Beyond Efficiency: Algorithmic Analysis and Social Impact

What has been the most interesting application of recursion that you've encountered so far? (put your answers the chat)
Today’s question

What problems should we solve with recursive backtracking?
Today’s topics

1. Review and Recursion Overview
2. Beyond Efficiency: Algorithmic Analysis and Social Impact
3. Optimization and Gerrymandering
Solving mazes with Depth-First Search (DFS)
What defines our maze decision tree?

- **Decision** at each step (each level of the tree):
  - Which valid move will we take?

- **Options** at each decision (branches from each node):
  - All valid moves (in bounds, not a wall, not previously visited) that are either North, South, East, or West of the current location

- **Information we need to store along the way:**
  - The path we’ve taken so far (a Stack we’re building up)
  - Where we’ve already visited
  - Our current location
Pseudocode

- Our helper function will have as **parameters**: the maze itself, the path we’re building up, and the current location.
  - **Idea**: Use the boolean Grid (the maze itself) to store information about whether or not a location has been visited by flipping the cell to false once it’s in the path (to avoid loops) → This works with our existing `generateValidMoves()` function

- **Recursive case**: Iterate over valid moves from `generateValidMoves()` and try adding them to our path
  - If any recursive call returns true, we have a solution
  - If all fail, return false

- **Base case**: We can stop exploring when we’ve reached the exit → return true if the current location is the exit
Breadth-First Search vs. Depth-First Search

Which do you think will be faster?
Option 2: Include this person
You need at least five US Supreme Court justices to agree to set a precedent.

What are all the ways you can pick five justices of the US Supreme Court?
Combinations versus Subsets

- Making combinations is very similar to our recursive process for generating subsets!

- The differences:
  - We’re constraining the subsets’ size.
  - We’re building up a set of all valid subsets of that particular size (i.e. combinations).

- Instead of printing out subsets in our base case, we have to return individual sets in our base case and then build up and return our resulting set of sets in our recursive case.
You now know how to use recursion to view problems from a different perspective that can lead to short and elegant solutions.
Organizing Your Recursive Toolbox
Two types of recursion

**Basic recursion**
- One repeated task that builds up a solution as you come back up the call stack
- The final base case defines the initial seed of the solution and each call contributes a little bit to the solution
- Initial call to recursive function produces final solution

**Backtracking recursion**
- Build up many possible solutions through multiple recursive calls at each step
- Seed the initial recursive call with an “empty” solution
- At each base case, you have a potential solution
We’ve seen lots of different backtracking strategies...

Questions to ask yourself when planning your strategy:

- What does my decision tree look like? (decisions, options, what to keep track of)
- What are our base and recursive cases?
- What’s the provided function prototype and requirements? Do we need a helper function?
- Do we care about returning or keeping track of the path we took to get to our solution?
- Which of our three use cases does our problem fall into? (generate/count all solutions, find one solution/prove its existence, pick one best solution)
- What are we returning as our solution? (a boolean, a final value, a set of results, etc.)
- What are we building up as our “many possibilities” in order to find our solution? (subsets, permutations, combinations, or something else)
Backtracking recursion: **Exploring many possible solutions**
Overall paradigm: choose/explore/unchoose

**Two ways of doing it**

- **Choose explore undo**
  - Uses pass by reference; usually with large data structures
  - Explicit unchoose step by "undoing" prior modifications to structure
  - E.g. Generating subsets (one set passed around by reference to track subsets)

- **Copy edit explore**
  - Pass by value; usually when memory constraints aren’t an issue
  - Implicit unchoose step by virtue of making edits to copy
  - E.g. Building up a string over time

**Three use cases for backtracking**

1. Generate/count all solutions (enumeration)
2. Find one solution (or prove existence)
3. Pick one best solution

General examples of things you can do:
- Permutations
- Subsets
- Combinations
- etc.
You’ve seen how to use recursive backtracking to determine whether something is possible and, if so, to find some way to do it.
You’ve seen how to use recursive backtracking to **enumerate all objects of some type**, which you can use to find the **optimal solution** to a problem.
The Knapsack Problem
Pseudocode

- **Recursive case:**
  - Select an unconsidered item based on the index.
  - Recursively calculate the values both with and without the item.
  - Return the higher value.

- **Base cases:**
  - No remaining capacity in the knapsack → return 0
    (not a valid combination with weight <= 5)
  - No more items to choose from → return current value
Let’s see the code!
Takeaways

● Finding the best solution to a problem (optimization) can often be thought of as an additional layer of complexity/decision making on top of the recursive enumeration we've seen before.

● For "hard" problems, the best solution can only be found by enumerating all possible options and selecting the best one.

● Creative use of the return value of recursive functions can make applying optimization to an existing function straightforward.
Learning goals

After completing this assignment, you will be able to:

- Appreciate the elegance and power of recursive problem-solving and identify problems that are well-suited to be solved recursively.
- Understand and trace how data is stored and altered across multiple recursive function calls.
- Identify and carry out techniques for testing and debugging recursive functions.
- Break down a problem into a collection of smaller, self-similar tasks.
- Develop a recursive algorithm by dividing a problem into one or more base cases and one or more recursive cases.
- Apply the general frameworks of recursive backtracking to solve problems that cannot easily be solved using an iterative approach.
- Implement more advanced recursive algorithms to solve problems that cannot be easily solved using an iterative approach.

Assignment parts

This assignment consists of a collection of multiple different recursive exercises, grouped into two parts.

- Part 1: Fundamental Recursion
  The first collection of recursive problems walks you through some fundamental recursion problems that increase in difficulty and complexity over time. These problems will serve as the foundational practice for you to begin to grapple with the core tenants of recursive problem-solving. The material needed to solve these problems was covered in lecture on Tuesday-Thursday (7/6-7/8).
  - **Fundamental Recursion Warmup**
    Practice with unit tests and debugging on core recursive functions.
  - **Balanced Operators**
    Determine whether a snippet of code has properly matched pairs of bracketing characters.
  - **Sierpinski Fractal**
    Draw a beautiful self-similar fractal triangle.
  - **Merging Sorted Sequences**
    Implement an efficient divide-and-conquer algorithm for merging a collection of sorted sequences.

- Part 2: Recursive Backtracking
  The second collection of recursive problems pivots towards applying the powerful nature of recursive backtracking to solve some real-world problems. After you've built your solid base of recursive fundamentals in Part 1, you will be well set to apply your newfound skills to solve some practical real-world problems. The material needed to work through the
GET OUT THERE & MAKE SOMETHING!
Algorithmic Bias Poll

(a) Biased Input

(b) Biased Output

(c) Biased Input
Software is ubiquitous
North America Machine Learning Market Size, 2017-2028 (USD Billion)
Error-riddled data sets are warping our sense of how good AI really is

Our understanding of progress in machine learning has been colored by flawed testing data.

by Karen Hao

April 1, 2021

The 10 most cited AI data sets are riddled with label errors, according to a new study out of MIT, and it’s distorting our understanding of the field’s progress.

Data backbone: Data sets are the backbone of AI research, but some are more critical than others. There are a core set of them that researchers use to evaluate machine-learning models as a way to track how AI capabilities are advancing over time. One of the best-known is the canonical image.
here's a fun fact

d'hey's a funky introduction of how nice i am
Check The Rhime — Song by A Tribe Called Quest

d'hey's a fun fact gif

d'hey's a funky introduction of how nice i am lyrics

d'hey's a fun fact eurotrip

d'hey have a fungus

d'hey have a fungus meme

d'hey for a fun time

d'hey is a function machine

d'hey comes a funny
WE'RE HIRING

READ MORE
Shortest-Path Algorithms

- Find the “shortest” path from point A to point B
- “Shortest” in time, distance, cost, ...
- Numerous applications
  - Map navigation
  - Flight itineraries
  - Circuit wiring
  - Network routing
With soaring demand for social services, field-tested technology can help governments scale up
Design Justice
Who might be impacted by the software?

Who should be at the table helping to design the software?
Gaining Perspective
Breakout Room Discussion #1

There are two discussion prompts:

- Does the background of the programmer matter?

- How can we involve those impacted in development of software?
Moral Relativity

Who decides the target audience?

Who needs the app or software the most?

If there are unintended consequences, are these consequences fairly distributed among groups of people?

How do we define fairness?

If we need funds to develop our software, who is able to buy it and does the cost to develop it inherently make it inequitable?

Should the government play a role in regulating this?
MAN ACCUSED OF USING SNAPCHAT TO SELL DRUGS
TEWKSBURY
DOWNLOAD THE NBC10 BOSTON APP
Breakout Room Discussion #2

Would software usage labels be helpful?

What would the labels say?
Announcements
Announcements

• The mid-quarter diagnostic is coming up at the end of this week.
  • You will have a 72-hour period of time from Friday to Sunday to complete the diagnostic.
  • The diagnostic is designed to take about an hour and a half to complete, but you can have up to 3 hours to work on it if you so choose.
  • The diagnostic will be administered digitally using Gradescope.
  • A practice diagnostic and many additional review materials have been posted on the diagnostic page.

• Assignment 3 is due on Thursday, July 15 at 11:59pm.
Gerrymandering & Algorithmic Thinking

based on slides created by Katie Creel
The Law of the Relational Database

If the only tool you have is a relational database, everything looks like a table.

A Walk in Graph Databases - 2012
Today’s question

What problems should we solve with recursive backtracking?
History of Voter Suppression
15th Amendment to the US Constitution (1870)

The “right of citizens of the United States to vote shall not be denied or abridged by the United States or by any state on account of race, color, or previous condition of servitude.”

Addresses racial gerrymandering, which abridges the right to vote on account of race.
US Population by Region, 1790, 2000
Stacked area (in millions)
Redistricting for House Elections

435 Districts of (roughly) equal population. Each elects a Representative.

... but populations shift over time.

US Census measures population every 10 years.

... so redistrict based on the new Census.
Precincts: tiny boxes
Districts: bigger ... boxes?
In order to find out, we need to understand the ... HISTORY OF GERRYMANDERING
“The Gerry Mander” (1812)
The Computational Problem of Redistricting

1. count residents (Census)
2. apportion Congressional Representatives to the states
3. partition the states by sorting precincts into districts

The Math (combinatorics) Problem of Redistricting

If a state has \( n \) residents (with attributes) and \( k \) Representatives, how to form \( k \) groups of approximately size \( n \) out of census blocks \( k \) in accordance with various rules and values.
Ways Voting Rights can be Restricted

- There are four districts, each with 9 people.
- 36 people total
- 20 are in the majority group, 16 are in the minority group
Ways Voting Rights can be Restricted

- If the minority and majority groups in are politically polarized and tend to prefer different parties, we would expect 2 candidates from each party to be elected.
Racially Polarized Voting

only 1 minority-preferred candidate elected

0 minority-preferred candidates elected
1965: Voting Rights Act

“The Civil Rights Division has the responsibility for enforcement of provisions of the Voting Rights Act that seek to ensure that redistricting plans do not discriminate on the basis of race, color, or membership in a protected language minority group.”

“[The Civil Rights Act] prohibits not only election-related practices and procedures that are intended to be racially discriminatory, but also those that are shown to have a racially discriminatory result.”
Republican lawmakers in the state, after the 2010 census, had redrawn the map to add more black voters into Districts 1 and 12. The Supreme Court concluded 5-3 that North Carolina violated the Equal Protections Clause of the 14th Amendment by separating voters in different districts on the basis of race without “sufficient justification” for doing so.
Partisan Gerrymandering defined

This is redistricting for the purpose of gaining or preserving an advantage for one political party at the cost of equitable political representation for voters.
Why has Gerrymandering Gotten Worse?

Veteran redistricter: “Give the chairman of a state redistricting committee a powerful enough computer and neighborhood-block-level Census data, so that he suddenly discovers he can draw really weird and aggressive districts—and he will.”

Software + big data making a problem worse?

- New Software: Maptitude, RedAppl, and autoBound
- New Data: block-by-block census data
What Problem-Solving Technique or Algorithm Should We Use?
Recursion to the Rescue!

- Exhaustive algorithm -- generate all the possible solutions, constrained by our rules and ideally matched to our values.
- Recursive backtracking (choose-explore-unchoose) to find all the potential districting maps
- Optimize: identify the worst (or best!) maps
CONSTRAINTS AND PRINCIPLES

What are our metrics?

How should we redistrict responsibly?
Principle 1: One person, one vote

“... as nearly as is practicable one man’s vote in a Congressional election is worth as much as another’s.”
One Person, One Vote: The Efficiency Gap Metric

• Any vote cast for the losing party is a wasted vote for that party

• Any vote cast for the winning party that was more than the simple majority needed to win is also a wasted vote for that party

<table>
<thead>
<tr>
<th>District</th>
<th>D Votes</th>
<th>R Votes</th>
<th>D Wasted Votes</th>
<th>R Wasted Votes</th>
<th>Net Wasted Votes</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>75</td>
<td>25</td>
<td>24</td>
<td>25</td>
<td>1 R</td>
</tr>
<tr>
<td>2</td>
<td>60</td>
<td>40</td>
<td>9</td>
<td>40</td>
<td>31 R</td>
</tr>
<tr>
<td>3</td>
<td>43</td>
<td>57</td>
<td>43</td>
<td>6</td>
<td>37 D</td>
</tr>
<tr>
<td>4</td>
<td>48</td>
<td>52</td>
<td>48</td>
<td>1</td>
<td>47 D</td>
</tr>
<tr>
<td>5</td>
<td>49</td>
<td>51</td>
<td>49</td>
<td>0</td>
<td>49 D</td>
</tr>
<tr>
<td>Total</td>
<td>275</td>
<td>225</td>
<td>173</td>
<td>72</td>
<td>101 D</td>
</tr>
</tbody>
</table>
One Person One Vote: The Efficiency Gap Metric

The circled votes on the top are “wasted” in that they didn’t influence the outcome in that district.

6 Congressional Districts
Lime Party wins 4 Congressional seats.
Teal Party wins 2 seats.
CALCULATING THE EFFICIENCY GAP

Efficiency Gap (EG) =

\[
\text{abs}(\text{wasted green votes} - \text{wasted teal votes}) \over \text{total votes cast}
\]
Principle 2: Communities of Interest

Interpretation A:
a community of interest is a community that shares an identity group

Interpretation B:
a community of interest is “compact” or geographically contiguous
Community of Interest: IL-04: “The Earmuffs”

- Follows a predominantly Latine community
- Created in the 1990s; elected the first Latine representative to Congress from the Midwest.
Community of Interest: IL-04: “The Earmuffs”

- Follows a predominantly Latine community
- Created in the 1990s; elected the first Latine representative to Congress from the Midwest.
Interpretation B: Compactness

- Geography also matters for political representation
- People who live in the same physical area often have interests in common just in virtue of their location
What are the Interests of Santa Clara County & CA District 18?

- Silicon Valley
- Housing & Land Use Policy
- Transportation
Silicon Valley: A Different Story

- Semiconductors & microprocessor manufacturing (silicon!) in the 1950s-1990s by Fairchild, Hewlett-Packard, Intel, Apple, Atari, Xerox, etc.
- Hardware is mostly gone, but left behind are 23 Superfund sites contaminated with toxic chemicals
- Highest density of Superfund sites in the country
- Groundwater is safe, but plumes of toxic gas can escape, including at Google’s Quad Campus in 2012-2013
- **What do we in Santa Clara Country have in common?**
  A special desire to clean up our Superfund sites!
THE ROLE OF COMPUTER SCIENTISTS
What should we prioritize in redistricting? And what is a legitimate way to choose?

PRINCIPLE 1: ONE PERSON, ONE VOTE

PRINCIPLE 2: COMMUNITIES OF INTEREST
Who Should Control Redistricting?

- Biggest problem: US is one of the few countries in which politicians have control over redistricting.
- In most other countries, all redistricting is done by an independent commission.
- Politicians districting based on their own interests is unfair on either principle.
Consistency for Fairness

- However we balance these two, the same principles apply to every district, regardless of who is in the majority.
- Redistricting principles should not change based on local circumstances (or politician interests).
- Easier to implement consistency with a non-partisan independent commission.
Community Control for Justice

- Allow local communities to directly vote on or otherwise choose the balance of principles which should guide districting (and thus their own representation).

- Have an independent non-partisan commission implement the principles in drawing up a map.
Iterating and Revisiting our Solutions

- Sometimes our first algorithmic solution does not work as expected.
- The Voting Rights Act (VRA) attempts to indicate the fair middle ground between packing and cracking, but the interpretation of the VRA has been closer to packing.
Ethics of Care

- Few social problems can be solved exactly once by an algorithm.

- Any algorithmic solution should be revisited often as society changes and we understand its implications better.

- Caring for an algorithm, and for the community that relies on it, can mean updating the data ... or changing the algorithm and implementation.

- Creating “living” algorithms and machine learning models that can grow and change over time is a huge focus in data science and AI.
Roles you will be ready to take on after CS106B

- Make better systems yourself!
- Using your CS106B knowledge to advocate for communities affected by unfair practices that rely on algorithmic decision-making
- Formulating problems carefully based on knowledge of the history of the problem and understanding what people affected by the problem may see as a solution
What problems should we solve with recursive backtracking?
Congratulations on making it this far!
What’s next?
Roadmap

C++ basics

User/client

- vectors + grids
- stacks + queues
- sets + maps

Core Tools

- testing
- algorithmic analysis

Object-Oriented Programming

- arrays
- dynamic memory management
- linked data structures

Implementation

Life after CS106B!

- real-world algorithms
- recursive problem-solving

Diagnostic
vectors + grids
stacks + queues
sets + maps
arrays
dynamic memory management
linked data structures
real-world algorithms
recursive problem-solving
Life after CS106B!
C++ basics
User/client
Core Tools
testing
diagnostic analysis
Classes and Object-Oriented Programming