CSE 417
Practical Algorithms
(a.k.a. Algorithms & Computational Complexity)

UNIVERSITY of WASHINGTON
Outline for Today

- Course Goals & Overview
- Administrivia
- Greedy Algorithms
Why study algorithms?

> Learn the history of important algorithms
Why study algorithms?

- Learn the history of important algorithms
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
- Impress your friends with your knowledge
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
- Impress your friends with your knowledge
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
- Impress your friends with your knowledge
- Inventing new algorithms is part of the everyday work of computer scientists in practice
Why study algorithms?

➢ Learn the history of important algorithms
➢ Appreciate their beauty
➢ Impress your friends with your knowledge
➢ Inventing new algorithms is part of the everyday work of computer scientists in practice
Why study algorithms?

- Learn the history of important algorithms
- Appreciate their beauty
- Impress your friends with your knowledge
- Inventing new algorithms is part of the everyday work of computer scientists in practice
- Algorithms are critical to the successful use of computers in every subfield of CS
Applications of Important Algorithms

> compilers
> databases
> networking
> cryptography
> AI & machine learning
> computational biology
> signal processing
> computer graphics
> scientific computing
> web search
> big data analytics
> ...
Applications of Important Algorithms

“Everyone knows Moore’s law — a prediction made in 1965 by Intel co-founder Gordon Moore that the density of transistors in integrated circuits would continue to double every 1 to 2 years.... In many areas, performance gains due to improvements in algorithms have vastly exceeded the dramatic performance gains due to increased processor speed.”

— Excerpt from Report to the President and Congress: Designing a Digital Future, December 2010 (page 71)
Why study algorithms?

> Algorithms are critical to the successful use of computers in every subfield of computer science
  - understanding the underlying techniques will make these algorithms easier to follow when you encounter them in other subfields

> There will be opportunities to invent new algorithms in practice
  - and when you do so, it often has a huge impact
  - the previous slides are examples of this

> (Algorithms also come up in coding interviews.)
Course Goal

> Teach you techniques that you can use to create new algorithms in practice when the opportunity arises
  – (or in coding interviews)
  – they will also help you understand existing algorithms
Course Non-Goals

> Teach you the most historically important algorithms
  – we will see some important algorithms, but this will not be a survey course on important algorithms

> Teach you the *fastest* known algorithms for problems
  – they are usually not the best demonstrations of the techniques that we want to discuss
Course Topics

1. Shortest Paths (mainly in HW)
2. Binary Search
3. Divide & Conquer
4. Dynamic Programming
5. Network Flows
6. Branch & Bound
Course Topics

**Design Techniques**
1. Divide & Conquer
2. Dynamic Programming
3. Branch & Bound

**Modeling Techniques**
1. Shortest Paths
2. Binary Search
3. Network Flows
Course Topics

Design Techniques
1. Divide & Conquer
2. Dynamic Programming
3. Branch & Bound

Techniques that you can apply to design new algorithms
   – each of these has a good chance of being useful in practice
Course Topics

Modeling Techniques
1. Shortest Paths
2. Binary Search
3. Network Flows

Solve new problems by transforming them into familiar ones
– these three are the most likely to show up in practice
– learning to recognize them is a useful skill
Course Topics vs Usual

1. Binary Search
2. Divide & Conquer
3. Dynamic Programming
4. Network Flows
5. Branch & Bound

1. Greedy
2. Divide & Conquer
3. Dynamic Programming
4. Network Flows
5. NP-Completeness
Course Topics vs Usual

1. Binary Search
2. Divide & Conquer
3. Dynamic Programming
4. Network Flows
5. Branch & Bound
   – (and NP-completeness)

1. Greedy (today only)
2. Divide & Conquer
3. Dynamic Programming
4. Network Flows
5. NP-Completeness

Topics will be fairly standard, but emphasis will be different
Outline for Today

- Course Goals & Overview
- Administrivia
- Greedy Algorithms
# People

- **Instructor**: Kevin Zatloukal, kevinz at cs
- **TAs**: Phillip Dang, phdang1 at uw
  Angli Liu, anglil at uw
  Alon Milchgrub, alonmil at uw
About Me

> UW graduate

> Ph.D. from MIT
  – quantum algorithms

> Worked in industry for 15 years
  – Google, Microsoft, BEA Systems, startup
Prerequisites (from CSE 373)

> Asymptotic complexity and big-O notation

> Familiar with some algorithms for
  – sorting
  – shortest paths
  – minimum spanning trees
Format

> Lectures Mon, Wed, Fri
  – slides will be posted
  – but they are just visual aids

> No quiz sections

> Office hours Tue, Wed, Thu, Fri
Workload

> 9 homework assignments
  – 4 on paper
  – 5 coding

> final exam (no midterm)
Grading

> 25% written assignments
> 50% coding assignments
> 25% final exam

Coding assignments best test the skills I care about
Late Policy

> 10 percent penalty per late day
  – but life happens so...

> 3 free late days for the quarter
  – each is a 24 hour extension
  – save for true emergencies
Collaboration

> Discussing course content with others is encouraged
> Be sure to use the Piazza discussion board

> BUT assignments are to be completed **individually**
  – misrepresenting others’ work as your own is academic misconduct
> See the course web site for detailed policy
Main source of info:
- OO times & locations
- assignments
- calendar
- link to Piazza (discussion board)

> https://courses.cs.washington.edu/courses/cse417/18wi
Textbook

- *Algorithm Design* by Kleinberg & Tardos
  - good introduction and useful reference

- Many other good books:
  - e.g. *Introduction to Algorithms* by Cormen, Leiserson, & Rivest

- I may also make use of other sources, especially:
  - *Combinatorial Optimization* by Papadimitriou & Steiglitz
  - *Network Flows* by Ahuja, Magnanti, & Orlin
  - both are advanced texts but could be useful after the course
Other Materials

- Tim Roughgarden (Stanford) has video lectures posted on youtube

- They cover many of the topics that we will discuss
  - He also has lectures from follow-on courses on some more advanced topics

- Also testing out recording of these lectures...
Warning

> Designing this version of the course from scratch
  – all new assignments, some new topics, etc.

> Will need your feedback
  – reasonableness of workload
  – choice of due dates
  – topics that do and don’t make sense
  – etc.
Outline for Today

- Course Goals & Overview
- Administrivia
- Greedy Algorithms
What is a *greedy* algorithm?

> For problems that involve a series of **choices**
>   – often an optimization problem (e.g., max or min something)

> Greedy approach makes each decision in a way that is optimal for that **individual choice** ignoring implications for future choices

> Rarely does this ever produce optimal solutions...
Real-life example

> When the alarm goes off in the morning, hitting the snooze button six times in a row seems like a great idea
  > maximizes happiness during that time period

> But those decisions affect future options available...

> Sleeping through lecture is probably not optimal
What is a *greedy* algorithm?

- The greedy approach rarely produces optimal solutions
  But sometimes it does!

- Those are **greedy algorithms**
  - they compute the actual solution
  - (otherwise, it is called a “greedy heuristic”)
Example: minimum spanning tree

> **Input:** connected, weighted graph G with n nodes

> let F be an empty graph on the same nodes
> for i = 1 to n – 1:
>     add the lowest weight edge of G that does not create a cycle

> This is Kruskal's algorithm
>     returns a spanning tree: a graph with n – 1 edges and no cycles

> For each of the n – 1 decisions of which edge to add, it chooses the edge of lowest weight
>     no regard for implications on future choices
Example: minimum spanning tree

> Clear that it returns a spanning tree
> Not clear that it returns the minimum weight spanning tree

> Choices we make in one iteration affect later iterations
  – by picking the lowest weight edge in one cycle, we could miss out on an edge that we need in a later iteration
  – adding one edge means some others won’t be allowed in the future because they now create a cycle

> We need an explanation of why this can’t happen...
Kruskal: proof of correctness

> Suppose that we pick edge $e$ that is not in the MST
> Removing $e$ from $F$ would disconnected graph into $S$ & $T$
Kruskal: proof of correctness

> Suppose that we pick edge $e$ that is not in the MST
> Removing $e$ from $F$ would disconnected graph into $S$ & $T$
> In the MST, adding $e$ would create a cycle spanning $S$ & $T$
> Let $f$ be another edge on this cycle that connects $S$ and $T$:
Kruskal: proof of correctness

> At the point in Kruskal’s algorithm when $e$ was added:
  > - graph $F$ had a subset of the final edges, so $S$ & $T$ were disconnected
  > - hence, adding $f$ would not have created a cycle either... so it was eligible to add
> Since $e$ had lowest weight amongst those: $\text{weight } e \leq \text{weight } f$
Replacing $f$ with $e$ in the MST cannot increase its weight
Hence, there is an MST that includes $e$!
Now, repeat this for every edge until the MST is our tree...

**Kruskal: proof of correctness**
Why not cover greedy?

> Usually have complex proofs of correctness
  – skipped important details in the Kruskal example
  – better for a math course than this one (see MATH 409)

> Rarely encountered in nature
  – most have names you know: Dijkstra, Kruskal, Prim

> Lulls you into thinking the greedy approach is usually correct
  – it isn’t!
  – even when you aren’t looking for exact solutions, greedy heuristics are rarely the best approach
Reminders

> HW1 is posted
  – due next Wednesday

> Information on overloading on Friday