Lucrarea 4

Procedeul de codare Huffman

4.1 Obiectivul lucrării

În această lucrare este studiată codarea surselor de informaţie discrete pentru canale fără perturbaţii prin procedeul Huffman. Vor fi analizate codurile Huffman binare şi n-are şi se va studia, de asemenea, aplicarea algoritmului Huffman pentru compresia datelor.

4.2 Introducere teoretică

Algoritmul de codare Huffman este un algoritm optimal de codare a surselor discrete de informaţie, în sensul că prin niciun alt algoritm nu se poate obţine o lungime a cuvintelor de cod mai mică. Comparativ cu algoritmul Shannon-Fano, util numai pentru codarea surselor care generează simboluri ale căror probabilităţi sunt puteri negative ale lui 2, procedeul lui Huffman este aplicabil pentru orice distribuţie de probabilităţi ale simbolurilor sursei de informaţie. El permite obţinerea unui cod ale cărui cuvinte au lungimi variabile, fără prefix. Proprietatea codurilor fără prefix este aceea că niciun cuvânt de cod nu este un prefix pentru un alt cuvânt de cod.

Un cod fără prefix este un cod instantaneu, în cazul căruia decodarea unui cuvânt se efectuează pe măsura recepţionării lui, fără a fi necesară examinarea cuvântului următor.

În cazul codării surselor pentru canale fără perturbaţii, sursa de informaţie primară generează un număr de simboluri sau mesaje care alcătuiesc alfabetul sursei a cărui dimensiune este N:

$$[\mathbf{S}] = \left[\begin{array}{cccc} s_1 & s_2 & \dots & s_N \end{array} \right]. \tag{4.1}$$

Aceste mesaje s_i sunt puse în corespondenţă biunivocă cu tot atâtea cuvinte de cod c_i având lungimile l_i. Lungimea l_i a unui cuvânt de cod c_i este dată de numărul de simboluri x_i din structura sa. Pentru construcţia acestor cuvinte de cod se folosesc literele x_i ale unei surse secundare, care aparţin alfabetului ce conţine D litere:

$$[\mathbf{X}] = \left[\begin{array}{cccc} x_1 & x_2 & \dots & x_D \end{array} \right] \tag{4.2}$$

cu probabilităţile $P[\mathbf{X}] = \left[\begin{array}{cccc} p(x_1) & p(x_2) & \dots & p(x_D) \end{array} \right]$.

Pentru a evalua performanţele codului

$$[\mathbf{C}] = \left[\begin{array}{cccc} c_1 & c_2 & \dots & c_N \end{array} \right] \tag{4.3}$$

se defineşte lungimea medie a cuvintelor de cod:

$$\bar{l} = \sum_{i=1}^{N} p(s_i) \cdot l_i. \tag{4.4}$$

Valoarea minimă a acestei mărimi, pentru o distribuţie dată a probabilităţilor sursei primare, se obţine atunci când probabilităţile simbolurilor sursei secundare sunt egale între ele; ea este dată de relaţia:

$$\bar{l}_{\min} = \frac{H(\mathbf{S})}{\log_2 D}. \tag{4.5}$$

unde $H(\mathbf{S})$ reprezintă entropia sursei, calculată cu formula:

$$H(\mathbf{S}) = -\sum_{i=1}^{N} p(s_i) \log_2 p(s_i). \tag{4.6}$$

Este de dorit ca lungimea medie $\bar{l}$ să fie cât mai apropiată de $\bar{l}_{min}$ şi, în consecinţă, eficienţa unui cod este definită ca:

$$\eta = \frac{\bar{l}_{\min}}{\bar{l}}. \tag{4.7}$$

Atunci când eficienţa unui cod are valoarea 1, se spune că acesta este absolut optimal, iar dacă eficienţa sa are valoarea maximă posibilă, dar este subunitară, se spune că este optimal.

În cazul general, al unei distribuţii oarecare de probabilităţi ale simbolurilor sursei primare, prin procedeul Huffman se obţine un cod optimal, iar dacă aceste probabilităţi sunt puteri negative ale lui 2, se obţine un cod absolut optimal.

Redundanţa unui cod este dată de relaţia:

$$\rho = 1 - \eta. \tag{4.8}$$

Pentru ca un cod fără prefix să existe, trebuie să fie îndeplinită inegalitatea Kraft-McMillan:

$$K = \sum_{i=1}^{N} D^{-l_i} \leq 1. \tag{4.9}$$

Această relaţie exprimă condiţia suficientă pentru un cod fără prefix şi condiţia necesară pentru un cod unic decodabil.

Dacă inegalitatea nu este indeplinită, atunci codul nu este unic decodabil.

Un cod optimal fără prefix are următoarele proprietăţi:

1. Simbolurile cu probabilitatea cea mai mare vor corespunde cuvintelor de cod cele mai scurte. Pentru $p(s_i) > p(s_j)$, rezultă $l_i \leq l_j$.

2. Ultimele D simboluri cu cea mai mică probabilitate vor corespunde unor cuvinte de cod de lungime egală.

Codul Huffman se creează după următorul algoritm:

1. Se ordonează simbolurile sursei primare în ordine descrescătoare a probabilităţilor: $p(s_1) \geq p(s_2) \geq \ldots$

2. Ultimele D simboluri (unde D este dimensiunea alfabetului codului) cu cele mai mici probabilităţi se grupează într-un nou simbol a cărui probabilitate este egală cu suma probabilităţilor simbolurilor grupate: $p(r_1) = \sum_{i=1}^{D} p(s_i)$. Se obţine o sursă restrânsă cu $N - D + 1$ simboluri.

3. Se repetă paşii 1 şi 2 pentru sursele restrânse astfel obţinute, până când se obţine o sursă restrânsă cu D simboluri.

4. Cuvântul de cod pentru fiecare simbol este obţinut prin parcurgerea schemei de codare, în sens opus restrângerii (echivalent cu parcurgerea unui arbore de la nodul rădăcină către frunze). La fiecare ramificație se va adăuga o literă din cele D litere ce compun alfabetul codului.

Exemplu: Se consideră o sursă discretă de informaţie cu următoarea distribuţie de probabilităţi:

$$P(\mathbf{S}) = \left[\begin{array}{cccccc} 0.30 & 0.10 & 0.05 & 0.25 & 0.20 & 0.10 \end{array} \right].$$

Sursa va fi codată cu ajutorul algoritmului Huffman binar ($D = 2$) și Huffman ternar ($D = 3$).

a. **Codarea binară** ($D = 2$)

Schema de codare și codul obținut sunt prezentate în Fig. 4.1 și în Tabelul 4.1.

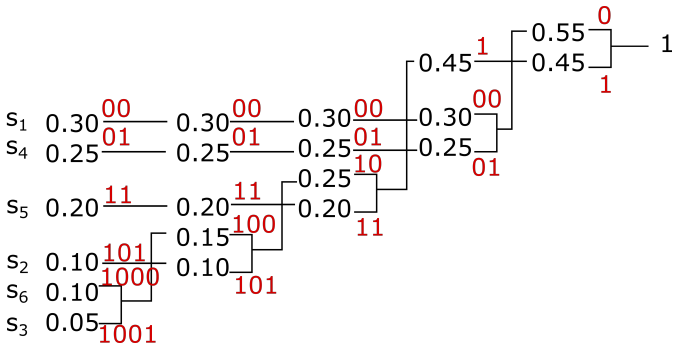

Figura 4.1: Procedeul de codare Huffman folosind un alfabet binar

Simbol	Probabilitate	Cuvânt de cod	Lungimea cuvântului de cod
s_1	0.30	00	2
s_2	0.10	101	3
s_3	0.05	1001	4
s_4	0.25	01	2
s_5	0.20	11	2
s_6	0.10	1000	4

Tabelul 4.1: Codul Huffman binar pentru sursa S

Entropia sursei este calculată cu ajutorul relaţiei (4.6):

$$H(\mathbf{S}) = 2.37 \quad \text{[biti/simbol]}.$$

Lungimea medie a cuvintelor de cod, calculată cu formula (4.4) este:

$$\bar{l} = 2.4,$$

iar lungimea medie minimă, calculată cu relaţia (4.5) este:

$$\bar{l}_{min} = 2.37.$$

În aceste condiţii, eficienţa codului este, conform relaţiei (4.7):

$$\eta = 0.9875.$$

În acest caz, relaţia Kraft-McMillan (4.9) este îndeplinită cu egalitate:

$$K = 1.$$

b. **Codarea ternară** ($D = 3$)

Pentru a realiza codarea ternară şi, în general, codarea n-ară, trebuie ca ultima sursă restrânsă să genereze exact D simboluri. După n paşi de restrângere, o sursă va genera $N - nD + n$ simboluri.

Dacă după n restrângeri, sursa trebuie să genereze D simboluri, atunci trebuie să fie îndeplinită următoarea relaţie:

$$D = (N - nD) + n,$$

de unde rezultă următoarea relaţie:

$$n = \frac{N - D}{D - 1}. \tag{4.10}$$

În mod evident, numărul de restrângeri este întotdeauna natural ($n \in \mathbb{N}$).

Pentru a îndeplini această condiţie, în cazul codării n-are poate fi necesară adăugarea de simboluri artificiale sursei primare, astfel încât să fie îndeplinită această condiţie.

Pentru sursa propusă, se cunoaşte $N = 6, D = 3$. Atunci:

$$n = \frac{6 - 3}{3 - 1} = \frac{3}{2} \notin \mathbb{N}.$$

Se va adăuga un simbol artificial cu probabilitatea 0, astfel încât $N' = 7$, şi atunci:

$$n = \frac{7 - 3}{3 - 1} = 2 \in \mathbb{N}.$$

Schema de codare și codul obținut sunt prezentate în Fig. 4.2 și în Tabelul 4.2.

Pentru a analiza performanţele codului, se vor calcula aceleaşi mărimi ca în cazul codării binare.

Entropia sursei primare rămâne neschimbată, ea depinzând doar de probabilităţile simbolurilor sursei:

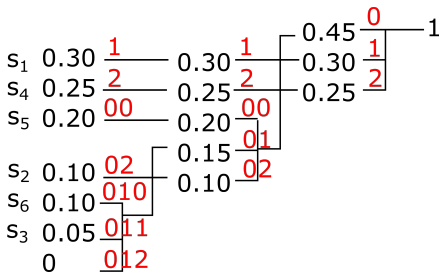

Figura 4.2: Procedeul de codare Huffman folosind un alfabet ternar

Simbol	**Probabilitate**	**Cuvânt de cod**	**Lungimea cuvântului de cod**
s_1	0.30	1	1
s_2	0.10	02	2
s_3	0.05	011	3
s_4	0.25	2	1
s_5	0.20	00	2
s_6	0.10	010	3

Tabelul 4.2: Codul Huffman ternar pentru sursa S

$$H(\mathbf{S}) = 2.37 \quad \text{[biti/simbol]}.$$

Lungimea medie a cuvintelor de cod, calculată cu formula (4.4), va fi:

$$\bar{l} = 1.6,$$

iar lungimea medie minimă, calculată cu relaţia (4.5) este:

$$\bar{l}_{min} = 1.49.$$

Eficienţa codului, calculată conform relaţiei (4.7) este:

$$\eta = 0.93.$$

Din relaţia (4.9) rezultă că ecuaţia Kraft-McMillan este egală cu:

$$K = 0.96.$$

Pentru codarea ternară, ecuaţia Kraft-McMillan este indeplinită cu inegaliate, ceea ce înseamnă că acest cod are redundanţă.

4.3 Desfăşurarea lucrării

În continuare va fi prezentat un algoritm de codare Huffman binară implementat în Python folosind cozi cu prioritate (engl. priority queues).

```
import heapq

class Node:
    def __init__(self, p, s):
        self.left = None
        self.right = None
        self.prob = p
        self.symbol = s

    def __lt__(self, other):
        return self.prob < other.prob

    def __repr__(self):
        return "Node({}, {}, {})".format(repr([self.prob, self.symbol]), ...
            repr(self.left), repr(self.right))

def HuffmanTree(SP):
    pq = []
    for symbol, prob in SP.items():
        pq.append(Node(prob, symbol))
    heapq.heapify(pq)

    while len(pq) > 1:
        n1 = heapq.heappop(pq)
        n2 = heapq.heappop(pq)
        top = Node(n1.prob+n2.prob, n1.symbol+n2.symbol)
        top.left = n1
        top.right = n2
        heapq.heappush(pq, top)
        # print(n1)
        # print(n2)
        print(pq, '\n')
    return pq


def encode(dic_code, root, code):
    if root.left is None and root.right is None:
        dic_code[root.symbol] = code
    else:
        encode(dic_code, root.left, code+'0')
        encode(dic_code, root.right, code+'1')

S = ['s1', 's2', 's3', 's4', 's5', 's6']
P = [0.30, 0.10, 0.05, 0.25, 0.20, 0.10]
SP = dict((S[i], P[i]) for i in range(len(S)))
dic_code = dict((S[i], '') for i in range(len(S)))

PQ = HuffmanTree(SP)
encode(dic_code=dic_code, root=PQ[0], code='')
print(dic_code)
```

Rezolvarea problemei propuse folosește *heapq*, un modul Python ce permite implementarea structurilor de date *heap*, ce permit efectuarea unor operații de bază asupra cozilor cu prioritate. Structura *heap* este o structură de date în care fiecare cheie este mai mare decât cheile de pe două poziții determinate, adică:

$$\begin{aligned} heap[k] &<= heap[2*k+1] \\ heap[k] &<= heap[2*k+2] \end{aligned}$$

pentru orice k pornind de la 0. O proprietate interesantă a unei structuri *heap* este faptul că cel mai mic element este întotdeauna *heap*[0].

În cazul algoritmului de codare Huffman binar prezentat mai sus, fiecare pas al implementării presupune (1) extragerea din *heap* a două noduri ce au probabilitățile cele mai mici (folosind metoda $heappop(heap)$) (2) gruparea lor într-un nod restrâns având probabilitatea egală cu suma celor două probabilități corespunzătoare nodurilor extrase, și (3) inserarea nodului restrâns înapoi în coada cu priorități (folosind metoda $heappush(heap, item)$) pentru a participa la pașii următori de codare. Restrângerea se finalizează în momentul în care în coadă rămâne un singur simbol restrâns. În Python, scrierea $n1.symbol + n2.symbol$ reprezintă concatenarea a două elemente de tip *string*. Pentru a înțelege modul de creare al arborelui, se poate urmări crearea sa folosind instrucțiunea *print*. În ceea ce privește nodul, acesta conține în structura sa atât simbolul, cât și probabilitate asociată. Metoda $\_\_lt\_\_(self, other)$ este suprascrisă pentru a permite compararea directă a nodurilor prin valorile probabilităților aferente. Metoda $\_\_repr\_\_(self)$ întoarce reprezentarea obiectului într-o anumită formă, de obicei un string ce conține o reprezentare a obiectului ce poate fi afișată în consolă.

Funcția *encode* începe codarea de la rădăcina arborelui ce are asociat drept cod un string gol ' '. Pornind de la rădăcină către frunzele arborelui, prin frunze înțelegând acele noduri ce nu au descendenți, se va aloca câte un bit de 0 și 1 pe fiecare ramură. În final, codul asociat sursei va fi reținut în *dic_code*.

4.4 Exerciţii

Pornind de la algoritmul prezentat în platformă, rezolvați următoarele exerciții.

1. Implementați funcția:

   p1(S: List[str], P: List[float]) -> List[float]

   Funcția primește ca parametri:

   - S - o listă de simboluri
   - P - o listă de probabilități pentru fiecare simbol.

   Se garantează:

   - toate simbolurile sunt diferite între ele;
   - cele 2 liste au aceeași lungime;
   - suma tuturor probabilităților este 1.

   Funcția codează simbolurile folosind codarea Huffman binară și returnează, într-o listă, următoarele 3 valori în această ordine:

   - Lungimea medie a cuvântului de cod (calculată după formula (4.4));
   - Eficiența codului (calculată după formula (4.7));
   - Valoarea lui K pentru inegalitatea Kraft-McMillan (calculată după formula (4.9)).

2. Implementați funcția:

   p2(S: List[str], P: List[float]) -> float

   Funcția primește ca parametri:

- S - o listă de simboluri.
- P - o listă de probabilități pentru fiecare simbol.

Se garantează:

- toate simbolurile sunt diferite între ele;
- cele 2 liste au aceeași lungime;
- suma tuturor probabilităților este 1.

Funcția codează simbolurile folosind codarea Huffman ternară și returnează lungimea medie a cuvântului de cod (calculată după formula (4.4)).
Sfaturi:

- Modificați clasa Node astfel încât să permită 3 copii (descendenți) în loc de 2.
- Modificați funcția HuffmanTree astfel încât să țină cont de relația (4.10) și să adauge un nou simbol (' ') cu probabilitatea 0.0 atunci când relația nu este îndeplinită.
- Modificați funcția HuffmanTree astfel încât să realizeze suma ultimelor 3 noduri cu cele mai mici probabilitați în loc de suma ultimelor 2.
- Modificați funcția encode astfel încât să țină cont că un nod poate avea 3 copii (descendenți) în loc de 2.
- Modificați funcția encode astfel încât să țină cont că poate exista și simbolul '2' pentru codare.

3. Implementați funcția:

p4(text: str) -> List[int]

Funcția primește ca parametru un text.
Funcția returnează o listă de 2 întregi: numărul de biți necesari codării întregului text folosind codarea Huffman binară precum și numărul de biți necesari codării întregului text dacă fiecare caracter se codează pe $\lceil \log_2(n) \rceil$ biți.