Lecture 18
what we’ve done and what’s to come
Announcements

• HW8 (last one) due today

• Don’t forget about the final exam on March 16 (from 3:30pm – 6:30pm).

• Two pages of handwritten notes (front and back) allowed for the final exam.
Today

- What just happened?
  - A whirlwind tour of CS161

- What’s next?
  - A few gems from future algorithms classes
It’s been a fun ride...

- Sorting and friends!
- Divide-and-conquer and recurrence relations
- \(O()\) and worst-case analysis
- Data structures: BSTs and Hashing!
- Randomized algorithms
- BFS, DFS, SCCs
- Graphs!
- Dynamic Programming!
- Greedy algorithms!
- Scheduling and etc.
- MinCuts and MaxFlows
- Stable Matchings
- LCS, Knapsack(s)
- Bellman-Ford, Floyd-Warshall
- MSTs: Prim and Kruskal
- Ford-Fulkerson
- Dijkstra’s algorithm
- Greedy algorithms!
- MinCuts and MaxFlows
What have we learned?

17 lectures in 12 slides.
General approach to algorithm design and analysis

Can I do better?

To answer this question we need both **rigor** and **intuition**:

- **Algorithm designer**
  - Detail-oriented
  - Precise
  - Rigorous

- **Plucky the Pedantic Penguin**
  - Big-picture
  - Intuitive
  - Hand-wavey

- **Lucky the Lackadaisical Lemur**
  - **rigor** and **intuition**:
We needed more details

Does it work?
Is it fast?

What does that mean??

Worst-case analysis

big-Oh notation

Here is an input!

\[ T(n) = O(f(n)) \]
\[ \iff \exists c, n_0 > 0 \text{ s.t. } \forall n \geq n_0, \quad 0 \leq T(n) \leq c \cdot f(n) \]
Algorithm design paradigm: divide and conquer

- Like MergeSort!
- Or Karatsuba’s algorithm!
- Or SELECT!
- How do we analyze these?

By careful analysis! Useful shortcut, the master method is.

Plucky the Pedantic Penguin  Jedi master Yoda
While we’re on the topic of sorting
Why not use randomness?

• We analyzed **QuickSort!**
• Still worst-case input, but we use randomness after the input is chosen.
• Always correct, usually fast.
  • This is a Las Vegas algorithm
All this sorting is making me wonder...
Can we do better?

• Depends on who you ask:

  • RadixSort takes time $O(n)$ if the objects are, for example, small integers!
  • Can’t do better in a comparison-based model.
beyond sorted arrays/linked lists: Binary Search Trees!

• Useful data structure!
• Especially the self-balancing ones!

Red-Black tree!

Maintain balance by stipulating that black nodes are balanced, and that there aren’t too many red nodes.

It’s just good sense!
Another way to store things

Hash tables!

All of the hash functions
\( h : U \rightarrow \{1, \ldots, n\} \)

Choose \( h \) randomly from a universal hash family.

It’s better if the hash family is small!
Then it takes less space to store \( h \).
OMG GRAPHS

• BFS, DFS, and applications!
• SCCs, Topological sorting, ...
A fundamental graph problem: shortest paths

- E.g., transit planning, packet routing, ...
- Dijkstra!
- Bellman-Ford!
- Floyd-Warshall!
• Not programming in an action movie.

• **Step 1:** Identify **optimal substructure**.

• **Step 2:** Find a **recursive formulation** for the value of the optimal solution.

• **Steps 3-5:** Use dynamic programming: fill in a table to find the answer!

Bellman-Ford and Floyd-Warshall were examples of...

We saw many other examples, including Longest Common Subsequence and Knapsack Problems.
Sometimes we can take even better advantage of optimal substructure...with

**Greedy algorithms**

- Make a series of choices, and commit!

- Intuitively we want to show that our greedy choices never rule out success.

- Rigorously, we usually analyzed these by induction.

- Examples!
  - Activity Selection
  - Job Scheduling
  - Huffman Coding
  - Minimum Spanning Trees

**Prim’s algorithm:** greedily grow a tree

**Kruskal’s algorithm:** greedily grow a forest
Cuts and flows

- Minimum s-t cut:
  - is the same as maximum s-t flow!
  - Ford-Fulkerson can find them!
    - useful for routing
    - also assignment problems
Stable matching

How to convince actors to use our matching?
Where do preferences come from?
Are the incentives set correctly?

Deferred acceptance: a different kind of greedy algorithm, this time with recourse.
And now we’re here
What have we learned?

• A few algorithm design paradigms:
  • Divide and conquer, dynamic programming, greedy

• A few analysis tools:
  • Worst-case analysis, asymptotic analysis, recurrence relations, probability tricks, proofs by induction

• A few common objects:
  • Graphs, arrays, trees, hash functions

• A LOT of examples!
What have we learned?
We’ve filled out a toolbox

• Tons of examples give us intuition about what algorithmic techniques might work when.
• The technical skills make sure our intuition works out.
But there's lots more out there

• What's next???
A taste of what’s to come

- CS154 – Introduction to Automata and Complexity
- CS163 – The Practice of Theory Research
- CS166 – Data Structures
- CS168 – The Modern Algorithmic Toolbox
- MS&E 212 – Combinatorial Optimization
- CS250 – Error Correcting Codes
- CS252 – Analysis of Boolean Functions
- CS254 – Computational Complexity
- CS255 – Introduction to Cryptography
- CS259Q – Quantum Computing
- CS260 – Geometry of Polynomials in Algorithm Design
- CS261 – Optimization and Algorithmic Paradigms
- CS263 – Counting and Sampling
- CS265 – Randomized Algorithms
- CS269O – Introduction to Optimization Theory
- MS&E 316 – Discrete Mathematics and Algorithms
- CS352 – Pseudorandomness
- CS366 – Computational Social Choice
- CS368 – Algorithmic Techniques for Big Data
- EE364A/B – Convex Optimization I and II

...and many many more!

findSomeTheoryCourses():
- go to theory.stanford.edu
- Click on “People”
- Look at what we’re teaching!
Today
A few gems

- Linear programming
- Random projections
- Low-degree polynomials

This will be fluffy, without much detail – take more CS theory classes for more detail!
Linear Programming

• This is a fancy name for optimizing a linear function subject to linear constraints.

• For example:

Maximize

\[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]

• It turns out to be an extremely general problem.
We’ve already seen an example!

Maximize
the sum of the flows leaving $s$

subject to
• None of the flows are bigger than the edge capacities
• At every vertex, stuff going in = stuff going out.
Linear Programming
Has a really nice geometric intuition

Maximize
\[ x + y \]
subject to
\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]
Linear Programming

Has a really nice geometric intuition

Maximize

\[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]
Linear Programming
Has a really nice geometric intuition

Maximize
\[ x + y \]
subject to
\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]

The function is maximized here!
In general

• The constraints define a polytope
• The function defines a direction
• We just want to find the vertex that is furthest in that direction.
Duality

How do we know we have an optimal solution?

I claim that the optimum is 5/7.

Proof: say x and y satisfy the constraints.

- \[ x + y = \frac{1}{7} (4x + y) + \frac{3}{7} (x + 2y) \]
- \[ \leq \frac{1}{7} \cdot 2 + \frac{3}{7} \cdot 1 \]
- \[ = \frac{5}{7} \]

Maximize \[ x + y \]

subject to

\[ x \geq 0 \]
\[ y \geq 0 \]
\[ 4x + y \leq 2 \]
\[ x + 2y \leq 1 \]

You can check this point has value 5/7...but how would we prove it’s optimal other than by eyeballing it?
cute, but

How did you come up with $1/7, 3/7$?

I claim that the optimum is $5/7$.

**Proof:** say $x$ and $y$ satisfy the constraints.

- $x + y \leq (4x + y) + (x + 2y)$
- $\leq 2 + 1$
- $= \frac{5}{7}$

I want to choose things to put here

So that I minimize this

Subject to these things

Maximize $x + y$

subject to

$x \geq 0$

$y \geq 0$

$4x + y \leq 2$

$x + 2y \leq 1$
That’s a linear program!

• How did I find those special values 1/7, 3/7?
• I solved some linear program.
• It’s called the dual program.

Minimize the upper bound you get, subject to the proof working.

Note: it’s not immediately obvious how to turn that into a linear program, this is just meant to convince you that it’s plausible.

In this case the dual is:
\[
\min 2w + z \text{ s.t. } w, z \geq 0, \\
4w + z \geq 1 \text{ and } w + 2z \geq 1
\]
We’ve actually already seen this too

The Min-Cut Max-Flow Theorem!

Maximize the sum of the flows leaving $s$

s.t.

All the flow constraints are satisfied

Minimize the sum of the capacities on a cut

s.t.

it’s a legit cut

The optimal values are the same!
LPs and Duality are really powerful

• This **general phenomenon** shows up all over the place
  • Min-Cut Max-Flow is a special case.

• Duality helps us reason about an optimization problem
  • The dual provides a **certificate** that we’ve solved the primal.
  • E.g., if you have a cut and a flow with the same value, you must have found a max flow and a min cut.

• We can solve LPs quickly!
  • For example, by intelligently bouncing around the vertices of the feasible region.
  • This is an **extremely powerful algorithmic primitive**.
Today
A few gems

• Linear programming

• Random projections

• Low-degree polynomials
A very useful trick
Take a random projection and hope for the best.

High-dimensional set of points
For example, each data point is a vector
(age, height, shoe size, ...)

Their shadow is a projection onto the ground.
Why would we do this?

• High dimensional data takes a long time to process.
• Low dimensional data can be processed quickly.
• “THEOREM”: Random projections approximately preserve properties of data that you care about.
Example: nearest neighbors

- I want to find which point is closest to **this one**.

Johnson-Lindenstrauss Lemma:
*Euclidean distance is approximately preserved by random projections.*

That takes a really long time in high dimensions.
Another example:

Compressed Sensing

• Start with a sparse vector
  • Mostly zero or close to zero

(5, 0, 0, 0, 0.01, 0.01, 5.8, 32, 14, 0, 0, 0, 12, 0, 0, 5, 0, .03)

• For example:

This image is sparse

This image is sparse after I take a wavelet transform.
Compressed sensing continued

• Take a random projection of that sparse vector:

Random short fat matrix

Long sparse vector

Goal: Given the **short vector**, recover the **long sparse vector**.
Why would I want to do that?

- Image compression and signal processing
- Especially when you **never have space to store the whole sparse vector to begin with**.

Randomly sampling (in the time domain) a signal that is sparse in the Fourier domain.

Random measurements in an fMRI means you spend less time inside an fMRI

A “single pixel camera” is a thing.
All examples of this:

Goal: Given the short vector, recover the long sparse vector.
But why should this be possible?

- There are tons of long vectors that map to the short vector!

Goal: Given the **short vector**, recover the **long sparse vector**.
Back to the geometry

Theorem:
random projections preserve the geometry of sparse vectors too.
If we don’t care about algorithms, that’s more than enough.

This means that, in theory, we can invert that arrow.

How do we do this efficiently??

All of the sparse vectors

Multiply by Random short fat matrix

There may be tons of vectors that map to this point, but only one of them is sparse!
An efficient algorithm?

What we’d like to do is:

Minimize number of nonzero entries in \( x \)

\[ \text{Minimize } \|x\|_1 \quad \text{s.t.} \quad Ax = y \]

This norm is the sum of the absolute values of the entries of \( x \)

This isn’t a nice function

Instead:

Minimize \( \|x\|_1 \)

\[ \text{s.t.} \quad Ax = y \]

Problem: I don’t know how to do that efficiently!

• It turns out that because the geometry of sparse vectors is preserved, this optimization problem gives the same answer.

• We can use linear programming to solve this quickly!
Today
A few gems

• Linear programming
• Random projections
• Low-degree polynomials
Another very useful trick

Polynomial interpolation

• Say we have a few evaluation points of a low-degree polynomial.

• We can recover the polynomial.
  • 2 pts determine a line, 3 pts determine a parabola, etc.

• We can recover the whole polynomial really fast.

• Even works if some of the points are wrong.
One application: Communication and Storage

- Alice wants to send a message to Bob

“Hi, Bob!”

\[ f(x) = H + I \cdot x + B \cdot x^2 + O \cdot x^3 + B \cdot x^4 \]
This is used in practice

• It’s called “Reed-Solomon Encoding”
Another application:
Designing “random” projections that are better than random

Random short fat matrix

= 

The matrix that treats the big long vector as Alice’s message polynomial and evaluates it REALLY FAST at random points.

• This is still “random enough” to make the LP solution work.
• It is much more efficient to manipulate and store!
Today
A few gems
• Linear programming
• Random projections
• Low-degree polynomials

To learn more:
CS168, CS261, ...
CS168, CS261, CS265, ...
CS168, CS250, ...
What have we learned?

Tons more cool algorithms stuff!
To see more...

• Take more classes!
• Come hang out with the theory group!
  • Theory lunch, most Thursdays at noon.
  • Join the theory-seminar mailing list for updates.

[Image of group photo]

theory.stanford.edu

Stanford theory group (circa 2017):
We are very friendly.
A few final messages...
Thanks to our course coordinator Amelie Byun!

- Amelie has been making all the logistics work behind the scenes.
Thanks to Diana Acosta-Navas!

• Diana has been helping integrate EthiCS components into the course.
Thanks to our superstar CAs!!!
tell them you appreciate them!

Yu Shen  Avery  Manda  Amrita  Andre  Goli  Jerry  Jiazheng

Carmen  June  Andrew  Jose  Manda  Nash  Peter  Sam  Samar

Aditya  Emily  Yuchen  Ziang  Seiji  Shubham  Teresa  Tim
THANKS to you!!!!!!