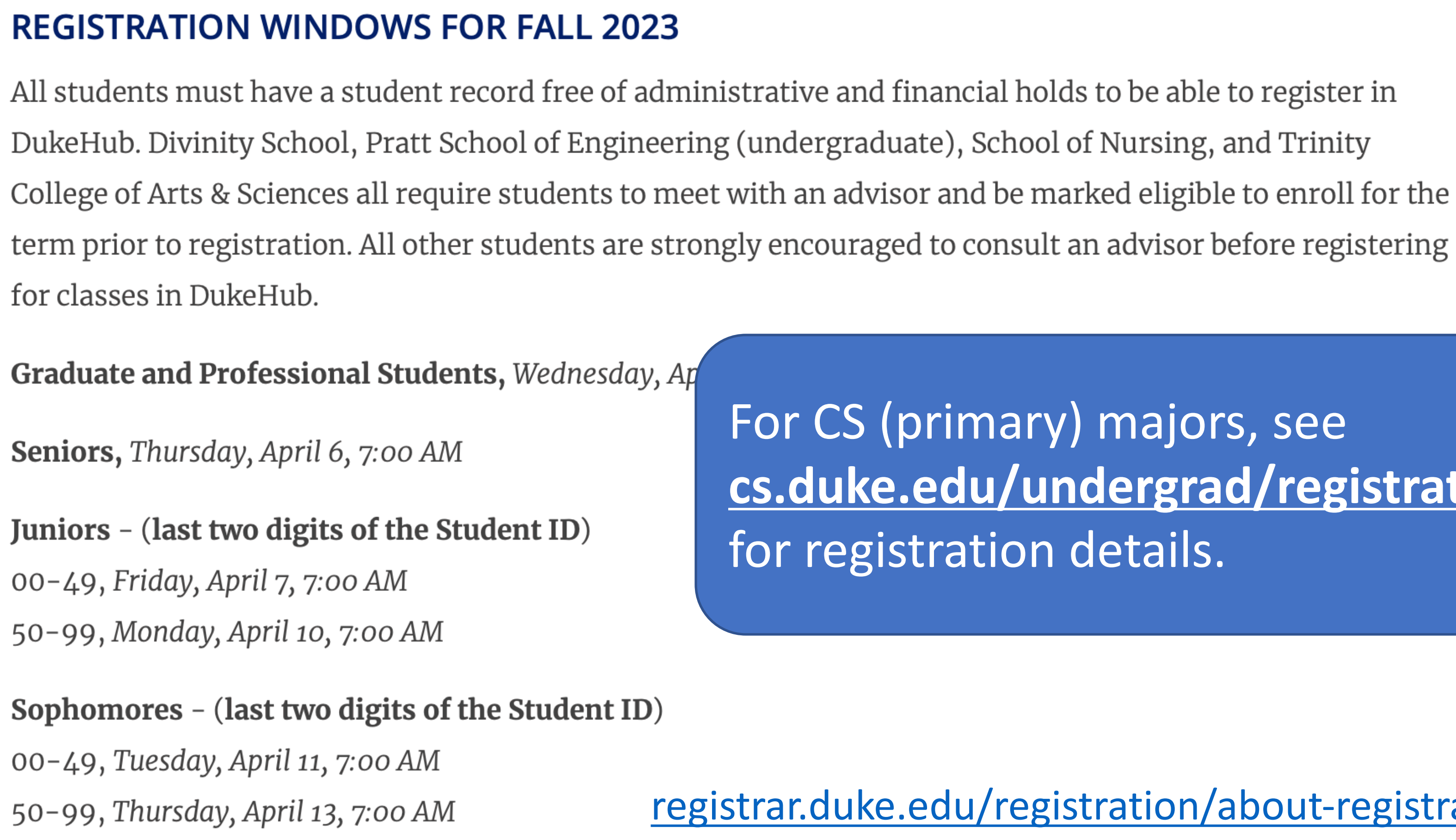
CompSci 201, L19: Greedy Algorithms, Huffman

Public Service Announcement: Fall 2023 Registration



REGISTRATION WINDOWS FOR FALL 2023

All students must have a student record free of administrative and financial holds to be able to register in DukeHub. Divinity School, Pratt School of Engineering (undergraduate), School of Nursing, and Trinity College of Arts & Sciences all require students to meet with an advisor and be marked eligible to enroll for the term prior to registration. All other students are strongly encouraged to consult an advisor before registering for classes in DukeHub.

Graduate and Professional Students, *Wednesday, Ap*

Seniors, *Thursday, April 6, 7:00 AM*

Juniors – (**last two digits of the Student ID**)
00–49, *Friday, April 7, 7:00 AM*
50–99, *Monday, April 10, 7:00 AM*

Sophomores – (**last two digits of the Student ID**)
00–49, *Tuesday, April 11, 7:00 AM*
50–99, *Thursday, April 13, 7:00 AM*

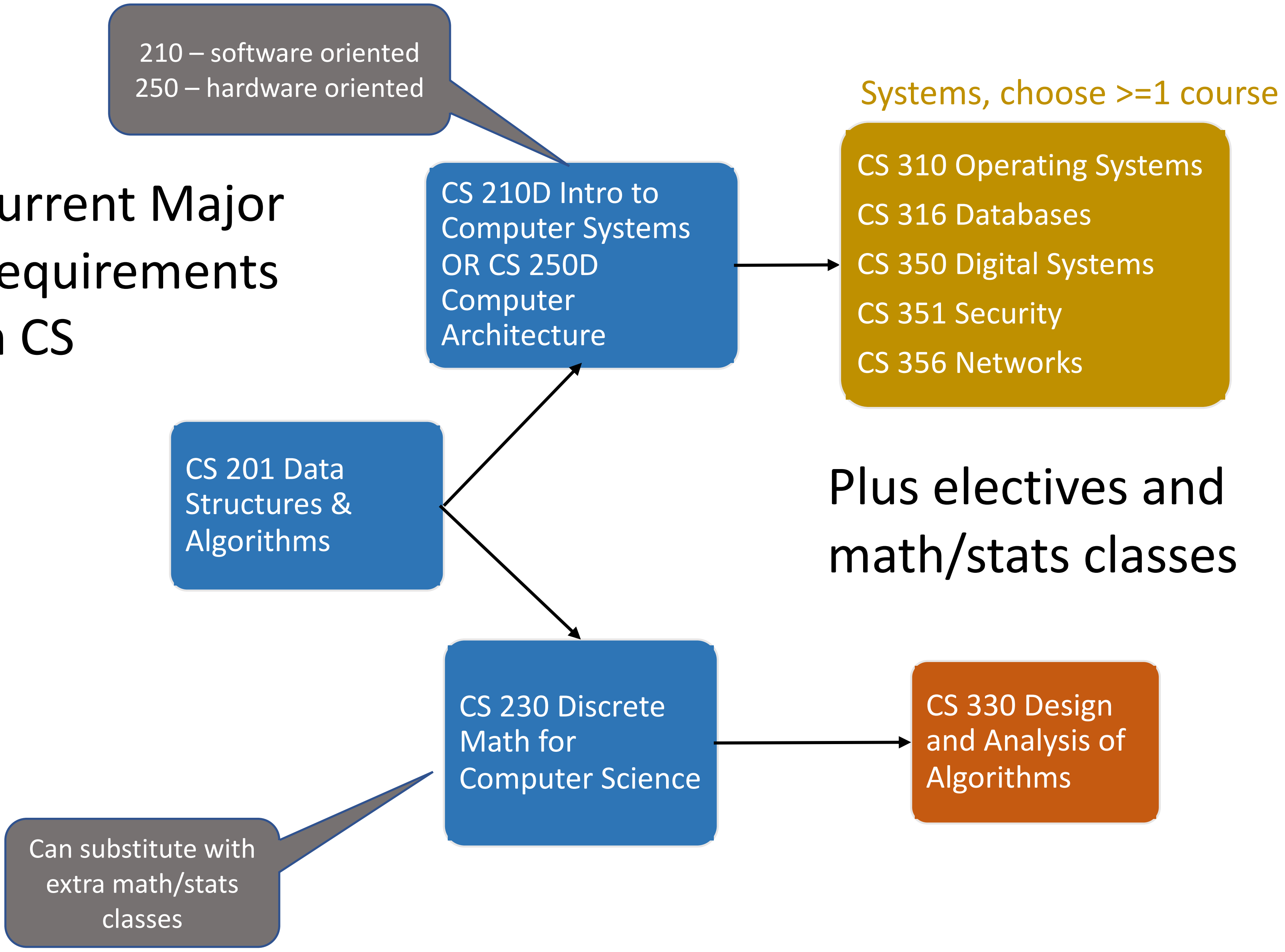
For CS (primary) majors, see **cs.duke.edu/undergrad/registration** for registration details.

registrar.duke.edu/registration/about-registration

CS Advising and Book bagging

- Considering a major/idm/minor/concentration? There are many pathways!
- Computer Science Majors
    - B.S. or B.A., same core CS requirements, different math and electives requirements.
- Interdepartmental Majors: 7 from CS, 7 from another
    - CS+Stats Data Science, CS+Math Data Science, CS+Linguistics, CS+VSM Computational Media.
- Computer Science Minor: 5 CS courses

Current Major Requirements in CS


210 – software oriented
250 – hardware oriented
Systems, choose >=1 course
CS 310 Operating Systems
CS 316 Databases
CS 350 Digital Systems
CS 351 Security
CS 356 Networks
CS 210D Intro to Computer Systems OR CS 250D Computer Architecture
CS 201 Data Structures & Algorithms
CS 230 Discrete Math for Computer Science
CS 330 Design and Analysis of Algorithms
Can substitute with extra math/stats classes

Plus electives and math/stats classes

Common post-201 CS Courses available in Fall 2022

Next required CS major courses:

- CS 210D Intro to Computer Systems OR CS 250D Computer Architecture.
- CS 230 Discrete Math for Computer Science.

Some electives in Fall 23 with no other prereqs (not exhaustive of the options!)

- CS 216 Everything Data
- CS 240 Race Gender Class & Computing

Logistics, Coming up

- Today: Monday 3/27
  - Project P4: Autocomplete due

- This Wednesday 3/29
  - APT 7 (tree recursion problems) due

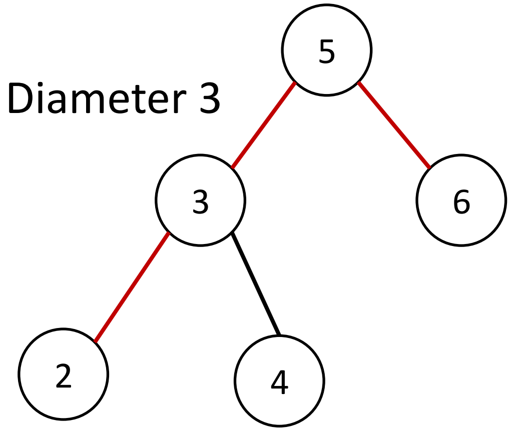
Today’s agenda

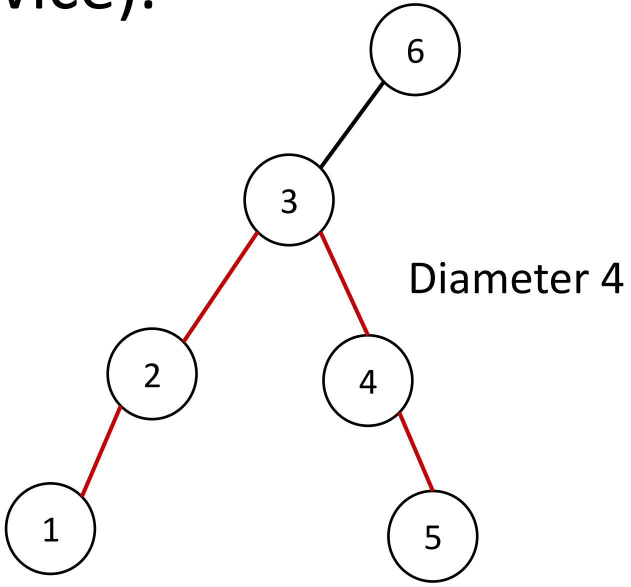
- Solve the tree diameter problem recursively
- Introduce Greedy Algorithms
- Huffman Coding (Project 5: Huffman)

Diameter Problem

leetcode.com/problems/diameter-of-binary-tree

Calculate the *diameter* of a binary tree, the length of the longest path (maybe through root, maybe not, can't visit any node twice).





Live Coding

Greedy Algorithms for Discrete Optimization

Optimization

- Find the solution that maximizes or minimizes some objective
- Example: Knapsack
  - Find the bundle of items with maximum value without exceeding a budget.
  - What should you buy if you have $10?

Items	Value	Cost
🍎	2	$1
🍌	1	$1
🍕	10	$10

Greedily Searching for Optima

- Start with a partial solution. In each iteration make a step toward a complete solution.
- Greedy principle: In each iteration, make the lowest cost or highest value step.
- Knapsack:
  - Partial solution is a set of items you can afford.
  - Greedy step: Add the next best value per cost item that you can afford.

Local Optima vs Global Optima?

Greedy algorithms do **not** always guarantee to find the best overall solution, called global optima.

Greedy picks:

1. The apple, best value/cost.
2. Then the banana, can't afford pizza.

Items	Value	Cost	Value/Cost
🍎	2	$1	2
🍌	1	$1	1
🍕	10	$10	1

Total value = 3.

But just buying the pizza give value 10.

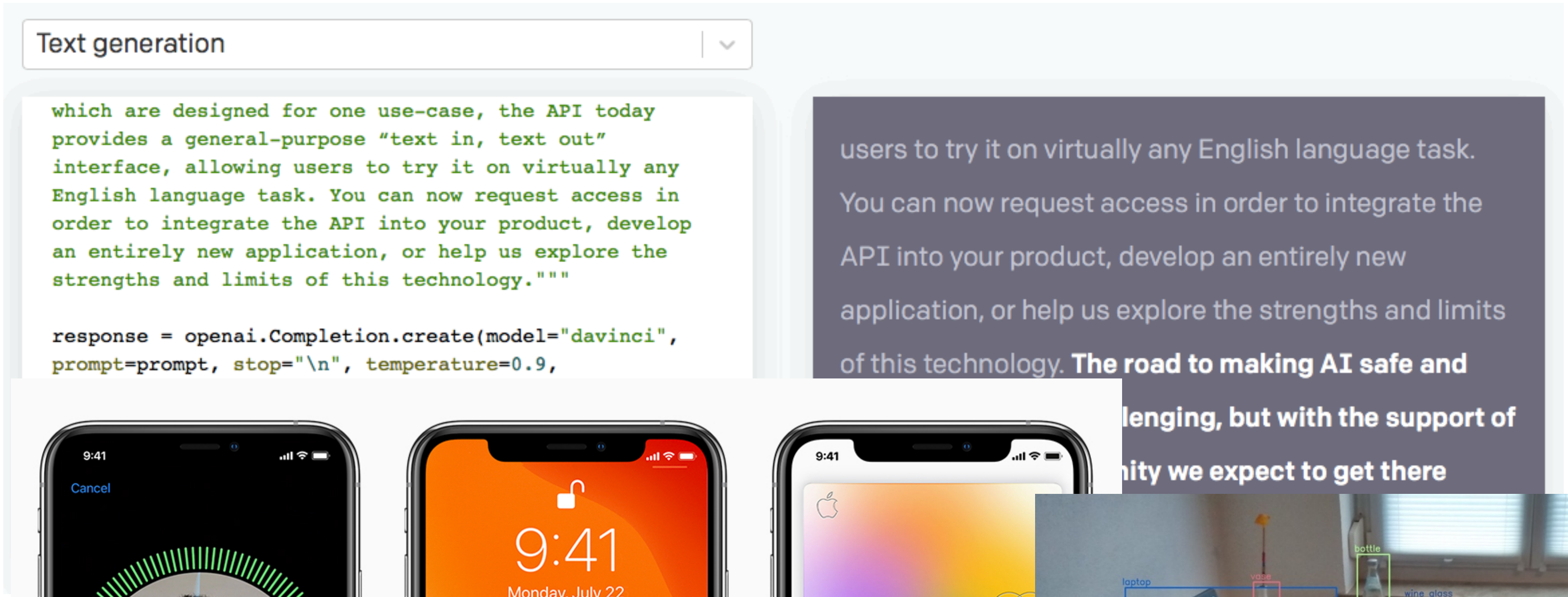
Why Learn Greedy Algorithms?

1. Sometimes a greedy algorithm is optimal. For example, we will study:
   - Huffman Compression (Today, Project 5)
   - Minimum Spanning Tree, in graphs

2. Sometimes the greedy algorithm isn't provably optimal but works well in practice.

3. A greedy algorithm is typically easy to start with for optimization problems.

Aside: What is Machine Learning?



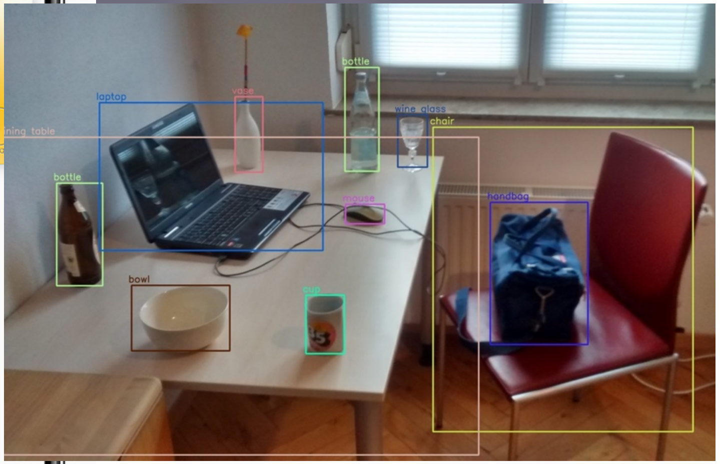
```
which are designed for one use-case, the API today
provides a general-purpose "text in, text out"
interface, allowing users to try it on virtually any
English language task. You can now request access in
order to integrate the API into your product, develop
an entirely new application, or help us explore the
strengths and limits of this technology."""

response = openai.Completion.create(model="davinci",
prompt=prompt, stop="\n", temperature=0.9,
```

users to try it on virtually any English language task. You can now request access in order to integrate the API into your product, develop an entirely new application, or help us explore the strengths and limits of this technology. **The road to making AI safe and**

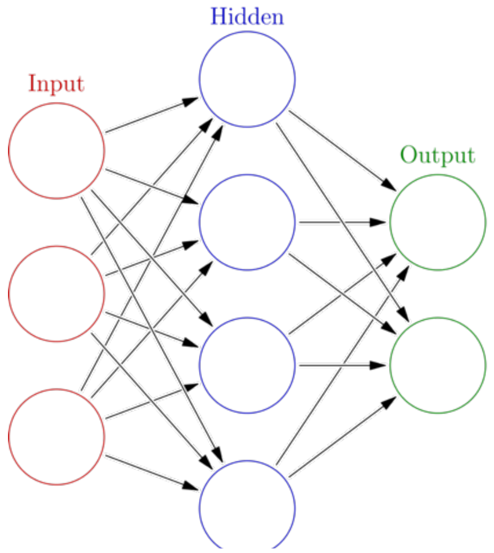
lenging, but with the support of

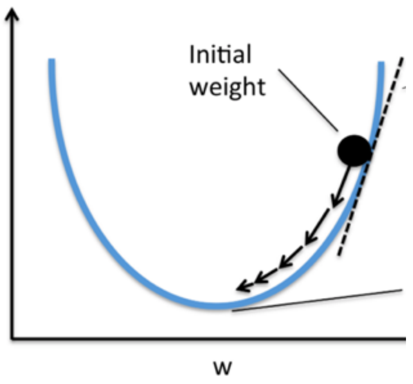
ity we expect to get there



Aside continued – How do you "learn a model" greedily?

- Often (in deep learning) represent a model with a **neural network**.
- Learn model = optimize parameters of network on data.
- How to optimize the parameters?
  - Greedy algorithm called gradient descent
  - At each step, make a small change that best improves model performance





Huffman Coding

Topic of Project 5: Huffman

Huffman Compression

Representing data with bits: Preferably fewer bits

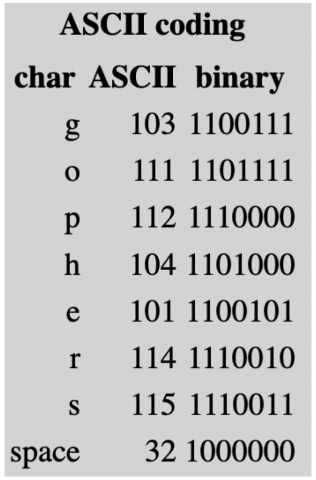
- Zip
- Unicode
- JPEG
- MP3

Huffman compression used in all of these and more!

Encoding

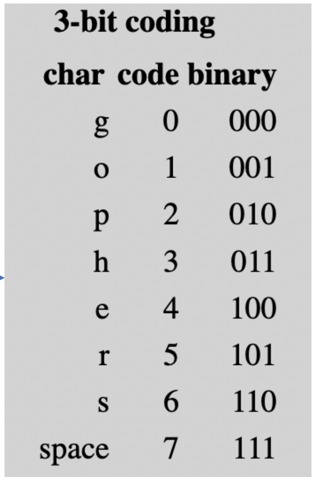
- Eventually, everything stored as bit sequence: 011001011…

- Fixed length encoding
  - Each value has a unique bit sequence of the same length stored in a table.
  - With N unique values to encode, need $\lceil \log_2(N) \rceil$ bits per value.
  - E.g., with 8 characters, need 3 bits per character.



ASCII coding

char	ASCII	binary
g	103	1100111
o	111	1101111
p	112	1110000
h	104	1101000
e	101	1100101
r	114	1110010
s	115	1110011
space	32	1000000



3-bit coding

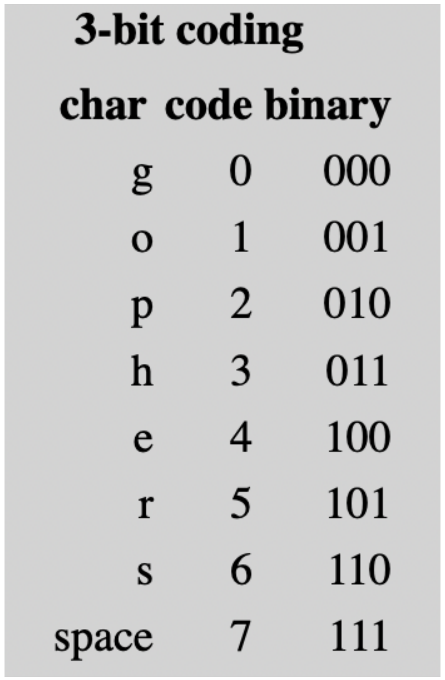
char	code	binary
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101
s	6	110
space	7	111

Optimizing Encoding?

- Suppose we have three characters {a, b, c}:
  - a appears 1,000,000 times
  - b and c appear 50,000 times each
- Fixed length encoding uses 2,200,000 bits.
  - $\lceil \log_2(3) \rceil = 2$
  - 2 times 1,100,000 values = 2,200,000 bits
- Variable length encoding: Use fewer bits to encode more common values, more bits to encode less common values.
  - What if we encode: a = 1, b = 10, c = 11?
  - Only uses 1,200,000 bits.

Decoding Fixed Length

- Fixed Length with length k
  - Every k bits, look up in table
  - $\underbrace{001}\ \underbrace{001}\ \underbrace{010}\ \underbrace{110}$
    - 001 -> o
    - 001 -> o
    - 010 -> p
    - 110 -> s



3-bit coding

char	code	binary
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101
s	6	110
space	7	111

Decoding Variable Length

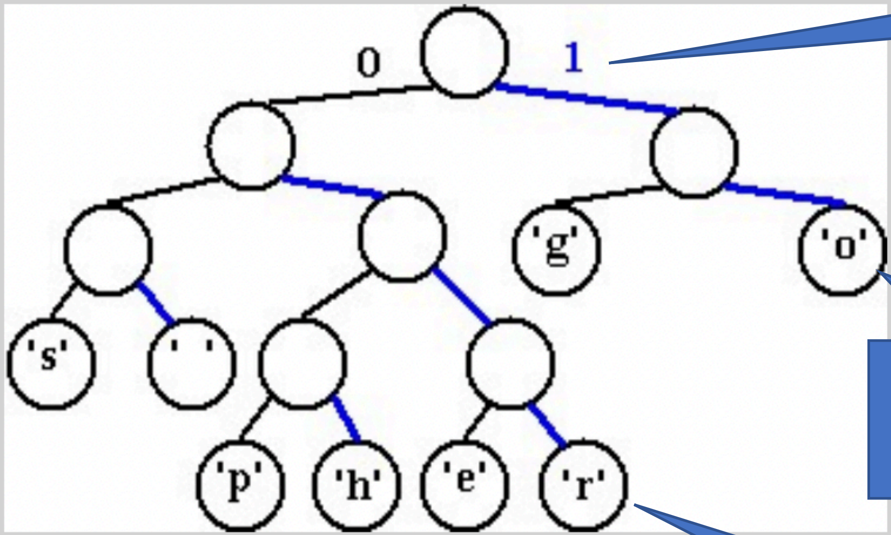
- What if we use
  - a = 1
  - b = 10
  - c = 11

- How would we decode 1011?
  - “baa” or “bc?”

- Problem: Encoding of a (1) is a *prefix* of the encoding for c (11).

Prefix property encoding as a tree

char	binary
'g'	10
'o'	11
'p'	0100
'h'	0101
'e'	0110
'r'	0111
's'	000
' '	001



Convention: 0 for left and 1 for right

Values you want to encode are leaves: Ensures prefix property.

Encoding is the sequence of 0's and 1's on root to leaf path

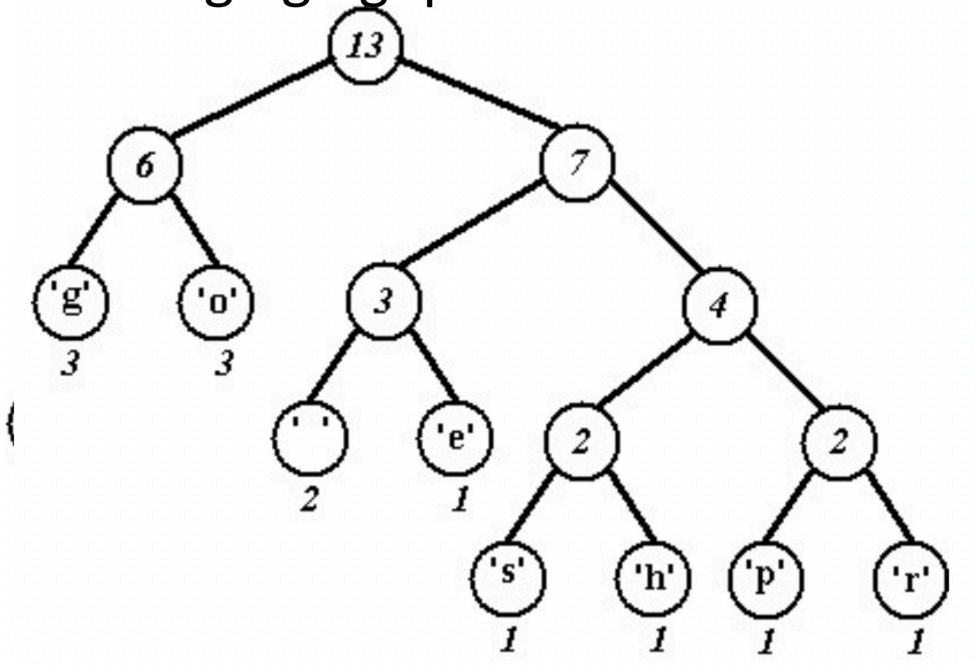
Values deeper in tree encoded with more bits than those earlier in the tree.

Huffman Coding

- Greedy algorithm for building an optimal variable length encoding tree.
- High level idea:
    - Start with the leaves/values you want to encode with weights = frequency.
    - Iteratively choose the ***lowest weight nodes*** to connect “up” to a new node with weight = sum of children.
- Implementation? Priority queue!

Visualizing the algorithm

Encoding the text “go go gophers”



char	binary
'g'	00
'o'	01
'p'	1110
'h'	1101
'e'	101
'r'	1111
's'	1100
' '	100

P5 Outline

1. Write Decompress first
   - Takes a compressed file (we give you some)
   - Reads Huffman tree from bits
   - Uses tree to decode bits to text

2. Write Compress second
   - Count frequencies of values/characters
   - Greedy algorithm to build Huffman tree
   - Save tree and file encoded as bits