Huffcomp Toolbox for SciLab

This small toolbox shows the principles of huffman coding. It consists of 4 routines and the corresponding help files. The Aim of the toolbox is to demonstrate the principles. By the way, the coding functions are quite fast, the decoding function could be optimized.

Installation of the Toolbox

- The toolbox is contained in a ZIP file. Unpack the ZIP file directly in your Scilab directory. Please activate the option in your Unzip program „with subdirectories" or „recursivly".
  You will get a new directory called „huffcomp".
- In order to have a proper installation, delete the files with extension *.bin.
- Start your SciLab program
- In the command window execute the command
  „genlib('Huffcomp','SCI/huffcomp');"
- Exit SciLab
- Edit the file SciLab.star (contained in the main scilab directory) and add the lines
  - load('SCI/huffcomp/lib');
  - add_help_chapter('Huffman Coding','SCI/huffcomp/man');
    The line with „add_help_chapter" should be written to the last line in the file scilab.star
- Now you can start Scilab again, the toolbox will be loaded on startup

Content of the toolbox

4 routines are contained. These routines are

- fasthist
- huffman
- huffcode
- huffdeco

The documentation is only available in English language.
There is no explicit demo file. Please use the example code in the SciLab Help files.

fasthist - fast calculation of histogram

Calling Sequence

```
H = fasthist(A)
```

Parameters

- **H** : sparse vector, containing the number of entries of the matrix/vector A. A is converted inside the function to INT.
- **A** : A is a vector or matrix with positive numbers. The number of occurences of numbers is counted.

Description

The function fasthist uses sparse functions to calculate the histogram in a single step. No while loops are needed. The matrix A may contain positive numbers larger or equal 1.

The result vector h (sparse) represents the histogram, that means the number of occurences of elements of the vector A. H has the length max(int(a)).

Examples

```
// Generate a Testmatrix
A=testmatrix('frk',10)+1;
H1=fasthist(A);
// Now, we add a constant to A...
// let us see, what happens
H2=fasthist(A+6);
disp(H1);
disp(H2);
// End of Example
```

See Also

huffman, **huffcode**

huffman - Huffman Coding based upon a histogram vector

Calling Sequence

```
[SB,h,L,QM]=huffman(H)
```

Parameters

- **H** : a histogram vector (also sparse input possible)
- **SB** : a vector, representing symbols
- **h** : the normalized histogram vector
- **L** : number of bits used for coding the corresponding symbol
- **QM** : the complete code table as string vector. The symbol, the bit sequence and the number of bits are contained.

Description

This function creates the coding tree based upon a given histogram.

Examples

```
// Generate a Testmatrix
A=testmatrix('frk',10)+1;
H1=fasthist(A);
//
[SB,h,L,QM]=huffman(H1);
//SB contains the symbols
disp(SB);
// h is the normalized histogram
disp(h);
// L contains the number of bits used for the symbols
disp(L);
// QM is the complete code table,
// containing symbol, bits and no. of bits
disp(QM);
// End of demo
```

See Also

Huffcode, **fasthist**

huffcode - huffman coding of a sequence

Calling Sequence

```
[QT,QM]=huffcode(A)
```

Parameters

- **A** : vector representing a input sequence. A is converted in the function to type INT. The elements of A must be greater or equal 1.
- **QT** : A string containing 1s and 0s. QT represents the binary output sequence of the huffman coded signal. QT can be converted into a scalar vector with str2code.
- **QM** : the huffman code table, containg symbols, bit sequences and number of bits of a symbol

Description

The function converts the scalar input sequence (vector or matrix) into a compressed bit sequence. The bit sequence is represented by a string containing 1s and 0s.

huffcode requires the function "fasthist" and "huffman". The reverse operation is performed by the scilab function "huffdeco".

Examples

```
// Generate a Testmatrix
A=testmatrix('frk',10)+1;
A=A(:).';
[QT,QM]=huffcode(A);
disp('compressed Bit sequence:');
disp(QT);
disp('Code Table:');
disp(QM);
// End of Demo
```

See Also

fasthist, **huffman**, **huffdeco**

huffdeco - Decoding of a compressed bit sequence.

Calling Sequence

```
B=huffdeco(QT,QM);
```

Parameters

- **QT** : the compressed bit sequence, represented by a string containing 1s and 0s
- **QM** : The huffman code table, containing symbols, bit sequence of a symbol and number of bits per symbol
- **B** : the decoded (uncompressed) sequence.

Description

huffdeco is the reverse operation to huffcode. The output arguments of huffcode can directly be used as input arguments for huffdeco

Examples

```
// Generate a Testmatrix
A=testmatrix('frk',10)+1;
A=A(:).';
[QT,QM]=huffcode(A);
disp('compressed Bit sequence:');
disp(QT);
disp('Code Table:');
disp(QM);
// Now, the reverse operation
B=huffdeco(QT,QM);
disp('Original:');
disp(A);
disp('Result after Uncompress:');
disp(B);
plot2d(B-A);  // A line with y=0 should appear !
// End of Demo
```

See Also

huffcode, **huffman**

Performance

The routines are quite fast. On my system (Laptop Dell Latitude 800), the performance values below have been achieved.

Example 1:

Histogram of a vector with 10000 elements, value range [1,...,255]

```
-->A=(rand(1,10000) .^2)*255+1; A=int(A);
-->tic(); H=fasthist(A); toc()
 ans  = 0.02
```

Coding the vector

```
-->tic(); [QT,QM]=huffcode(A); toc()
 ans  = 1.647
```

Decoding the vector

```
-->tic(); B=huffdeco(QT,QM); toc()
 ans = 7.249
```

Show the differences (hopefully 0 everywhere !)

-->plot2d(B-A);

Example 2

Histogram of an image (640 x 480 pixels)

```
-->A=(rand(640,480) .^2)*255+1; A=int(A);
-->tic(); H=fasthist(A); toc()
 ans  =  1.265
```

Huffman Coding of this image

```
-->tic(); [QT,QM]=huffcode(A); toc()
 ans  = 4.156
-->QM
 QM  =
!1:0111:4         !
!2:10111:5        !
!3:001000:6       !
!4:011000:6       !
!5:100100:6       !
!6:101011:6       !
!7:111000:6       !
!8:111110:6       !
...
[More (y or n ) ?]
```

Decoding the image

Decoding is much slower. May be, you have some ideas to improve the functions. One improvement could be the implementation of a waitbar. Here, the routine was interrupted by me after 1 minute.

Have much fun !