Section 10 Overview

Agenda

- Mini-assignment review

  Problem 1.

  Part 1:
  ```
  def strings_length(strs):
          return map(len,strs)
  ```
  Part 2:
  ```
  def max_strings_length(strs):
          return reduce(max,map(len,strs), strs[0])
  ```
  Problem 2
  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
  - Multiply all elements in a list by 2
    ```
    map(lambda x: 2*x, list)
    ```
  - Eliminate consecutive duplicates in a list using only `reduce`
    ```
    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`
    ```
    def my_map(function, list):
            return reduce(lambda x, y: x + [function(y)],
    list, [ ])
    ```

- Problem complexity
  - **P:** We have a polynomial-time solution, and can check if an answer is correct in polynomial time
    - Example: raising all elements in a list to the nth power
  - **NP:** We may or may not have a solution, but we can check if an 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)
  - **NP-Complete:** We can reduce (or 'rephrase') any NP-Complete problem to any other NP-Complete problem in polynomial time.
    - 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)
            - Other examples: knapsack problem, boolean satisfiability, vertex cover, independent set, graph coloring
    - **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.
        - Halting Problem
            - 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
    - Review of experts algorithm -- see the lecture slides on online algorithms!