Algoritmusok és adatszerkezetek II. előadásjegyzet:

Mintaillesztés, Tömörítés

Ásványi Tibor – asvanyi@inf.elte.hu

2020. augusztus 4.

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.,
magyarul: Új Algoritmusok, *Scolar Kiadó*, Budapest, 2003.
ISBN 963 9193 90 9
angolul: Introduction to Algorithms (Third Edititon),
The MIT Press, 2009.

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

[4] Koruhely Gábor, Szalay Richárd,
Algoritmusok és adatszerkezetek 2, 2015/16 tavaszi félév
(hallgatói jegyzet, lektorált és javított)
http://aszt.inf.elte.hu/~asvanyi/ad/

[5] 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

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

[7] Ásványi Tibor, Algoritmusok és adatszerkezetek I. előadásjegyzet (2019)
http://aszt.inf.elte.hu/~asvanyi/ad/ad1jegyzet.pdf

1. Mintaillesztés ([2] 32; [4])

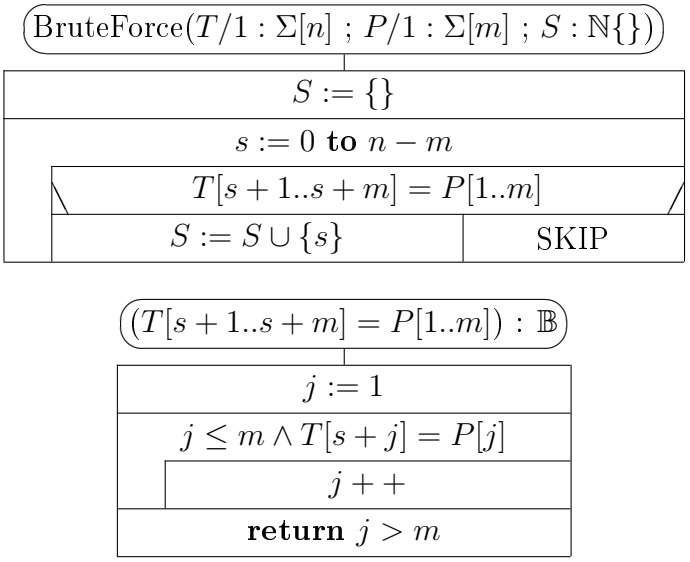
Adott a $\Sigma = \{\sigma_1, \sigma_2, \ldots, \sigma_d\}$ ábécé. $(1 \leq d < \infty$ konstans).

A $T/1 : \Sigma[n]$ szövegben keressük a $P/1 : \Sigma[m]$ minta előfordulásait $(1 \leq m \leq n)$. A fenti szimbólumokat az egész fejezetben így fogjuk használni.

1.1. Definíció. *$s \in 0..(n-m)$ pontosan akkor a P érvényes eltolása T-n, ha* $T[s+1..s+m] = P[1..m]$.

Az érvényes eltolások halmazát szeretnénk meghatározni, azaz az $S = \{\, s \in 0..(n-m) \mid T[s+1..s+m] = P[1..m] \,\}$ halmazt.

1.1. Egyszerű mintaillesztő (brute-force) algoritmus



1.2. Quicksearch

A gyorsabb keresés érdekében ennél és a következő (KMP) algoritmusnál általában egynél nagyobb lépésekben növeljük a $P[1..m]$ minta eltolását a $T[1..n]$ 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 $P[1..m]$ a keresett minta.

Tegyük fel most, hogy $\sigma = T[s+m+1]$. Ekkor a $shift(\sigma)$ érték megmondja, hogy a $T[s+1..s+m] = P[1..m]$ összehasonlítás után legalább mennyivel kell

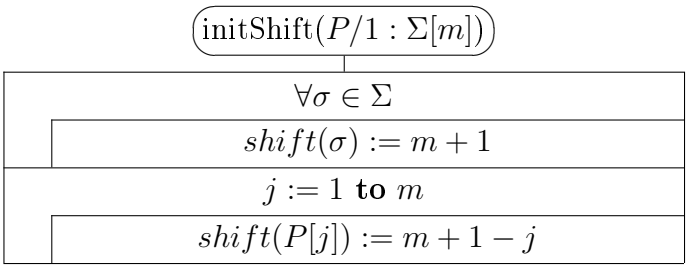
(jobbra) eltolni a P mintát a szövegen ahhoz, hogy a $T[s+m+1]$ alapján legyen esély a mintának a megfelelő szövegrészhez való illeszkedésére.
– $\sigma \in P[1..m]$ esetén a $shift(\sigma) \in 1..m$ érték azt mondja meg, hogy legalább mennyivel kell tovább tolni a P mintát ahhoz, hogy a $T[s+m+1]$ betűhöz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

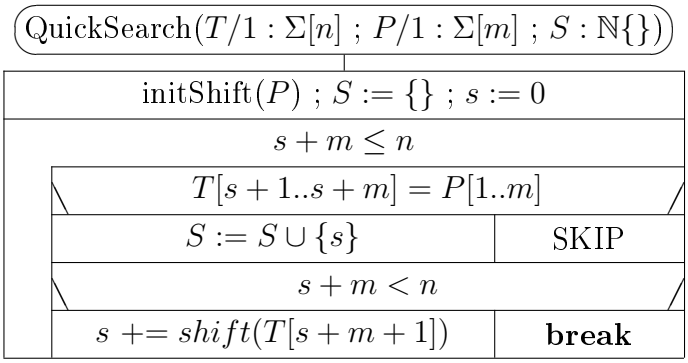
Arra az esetre, amikor az ábécé Σ = {A,B,C,D}, a minta pedig $P[1..4]$=CADA, az alábbi félig absztrakt példákban xxxx mutatja a CADA mintával az eltolás előtt összehasonlított szövegrészt, maga a CADA pedig a minta eltolás utáni helyzetét. (Ezután természetesen újabb összehasonlítás kezdődik a szöveg megfelelő része és a minta között stb.)

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

A megfelelő $shift$ értékeket a következő táblázat mutatja.

σ	A	B	C	D
$shift(\sigma)$	1	5	4	2





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

Tekintsük bevezetésként a következő példát! A $P[1..8] = BABABBAB$ mintát keressük a $T[1..18] = ABABABABBABABABBAB$ 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.)

$i=$	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
$T[i]=$	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

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

1.2. Jelölések.
– $P_j = P[1..j]$ *(csak ebben az alfejezetben)* P_j *a* P *sztring* j *hosszúságú prefixe, azaz kezdőszelete.* P_0 *az üres prefixe. Hasonlóan* $T_i = T[1..i]$.
– *Ha* x *és* y *két sztring, akkor* $x + y$ *a konkatenáltjuk.*
– *Ha* y *és* z *két sztring, akkor* $y \sqsupset z$ *(*y *a* z rövidebb szuffixe*) azt jelenti, hogy* $\exists x$ *nemüres sztring, amire* $x + y = z$.
(Eszerint minden nemüres sztringnek rövidebb szuffixe *az üres sztring, azaz* $P_0 \sqsupset P_j$ *ha* $j \in 1..m$*.)*
– $\max_i H$ *a* H *halmaz* i*-edik legnagyobb eleme* $\quad (i \in 1..|H|)$.
(Ezért $\max_1 H = \max H$*, és*
ha H *véges halmaz, akkor* $\max_{|H|} H = \min H$*.)*
– $H(j) = \{\, h \in 0..j-1 \mid P_h \sqsupset P_j \,\} \quad (j \in 1..m)$
(Így $0 \in H(j)$, $\max_1 H(j) = \max H(j)$, $\max_{|H(j)|} H(j) = \min H(j) = 0$*.)*
– $next(j) = \max H(j) \quad (j \in 1..m)$

1.3. Tulajdonság. $0 \le h < j \le m$ *és* $P_j \sqsupset T_i$ *esetén*
$P_h \sqsupset T_i \iff P_h \sqsupset P_j$.

1.4. Tulajdonság. $P_h \sqsupset T_i \wedge P[h+1] = T[i+1] \iff P_{h+1} \sqsupset T_{i+1}$.

1.5. Tulajdonság. $next(j) \in 0..(j-1) \quad (j \in 1..m)$

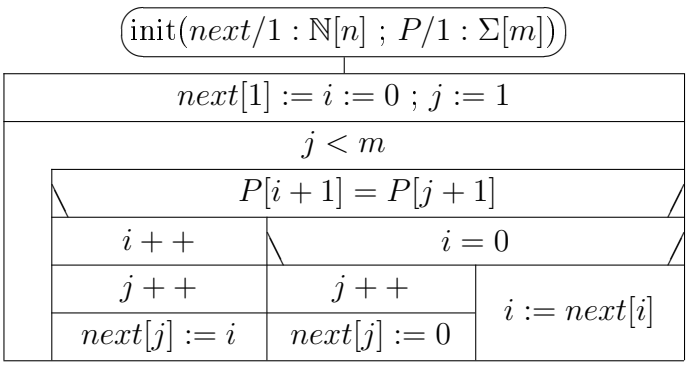
1.6. Tulajdonság. $next(j+1) \leq next(j)+1 \quad (j \in 1..m-1)$
(A $next(j)$ függvény legfeljebb egyesével növekszik.)

$P[j] =$	B	A	B	A	B	B	A	B
$j =$	1	2	3	4	5	6	7	8
$next(j) =$	0	0	1	2	3	1	2	3

1.7. Tulajdonság. $\max_{l+1} H(j) = next(\max_l H(j))$
$(j \in 1..m, l \in 1..|H(j)|-1)$

1.8. Definíció.
$next^1(j) = next(j) \quad (j \in 1..m)$
$next^{k+1}(j) = next(next^k(j)) \quad (next^k(j) \in 1..m)$
Szemléletesen: $next^k(j) = \underbrace{next(\ldots next(}_{k} j) \ldots)$

1.9. Tétel. $next^i(j) = \max_i H(j) \quad (i \in 1..|H(j)|)$

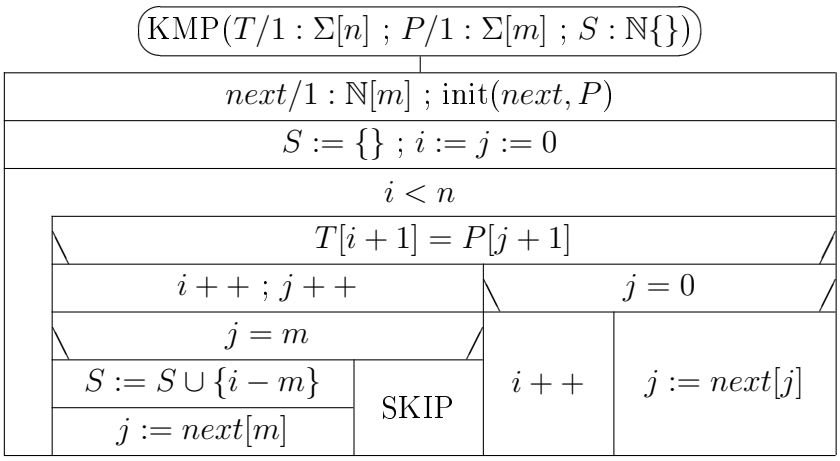


Az init($next, P$) algoritmus szemléltetése az $ABABBABA$ mintán:
(A három programág mindegyikének az elején kezdünk új sort.)

i	j	$next[j]$	$\overset{1}{A}$	$\overset{2}{B}$	$\overset{3}{A}$	$\overset{4}{B}$	$\overset{5}{B}$	$\overset{6}{A}$	$\overset{7}{B}$	$\overset{8}{A}$
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:

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



A $P[1..8] = ABABBABA$ mintát keressük
a $T[1..17] = ABABABBABABBABABA$ szövegben.

$i =$	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	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	$\underline{A}$

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

2. Információtömörítés ([5] 5; [4])

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.

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

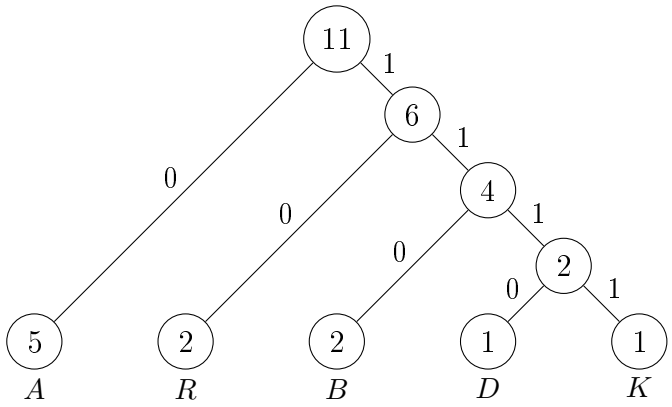
A fenti eljárást addig ismételjük, amíg már csak egy fánk marad. Ezt végül kivesszük a minimum-prioritásos sorból: ez a Huffman-féle kódfa.

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

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

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

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.

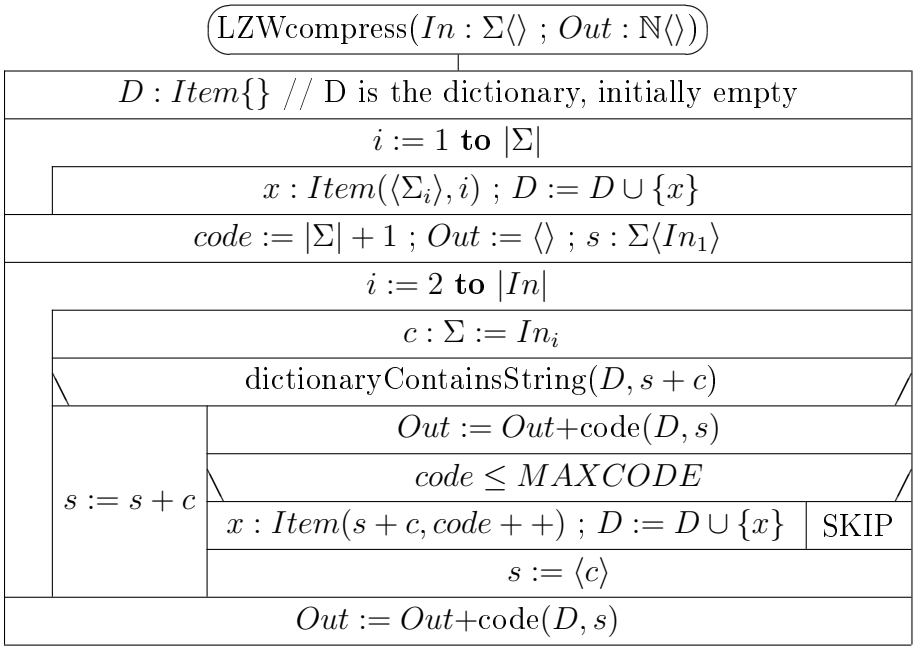
2.1. Feladat. *Próbáljuk ki, hogy ha a Huffman-kódolásban lévő indeterminizmusokat a fenti alfabetikus sorrendtől eltérően oldjuk fel, ugyanarra a tömörítendő szövegre mégis mindig ugyanolyan hosszú Huffman-kódot kapunk! (Ha például a minimum-prioritásos sorból azonos fa-gyakoriság-értekek esetén az 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.)*

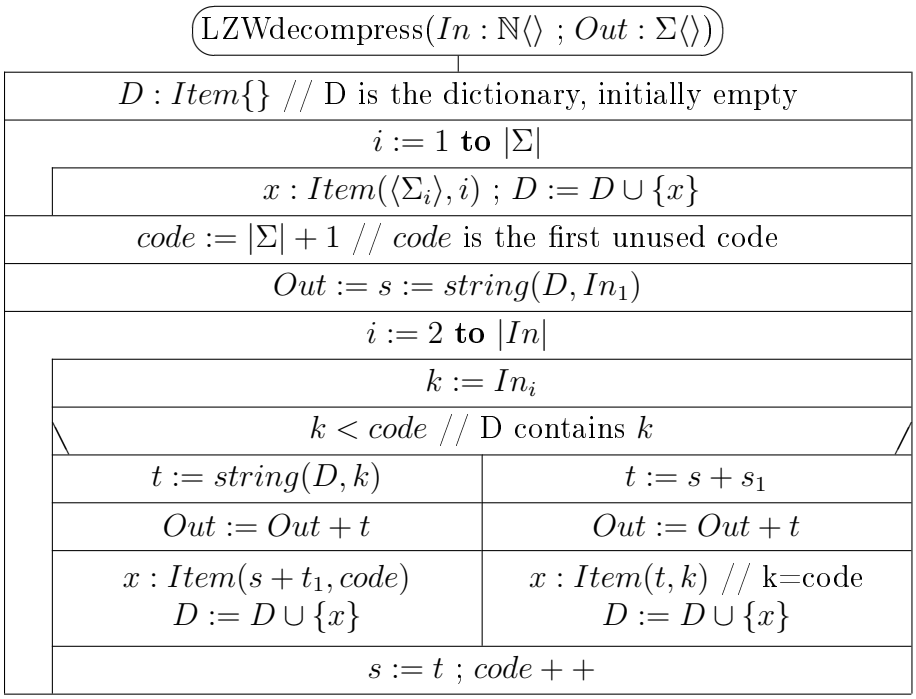
2.3. Lempel–Ziv–Welch (LZW) módszer

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 [5]. (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
$D := D \cup \{x\}$
$s := t$; $code + +$