CompSci 201, L22: Greedy Algorithms, Huffman

Logistics, Coming up

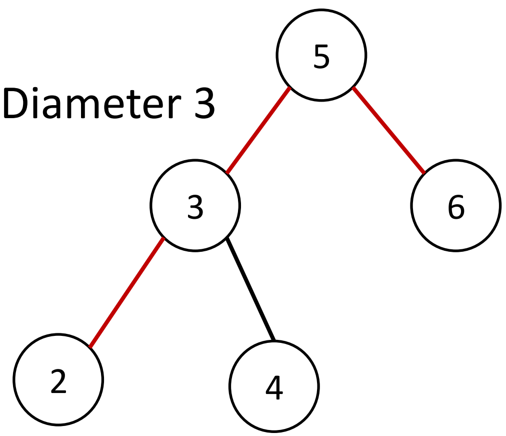
- APT8 (tree problems) due today
- No project due next Monday, start on P5 Huffman due the following Monday 11/14
- APT9 (more tree problems) due next Wednesday 11/9. Will see TreeTighten and LeafCollector problems in discussion this Friday.
- Wrapping up BSTs and binary heaps next week, then on to graphs!

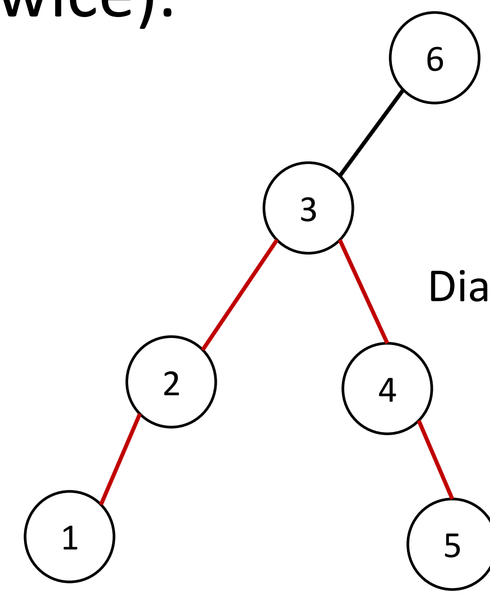
Diameter Problem

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).

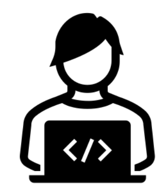
Diameter 3





Diameter 4

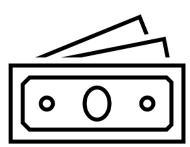
Live Coding

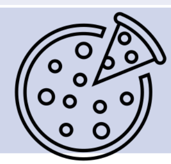


Greedy Algorithms for Discrete Optimization

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.

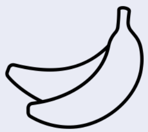
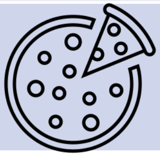
- Knapsack:
  - Partial solution is a set of items you can afford.
  - Greedy step: Add the next best value per cost item that you can afford.

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 cherries give value 10.

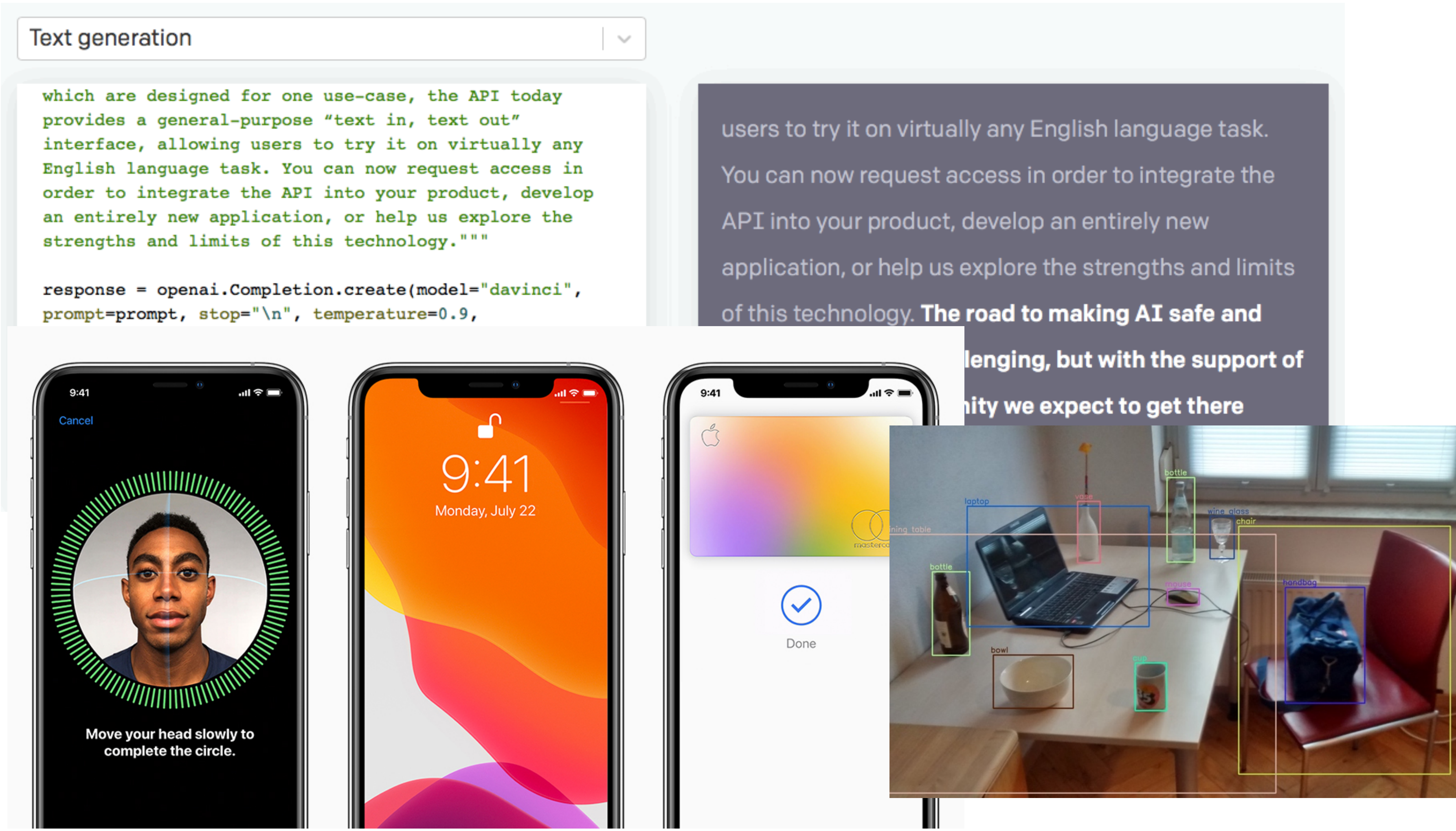
Why Learn Greedy Algorithms?

1. Sometimes a greedy algorithm is optimal. Many of the algorithms we study in the rest of this course are greedy!
    - Huffman Compression (Today, Project 5)
    - Breadth-first search, Dijkstra’s algorithm, in graphs
    - Minimum Spanning Tree, in graphs

1. Sometimes the greedy algorithm isn’t provably optimal but works well in practice.

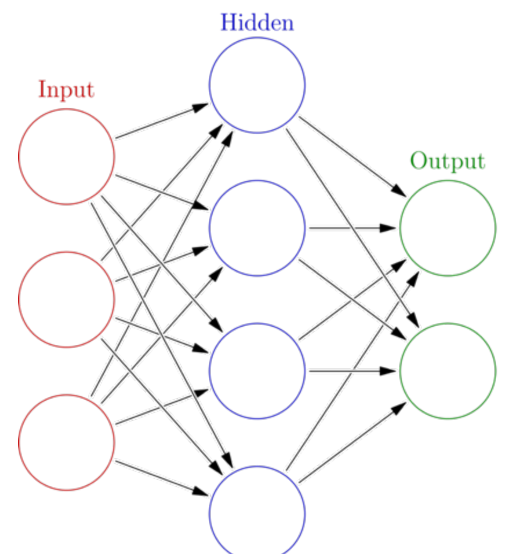
2. A greedy algorithm is typically easy to start with for optimization problems.

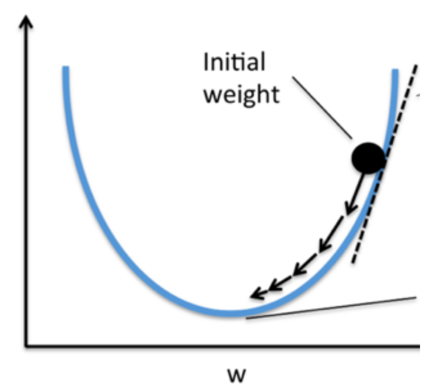
Aside: What is Machine Learning?



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





VoteRigging APT

- https://www2.cs.duke.edu/csed/newapt/voterigging.html
- Given votes for candidates, **`{5,10,7,3,8}`**
  - You are candidate with index 0
  - Minimal votes to buy to win election?
- Buy from index/1: {6,9,3,8}
- Buy from index/1: {7,8,3,8}
- Buy from index/1: {8,7,3,8}
- Buy from index/3: {9,7,3,7} Winner! 4 votes

Greedy algorithm for buying votes

- Buy a vote from opponent with the most
  - You must beat them, you must buy a vote

- In this case, the greedy decision “buy from leader” leads to ***globally optimal*** solution that buys the fewest votes overall.

- How to realize algorithm in code?

The big picture

```
public int minimumVotes(int[] votes) {
    int req = 0;
    int winner = getMax(votes);
    while (winner != 0) {...}
    return req;
}
```

- Need a while loop – don't know how many votes we need to buy!
- Helper method to get who is currently winning.

A greedy step toward a solution

- Inside loop, make a greedy step toward a solution:

- Buy a vote from the current winning candidate

- Update to get new current winner

```
public int minimumVotes(int[] votes) {
    int req = 0;
    int winner = getMax(votes);
    while (winner != 0) {
        votes[winner]--;
        votes[0]++;
        req++;
        winner = getMax(votes);
    }
    return req;
}
```

Efficiency?

- R iterations through loop, where R is the minimum number of votes to buy.
- getMax(votes) loops over votes, so O(N) with N candidates.
- Overall: **O(R*N)**

```
public int minimumVotes(int[] votes) {
    int req = 0;
    int winner = getMax(votes);
    while (winner != 0) {
        votes[winner]--;
        votes[0]++;
        req++;
        winner = getMax(votes);
    }
    return req;
}
```

Priority Queue Implementation

```
public int minimumVotes(int[] votes) {
    if (votes.length == 1) { return 0; }
    int req = 0;
    int ourCount = votes[0];

    PriorityQueue<Integer> pq;
    pq = new PriorityQueue<Integer>(Collections.reverseOrder());
    for (int i = 1; i < votes.length; i++) {
        pq.add(votes[i]);
    }

    int winCount = pq.remove();
    while (ourCount <= winCount) {...}
    return req;
}
```

Nothing to do if only one candidate

Keep track of votes for candidate 0

Max-heap priority queue

Same while loop written differently: While our candidate isn't winning…

Zooming in on the loop

- Same algorithm but store vote counts in pq.
- Add/remove for pq (implemented as a heap) are **O(log(N))**.
- Overall: **O(R*log(N))** [plus O(Nlog(N)) pre-processing]

```
while (ourCount <= winCount) {
    ourCount++;
    pq.add(winCount - 1);
    req++;
    winCount = pq.remove();
}
```

WOTO

Go to duke.is/6pczd

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

Project 5 Huffman

Huffman Compression

Representing data with bits: Preferably fewer bits

- Zip

- Unicode

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

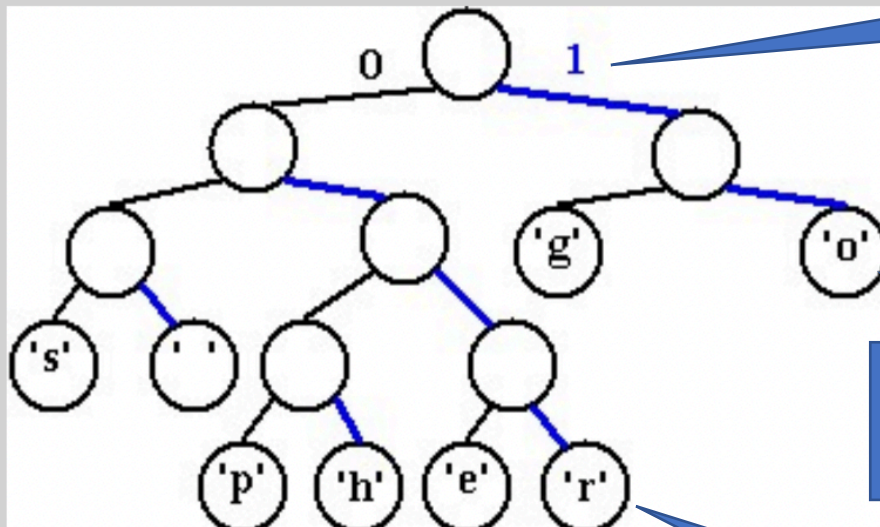
char	code	binary
g	0	000
o	1	001
p	2	010
h	3	011
e	4	100
r	5	101
s	6	110
space	7	111

Decoding Variable Length

- What if we use
  - a = 1
  - b = 10
  - c = 11
- How would we decode 1011?
- Is it “baa” or “bc?”
- Problem: Encoding of a (1) is a *prefix* of the encoding for c (11).
- Solution: Don’t allow these encodings!

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.

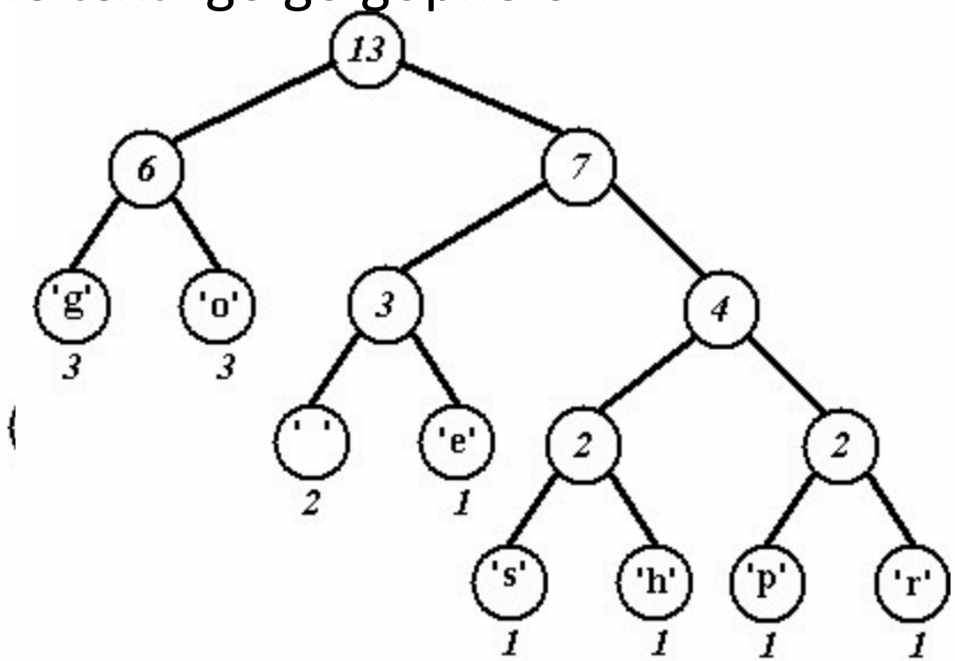
Huffman Coding

- 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.

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

WOTO

Go to duke.is/pge7f

Not graded for correctness, just participation.

Try to answer *without* looking back at slides and notes.

But do talk to your neighbors!

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

People in CS: Clarence “Skip” Ellis

- Born 1943 in Chicago. PhD in CS from U. Illinois UC in 1969
  - First African American anywhere in US to complete a PhD in CS
- Founding member of the CS department at U. Colorado, also worked in industry.
  - Developing original graphical user interfaces, object-oriented programming, collaboration tools.

“People put together an image of what I was supposed to be,” he recalled. “So I always tell my students to push.”

[Read more here]