Laborator PS
Algoritmi de compresie a datelor

Lucrare 1: Introducere în Compresia Datelor

Lucrare 2: Algoritmul Shannon-Fano

Lucrare 3: Algoritmul Huffman static

Lucrare 4: Algoritmul Huffman dinamic

Prof. Dan Ştefănoiu <dan.stefanoiu@acse.pub.ro>

Ş.l. Alexandru Dumitraşcu <alexandru.dumitrascu@acse.pub.ro>

A. Obiective

Implementarea Algoritmului de compresie dinamică al lui Huffman. Testarea performanței acestuia, prin comparație cu cele ale Algoritmilor Shannon-Fano, Huffman static şi ale programelor de uz general **WinZIP** şi **WinRAR**.

B. Suport teoretic

Performanțele algoritmilor de compresie bazați pe modele statistice de tip static (neadaptiv) sunt limitate. Pentru a le ameliora, în loc să se efectueze construcția modelului statistic la sfârşitul citirii întregului set de date, aceasta se realizează „din mers", adică în cursul citirii datelor. Au fost elaborate, în acest fel, metode de compresie de tip *adaptiv*, unde modelul statistic suferă o permanentă îmbunătățire, în funcție de noile date citite de pe fluxul de intrare.

În cazul utilizării alfabetului static de ordin 0, arborele Huffman este construit în mod identic, atât în faza de compresie, cât şi în cea de decompresie, pe baza setului de contoare asociate simbolilor din alfabet. Alfabetul şi contoarele trebuie transmise alături de setul comprimat de date, pentru ca decriptarea să fie exactă.

O primă idee de a construi arborele binar în manieră adaptivă este următoarea: odată cu fiecare simbol citit/decriptat se reconstruieşte complet arborele. Această manieră de construcție este, însă, extrem de ineficientă, deoarece efortul de construcție de la pasul anterior se pierde integral. Ca urmare, algoritmul adaptiv astfel proiectat este cronofag, ceea ce face inoportună utilizarea sa într-o aplicație de timp real. Rezultă că strategia de construcție trebuie să se bazeze pe conceptul de *reactualizare*. Aceasta presupune

conservarea unei cât mai mari părți din efortul de construcție a arborelui binar, efort efectuat în cursul citirilor/decriptărilor anterioare. Mai precis, este de dorit ca arborele existent înainte de citirea/decriptarea simbolului curent să fie modificat ulterior, în funcție de informația nouă adusă de acest simbol. Astfel:

- dacă simbolul respectiv apare pentru prima dată, atunci arborelui i se va adăuga un nou nod în zona inferioară (opusă rădăcinii); un fiu al acestui nod este ocupat de noul simbol;
- dacă simbolul există deja în arbore, atunci nodul ocupat de el trebuie reactualizat prin:
  - creşterea cu o unitate a contorului simbolului (a ponderii nodului ocupat de acesta);
  - rearanjarea ramurilor arborelui în funcție de noua structură de contoare, prin schimbări reciproce de locuri între noduri.

În cazul adaptiv, deoarece nu se pot cunoaşte dinainte dimensiunile setului de date şi ale alfabetului corespondent, nu se mai poate adopta aceeaşi manieră de construcție. Totuşi, se poate păstra ideea care stă la baza relației de ordine dintre noduri, pentru a o redefini, cu scopul utilizării ei în cazul construcției adaptive a arborelui. Această nouă relație de ordine este sintetizată de *proprietatea de fraternitate*.

Prin definiție, un arbore binar cu noduri ponderate posedă *proprietatea de fraternitate* dacă:

- fiii aceluiaşi nod părinte (numite şi *(noduri) frați*) sunt indexați consecutiv;
- ponderea fiecărui nod descreşte sau rămâne constantă odată cu creşterea indexului asociat;
- când două noduri ocupă acelaşi nivel ierarhic, fiii nodului de indice mai mic au indici inferiori indicilor fiilor celuilalt nod.

Tot prin definiție, un arbore binar cu noduri ponderate având proprietățile:

- fraternitate;
- ponderea unui nod părinte este egală cu suma ponderilor fiilor săi direcți;

se numeşte *arbore Huffman*. Prin Metoda Huffman de tip adaptiv, este construit şi reactualizat un arbore Huffman asociat setului de date procesat. Fiecare nod al unui arbore Huffman este caracterizat de

3 parametri: ponderea asociată, poziția sa în cadrul arborelui şi tipul său. La reprezentarea în memorie a unui astfel de arbore, aceşti parametri trebuie luați în considerare, construcția, reactualizarea şi parcurgerea arborelui depinzînd în mod esențial de ei.

Inițializarea arborelui Huffman

Necunoaşterea a priori a dimensiunilor setului de date şi alfabetului de ordin 0 asociat impune adăugarea la alfabet a 2 simboli virtuali, diferiți de oricare dintre simbolii ce ar putea apărea în setul de date. Primul dintre ei se numeşte *simbol de sfârşit* şi este notat prin **EOS** (*end of string*). Rolul său este de a marca sfârşitul setului de date (în varianta originală sau comprimată). (Acest simbol poate fi utilizat şi în varianta neadaptivă a algoritmului.) Simbolul **EOS** este reprezentat inițial pe doi octeți, având adesea valoarea **256=0x100**. Al doilea simbol virtual se numeşte *simbol de evitare* şi este notat prin **ESC** (*escape*). Rolul său este extrem de important atît în faza de inițializare a arborelui, cât şi în cea de reactualizare, în cazul în care un nou simbol (care nu a mai apărut pe fluxul de intrare) este citit/decriptat. Mai precis, **ESC** va reprezenta fratele noului simbol, adică fiul pereche al nodului părinte ce trebuie creat cu ocazia apariției noului simbol, ca în Figura 3.1.

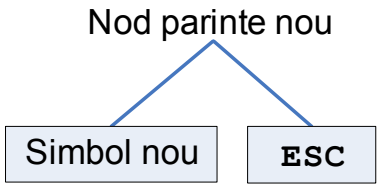


Figura 4.1. *Adăugarea unui nou simbol în arborele Huffman dinamic.*

Codul său original (inițial) este reprezentat tot pe 2 octeți, având valoarea uzuală de **257=0x101**. În cursul compresiei, cei 2 simboli virtuali primesc coduri diferite de cele originale, în funcție de poziția lor în arbore. Simbolul de evitare (**ESC**) poate apărea de mai multe ori în şirul de date utile comprimate (ori de câte ori codul care îi urmează are 8 biți şi descrie un nou simbol original, diferit de toți cei anteriori), în timp ce simbolul de

sfârşit (**EOS**) apare o singură dată (pe ultima poziție a şirului de coduri, marcând sfârşitul acestuia). Datorită acestui fapt, lor li se atribuie câte un contor fix, nemodificabil în cursul compresiei sau al decompresiei, de valoare egală cu 1.

În aceste condiții, arborele inițial (de la care se începe construcția) are doar 3 noduri: rădăcina şi două frunze ocupate de simbolii virtuali **EOS** şi **ESC** (ca în Figura 4.2). Alfabetul inițial conține de asemenea doar cei doi simboli virtuali.

$$\mathcal{A}^{0,2} = \{\mathbf{EOS}, \mathbf{ESC}\}$$

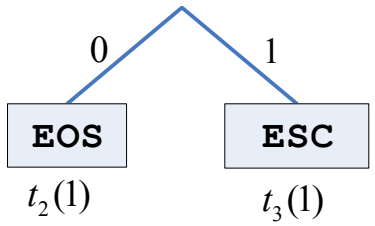


Figura 4.2. *Arborele Huffman dinamic inițial.*

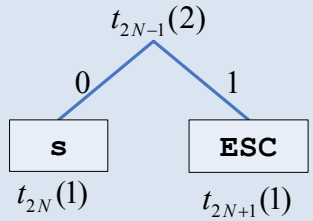
Arborele inițial posedă proprietatea de fraternitate, fiind un arbore Huffman unic determinat. Mai mult, aşa cum vom vedea, ultimul nod (de indice maxim) va fi întotdeauna ocupat de simbolul de evitare **ESC**. Noul cod al simbolului de evitare este 1, în această fază inițială. Ordinea celor 2 simboli virtuali este extrem de importantă aici, deşi ea nu corespunde întru totul intuiției (paradoxal, nu simbolul de sfârşit este ultimul din arbore, ci simbolul de evitare). Dacă ei ar figura în ordine inversă, atunci, prin reactualizare, indicele atribuit simbolului de sfârşit va fi mereu modificat, operație evitată în cazul ordinii precizate. De altfel, în Figura 4.2 este respectată ordinea lexicografică pe acelaşi nivel ierarhic, aşa cum impune definiția arborelui Huffman.

Reactualizarea arborelui Huffman în etapa de compresie

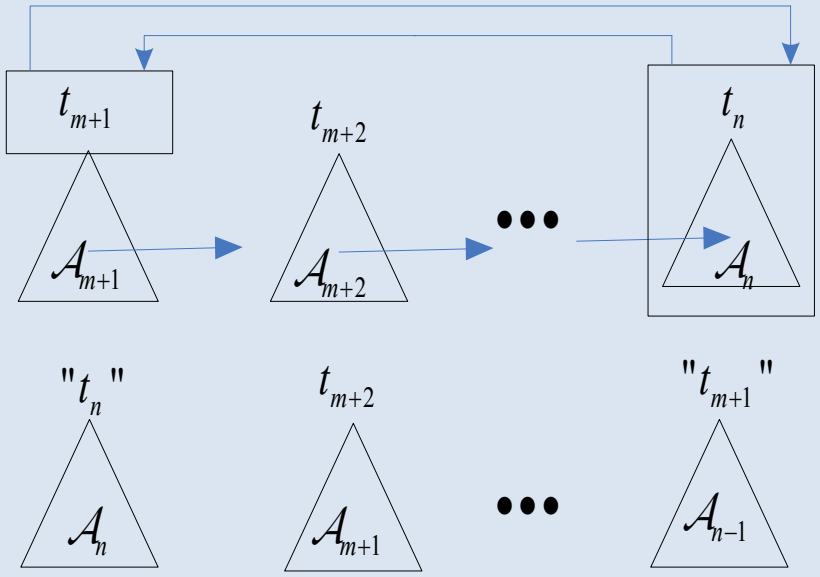
Etapa de reactualizare a arborelui Huffman în faza de compresie este cea mai dificilă şi complexă. Aceasta deoarece, chiar dacă este conservat efortul de construcție de la pasul anterior, cei doi arbori Huffman succesivi construiți recursiv pot avea structuri extrem de diferite prin informația transportată de simbolul curent citit/decriptat. Procedura de reactualizare este descrisă de Algoritmul 4.1.

Algoritmul 4.1. *Reactualizarea arborelui Huffman pentru compresie.*

1. Dacă noul simbol nu se regăseşte în alfabetul curent (reprezentat de frunzele arborelui), atunci el trebuie adăugat alfabetului după strategia care urmează:
   a) Nodul frunză ocupat de simbolul de evitare devine nod intermediar, t_{2N-1}, iar ponderea sa devine egală cu 2.
   b) Noul nod intermediar, t_{2N-1}, devine nod părinte pentru două noduri fii: t_{2N} (ocupat de simbolul s), cu ponderea $\mathcal{N}(t_{2N})$ (căci s apare pentru prima dată) şi t_{2N+1} (ocupat de simbolul **ESC**), cu aceeaşi pondere ($\mathcal{N}(t_{2N+1}) = \mathcal{N}(t_{2N})$).
   c) Deoarece ponderea nodului t_{2N-1} s-a modificat, este posibil ca proprietatea de fraternitate a arborelui să se fi pierdut. De aceea, ea trebuie refăcută atât prin mutări de subarbori cât şi prin reactualizări ale tuturor contoarelor nodurilor noului arbore. Figura următoare ilustrează mai clar modificarea locală adusă de simbolul s în acest caz:



2. Dacă noul simbol există deja în alfabetul curent, atunci se apelează la procedura de mai jos:
   a) Contorul simbolului s trebuie incrementat cu o unitate. Dacă nodul frunză ocupat de s este t_n (cu $n \in \overline{2,2N-2}$), atunci noul contor asociat va fi egal cu $\mathcal{N}(t_n)+1$.
   b) Schimbarea ponderii nodului t_n poate conduce la pierderea proprietății de fraternitate a arborelui.
3. Refacerea proprietății de fraternitate şi reactualizarea ponderilor întregului arbore binar. În arborele binar se află un *nod critic* t_n (cu $n \in \overline{2,2N-2}$), a cărui pondere a crescut cu o unitate. Pentru refacerea proprietății de fraternitate, trebuie găsit noul loc pe care îl va ocupa nodul critic t_n împreună cu toți descendenții săi (dacă există) şi să se incrementeze cu o unitate ponderea noului său părinte. Acesta va deveni, la rândul sau, noul nod critic şi astfel se va relua procedura pînă cînd se va ajunge la rădăcină. Reindexarea nodului critic t_n şi a altor noduri în scopul recâştigării fraternității se realizează astfel:
   a) Plecând de la indicele n, se caută, în ordinea descrescătoare a indicilor, primul nod cu pondere cel puțin egală valorii $\mathcal{N}(t_n)+1$. Fie acesta nodul t_m.
   b) Dacă $m = n-1$, nu mai este necesară mutarea lui t_n şi a descendeților săi, ci se incrementează ponderea părintelui lui t_n, care devine la rândul său nod critic şi algoritmul se reia de la pasul 3.
   c) Dacă $m < n-1$, între cele două noduri există cel puțin un nod cu pondere strict inferioară ponderii nodului critic. Pentru refacerea proprietății de fraternitate este necesară interschimbarea pozitiilor nodurilor t_{m+1} si t_n, împreună cu subarborii lor. Astfel:
      - dacă $m = n-2$, între t_m şi t_n există un singur nod, t_{n-1};
      - dacă $m < n-2$, între t_m şi t_n există cel puțin două noduri intermediare (t_{m+1}, t_{m+2}, ..., t_{n-1}); în acest caz, interschimbarea pozițiilor lui t_{m+1} şi t_n se realizează astfel (conform figurii care urmează):



- t_n se mută împreună cu arborele lui descendent în noua poziție, t_{m+1};
- arborele descendent al lui t_{n-1} devine noul arbore descendent al lui t_{m+1};
- subarborii nodurilor intermediare îşi schimbă părintele direct cu o poziție spre stânga;
- în final, nodul t_{m+2} primeşte arborele descendent al nodului t_{m+1};
- dacă printre nodurile intermediare se întîlnesc frunze, ele vor fi sărite;
- după interschimbare, ponderea nodului t_{m+1} se incrementează şi astfel devine noul nod critic, algoritmul reluându-se de la pasul 3.

Mutarea arborilor descendenți ai nodurilor în cursul reactualizării *nu necesită* reindexarea nodurilor componente. Se poate demonstra că această proprietate (care simplifică foarte mult algoritmul) este corectă, ea datorându-se proprietății de fraternitate.

Scalarea contoarelor în cursul reactualizării

În cazul unor şiruri lungi de date destinate compresiei/decriptării, este posibil ca operația de incrementare a contoarelor asociate nodurilor arborelui Huffman să conducă la depăşirea numărului maxim ce poate fi reprezentat în memorie. Acest neajuns se poate remedia printr-o (re)scalare a contoarelor asociate nodurilor. Scalarea constă în două operații:

1. Împărțirea la 2 a tuturor contoarelor asociate frunzelor (cu păstrarea intactă a contoarelor unitare).
2. Refacerea integrală a structurii arborelui HUFFMAN, în manieră statică, respectând proprietatea de fraternitate.

Împărțirea la 2 poate conduce la schimbarea fundamentală a formei arborelui, deoarece această operație poate degrada frecvența reală de apariție a simbolilor pe fluxul de intrare.

Să considerăm arborele Huffman din stânga Figurii 4.3, surprins înainte de scalarea contoarelor sale. După scalare, noul arbore nu mai verifică proprietatea de fraternitate. De aceea, el trebuie reactualizat. Se poate constata uşor că forma finală corectă (ilustrată în dreapta Figurii 4.3) este extrem de diferită de cea a arborelui inițial, existent înaintea rescalării.

Comparînd cei doi arbori Huffman din acest exemplu, se constată că a avut loc o transformare importantă a lungimilor codurilor asociate frunzelor. Astfel, lungimea maximă a codului, de 4 biți pentru simbolii s_4 şi **ESC**, a scăzut cu o unitate, lungimile de cod ale simbolilor s_1, s_2 şi **EOS** s-au conservat, iar lungimea de cod a simbolului s_3 a crescut cu o unitate.

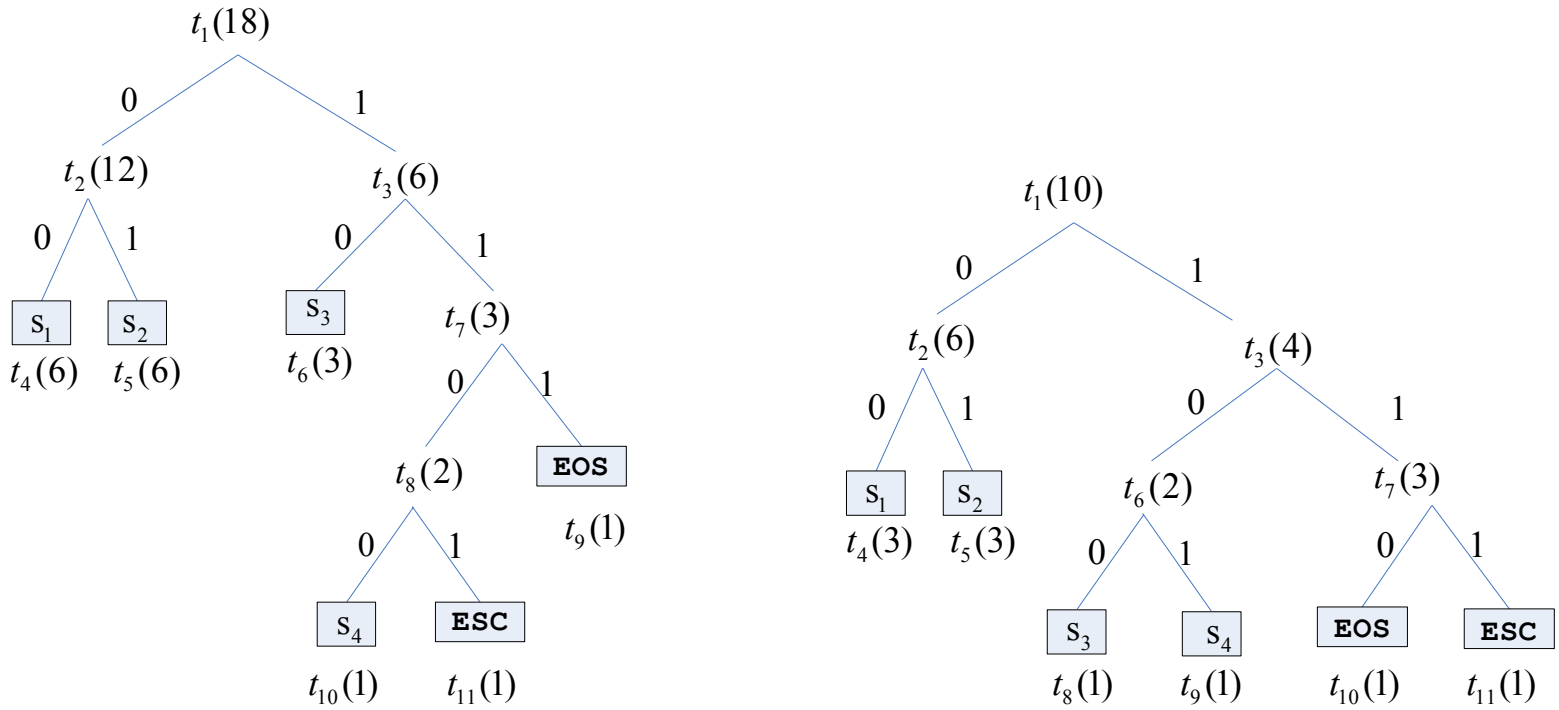


Figura 4.3. *Un exemplu de arbore Huffman dinamic înainte de (stânga) şi după (dreapta) scalarea contoarelor aferente.*

Aşadar, se poate considera că scalarea contoarelor a adus un câştig în materie de compresie (codul cel mai lung are acum doar 3 biți), deşi arborele Huffman final este foarte diferit de cel inițial. Totodată, aceste modificări ale lungimilor de cod pun în evidență şi un mecanism autocompensator de eroare (unele lungimi scad, altele cresc).

Aşa cum reiese şi din exemplul prezentat, scalarea contoarelor este o operație necesară pentru a evita blocarea algoritmului. Realizarea ei presupune, însă, stabilirea unui criteriu care să definească precis momentul cînd trebuie declanşată scalarea. Obiectivul urmărit este dublu: prevenirea depăşirii capacității de reprezentare a contoarelor şi evitarea emisiei de coduri a căror lungime depăşeşte o valoare dată (chiar dacă nu a fost depăşită capacitatea de reprezentare a contoarelor).

Reactualizarea arborelui Huffman în etapa de decompresie

Spre deosebire de construcția adaptivă a unui cod, decriptarea sa adaptivă este o operație mult mai simplă, ea presupunînd doar o parcurgere descendentă a arborelui. Vom prezenta în continuare algoritmul asociat acestei operații.

Algoritmul 4.2. *Reactualizarea arborelui Huffman pentru decompresie.*

1. Se citeşte fluxul de intrare la nivel de bit (în mod normal, se citeşte câte un octet, în cadrul căruia se analizează fiecare bit). În paralel, se construieşte şi se reactualizează arborele Huffman întocmai ca în faza de compresie. Inițial, arborele are forma din Figura 4.2. Primul simbol decriptat va fi **EOS** (cod 0), dacă fluxul de intrare este vid, sau **EOS** (cod 1), în caz contrar. Simbolul de evitare va fi urmat de codul original (pe 8 biți) al primului simbol valid din setul de date.
2. Se decriptează simbolul curent de pe fluxul de intrare, prin parcurgerea arborelui Huffman existent, începând cu rădăcina şi terminînd cu una dintre frunzele sale. Parcurgerea arborelui este ghidată de succesiunea biților fluxului de intrare:
   a) Dacă simbolul decriptat este **EOS**, atunci se citeşte următorul octet de pe fluxul de intrare. Acesta va fi codul original al simbolului curent.
   b) Dacă simbolul decriptat este diferit de **EOS**, atunci simbolul a mai apărut pe fluxul de intrare, frunza ocupată de el în arborele Huffman oferind codul original (pe 8 biți) al acestuia.

   În ambele situații, codul original al simbolului trebuie înscris pe fluxul de ieşire. Dacă ultimul simbol decriptat este **EOS**, algoritmul se încheie. În caz contrar, se trece la pasul următor.
3. Se reactualizează arborele HUFFMAN utilizînd Algoritmul 4.1 nemodificat, deoarece, în această fază, nu mai este necesară o tabelă de coduri. Algoritmul se reia de la pasul 2.

Exemplul care urmează are rolul de a pune în evidență cîteva aspecte importante pentru algoritmul de generare şi manevrare a codurilor (mai ales în faza de compresie). Înainte de a-l prezenta, sunt necesare două remarci:

- ori de câte ori simbolul citit nu se regăseşte printre frunzele arborelui binar, sunt emise două coduri succesive: primul este cel al simbolului de evitare **ESC** (cod preluat din arborele Huffman înainte de reactualizare), iar al doilea este codul original pe 8 biți (nemodificat) al simbolului;
- odată cu mutarea unui nod în altă poziție (în cursul reactualizării arborelui), este posibil ca unul sau mai mulți arbori descendenți să fie, la rîndul lor, mutați în alte poziții.

Reactualizarea codurilor frunzelor constă în reactualizarea părții mai semnificative a vechiului cod, deoarece partea mai puțin semnificativă nu este afectată de mutare.

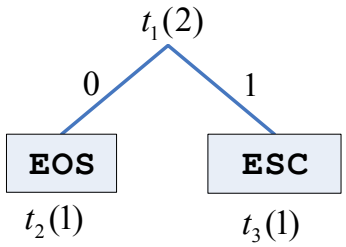
Exemplu

Se consideră setul de date format din 4 simboli, primii 3 fiind diferiți:

$$\mathcal{D}: \text{ADIA}$$

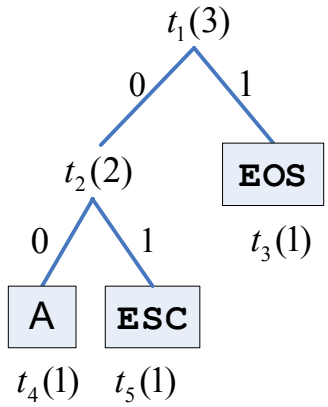
Se doreşte compresia adaptivă de tip Huffman a acestuia, urmată de decompresia adaptivă.

Pentru **compresie**, se pleacă de la arborele Huffman inițial din Figura 4.2, redat mai jos pentru uşurință.

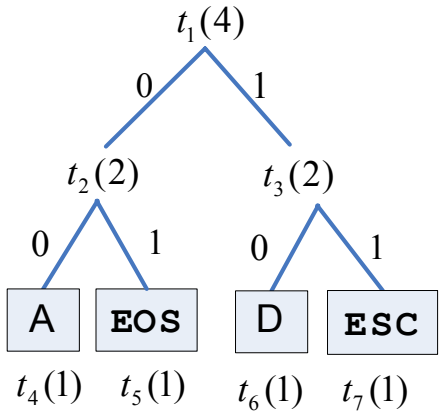


Evoluția ulterioară a acestuia este redată în continuare.

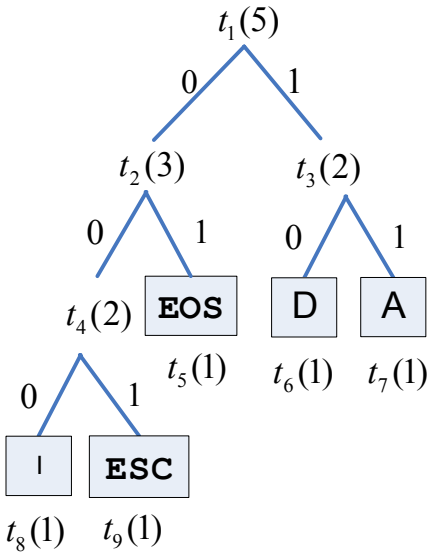
Se citeşte simbolul 'A' şi se emit două coduri (care însumează 9 biți): 1 (**ESC**) şi 01000001 (A). Arborele aferent devine:



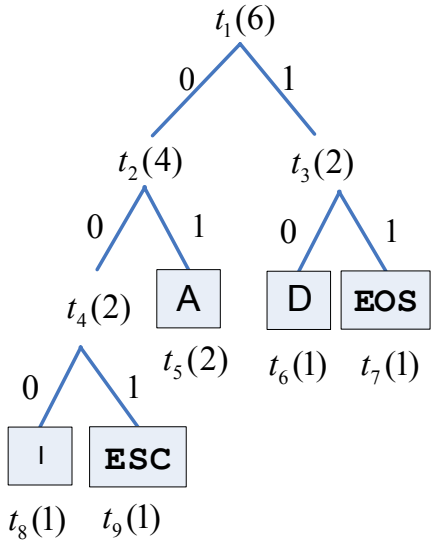
Se citeşte simbolul 'D' şi se emit două coduri (care însumează 10 biți): 01 (**ESC**)**)** şi 01000100 (D). Aceasta conduce la modificarea arborelui ca mai jos:



Se citeşte simbolul 'I' şi se emit două coduri (care însumează tot 10 biți): 11 (**ESC**)**)** şi 01001001 (I). Arborele se extinde către nivelul următor:



În fine, se citeşte simbolul 'A' şi se emite codul (de numai 2 biți): 11. Acesta înlocuieşte codul original pe 8 biți al simbolului 'A'. Arborele Huffmnan final arată astfel:



Şirul original de simboli este reprezentat pe cel puțin 41 de biți (32 de biți ai caracterelor, plus cel puțin 9 biți care indică sfîrşitul mesajului). Fluxul de date comprimate, care va fi utilizat în procedura de decompresie este constituit din următoarele coduri, numărînd 33 de biți:

1 01000001 01 01000100 11 01001001 11 11

Deşi setul de date este extrem de scurt, Algoritmul Huffman dinamic reuşeşte să realizeze compresia (pe seama faptului că unul dintre simboli se repetă). Această caracteristică este generală (cu condiția să existe repetiții de simboli) şi situează algoritmul în rândul celor mai eficiente rețete de compresie a datelor.

Pentru a realiza **decompresia (decriptarea)** şirului de biți de mai sus, se procedează astfel:

- Se citeşte bitul de valoare 1 şi se decriptează simbolul **ESC**, folosind arborele Huffman inițial. Codul său original nu este emis, însă, pe fluxul de ieşire. Operația care se execută în continuare este citirea următorului octet de pe fluxul de intrare: 01000001 (litera 'A'), care este înscris nemodificat pe fluxul de ieşire.
- Se reactualizează arborele Huffman şi se resetează parcurgerea sa din rădăcină.
- Se citesc biții succesivi de valori 01 (conform noului arbore) şi se constată că simbolul decriptat este tot **ESC**. În consecință, următorul octet al fluxului de intrare, 01000100 (litera 'D'), este înscris nemodificat pe fluxul de ieşire.
- Urmează o nouă reactualizare a arborelui.
- Cum următorii 2 biți citiți, 11, conduc tot la simbolul de evitare, următorul octet citit, 01001001 (litera 'I') este şi el descărcat direct pe fluxul de ieşire.
- Se reactualizează arborele, care va avea acum 5 frunze.

- Se reia citirea fluxului de intrare. După codul 11, se constată că, folosind arborele Huffman curent, s-a ajuns la nodul frunză ocupat de litera 'A'. Codul original al acesteia (01000001) este preluat din informația de nod a arborelui şi descărcat pe fluxul de ieşire.
- O nouă reactualizare a arborelui binar nu aduce noi frunze acestuia, ci o organizare diferită.
- Ultimii 2 biți citiți, aceiaşi 11, conduc la decriptarea simbolului **EOS**, care arată că decompresia a luat sfîrşit şi provoacă închiderea fluxului de ieşire.

Din acest exemplu, se observă clar cum un acelaşi cod (11) poate avea semnificații total diferite la momente de timp diferite (**ESC**, apoi 'A' şi, în fine, **EOS**). Cu toate acestea, decriptarea este exactă, deoarece structura arborelui Huffman este precisă, unic determinată de fiecare simbol decriptat şi identică în etapele de compresie-decompresie.

C. Sarcini de lucru

💻 **Tema 1 (Implementarea algoritmilor Huffman dinamici)**

a. Scrieți rutinele corespunzătoare Algoritmilor 4.1, 4.2 şi metodelor de compresie-decompresie din Metoda Huffman statică.

b. Verificați corectitudinea lor cu ajutorul unui program de test aplicat exemplului din secțiunile precedente, adică setului de date:

$$\mathcal{D}: \text{IT␣IS␣BETTER␣LATER␣THAN␣NEVER.}$$

(textul trebuie să fie reconstituit perfect).

c. Evaluați factorul de compresie realizat în acest exemplu (ținând cont şi de informația auxiliară). Este capabilă metoda să realizeze compresia?

💻 Tema 2 (Testarea algoritmilor Huffman dinamici)

a. Scrieți un program care realizează compresia şi decompresia celor 12 fişiere din corpusul de date descris în Lucrarea 1, cu ajutorul rutinelor de la tema precedentă.

b. Calculați norma diferenței dintre setul inițial de date şi cel decomprimat. Pentru fiecare set de date, indicați factorul de compresie aferent (incluzând informația auxiliară) şi evaluați duratele de compresie, respectiv decompresie. Comentați rezultatul obținut, cu referire la capacitatea de compresie a metodei pentru diferite tipuri de fişiere.

c. Comparați factorii de compresie de la punctul precedent cu cei obținuți prin utilizarea Metodelor Shannon-Fano, Huffman static şi a celor două programe de uz general **WinZIP** şi **WinRAR**. (Utilizați funcția **fread** pentru a evalua exact numărul de biți ai arhivelor produse de aceste utilitare.) Comentați rezultatele obținute, cu referire la compromisul dintre complexitatea algoritmilor de compresie şi performanțele acestora.