Trees: An Application

Compression Algorithms

Trees have an interesting application in **compression algorithms**.
Compression involves re-writing a string of symbols belonging to an “alphabet” into a shorter string (which can belong to a different alphabet), without loss of information.

This means that a string can be “compressed” and then “un-compressed” re-obtaining the starting one.

There are many compression techniques, but here we discuss one called **“Huffman coding”**, which has a strong relationship with the tree data structure.

Huffman coding was discovered by **David A. Huffman** (1925-1999) in 1952. He was a computer scientist professor at MIT and UCSC.

Huffman Coding: the idea

Let’s consider the string:

`“SUSIE SAYS IT IS EASY”`

The idea is to represent the most frequent characters with less bits and less frequent characters with more bits.
So, effectively we transfer “complexity” from frequent characters to non-frequent ones.

Symbol	Frequency
A	2
E	2
I	3
S	6
T	1
U	1
Y	2
Space	4
Linefeed	1

For doing this, we need the frequency table of the characters in the string.
How do we choose the encoding?

Huffman Coding

“SUSIE SAYS IT IS EASY”

The most frequent characters are S and the space. We associate to them the 2-bit combinations 10 and 00 respectively.
We cannot use 01 and 11 because other characters will start with such combinations and therefore in a decoding situation you will not be able to distinguish them.
So: **<u>No code can be the prefix of any other code</u>**.
The proposed compression strings according to the character frequency are:

Symbol	Frequency	ASCII	Huffman
A	2	“01000000”	“010”
E	2	“01000001”	“1111”
I	3	…	“110”
S	6	..	“10"
T	1	…	“0110"
U	1	..	“01111"
Y	2	…	“1110”
Space	4	..	“00"
Linefeed	1	….	“01110"

Huffman Coding

The (ASCII) message:

```
“SUSIE SAYS IT IS EASY”
```

becomes:

```
“10 0111 10 110 1111 00 10 010 1110 10 00 110 0110
00 110 10 00 1111 010 10 1110 01110”
```

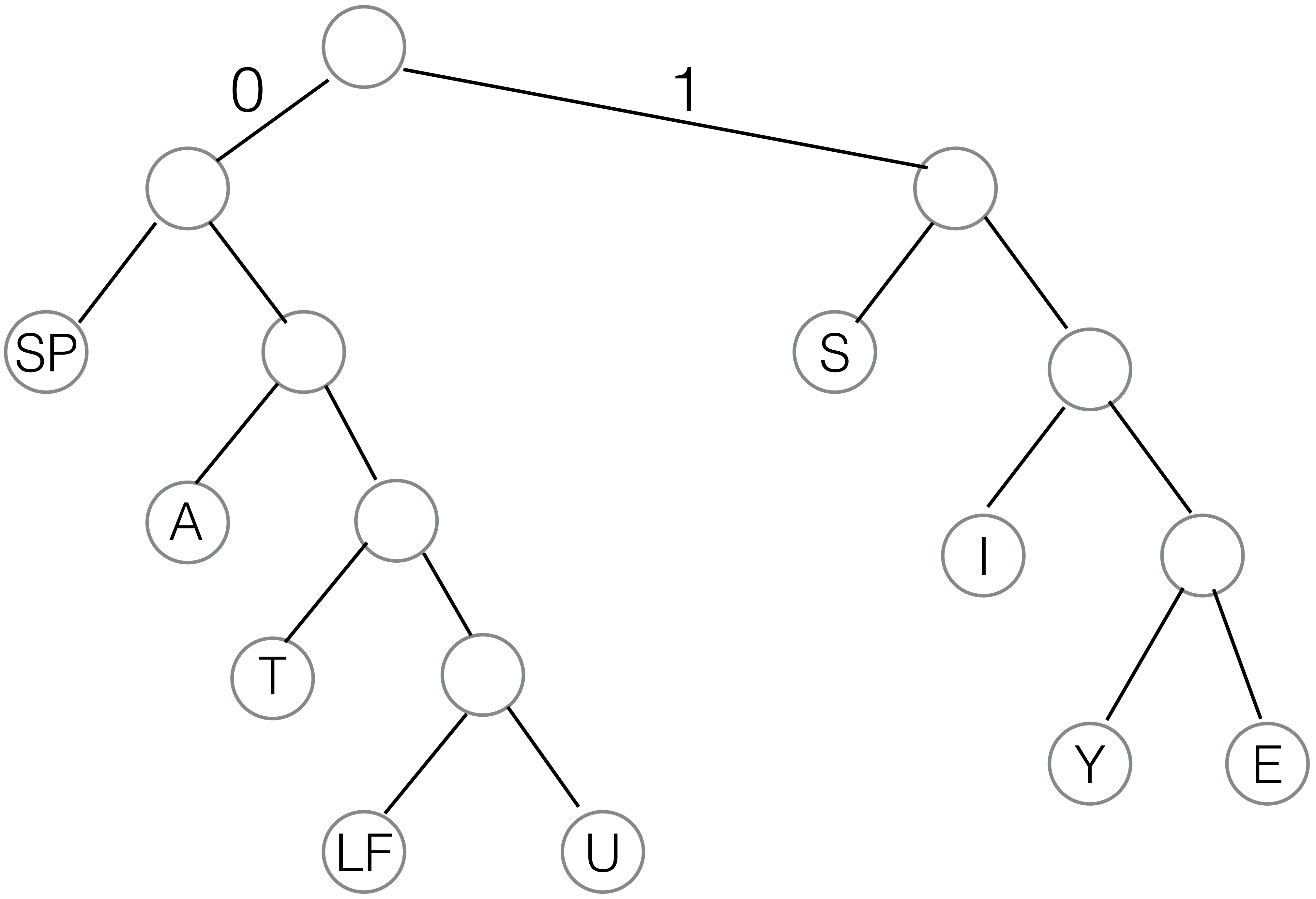
The spaces were added only for clarity. The compressed message is actually:

```
“10011110110111100100101110100011001100011010001111010101110011110”
```

We compressed a string of 22x8=**176** bits into a string of **64** bits.
A compression ratio of about 63% !
The algorithm achieves better performances if the frequency table shows very frequent characters. If the table is very well balanced, the result will be a very poor compression performance.

Decoding with the help of a Tree

Surprisingly, the tree data structure represents the logic behind Huffman coding.



Try to follow: “T”=“0110”

How to create the Decoding Tree

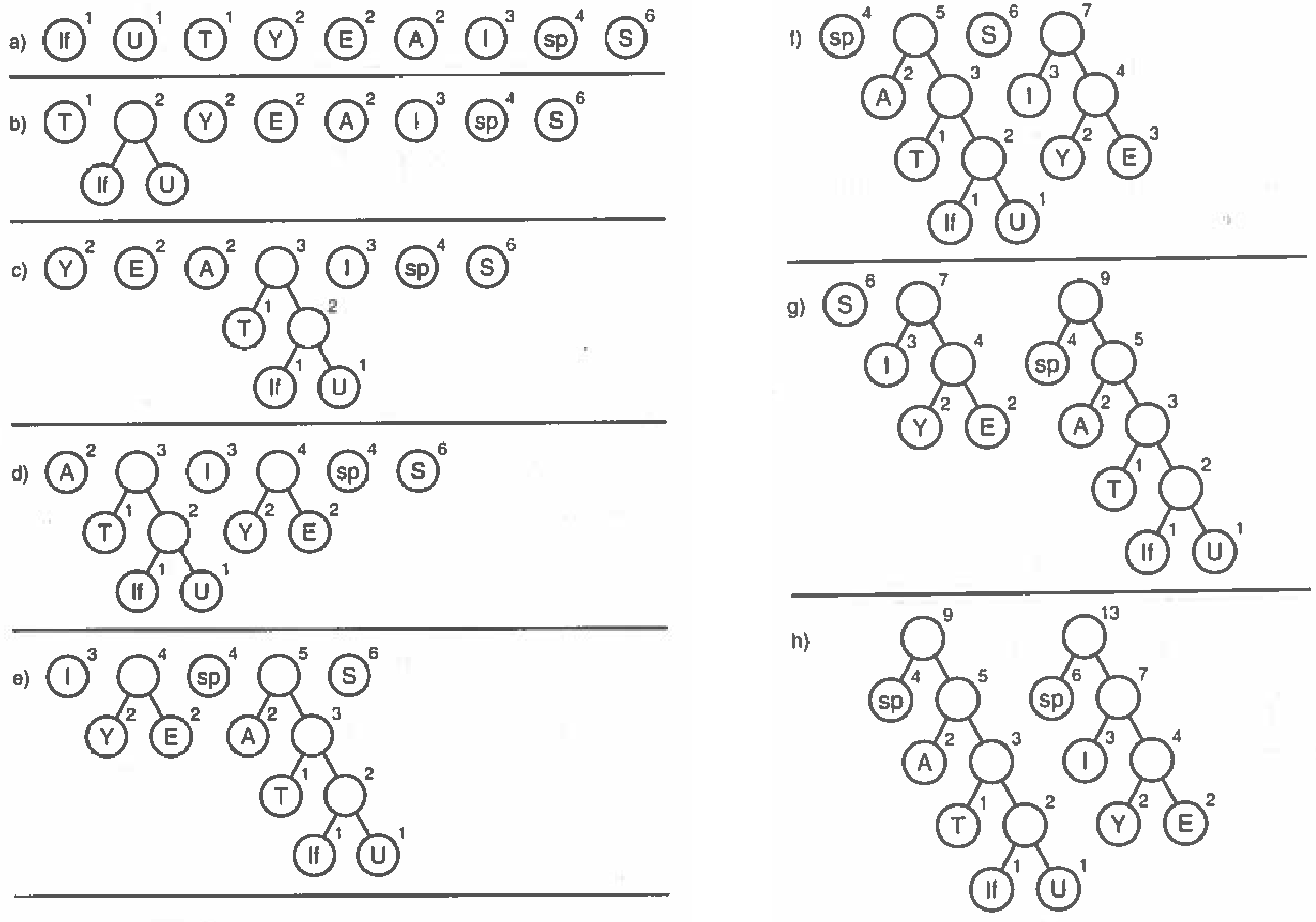
Initial construction:

1) Create nodes containing character and frequency.
2) Create trees with such nodes as roots.
3) Feed the trees into a priority queue.

Tree Construction:

1) Remove two trees from the priority queue. These 2 trees become children of a new node. The new node has a frequency equal to the sum of the children's frequencies and a blank character/frequency field.
2) Insert this new tree in the priority queue.
3) Repeat 1) and 2) until only one tree is left in the queue. Such tree is the Huffman decoding tree!

How to create the Decoding Tree



Coding

Once you have the coding tree, the coding step is simple.
You have just to create a table which associates a character with the right Huffman code.

For doing this, you can realize a tree traversal algorithm which keeps track of the path followed from the root to a leaf. The path will represent the Huffman code for the character contained in the leaf.