Codare **Shannon Fano** sau **Huffman Static** (2019-2020)

A. Tema

Sa se implementeze coderul si decodor ul pentru unul din algoritmii consideraţi: Shannon Fano sau Huffman static.

B. Aspecte teoretice

Ambii algoritmi consideraţi intră în categoria celor **statistici**. Implementarea cerută aici este cea consacrată adică cea cu **model semistatic** (notaţia Huffman static este incorectă dar consacrată, de aceea o vom folosi totuşi în continuare).

Codorul parcurge următoarele **etape**:

1. Determinare a statisticii
2. Construire model (coduri)
3. Codare pe baza modelului

Decodorul parcurge următoarele etape:

1. Preluare model
2. Decodare pe baza modelului

Pentru a simplifica transmiterea modelului spre decodor propun o soluţie în care de fapt se transmite spre decodor explicit statistica, **etapele** parcurse de **decodor** fiind în acest caz următoarele:

1. Preluare statistică (din antet)
2. Construire model (coduri)
3. Decodare pe baza modelului

B1. Codificare Shannon-Fano - exemplu

Să considerăm fluxul: `ABCDECDECD`

1. Determinare a statisticii

Obţinem frecvenţa de apariţie a fiecărui simbol:

A	B	C	D	E
1	1	3	3	2

2. Construire model

Primul pas: sortarea (de obicei crescătoare) după frecvenţa de apariţie.

A	B	E	C	D
1	1	2	3	3

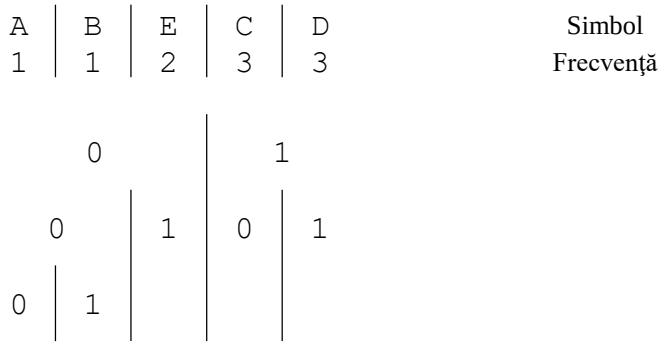
Se împarte apoi mulţimea de simboluri în două submulţimi având suma frecvenţelor cât mai apropiată şi atribuirea codului 0 respectiv 1 celor două ”jumătăţi” (se încearcă astfel atribuirea biţilor 0 şi 1 cu o probabilitate cât mai apropiată de 0.5, cât este optimul teoretic)

.

A	B	E	C	D	
1	1	2	3	3	Frecvenţă
-8	-6	-2	4		Diferenţa în cazul împărţirii în fiecare din poziţii

Diferenţa minimă (în valoare absolută) este dată la împărţirea între E şi C. Se împarte astfel şi se atribuie codul 0 primei “jumătăţi” şi codul 1 celei de a doua “jumătăţi”. Se continuă apoi (de obicei recursiv) asemănător pe fiecare din “jumătăţile” obţinute până se ajunge la simboluri individuale.

În figura următoare prezentăm întregul proces.



Rezultă următoarele coduri:

```
A =  000
B =  001
C =  10
D =  11
E =  01
```

Observăm că avem de a face cu o **abordare top-down** de construcţie a codurilor.

3. Codare pe baza modelului

Se parcurge fluxul (din nou) şi se înlocuieşte fiecare simbol cu codul aferent. Obţinem astfel:

`ABCDECDECD`	şir iniţial
`00000110110110110110011`	flux de date comprimate

Din păcate delimitarea între simboluri se pierde (codurile având lungime variabilă), ceea ce va complica puţin decodarea.

B2. Codificare Huffman Static - exemplu

Să considerăm fluxul: ABCDECDECD

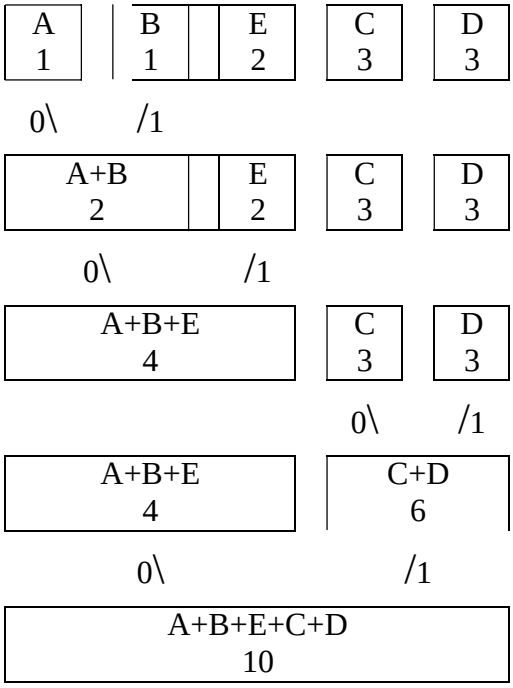
1. Determinare a statisticii

Obţinem (la fel):

A	B	C	D	E
1	1	3	3	2

2. Construire model

În acest caz se grupează la fiecare pas cele două simboluri cu frecvenţa cea mai mica şi se înlocuiesc cu un simbol având frecvenţa egală cu suma acestora. Celor două simboluri care s-au grupat li se atribuie biţii 0 respectiv 1. Se repetă acest procedeu până se grupează toate simbolurile existente.



În final codul este dat de secvenţa de biţi de pe calea spre simbolul iniţial (de la final spre simbol).

Rezultă următoarele coduri:

```
A =  000
B =  001
C =  10
D =  11
E =  01
```

Observăm că avem de a face cu o **abordare down-top** de construcţie a codurilor.

Pentru exemple mici (ca cel de faţă) codurile dau la fel cu cazul Shannon-Fano. Pentru exemple mai mari ele însă vor diferi (teoretic, Huffman este codul optim).

3. Codare pe baza modelului

Se face la fel ca în exemplul de la Shannon-Fano.

C. Indicaţii de implementare

Sunt doar indicaţii, nu sunt strict obligatorii, dar cred că simplifică semnificativ implementarea.

C1. Structura fişierului comprimat

După cum am amintit anterior propun salvarea în fişierul comprimat a informaţiilor legate de statistica apariţiei simbolurilor în fluxul de intrare. Schema generală:

Antet (model)	Flux de date comprimate (şirul de biţi rezultat)

devine în acest caz:

Antet (frecvenţele de apariţie)	Flux de date comprimate (şirul de biţi rezultat)

C2. Decodarea

1. Preluare statistică – se realizează foarte simplu, prin citirea frecvenţelor de apariţie din antet
2. Construire model (coduri) – simplu, se foloseşte aceeaşi funcţie ca la codor (pe baza aceloraşi frecvenţe se va genera acelaşi model).
3. Decodare pe baza modelului. Aici sunt două abordări posibile:

Prima: se citeşte câte un bit din fluxul de date comprimate până se obţine un cod valid. Această soluţie este destul de ineficientă (mai ales când sunt multe simboluri şi codurile sunt lungi).

A doua: se citeşte câte un bit şi se parcurge arborele (modelul). Se pleacă din rădăcină şi se parcurge stânga sau dreapta conform cu bitul următor din flux până se atinge o frunză, obţinând astfel simbolul decodificat. Se reia apoi din rădăcină pentru următorul simbol, etc.

C3. Problema sfârşitului fluxului

Deoarece în fluxul de biţi comprimaţi exista în final unii biţi adăugaţi în ultimul octet (maxim 7) care nu trebuie decodificaţi oprirea decodificării este o problemă. Deşi există mai multe soluţii posibile precum adăugarea unor informaţii suplimentare în antet (ex: lungimea fişierului, numărul de biţi adăugaţi în ultimul antet) sau folosirea în model a unui simbol special de tip end-of-file EOF propun soluţia simplă bazată pe cunoaşterea lungimii fişierului, lungime care poate fi determinată uşor ca sumă a frecvenţelor de apariţie existente în antet.

C4. Realizarea unei singure aplicaţii codor-decodor

Deşi cele două părţi codor respectiv decodor funcţionează complet independent propun realizarea lor într-o singură aplicaţie. În acest fel aplicaţia se poate realiza şi depana mai uşor (de exemplu construirea modelului fiind una şi aceeaşi). Atenţie însă să nu avem influenţe între codor şi decodor altele decât prin intermediul fişierului comprimat.

C5. Colectarea statisticii

Cea mai simplă soluţie este aceea de a contoriza apariţia fiecărui octet din fişierul de intrare într-un contor corespunzător astfel:

```
Int32 contoare[256] //initializat cu 0
```

Apoi, pentru fiecare octet din fişier, să avem:

```
c = citeste octet din fisier
contoare[c]++;
```

C6. Variante de antet

În varianta discutată până acum în antet trebuie **transmise toate cele 256 contoare** a câte 4 octeţi.

contoare[0]	contoare[1]	. . .	contoare[255]	Flux de date comprimate (şirul de biţi rezultat)

În acest caz antetul va ocupa 1024 octeţi. Pentru fluxuri mari acest lucru nu este semnificativ, însă pentru fluxuri mici poate fi semnificativ. Soluţia prezentată este **satisfăcătoare** pentru implementarea acestei teme.

Putem însă lua în considerare următoarele variaţii:

- Includerea în începutul antetului a unei **hărţi de alocare** a contoarelor (de 256 biţi) astfel:

101 . . .	contoare[0]	contoare[2]	. . .	Flux de date comprimate (şirul de biţi rezultat)

Fiecare bit din harta de alocare indică dacă contorul respectiv este nenul şi merită pus în antet. În acest caz antetul ocupă 32 octeţi pentru harta de alocare şi un număr variabil de octeţi pentru contoare. Dacă cel puţin 8 simboluri nu există în flux soluţia duce la câştig. Dacă nu, nu :)

Studentii de la ISM vor implementa antetul 2

- O altă idee este aceea de a avea **harta de alocare pe câte 2 biţi** cu următoarea convenţie (de exemplu):

  00 contorul nu se reprezintă (este nul)
  01 contorul se reprezintă pe 1 octet
  10 contorul se reprezintă pe 2 octeţi
  11 contorul se reprezintă pe 4 octeţi

Studentii C/TI vor implementa antetul 3

În acest caz harta de alocare ocupă 64 octeţi (pierdere) dar se speră ca câştigul care apare din zona contoarelor să compenseze acest lucru.

C7. Construirea arborelui

La **Shannon-Fano**, în procedura (recursivă) de împărţire a intervalului, în momentul împărţirii se adaugă şi doi fii nodului curent. În acest fel la terminarea împărţirilor avem gata şi arborele corespunzător.

La **Huffman static** se construieşte un listă (sau vector) de arbori, iniţial fiecare arbore având doar o frunză (ca rădăcină). Apoi, la fiecare pas, se sortează lista, se scot cei 2 arbori cu contorul minim, se construieşte un arbore având cei doi arbori scoşi anterior ca fii (şi contorul suma) şi se introduce înapoi în listă arborele tocmai construit. Astfel, la fiecare pas, lista scade cu un arbore. În final avem un singur arbore în listă, tocmai arborele Huffman căutat.

C8. Codare pe baza arborelui

În cazul în care facem codarea direct pe baza arborelui trebuie avut în vedere că, prin parcurgerea căii din frunză spre rădăcină, se obţine codul frunzei în ordinea inversă a biţilor. Deci, acest şir de biţi trebuie inversat înainte de a fi scris în fişierul comprimat.

Pentru a ajunge însă uşor în frunza corespunzătoare unui simbol se recomandă întreţinerea unui vector de referinţe (pointeri) spre fiecare frunză (vector indexat cu simbolul).

C9. Sugestii interfata

Codor

- Buton “Load file” – cand se va apasa se va deschide case standard the OpenFileDialog si se va putea selecta orice tip de fisier de pe HDD
- Buton “Encode file”– la apasarea acestui buton se va encode fisierul selectat anterior(fie cu codificare Huffman fie cu codificare Shannon).Fisierul encoded va fii salvat in acelasi director cu fisierul original. Filename-ul fisierul encoded va fii identic cu numele fisierul original la care se va adauga extensia .hs (huffman static) sau .sf (Shannon fano). E.x. daca fisierul input se numeste “poza mea.jpeg” -> fisierul encoded -> “poza mea.jpeg.hs”
- Text input – multiline text area unde se va putea introduce free text (Ascii or Unicode format)
- Buton “Encode input text”. La apasarea acestui buton se va encode textul introdus in (fie cu codificare Huffman fie cu codificare Shannon). Numele fisierul generat va fii InputArea.hs sau InputArea.sf (depinzand de algoritmul implementat)
- Show codes – checkbox (va exista atata la coder cat si la decoder functionalitatea). Daca checkbox-ul este checked – in momentul cand se va apasa pe butonul de “Encode file” sau “Encode input text” sau “Decode file” – se va afisa in list box-ul afferent checkbox-ului care este codificarea fiecarui symbol. Per exemplu dat anterior A – 001

Decodor

- Buton “Load encoded file” – cand se va apasa se va deschide case standard the OpenFileDialog si se va putea selecta orice fisier cu extensia .HS sau .SF
- Buton “Decode file” – la apasarea acestui buton se va decoda fisierul selectat anterior.Fisierul decoded va fii salvat in acelasi director cu fisierul selectat anterior. Filename-ul fisierul decoded va fii identic cu numele fisierul selectat la care se va adauga un timestamp + extensia originala. E.x. daca fisierul input se numeste “poza mea.jpeg” -> fisierul encoded -> “poza mea.jpeg.hs”
  ➔ fisierul decoded se va nume “poza mea.jpeg.hs**.13-10-2020-14-54.jpeg”**