INSA INSTITUT NATIONAL DES SCIENCES APPLIQUÉES RENNES

C language 2

Labs 8 and 9
S5 EII

2020-2021

Contents

INSA

LAB 10

Trees : Huffman coding

Objectives : (*3h*)
- Implement a binary tree ;
- Use recursive programming in the tree use case ;
- Build a tree by a bottom-up method ;
- First approach of a coding method : Huffman tree ;
- Read image data from binary files.

Requirement :
- read the course document on trees ;
- read the whole practical subject ;
- write the Makefile for the project before the practical session.

10.1 Overview of the practical works on Huffman coding and trees

You will use a tree to implement a coding method. It will require two labs. Create your Makefile and update it when necessary.

10.2 Context

In this lab, we want to code an image of size 64×64. The 64×64 pixels of the image are integer from 0 and 7, *ie.* 8 values. Each pixel value from 0 and 7 will be represented by a codeword. If we use fixed-length codewords, without any coding strategy, the coding rate is therefore 3 bit/pixel (codeword 000 for the value 0, 001 for 1, 010 for 2, ... , 111 for 7). Therefore the file size for the raw data will be $3 * 64 * 64 = 12288$ bits.

It is possible to decrease this rate using a variable-length coding (VLC) method to use les smemory on disk. The Huffman one is optimal for this case. The objective is to associate the shortest codewords to the most frequently used values and the longest ones to the least frequently used.

For example : after reading such an image, the occurrence of each value is counted. It gives us frequencies, or probabilities, for each value. For example, we can get the frequencies as described in Figure 10.1.

Values	0	1	2	3	4	5	6	7
frequencies	0.11	0.01	0.09	0.3	0.05	0.15	0.22	0.07

FIGURE 10.1 – Example of frequencies found in an image

Values	0	1	2	3	4	5	6	7
Codewords	001	10000	000	11	10001	101	01	1001

FIGURE 10.2 – Codewords given by the Huffman code given the previous frequencies

When using the codes of Figure 10.2, instead of 3 bit/point when using a fixed-length coding method, the rate would be :

$$\begin{aligned} \text{Rate} &= 2*(0.3+0.22)+3*(0.15+0.11+0.09)+4*0.07+5*0.06 \\ &= 1.04+1.05+0.28+0.3 \\ &= 2.67 \end{aligned}$$

The file size for the raw data will be then $2.67 * 64 * 64 = 10937$ bits, which is lower than the previous 12288 bits obtained without any coding strategy.

Before studying how to get such Huffman codes, first you will code how to read the image and estimate the frequencies for each value.

10.3 First step : reading data from files

Each provided image ("IM1.IM" and "IM2.IM") is a 64×64 image where each pixel is only an *short int* of size 2 bytes (*int16_t*). The 2D array of pixel has been saved in each file by a simple binary copy of the memory, row after row. Therefore they are both of size $2 * 64 * 64 = 8,2$ ko. To get the better code for an image, it is useful to estimate the frequencies from the 64×64 image values contained in a binary file.

Question 10.1 : Implement the following function :

```
int readProbaFromFile(char* filename, float pb[NBVALUES]);
```

where *filename* is the name of the image and *pb* the array of the frequencies of each value (NBVALUES = 8 in this lab). This latter array is filled by the function by counting the occurency for each value in this image and divide it by the total number of pixel.

Question 10.2 : Test it with the two provided image files "IM1.IM" and "IM2.IM". You should find the same results as in Figure 10.3 and Figure 10.4.

Values	0	1	2	3	4	5	6	7
frequencies	0.094	0.21	0.01	0.391	0.046	0.146	0.065	0.038

FIGURE 10.3 – Values frequencies for "IM1.IM"

Values	0	1	2	3	4	5	6	7
frequencies	0.033	0.059	0.015	0.082	0.606	0.04	0.126	0.04

FIGURE 10.4 – Values frequencies for "IM2.IM"

The next paragraph is about how to find the Huffman codes to decrease the coding rate.

It will requires a binary tree. Let's note that for implementing the binary tree structure, you do not need to fully understand the method as long as you respect the functionalities specifications, although it helps a lot.

10.4 Huffman method

10.4.A Huffman tree : principle

The codes are closely linked to a binary tree where the left child means adding a bit 0 in the code and right child means adding a bit 1 (cf Figure 10.5).

Extract codewords

The codeword associated to an image value is obtained by traversing the Huffman tree from the root to the leaf associated to the value. When traversing a left child, a 0 is added to the codeword, when traversing a right one, a 1 is added.

For example, in the Huffman tree in Figure 10.5, the codeword for the value 0 (frequency 0.11) is 001.

10.5 Second step : the tree structure

To handle properly such a tree, it is required to define a new binary tree structure with its functionalities.

```
typedef struct NodeTree {
    float proba;
    struct NodeTree * left;
    struct NodeTree * right;
} NodeTree;

typedef NodeTree * Binary_tree;

NodeTree * newNodeTree(float p, NodeTree * l, NodeTree * r);
void deleteNodeTree(NodeTree * n);
NodeTree * buildParentNode(NodeTree *p1, NodeTree *p2);
void printTree(Binary_tree t);
void printCodewords(Binary_tree t);
```

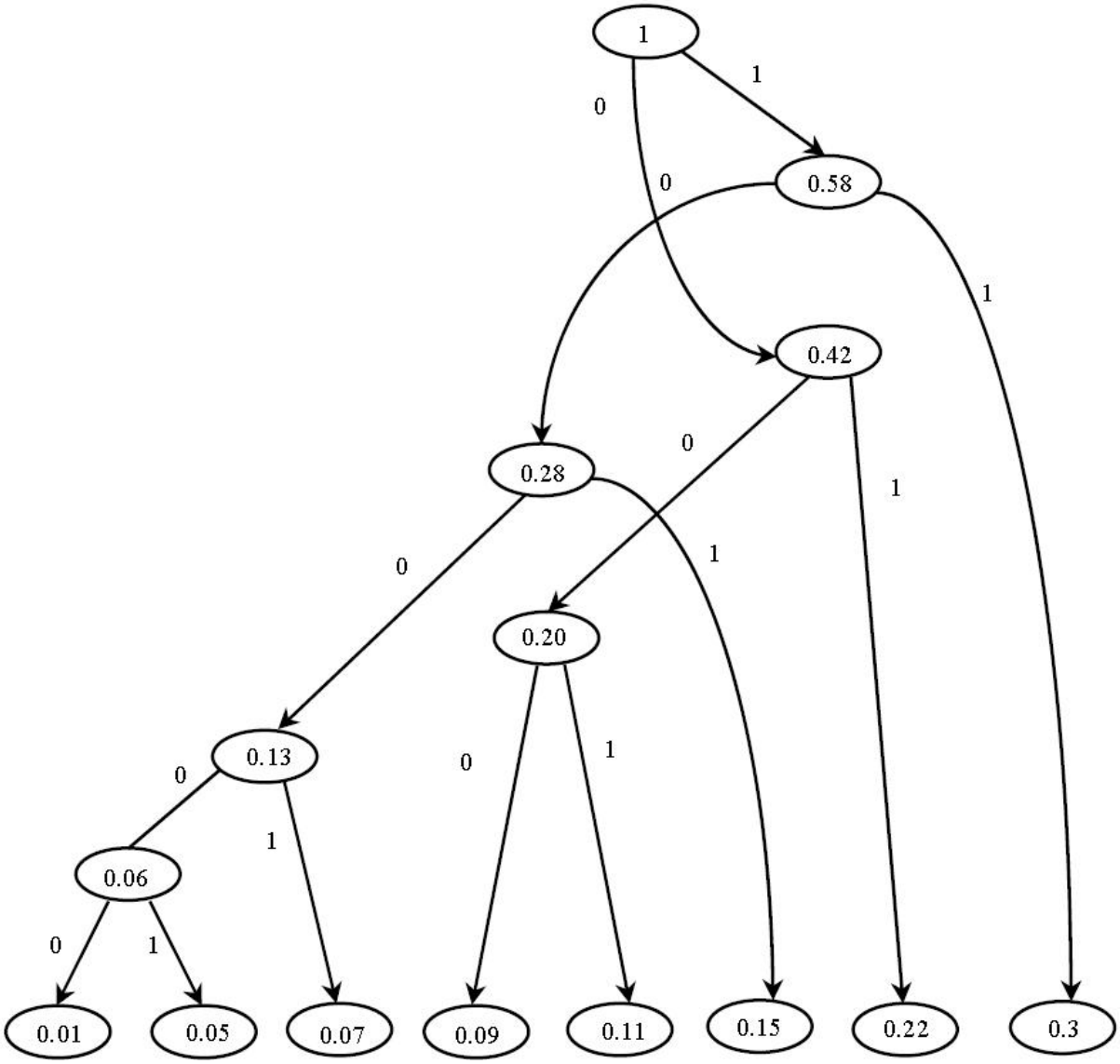

FIGURE 10.5 – Building the Huffman tree from the given frequencies if Figure 10.1

printCodewords() will print for each leaf its probability and the related codeword. For example, for the tree given in example in Figure 10.5, it will display the following values (may be in a different order if you prefer) :

```
le code dont la proba. est  0.01 est:  1 0 0 0 0
le code dont la proba. est  0.05 est:  1 0 0 0 1
le code dont la proba. est  0.07 est:  1 0 0 1
le code dont la proba. est  0.15 est:  1 0 1
le code dont la proba. est  0.3 est:  1 1
le code dont la proba. est  0.09 est:  0 0 0
le code dont la proba. est  0.11 est:  0 0 1
le code dont la proba. est  0.22 est:  0 1
```

Question 10.3 : Implement all these functions.
To ease the implementation of *printCodewords()* function, here is a hint : travel in the tree from root to leaf and store the current code. When at a leaf, print the code by using the following function :

```
static void printCode(char cod[CODEMAX], int size_code, float pb)
{
  int i;
  printf("Codeword for probability  %5.3f is : ",pb);
  for (i=0;i<size_code;i++)
    printf(" %c", cod[i]);
  printf("\n");
}
```

Question 10.4 : Write a program to test each functionality.

Question 10.5 : What is the meaning of the *static* keyword in the *printCode()* function prototype ? Why is it appropriate to use it in this case ?

10.6 Third step : build the tree structure

It is a bottom-up process in this case : it means starting from the leaves to reach the root. Each node is associated to a probability. At the beginning, we then have 8 leaves, each one with the frequency for each image value.

At each step, we look for the two smallest frequencies that have not been processed yet. They are replaced by a new node built such as its left child is the smallest probability (associated to 0), the other one is the right child (associated to 1). The probability of the new node is the sum of its children's frequencies.

The process is repeated until only one node remains. This latter is the root of the Huffman tree.

To do this process, we need a sorted list with nodes associated to the frequencies that have to be processed.

10.6.A The use of a list of frequencies to build the Huffman tree

This list will be sorted by increasing frequencies. For our example in Figure 10.1, the list at the beginning contains 8 leaves, the 8 leaves with the frequencies of each image value and the first node is the smallest probability, as described in Figure 10.6.

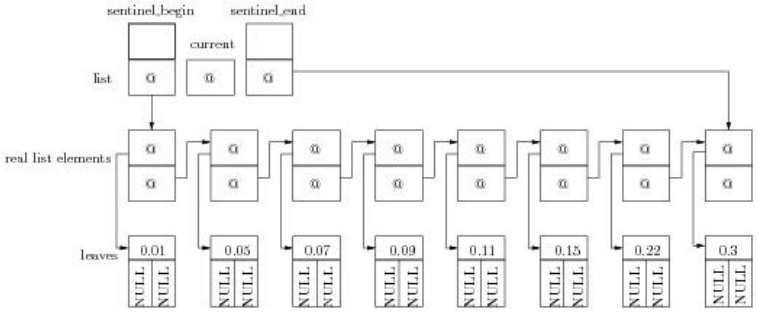

FIGURE 10.6 – List at the beginning of the process

To build the Huffman tree, you need to build a new sub-tree with the two sub-trees corresponding to the smallest frequencies, the left child being associated to the smallest probability. For example, in Figure 10.7 is the result after the first sub-tree fusion.

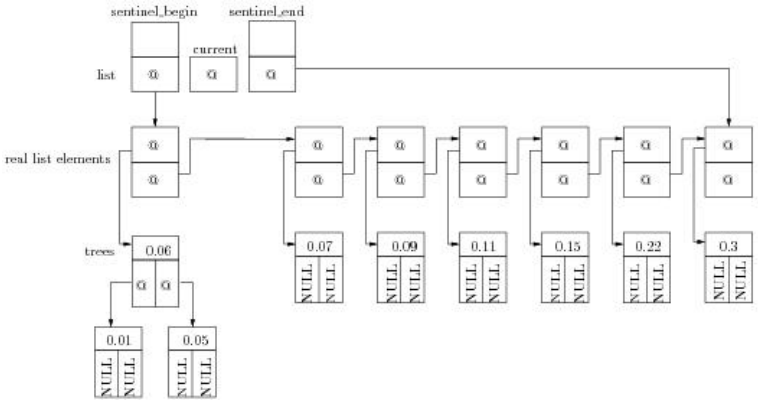

FIGURE 10.7 – List after the first node fusion

The next two steps are illustrated in Figure 10.8 and Figure 10.9.

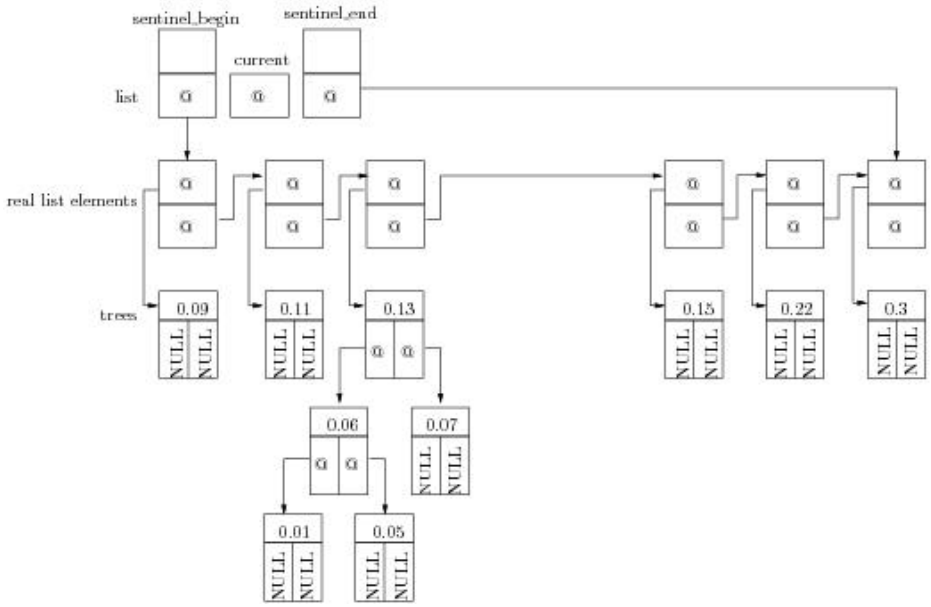

FIGURE 10.8 – List after the second node fusion

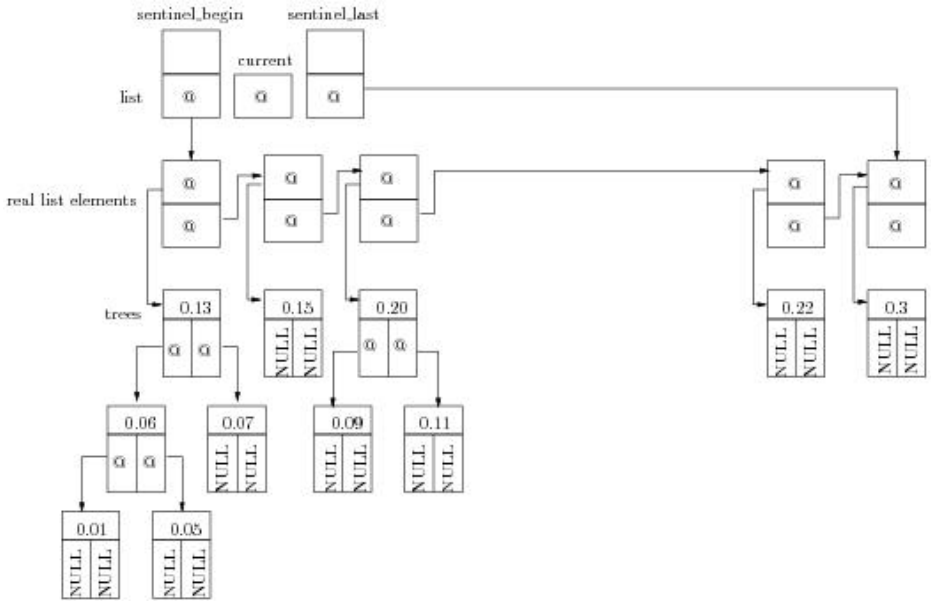

FIGURE 10.9 – List after the third node fusion

At the end, there is only one list node left, linked to the root of the Huffman tree, whose probability is equal to 1 as shown in Figure 10.10.

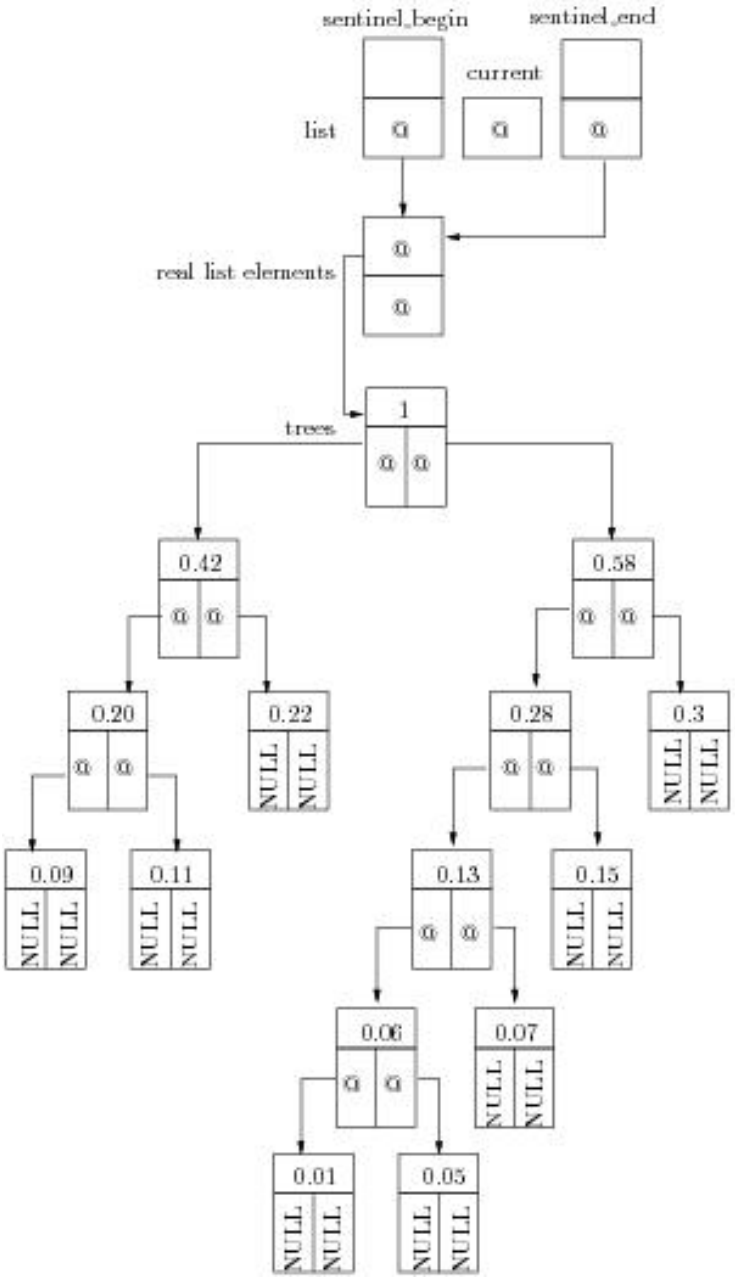

FIGURE 10.10 – List at the end of the process

10.6.B The list of frequencies

Therefore, a list structure has to be managed.

```
typedef struct NodeList{
  struct NodeTree *t;
  struct NodeList *next;
  struct NodeList *previous;
}NodeList;
typedef struct{
  struct NodeList sentinel_begin;
  struct NodeList *current;
  struct NodeList sentinel_end;
}List;

void initList(List *l);
void deleteList(List *l);
int isEmpty(List *l);
int isFirst(List *l);
int isLast(List *l);
int isOutOfList(List *l);
void setOnFirst(List *l);
void setOnLast(List *l);
void setOnNext (List *l);
void setOnPrevious (List *l);
NodeTree * getCurrentTree(List *l);
void printList(List *l);
int insertSort(List *l, NodeTree *p);
NodeTree * removeFirst(List *l);

int fillList(List * lp, float pb[NBVALUES]); //build a sorted list of
    leaves from a given array of frequencies
NodeTree * buildHuffmanTree(List *lp);       //build a Huffman tree from
    such a list using the provided list functions
```

The implementation is provided and compiled in the files "list_teacher.o" and "huffman_method_teacher".

Question 10.6 : Test them with the frequencies given in the example Figure 10.2.
Test it with the two provided image files. You should find the following results :

Values	0	1	2	3	4	5	6	7
frequencies	0.094	0.21	0.01	0.391	0.046	0.146	0.065	0.038
Codewords	100	111	110110	0	11010	101	1100	110111

FIGURE 10.11 – Huffman codewords for "IM1.IM"

Values	0	1	2	3	4	5	6	7
frequencies	0.033	0.059	0.015	0.082	0.606	0.04	0.126	0.04
Codewords	01001	0101	01000	001	1	0000	011	0001

FIGURE 10.12 – Huffman codewords for "IM2.IM"

Question 10.7 : OPTIONNAL THIS SPECIAL YEAR Implement the list functions and use your own code instead of using "list_teacher.o" and "huffman_method_teacher.o".

INSA Rennes
20 Avenue des Buttes de Coësmes
CS 70839
35708 Rennes Cedex 7

Tél. +33 (0) 2 23 23 82 00
Fax +33 (0) 2 23 23 83 96

www.insa-rennes.fr

INSA

UNIVERSITÉ EUROPÉENNE DE BRETAGNE

Commission des Titres d'Ingénieur

MINISTÈRE DE L'ENSEIGNEMENT SUPÉRIEUR ET DE LA RECHERCHE