Agenda

1. Pleasantrees
2. Mini Assignment
3. Functional Programming
4. Problem Complexity
What’s your favorite data structure and why?
Can you believe the semester is almost ending? :( We will miss you!
Mini Assignment
Problem 1a Review

How did you approach Problem 1?

Part 1: Length of each string in list

Write code for the `string_length` function, which takes in a list of strings and returns a list of their lengths.

Example: `string_length(['cat', 'a', 'square']) → [3,1,6]`
def strings_length(strs):
    return map(len, strs)
Problem 1b Review

How did you approach Problem 2?

Part 2: Max string length in list

Write code for the `max_string_length` function, which takes in a list of strings and returns the length of the longest string in the list.

Example: `max_string_length(["cat", "a", "orange","square"]) → 6`
Problem 1b Answer

def max_strings_length(strs):
    return reduce(max, map(len, strs), 0)
1. In comparing an online algorithm to its offline counterpart, we see that the biggest difference in their costs occurs when the former has a cost of 300 units and the latter has a cost of 100 units. What is the competitive ratio in this case?

2. What are the definitions of tractable and intractable problems?
1. The competitive ratio is 3. This means that the online solution will perform at worst 3 times worse than the offline solution.

2. Tractable problems have polynomial runtimes, intractable problems have super-polynomial runtimes (exponential). We generally think of the latter as 'hard' problems.
Functional Programming
Functional Programming

- A style of building the structure and elements of computer programs that treats computation as the evaluation of mathematical functions.
Conceptual Review: Map

- **Map** is a higher order function with the following specifications:

  - **Inputs**
    - func - a function that takes in an element
    - list - a list of elements

  - **Output**
    - A new list of elements, with func applied to each of the elements of list

```python
def map(func, list):
    for element in list:
        element = func(element)
    return list
```
Conceptual Review: Map

\[ \text{map}(\lambda x: x-2, [11, 9, 24, -5, 34, 4]) \]

<table>
<thead>
<tr>
<th>11</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>22</td>
</tr>
<tr>
<td>-5</td>
<td>-7</td>
</tr>
<tr>
<td>34</td>
<td>32</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>
Conceptual Review: Reduce

- It reduces a list of elements to one element using a binary function to successively combine the elements.

- Inputs
  - binary_func - a binary function
  - list - list of elements
  - acc - accumulator, the parameter that collects the return value

- Output: The value of binary_func sequentially applied and tracked in acc

```python
def reduce(binary_func, list, acc):
    for element in list:
        acc = binary_func(acc, element)
    return acc
```
Conceptual Review: Reduce

```python
# binary function 'add'
add = lambda x, y: x + y

# use 'reduce' to sum a list of numbers
>>> print reduce(add, [1,2,3], 0)
6
```
Conceptual Review: Reduce

Math

- \(((0 + 1) + 2) + 3\) = ?
- \((1 + 2) + 3\) = ?
- \((3 + 3)\) = ?

6

Python

- `reduce(add, [1,2,3], 0)` = ?
- `reduce(add, [2,3], 1)` = ?
- `reduce(add, [3], 3)` = ?

6

final accumulator/return value
Functional Programming Problems

Take out a piece of paper and with a partner, write the way in which you would go about solving the following problems

1. Write an anonymous function that doubles a single argument ‘n’
2. Write an anonymous function that doubles all elements in a list.
3. Create an anonymous function that raises an argument x to nth power
1. $\lambda x: 2^x$
2. $\text{map}(\lambda x: 2^x, \text{list})$
3. $\lambda x, n: x^{**n}$
4. Write a function to eliminate consecutive duplicates in a list using only reduce
5. Implement map using reduce
Eliminate Duplicates using Reduce Only

4.
def compress(my_list):
    return reduce(lambda x, y: x + [y] if x[-1] is not y else x, my_list, my_list[:1])
Implement Map using Reduce

5.

def my_map(function, list):
    return reduce(lambda x, y: x +
                  [function(y)], list, [ ])

Problem Complexity
Complexity

- NP
  - NP-Complete
  - NP-Hard
- P
  - P ≠ NP

- P = NP
  - P = NP = NP-Complete
  - NP-Hard
**P**: We have a polynomial-time solution, and can check if the answer is correct in polynomial time

- Example: raising all elements in a list to the nth power
**Complexity**

**NP**: We may or may not have a solution (ability to solve), but we can check if the answer is correct in polynomial time

- Example: determining if two graphs are isomorphic: Two graphs are isomorphic if they contain the same number of vertices and edges and are connected in the same way (they just may not visually look the same)
Complexity

**NP-Complete**: A problem $x$ that is in NP is also in NP-Complete if and only if every other problem in NP can be quickly (i.e. in polynomial time) transformed into $x$.

- Example: Traveling Salesman - Given a list of cities and the distances between each pair of cities, what is the shortest possible route that visits each city exactly once and returns to the origin city? TLDR: Connect all the cities as cheaply as possible.
- Getting the absolute best solution is impossible. We can use MST to get close to the answer in reasonable time (a simpler version of the problem).
**NP-Hard**: A problem is NP-hard, when an NP-Complete problem can be reduced to it. All NP-Complete problems are NP-Hard, but not all NP-Hard problems are NP-Complete.

- Example: Halting problem - problem of determining, from a description of an arbitrary computer program and an input, whether the program will finish running or continue to run forever.
- The halting problem can’t be solved even in non-polynomial time.
Online Algorithms
Online Algos

- What is the difference between an online and offline algorithm?
  - An online algorithm does not have access to all of the data at the start
    - Offline algorithms have access to all of the data at the start
  - Data is received serially, with no knowledge of what comes next
Online Algos

Experts Problem

- You know nothing so you should ask for help
- You know experts who can give you advice before you make each decision but you don’t if they know the correct answer
- If you make the right decision, you gain nothing
- If you make the wrong decision, you get 1 unit of embarrassment
- Total embarrassment = number of mistakes
- Goal: Minimize total embarrassment (relative to what the best expert would’ve gotten)
Online Algos - Example

- Come up with a question to answer (ex: Chipotle vs Bajas)
  - TAs choose an answer to be “correct”
- Assign every expert a weight of 1, for total weight of $W = n$ across all experts
- Repeat for every decision:
  - Ask every expert for their advice
  - Weigh their advice and decide by majority vote
  - After the outcome is known, take every expert who gave bad advice and cut their weight in half, regardless of whether your bet was good or bad
Online Algos

**Multiplicative Weights Algorithm - Analysis**

- To analyze how good this is, we need to relate the number of mistakes we make \((m)\) to the number of mistakes the best expert makes \((b)\).
- How can we do that? Use the weights!
- Let \(W\) represent the sum of the weights across the \(n\) experts at an arbitrary point in the algorithm.

**Experts Algorithm - Analysis**

- Look at the total weight assigned to the experts.
- When the best expert makes the wrong decision...
  - We cut their weight in half
  - They started out with a weight of 1

\[
\left(\frac{1}{2}\right)^b \leq W
\]
Online Algos

Experts Algorithm - Analysis

- Look at the total weight assigned to the experts
  - When we made the wrong decision...
    - At least ½ weight was placed on the wrong decision
  - We will cut at least ¼ of $W$, so we will reduce the total weight to at most ¾ of $W$
  - Since we gave the experts $n$ total weight at the start:

$$W \leq n \left(\frac{3}{4}\right)^m$$

Experts Algorithm - Analysis

$$\left(\frac{1}{2}\right)^b \leq W \leq n \left(\frac{3}{4}\right)^m$$

$$\left(\frac{1}{2^b}\right) \leq W \leq n \left(\frac{3}{4}\right)^m$$

$$-b \leq \log_2 n + m \log_2 \left(\frac{3}{4}\right)$$

We know that $b + \log_2 n \geq m \log_2 \left(\frac{4}{3}\right)$

So the number of mistakes we make, $m$, is at most 2.41 times the number of mistakes the best expert makes, $b$, plus some change.

$$\frac{b + \log_2 n}{\log_2 \left(\frac{4}{3}\right)} \geq m$$
Topics to review?