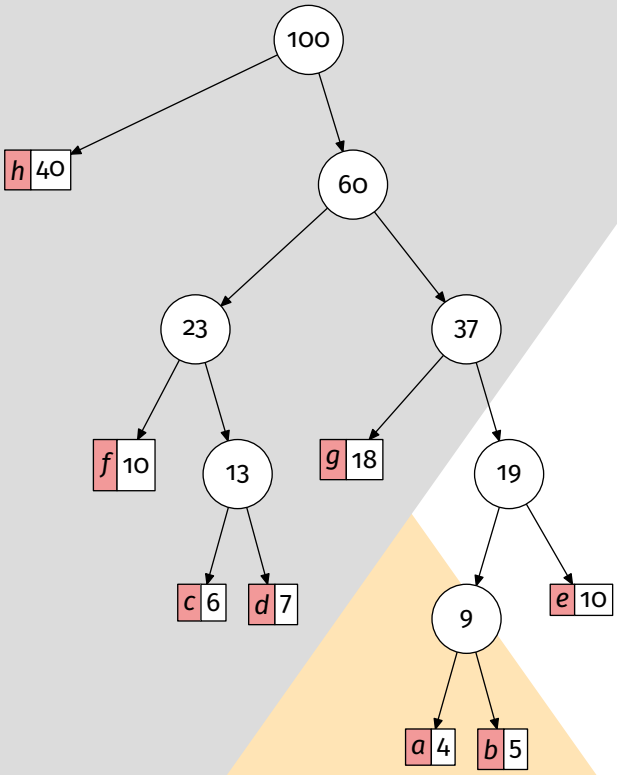
huffman

drawing binary Huffman trees
with METAPOST and METAOBJ



Contributor
Maxime CHUPIN
notezik@gmail.com

Version 0.1, 2023, April, 21th
https://plmlab.math.cnrs.fr/mchupin/huffman

Abstract

This METAPOST package allows to draw binary Huffman trees from two arrays : a string one, and a value one. It is based on METAOBJ package which provides many tools to build trees in general.

https://plmlab.math.cnrs.fr/mchupin/huffman
https://github.com/chupinmaxime/huffman

Contents

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 TEXlive under Linux or macOS

To install huffman with TEXLive, you will have to create the directory texmf directory in your home.

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

Then, you will have to place the huffman.mp file in the

~/texmf/metapost/huffman/

Once this is done, huffman will be loaded with the classic METAPOSTinput 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 MikTEXdocumentation 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 LuaLATEX and the luamplib package—the latexmp package.

2 Main Command

The package huffman provides one principal command (which is a METAOBJ like constructor):

newBinHuffman.⟨*name*⟩(⟨*sizeofarrays*⟩)(⟨*symbarray*⟩,⟨*valuearray*⟩)

⟨***name***⟩**:** is the name of the object;

⟨***sizeofarray***⟩**:** is the size (integer) of the arrays;

⟨***symbarray***⟩**:** is the array of string containing the symboles;

⟨***valuearray***⟩**:** is the array of numeric containing the values associated to the symboles.

The data arrays should begin at index 1.

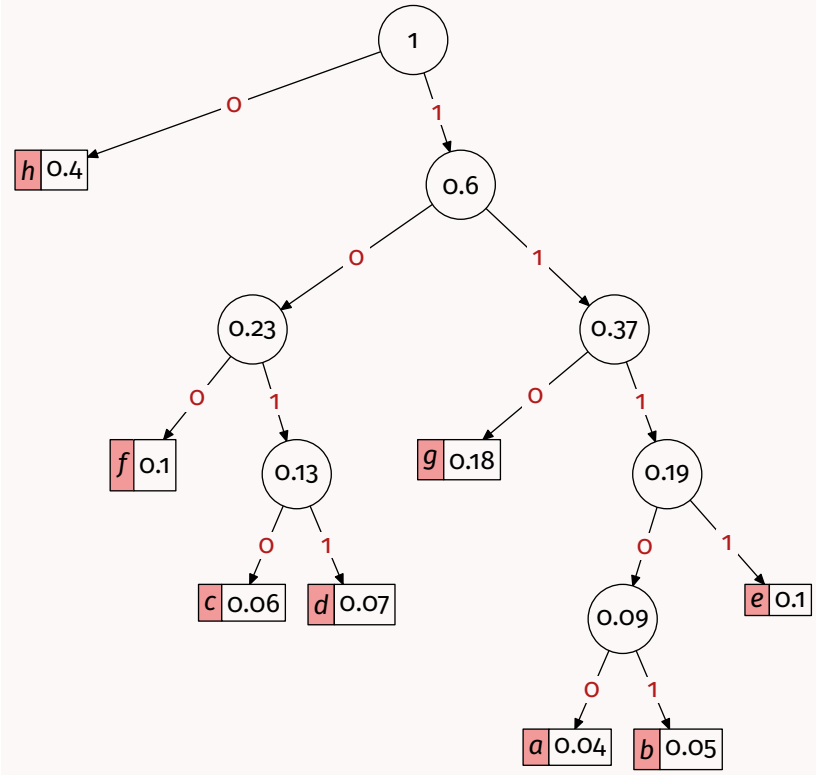
Exemple 1

```
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;
```



Beware, the symbols are composed in mathematical TEX mode.

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 of the circle with unity (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 in the edges of the tree with the following boolean (`true` by default):

`show_bits`

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`

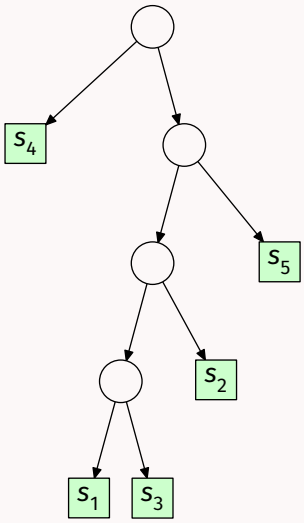
Here an example combining all these commands and variables.

Exemple 2

```
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 build using 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 build using elementary trees, to which the options we give to the Huffman constructor is passed to all the subtrees.

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

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 be- tween the root and the subtrees
vsep	numeric	1cm	for a vertical tree, this is the separation be- tween 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 (se 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.

Here is an example of using some of these options.

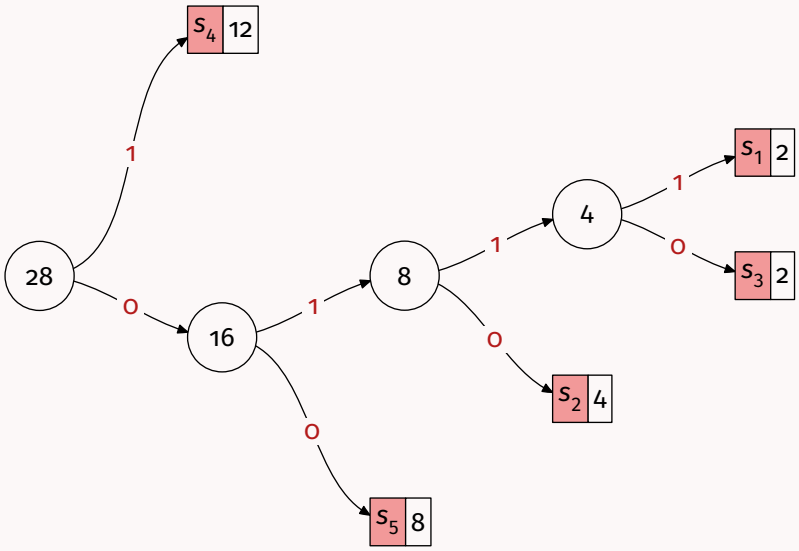
Exemple 3

```
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 trees, 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*.

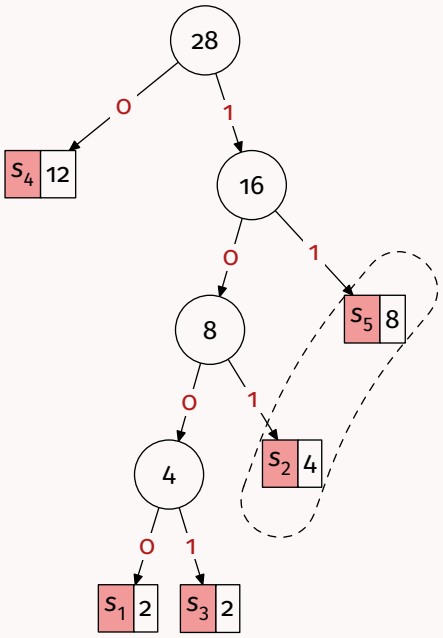
The following example shows a use of this mecanism.

Exemple 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;

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

You can also access the leaves and the nodes using names. During 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:

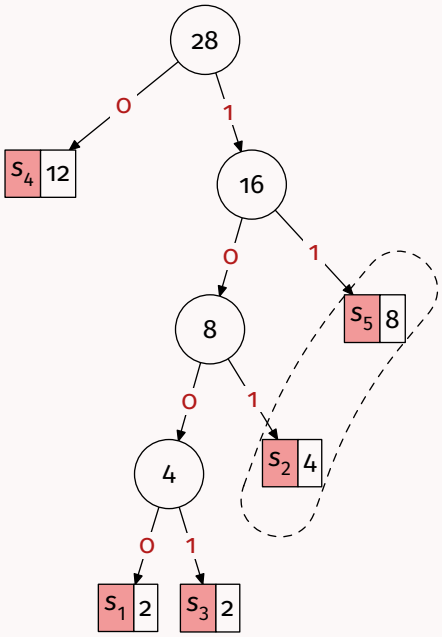
`node⟨`*node number*`⟩_⟨`*tree number*`⟩`

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

Exemple 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);
myHuff.c=origin;
ncarcbox(leaf3_1)(leaf4_1)
"linestyle(dashed evenly)", "nodesepA(5mm)", "nodesepB(5mm)" ;
drawObj(myHuff);
endfig;
```



6 Constructors

The Huffman algorithm use 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 no discus 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 boites de même
    % hauteur
    _text_v := textext(v);
    _text_token := textext("$"&ch&"$");
    _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"))(textext(
            "$" & ch & "$"));
        newEmptyBox.scantokens(str @# & "ch2")(_width_v
            +4,2_height_max)
        "framed(true)", options;
        ObjLabel.Obj(scantokens(str @# & "ch2"))(textext(
            v));
        % on fixe relativement les coordonnées des deux
            boites pour 'quelles se % touchent
        scantokens(str @# & "ch1").e=scantokens(str @# &
            "ch2").w;
        % on fabrique un container qui les regroupes et
            qui sera la feuille
        newContainer.@#(scantokens(str @# & "ch1"),
            scantokens(str @# & "ch2"));
    else:
```

```
        % si seulement le symbole
        newBox.@#(textext("$" & ch & "$"))
        "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 @#))(textext(v));
    fi
enddef;
```

Tree Code

```
% style de 'larbre binaire de Huffman
vardef newHuffmanBinTree@#(suffix theroot)(text subtrees)
      text options=
    % un simple arbre
    newTree.@#(theroot)(subtrees) "Dalign(top)" , "hbsep
        (0.3cm)",options;
    if(show_bits):
        % et on met 0 et 1 sur ses deux connections
        ObjLabel.Obj(@#)(btex 0 etex)
        "labpathid(1)", "laberase(true)", "labcolor(
            _huffmanBit)";
        ObjLabel.Obj(@#)(btex 1 etex)
        "labpathid(2)", "laberase(true)", "labcolor(
            _huffmanBit)";
    fi
enddef;
```

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