COMPRESIA datelor

Claudiu Soroiu

În cadrul acestui episod al serialului dedicat compresiilor de date vă vom prezenta o serie de transformări care au ca scop creșterea performanțelor diferiților algoritmi de compresie.

De-a lungul istoriei compresiilor de date s-a pus întrebarea: "*Există vreo posibilitate de a face ca datele să devină mai comprimabile?*" Ei bine, răspunsul la această întrebare s-a dovedit a fi "*Da, există!*".

În paginile următoare vă vom prezenta câteva transformări care se pot efectua asupra datelor pentru ca acestea să devină *mai comprimabile*, dar mai întâi vom prezenta câteva noțiuni teoretice.

Pentru un algoritm de compresie entropic, un șir de simboluri generat de o sursă de informație *S* se comprimă mai bine (devine *mai comprimabil*) dacă entropia acestuia scade în urma efectuării unei transformări.

Pentru ca o transformare să fie performantă, aceasta trebuie să îndeplinească următoarele condiții:
- ordinul de complexitate al algoritmului de transformare să fie cât mai mic posibil;
- diferența dintre lungimea șirului de simboluri obținut după aplicarea transformării să fie cât mai mică posibil (de cele mai multe ori lungimea șirului de simboluri obținut după aplicarea unei transformări este mai mare sau egală cu lungimea șirului inițial);
- lungimea șirului de simboluri obținut după aplicarea unui algoritm de compresie asupra rezultatului transformării să fie, de cele mai multe ori, mai mică sau egală cu lungimea șirului de simboluri obținut după aplicarea aceluiași algoritm de compresie asupra șirului netransformat;
- transformarea trebuie să fie reversibilă.

Transformările pe care le vom prezenta în continuare sunt transformări care, după aplicare, nu duc la pierdere de informație.

Compresia RLE

Să presupunem că o sursă de informație *S* generează un șir de *n* simboluri cu proprietatea că acesta conține subșiruri cu lungimea cuprinsă între a ($1 < a < n$) și b ($a < b \leq n$) simboluri, și toate simbolurile subșirurilor sunt identice. De exemplu, pentru șirul de 16 simboluri *aabcdeeeeffffgah*, subșirurile formate din același simbol care au lungimea cuprinsă între 1 și 8 sunt: *aa*, *eeee* și *ffff*.

Datorită acestor subșiruri, dimensiunea unui șir de simboluri generate de o sursă se poate reduce.

Metoda prin care se realizează comprimarea șirurilor de simboluri care conțin subșiruri formate din același simbol poartă numele de ***R**un **L**ength **E**ncoding* (***RLE** - compresie în lungul șirului*).

În practică, valorile a și b se consideră a fi 1, respectiv multiplii de m', unde m' este cea mai mică putere a lui 2 mai mare sau egală cu numărul total de simboluri pe care le poate genera o sursă de informație *S*. În cazul șirului prezentat anterior, numărul total de simboluri este 8, deci valoarea lui m' este 8.

Compresia *RLE* constă din următorii pași:
- se inițializează un contor i cu valoarea 1 și se reține primul simbol generat de sursa *S*.
- șirul generat de sursa de informație se parcurge simbol cu simbol;
- la fiecare pas, dacă simbolul curent este identic cu cel reținut la pasul anterior:
  - atunci dacă i este mai mic decât m':
    - atunci incrementează valoarea contorului i și se trece la pasul următor;
    - altfel transmite la ieșire simbolul reținut împreună cu valoarea i și inițializează i cu 1.
  - altfel, dacă valoarea i este egală cu 1:
    - atunci transmite simbolul reținut și reține simbolul nou citit;
    - altfel transmite la ieșire simbolul reținut și valoarea i, reține noul simbol citit și inițializează pe i cu valoarea 1.

În momentul în care sursa de informație *S* nu mai transmite simboluri se mai parcurge încă o dată pasul din

iterație, cu deosebirea că nu se mai reține nimic, ci se transmit valorile reținute.

Se observă că pentru șirul *aabcdeeeeffffgah* se transmite șirul *a* 2 *b c d e* 4 *f* 4 *g a h*. Lungimea noului șir este 12, față 16 cât are șirul inițial.

Pentru a decodifica acest șir, se citește primul simbol și se reține, iar apoi se parcurge secvențial șirul de simboluri. La fiecare pas, dacă simbolul este un număr, atunci se transmite simbolul reținut de câte ori indică numărul citit și se reține un nou simbol, în caz contrar se transmite simbolul reținut și se citește un nou simbol. După parcurgerea șirului de simboluri se transmite ultimul simbol citit, dacă acesta nu reprezintă un număr.

În condițiile prezentate anterior, decodificarea șirului *a* 2 *b c d e* 4 *f* 4 *g a h* are loc astfel:
- reține *a*;
- transmite *a* de 2 ori și reține *b*;
- transmite *b* și reține *c*;
- transmite *c* și reține *d*;
- transmite *d* și reține *e*;
- transmite *e* de 4 ori și reține *f*;
- transmite *f* de 4 ori și reține *g*;
- transmite *g* și reține *a*;
- transmite *a* și reține *h*;
- transmite *h*.

În practică, deoarece atât simbolurile sursei de informație, cât și numerele sunt codificate cu ajutorul unor simboluri care aparțin aceluiași alfabet, apare problema că nu putem decide dacă o reprezentare este simbol sau număr. Pentru a elimina această ambiguitate, în momentul în care trebuie să transmitem un simbol și numărul de apariții consecutive ale acestuia, se va transmite simbolul de două ori, iar la decodificare, în momentul în care se citesc două simboluri identice, vom putea ști cu siguranță că trebuie să citim o reprezentare care specifică numărul de apariții consecutive ale simbolului respectiv.

Dacă m' este diferit de numărul total de simboluri m pe care le poate genera o sursă de informație S, atunci acest alfabet se va extinde cu $m - m'$ simboluri.

Pentru a reprezenta un număr care are valoarea maximă m', se folosește exact un simbol din alfabetul extins al sursei de informație S, și anume simbolul care are numărul de ordine m' în cadrul șirului de simboluri (în ordine lexicografică) ale alfabetului extins.

Considerăm numărul p cu proprietatea că $m' = 2^p$. Pentru a reprezenta un număr, care are valoarea maximă $2^{n \cdot p}$, se folosesc exact n simboluri ale alfabetului extins.

În practică, dacă numărul maxim de simboluri ale unui subșir se consideră a fi m', atunci în cel mai rău caz (când șirul de simboluri care trebuie comprimat este format numai din subșiruri formate din simboluri identice care au lungimea 2) dimensiunea șirului de simboluri transmis este cu 50% mai mare decât dimensiunea șirului original.

Există și alte metode de a elimina ambiguitatea, însă metoda prezentată este cea mai performantă dintre toate și cea mai des utilizată în practică. Metoda se mai poate îmbunătăți dacă în loc să transmitem lungimea l a șirului de simboluri identice găsit, transmitem reprezentarea numărului l - 2, deoarece șirul are cel puțin două simboluri. Transmiterea numărului l - 2 ne conduce la modificarea condiției "dacă i este mai mic decât m'" în "dacă i este mai mic decât m' + 2" din algoritmul de codificare. În algoritmul de decodificare, în momentul în care se întâlnesc două simboluri consecutive identice, se citește lungimea l a subșirului care ar trebui să fie transmis și se transmit l + 2 simboluri.

Ordinul de complexitate al acestui algoritm este $O(n)$, unde n este numărul simbolurilor generate de o sursă de informație S.

Algoritmul *RLE* se folosește de foarte multe ori în cadrul compresiei imaginilor, în combinație cu metodele *Huffman* sau *aritmetică*. În cazul fișierelor de tipul *BMP*, algoritmul *RLE* se folosește singur.

Algoritmii entropici de compresie nu pot comprima șiruri de simboluri a căror entropie este egală cu $\log_2 m$, unde m este numărul de simboluri ale alfabetului sursei care a generat șirul. Aplicarea algoritmului *RLE* înaintea unui algoritm entropic asupra unui șir de simboluri, duce, de cele mai multe ori, la micșorarea entropiei șirului, de unde reiese concluzia că șirul de simboluri a devenit *mai comprimabil*.

Transformarea delta

Fie m numărul de simboluri distincte care pot fi generate de o sursă de informație S. Vom considera mulțimea $M = \{0, 1, ..., m - 1\}$. Vom transforma șirul de n simboluri generat de sursa de informație S în șirul L format din numerele de ordine ale simbolurilor din cadrul alfabetului. Toate elementele șirului L sunt din mulțimea M.

Transformarea *delta* constă în obținerea șirului L' din șirul L folosind următoarea relație de recurență:
$L'_1 = L_1$;
$L'_i = (L_i - L_{i-1})$ ***modulo*** m, $1 < i \le n$, unde L_i reprezintă al i-lea număr al șirului de numere L.

După ce se obține șirul L', pentru fiecare element L'_i, $1 \le i \le n$, la ieșire se transmite simbolul care are numărul de ordine L'_i în cadrul alfabetului sursei de informație S.

Transformarea inversă constă în a transforma șirul L' în șirul L și se realizează folosind următoarea relație de recurență:
$L_1 = L'_1$,
$L_i = (L'_i + L'_{i-1})$ ***modulo*** m, $1 < i \le n$.

Se observă că, după aplicarea transformării, lungimea șirului de simboluri obținut după transformare este aceeași cu lungimea șirului de simboluri generat de sursa de informație S.

Algoritmul folosit în practică pentru transformarea *delta* este următorul:

```
ultim ← 0
m ← numărul de simboluri ale alfabetului sursei S
```

Alfabetul $A = \{a, b, c, d, e, f, g, h\}$
$m = 8$
ultim = 0
$S = aabcdeeeeffffgah$

Pas	Valoare *ultim*	Simbol citit	Cod simbol (*c*)	Ieşire
1	0	*a*	0	*a*
2	0	*a*	0	*a*
3	0	*b*	1	*b*
4	1	*c*	2	*b*
5	2	*d*	3	*b*
6	3	*e*	4	*b*
7	4	*e*	4	*a*
8	4	*e*	4	*a*
9	4	*e*	4	*a*
10	4	*f*	5	*b*
11	5	*f*	5	*a*
12	5	*f*	5	*a*
13	5	*f*	5	*a*
14	5	*g*	6	*b*
15	6	*a*	0	*c*
16	0	*h*	7	*h*

Figura 1

```
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   Transmite(E, Sim(S, (c - ultim) modulo m))
   ultim ← c
sfârşit cât timp
```

Algoritmul folosit în practică pentru inversarea transformării *delta* este următorul:

```
ultim ← 0
m ← numărul de simboluri ale alfabetului sursei S
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   ultim ← (ultim + c) modulo m
   Transmite(E, Sim(S, ultim))
sfârşit cât timp
```

Notaţia `a modulo b` semnifică restul împărţirii lui `a` la `b`.

Pentru a descrie aceşti algoritmi, pe lângă subalgoritmii prezentaţi în episodul anterior al compresiilor de date, am introdus subalgoritmii:

- `Ord(S, c)` - returnează ca rezultat numărul de ordine al simbolului `c` în cadrul alfabetului sursei de informaţie `S`;
- `Sim(S, i)` - returnează ca rezultat simbolul cu numărul de ordine `i` în cadrul alfabetului sursei de informaţie `S`.

În figura 1 se poate observa modul de transformare a şirului de simboluri *aabcdeeeeffffgah*, generat de sursa de informaţie *S*, care are alfabetul format din simbolurile {*a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*}. Se observă că şirul *aabbbbaaabaaabch* obţinut după aplicarea transformării conţine mai puţine simboluri distincte.

Transformarea *delta* are următoarele proprietăţi:

- în cazul în care în şirul iniţial se află multe secvenţe formate din simboluri identice, atunci acestea se transformă în secvenţe de simboluri care au numărul de ordine 0 în cadrul alfabetului sursei de informaţie *S*;
- în cazul în care în şirul generat de sursa *S* se află multe secvenţe formate din simboluri care au proprietatea că diferenţa dintre numerele de ordine a două simboluri consecutive este o constantă *c*, atunci acestea se transformă în şiruri de simboluri care au numărul de ordine (*c* ***modulo*** *m*) în cadrul alfabetului sursei de informaţie;
- ordinul de complexitate al transformării *delta* şi al inversei acesteia este $O(n)$, unde *n* reprezintă numărul de simboluri al şirului generat de sursa *S*;
- entropia şirului de simboluri după aplicarea transformării se modifică şi în cazul folosirii pentru şiruri de simboluri generate de *surse de informaţie multimedia* (sursă de informaţie care transmite şiruri de simboluri reprezentabile *audio* sau *video*) scade.

Transformarea MTF

Considerăm şirul de numere *L* construit din numerele de ordine ale simbolurilor, generate de sursa *S*, în cadrul alfabetului sursei. Fiecare element al şirului *L* aparţine mulţimii $M = \{0, 1, ..., m - 1\}$, unde *m* reprezintă numărul de simboluri distincte care pot fi generate de sursa *S*. Şirul *L* are lungimea *n*, unde *n* este numărul de simboluri generate de sursa *S*.

Transformarea ***MTF*** (***M**ove **T**o **F**ront - mută în faţă*) constă în obţinerea şirului *L'* din şirul *L* astfel:

- iniţial se consideră *T* cu *m* elemente şi $T_i = i$, $i \in M$;
- se parcurge secvenţial şirul *L* şi pentru fiecare element L_j al şirului, L'_j va primi ca valoare numărul T_k, unde *k* reprezintă poziţia pe care se află elementul L_j în şirul *T*, şi elementul T_k se va muta pe poziţia 0 prin deplasarea la dreapta cu o poziţie a elementelor $T_0, T_1, ..., T_{k-1}$;
- pentru fiecare element L'_j, $1 \le j \le n$, la ieşire se va transmite simbolul care are numărul de ordine L'_j în cadrul alfabetului sursei *S*.

Inversa transformării *MTF* constă în obţinerea şirului *L* din şirul *L'* astfel:

- iniţial se consideră *T* cu *m* elemente şi $T_i = i$, $i \in M$;
- se parcurge secvenţial şirul *L'* şi pentru fiecare element L'_j al şirului, L'_j va primi ca valoare numărul T_k, $k = L_j$, şi elementul T_k se va muta pe poziţia 0 prin deplasarea la dreapta cu o poziţie a elementelor $T_0, T_1, ..., T_{k-1}$;
- pentru fiecare element L_j, $1 \le j \le n$, la ieşire se va transmite simbolul care are numărul de ordine L_j în cadrul alfabetului sursei *S*.

În exemplul din figura 2 se poate observa modul de funcţionare al transformării *MTF*.

În continuare vom prezenta în pseudocod algoritmii utilizaţi în practică pentru transformarea *MTF* şi inversa acesteia.

$A = \{a, b, c, d, e, f, g, h\}$
$S = aabcdeeeeffffgah$
$T = \{0, 1, 2, 3, 4, 5, 6, 7\}$

Pas	Simbol citit	Cod simbol (L_i)	Poziția k	Valoare T	Ieșire
1	*a*	0	0	0, 1, 2, 3, 4, 5, 6, 7	*a*
2	*a*	0	0	0, 1, 2, 3, 4, 5, 6, 7	*a*
3	*b*	1	1	1, 0, 2, 3, 4, 5, 6, 7	*b*
4	*c*	2	2	2, 1, 0, 3, 4, 5, 6, 7	*c*
5	*d*	3	3	3, 2, 1, 0, 4, 5, 6, 7	*d*
6	*e*	4	4	4, 3, 2, 1, 0, 5, 6, 7	*e*
7	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
8	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
9	*e*	4	0	4, 3, 2, 1, 0, 5, 6, 7	*a*
10	*f*	5	5	5, 4, 3, 2, 1, 0, 6, 7	*f*
11	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
12	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
13	*f*	5	0	5, 4, 3, 2, 1, 0, 6, 7	*a*
14	*g*	6	6	6, 5, 4, 3, 2, 1, 0, 7	*g*
15	*a*	0	6	0, 6, 5, 4, 3, 2, 1, 7	*g*
16	*h*	7	7	7, 0, 6, 5, 4, 3, 2, 1	*h*

Figura 2

Deoarece șirul *T* conține *m* numere distincte care aparțin mulțimii *M*, putem considera că *T* este o permutare a mulțimii *M*. În practică nu se folosește șirul *T*, ci un alt șir *W*, care reprezintă inversa permutării *T*, astfel că pe poziția *i*, $i \in M$, din șirul *W* se află poziția pe care o ocupă simbolul cu număul de ordine *i* în șirul *T*. Acest lucru diminuează numărul de calcule care trebuie efectuate pentru a determina poziția unui simbol în șirul *T*. Cu ajutorul șirului *W* determinarea poziției se efectuează în $O(1)$ pași, spre deosebire de cei $O(m)$ necesari în cazul folosirii șirului *T*.

Algoritmul pentru transformarea *MTF* este:

```
m ← numărul de simboluri ale alfabetului sursei S
W[i] ← i, i = 1, 2, ..., m
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   Transmite(E, Sim(W[c]))
   j ← 0
   cât timp j < m execută
      dacă W[j] < W[c] atunci
         W[j] ← W[j] + 1
      sfârșit dacă
      j ← j + 1
   sfârșit cât timp
   W[c]← 0
sfârșit cât timp
```

În cazul algoritmului de inversare al transformării *MTF* se folosește șirul *T*. Acest algoritm este:

```
m ← numărul de simboluri ale alfabetului sursei S
T[i] ← i, i = 1, 2, ..., m
cât timp Generează(S) execută
   c ← Ord(S, CitesteSim(S))
   Transmite(E, Sim(T[c]))
   aux ← T[c]
   j ← c
   cât timp j > 0 execută
      T[j] ← T[j- 1]
      j ← j - 1
   sfârșit cât timp
   T[0] ← aux
sfârșit cât timp
```

Transformarea *MTF* are următoarele proprietăți:

- la fel ca în cazul transformării *delta*, în situația în care șirul inițial conține multe secvențe formate din simboluri identice, acestea se transformă în secvențe de simboluri care au numărul de ordine 0 în cadrul alfabetului sursei de informație *S*;
- la fel ca în cazul transformării *delta*, șirul de simboluri obținut după aplicarea transformării *MTF* are aceeași lungime cu șirul inițial;
- în cazul în care în șirul generat de sursa *S* se află multe secvențe formate din simboluri care au proprietatea că diferența dintre numerele de ordine, în cadrul șirului *T* la un anumit pas, a două simboluri consecutive este 1, acestea se transformă în șiruri de simboluri care au numărul de ordine o constantă în cadrul alfabetului sursei de informație;
- ordinul de complexitate al transformării *MTF* și al inversei acesteia este $O(n \cdot m)$, unde *n* reprezintă numărul de simboluri al șirului generat de sursa *S*, iar *m* este numărul de simboluri ale alfabetului sursei; în cazul în care alfabetul sursei de informație are puține simboluri (de exemplu, caracterele codului *ASCII* sunt în număr de 256) și sursa generează foarte multe simboluri, atunci complexitatea transformării devine $O(n)$;
- entropia șirului de simboluri după aplicarea transformării se modifică și în cazul folosirii în combinație cu transformarea *BWT*, pe care o vom prezenta în continuare, scade.

Pentru transformarea *MTF* mai există și alte variante care mută un simbol găsit în șirul *T* la începutul șirului numai dacă acesta se află pe o poziție mai mare decât un număr dat *l*. De obicei, mutarea se face numai dacă poziția pe care se află simbolul este mai mare decât 2. Astfel se obține o transformare care duce la o creștere importantă a performanțelor algoritmilor de compresie.

Transformarea BWT

Cele trei transformări prezentate până acum se pot aplica *on-line* (la citirea unui simbol) asupra datelor, nefiind necesară memorie suplimentară pentru a stoca simbolurile generate de către o sursă de informație.

Pentru transformarea ***BWT*** (***B**urrows-**W**heeler **T**ransform*) este necesară o cantitate de memorie suplimentară destul de mare.

Transformarea *BWT* constă din următoarele:

- șirul de simboluri generate de o sursă de informație *S* se împarte în subșiruri care au *k* simboluri fiecare, ultimul segment având (*k* ***modulo*** *n*) simboluri;
- pentru fiecare subșir se generează toate permutările circulare care se ordonează lexicografic și se construiește un șir *S'* de simboluri format din ultimul simbol al fiecărei permutări;
- în final, pentru fiecare subșir, se transmite șirul corespunzător obținut împreună cu numărul de ordine *p* al subșirului obținut după sortare.

Această transformare a fost descoperită de către cercetătorii *Michael Burrows* și *David Wheeler* în anul 1983 dar nu a putut fi implementată eficient până în anul 1994 datorită faptului că nu se cunoștea un algoritm performant de sortare a permutărilor circulare ale unui șir de simboluri.

În continuare vom prezenta modul de transformare a unui singur șir de simboluri fără a-l împărți în subșiruri.

Fie șirul de simboluri *S* = "gazeta_de_informatică". Am notat prin "_" simbolul spațiu pentru o mai ușoară înțelegere a transformării *BWT*.

Pentru șirul *S* permutările circulare sunt:

```
gazeta_de_informatică    nformatică gazeta_de_i
azeta_de_informatică g   formatică gazeta_de_in
zeta_de_informatică ga   ormatică gazeta_de_inf
eta_de_informatică gaz   rmatică gazeta_de_info
ta_de_informatică gaze   matică gazeta_de_infor
a_de_informatică gazet   atică gazeta_de_inform
_de_informatică gazeta   tică gazeta_de_informa
de_informatică gazeta_   ică gazeta_de_informat
e_informatică gazeta_d   că gazeta_de_informati
_informatică gazeta_de   ă gazeta_de_informatic
informatică gazeta_de_
```

Ordonăm lexicografic aceste permutări:

```
_de_informatică gazeta   gazeta_de_informatică
_informatică gazeta_de   ică gazeta_de_informat
a_de_informatică gazet   informatică gazeta_de_
atică gazeta_de_inform   matică gazeta_de_infor
azeta_de_informatică g   nformatică gazeta_de_i
ă gazeta_de_informatic   ormatică gazeta_de_inf
că gazeta_de_informati   rmatică gazeta_de_info
de_informatică gazeta_   ta_de_informatică gaze
e_informatică gazeta_d   tică gazeta_de_informa
eta_de_informatică gaz   zeta_de_informatică ga
formatică gazeta_de_in
```

Dacă luăm ultimul simbol al fiecărei permutări după sortare se obține șirul *S'* = "aetmgci_dznăt_rifoeaa", iar numărul de ordine al șirului *S* este 11, primul șir având numărul de ordine 0. Șirul *S'* astfel obținut se transmite împreună cu numărul *p* = 11.

În prezent se cunoaște o clasă de algoritmi pentru sortarea permutărilor circulare ale unui șir de simboluri care au complexitatea $O(n \cdot \log n)$.

Inversa transformării *BWT* are la bază observația că șirul de simboluri *S'* este o permutare a șirului *S* și prin ordonarea lexicografică a simbolurilor acestuia se obține un șir de simboluri *F* identic cu cel care se obține dacă din fiecare șir din ordinea sortată luăm primul simbol. Așadar, dându-se un simbol S'_j în șirul *F* există un simbol *c* pe poziția *j*. Știind că aceasta este a *i*-a apariție a simbolului *c* în șirul *F*, căutăm și păstrăm a *i*-a apariție a simbolului *c* în șirul *S'*. O altă observație interesantă este aceea că, în șirul *S*, simbolul *c* precede simbolul S_j.

Dacă plecăm de la poziția *p* din șirul *S'* și efectuăm de *n* ori operația descrisă anterior, se obține un șir de simboluri identic cu șirul *S* ale cărui simboluri au fost puse în ordine inversă.

Algoritmul de obținere a șirului *S* din șirul *S'* și numărul *p* este următorul:

```
m ← numărul de simboluri ale alfabetului sursei
S' ← șirul de simboluri care trebuie inversat BWT
p ← poziția pe care o ocupă șirul S în secvența sortată
S ← șirul de simboluri care trebuie obținut
n ← lungimea șirurilor S' și S
pos[i] ← 0, i = 0, 1, ..., m - 1
i ← 0
cât timp i < n execută
   prev[i] ← pos[Ord(A ,S'[i])]
   pos[Ord(A ,S'[i])] ← pos[Ord(A ,S'[i])] + 1
   i ← i + 1
sfârșit cât timp
k ← n
i ← m-1
cât timp i ≥ 0 execută
   k ← k - pos[i]
   pos[i] ← k
   i ← i - 1
sfârșit cât timp
k ← p
i ← n-1
cât timp i ≥ 0 execută
   S[i] ← S'[k]
   k ← prev[k] + pos[Ord(A ,S'[k])]
   i ← i - 1
sfârșit cât timp
```

În algoritmul anterior, `pos[i]` reprezintă poziția corespunzătoare primei apariții a simbolului cu numărul de ordine `i` în șirul sortat al simbolurilor lui `S'`, iar `prev[k]` numărul de apariții ale simbolului cu numărul de ordine `k`. Am considerat că `A` este sursa de informație.

Algoritmul pentru inversarea transformării *BWT* are ordinul de complexitate $O(n)$, unde *n* este numărul total de simboluri generate de o sursă de informație.

Această transformare este importantă pentru că, dacă în șirul de simboluri generate de o sursă de informație sunt grupuri care încep cu același simbol, în șirul *S'* vor exista subșiruri formate din același simbol.

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