# Lecture # 15

# Uses of Binary Trees

Expression Trees

# Expression Trees

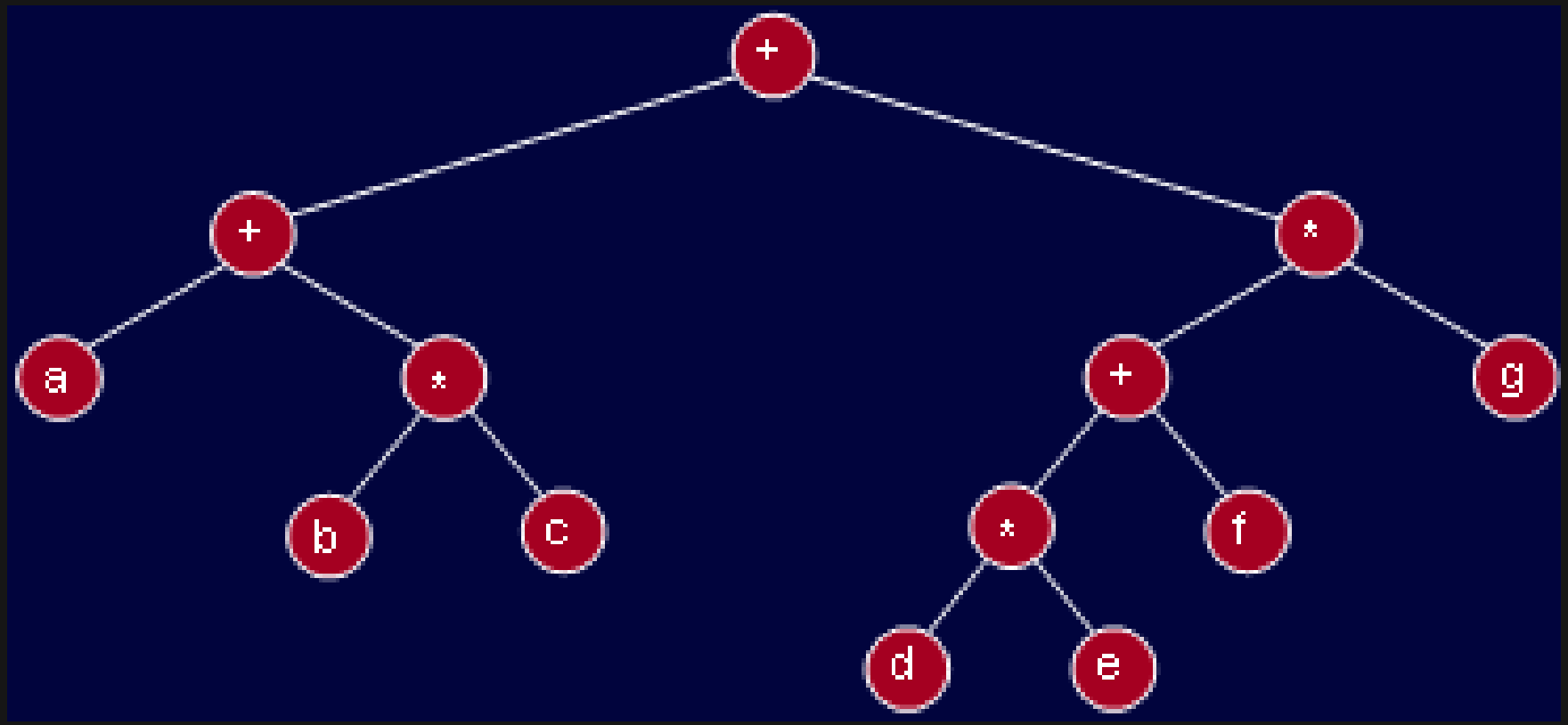
- *Expression trees*, and the more general parse trees and abstract syntax trees are significant components of *compilers*.

- Compilers usually transforms high level language code into assembly language and then assembler transforms it into machine language which is understandable to low level machine.

- Let us consider the expression tree.

# Expression Tree

- (a+b*c)+((d*e+f)*g)

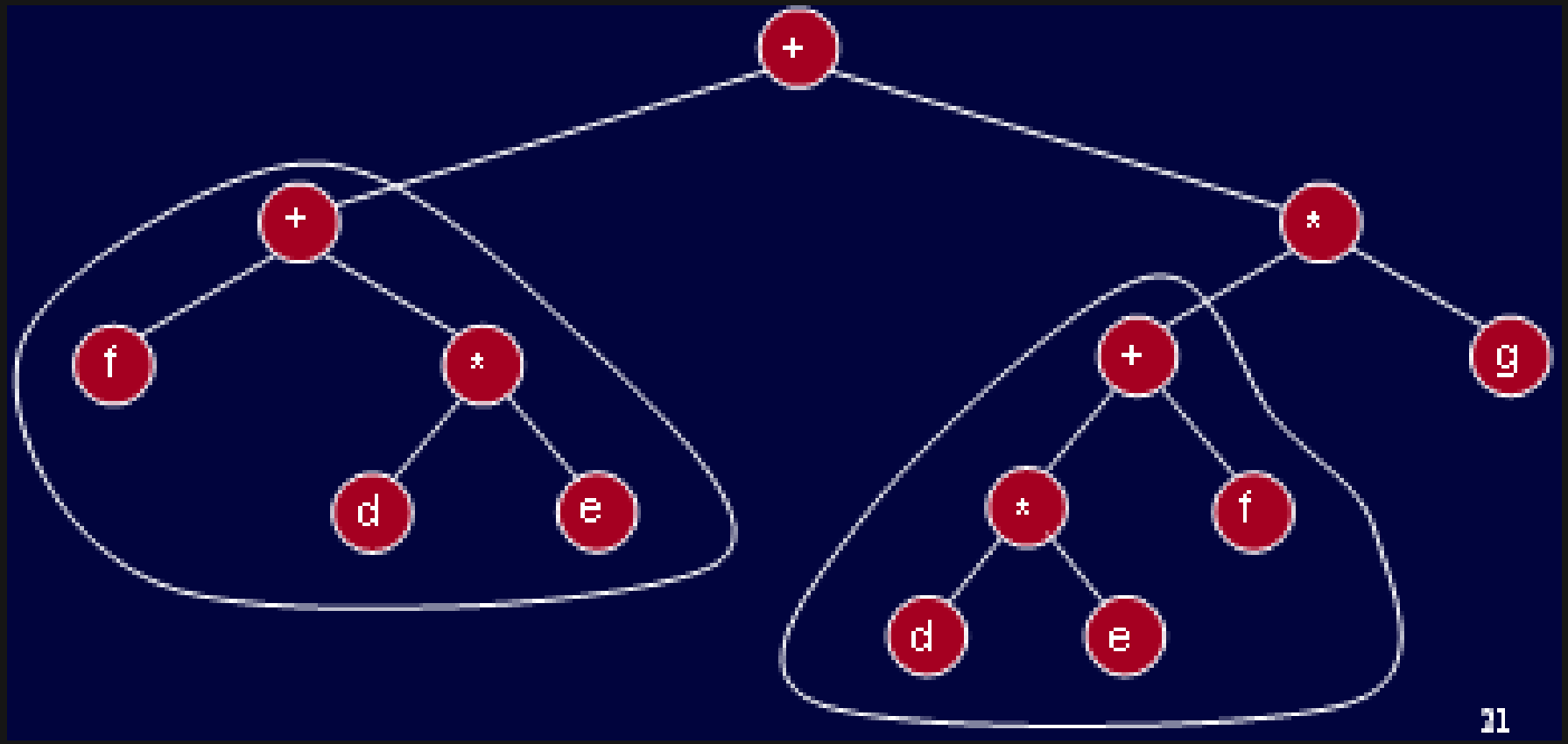


# Expression Tree

- Expression trees are generated inside compilers as in previous example.
- Another usage is in spreadsheet software where we write formula for a cell and at the back end the expression tree for that formula is generated.

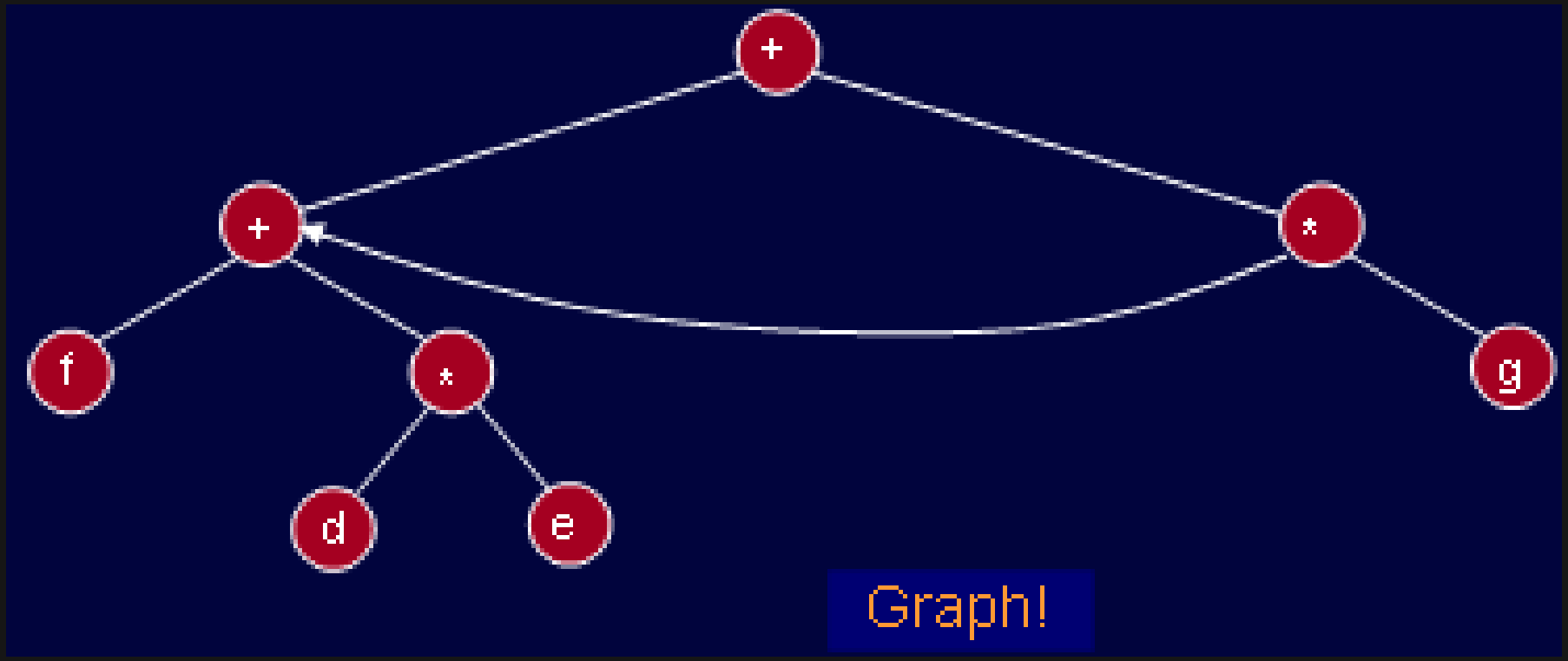
# Compiler Optimization

- Common subexpression:
  (f+d*e)+((d*e+f)*g)



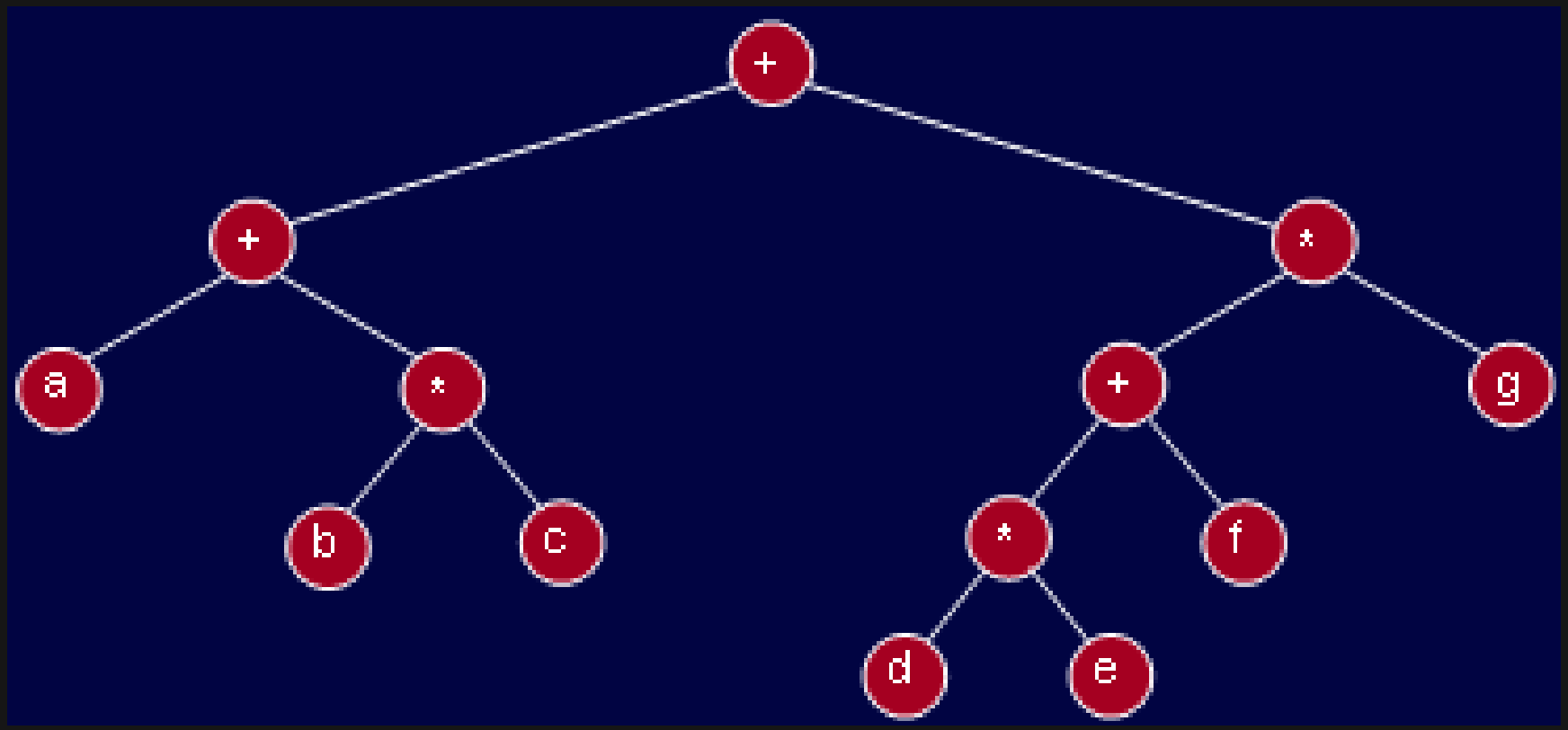
# Compiler Optimization

- Common subexpression:
  (f+d*e)+((d*e+f)*g)



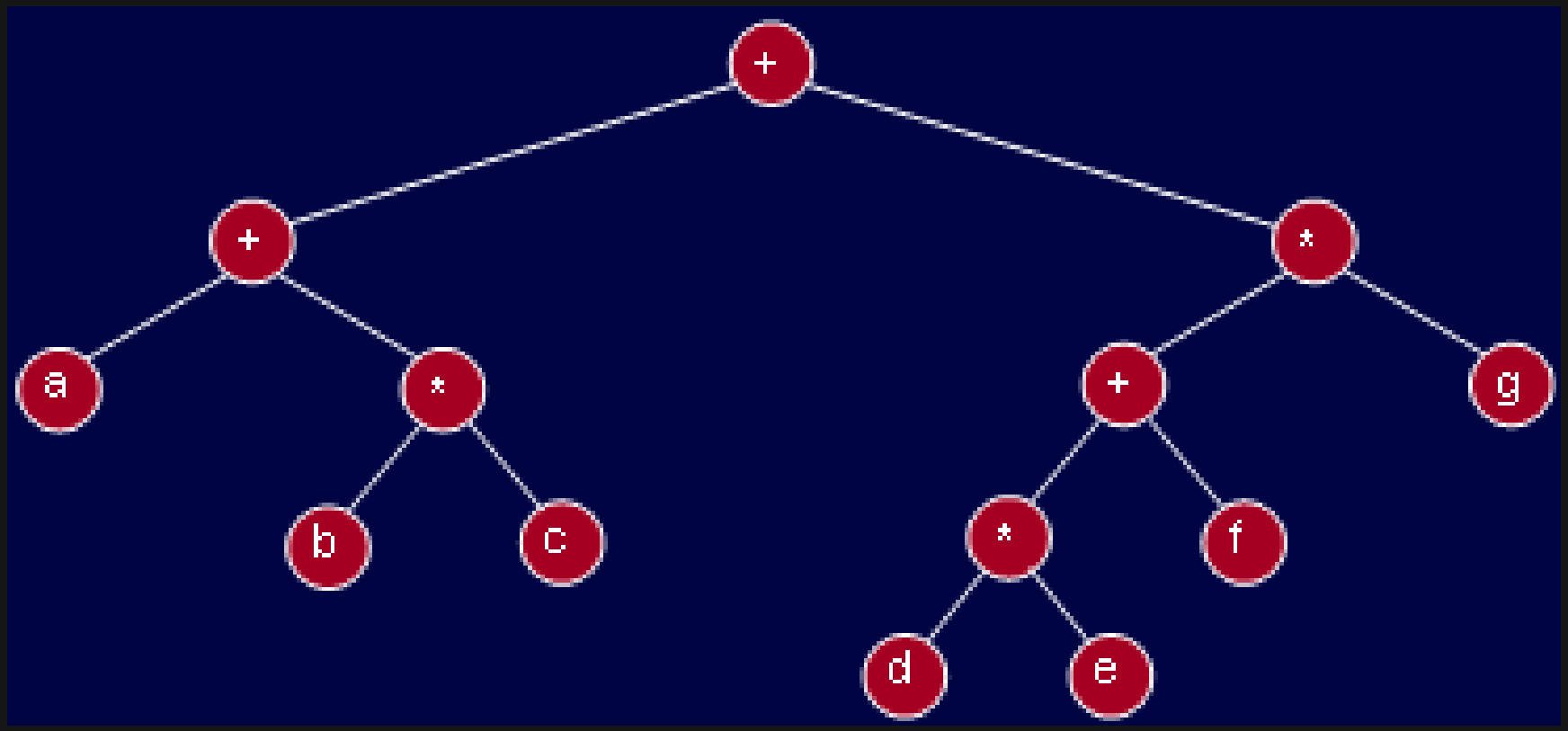
# Expression Tree

- The inner nodes contain operators while leaf nodes contain operands.



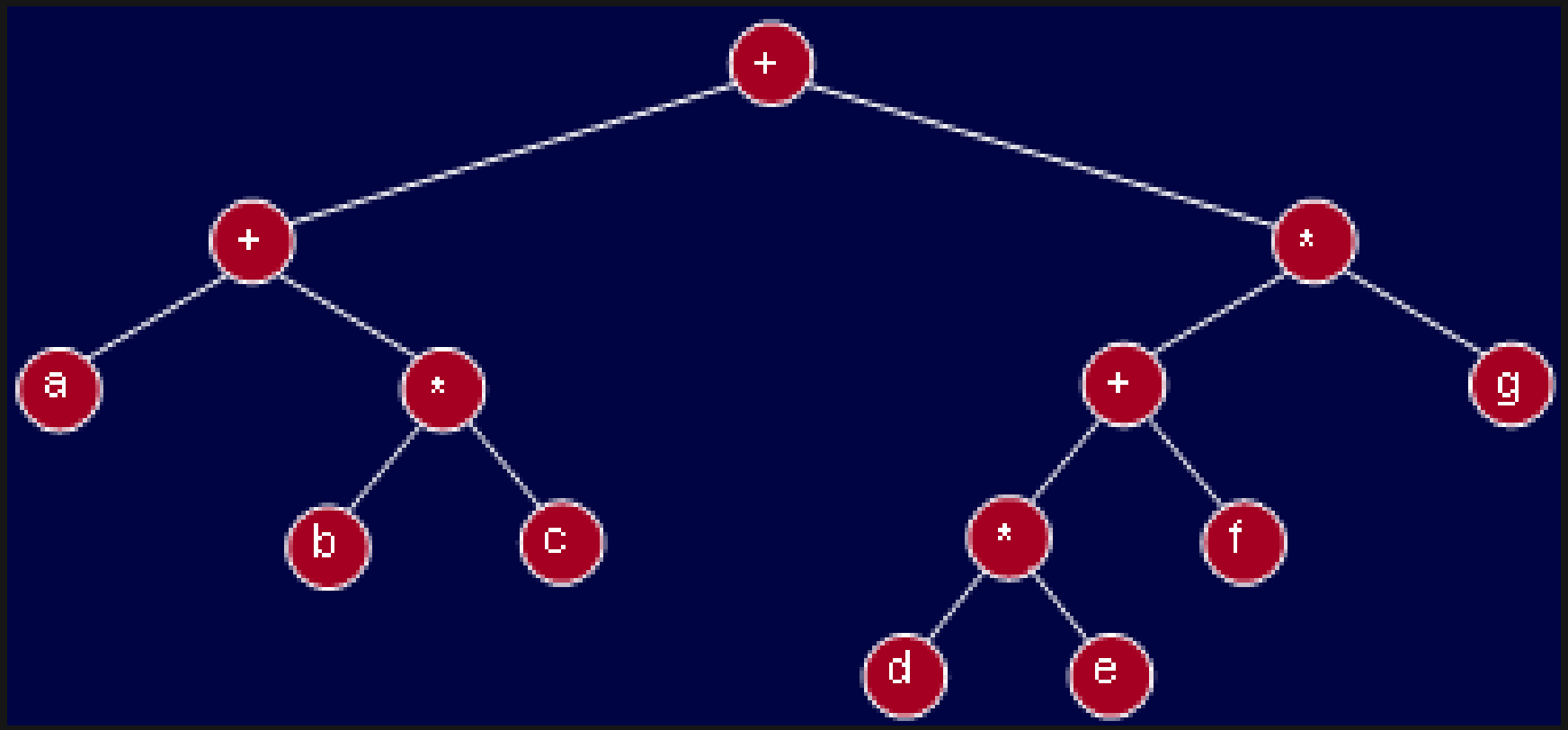
# Expression Tree

- The tree is binary because the operators are binary.



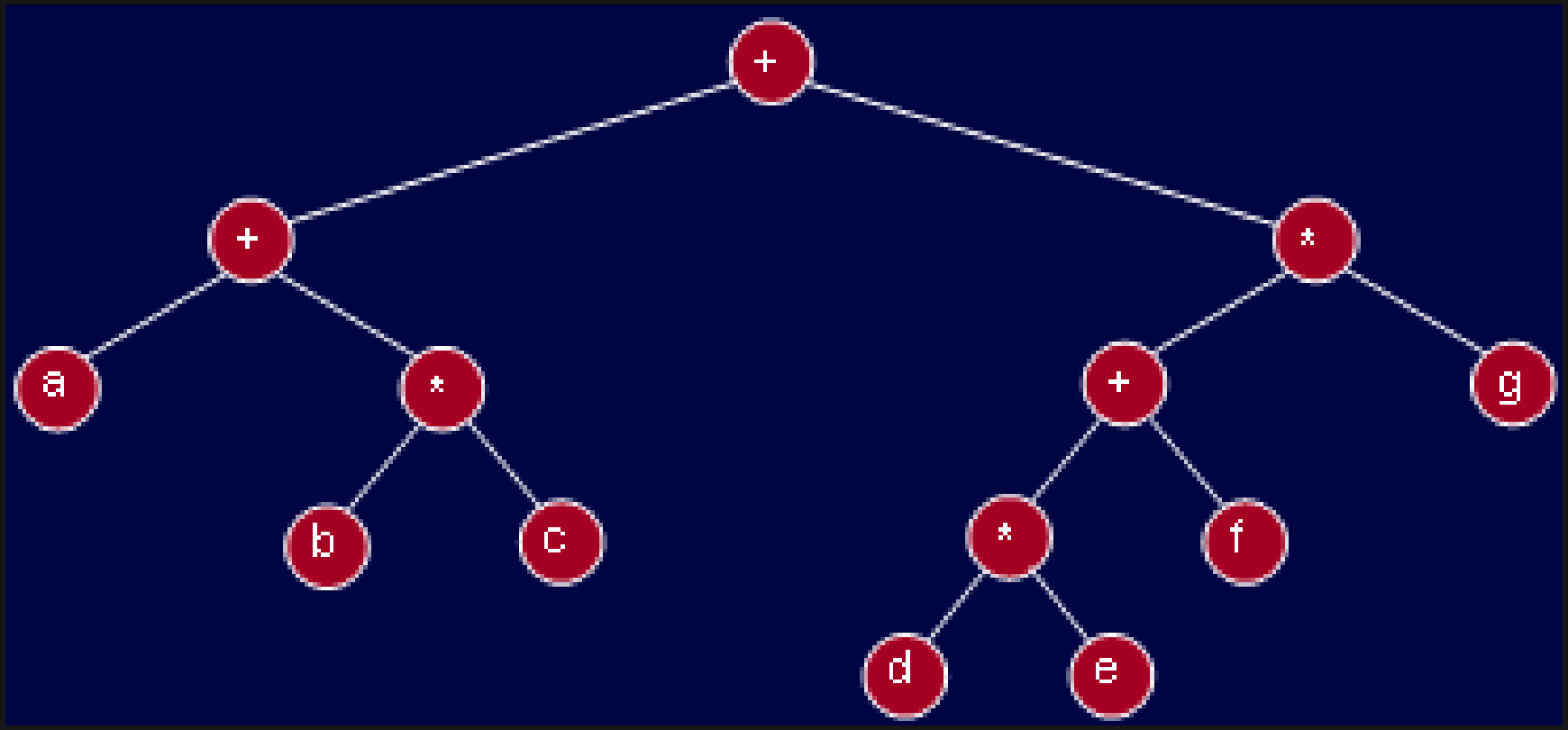
# Expression Tree

- This is not necessary. A unary operator (!, e.g.) will have only one subtree.



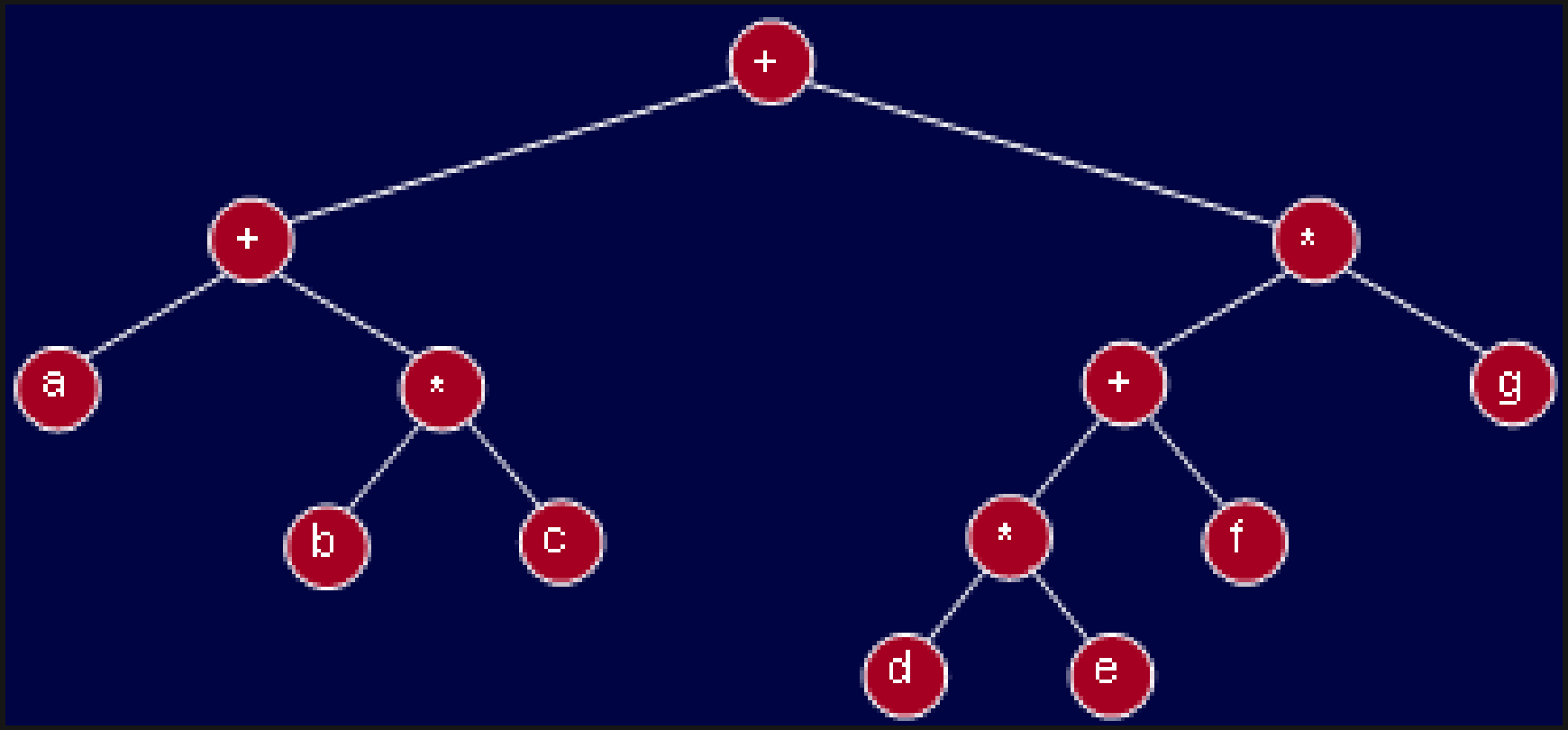
# Expression Tree

- In-order traversal yields: a+b*c+d*e+f*g



# Expression Tree

- Postorder traversal: a b c * + d e * f + g * +
  which is the *postfix form*.

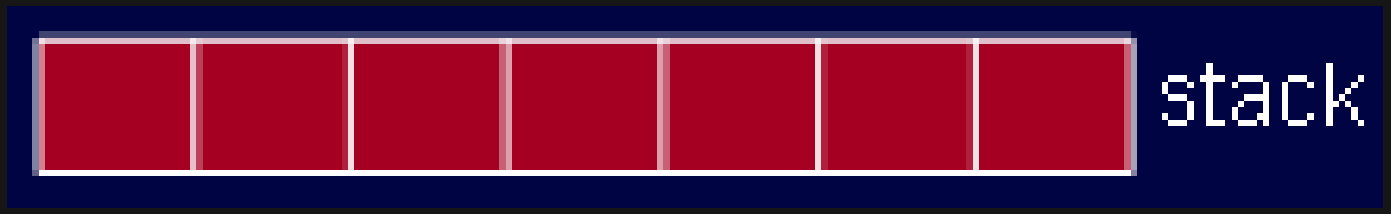


# Constructing Expression Tree

- Algorithm to convert postfix expression into an expression tree.
- We already have an expression to convert an infix expression to postfix.
- Read a symbol from the postfix expression.
- If symbol is an operand, put it in a one node tree and push it on a stack.
- If symbol is an operator, pop two trees from the stack, form a new tree with operator as the root and T1 and T2 as left and right subtrees and push this tree on the stack.

# Constructing Expression Tree

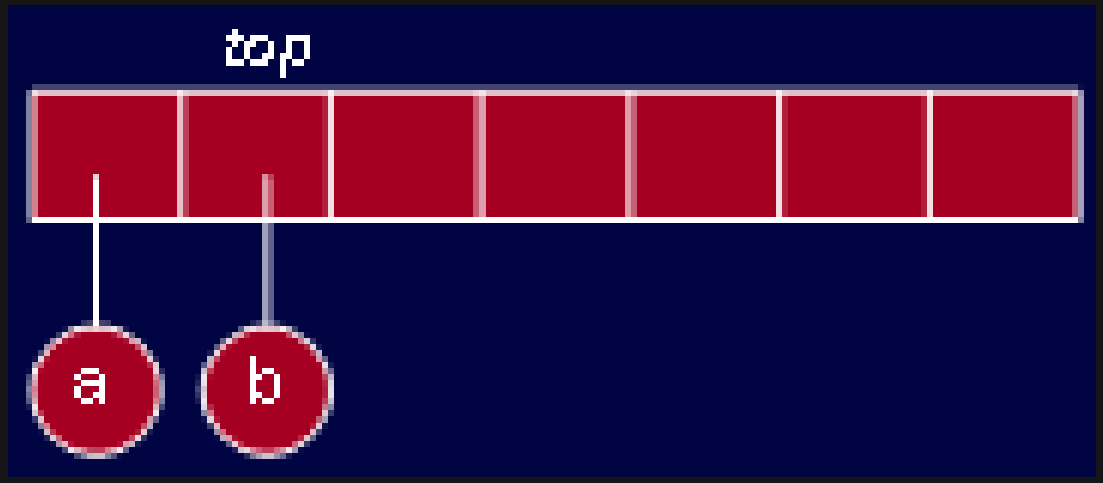
- a b + c d e + * *




*Stack is growing left to right*

# Constructing Expression Tree

- a b + c d e + * *

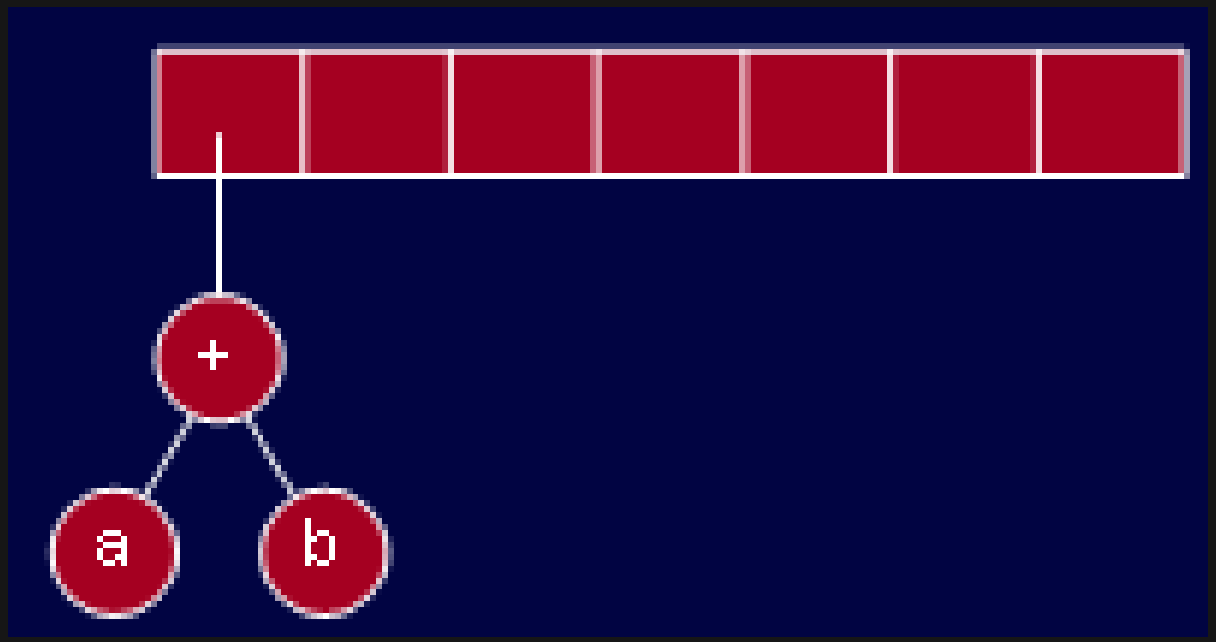



- if symbol is an operand, put it in a one node tree and push it on a stack

*Stack is growing left to right*

# Constructing Expression Tree

- a b + c d e + * *

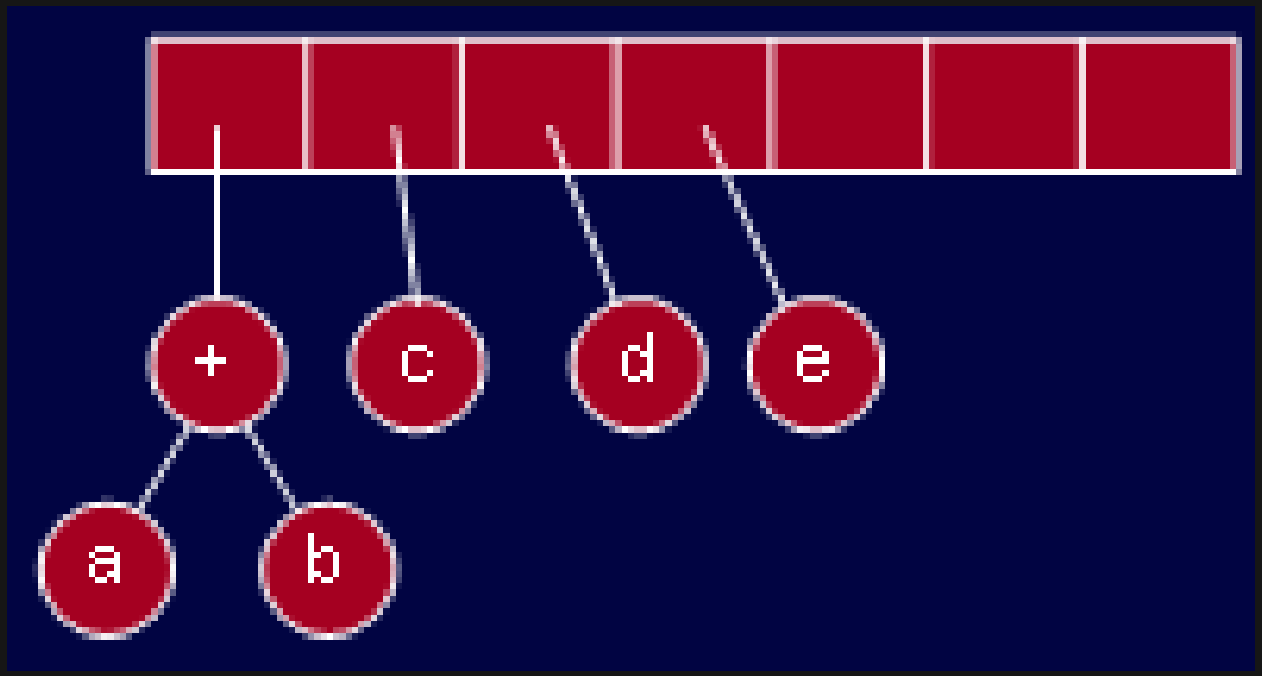



*Stack is growing left to right*

- If symbol is an operator, pop two trees from the stack, form a new tree with operator as the root and T1 and T2 as left and right subtrees and push this tree on the stack.

# Constructing Expression Tree

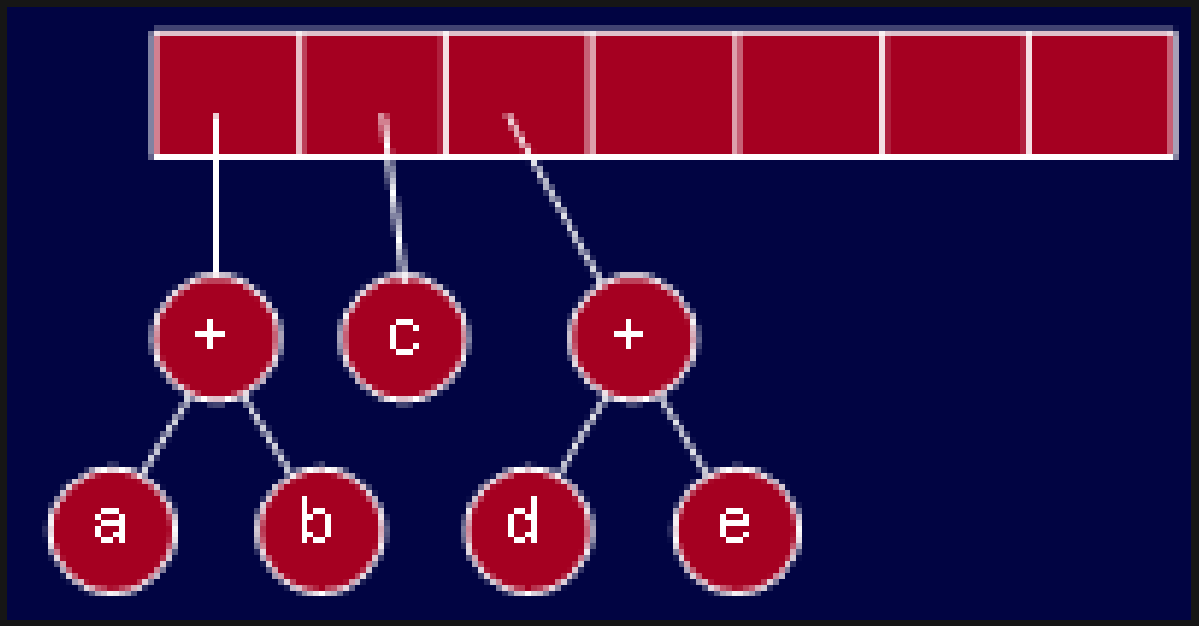
- a b + c d e + * *




*Stack is growing left to right*

# Constructing Expression Tree

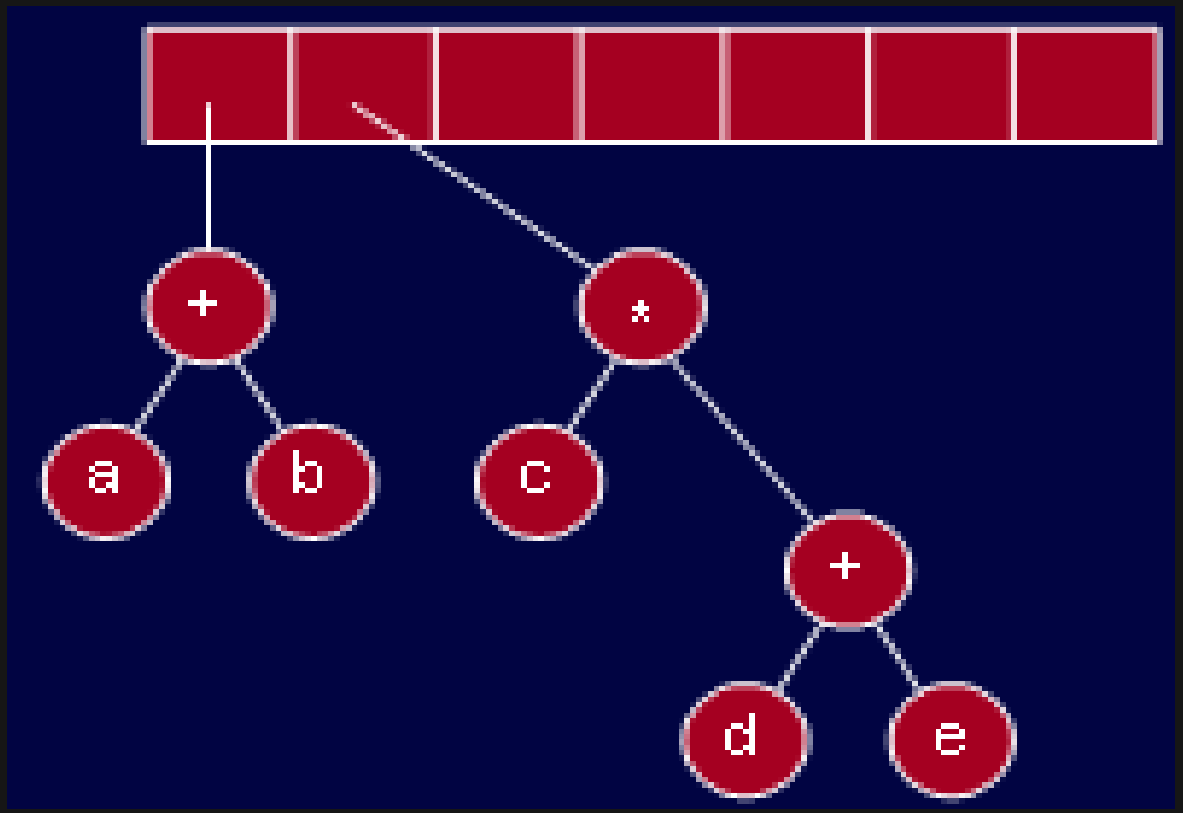
- a b + c d e + * *




*Stack is growing left to right*

# Constructing Expression Tree

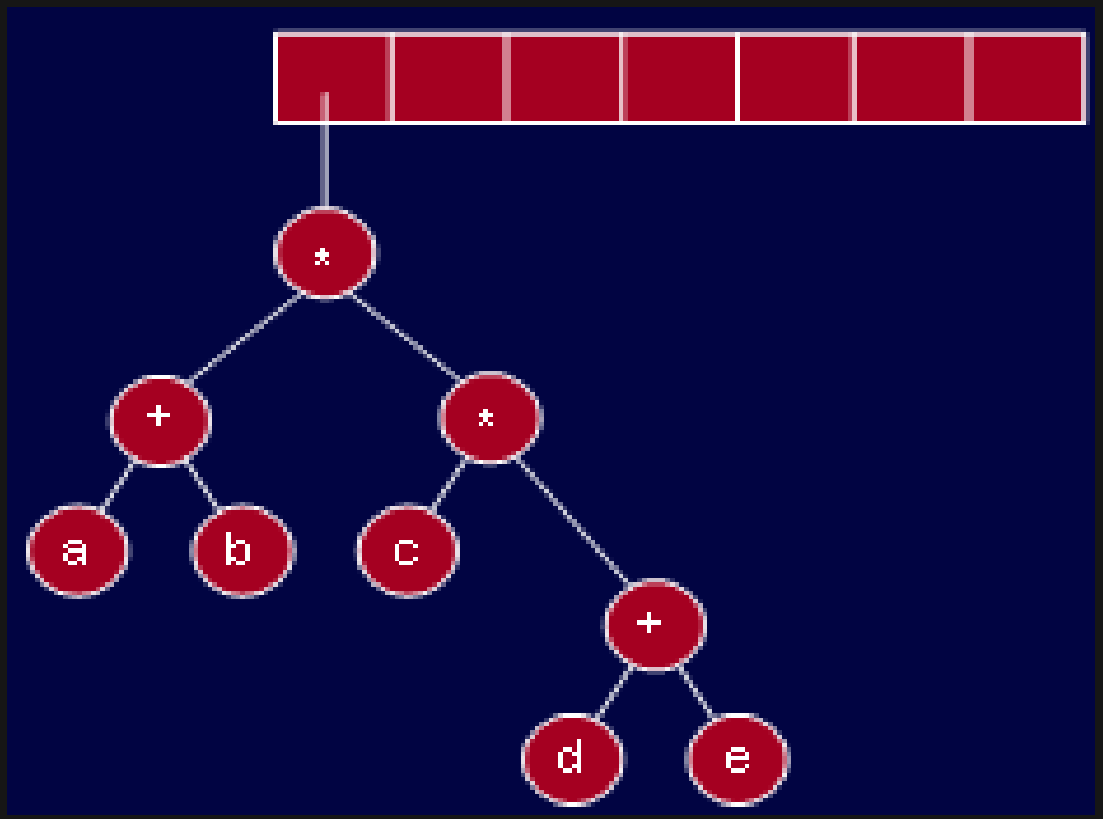
- a b + c d e + * *




*Stack is growing left to right*

# Constructing Expression Tree

- a b + c d e + * *




*Stack is growing left to right*

# Other Uses of Binary Trees

Huffman Encoding

# Huffman Encoding

- Data compression plays a significant role in computer networks.
- To transmit data to its destination faster, it is necessary to either increase the data rate of the transmission media or to simply send less data.
- Improvements with regard to the transmission media has led to increase in the rate.
- The other options is to send less data by means of data compression.
- Compression methods are used for text, images, voice and other types of data.

# Huffman Encoding

- Huffman code is method for the compression for standard text documents.
- It makes use of a binary tree to develop codes of varying lengths for the letters used in the original message.
- Huffman code is also part of the JPEG image compression scheme.
- The algorithm was introduced by David Huffman in 1952 as part of a course assignment at MIT.

# Huffman Encoding

- To understand Huffman encoding, it is best to use a simple example.
- Encoding the 32-character phrase:

  "*traversing threaded binary trees*",

- If we send the phrase as a message in a network using standard 8-bit ASCII codes, we would have to send 8*32= 256 bits.
- Using the Huffman algorithm, we can send the message with only 116 bits.

# Huffman Encoding

- List all the letters used, including the "space" character, along with the frequency with which they occur in the message.

- Consider each of these (character,frequency) pairs to be nodes; they are actually leaf nodes, as we will see.
- Pick the two nodes with the lowest frequency, and if there is a *tie*, pick randomly amongst those with equal frequencies.

# Huffman Encoding

- Make a new node out of these two, and make the two nodes its children.
- This new node is assigned the sum of the frequencies of its children.
- Continue the process of combining the two nodes of lowest frequency until only one node, the root, remains.

# Huffman Encoding

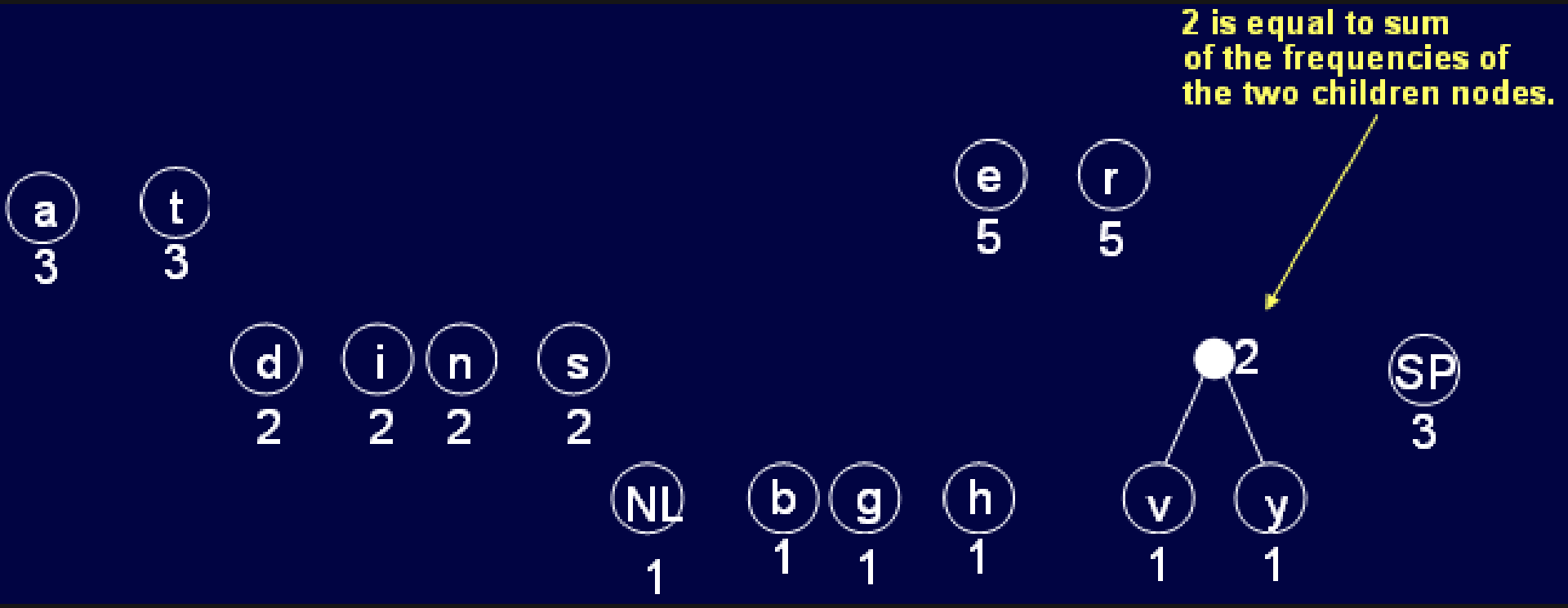
- Original text:

  *traversing threaded binary trees*

- size: 33 characters (space and newline)

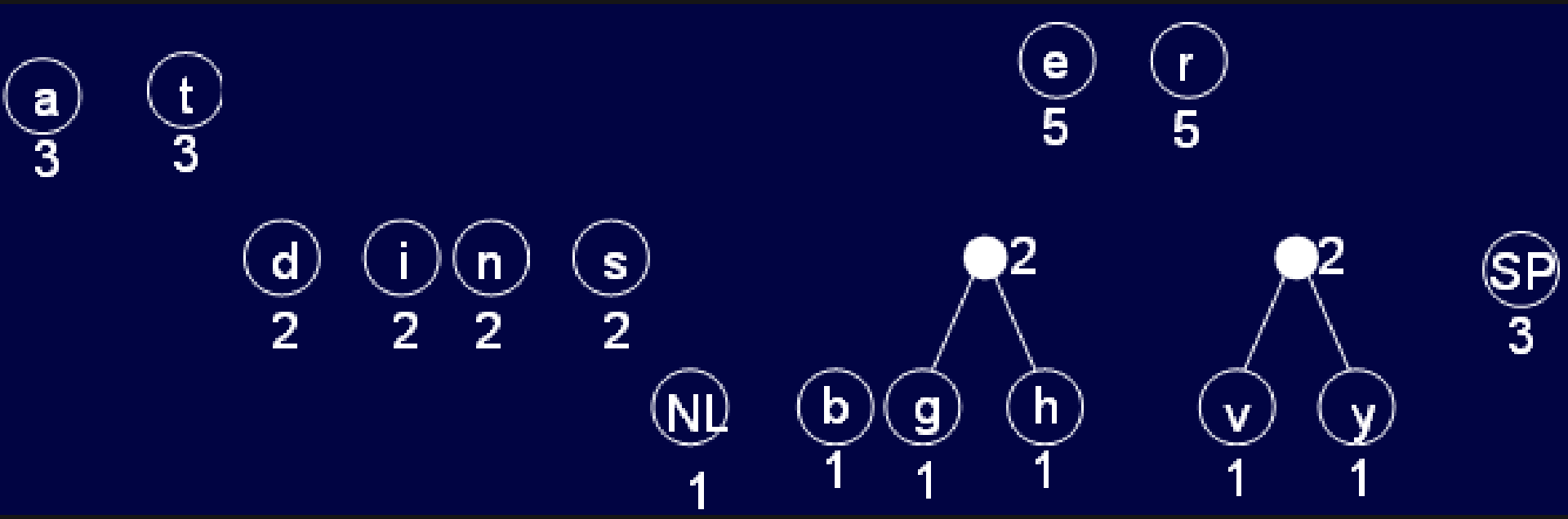
| | | | |
|---|---|---|---|
| NL: | 1 | i: | 2 |
| SP: | 3 | n: | 2 |
| a: | 3 | r: | 5 |
| b: | 1 | s: | 2 |
| d: | 2 | t: | 3 |
| e: | 5 | v: | 1 |
| g: | 1 | y: | 1 |
| h: | 1 | | |

# Huffman Encoding

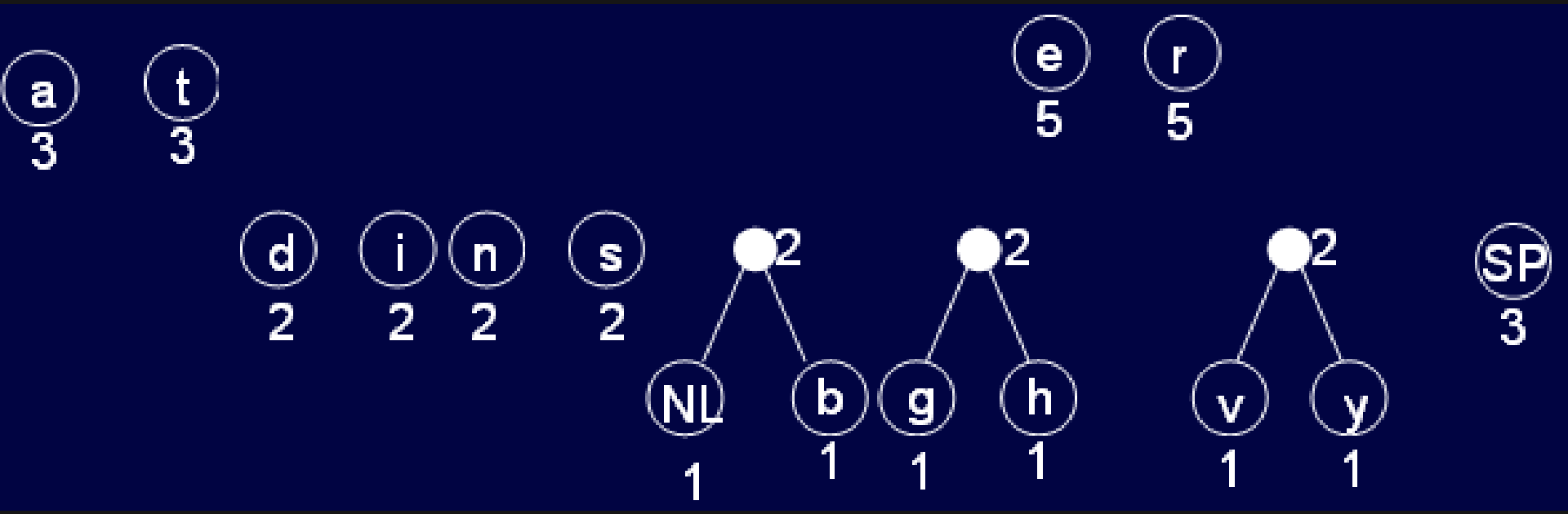


# Huffman Encoding

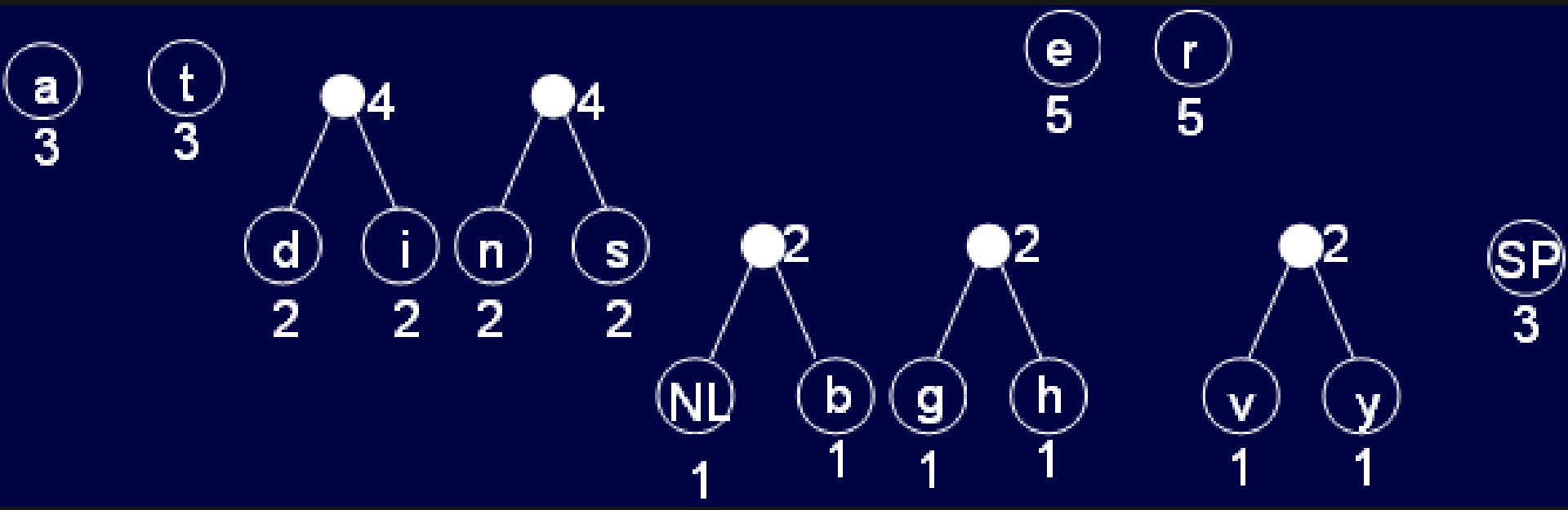
- There a number of ways to combine nodes. We have chosen just one such way.



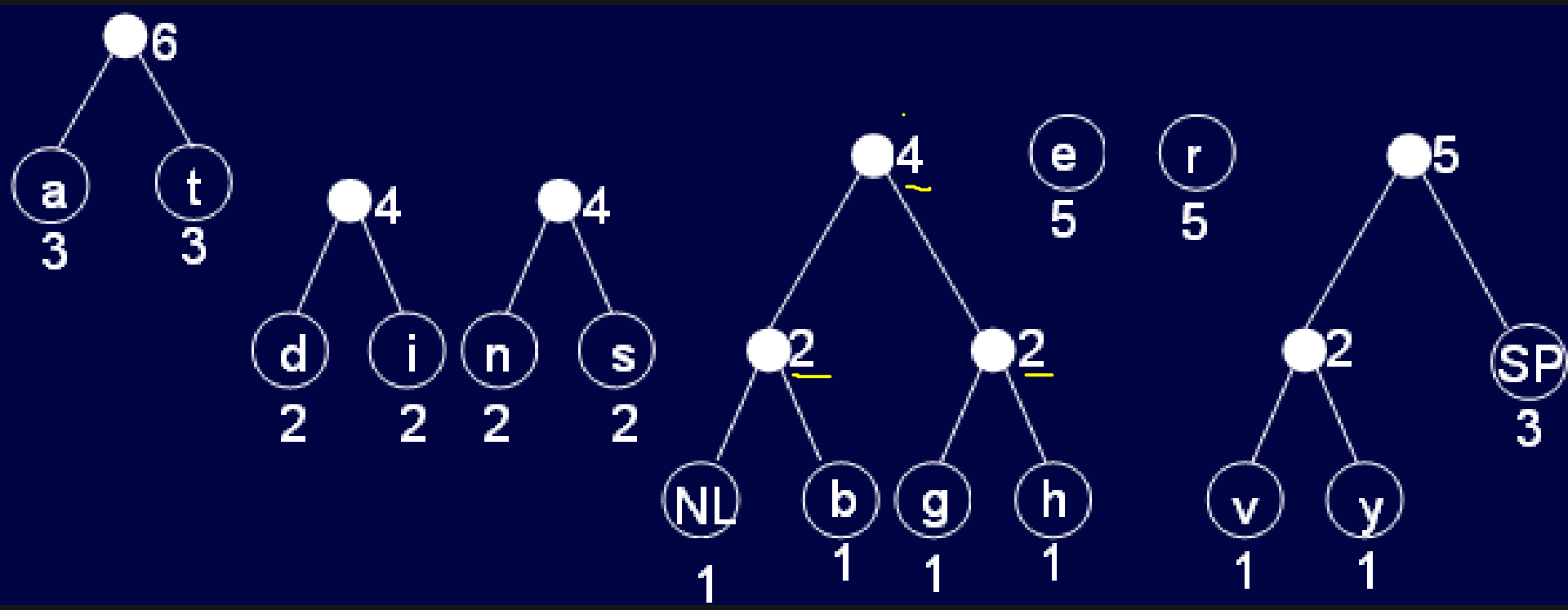
# Huffman Encoding



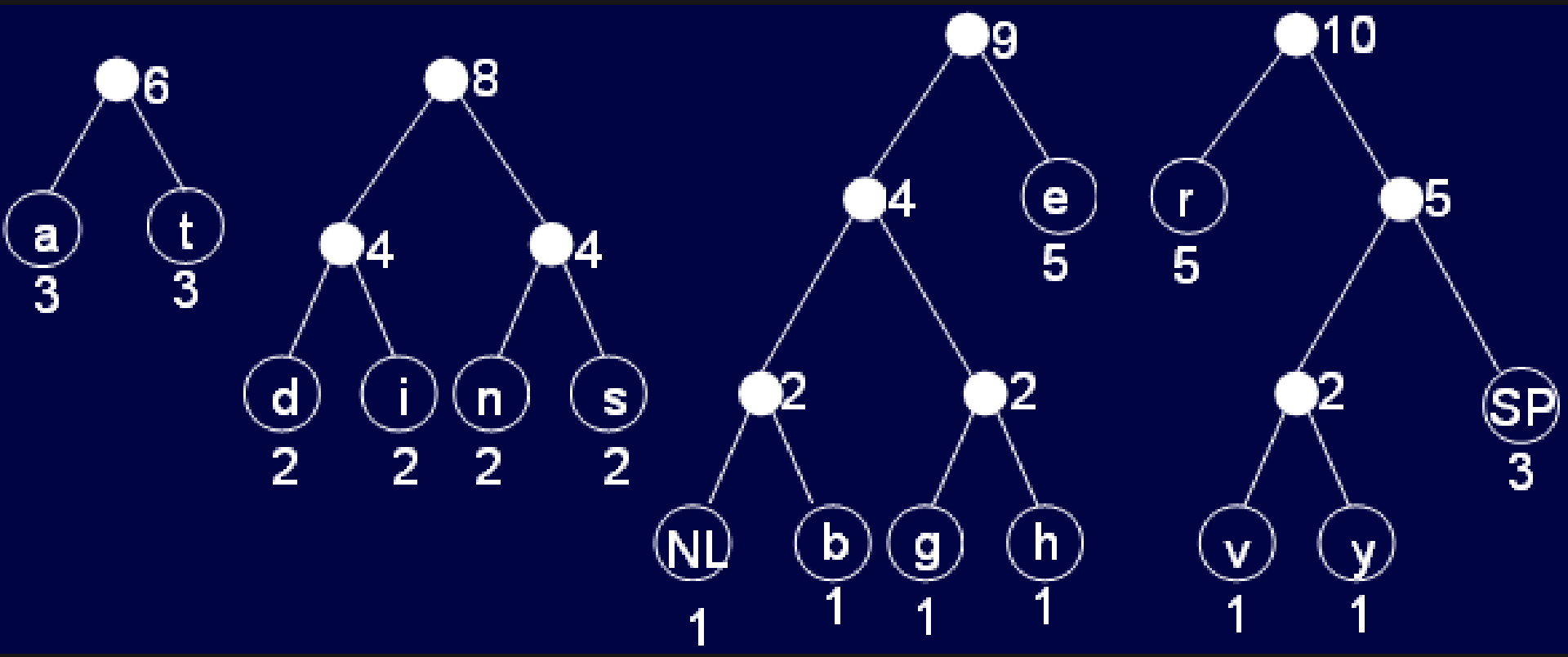
# Huffman Encoding



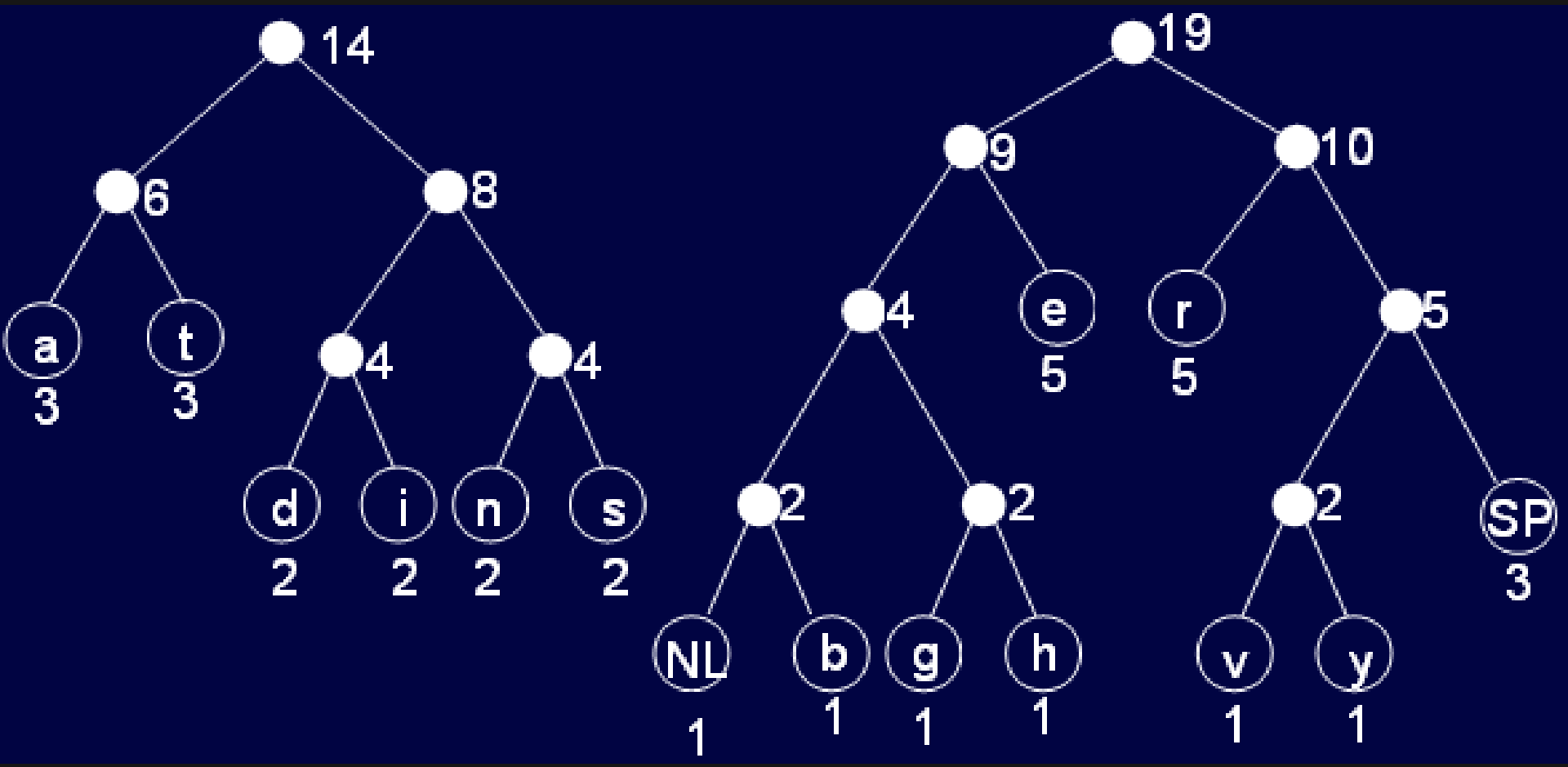
# Huffman Encoding



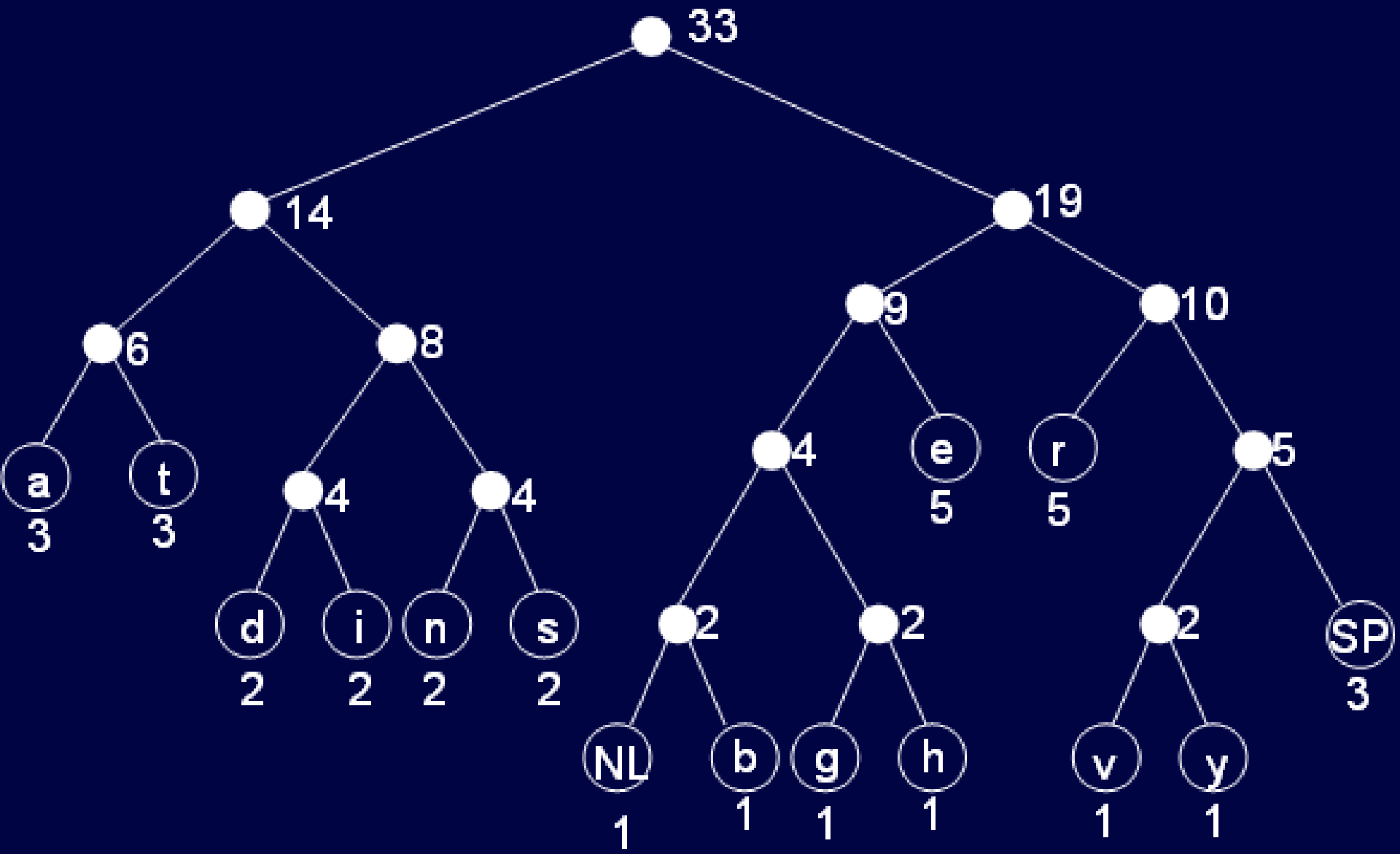
# Huffman Encoding



# Huffman Encoding



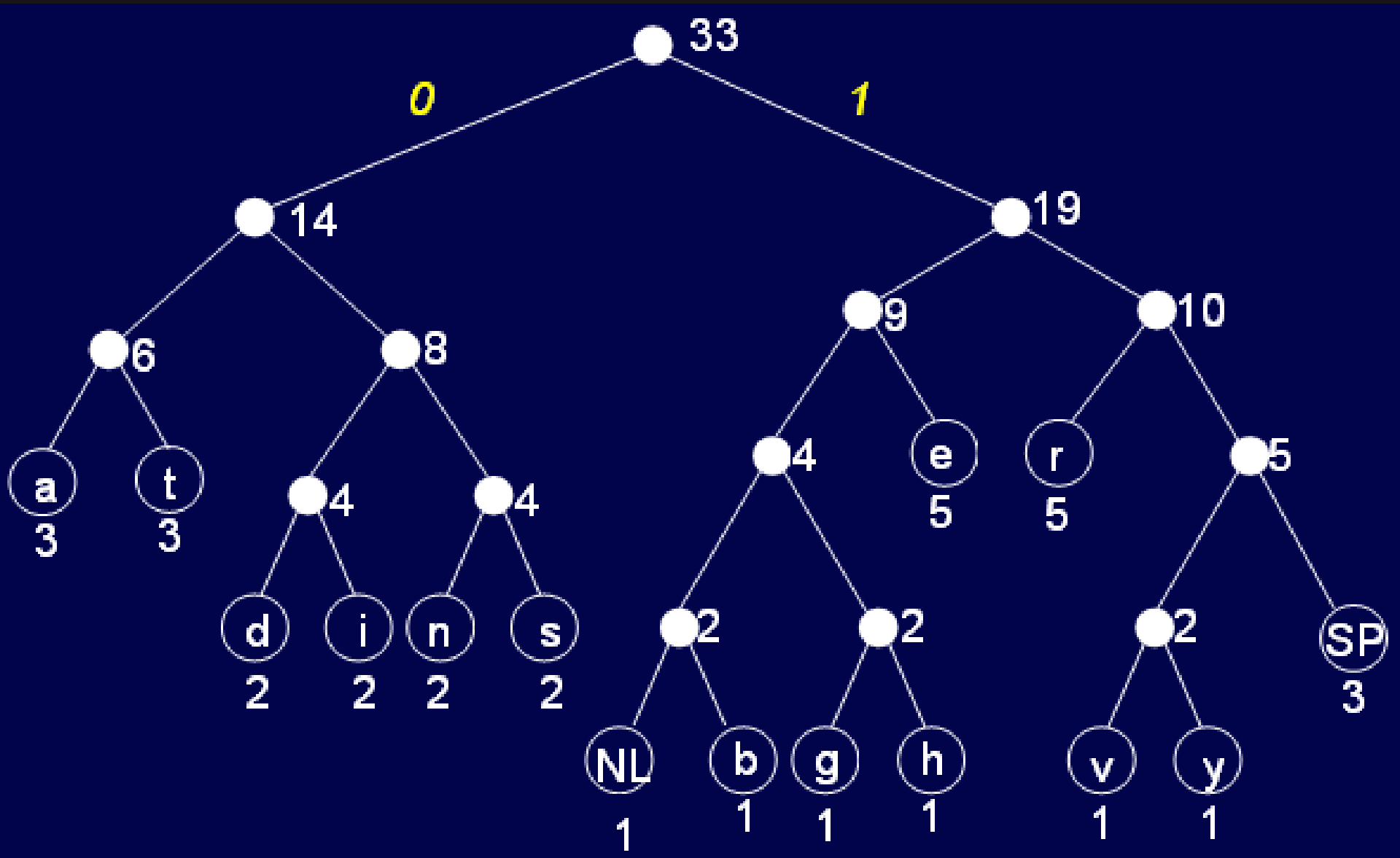
# Huffman Encoding



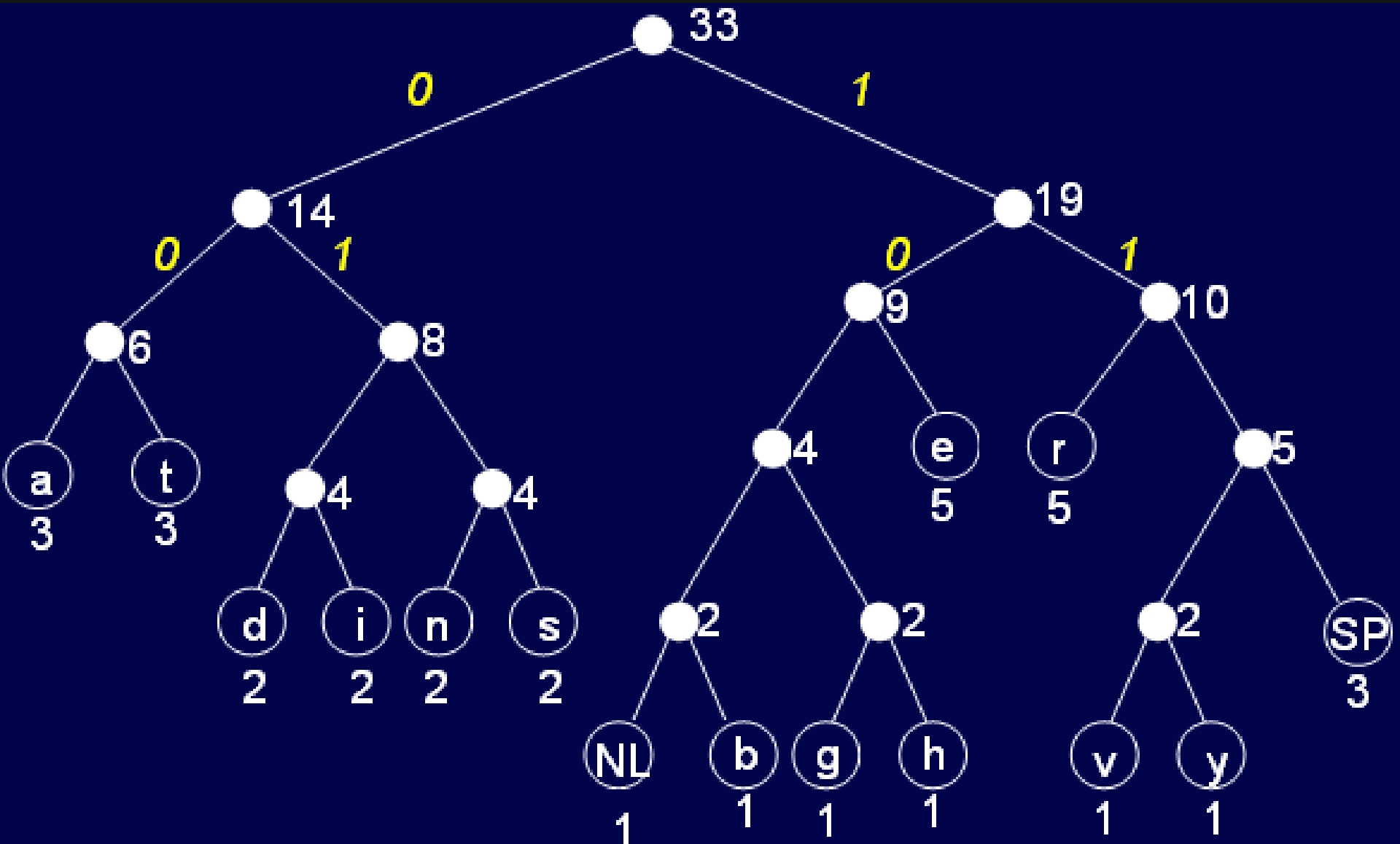
# Huffman Encoding

- Start at the root. Assign 0 to left branch and 1 to the right branch.
- Repeat the process down the left and right subtrees.
- To get the code for a character, traverse the tree from the root to the character leaf node and read off the 0 and 1 along the path

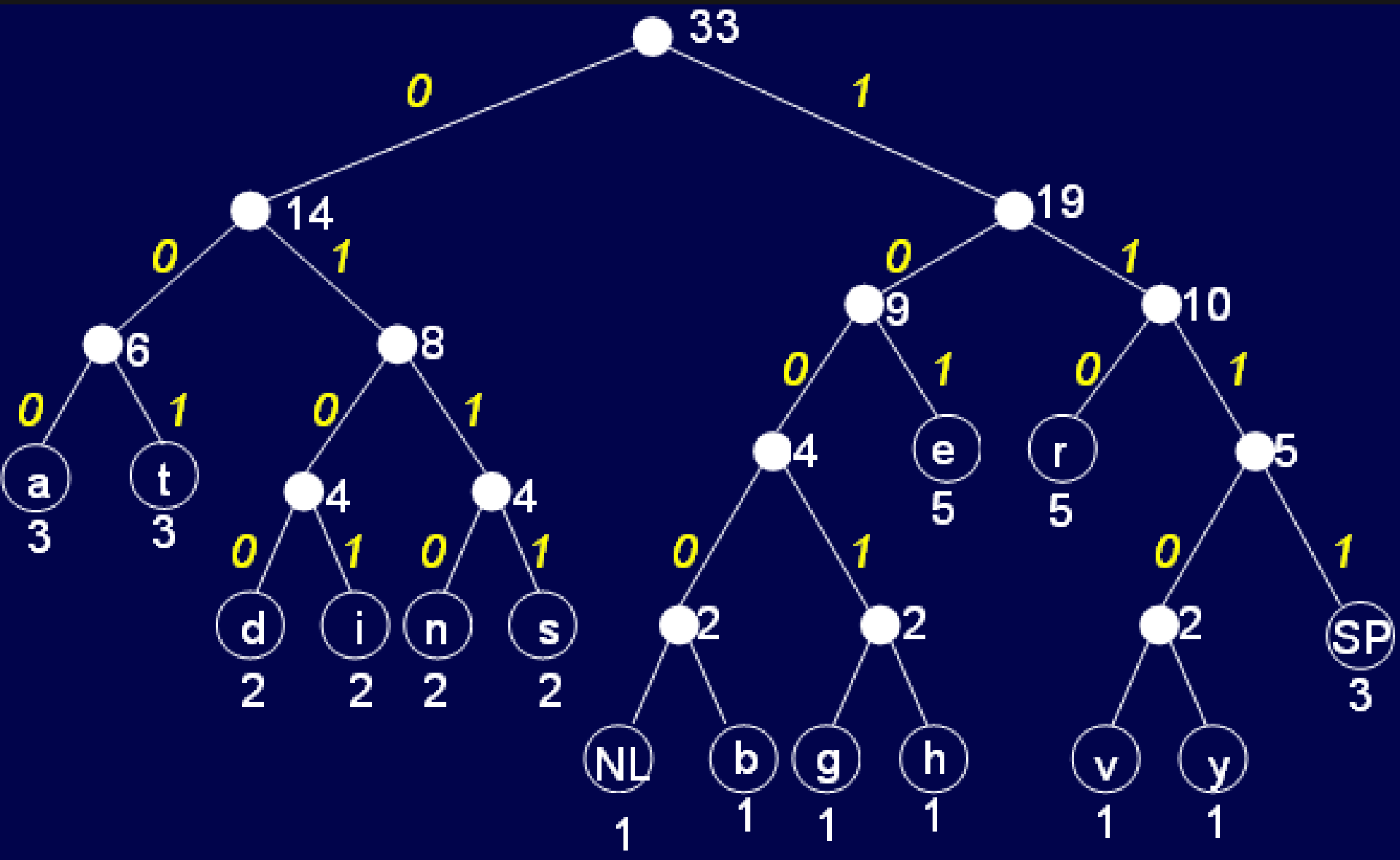
# Huffman Encoding



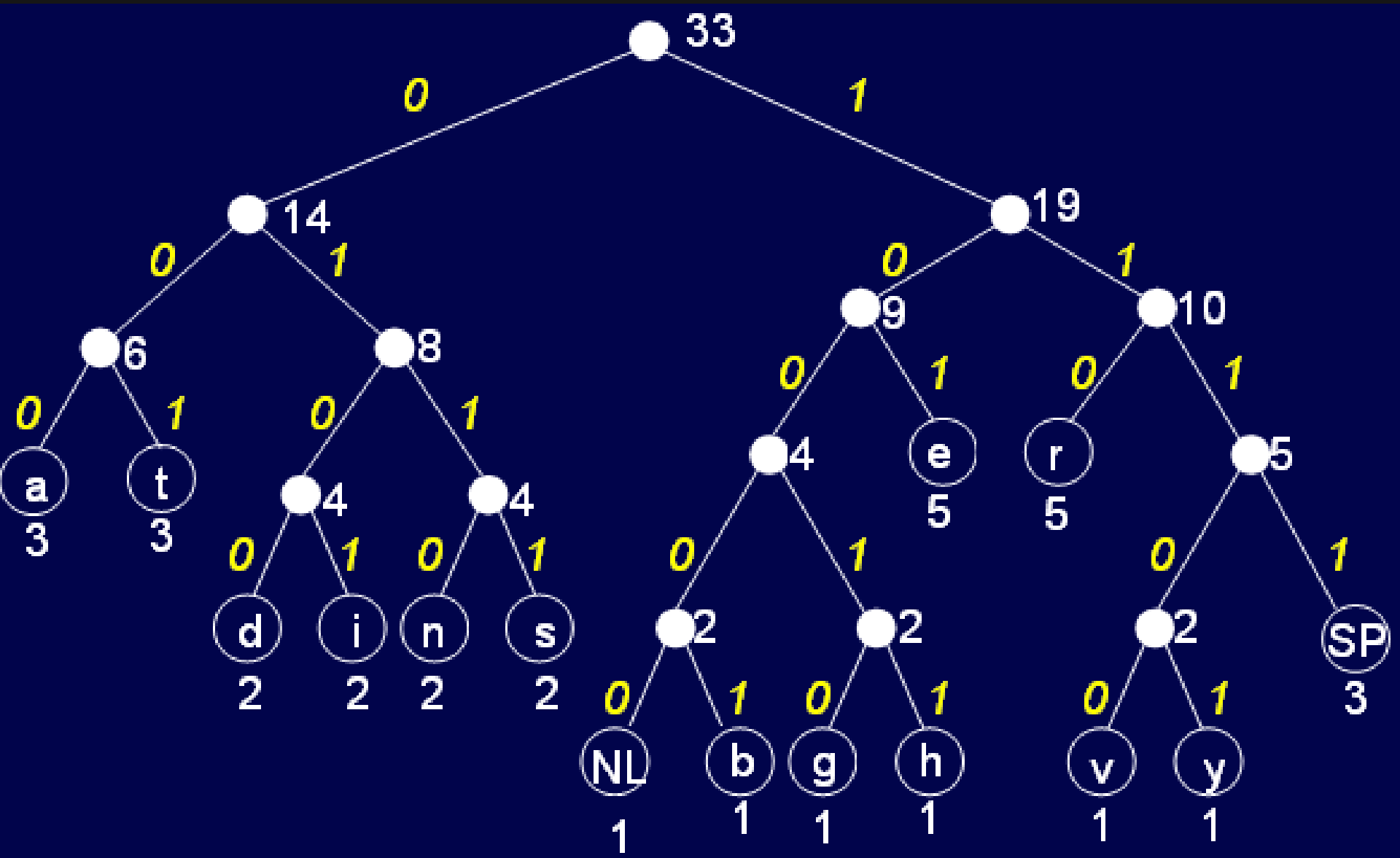
# Huffman Encoding



# Huffman Encoding



# Huffman Encoding



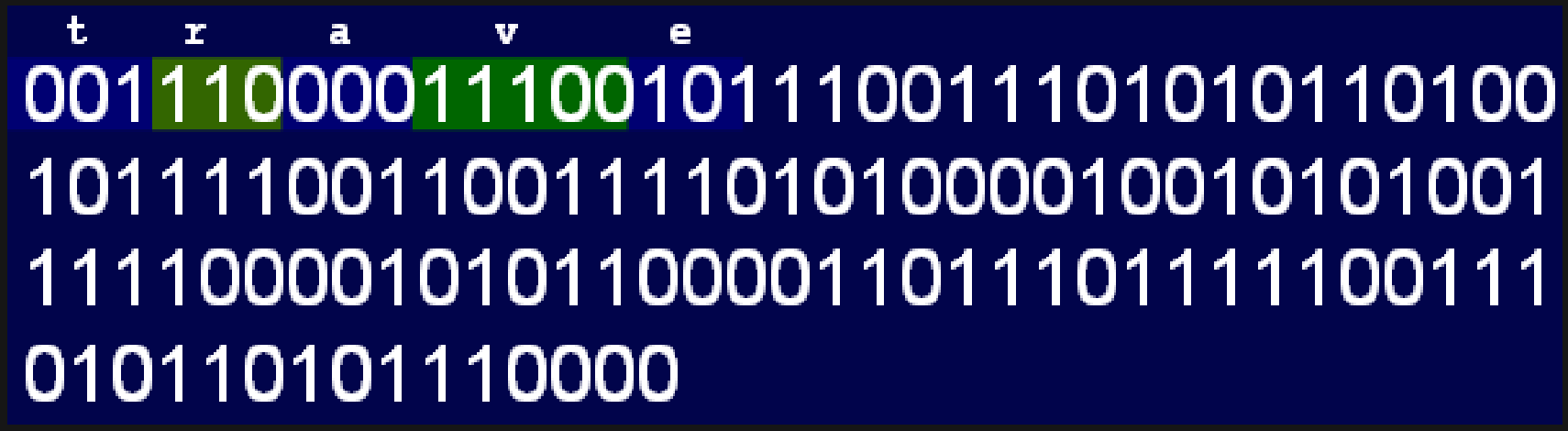
# Huffman Encoding

NL⇨ 10000
SP⇨ 1111
a ⇨ 000
b ⇨ 10001
d ⇨ 0100
e ⇨ 101
g ⇨ 10010
h ⇨ 10011
i ⇨ 0101
n ⇨ 0110
r ⇨ 110
s ⇨ 0111
t ⇨ 001
v ⇨ 11100
y ⇨11101

- Notice that the code is variable length.
- Letters with higher frequencies have shorter codes.
- The tree could have been built in a number of ways; each would have yielded different codes, but the code would still be minimal.

# Huffman Encoding

- Original: *traversing threaded binary trees*
- Encoded:



# Huffman Encoding

- Original: *traversing threaded binary trees* With 8 bits per character, length is 264.
- Encoded:

  00111000011100101110011101010110100
  10111100110011110101000010010101001
  11110000101011000011011101111100111
  010110101110000
- Compressed into 122 bits, 54% reduction.

# Thank You ...