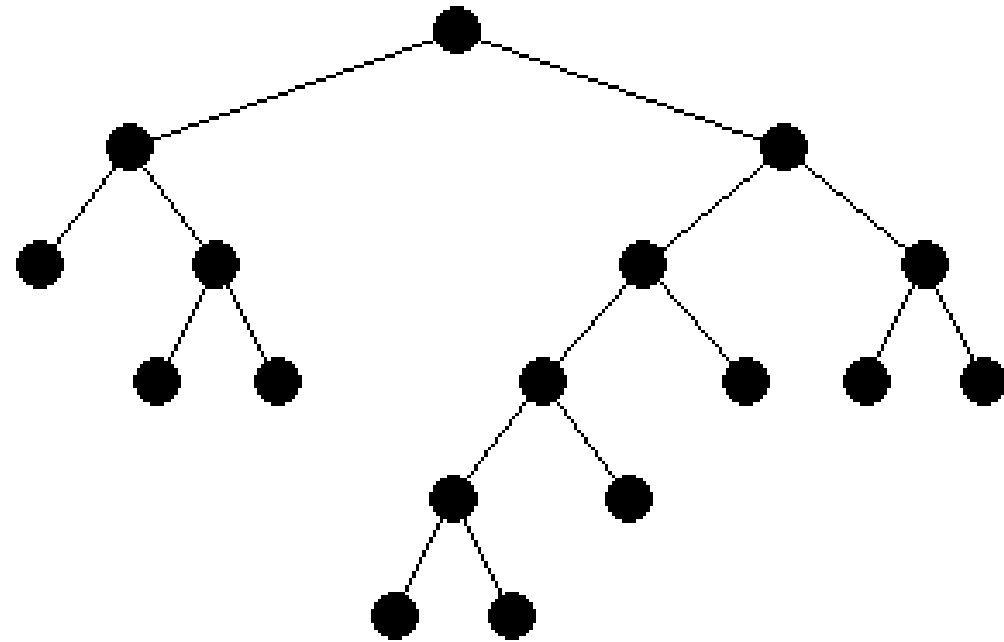


# Algorithm Design: Huffman Codes

# Review

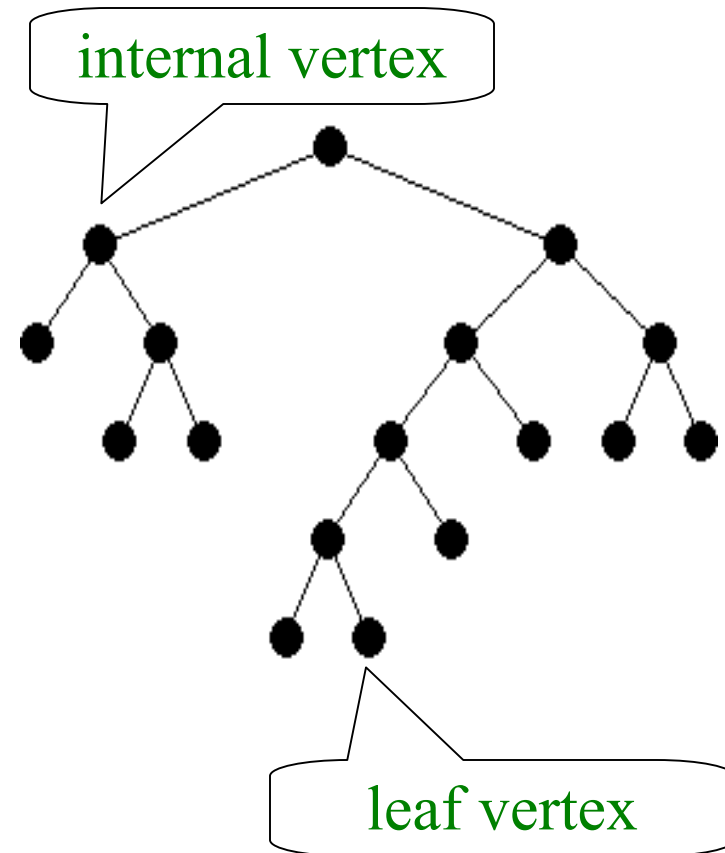
- Algorithms seen so far: sequential search, insertion sort, binary search, Prim's algorithm for “paving roads”
- Existence results: Euler tour, Euler path
- Concepts:
  - comparing algorithms in terms of their worst-case time complexity
  - graph as a model
  - brute-force and greedy algorithms
    - brute-force generally *prohibitively* slow
    - greedy algorithms don't always produce optimal solution (e.g. Traveling Salesman Problem)

# Trees



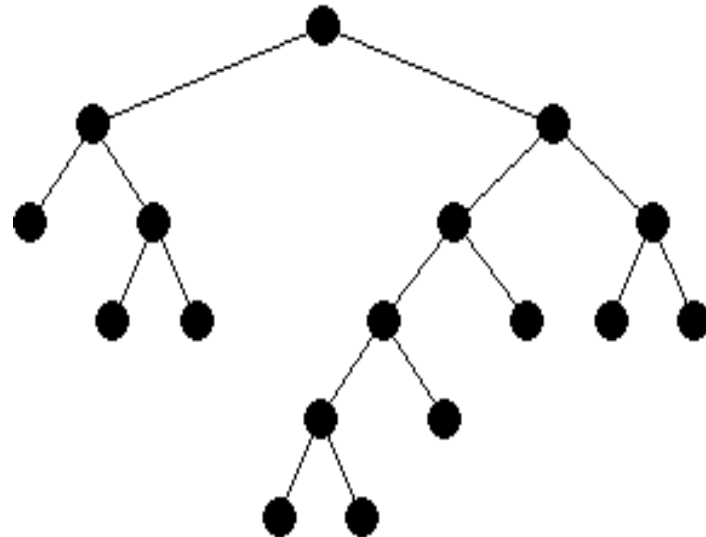
# Trees

- A **tree** is a graph where each vertex is either an
  - **internal** vertex
  - **leaf** vertex
- **Internal** vertices have children: other internal vertices or leaf nodes
- **Leaf** vertices don't have any children
- Edges in a tree cannot form a “**cycle**” (cannot follow new edges to return to a node already visited)



# Binary Tree

- A **binary tree** is a tree where each internal node has at most 2 children



# Data Compression

- Lossy or lossless
- Lossless:
  - Lempel-Ziv (repeating patterns)
  - .gif (using less colours if possible)
  - triangle encoding
    - run-length encoding (aaaaaa → 6a)
- Lossy:
  - .mp3
  - .jpg

# Data Compression

- Text representation:
  - ASCII: 1 byte per character (fixed length code)

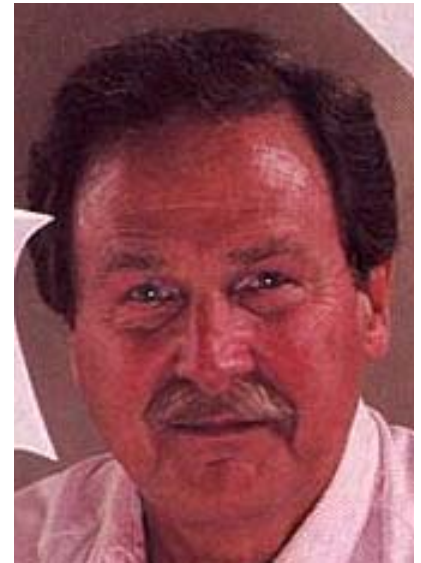
“M                      a                      r                      c”

1st byte	2nd byte	3rd byte	4th byte
77	97	114	99
01001101	01100001	01110010	01100011

- In the English language, the letter ‘e’ is quite common, ‘x’ is less common
- Why should all characters be represented using the same number of bits?

# Huffman Codes

- 1951, David Huffman found the “most efficient method of representing numbers, letters, and other symbols using binary code”
- Now standard method used for data compression



# Huffman Codes

- General idea:
  - assign those characters that occur **more frequently** a **shorter** code
- Would this work?
  - E: 10, T: 7, O: 5, A: 3 (characters and their frequencies)
  - E: 0, T: 1, O: 00, A: 01 (characters and their codes)
  - How would you decode 01?
    - 01 could either be A or ET!**
    - ...not that easy...

# Huffman Codes

- E: 10, T: 7, O: 5, A: 3 (characters and their frequencies)
- **Bad** code:
  - E: 0, T: 1, O: 00, A: 01 (characters and their codes)
- How about this one?
  - E: 0, T: 10, A: 110, O: 111
  - No code is the *prefix* of another code
  - When reading 100110, this is *unambiguously* TEA

# Huffman Codes

- How to construct Huffman codes?
  - Build a binary tree!
1. Take the characters and their frequencies, and *sort* this list by **increasing** frequency

E: 10, T: 7, O: 5, A: 3 →

A: 3, O: 5, T: 7, E:10

# Huffman Codes

2. All the characters are vertices of the tree:

A: 3

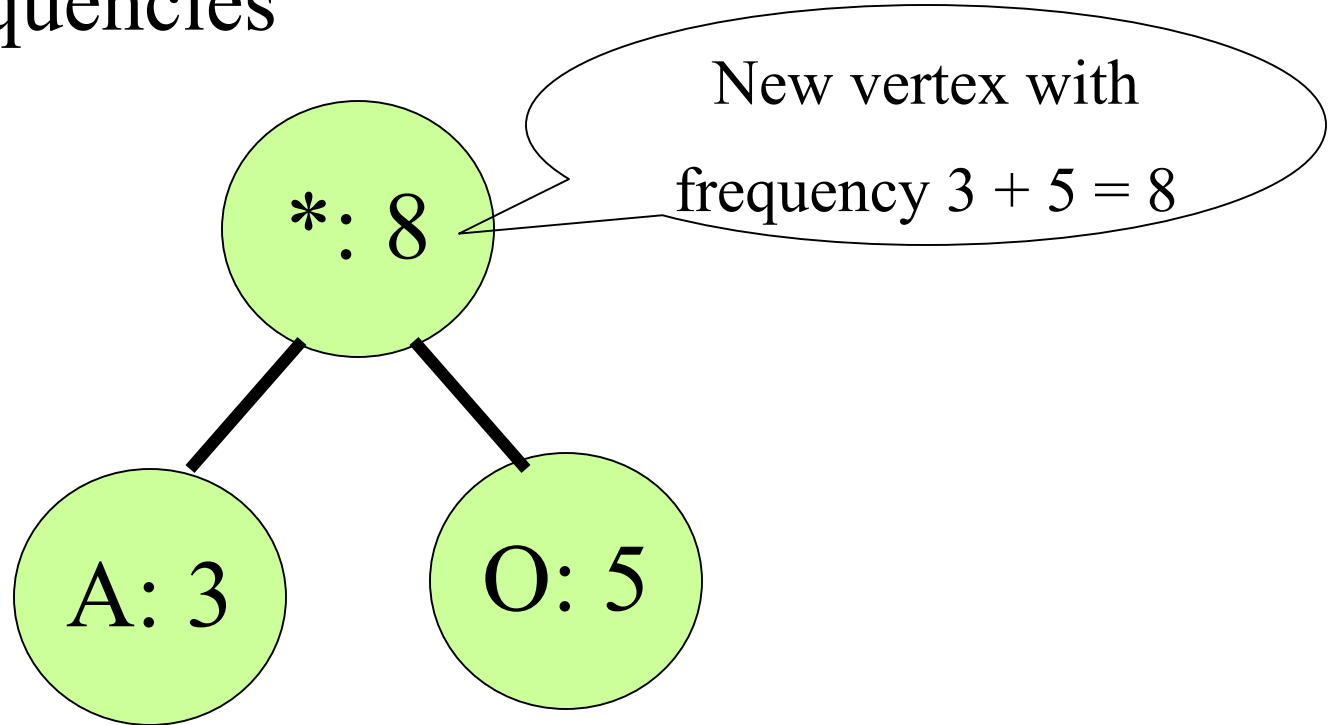
O: 5

T: 7

E: 10

# Huffman Codes

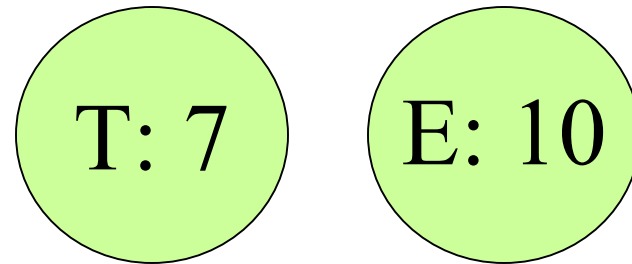
3. Take the first 2 vertices from the list and make them children of a vertex having the sum of their frequencies



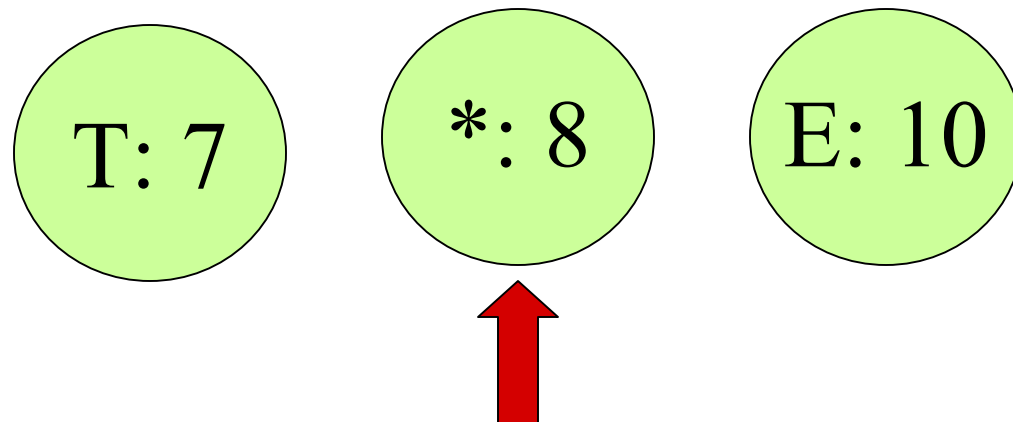
# Huffman Codes

4. Insert the new vertex into the sorted list of vertices waiting to be put into the tree

List of remaining vertices:

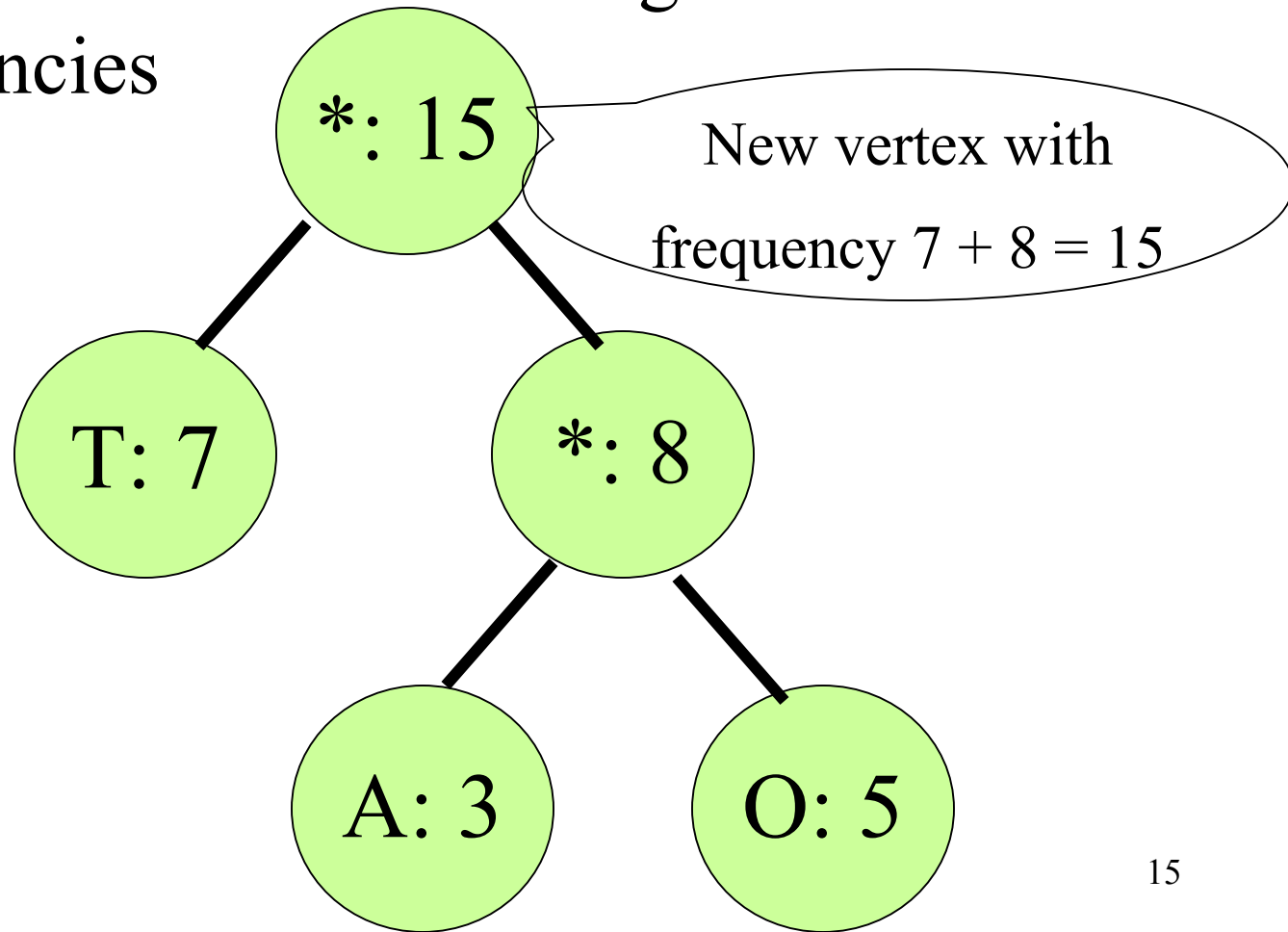


New list, with the new vertex inserted :



# Huffman Codes

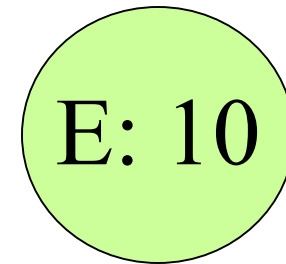
5. Take the first 2 vertices from the list and make them children of a vertex having the sum of their frequencies



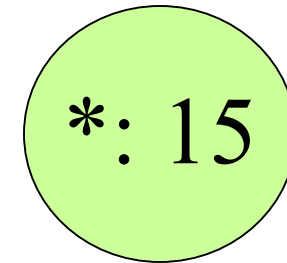
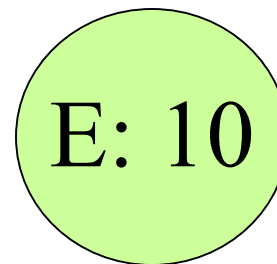
# Huffman Codes

6. Insert the new vertex into the sorted list of vertices waiting to be put into the tree

List of remaining  
vertices:

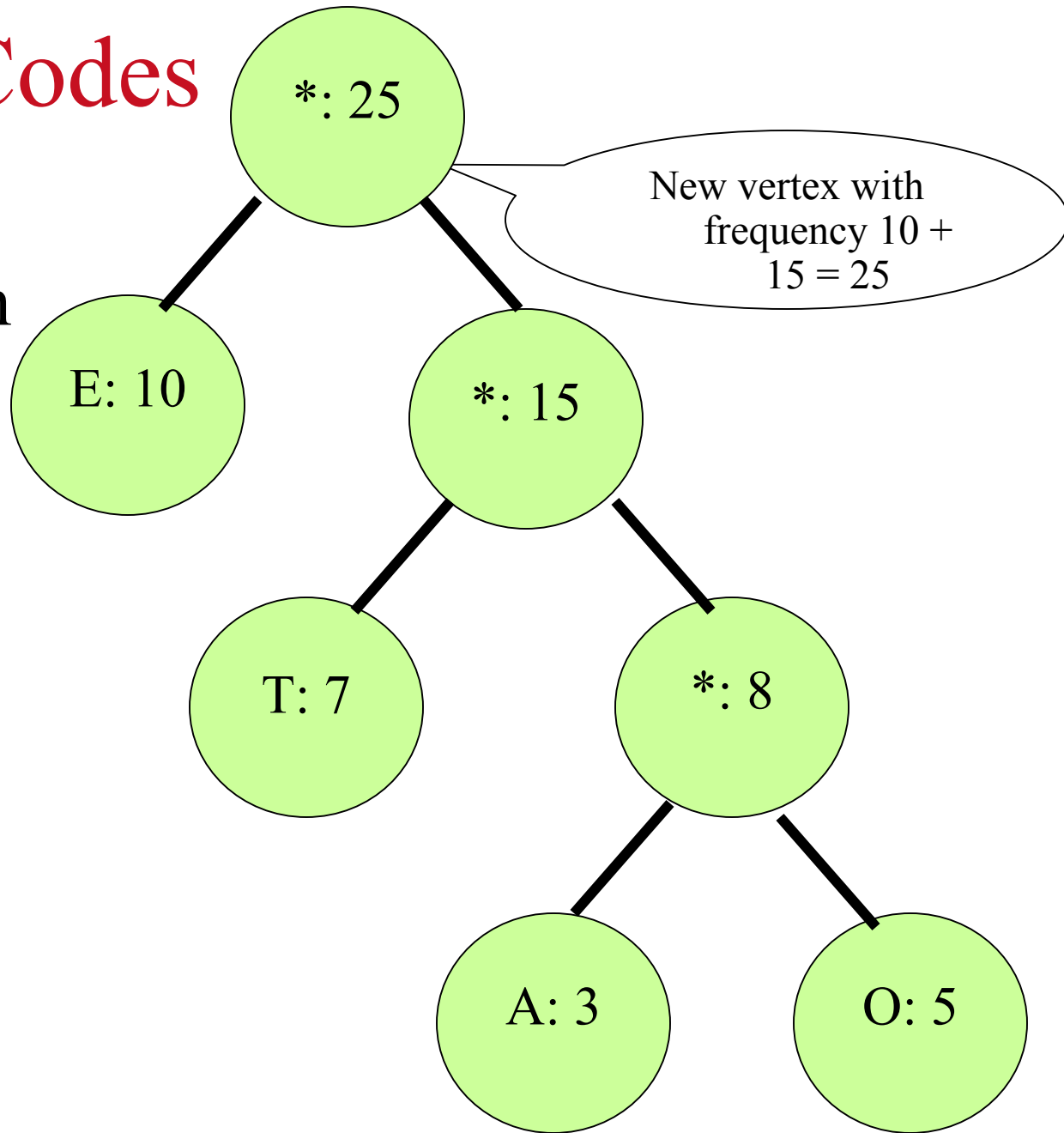


New list, with the  
new vertex  
inserted :



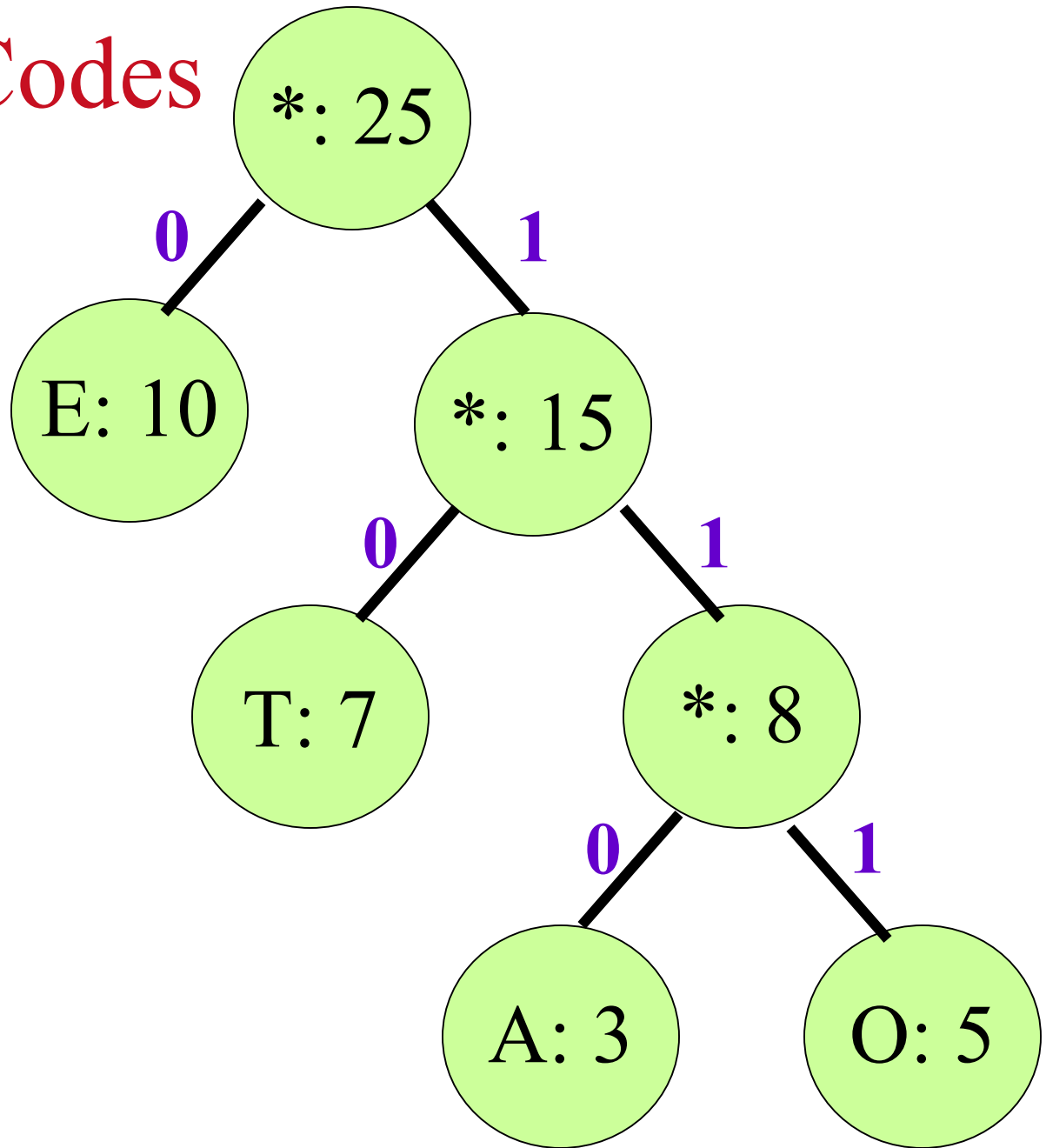
# Huffman Codes

7. Take the first 2 vertices from the list and make them children of a vertex having the sum of their frequencies



# Huffman Codes

- Left branch is 0
- Right branch is 1



Huffman code:

E: 0

T: 10

A: 110

O: 111

# Huffman Codes

1. Take the characters and their frequencies, and *sort* this list by **increasing** frequency
2. All the characters are vertices of the tree
3. Take the first 2 vertices from the list and make them children of a vertex having the sum of their frequencies
4. Insert the new vertex into the sorted list of vertices waiting to be put into the tree
5. If there are at least 2 vertices in the list, go to step 3.
6. Read the Huffman code from the tree

# Huffman Codes

- Reduce size of data by 20%-90% in general
- If no characters occur more frequently than others, then no advantage over ASCII
- Encoding:
  - Given the characters and their frequencies, perform the algorithm and generate a code. Write the characters using the code
- Decoding:
  - Given the Huffman tree, figure out what each character is (possible because of prefix property)

# Applications

- Both the .mp3 and .jpg file formats use Huffman coding at one stage of the compression
- Alternative method that achieves higher compression but is slower is patented by IBM, making Huffman Codes attractive

# References

Optional reading:

- [http://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman\\_tutorial.html](http://www.siggraph.org/education/materials/HyperGraph/video/mpeg/mpegfaq/huffman_tutorial.html)
- <http://www.huffmancoding.com/david/algorithm.html>