8. Adattömörítés – nemparaméteres módszerek

Kódolástechnika

Adaptív Huffman kódok

A Shannon–Fano kód és a Huffman kód egyaránt használják a forrás eloszlását. De mi a helyzet, ha a forrás eloszlása nem ismert?

Adaptív Huffman kódok

A Shannon–Fano kód és a Huffman kód egyaránt használják a forrás eloszlását. De mi a helyzet, ha a forrás eloszlása nem ismert?

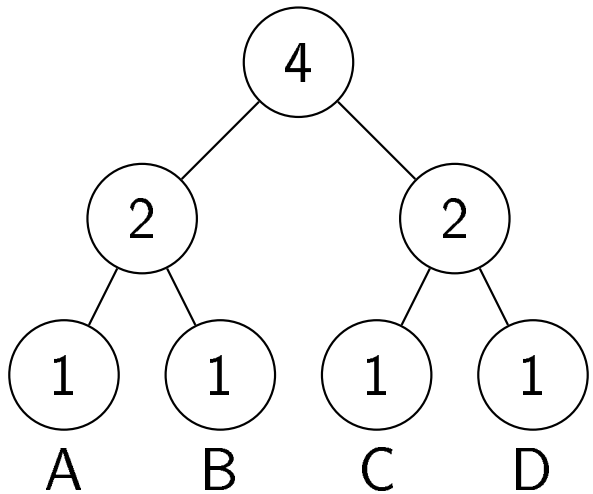
Adaptív Huffman kódolás: a forrás eloszlása helyett az egyes karakterek addigi előfordulási számai alapján építjük a fát. Kezdetben induljunk egy olyan fából, melyben az ábécé minden karaktere 1-es súlyt kap.

Adaptív Huffman kódok

A Shannon–Fano kód és a Huffman kód egyaránt használják a forrás eloszlását. De mi a helyzet, ha a forrás eloszlása nem ismert?

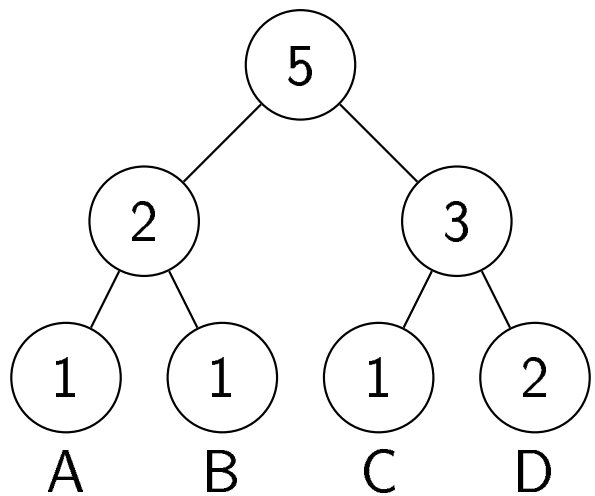
Adaptív Huffman kódolás: a forrás eloszlása helyett az egyes karakterek addigi előfordulási számai alapján építjük a fát. Kezdetben induljunk egy olyan fából, melyben az ábécé minden karaktere 1-es súlyt kap.

Példa. Ha az ábécé {A,B,C,D}, akkor a kezdeti Huffman fa:



Adaptív Huffman kódok

Ezután az előfordulási számokat annak megfelelően növeljük, ahogy a karakterek a szövegben érkeznek. Például ha az első karakter D, akkor 1-gyel megnöveljük a D levélben található számot (és a megfelelő belső csúcsokat is):



És így tovább.

Adaptív Huffman kódok – a testvérpár tulajdonság

A testvér csúcsokat (akiknek közös a szülője) mindig balról jobbra növő módon rendezzük.

Adaptív Huffman kódok – a testvérpár tulajdonság

A testvér csúcsokat (akiknek közös a szülője) mindig balról jobbra növő módon rendezzük.

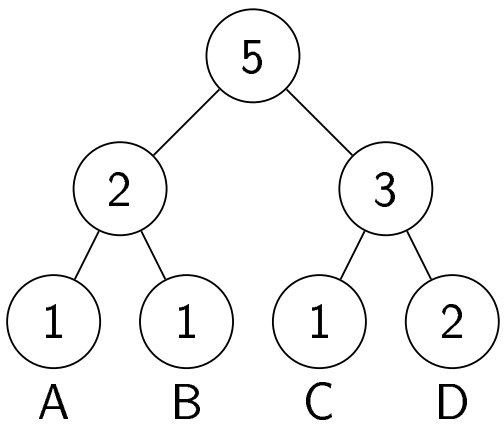
Ezen kívül még egy fontos tulajdonságnak teljesülnie kell a Huffman fára: ha kiolvassuk a számokat az alsó sorban balról jobbra, majd a felette lévő sorban balról jobbra és így tovább, a kapott számsorozatnak nemcsökkenőnek kell lennie. Ennek a neve *testvérpár tulajdonság*, az ilyen tulajdonságú fákat pedig *szabályos Huffman fának* hívjuk.

Adaptív Huffman kódok – a testvérpár tulajdonság

A testvér csúcsokat (akiknek közös a szülője) mindig balról jobbra növő módon rendezzük.

Ezen kívül még egy fontos tulajdonságnak teljesülnie kell a Huffman fára: ha kiolvassuk a számokat az alsó sorban balról jobbra, majd a felette lévő sorban balról jobbra és így tovább, a kapott számsorozatnak nemcsökkenőnek kell lennie. Ennek a neve *testvérpár tulajdonság*, az ilyen tulajdonságú fákat pedig *szabályos Huffman fának* hívjuk.

Például



esetén a sorozat 1, 1, 1, 2, 2, 3, 5, tehát ez szabályos Huffman fa.

Tömörítés

A tömörítési eljárás:

1. Inicializáljuk a Huffman fát.
2. Beolvassuk a szöveg következő karakterét és kódoljuk a Huffman fa aktuális állapota szerint.
3. Hozzáadjuk a fához a karaktert.
4. Ellenőrizzük a testvérpár tulajdonságot, és ha sérül, helyreállítjuk a fa megfelelő módosításával (mindjárt megnézzük, hogyan).
5. Előrelépünk a következő karakterre és a 2. lépéstől ismételjük az eljárást.

Huffman-fa helyreállítása

Ha sérül a testvérpár-tulajdonság, akkor a helyreállító lépés a következő:

- vegyük a legutolsónak hozzáadott karakternek megfelelő levelet (itt fog sérülni), és
- vegyük a tőle a fában legtávolabb lévő (leghosszabb úton elérhető), 1-gyel kisebb súlyú csúcsot.

Huffman-fa helyreállítása

Ha sérül a testvérpár-tulajdonság, akkor a helyreállító lépés a következő:

- vegyük a legutolsónak hozzáadott karakternek megfelelő levelet (itt fog sérülni), és
- vegyük a tőle a fában legtávolabb lévő (leghosszabb úton elérhető), 1-gyel kisebb súlyú csúcsot.

Ezt a két csúcsot cseréljük meg teljes részfákkal együtt. A felcserélt csúcsok szüleinek súlyát is megfelelően módosítjuk.

Huffman-fa helyreállítása

Ha sérül a testvérpár-tulajdonság, akkor a helyreállító lépés a következő:

- vegyük a legutolsónak hozzáadott karakternek megfelelő levelet (itt fog sérülni), és
- vegyük a tőle a fában legtávolabb lévő (leghosszabb úton elérhető), 1-gyel kisebb súlyú csúcsot.

Ezt a két csúcsot cseréljük meg teljes részfákkal együtt. A felcserélt csúcsok szüleinek súlyát is megfelelően módosítjuk.

Ezen kívül a testvér csúcsokat (akiknek közös a szülője) mindig balról jobbra növő módon rendezzük; ha valahol fordítva állnának, megcseréljük a két testvért.

Kicsomagolás

A kicsomagoló eljárás:

1. Inicializáljuk a Huffman fát.
2. Dekódolunk 1 karaktert a Huffman fa aktuális állapota szerint.
3. Hozzáadjuk a fához a karaktert.
4. Ellenőrizzük a testvérpár tulajdonságot, és ha sérül, helyreállítjuk a fa megfelelő módosításával.
5. Előrelépünk a következő karakterre és a 2. lépéstől ismételjük az eljárást.

Kicsomagolás

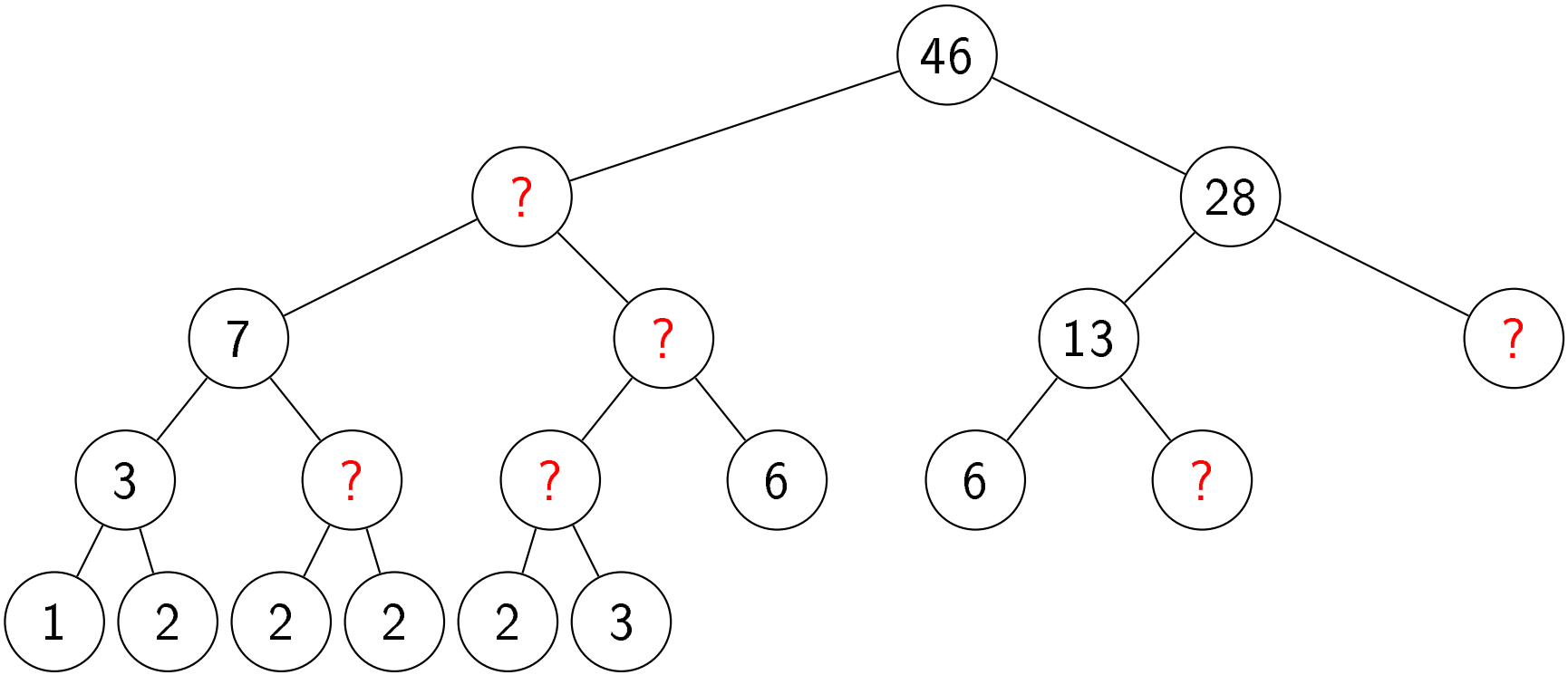
A kicsomagoló eljárás:

1. Inicializáljuk a Huffman fát.
2. Dekódolunk 1 karaktert a Huffman fa aktuális állapota szerint.
3. Hozzáadjuk a fához a karaktert.
4. Ellenőrizzük a testvérpár tulajdonságot, és ha sérül, helyreállítjuk a fa megfelelő módosításával.
5. Előrelépünk a következő karakterre és a 2. lépéstől ismételjük az eljárást.

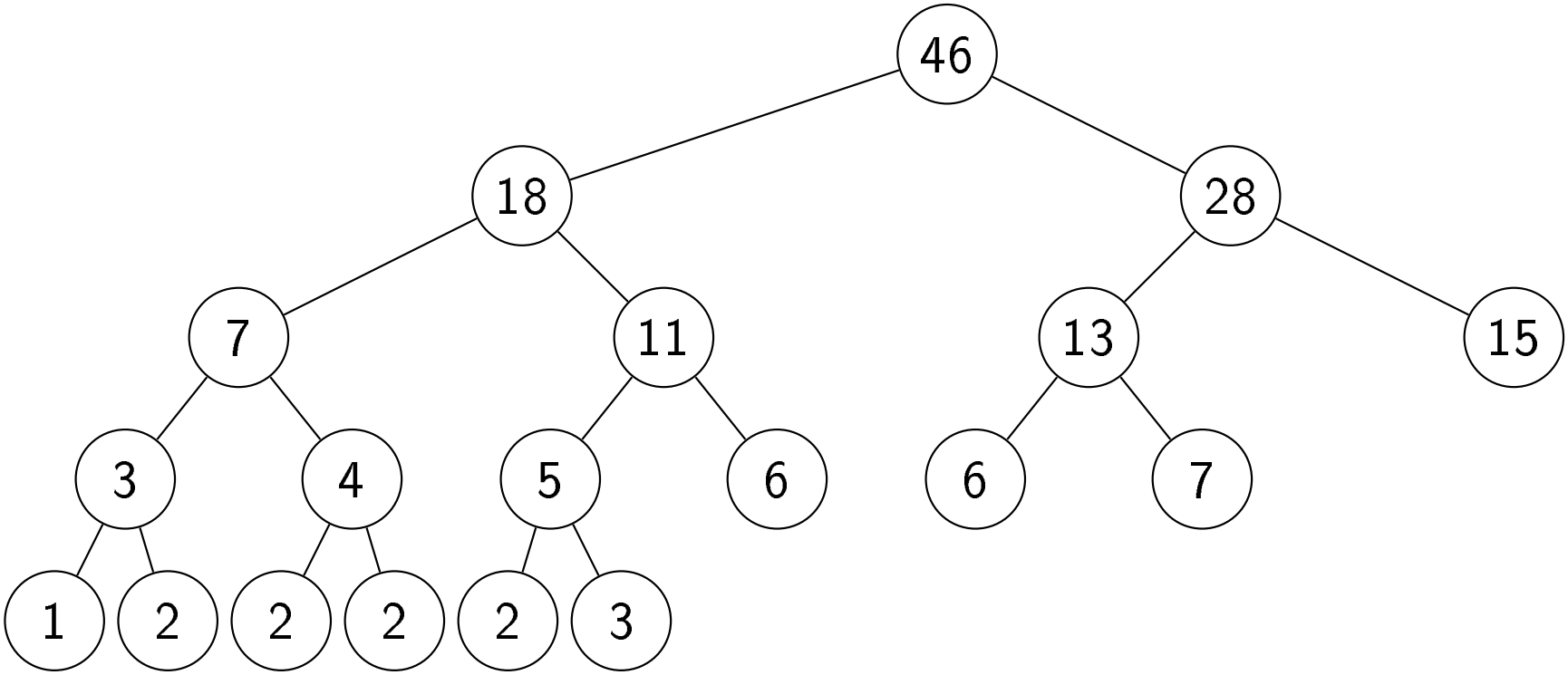
(Az 1. lépést kötelező pontosan ugyanúgy megvalósítani a tömörítés és a kicsomagolás során. A 4. lépést szintén.)

1. feladat

Határozzuk meg a hiányzó értékeket. Ez a gráf egy szabályos Huffman-fa?



1. feladat



A bal alsó sarokból indulva, balról jobbra, majd soronként egyesével felfelé haladva kiolvassuk a súlyokat: az 1, 2, 2, 2, 2, 3, 3, 4, 5, 6, 6, 7, 7, 11, 13, 15, 18, 28, 46 sorozat növekvő, így ez egy szabályos Huffman-fa.

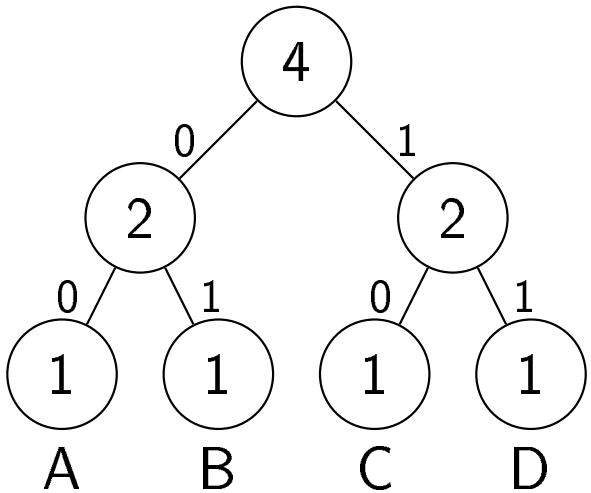
2. feladat

Az {A,B,C,D} ábécé felett adaptív Huffman-kódot készítünk a DCDADD sorozathoz. Ábrázoljuk a Huffman-fa és a kód alakulását.

2. feladat

Az {A,B,C,D} ábécé felett adaptív Huffman-kódot készítünk a DCDADD sorozathoz. Ábrázoljuk a Huffman-fa és a kód alakulását.

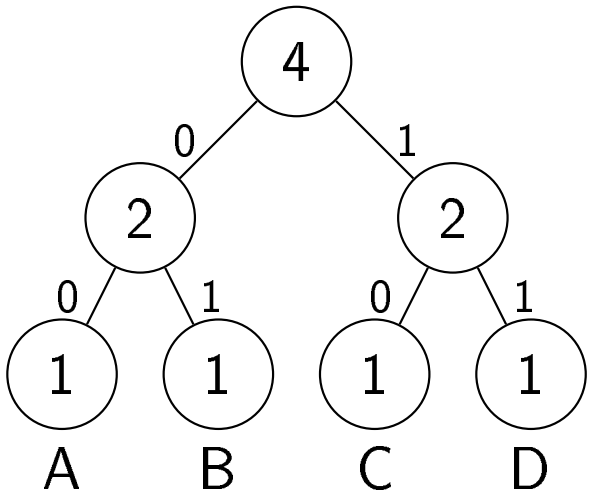
Megoldás.



2. feladat

Az {A,B,C,D} ábécé felett adaptív Huffman-kódot készítünk a DCDADD sorozathoz. Ábrázoljuk a Huffman-fa és a kód alakulását.

Megoldás.

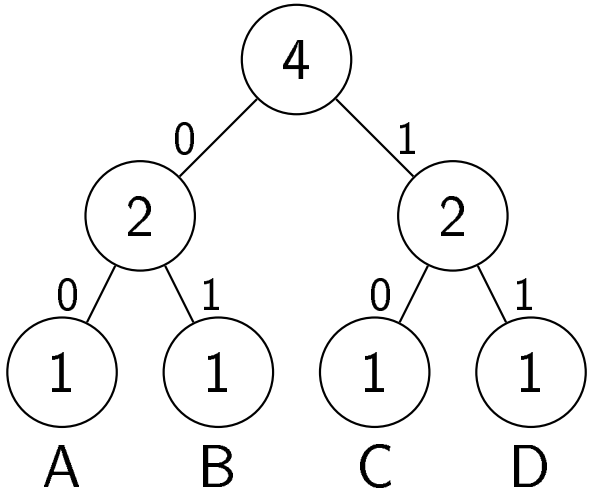


D $\rightarrow$ 11

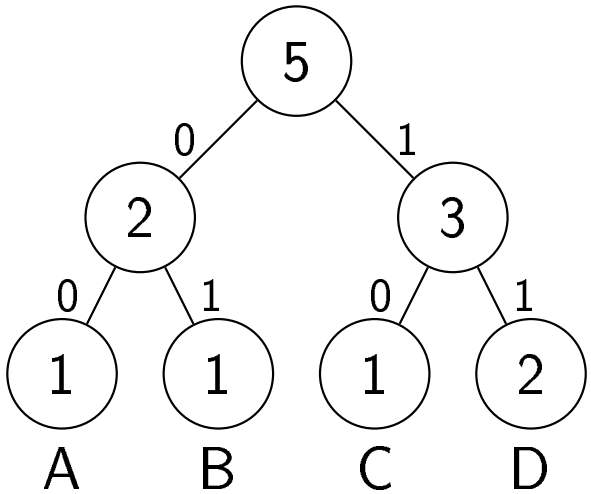
2. feladat

Az {A,B,C,D} ábécé felett adaptív Huffman-kódot készítünk a DCDADD sorozathoz. Ábrázoljuk a Huffman-fa és a kód alakulását.

Megoldás.



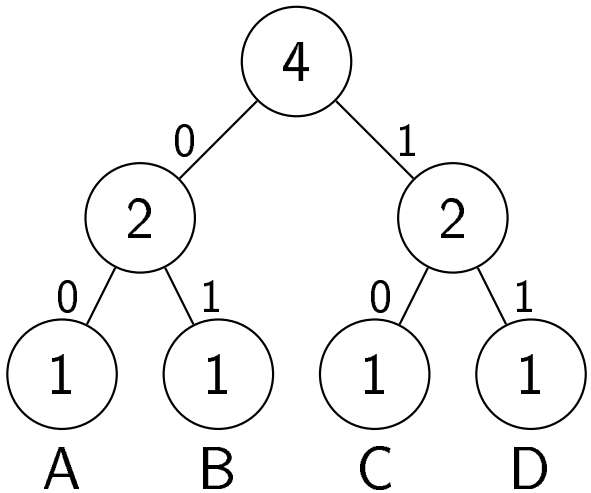
D $\rightarrow$ 11



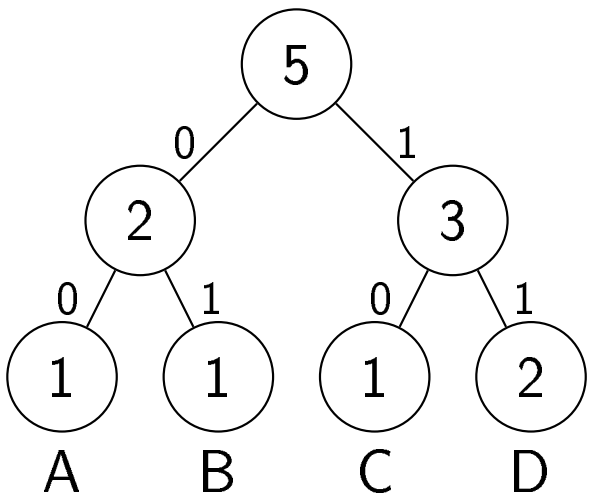
2. feladat

Az {A,B,C,D} ábécé felett adaptív Huffman-kódot készítünk a DCDADD sorozathoz. Ábrázoljuk a Huffman-fa és a kód alakulását.

Megoldás.

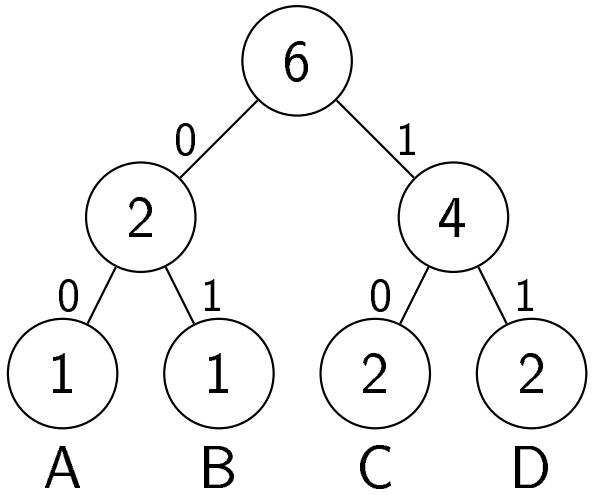


D → 11

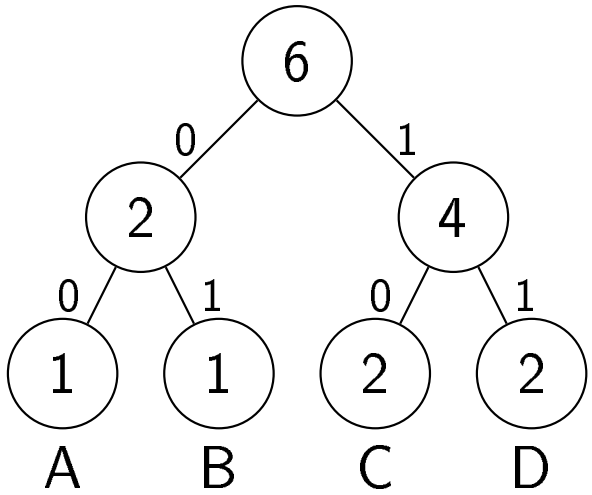


DC → 1110

2. feladat

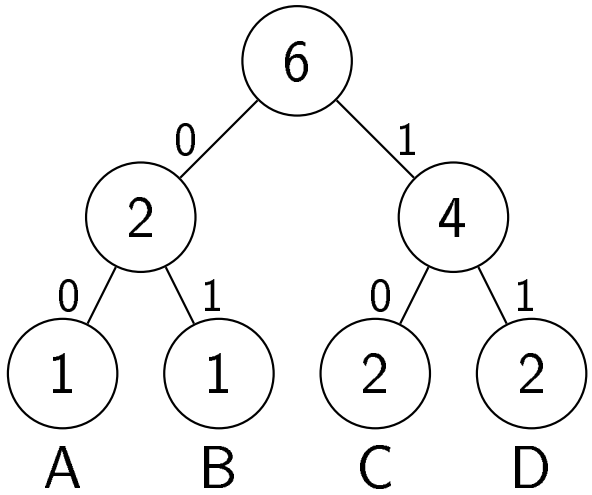


2. feladat

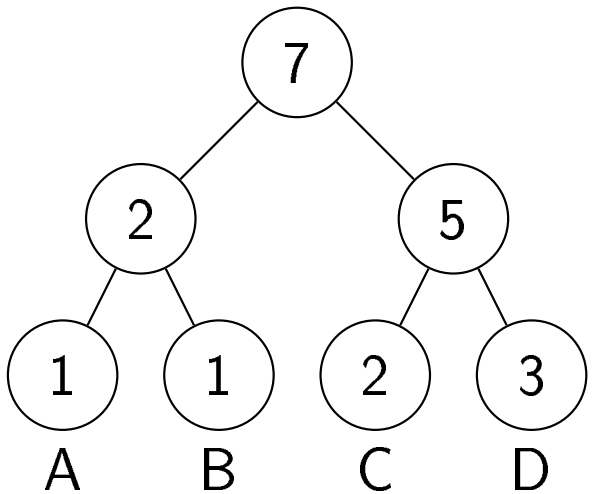


DCD $\rightarrow$ 111011

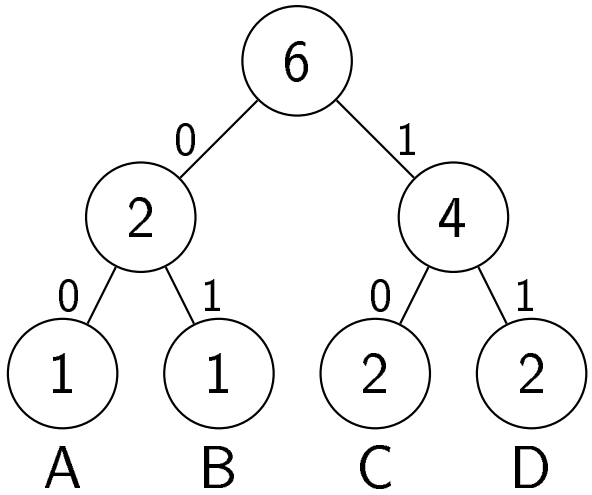
2. feladat



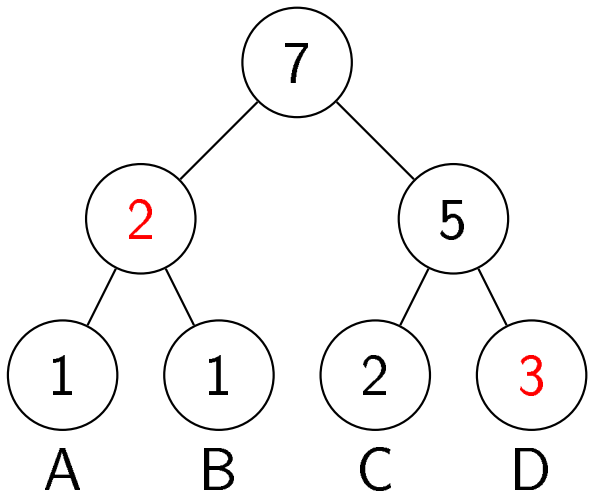
DCD $\rightarrow$ 111011



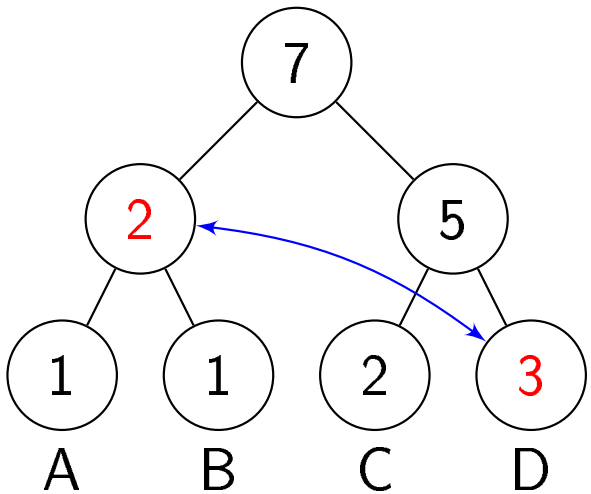
2. feladat



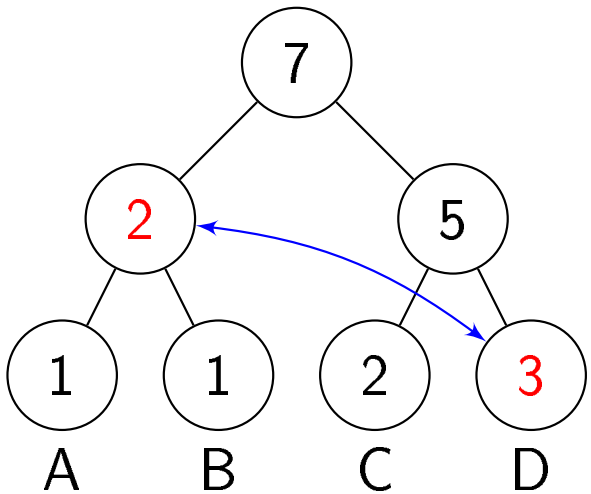
DCD $\rightarrow$ 111011

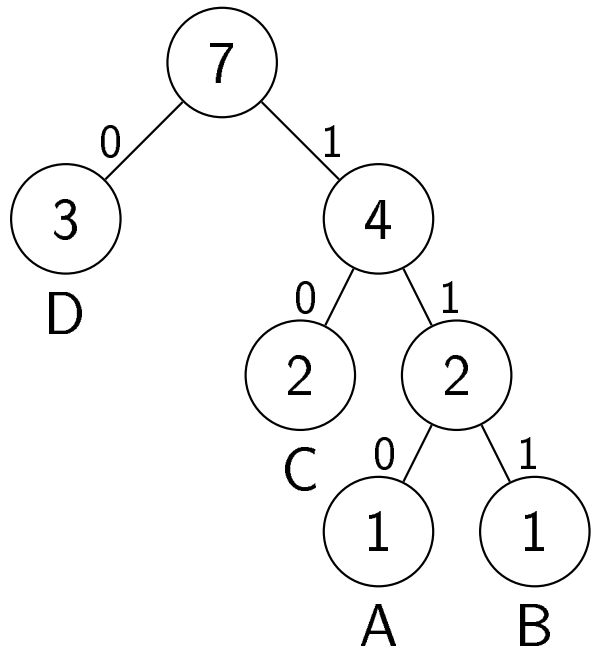


2. feladat

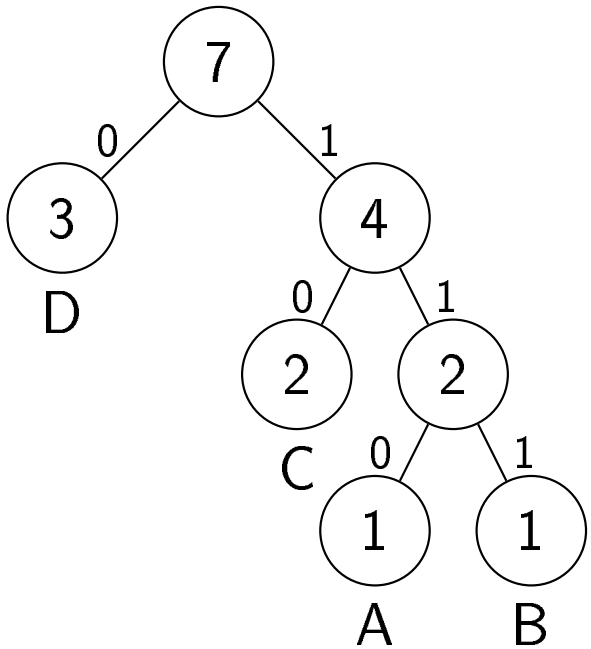


2. feladat

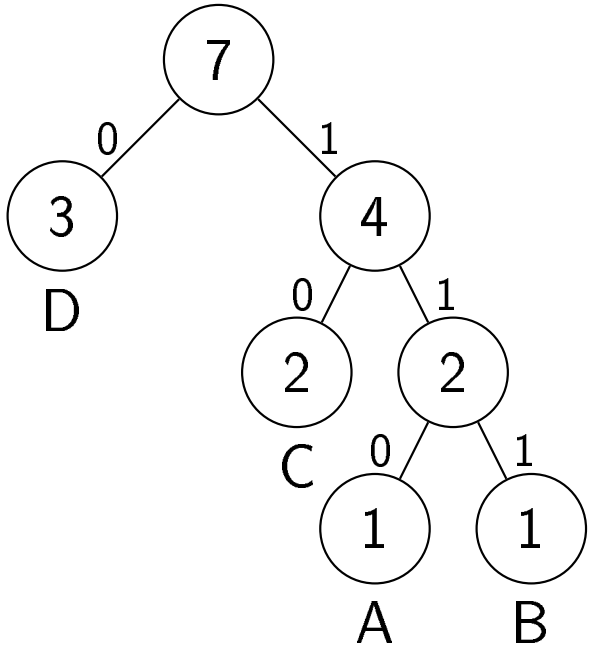




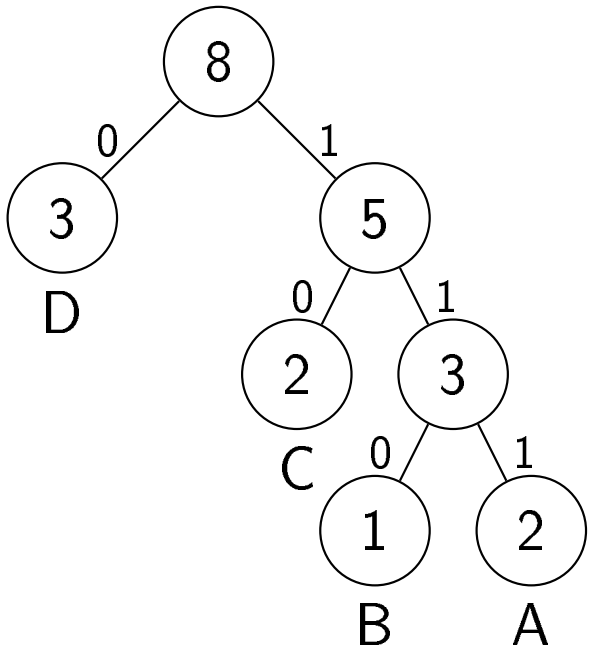
2. feladat



2. feladat

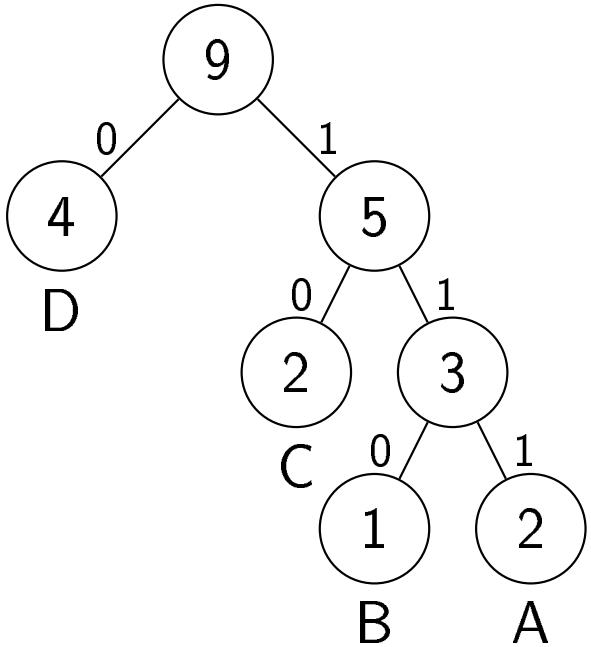


DCDA → 111011110



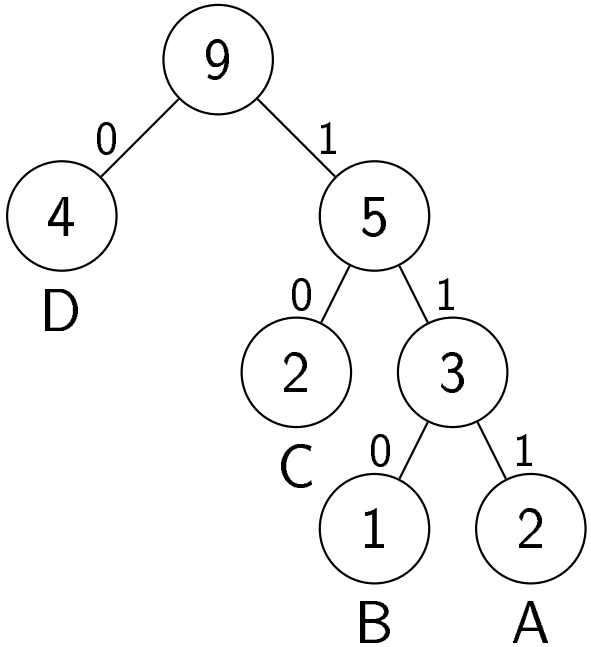
DCDAD → 1110111100

2. feladat

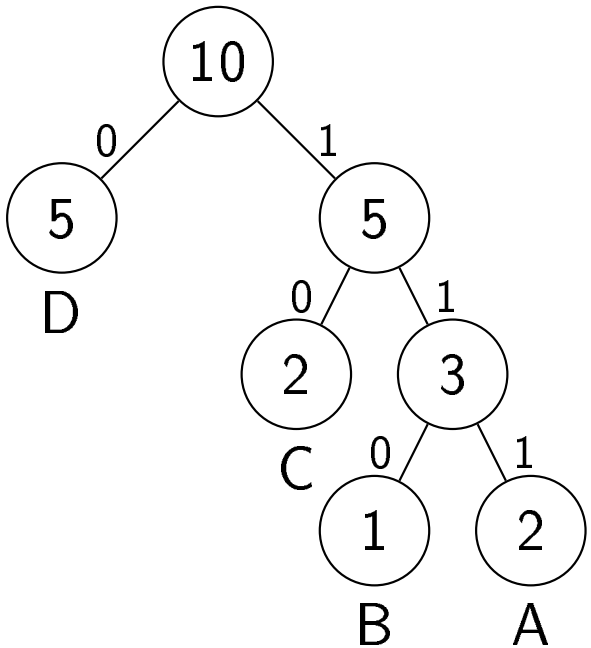


DCDADD $\rightarrow$ 11101111000

2. feladat



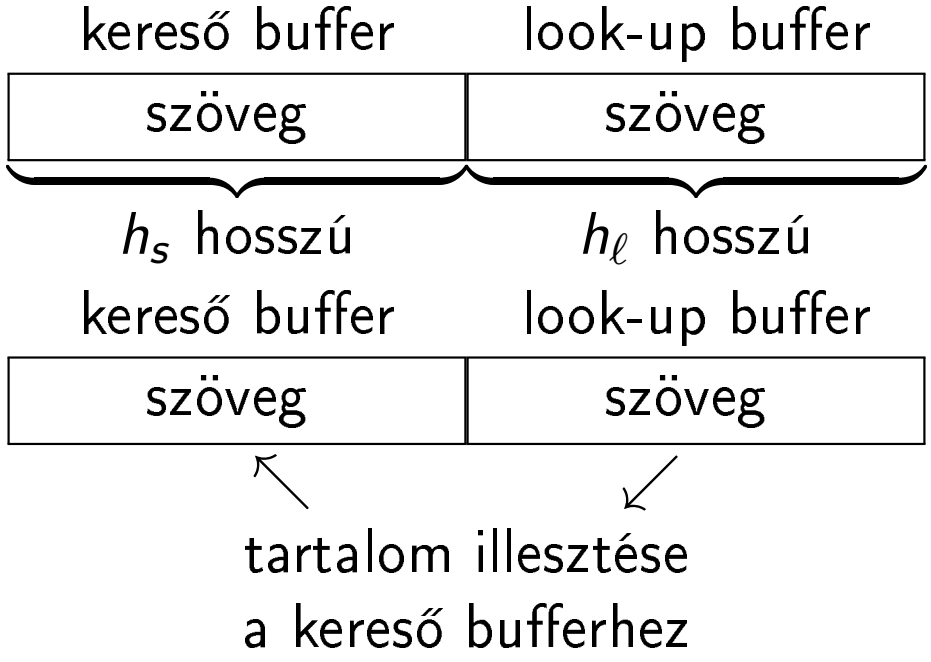
DCDADD$\rightarrow$11101111000



DCDADD$\rightarrow$11101111000

LZ77 algoritmus

A fő ötlet: előre tekintve olyan szakaszokat keresünk a szövegben, amelyeket a közelmúltban már láttunk.



Kimenet: (p, ℓ, n), ahol:

- p: az egyező szakasz elejének pozíciója (a kurzorhoz képest visszafelé)
- ℓ: az egyező szakasz hossza
- n: a következő karakter (kódja)

A kimenet teljes mérete: $\lceil \log_2 h_s \rceil + \lceil \log_2 h_\ell \rceil + \lceil \log_2 \chi \rceil$ bit.

3. feladat

LZ77 algoritmussal tömörítjük a “c a b r a c a d a b r a r r a r r a d. . .” szöveget; a paraméterek értéke $h_s = 7, h_\ell = 6$. A kurzor kezdetben a 7-es pozícióban áll. Adjuk még az algoritmus kimenetét a kezdeti állapotra és a következő két lépésre.

3. feladat

LZ77 algoritmussal tömörítjük a “c a b r a c a d a b r a r r a r r a d. . .” szöveget; a paraméterek értéke $h_s = 7, h_\ell = 6$. A kurzor kezdetben a 7-es pozícióban áll. Adjuk még az algoritmus kimenetét a kezdeti állapotra és a következő két lépésre.

Megoldás.

- kezdeti állapot:

c a b r a c a	d a b r a r	r a r r a d. . .

kimenet: $(0, 0, d)$

3. feladat

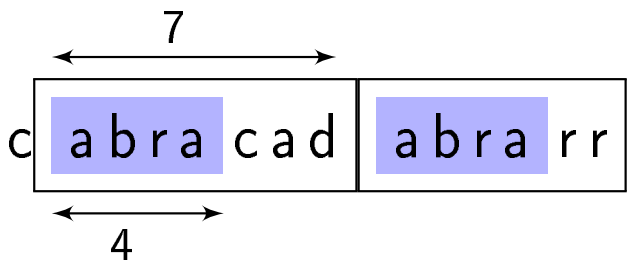
LZ77 algoritmussal tömörítjük a "c a b r a c a d a b r a r r a r r a d. . ." szöveget; a paraméterek értéke $h_s = 7, h_\ell = 6$. A kurzor kezdetben a 7-es pozícióban áll. Adjuk még az algoritmus kimenetét a kezdeti állapotra és a következő két lépésre.

Megoldás.

- kezdeti állapot:

  c a b r a c a | d a b r a r | r a r r a d. . . kimenet: $(0, 0, d)$

- következő lépés:



kimenet: $(7, 4, r)$

3. feladat

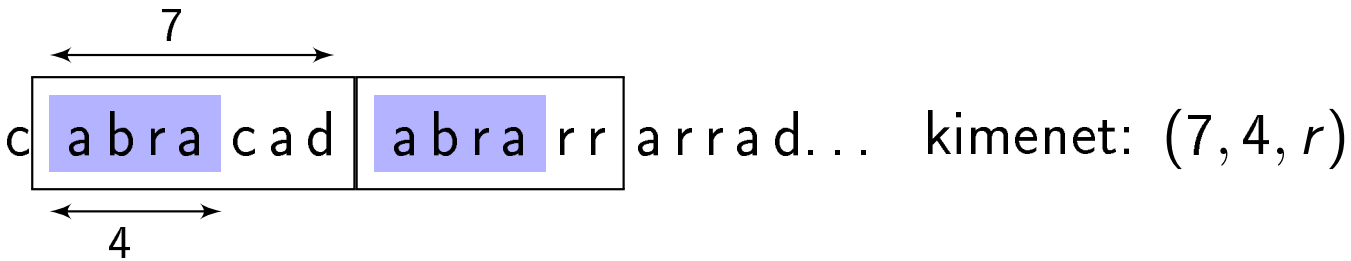
LZ77 algoritmussal tömörítjük a "c a b r a c a d a b r a r r a r r a d. . ." szöveget; a paraméterek értéke $h_s = 7, h_\ell = 6$. A kurzor kezdetben a 7-es pozícióban áll. Adjuk még az algoritmus kimenetét a kezdeti állapotra és a következő két lépésre.

Megoldás.

- kezdeti állapot:

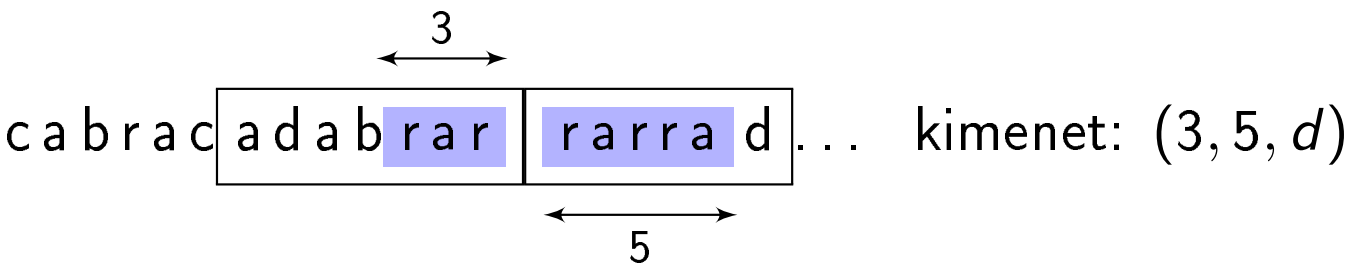
  c a b r a c a | d a b r a r r a r r a d. . . kimenet: $(0, 0, d)$

- következő lépés:



kimenet: $(7, 4, r)$

- következő lépés:



kimenet: $(3, 5, d)$

4. feladat

LZ77 algoritmussal tömörítünk egy 32 karakteres ábécében írt szöveget. A kereső buffer hossza 32, a look-up buffer hossza 16. Hány bit lesz a kimenet mérete az algoritmus egy lépése során?

4. feladat

LZ77 algoritmussal tömörítünk egy 32 karakteres ábécében írt szöveget. A kereső buffer hossza 32, a look-up buffer hossza 16. Hány bit lesz a kimenet mérete az algoritmus egy lépése során?

Megoldás.

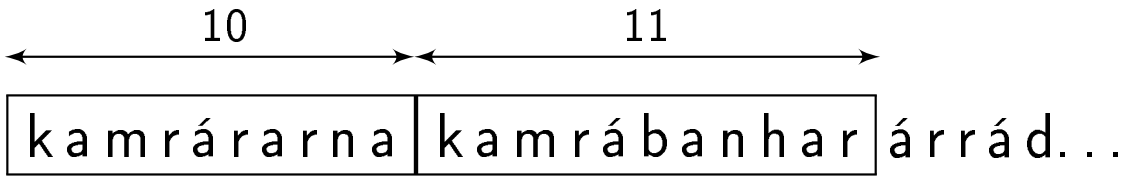
$$\chi = 32, \quad h_s = 32, \quad h_\ell = 16,$$

így

$$\lceil \log_2 h_s \rceil + \lceil \log_2 h_\ell \rceil + \lceil \log_2 \chi \rceil = 5 + 4 + 5 \text{ bit.}$$

5. feladat

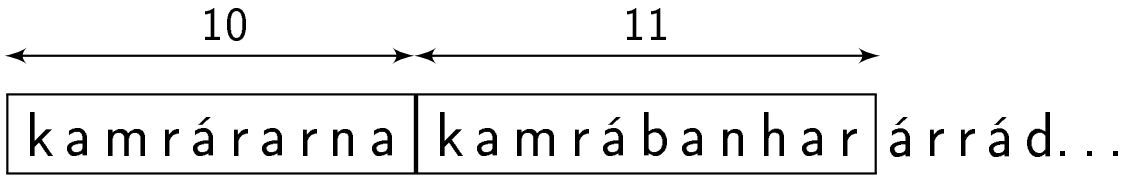
LZ77 algoritmussal tömörítés során az eltolás regiszter állapota



Adjuk meg a következő lépés kimenetét.

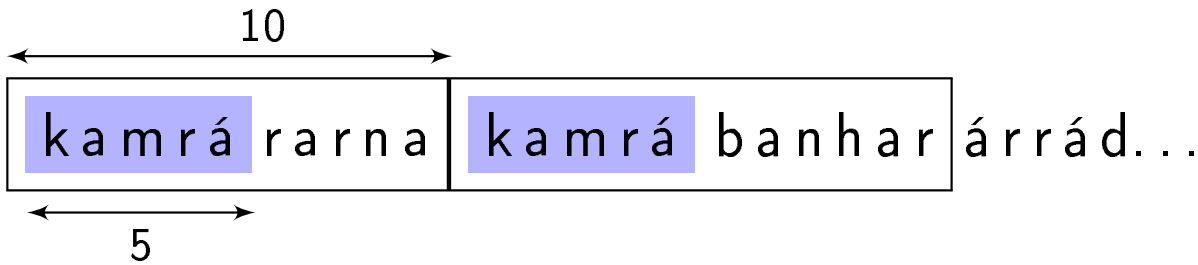
5. feladat

LZ77 algoritmussal tömörítés során az eltolás regiszter állapota



Adjuk meg a következő lépés kimenetét.

Megoldás.



kimenet: $(10, 5, b)$

LZ78 algoritmus

Fő ötlet: A szöveget olyan szakaszokra vágjuk, melyek mindegyike 1 karakterrel hosszabb, mint egy korábban látott szakasz. A kimenet (az új szakasz kódolása) (i, c), ahol

- i a régi (1 karakterrel rövidebb) szakasz címe,
- c az új karakter.

Az aktuális szakasz címét egyesével növeljük minden új szakaszra.

LZ78 algoritmus

Példa.

a b a b b b b b a b b a b

LZ78 algoritmus

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

LZ78 algoritmus

1	(0,a)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).

LZ78 algoritmus

1	(0,a)
2	(0,b)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).
2. “b” egy új szakasz, így a kimenet (0, b).

LZ78 algoritmus

1	(0,a)
2	(0,b)
3	(1,b)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).
2. “b” egy új szakasz, így a kimenet (0, b).
3. “ab” egy új szakasz; a régi “a” szakasz címe 1, és az új karakter “b”, így a kimenet (1, b).

LZ78 algoritmus

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).
2. “b” egy új szakasz, így a kimenet (0, b).
3. “ab” egy új szakasz; a régi “a” szakasz címe 1, és az új karakter “b”, így a kimenet (1, b).
4. “bb” egy új szakasz; a régi “b” szakasz címe 1, és az új karakter “b”, így a kimenet (2, b).

LZ78 algoritmus

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)
5	(4,a)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).
2. “b” egy új szakasz, így a kimenet (0, b).
3. “ab” egy új szakasz; a régi “a” szakasz címe 1, és az új karakter “b”, így a kimenet (1, b).
4. “bb” egy új szakasz; a régi “b” szakasz címe 1, és az új karakter “b”, így a kimenet (2, b).
5. “bba” egy új szakasz; a régi “bb” szakasz címe 4, és az új karakter “a”, így a kimenet (4, a).

LZ78 algoritmus

1	(0,a)
2	(0,b)
3	(1,b)
4	(2,b)
5	(4,a)
6	(5,b)

Példa.

a b a b b b b b a b b a b

1. fázis: a szöveget olyan szakaszokra tagoljuk, melyek korábban nem fordultak elő:

a b ab bb bba bbab

2. fázis: kódolás.

1. “a” egy új szakasz, így a kimenet (0, a).
2. “b” egy új szakasz, így a kimenet (0, b).
3. “ab” egy új szakasz; a régi “a” szakasz címe 1, és az új karakter “b”, így a kimenet (1, b).
4. “bb” egy új szakasz; a régi “b” szakasz címe 1, és az új karakter “b”, így a kimenet (2, b).
5. “bba” egy új szakasz; a régi “bb” szakasz címe 4, és az új karakter “a”, így a kimenet (4, a).
6. “bbab” egy új szakasz; a régi “bba”szakasz címe 5, és az új karakter “b”, így a kimenet (5, b).

6. feladat

Tömörítsük az 0100010100010100011101011 bitsorozatot az LZ78 algoritmussal. Használjunk bináris címeket.

6. feladat

Tömörítsük az 0100010100010100011101011 bitsorozatot az LZ78 algoritmussal. Használjunk bináris címeket.

Megoldás.

Tagolás:

szakasz	0	1	00	01	010	0101	000	11	01011
cím	1	2	3	4	5	6	7	8	9
bin. cím	0001	0010	0011	0100	0101	0110	0111	1000	1001

6. feladat

Tömörítsük az 0100010100010100011101011 bitsorozatot az LZ78 algoritmussal. Használjunk bináris címeket.

Megoldás.

Tagolás:

szakasz	0	1	00	01	010	0101	000	11	01011
cím	1	2	3	4	5	6	7	8	9
bin. cím	0001	0010	0011	0100	0101	0110	0111	1000	1001

A tömörített sorozat
(0000,0) (0000,1) (0001,0) (0001,1) (0100,0) (0101,1) (0011,0) (0010,1) (0110,1)

7. feladat

Csomagoljuk ki a 000100000010011010011010 1101 szöveget az LZ78 algoritmus segítségével.

7. feladat

Csomagoljuk ki a 000100000010011010011010 1101 szöveget az LZ78 algoritmus segítségével.

Megoldás. 0. lépés: meghatározzuk, hány bit a címek hossza.

7. feladat

Csomagoljuk ki a 000100000010011010011010 1101 szöveget az LZ78 algoritmus segítségével.

Megoldás. 0. lépés: meghatározzuk, hány bit a címek hossza.

Pl. 2 bittel legfeljebb 4 szakasz lenne megadható, azaz legfeljebb 12 bit lehetne a teljes kódolt sorozat. Tehát legalább 3 bit a címek hossza.

7. feladat

Csomagoljuk ki a 000100000010011010011010 1101 szöveget az LZ78 algoritmus segítségével.

Megoldás. 0. lépés: meghatározzuk, hány bit a címek hossza.

Pl. 2 bittel legfeljebb 4 szakasz lenne megadható, azaz legfeljebb 12 bit lehetne a teljes kódolt sorozat. Tehát legalább 3 bit a címek hossza. 4 bit viszont már nem lehet, mert az első cím mindig csupa 0. Tehát a címek 3 bit hosszúak.

7. feladat

Csomagoljuk ki a 000100000010011010011010 1101 szöveget az LZ78 algoritmus segítségével.

Megoldás. 0. lépés: meghatározzuk, hány bit a címek hossza.

Pl. 2 bittel legfeljebb 4 szakasz lenne megadható, azaz legfeljebb 12 bit lehetne a teljes kódolt sorozat. Tehát legalább 3 bit a címek hossza. 4 bit viszont már nem lehet, mert az első cím mindig csupa 0. Tehát a címek 3 bit hosszúak.

1. lépés. Felbontjuk a sorozatot címekre és új karakterekre:

$$(000, 1)\,(000, 0)\,(001, 0)\,(011, 0)\,(100, 1)\,(101, 0)\,(110, 1)$$

2. lépés. Az eredeti sorozat rekonstrukciója:

$$1\ 0\ 10\ 100\ 1001\ 10010\ 100101$$