COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui episod al serialului dedicat compresiilor de date vă vom prezenta metoda de compresie cu dicționare Lempel Ziv, cea mai rapidă metodă de compresie, și variante ale acesteia.

Metoda de compresie ***LZ*** (***L**empel-**Z**iv*) a fost implementată pentru prima dată de către ***Abraham Lempel*** și ***Jacob Ziv*** în anul 1977 și această implementare este denumită ***LZ77***. Ulterior metoda a suferit unele modificări și au apărut variantele ***LZ78***, ***LZW*** (*L**empel-Z**iv-W**elch* - 1984), ***LZSS*** (*L**empel-Z**iv-S**torer-S**zymanski* - 1986).

Metoda LZ77

Metoda de compresie *LZ77* constă în păstrarea ultimelor *n* simboluri generate de o sursă de informație *S* și găsirea celei mai lungi secvențe de lungime maximă L_s ($L_s < n$) de simboluri generate de sursa *S* în cadrul secvenței de *n* simboluri stocate.

Algoritmul de compresie

Algoritmul de compresie pentru varianta *LZ77* este foarte simplu. La început avem un șir *T* care conține *n* simboluri. Fiecare dintre cele *n* simboluri se inițializează cu primul simbol al alfabetului unei surse de informație *S*. În continuare, se păstrează primele L_S simboluri generate de sursa de informație *S* pe ultimele L_S poziții din șirul *T*. Cele L_S simboluri se transmit la ieșirea *E*.

La pasul următor avem un șir *w* inițial de lungime 0. Atâta timp cât șirul *w* se regăsește în șirul *T* acesta este concatenat cu următorul simbol generat de sursa de informație *S*. Vom nota prin w_i șirul obținut din primele *i* simboluri ale șirului *w*. Fie *i* lungimea lui *w* în momentul în care acesta nu se mai regăsește în șirul *T*; aceasta înseamnă că șirul w_{i-1} este cel mai lung șir generat de sursa *S* care se află în șirul *T* începând de la poziția *p*. La ieșirea *E* se transmite poziția *p*, numărul (*i* - 1) și simbolul *c* de pe ultima poziție a șirului *w*, ceea ce se explică prin faptul că *următoarele i elemente ale șirului inițial se obțin din i - 1 simboluri consecutive din șirul T începând de la poziția p și simbolul c*. Șirul *T* se deplasează la stânga cu *i* poziții, iar ultimele *i* poziții vor fi umplute cu simbolurile șirului *w*. Acest pas se execută până când sursa de informație *S* nu mai generează simboluri.

Pentru a descrie în *pseudocod* algoritmii din cadrul acestui articol, vom nota cu '+' operația de concatenare dintre două șiruri de simboluri sau dintre un șir de simboluri și un caracter, vom considera că prima poziție a unui șir de simboluri este `0` și vom avea nevoie de următorii subalgoritmi:

- `CitesteSir(S, I)` - citește de la sursa de informație `S` un șir de maxim `I` simboluri pe care îl returnează ca rezultat;
- `CitesteSim(S)` - citește de la sursa de informație `S` un singur simbol pe care îl returnează ca rezultat;
- `CitesteNum(S)` - citește de la sursa de informație `S` un șir de simboluri pe care îl convertește la un număr care este returnat ca rezultat;
- `CitesteBit(S)` - citește de la sursa de informație `S` un șir de simboluri pe care îl convertește la o valoare logică (`0` sau 1) care constituie rezultatul subalgoritmului;
- `Generează(S)` - returnează ca rezultat valoarea logică `adevărat` dacă sursa `S` mai generează simboluri și valoarea `fals` în caz contrar;
- `Transmite(E, I1, ..., In)` - transmite la ieșirea `E` conținutul variabilelor I_1, ..., I_n;
- `Lungime(w)` - returnează ca rezultat lungimea șirului de simboluri `w`;
- `Primele(s, i)` - returnează ca rezultat șirul de simboluri obținut din primele `i` simboluri ale șirului `s`;
- `Ultimele(s, i)` - returnează ca rezultat șirul de simboluri obținut din ultimele `i` simboluri ale șirului `s`;
- `Găsit(s, s1)` - returnează ca rezultat valoarea logică `adevărat` dacă șirul s_1 este subșir al șirului `s` și valoarea `fals` în caz contrar;
- `Poziție(s, s1)` - returnează ca rezultat poziția de început a primei apariții a șirului s_1 în șirul `s` dacă șirul s_1 este subșir al șirului `s` și valoarea `-1` în caz contrar.

În continuare vom prezenta algoritmul de compresie în *pseudocod*:

```
se inițializează n cu o constantă
se inițializează L_S cu o constantă
se inițializează T
w ← CitețteȘir(S, L_S)
Transmite(E, w)
T ← Primele(T, n - L_S) + w
dacă nu Generează(S) atunci
   ieșire
sfârșit dacă
cât timp Generează(S) execută
   w ← ""
   cât timp Generează(S) și Găsit(T, w) execută
      c ← CitețteSim(S)
      w ← w + c
   sfârșit cât timp
   i ← Lungime(w)
   p ← Poziție(T, Primele(w, i - 1))
   Transmite(E, p, i - 1, c)
   T ← Ultimele(T, n - i) + w
sfârșit cât timp
```

În figura 1 este ilustrat modul în care se realizează compresia unui șir de simboluri folosind metoda *LZ77*. Se poate observa foarte ușor felul în care este actualizat șirul de simboluri *T* și cum variază șirul *w* la fiecare pas. Pasul 0 reprezintă pasul de dinaintea instrucțiunii repetitive **cât timp**.

Alfabetul A= {a, b, c, d}
S = "abbaabbaababbaacdaabaaba"
T = "aaaaaaaa"; n = 8; L_S = 4

Pas	Valoare T	Valoare w	Ieșire (p, i, c)
0	"aaaaabba"	"abba"	"abba"
1	"aaaaabba"	"abbaa"	(4, 4, "a")
2	"bbaabbaa"	"bab"	(1, 2, "b")
3	"abbaabab"	"baac"	(2, 3, "c")
4	"ababbaac"	"d"	(0, 0, "d")
5	"babbaacd"	"aab"	(4, 2, "b")
6	"baacdaab"	"aaba"	(5, 3, "a")

Figura 1: Exemplu de compresie cu *LZ77*

Algoritmul de decompresie

Algoritmul de decompresie pentru metoda *LZ77* constă în interpretarea șirului comprimat de simboluri generate de o sursă de informație *S*.

Inițial se citesc L_S simboluri care se transmit și sunt stocate și pe ultimele L_S poziții ale șirului *T* de lungime *n*.

În continuare, la fiecare pas se citește un triplet (*p*, *i*, *c*), se transmit *i* simboluri din *T* începând cu poziția *p*, se transmite simbolul *c*, *T* se deplasează la stânga cu *i* + 1 poziții și ultimele *i* + 1 poziții din *T* se completează cu șirul obținut din cele *i* + 1 simboluri transmise.

În pseudocod acest algoritm este:

```
se inițializează n cu o constantă
se inițializează L_S cu o constantă
se inițializează T
w ← CitețteȘir(S, L_S)
Transmite(E, w)
T ← Primele(T, n - L_S) + w
dacă nu Generează(S) atunci
   ieșire
sfârșit dacă
cât timp Generează(S) execută
   p ← CitețteNum(S)
   i ← CitețteNum(S)
   c ← CitețteSim(S)
   w ← Primele(Ultimele(T, n - p), i) + c
   Transmite(E, w)
   T ← Ultimele(T, n - i - 1) + w
sfârșit cât timp
```

Observații

Complexitatea algoritmului de compresie este $O(n \cdot nr)$, unde *nr* este numărul de simboluri generate de sursa de informație *S*, deoarece funcția de căutare a unui subșir într-un șir are complexitatea *O(n)* și modificarea șirului *T* se realizează în maxim *O(n)* pași, în funcție de algoritmul utilizat (dacă se utilizează un șir circular, modificarea poate avea loc în *O(1)* pași).

Complexitatea algoritmului de decompresie este ***O(nr)*** în cazul în care modificarea șirului *T* se face în *O(1)* pași și construirea lui *w* se face la fiecare pas folosind *O(i)* instrucțiuni.

În general șirul *T* de simboluri poartă numele de ***fereastră mobilă***, acest algoritm fiind cel mai simplu algoritm de compresie din clasa *algoritmilor de compresie cu ferestre mobile*. *Dicționarul* în cazul algoritmului *LZ77* este constituit chiar de fereastra mobilă *T*.

Din cauză că șirul *T* este format numai din ultimele *n* simboluri generate de sursa de informație *S* algoritmul *LZ77* nu este optim, deoarece nu mai apar referințe la codificări anterioare deci, dacă mai apare un șir de simboluri care trebuie codificate identic cu un șir codificat anterior, se întâmplă destul de des să nu fie codificat în aceeași manieră (folosind un singur triplet (*p*, *i*, *c*)) fapt care duce la creșterea lungimii codificării întregului șir de simboluri generate de sursa *S*. Din acest motiv, cercetătorii ***Abraham Lempel*** și ***Jacob Ziv*** au renunțat la fereastra mobilă și în anul 1978 au realizat o nouă implementare în care *dicționarul* de compresie era constituit de șirurile de simboluri codificate anterior.

LZ77 este un algoritm fundamental de compresie a datelor și stă la baza clasei de algoritmi de compresie cu fereastră mobilă, în timp ce *LZ78* este baza unei noi clase de algoritmi de compresie.

Metoda LZSS

În cazul metodei *LZ77*, se observă foarte ușor că dacă ultimul șir de simboluri w_{i-1} care se regăsește în șirul *T* este

chiar șirul vid atunci, pentru a reprezenta șirul *w* care se codifică se transmit numerele *p* și (*i* - 1) care au valoarea 0 și simbolul din șirul *w*. Aceasta înseamnă că pentru a codifica un singur simbol trebuie două numere care sunt inutile.

Metoda *LZSS* constituie o îmbunătățire asupra metodei *LZ77*. Îmbunătățirea este aceea că în cazul enunțat anterior, dacă avem de codificat un singur simbol, se transmite bitul 0 urmat de simbol, iar dacă avem de codificat un șir mai lung de simboluri, se transmite bitul 1 urmat de perechea (*p*, *i*). În acest ultim caz, la iterația următoare șirul *w* va fi inițializat cu șirul format din ultimul simbol generat de sursa *S*.

În concluzie, dacă lungimea codificării unui șir este mai mare decât șirul necodificat, atunci se va transmite la ieșire necodificat, precedat de un bit setat pe 0. Se observă că în cazul acestui algoritm, dacă este nevoie să se codifice un șir, în fereastra mobilă se va introduce doar cel mai lung subșir care se regăsește în aceasta și nu primul subșir care nu se regăsește (cum se întâmplă în cazul algoritmului de compresie *LZ77*).

Algoritmul de compresie *LZSS* este:

```
se inițializează n cu o constantă
se inițializează L_S cu o constantă
se inițializează T
w ← CitesteSir(S, L_S)
Transmite(E, w)
T ← Primele(T, n - L_S) + w
dacă nu Generează(S) atunci
  ieșire
sfârșit dacă
  w ← ""
cât timp Generează(S) execută
  cât timp Generează(S) și Găsit(T, w) execută
    c ← CitesteSim(S)
    w ← w + c
  sfârșit cât timp
  i ← Lungime(w)
  dacă i = 1 atunci
    Transmite(E, 0, c)
    w' ← ""
  altfel
    p ← Poziție(T, Primele(w, i))
    dacă p = -1 atunci
      i ← i - 1
      p ← Poziție(T, Primele(w, i))
      w' ← c
      dacă i = 1 atunci
        Transmite(E, 0, Primele(w, 1))
      altfel
        Transmite(E, 1, p, i )
      sfârșit dacă
    altfel
      w' ← ""
      Transmite(E, 1, p, i )
    sfârșit dacă
  sfârșit dacă
  T ← Ultimele(T, n - i) + Primele(w, i)
  w ← w'
sfârșit cât timp
```

În figura următoare se poate observa modul de compresie a unui șir de simboluri generate de o sursă *S*, folosind algoritmul *LZSS*:

Alfabetul A= {a, b, c, d}
S = "abbaabbaabbabbaacdaabaaba"
T = "aaaaaaaa"; n = 8; L_S = 4

Pas	Valoare T	Valoare w	Ieşire (1, p, i) sau (0, c)
0	"aaaaabba"	"abba"	"abba"
1	"aaaaabba"	"abbaa"	(1, 4, 4)
2	"abbaabba"	"aba"	(1, 0, 2)
3	"baabbaab"	"abbaac"	(1, 2, 5)
4	"aababbaa"	"c"	(0, "c")
5	"ababbaac"	"d"	(0, "d")
6	"babbaacd"	"aab"	(1, 4, 2)
7	"bbaacdaa"	"baab"	(1, 2, 3)
8	"acdaabaa"	"ba"	(1, 5, 2)

Figura 2: Exemplu de compresie cu *LZSS*

Decompresia unui șir de simboluri comprimat cu metoda *LZSS* se realizează la fel ca în cazul algoritmului *LZ77* cu precizarea că în momentul decodificării unei perechi se verifică bitul care o precede, și anume, dacă bitul este 0, atunci, la ieșire se va transmite un simbol *c*, iar dacă este 1, atunci, la ieșire se va transmite subșirul din fereastra mobilă care începe de la o poziție *p* și are lungimea *i*. Acțiunea de decodificare va fi urmată de actualizarea ferestrei mobile.

În continuare prezentăm varianta pseudocod a algoritmului de decompresie *LZSS*:

```
se inițializează n cu o constantă
se inițializează L_S cu o constantă
se inițializează T
w ← CitesteSir(S, L_S)
Transmite(E, w)
T ← Primele(T, n - L_S) + w
dacă nu Generează(S) atunci
  ieșire
sfârșit dacă
cât timp Generează(S) execută
  b ← CitesteBit(S)
  dacă b = 0 atunci
    c ← CitesteSim(S)
    w ← c
    i ← 1
  altfel
```

```
    p ← CitesteNum(S)
    i ← CitesteNum(S)
    w ← Primele(Ultimele(T, n - p), i)
  sfârșit dacă
  Transmite(E, w)
  T ← Ultimele(T, n - i) + w
sfârșit cât timp
```

Ordinele de complexitate ale algoritmilor de compresie și decompresie pentru metoda *LZSS* sunt aceleași cu cele ale algoritmilor de compresie, respectiv decompresie, pentru algoritmul *LZ77*, în schimb, un șir de simboluri generat de o sursă de informație *S* este comprimat mai bine dacă se folosește algoritmul *LZSS*.

Metoda LZ78

Spre deosebire de cele două metode de compresie a datelor prezentate anterior, metoda *LZ78* nu folosește o fereastră mobilă, ci reține toate șirurile de simboluri codificate anterior într-o listă.

La începutul compresiei, mărimea listei de elemente T codificate anterior este 1 și conține pe poziția 0 șirul vid. La un pas, se consideră lista elementelor obținută la pasul anterior și se citește de la o sursă de informație *S* cel mai scurt șir de simboluri w care nu se află în lista elementelor T. În această listă se va adăuga șirul w, iar la ieșirea E se va transmite o pereche formată din numărul de ordine al primei apariții a șirului obținut din șirul w prin eliminarea ultimului simbol din acesta.

Pentru a putea prezenta algoritmul de compresie *LZ78* avem nevoie de subalgoritmul Caută(s, w), care primește ca parametri o listă de șiruri de simboluri s și un șir de simboluri w, returnând ca rezultat numărul de ordine al primei apariții a șirului w în lista s. Dacă w nu apare printre elementele listei s, atunci rezultatul subalgoritmului va fi -1. Vom considera că numărul de ordine al primului șir care apare în lista s este 0.

```
T[0] ← w              //inițial, lista T conține doar șirul vid
k ← 1                 //k reprezintă mărimea listei T
cât timp Generează(S) execută
  w ← ""
  cât timp Generează(S) și Caută(T, w) ≥ 0 execută
    c ← CitesteSim(S)
    w ← w + c
  sfârșit cât timp
  T[k] ← w
  k ← k + 1
  i ← Lungime(w)
  w ← Primele(w, i - 1)
  i ← Caută(T, w)
  Transmite(E, i, c)
sfârșit cât timp
```

În practică, atunci când dimensiunea listei T ajunge la o valoare prestabilită (datorită insuficienței resurselor de memorie) nu se mai adaugă elemente în aceasta, acest fapt ducând la ineficiența algoritmului pentru date foarte multe și pentru multe apariții ale unor șiruri de simboluri care nu se află printre elementele listei.

În figura 3 se poate observa modul în care se realizează compresia datelor folosind algoritmul prezentat anterior.

Alfabetul A = {a, b, c, d}
S = "abbaabbaababbaacdaabaaba"
T = [" "]

Pas	Valoare w	Indice element adăugat în T	Ieșire (i, c)
1	"a"	1	(0, "a")
2	"b"	2	(0, "b")
3	"ba"	3	(2, "a")
4	"ab"	4	(1, "b")
5	"baa"	5	(3, "a")
6	"bab"	6	(3, "b")
7	"baac"	7	(5, "c")
8	"d"	8	(0, "d")
9	"aa"	9	(1, "a")
10	"baab"	10	(5, "b")
11	"a"	11	(0, "a")

Figura 3: Exemplu de compresie cu *LZ78*

Algoritmul de decompresie devine acum foarte simplu deoarece, la fiecare pas, se interpretează o pereche (i, T) care se citește de la sursa de informații și se transmite șirul care se află pe poziția i în lista T urmat de caracterul din pereche.

Inițial, ca și în cazul compresiei, lista T conține doar șirul vid. La fiecare pas se adaugă în lista T șirul format din elementul de pe poziția i din T și din caracterul care a fost citit.

În pseudocod, algoritmul de decompresie *LZ78* este:

```
T[0] ← w              //inițial, lista T conține doar șirul vid
k ← 1                 //k reprezintă mărimea listei T
cât timp Generează(S) execută
  i ← CitesteNum(S)
  c ← CitesteSim(S)
  Transmite(E, T[i], c)
  T[k] ← T[i] + c
  k ← k + 1
sfârșit cât timp
```

Complexitatea algoritmului de compresie are ordinul $O(m \cdot n)$, unde m reprezintă media lungimii șirurilor de simboluri care se află în lista T, în cazul în care se folosește un algoritm banal de căutare, iar n reprezintă numărul total de simboluri generate de o sursă de informație *S*. În cazul în care se folosește un algoritm performant de căutare și lista T este reprezentată folosind *arbori de sufixe*, atunci această complexitate scade foarte mult.

Algoritmul de decompresie are ordinul de complexitate *O(n)*, unde *n* reprezintă numărul de simboluri care trebuie decodificate.

Metoda LZW

Welch a observat o deficiență a algoritmului *LZ78*, ajungând la concluzia că pentru a decodifica un șir de simboluri nu este nevoie de o pereche formată dintr-un indice și un simbol.

Prin urmare, acesta a modificat algoritmii de compresie și decompresie astfel:

- lista *T* era inițializată cu șiruri de simboluri, fiecare șir de simboluri fiind format din câte un simbol al alfabetului sursei de informație *S*. În lista *T* sunt atâtea șiruri câte simboluri are alfabetul;
- la fiecare pas din iterație:
  - ♦ șirul *w* este inițializat cu ultimul simbol al șirul *w* obținut la pasul anterior;
  - ♦ în lista *T* se adaugă cel mai scurt șir *w*, obținut prin concatenări succesive cu simboluri generate de sursa *S*, care nu se alfă în *T*;
  - ♦ la ieșire se transmite un singur cod care reprezintă numărul de ordine al apariției lui *w* din care se extrage ultimul simbol.

Acest algoritm este cunoscut sub denumirea ***LZW***.

În continuare prezentăm în *pseudocod* algoritmul care realizează compresia unui șir de simboluri generate de o sursă de informație *S* pentru metoda *LZW*. În figura 4 se poate observa modul în care se realizează compresia unui șir de simboluri.

```
se inițializează T cu cele m simboluri ale alfabetului sursei S
k ← m                          //k reprezintă mărimea listei, T
w ← ""
cât timp Generează(S) execută
  cât timp Generează(S) și Caută(T, w) ≥ 0 execută
    c ← CitesteSim(S)
    w ← w + c
  sfârșit cât timp
  i ← Caută(T, w)
  dacă i ≥ 0 atunci      //sursa S nu mai generează simboluri, sfârșitul compresiei
    Transmite(E, i)
  altfel
    T[k] ← w
    k ← k + 1
    i ← Lungime(w)
    w ← Primele(w, i - 1)
    i ← Caută(T, w)
    Transmite(E, i)
    w ← c
  sfârșit dacă
sfârșit cât timp
```

Pe exemplul din figura 4 se poate observa că pentru a comprima șirul se transmit 14 coduri, spre deosebire de algoritmul *LZ78* care, pentru a comprima același șir de simboluri, transmite 11 coduri și 11 simboluri.

Alfabetul A= {a, b, c, d}
S = "abbaabbaabbaacdaabaaba"
T = ["a", "b", "c", "d"]
k = 4

Pas	Valoare w	Indice element adăugat în T	Ieșire (i)
1	"ab"	4	(0)
2	"bb"	5	(1)
3	"ba"	6	(1)
4	"abb"	7	(4)
5	"baa"	8	(6)
6	"aba"	9	(4)
7	"abba"	10	(7)
8	"aa"	11	(0)
9	"ac"	12	(0)
10	"cd"	13	(2)
11	"da"	14	(3)
12	"aab"	15	(11)
13	"baab"	16	(8)
14	"ba"		(6)

Figura 4: Exemplu de compresie cu *LZW*

Algoritmul de decompresie este asemănător cu cel de la metoda *LZ78* cu deosebirea că în lista *T* se adaugă șirul format din șirul decodificat la pasul anterior și primul simbol al șirului decodificat la pasul curent.

În *pseudocod*, acest algoritm este:

```
se inițializează T cu cele m simboluri ale alfabetului sursei S
k ← m                          //k reprezintă mărimea listei, T
i ← CitesteNum(S)
w ← T[i]
Transmite(E, w)
cât timp Generează(S) execută
  i ← CitesteNum(S)
  T[k] ← w + Primele(T[i], 1)
  w = T[i]
  Transmite(E, T[i])
  k ← k + 1
sfârșit cât timp
```

Ordinele de complexitate ale algoritmilor de compresie și decompresie ale metodei *LZW* sunt aceleași ca și în cazul metodei *LZ78*.

Algoritmii *LZ78* și *LZW*, datorită rapidității, sunt utilizați des în programele comerciale de compresie singurul inconvenient fiind acela că mărimea listei *T* trebuie limitată datorită cantității mici de resurse disponibile pe sistemele de calcul.

Claudiu Soroiu este redactor al GInfo și poate fi contactat prin e-mail la adresa csoroiu@yahoo.com.