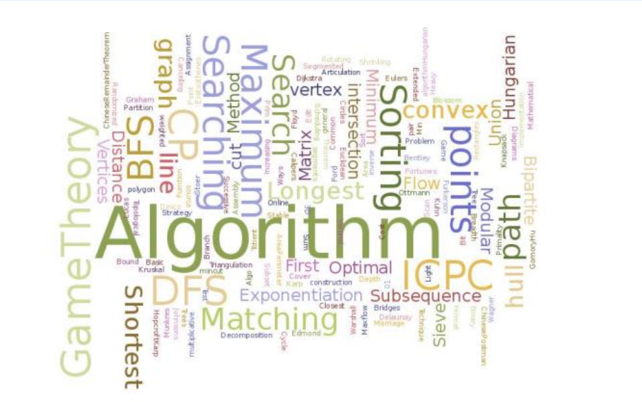
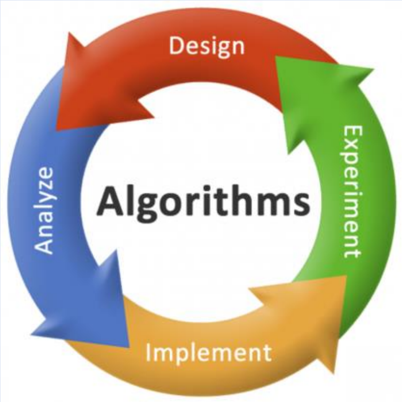
Greedy Algorithms

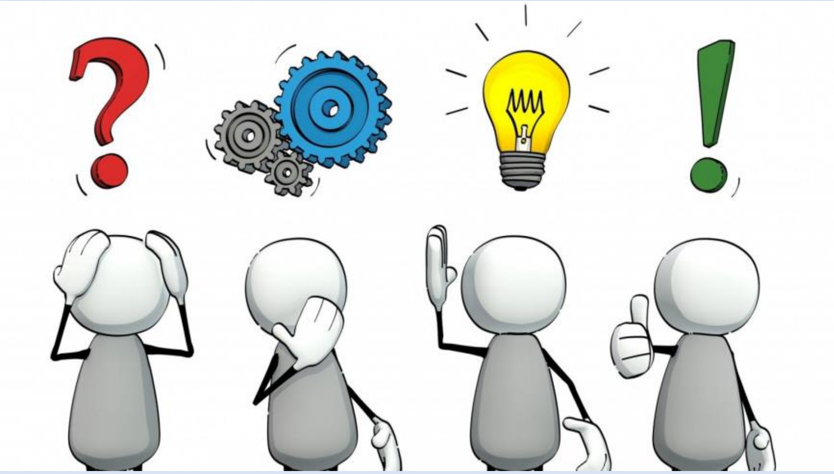
COP 3503
Fall 2021
Department of Computer Science
University of Central Florida
Dr. Steinberg

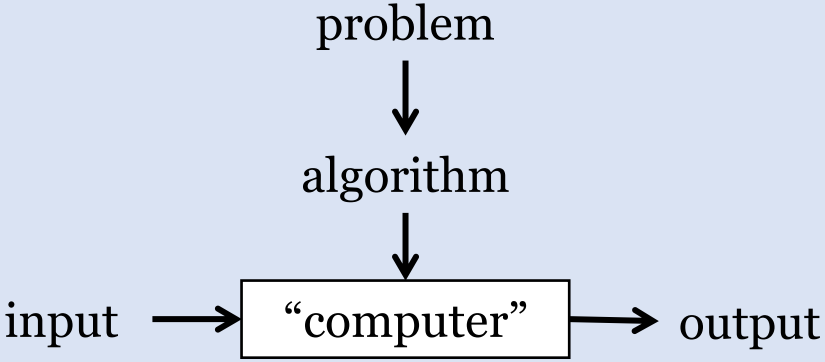




What is an Algorithm? (review)

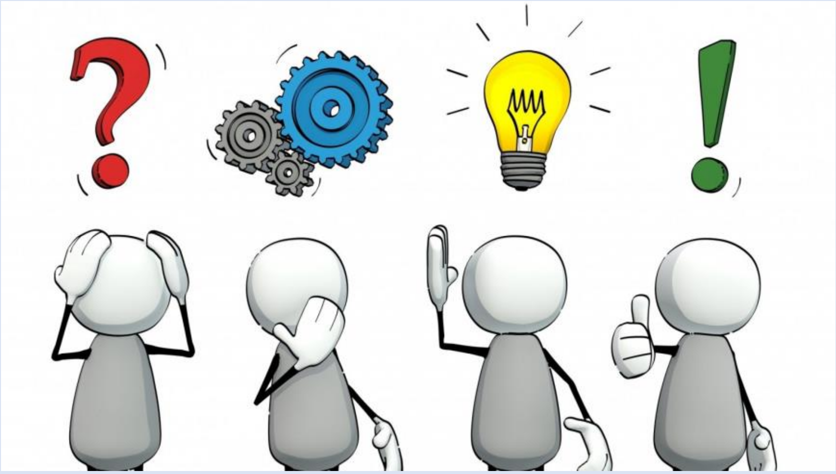
- A well-defined computational procedure which takes a value (or even set of values) as input and produces a value (or set of values) as output.
- An algorithm is said to be **correct** if, for every input instance, it halts with the correct output.





The Output Produced by our Algorithms

- Something to consider with our problems we are solving as programmers and computer scientists.
- Does there exist a group of solutions to a problem?



Greedy Algorithms

- Our objective is to produce the best output to a solution.
- Greedy algorithms incorporate the concept of making the best the decision at the current moment (without looking at the big picture overall).
- Greedy algorithms make a greedy choice
  - This results in looking at only one subproblem.
- Does a greedy algorithm produce the optimal solution always?

The Change Making Problem

- Problem Definition
  - We are provided a coinage system (such as pennies, nickels, dimes, and quarters). Each coin has an integer value (1, 5, 10, and 25). Given a value n, we want to know how many coins to give. Lets assume that we have unlimited coins to use.

The Greedy Solution

<u>MakeChangeGreedy(n)</u>

$$q = \left\lfloor \frac{n}{25} \right\rfloor$$

$$n_q = n \, mod \, 25$$

$$d = \left\lfloor \frac{n_q}{10} \right\rfloor$$

$$n_d = n_q mod \, 10$$

$$k = \left\lfloor \frac{n_d}{5} \right\rfloor$$

$$n_k = n_d \, mod \, 5$$

$$p = n_k$$

Greedy Solution to Change Making

- if n is 0, then the optimal solution is NO COINS.
- if n is positive, we start with the largest coin value c. Then we use the coin c and recursively solve for n – c cents until all coins are observed.

Huffman Code (Greedy Application)

- Huffman codes compress data effectively
- Data can be represented as a sequence of characters.
- The objective is designing a binary character code for each character. This allows for the creation of codewords in binary.
- Fixed-length code: max length of bits needed
- Variable-length code: vary length for each character

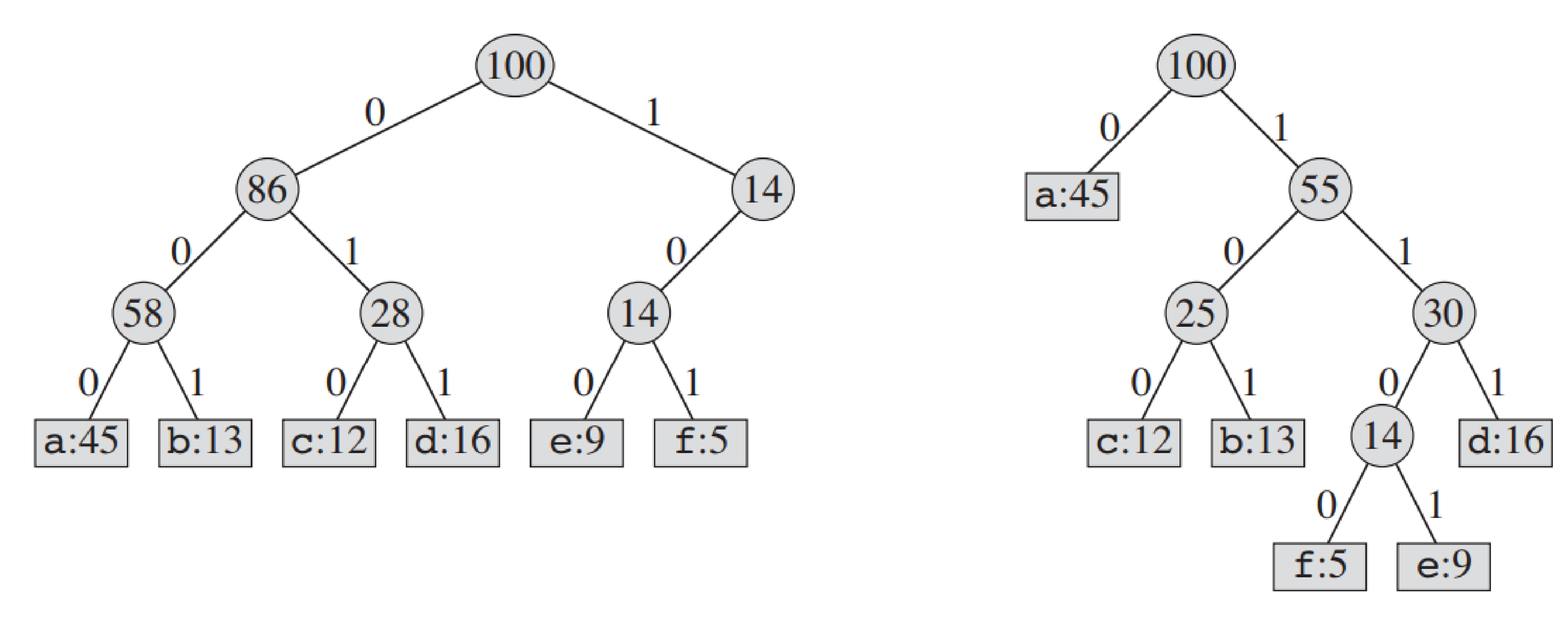
	a	b	c	d	e	f
Frequency (in thousands)	45	13	12	16	9	5
Fixed-length codeword	000	001	010	011	100	101
Variable-length codeword	0	101	100	111	1101	1100

$$file\ length = 3(45 + 13 + 12 + 16 + 9 + 5) = 3 * 100 = 300\ bits$$
$$variable\ length = 45 * 1 + 13 * 3 + 12 * 3 + 16 * 3 + 9 * 4 + 5 * 4 = 219\ bits$$

We can observe that variable length codewords will provide an optimal length. Meaning we can minimize the number of bits.

Binary Trees and Huffman Codes

- Binary code can be represented as a binary tree.



Some Terminology Regarding Huffman Codes

- Encoding – concatenating binary codewords
  - abba – 01011010
- Decoding – traversing the binary sequence from left to right until the first character is recognized
    - a = 0
    - b = 101
- Objective of Huffman Code: Find an optimal prefix code

Property of Optimal Prefix Code

- An optimal prefix code can always be represented by a full binary (each node has 0 or 2 children).
- Length of the file can be computed using a cost function B(T) associated to the binary tree T:
- $B(T) = \sum_{c \in C} c.freq * d_T(c)$
  - $d_T(c) \rightarrow$ depth of c′s leaf in the tree
  - $c.freq \rightarrow$ frequency of c in the

Huffman's Algorithm

- Huffman's Algorithm
  - Greedy Algorithm utilize in computing the optimal prefix code
  - Creates the binary tree representing the optimal prefix code

```
HUFFMAN(C)
n = |C|
Q = C
for i = 1 to n - 1
    allocate a new node z
    z.left = x = EXTRACT-MIN(Q)
    z.right = y = EXTRACT-MIN(Q)
    z.freq = x.freq + y.freq
    INSERT(Q, z)
return EXTRACT-MIN(Q)      // return the root of the tree
```

Huffman's Running Time Analysis

- The queue in the algorithm is a minimum priority queue using the minimum heap O(logn)
- Since the queue extract function is inside the loop, running time can be derived as O(nlogn)

The Correctness of Huffman's Algorithm

- Greedy Choice Property: Let C be represented as the alphabet and x, y represent characters with the lowest frequency. There exists an optimal prefix code where x and y are sibling nodes with highest depth.
- Optimal Substructure Property:
  - Given C as alphabet and x, y characters with lowest frequency
  - $C' = C - \{x, y\} \cup \{z\}$ where z.f = x.f + y.f
  - Given T' as the optimal prefix code for C'
  - T can be obtained from T' by replacing the leaf z with an internal node with 2 children (both of which are x and y)
  - Then T is an optimal prefix code for C.