# Victory Lap

CSE 421 Fall 2022
Lecture 30

# Announcements

Please fill out course evals!

https://uw.iasystem.org/survey/263630

First time this course has been 4 credit hours and sections

First time I've taught this course.

We want to hear what to change

and we're teaching it again next quarter.

# What have we seen this quarter?

- Stable Matchings
- Graph Search
  - BFS/DFS
  - Graph modeling
- Greedy Algorithms
- Divide and Conquer
- Dynamic Programming
  - DP on arrays
  - Adding parameters
  - DP on trees
- Network Flow
  - Using Max-Flow
  - Using Min Cuts
  - Assignment problems
- NP-completeness
  - Reductions
  - Showing a problem is NP-hard
  - P vs. NP
- Approximation Algorithms
- A Randomized Algorithm

## Stable Matchings

Modeling matters!

- It’s better to be a proposer than a chooser!

Algorithms can be used to prove ‘non-computational’ facts

- Stable Matchings always exist is easiest to prove by saying “here’s how to find one.”

Reductions

- Sometimes there’s a clever way to use an existing library (we’ll need these a lot later in the quarter).

## Graph Search

BFS and DFS search through a graph differently

- So you can adapt them to solve different problems!

Use libraries

Finding SCCs and Topological sorting are “almost free” preprocessing

2-Coloring can be performed in linear time.

# Greedy

Code is easy; proofs are hard.

Generating examples is **extremely** important.

To frame your thinking for proofs

- Greedy stays ahead
- Exchange argument
- Structural result

## Divide and Conquer

Trust the recursion.

Don’t be afraid to change what the recursive call gives you!

Add extra parameters!

State in English what the recursive call gives you.

In your “combine” step, make sure you’re beating baseline!

# Dynamic Programming

Focus on solving the problem recursively; everything else is (mostly) formulaic once you've done that.

Write exactly the problem you're solving in English.

It's better to get down a "guess" at the problem and then see where you get stuck.

- Don't be afraid to add a second recurrence or extra parameters.

Don't try to cleverly figure out which option is best. Try them all.

- The magic of recursion tells you which is best for a particular situation.

DP is very useful on trees!

## (one) Randomized Algorithm

You saw Karger’s Algorithm

Gives you a min-cut (in an undirected, unweighted graph) with high probability (at least $1 - 1/n$)

In general, can be a way to solve problems more simply or quickly.

Or avoid “worst-case” behavior

## Network Flow

The value of the maximum flow is always equal to the capacity of the minimum cut.

- Ford-Fulkerson finds you both.

Useful for modeling ("assigning" things)

- Can the Mariners make the playoffs? (Assign who wins games)
- Who does which chores?

And solving other problems

- Especially on bipartite graphs, like vertex cover and maximum matching.

## NP-completeness

$A \leq_P B$ means that you can use a library that solves problem $B$ as a way to solve problem $A$

(with only polynomial time and polynomial calls to the library).

3-SAT (and other problems!) are NP-hard, which means for every problem $A \in NP$, $A \leq_P$3-SAT.

Showing a reduction from an already known $\mathbf{NP}$-hard problem to a new problem shows the new problem is also $\mathbf{NP}$-hard.

The order of the reduction is VERY important.

## Approximation Algorithms

A way to cope with NP-hardness.

Approximation ratio:

The number (or function) $> 1$ that represents the ratio between alg and opt.

Minimization problem: for all instances $I$: $\alpha \cdot OPT(I) \geq ALG(I)$

Maximization problem: for all instances $I$: $OPT(I) \leq \alpha \cdot ALG(I)$

Rounding LPs is a common strategy

All our tools from the rest of the quarter can be used as well!

Just ask “I want a good answer” instead of “I want the best”

Some problems can be approximated well, others can’t (unless $\mathrm{P} = \mathrm{NP}$).

## How To Approach Problems

In section, we’ve made you follow these steps:

1. Read the problem carefully (make sure you know what problem you’re actually solving)

2. Make some sample inputs/outputs

3. Set a “baseline.”

4. Then try to generate the algorithm.

It’s hard to take the time to do these in an exam, but at least make sure you do #1. Solving the wrong problem is not good for test-taking.

PLEASE do these steps in real life.

# What have you learned?

Algorithm design techniques/paradigms

Ways to frame your thinking to design a new algorithm

Methods of modeling new problems in terms of familiar ones.

Solve a new problem without (many) new ideas.

How to approach a problem

Techniques to avoid getting stuck or get yourself un-stuck.

Some famous algorithms

You’ll know Ford-Fulkerson, Bellman-Ford, etc. when you see them in libraries.

Or at least where to look them up.

## What should you do next?

CSE 431 (complexity theory)

What ***can't*** you do? (in polynomial time, at all, or in limited memory)

CSE 422

Toolkit for modern algorithms: algorithmic principles behind modern stats and ML

CSE 426 [490C]

Cryptography: a mix of math, algorithms, and complexity.

CSE 521 and 525

Graduate level courses in algorithms and randomized algorithms.

Look ahead! These courses usually run once-per-year.