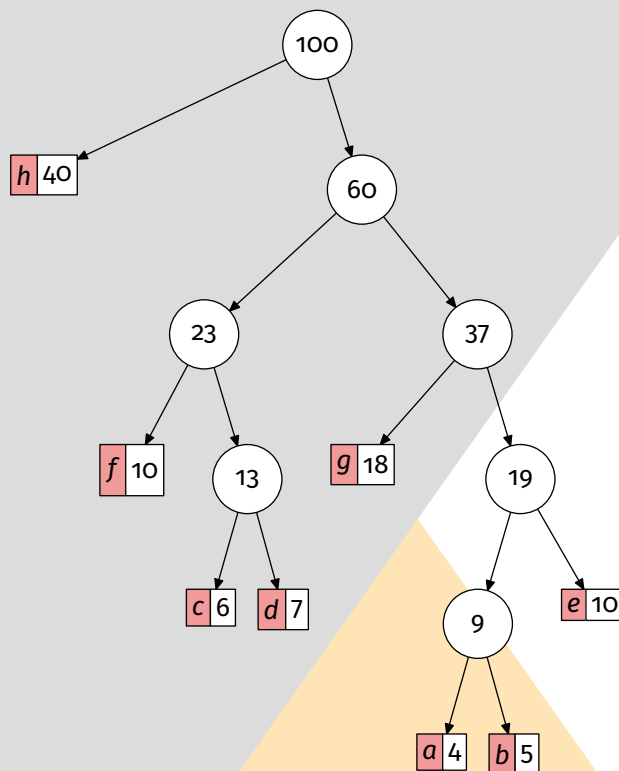


# huffman

drawing binary Huffman trees  
with METAPOST and METAOBJ



**Contributor**

Maxime CHUPIN

[maxime.chupin@tuta.com](mailto:maxime.chupin@tuta.com)

Version 0.2, 2026, june, 4th

<https://gitlab.gutenberg-asso.fr/mchupin/huffman>

## Abstract

This METAPOST package allows to draw Huffman trees from two arrays : an array of strings, and an array of weights (`numeric`). It is based on the METAOBJ package which provides many tools to build trees in general.

<https://gitlab.gutenberg-asso.fr/mchupin/huffman>

## Contents

<b>1</b>	<b>Installation</b>	<b>2</b>
1.1	With T <sub>E</sub> Xlive under Linux or macOS . . . . .	2
1.2	With MikT <sub>E</sub> X and Windows . . . . .	3
1.3	Dependencies . . . . .	3
<b>2</b>	<b>Main Commands</b>	<b>3</b>
<b>3</b>	<b>Package Options</b>	<b>5</b>
<b>4</b>	<b>METAOBJ Tree Options</b>	<b>7</b>
<b>5</b>	<b>Access to Nodes and Leaves</b>	<b>9</b>
<b>6</b>	<b>Constructors</b>	<b>11</b>
<b>7</b>	<b>History</b>	<b>13</b>

*This package is in beta version—do not hesitate to report bugs, as well as requests for improvement.*

## 1 Installation

`huffman` is on CTAN and can also be installed via the package manager of your distribution.

<https://www.ctan.org/pkg/huffman>

### 1.1 With T<sub>E</sub>Xlive under Linux or macOS

To install `huffman` with T<sub>E</sub>XLive, you will have to create the directory `texmf` in your home.

```
user $> mkdir ~/texmf
```

Then, you will have to place the `huffman.mp` file in  
`~/texmf/metapost/huffman/`

Once this is done, `huffman` will be loaded with the classic METAPOST input code

```
input huffman
```

## 1.2 With MikTeX and Windows

These two systems are unknown to the author of `huffman`, so we refer you to the MikTeX documentation concerning the addition of local packages:

<http://docs.miktex.org/manual/localadditions.html>

## 1.3 Dependencies

`huffman` depends, of course on METAPOST [2], as well as the packages `metaobj` [1] and—if `huffman` is not used with LuaTeX and the `luamplib` package—the `latexmp` package.

## 2 Main Commands

The package `huffman` provides two principal commands (which is a METAOBJ like constructor):

First, the following command build a *binary* Huffman tree.

```
newBinHuffman.<name>(<sizeofarrays>)(<symbarray>,<valuearray>)
```

**<name>**: is the name of the object;

**<sizeofarrays>**: is the size (integer) of the arrays;

**<symbarray>**: is the array of **string** containing the symbols;

**<valuearray>**: is the array of **numeric** containing the weights associated to the symbols.

The data arrays should begin at index 1.

### Exemple 1

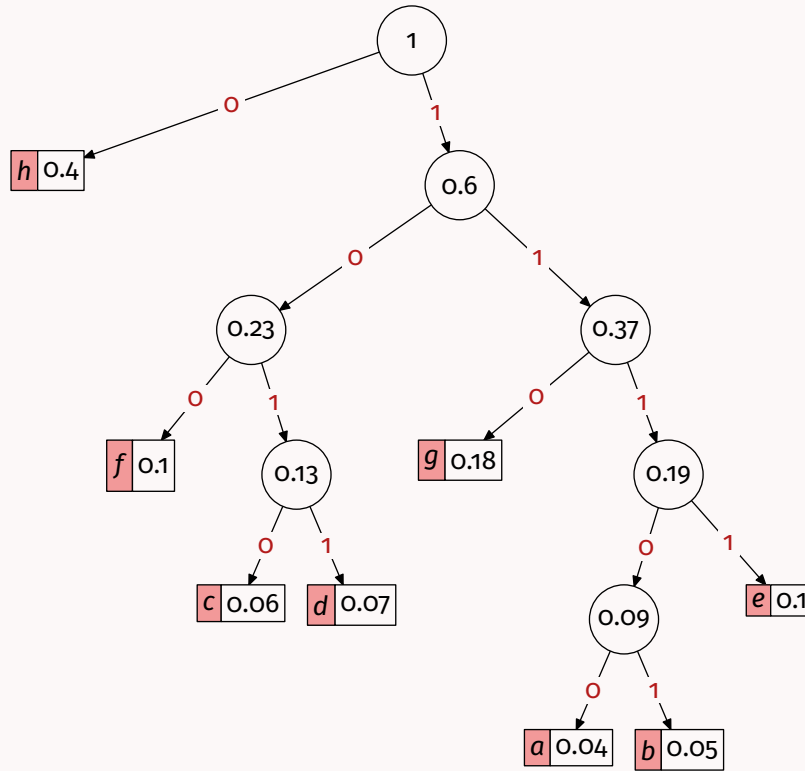
```
input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="a"; frequency[1]:=0.04;
charList[2]:="b"; frequency[2]:=0.05;
charList[3]:="c"; frequency[3]:=0.06;
charList[4]:="d"; frequency[4]:=0.07;
charList[5]:="e"; frequency[5]:=0.1;
charList[6]:="f"; frequency[6]:=0.1;
charList[7]:="g"; frequency[7]:=0.18;
charList[8]:="h"; frequency[8]:=0.4;
```

```

newBinHuffman.myHuff(8)(charList,frequency);
myHuff.c=origin;
drawObj(myHuff);
endfig;

```



Note that the symbols are composed in mathematical T<sub>E</sub>X mode.  
This package provides another command to build general  $q$ -ary Huffman tree.

`newHuffman.<name>(<sizeofarrays>,<sizeofalphabet>)(<symparray>,<valuearray>)`

**<name>**: is the name of the object;

**<sizeofarrays>**: is the size (integer) of the arrays;

**<sizeofalphabet>**: is the size (integer) of the coding alphabet (if `sizeofalphabet=2` we get the behaviour of `newBinHuffman`<sup>1</sup>);

**<symparray>**: is the array of `string` containing the symbols;

**<valuearray>**: is the array of `numeric` containing the weights associated to the symbols.

Once again, the data arrays should begin at index 1.

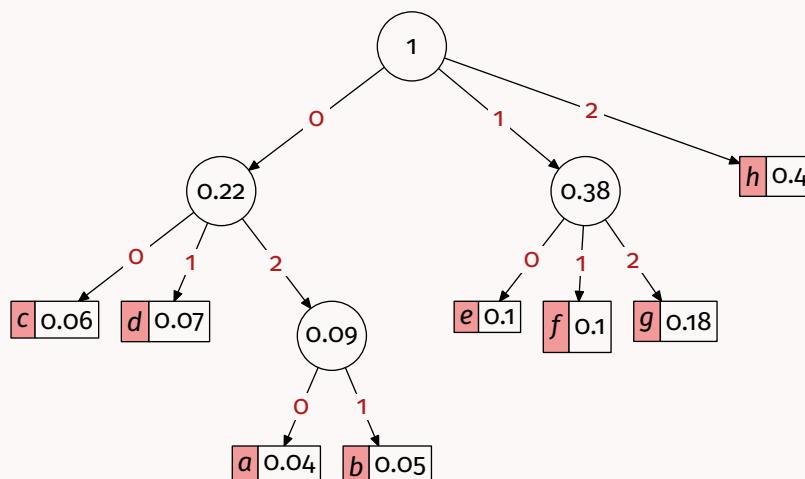
<sup>1</sup>In fact, since version 0.2, `newBinHuffman` calls `newHuffman` to build the binary tree.

## Exemple 2

```
input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]="a"; frequency[1]:=0.04;
charList[2]="b"; frequency[2]:=0.05;
charList[3]="c"; frequency[3]:=0.06;
charList[4]="d"; frequency[4]:=0.07;
charList[5]="e"; frequency[5]:=0.1;
charList[6]="f"; frequency[6]:=0.1;
charList[7]="g"; frequency[7]:=0.18;
charList[8]="h"; frequency[8]:=0.4;

newHuffman.myHuff(8,3)(charList,frequency);
myHuff.c=origin;
drawObj(myHuff);
endfig;
```



## 3 Package Options

You can modify the size of the internal nodes of the tree with the following command:

```
set_node_size(<dim>)
```

<dim>: is the diameter with unit of the circle (default: 13pt).

You can change the color for the symbol boxes with the following command:

```
set_leaf_color(<color>)
```

`<color>`: is a METAPOST `color`.

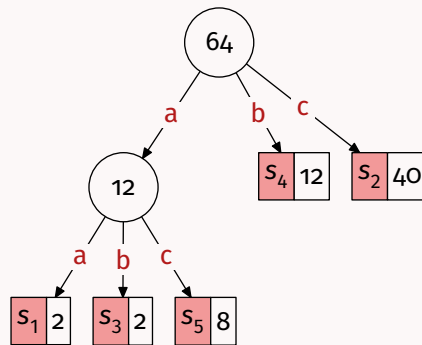
You can hide the bit values printed on the edges of the tree with the following boolean (`true` by default):

`show_bits`

To change the alphabet for *bits* which is for a general  $q$ -ary tree the  $q$  first integer starting at 0, you can modify the string tabular `huffmanAlphabet` (this tabular starts indexing at 1).

### Example 3

```
input huffman
beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=40;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;
huffmanAlphabet[1]:="a";
huffmanAlphabet[2]:="b";
huffmanAlphabet[3]:="c";
newHuffman.myHuff(5,3)(charList,frequency);
myHuff.c=origin;
drawObj(myHuff);
endfig;
```



Similarly, you can set the following boolean to `false` to hide the node values:

`show_node_values`

Finally, you can set the following boolean to `false` to hide the leaf values:

`show_leaf_values`

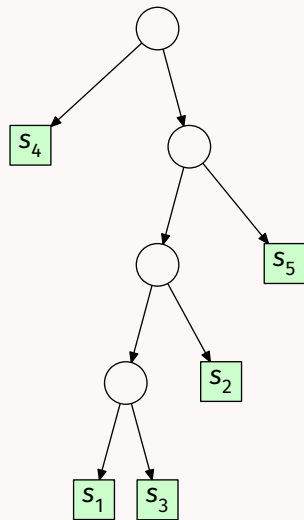
Here an example combining all these commands and variables.

#### Example 4

```
input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

set_leaf_color(0.2[white,green]);
set_node_size(8pt);
show_bits:=false;
show_node_values:=false;
show_leaf_values:=false;
newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c=origin;
drawObj(myHuff);
endfig;
```



## 4 METAOBJ Tree Options

Because the Huffman tree is built using the METAOBJ tree constructor, the METAOBJ tree options are available. All of them are not well suited for this application mostly because the Huffman tree is built using elementary trees. The options we give to the Huffman constructor are passed to all the subtrees.

We give in table 1 the METAOBJ options for the trees that could be used for the Huffman constructor.

Here is an example using some of these options.

Option	Type	Default	Description
treemode	string	"D"	direction in which the tree develops; there are four different possible values: "D" (default), "U", "L" and "R"
treeflip	boolean	false	if true, reverses the order of the subtrees
treenodehsize	numeric	-1pt	if non-negative, all the nodes are assumed to have this width
treenodevsize	numeric	-1pt	if non-negative, all the nodes are assumed to have this height
dx	numeric	0	horizontal clearance around the tree
dy	numeric	0	vertical clearance around the tree
hsep	numeric	1cm	for a horizontal tree, this is the separation between the root and the subtrees
vsep	numeric	1cm	for a vertical tree, this is the separation between the root and the subtrees
hbsep	numeric	1cm	for a vertical tree, this is the horizontal separation between subtrees; the subtrees are actually put in a HBox and the value of this option is passed to the HBox constructor
vbsep	numeric	1cm	for an horizontal tree, this is the vertical separation between subtrees; the subtrees are actually put in a HBox and the value of this option is passed to the HBox constructor
edge	string	"ncline"	name of a connection command (see METAOBJ documentation)
Dalign	string	"top"	vertical alignment of subtrees for trees that go down (the root on the top); the other possible values are "center" and "bot"

Table 1: Table of METAOBJ tree options.

#### Example 5

```
input huffman

beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

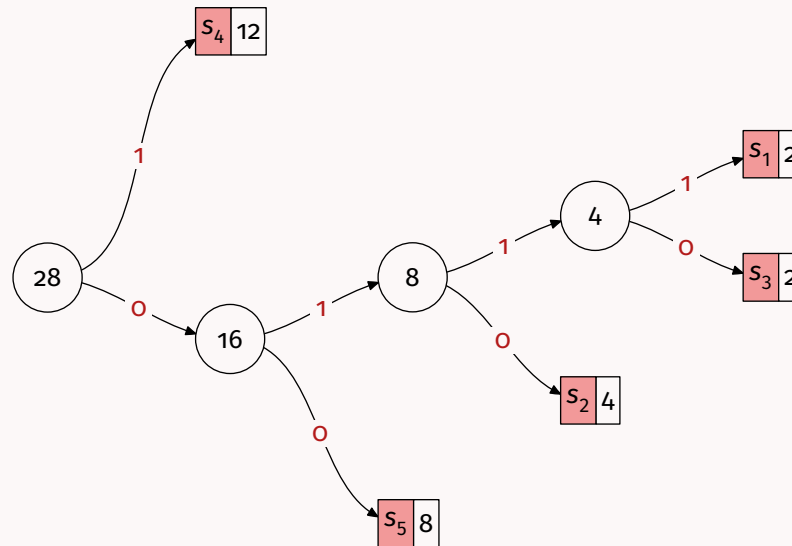
newBinHuffman.myHuff(5)(charList,frequency)
"treemode(R)", "treeflip(true)", "hsep(1.5cm)", "edge(nccurve)" ,
```



```

    "angleA(0)", "angleB(0)";
myHuff.c=origin;
drawObj(myHuff);
endfig;

```



## 5 Access to Nodes and Leaves

To access the nodes and the leaves, you can use the `treeroot` command from `METAOBJ`, see the documentation for details.

`ntreepos(Obj(<name>))(<int>,<int>,<etc.>)`

The sequence of `<int>` gives the choice of branch where the children are numbered from 1 to  $n$ .

The following example shows a use of this mechanism.

### Exemple 6

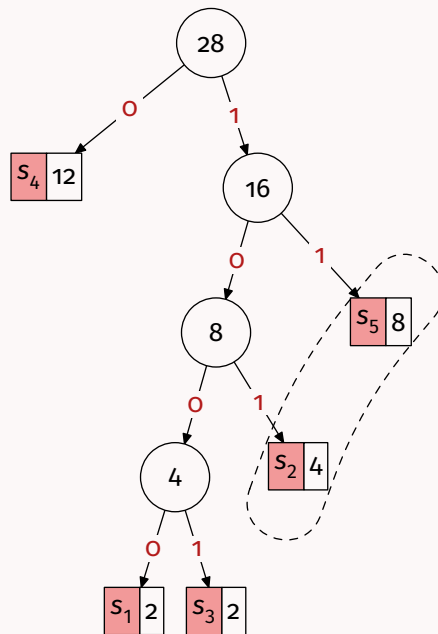
```

input huffman;
beginfig(0);
string charList[];
numeric frequency[];
charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c=origin;
ncarcbox(ntreepos(Obj(myHuff))(2,1,2))(ntreepos(Obj(myHuff))
(2,2))

```

```
"linestyle(dashed evenly)", "nodesepA(5mm)", "nodesepB(5mm)" ;
drawObj(myHuff);
endfig;
```



Of course, this can only be used in two steps, first build the tree, then annotate it.

You can also access the leaves and the nodes using names. During the construction of the tree, names are given to leaves and nodes. Because you may want to build several Huffman trees, the trees are numbered. You can get the current tree number with the following command:

```
get_huffmanTreeNbr()
```

The leaves are numbered during the construction, and the corresponding METAObj object is named as follows:

```
leaf<leaf number>_{tree number}
```

The nodes are numbered during the construction, and the corresponding METAObj object is named as follows:

```
node<node number>_{tree number}
```

Thanks to METAObj you can annotate the tree using all the tools METAObj provides.

### Exemple 7

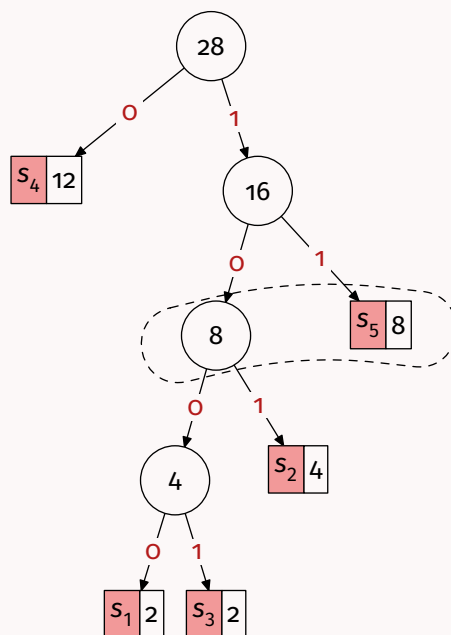
```
input huffman;
beginfig(0);
string charList[];
numeric frequency[];
```

```

charList[1]:="s_1"; frequency[1]:=2;
charList[2]:="s_2"; frequency[2]:=4;
charList[3]:="s_3"; frequency[3]:=2;
charList[4]:="s_4"; frequency[4]:=12;
charList[5]:="s_5"; frequency[5]:=8;

newBinHuffman.myHuff(5)(charList,frequency);
myHuff.c:=origin;
ncarcbox(node2_1)(leaf4_1)
"linestyle(dashed evenly)", "nodesepA(5mm)", "nodesepB(5mm)" ;
drawObj(myHuff);
endfig;

```



## 6 Constructors

The Huffman algorithm uses only three constructors that you can redefine to adapt the tree to your needs. Here are the three constructors (roughly commented in French) defined in this package. We will not discuss the code here but you are free to redefine and adapt it.

### Leaf Code

```

% style 'dune feuille caractère et proba
vardef newHuffmanLeaf@#(expr ch)(expr v) text options=
  % @# est 'lidentifiant de la feuille
  % c est le caractère considéré (ou la chaine)
  % v est la proba ou 'lentier associé

```

```

save _text_v,
_text_token,_height_v,_height_token,_height_max,
_width_token,_width_v;
picture _text_v,_text_token;
% on calcule le height max des deux écritures pour
% faire deux boîtes de même
% hauteur
_text_v := texttext(v);
_text_token := texttext("$"&ch&"$");
_height_v := abs((ulcorner _text_v) - (llcorner
_text_v));
_width_v := abs(urcorner _text_v - ulcorner _text_v);
_height_token := abs(ulcorner _text_token - llcorner
_text_token);
_width_token := abs(urcorner _text_token - ulcorner
_text_token);
_height_max := max(_height_token,_height_v);
% on fabrique deux boîtes vides aux bonne dimensions
% et on ajoute un label au centre de celles-ci
if(show_leaf_values):
  newEmptyBox.scantokens(str @# & "ch1")(
    _width_token+4,2_height_max)
  "framed(true)","fillcolor(_huffmanLeaf)", "filled
  (true)", options;
  ObjLabel.Obj(scantokens(str @# & "ch1"))(texttext(
    "$" & ch & "$"));
  newEmptyBox.scantokens(str @# & "ch2")(_width_v
    +4,2_height_max)
  "framed(true)", options;
  ObjLabel.Obj(scantokens(str @# & "ch2"))(texttext(
    v));
  % on fixe relativement les coordonnées des deux
  % boîtes pour 'quelles se % touchent
  scantokens(str @# & "ch1").e=scantokens(str @# &
    "ch2").w;
  % on fabrique un container qui les regroupe et
  % qui sera la feuille
  newContainer.@#(scantokens(str @# & "ch1"),
    scantokens(str @# & "ch2"));
else:
  % si seulement le symbole
  newBox.@#(texttext("$" & ch & "$"))
  "framed(true)","fillcolor(_huffmanLeaf)", "filled
  (true)", options;
fi
enddef;

```

#### Node Code

```
% style 'dun œnud interne (non feuille) de 'larbre
vardef newHuffmanNode@#(expr v) text options=
  newCircle.@#("") "circmargin(_node_size)",options;
  if(show_node_values):
    ObjLabel.Obj(scantokens(str @#))(texttext(v));
  fi
enddef;
```

#### Tree Code

```
% style de 'larbre de Huffman
vardef newHuffmanTree@#(suffix theroot)(text subtrees)(
  expr q) text options=
  % un simple arbre
  newTree.@#(theroot)(subtrees) "Dalign(top)" , "hbsep
    (0.3cm)",options;
  if(show_bits):
    %
    for i:=1 upto q:
      ObjLabel.Obj(@#)(texttext(huffmanAlphabet[i]))
      "labpathid("&decimal i&")", "laberase(true)", "
      labcolor(_huffmanBit)";
    endfor
  fi
enddef;
```

## 7 History

**june 2026:** Rewrite of the Huffman algorithm and a new command to represent a  $q$ -ary tree (`newHuffman`). Addition of the `\lstinline huffmanAlphabet+` array for encoding symbols.

**may 2023:** Initial version.

## References

- [1] Denis B. Roegel. *The metaobj package. MetaPost package providing high-level objects*. Version 0.93. June 24, 2016. URL: <https://ctan.org/pkg/metaobj>.
- [2] The MetaPost Team and John Hobby. *The metapost package. A development of Metafont for creating graphics*. Aug. 26, 2021. URL: <https://ctan.org/pkg/metapost>.

## Command Index

`get_huffmanTreeNbr`, 10

`newBinHuffman`, 3

`newHuffman`, 4

`ntreepos`, 9

`set_leaf_color`, 5

`set_node_size`, 5

`show_bits`, 6

`show_leaf_values`, 6

`show_node_values`, 6