STRUCTURI DE DATE

TEMA 2

Dată publicare: 19 Aprilie 2015

Deadline: 10 Mai 2015

Sevastian Emma
Facultatea de Automatică şi Calculatoare

1. Contextul cerinţei

1.1 Compresia este procesul de minimizare a spațiului ocupat sau a timpului necesar transmiterii unei anumite cantități de informație.

Termenul de comprimare a datelor a apărut în contextul în care se manifesta o necesitate evidentă de a atinge rate mari de transfer în reţele sau de a stoca o cantitate cât mai mare de informaţii folosind cât mai puțin spaţiu.

Ca o formă primitivă a compresiei de date putem considera prescurtările din viața de zi cu zi. Compresia de date fără pierdere, prezentă în programele de arhivare, în sistemele de transmisie a datelor, a evoluat de-a lungul timpului pornind de la algoritmi simpli (suprimarea zerourilor, codarea pe şiruri) şi ajungând la algoritmii complecşi folosiți în prezent.

1.2 Codificare Huffman

Codificarea Huffman se bazează pe frecvenţa de apariţie a simbolurilor. Ideea algoritmului este de a ataşa coduri de lungime mică simbolurilor cu frecvenţă mare, iar celor cu frecvenţă mică, coduri de lungime mai mare.

Algoritmul de codificare Huffman presupune că simbolurile ce doresc a fi codificate fac parte dintr-o mulţime finită şi au asociată o probabilitate de apariţie. Pentru exemplul de faţă vom considera cazul şirurilor de caractere şi vom explica algoritmul pe baza acestui exemplu.

“ana are mere”

mulţimea simbolurilor S = {‘a’, ‘n’, ‘ ‘, ‘r’, ‘m’, ‘e’}
lungimea şirului iniţial = 12

Asociem fiecărui caracter probabilitatea de apariţie calculată ca: numărul de repetiţii a caracterului respectiv în şirul iniţial / lungimea şirului.

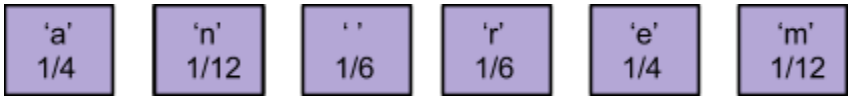
Simbol	Probabilitate
‘a’	1/4
‘n’	1/12
‘ ‘	1/6
‘r’	1/6
‘e’	1/4
‘m’	1/12

Bazându-ne pe frecvenţa de apariţie a fiecărui caracter vom crea noua codificare asociată fiecărui caracter.

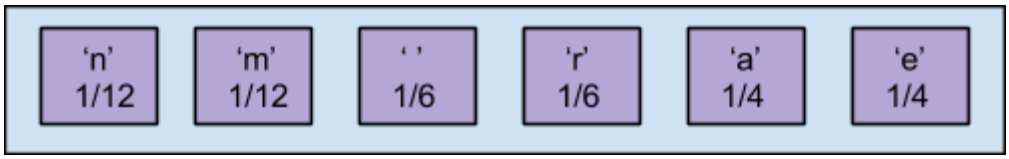
Crearea arborelui Huffman

Arborele Huffman este un arbore binar ce conţine în fiecare frunză un simbol/caracter alături de probabilitatea sa de apariţie. Nodurile interne conţin doar un număr egal cu suma probabilităţilor de apariţie ale descendenţilor direcţi. Plecând de la aceste reguli, rădăcina va avea asociat numarul 1. Suma probabilitaţilor de apariţie ale tuturor simbolurilor/caracterelor este 1.

Paşi pentru construirea arborelui:
1. se construieşte câte un nod frunză pentru fiecare simbol/caracter. Acest nod va conţine simbolul şi probabilitatea de apariţie asociată.



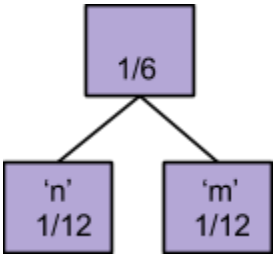
2. nodurile create anterior se adaugă într-o coadă cu priorităţi, sortată crescător după probabilităţi, astfel încât primele care vor fi extrase vor fi cele cu probabilitatea cea mai mică.



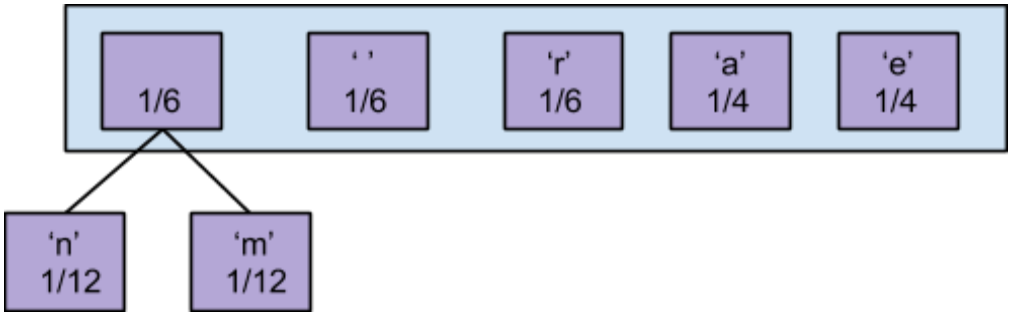
3. cât timp coada conţine cel puţin 2 elemente:

3.1 se extrag primele 2 elemente, cele cu frecvenţa de apariţie cea mai mică, şi se creează un nod nou, cu probabilitatea egală cu suma celor două noduri.

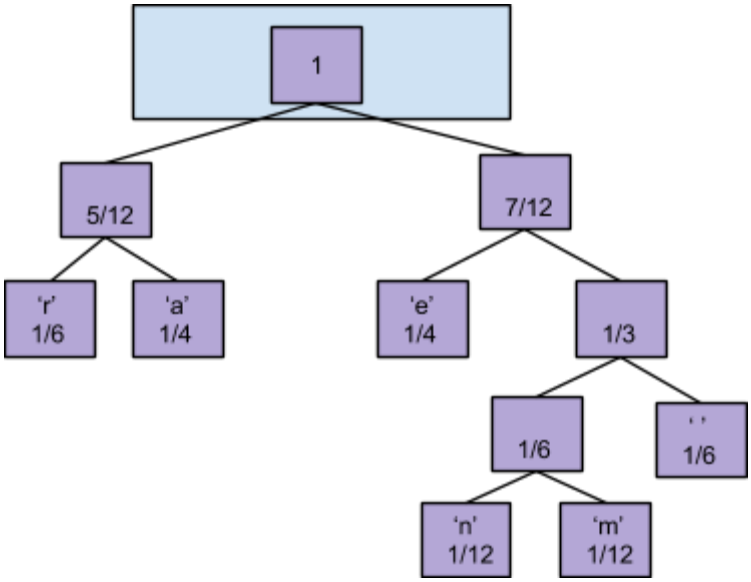
3.2 noul nod creat va conţine cele 2 noduri ca şi descendent stâng, respectiv drept.



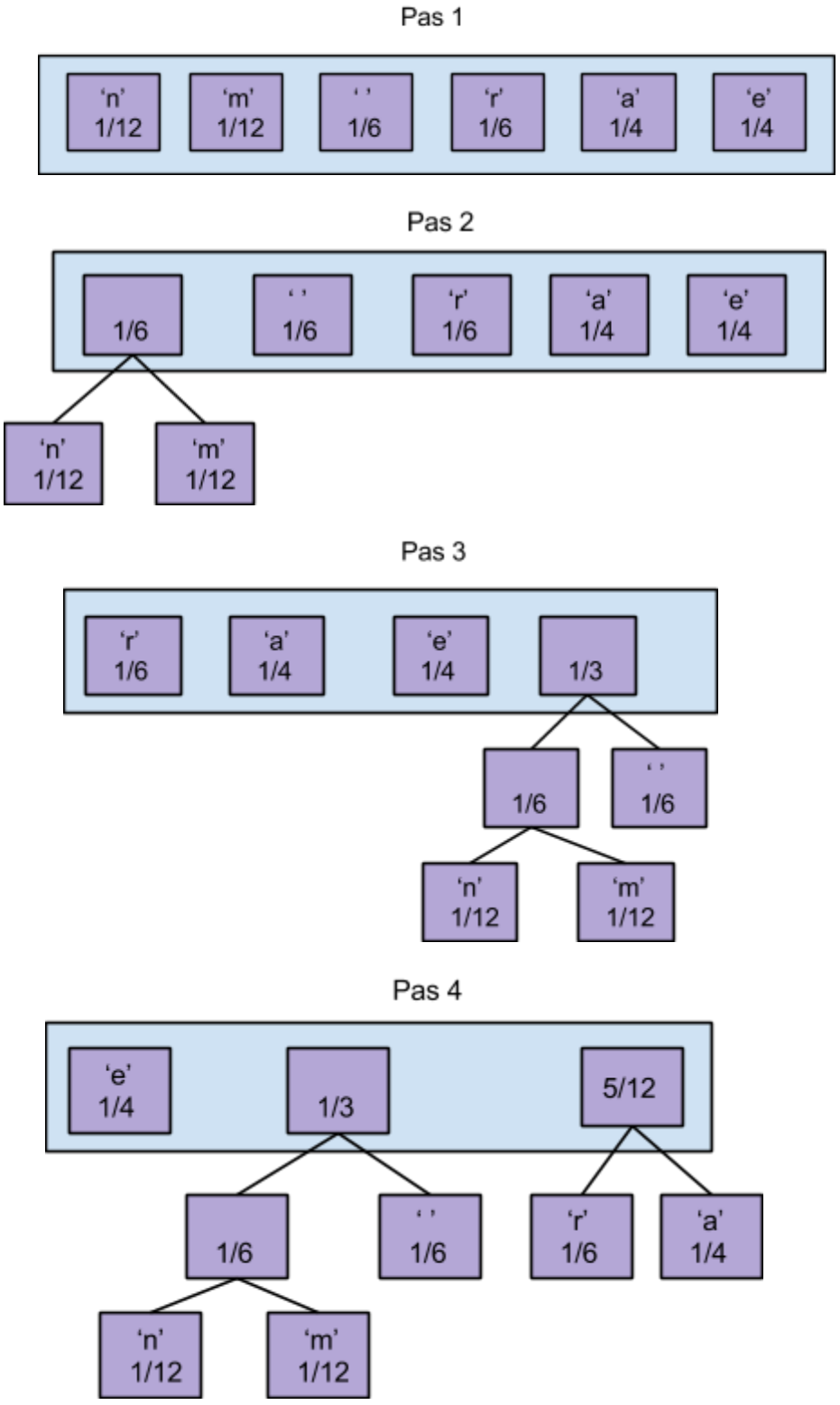
3.3 noul nod se inserează înapoi în coadă, cu păstrarea ordinii crescătoare după probabilităţi



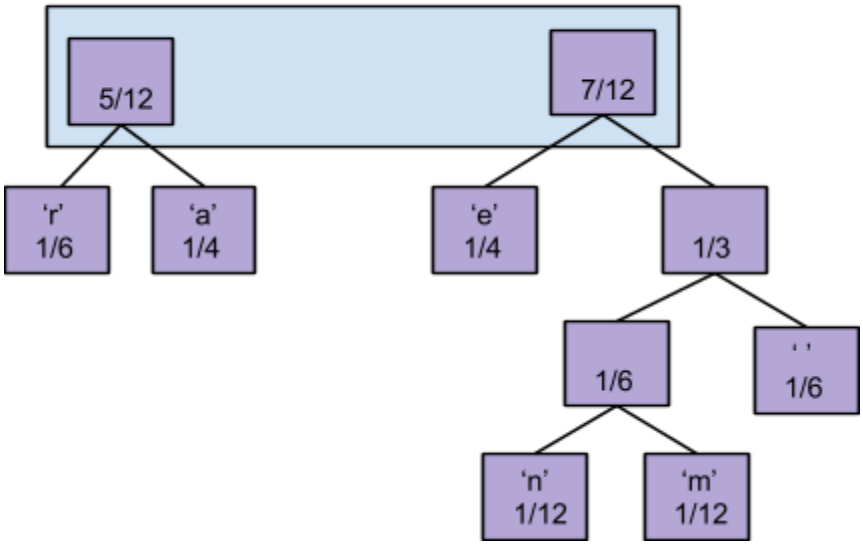
4. la final coada va conţine un singur element: rădăcina arborelui Huffman



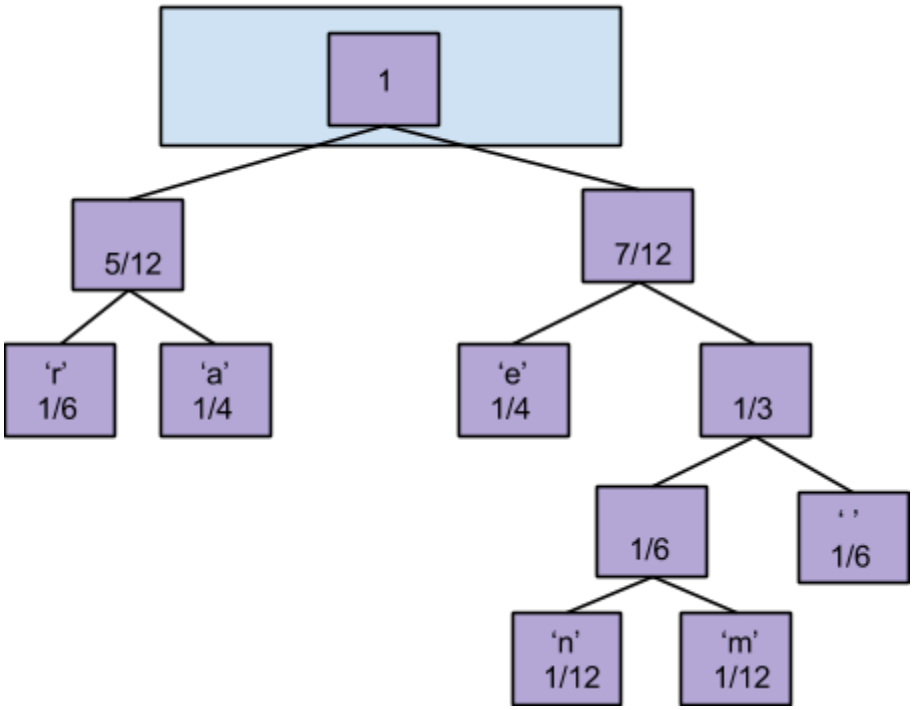
Procesul de construcţie al arborelui ilustrat pe exemplul de mai sus



Pas 5

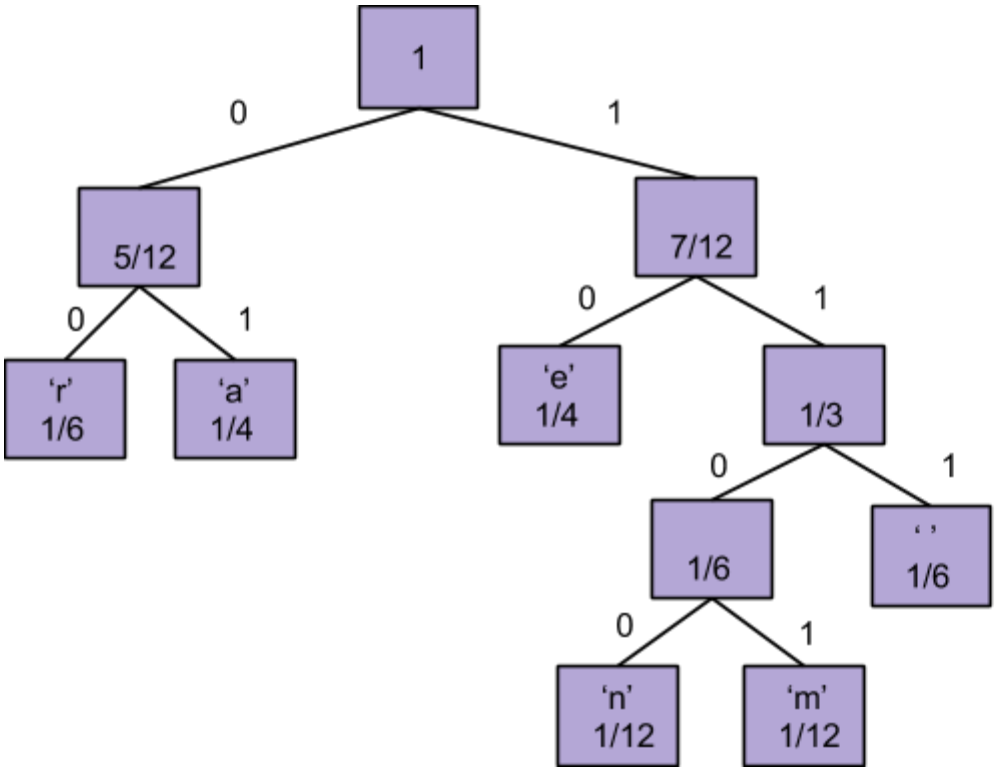

5/12
7/12
'r'
1/6
'a'
1/4
'e'
1/4
1/3
1/6
' '
1/6
'n'
1/12
'm'
1/12

Pas 6


1
5/12
7/12
'r'
1/6
'a'
1/4
'e'
1/4
1/3
1/6
' '
1/6
'n'
1/12
'm'
1/12

Codificarea caracterelor

Plecând de la rădăcina arborelui Huffman se poate genera codul fiecărui simbol/caracter astfel: la fiecare pas al parcurgerii, pentru o alegere a nodului stâng se adaugă un 0, pentru o alegere a nodului drept, se adaugă un 1.



Pe baza arborelui creat se generează următoarea codificare a caracterelor:

Simbol	**Codificare**
'r'	00
'a'	01
'e'	10
' '	111
'n'	1100
'm'	1101

După cum se poate observa, caracterele care au o frecvenţă mai mare în şirul iniţial au o lungime mai mică a codului asociat.

Şirul iniţial devine:

‘a’	‘n’	‘a’	‘ ‘	‘a’	‘r’	‘e’	‘ ‘	‘m’	‘e’	‘r’	‘e’
01	1100	01	111	01	00	10	111	1101	10	00	10

Codul ASCII asociat şirului iniţial are nevoie de 12 * 8 = 96 biţi pentru reprezentare, caracterele ASCII fiind reprezentate pe câte un octet (1 octet = 8 biţi). Pentru reţinerea noului şir este nevoie de 30 de biţi. Pentru a putea fi reprezentat în memoria calculatorului însă, este nevoie de un număr de biţi divizibil cu 8, astfel se vor adăuga 2 biţi ajungându-se la 32.

2. Cerinţă şi punctaj

Cerinţa temei este de a realiza codificarea unui fişier folosind codificarea Huffman. Se consideră că fişierele folosesc encodare extended ASCII (256 de caractere). Detalii la [1], [2].

Format fişier comprimat

1. Fişierul comprimat va conţine la început detaliile caracterelor care au apărut în fişierul iniţial astfel:

- numar_caractere de tip `uint32_t` ce specifică numărul de caractere din fişier
- numar_noduri de tip `uint16_t` ce specifică numărul de noduri ale arborelui Huffman creat
- numar_noduri structuri de tipul

```
typedef struct TagHuffmanNode {
      unsigned char value;
      int16_t left;
      int16_t right;
} __attribute__((__packed__)) TagHuffmanNode;
```

Detalii despre __attribute__ găsiţi la [3].

Fiecare element de tip `TagHuffmanNode` va reprezenta un nod al arborelui creat.
Detalii câmpuri structură:

- **value**: reprezintă caracterul asociat nodului respectiv (relevant doar în cazul frunzelor)
- **left**: reprezintă indicele în vectorul de structuri a nodului stâng
- **right**: reprezintă indicele în vectorul de structuri a nodului drept

Detalii noduri:

- dacă **nodul este frunză** el nu are descendenţi, caz în care left şi right vor fi setaţi la -1, iar value va avea valoarea caracterului din nodul respectiv.
- dacă **nodul este unul interior**, el are descendenţi, şi nu are un caracter asociat. left si right vor reprezenta indicii nodului stâng, respectiv drept din vectorul de structuri.

Numerotarea indicilor incepe de la 0, pe prima pozitie reţinându-se rădăcina arborelui Huffman.

2. Urmează datele efective, cu simbolurile scrise codificat în ordinea în care au fost citite din fişierul iniţial (în urma aplicării arborelui Huffman asupra simbolurilor, se obține un şir de biți). În cazul în care lungimea codificării nu este divizibilă cu 8, se va adăuga un padding de biţi de 0 până la cea mai apropiată valoare divizibilă cu 8.

Codificarea se va face pe fiecare octet de la bitul cel mai semnificativ spre cel nesemnificativ.

Bonus: implementarea decomprimării.

Detalii despre paşii urmaţi în cazul comprimării şi al decomprimării îi găsiţi în secţiunea Hints.

Punctaj:

- 80 puncte pentru implementarea corectă şi completă a comprimării
- 10 puncte coding-style
- 10 puncte README
- 20 puncte pentru implementarea corectă şi completă a decomprimării

Programul vostru va primi la intrare numele fişierului de intrare şi cel de ieşire şi o optiune în felul următor:

./huffman -c | -d nume_fisier_intrare nume_fisier_iesire
unde

-c indică faptul că programul va realiza o compresie

-d indică faptul că programul va realiza o decompresie

nume_fisier_intrare reprezintă numele fişierul sursă, asupra căruia se va opera cu compresia sau cu decompresia

nume_fisier_iesire reprezintă fişierul destinaţie, în care se va scrie, în funcţie de comanda dată, fişierul comprimat sau decomprimat.

3. Format arhivă şi testare

Nu se vor folosi biblioteci cu cozi, arbori deja implementate.
Se vor depuncta pierderile de memorie.

Temele vor fi testate automat pe vmchecker. Acesta suportă temele rezolvate în C sau C++.

Arhiva cu rezolvarea temei trebuie să fie .zip şi să conţină:

- fişierul / fişierele sursă
- Makefile
- README - va contine descrierea solutiei. Absenţa sau descrierea sumară poate duce la depunctari de până la 1 punct.

Fişierul pentru make trebuie denumit obligatoriu Makefile şi să conţină (cel puţin) următoarele reguli:

- build, care va compila sursele şi va obţine **executabilul care se va numi huffman**
- clean

Temele se vor puncta doar pe vmchecker.

4. Hints şi cerinţe

1. Se garantează că fişierele de intrare furnizate pentru compresie sau decompresie sunt corecte.
2. Se garantează că fişierele de intrare au cel puţin 2 caractere distincte.
3. Paşi pentru realizarea compresiei:

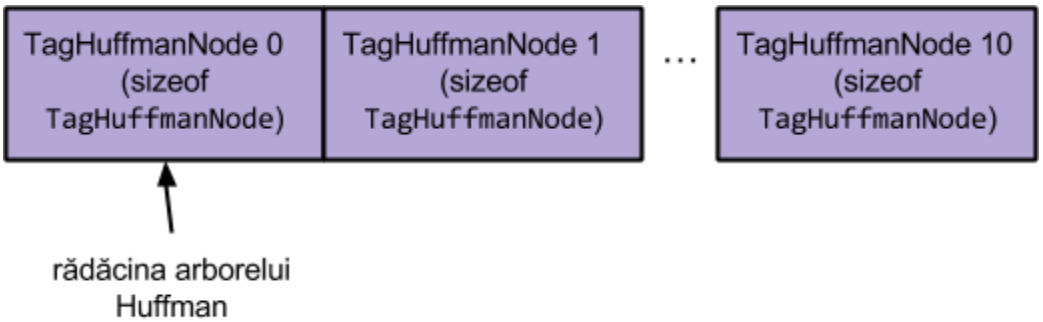
3.1 se calculează frecvenţa de apariţie a fiecărui caracter
se reţine numărul total de caractere din fişier
frecvenţa de apariţie a unui caracter este data de $n_a(c) / n_t$ unde
$n_a(c)$= numarul de apariţii a caracterului c, iar n_t = numarul total de caractere din fişier.

3.2 arborele Huffman se creează urmând specificaţiile prezentate mai sus.
3.3 folosind arborele creat, vom asocia fiecărui caracter un şir de biţi.
3.4 scrierea în fişier

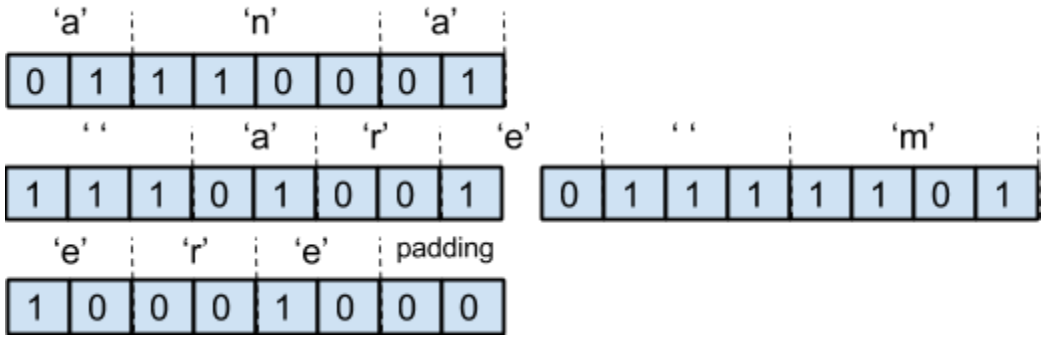
4. Exemplu format fişier comprimat ilustrat pentru exemplul de mai sus:

- iniţial fişierul va conţine numărul de caractere din fişier (12) urmat de numărul de noduri ale arborelui Huffman creat (11)

- urmat de descrierea arborelui Huffman (pentru acest exemplu 11 structuri de tipul TagHuffmanNode, prima reprezentând rădăcina arborelui Huffman)



- urmat de codificarea fişierul de intrare
    - pe fiecare octet, scrierea codificării se face de la bitul cel mai semnificativ spre cel nesemnificativ.
    - la final, în cazul in care numarul de biti ocupaţi de codificare nu este divizibil cu 8, se adaugă un padding de 0. Pentru acest exemplul, codificarea şirului (ana are mere) ocupă 30 de biţi, adăugând un padding de 0 până la 32 de biţi (4 octeţi).



5. Paşi pentru realizarea decompresiei:

5.1 citirea datelor necesare crearii arborelui Huffman

5.2 parcurgand restul fisierului (asociat fisierului iniţial) şi ţinând cont de codificarea fiecărui caracter, se reface fişierul iniţial.

5. Precizări

Tema este individuală
Deadline hard pe 10.05.2015, 23:55
Nu uitaţi să eliberați memoria. Pentru a vă testa programul ca să fiți siguri că nu are pierderi de memorie puteți folosi utilitarul numit Valgrind [4].

6. Documentaţie

[1] http://www.ascii-code.com/
[2] http://en.wikipedia.org/wiki/Extended_ASCII
[3] https://gcc.gnu.org/onlinedocs/gcc/Type-Attributes.html
[4] http://techblog.rosedu.org/valgrind-introduction.html