Huffman Coding and Decoding

To build a Huffman code use **Huffman(source)** from **sage.coding.source_coding.huffman.**

Here, **source** is

- a given string, or
- the information of a dictionary associating to each key a weight.
  - a key is an element of the alphabet.
  - a weight is a probability value or a number of occurrences (frecuency table)

```
from sage.coding.source_coding.huffman import Huffman,
frequency_table
```

Build a Huffman code *h1* from an string

```
h1 = Huffman("Shinshu University Nagano")
```

The encoding table can be seen with the function encoding_table():

```
h1.encoding_table()
```

```
{' ': '1010',
 'N': '1000',
 'S': '11101',
 'U': '0011',
 'a': '1001',
 'e': '11010',
 'g': '11011',
 'h': '1011',
 'i': '010',
 'n': '011',
 'o': '11100',
 'r': '11110',
 's': '1100',
 't': '0000',
 'u': '11111',
 'v': '0001',
 'y': '0010'}
```

The items of the encoding table can be accessed by

```
h1.encoding_table().items()
```

```
[('N', '1000'),
 ('S', '11101'),
 ('U', '0011'),
```

```
('a', '1001'),
(' ', '1010'),
('e', '11010'),
('g', '11011'),
('i', '010'),
('h', '1011'),
('o', '11100'),
('n', '011'),
('s', '1100'),
('r', '11110'),
('u', '11111'),
('t', '0000'),
('v', '0001'),
('y', '0010')]
```

Print nicely the letters (elements of the alphabet) and their codes

```
for letter, code in h1.encoding_table().items():
    print("'{}' : {}".format(letter, code))
```

```
'N' : 1000
'S' : 11101
'U' : 0011
'a' : 1001
' ' : 1010
'e' : 11010
'g' : 11011
'i' : 010
'h' : 1011
'o' : 11100
'n' : 011
's' : 1100
'r' : 11110
'u' : 11111
't' : 0000
'v' : 0001
'y' : 0010
```

Create a frecuency table *ft* from an string and verify its contents

```
ft = frequency_table("Shinshu University Nagano")
ft
```

```
{' ': 2,
 'N': 1,
 'S': 1,
 'U': 1,
 'a': 2,
 'e': 1,
 'g': 1,
 'h': 2,
 'i': 3,
 'n': 3,
 'o': 1,
```

```
'r': 1,
's': 2,
't': 1,
'u': 1,
'v': 1,
'y': 1}
```

Create a Huffman code *h2* from a frecuency table

```
h2 = Huffman(ft)
```

Verify the letters (elements of the alphabet) and their codes in *h2*. They should be the same as in *h1* because the frequency table for *h2* was obtained from the same string used to create *h1*.

```
for letter, code in h2.encoding_table().items():
    print("'{}' : {}".format(letter, code))
```

```
'N' : 1000
'S' : 11101
'U' : 0011
'a' : 1001
' ' : 1010
'e' : 11010
'g' : 11011
'i' : 010
'h' : 1011
'o' : 11100
'n' : 011
's' : 1100
'r' : 11110
'u' : 11111
't' : 0000
'v' : 0001
'y' : 0010
```

Once a Huffman code *h* has been created, it possesses an encoding table and it is possible to obtain the Huffman encoding of any string using this code:

```
encoded1 = h1.encode("Shinshu University Nagano")
```

```
encoded1
```

```
'111011011010011110010111111110100011011010000111010111101100
001010101000100111011100101111100'
```

```
encoded2 = h2.encode("Shinshu University Nagano")
```

```
encoded2
```

```
'111011011010011110010111111110100011011010000111010111101100
001010101000100111011100101111100'
```

```
for i in range(len(encoded1)):
    if encoded1[i] != encoded2[i]: print i, encoded1[i],
encoded2[i]
```

We can decode the above encoded string in the following way:

```
h1.decode(encoded1)
```

```
'Shinshu University Nagano'
```

```
h2.decode(encoded2)
```

```
'Shinshu University Nagano'
```

Obviously, if we try to decode a string using a Huffman instance which has been trained on a different sample (and hence has a different encoding table), we will get some random-looking string:

```
h3 = Huffman("Shinshu University Matsumoto")
```

```
h3.decode(encoded1)
```

```
'rut o ytsutir uihitnhomo '
```

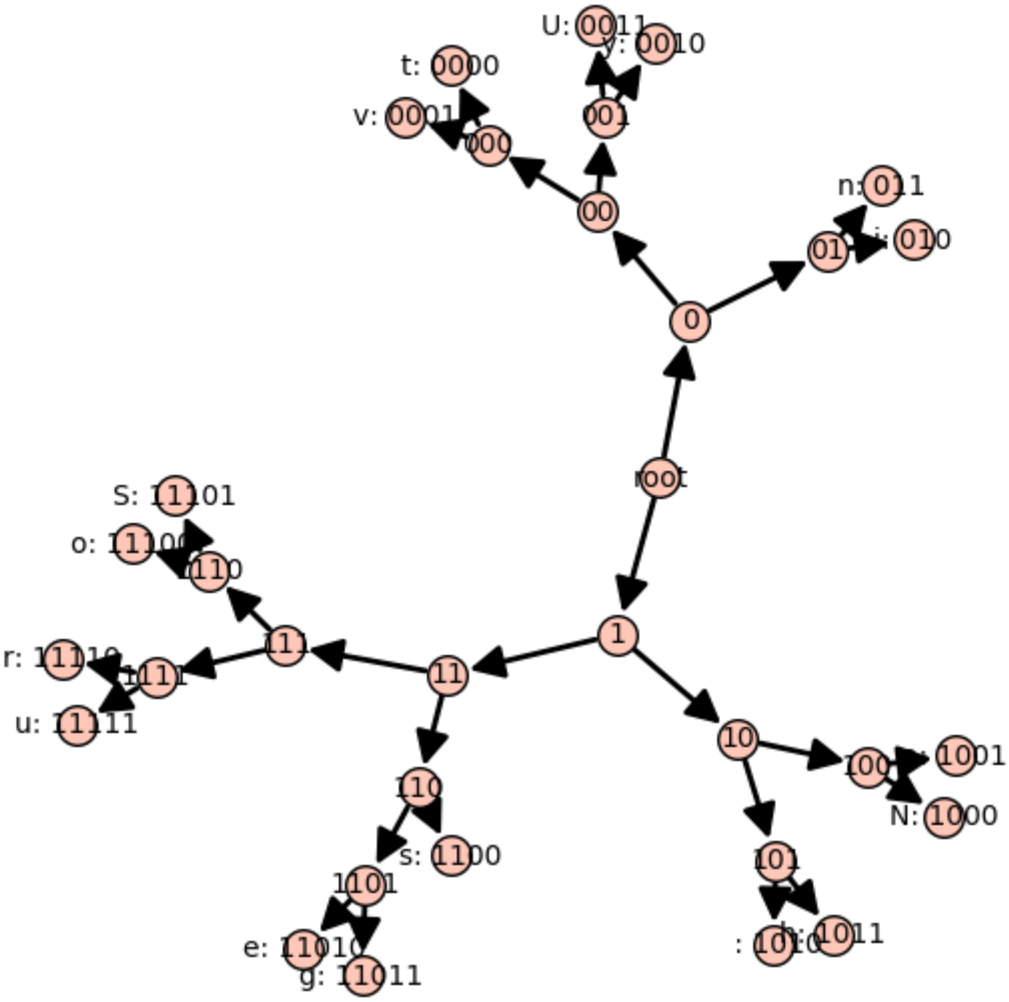
The Huffman tree corresponding to the current encoding can be created by

```
tree = h1.tree()
tree
```

```
Digraph on 33 vertices (use the .plot() method to plot)
```

To see the tree use .plot() or .show() method

```
tree.show(figsize=[5,8])
```


root
0
1
00
01
000
001
t: 0000
v: 0001
U: 0011
y: 0010
n: 011
10
11
100
101
N: 1000
h: 1011
110
111
1101
1110
1111
s: 1100
e: 11010
g: 11011
S: 11101
o: 11100
r: 11110
u: 11111