennyiben nem egyező karakterpárt találunk, a mintát eggyel jobbra toljuk a szövegen, és megint kezdjük a minta elejéről az összehasonlítást.

BF algoritmus stuktogramja



Műveletigény:

Legjobb eset: A minta első karaktere nincs a szövegben.
Ekkor $mÖ(n,m) = n - m + 1 \in \Theta(n)$

Legrosszabb eset: A minta minden eltolásnál csak a minta utolsó karakterénél romlik el az illeszkedés.
Ekkor $MÖ(n,m) = (n - m + 1) * m \in \Theta(n*m)$

Szekvenciális sorozatokra, fájlokra való alkalmazhatóság:

A szövegben „ugrálunk", ezért az olyan adatszerkezeteknél ahol nem megengedett az indexelés, szükség van buffer használatára.

Knuth-Morris-Pratt (KMP) algoritmus:

A KMP algoritmusnál nem szükséges minden esetben a minta elejéről kezdeni az illeszkedést. Amennyiben a mintával akkorát ugrunk, hogy a minta kezdőszelete (prefixe) egy valódi végszeletnél

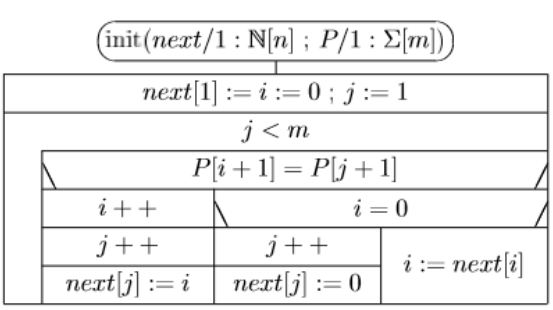
(szuffix) kezdődjön, azaz a prefix a szuffixel kerüljön fedésbe, a prefixet már nem kell újra vizsgálni. Ehhez a minta előfeldolgozására van szükség.

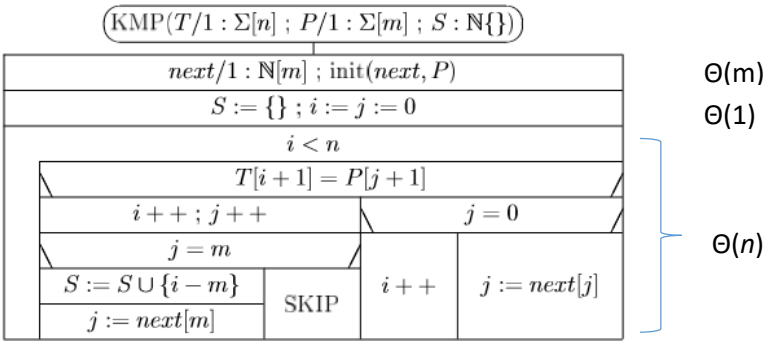
Az előfeldolgozás során definiálunk egy ***next* függvényt,** amely megadja a minta egyes kezdőrészleteire a leghosszabb egymással egyező prefix-szuffix párok hosszát. Ezt felhasználva tudjuk megadni a mintával való „ugrás” mértékét.

A next függvény alapvető tulajdonságai:

1. next(j) ∈ 0..(j-1) (j ∈ 1..m)
2. next(j+1) ≤ next(j)+1 (j ∈ 1..m-1)
3. $P_{h+1} \sqsupset T_{j+1} \Leftrightarrow P_h \sqsupset T_j \wedge P[h+1] = P[j+1]$
4. $0 \le h < j \le m$ és $P_j \sqsupset T_i$ esetén $P_h \sqsupset T_i \Longleftrightarrow P_h \sqsupset P_j$
5. $\max_{l+1} H(j) = next(\max_l H(j))$ (j ∈ 1..m, l ∈ 1..|$H(j)$|−1)

KMP algoritmus stuktogramja:





Műveletigény:

Az *init* műveletigénye **Θ(*m*)**. (Ahol *m* a minta hossza.)

Tegyük fel, hogy m<<n, ekkor a *KMP* műveletigénye legjobb és legrosszabb esetben is **Θ(*n*)**.

(T∈ Ω(n), mivel i növekedni egyesével tud, és n-ig nő ⟹ biztos van n lefutás

T ∈ O(n) 0 ≤ j ≤ i ≤ n, mivel a fő ciklus max 2*n*-szer fut le.)

KMP algoritmus előnye:

A szövegben nem kell visszaugrani. Ennek jelentősége pl szekvenciális sorozat/fájl formában adott szövegnél van, mivel ekkor buffer használata nélkül is tudjuk alkalmazni a KMP algoritmust.

1. példa: *P*[1..6]=*ABABAC*, *T*[1..9]=*ABABABACA*

Az init(*next,P*) algoritmus szemléltetése az *ABABAC* mintán:

i	*j*	*next*[*j*]	1 *A*	2 *B*	3 *A*	4 *B*	5 *A*	6 *C*
0	1	0		~~*A*~~				
0	2	0			A			
1	3	1			*A*	*B*		
2	4	2			*A*	*B*	*A*	
3	5	3			*A*	*B*	*A*	~~*B*~~
1	5	3					*A*	~~*B*~~
0	5	3						~~*A*~~
0	6	0						

A végeredmény:

P[*j*]=	*A*	*B*	*A*	*B*	*A*	*C*
j=	1	2	3	4	5	6
next[*j*]=	0	0	1	2	3	0

A ***P*[1..6] = *ABABAC*** mintát keressük a ***T*[1..9] = *ABABABACA*** szövegben.

i=	Hasonlítások száma	1	2	3	4	5	6	7	8	9	Magyarázat
T[*i*]=		*A*	*B*	*A*	*B*	*A*	*B*	*A*	*C*	*A*	
	6	*A*	*B*	*A*	*B*	*A*	~~*C*~~				Ekkor a szöveg és a minta 5. pozíciójánál vagyunk (i=j=5), és a szöveg 6. karakterét hasonlítjuk a minta 6. karakterével. Mivel nincs egyezés, a mintában az aktuális pozíciónkat (5) felülírjuk a next[5]-tel, azaz a szövegben az 5. pozíción maradunk, de a mintában a 3. –nál leszünk és a szöveg 6. pozícióját hasonlítjuk a mintánk 4. elemével.
s=2	3			*A*	*B*	*A*	*B*	*A*	*C*		A piros karaktereket nem hasonlítjuk, mivel a next[5]=3 azt jelenti, hogy a mintánk első három eleme megegyezik a 3.-5. elemével.
	1									*A*	

S={2}

2. példa: P[1..7]=BABABAB, T[1..9]=BABBABABABABBABABABAAB

Az init(*next,P*) algoritmus szemléltetése az *ABABAC* mintán:

i	*j*	*next*[*j*]	1 *B*	2 *A*	3 *B*	4 *A*	5 *B*	6 *A*	7 *B*
0	1	0		~~*B*~~					
0	2	0			B				
1	3	1			*B*	*A*			
2	4	2			*B*	*A*	*B*		
3	5	3			*B*	*A*	*B*	*A*	
4	6	4			*B*	*A*	*B*	*A*	*B*
5	7	5							

A végeredmény:

P[*j*]=	*B*	*A*	*B*	*A*	*B*	*A*	*B*
j=	1	2	3	4	5	6	7
next[*j*]=	0	0	1	2	3	4	5

A ***P*[1..7] = *BABABAB*** mintát keressük a ***T*[1..22] = *BABBABABAB ABBABABABAAB*** szövegben.

i=	H.SZ.	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
T[*i*]=		*B*	*A*	*B*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*B*	*A*	*B*	*A*	*B*	*A*	*B*	*A*	*A*	*B*
	4	*B*	*A*	*B*	~~*A*~~																		
	1			*B*	~~*A*~~																		
s=3	7				*B*	*A*	*B*	*A*	*B*	*A*	*B*												
s=5	2						*B*	*A*	*B*	*A*	*B*	*A*	*B*										
	1								*B*	*A*	*B*	*A*	*B*	~~*A*~~									
	1										*B*	*A*	*B*	~~*A*~~									
	1												*B*	~~*A*~~									
s=12	7													*B*	*A*	*B*	*A*	*B*	*A*	*B*			
	2															*B*	*A*	*B*	*A*	*B*	*A*	~~*B*~~	
	1																	*B*	*A*	*B*	*A*	~~*B*~~	
	1																			*B*	*A*	~~*B*~~	
	1																					~~*B*~~	
	1																						*B*

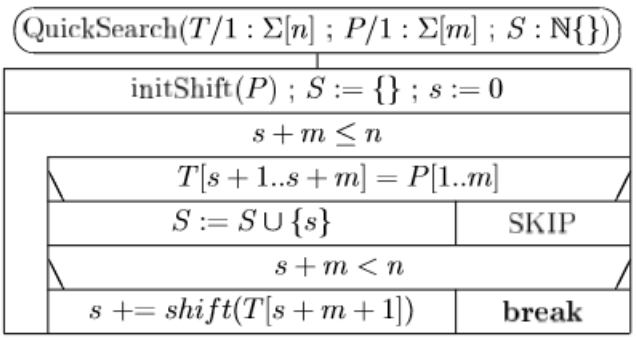
S={3; 5; 12}

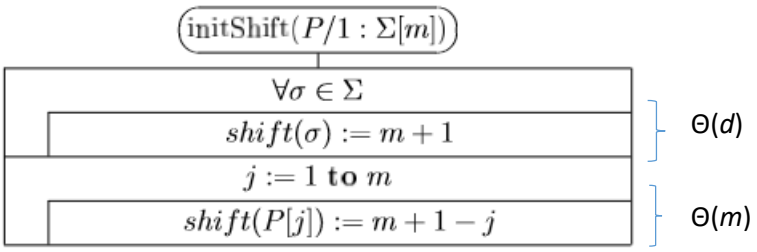
Quick Search (QS) algoritmus:

Alapötlet: Ha elromlik az illeszkedés, akkor nézzük a szövegben a minta utáni karaktert, és úgy toljuk el a mintát, hogy illeszkedjen a szöveg ezen karakteréhez. Ha a mintában nem szerepel ez a karakter, akkor átugorjuk a mintával. Az új vizsgálatot mindig a minta elejéről nézzük.

A mintával való „ugrás” végrehajtásához bevezetjük a shift függvényt. A shift függvény az ABC minden betűjére megadja az "ugrás" nagyságát, amelyet akkor tehetünk, ha az illeszkedés elromlása esetén az illető betű lenne a szöveg minta utáni első karaktere.

Quicksearch algoritmus stuktogramja:





Műveletigény:

Az initShift műveletigénye: *Θ(d)+Θ(m)* ∈ Θ*(m)* (*d:* az ábécé elemszáma, konstans)

legjobb eset: *mÖ(n,m)* ∈ *Θ(n/(m+1))* (A minta első karakterénél már elromlik az illeszkedés, továbbá a minta utáni karakter sem fordul elő a mintában, így azt „átugorjuk”.)

legrosszabb eset: *MÖ(n)* ∈ *Θ(n*m)* (A minta végén romlik el az illeszkedés, így kicsi az „ugrás”.)

Szekvenciális sorozatokra, fájlokra való alkalmazhatóság:

A szövegben „ugrálunk”, ezért az olyan adatszerkezeteknél ahol nem megengedett az indexelés, szükség van buffer használatára.

1. példa: *P*=ABCA, *T*=ABDAEBBCBBCABCBABCA

A szöveg alapján az ábécé: Σ={A, B ,C, D, E}

Előkészítő eljárás:

σ		*A* 5	*B* 5	*C* 5	*D* 5	*E* 5
A	1	4				
B	2		3			
C	3			2		
A	4	1				
SHIFT(σ)		1	3	2	5	5

Mintaillesztő eljárás:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
A	B	**D**	A	E	**B**	B	C	**B**	B	C	A	**B**	C	**B**	A	B	C	A
A	B	**C**	A															
					A	B	C	A										
								A	B	C	A							
											A	B	C	**A**				
												A	B	C	A			
															A	B	C	A

1. elromlás: D-C A minta utáni elem (5.) **E** shift(E)=5, ezért a mintát 5-tel toljuk el.

2. elromlás: B-A A minta utáni elem (10.) **B** shift(B)=3, ezért a mintát 3-mal toljuk el.

3. elromlás: B-A A minta utáni elem (13.) **B** shift(B)=3, ezért a mintát 3-mal toljuk el.

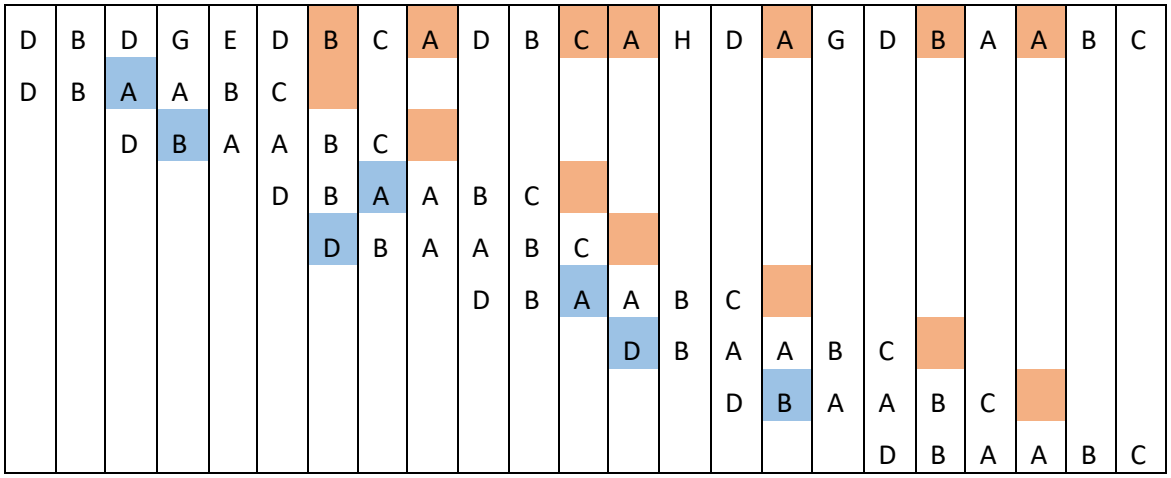
4. elromlás: B-A A minta utáni elem (16.) **A** shift(A)=1, ezért a mintát 1-gyel toljuk el.

5. elromlás: B-A A minta utáni elem (17.) **B** shift(B)=3, ezért a mintát 3-mal toljuk el.

2. PÉLDA: P= DBAABC, T=DBDGEDBCADBCAHDAGDBAABC

Σ={ A, B ,C, D,E ,G,H}

σ	A	B	C	D	E	G	H
SHIFT(σ)	3	2	1	6	7	7	7



eltolások:

shift(B)=2

shift(A)=3

shift(C)=1

shift(A)=3

shift(A)=3

shift(B)=2

shift(A)=3