Today’s Keywords

Dynamic Programming
Gerrymandering
Greedy Algorithms
Choice Function
Change Making

CLRS Readings: Chapter 15, 16
Midterm take-home due **tonight** at 11pm
  - Individual work

No office hours / regrade office hours until tomorrow (after midterm)

**HW5** released later today, due **Thursday, October 24, 11pm**
  - Seam Carving
  - Dynamic Programming (implementation)
  - Java or Python
Dynamic Algorithms Examples

Maximum Sum Continuous Subarray
Tiling Dominoes
Log Cutting
Matrix Chaining
Longest Common Subsequence
Seam Carving
Observation: No need to recurse! Just maintain two numbers and iterate from 1 to $n$: best value so far, best value ending at current position

$$BED(n) = \max(BED(n - 1) + arr[n], 0)$$

$$BSL(n) = \max(BSL(n - 1), BED(n))$$
Two ways to fill the final column:

Tile\( (n) = \text{Tile}(n - 1) + \text{Tile}(n - 2) \)

Tile\( (0) = \text{Tile}(1) = 1 \)
Log Cutting

\[ P[i] = \text{value of a cut of length } i \]
\[ \text{Cut}(n) = \text{value of best way to cut a log of length } n \]

\[
\text{Cut}(n) = \max \left\{ \begin{array}{l}
\text{Cut}(n-1) + P[1] \\
\text{Cut}(n-2) + P[2] \\
\vdots \\
\text{Cut}(0) + P[n]
\end{array} \right\}
\]

\[
\text{Cut}(n - \ell_n)
\]

best way to cut a log of length \( n - \ell_n \)
Matrix Chaining

Best(1, n) = cheapest way to multiply together \( M_1 \) through \( M_n \)

Best(1,4) = min

\[
\begin{align*}
\text{Best}(2,4) + n_1n_2n_5 \\
\text{Best}(1,2) + \text{Best}(3,4) + n_1n_3n_5 \\
\text{Best}(1,3) + n_1n_4n_5
\end{align*}
\]

Last product: \( M_{13} \times M_4 \)
Longest Common Subsequence

Let LCS\((i, j)\) denote the length of the longest common subsequence between the first \(i\) characters of \(X\) and first \(j\) character of \(Y\).

\[
\text{LCS}(i, j) = \begin{cases} 
0 & \text{if } i = 0 \text{ or } j = 0 \\
\text{LCS}(i-1, j-1) + 1 & \text{if } X[i] = Y[j] \\
\max(\text{LCS}(i-1, j), \text{LCS}(i, j-1)) & \text{otherwise}
\end{cases}
\]

\(X = \text{A T C T G A T} \)  
\(Y = \text{A T C A T} \)

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>T</th>
<th>C</th>
<th>T</th>
<th>G</th>
<th>A</th>
<th>T</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>G</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>T</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>A</td>
<td>0</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>3</td>
<td>4</td>
</tr>
</tbody>
</table>

Run Time: \(\Theta(n \cdot m)\) (for \(|X| = n, |Y| = m\))
Suppose we know the least energy seams for all rows up to $n - 1$ (i.e., we know $S(n - 1, \ell)$ for all $\ell$)

$$p_{n,k} \quad S(n, k) = \min \begin{cases} S(n - 1, k - 1) + e(p_{n,k}) \\ S(n - 1, k) + e(p_{n,k}) \\ S(n - 1, k + 1) + e(p_{n,k}) \end{cases}$$
Gerrymandering

Manipulating electoral district boundaries to favor one political party over others

Coined in an 1812 political cartoon after Governor Gerry signed a bill that redistricted Massachusetts to benefit his Democratic-Republican Party

The Gerrymander
Supreme Court Associate Justice Anthony Kennedy gave no sign that he has abandoned his view that extreme partisan gerrymandering might violate the Constitution. (Eric Thayer/Getty Images)

**Supreme Court eyes partisan gerrymandering**

Anthony Kennedy is seen as the swing vote that could blunt GOP's map-drawing successes.
Gerrymandering

Next Gerrymandering Battle in North Carolina: Congress

A North Carolina court threw out the state's legislative map as an illegal gerrymander. Now the same court could force the state to redraw the state's congressional districts as well.
According to the Supreme Court...

Gerrymandering cannot be used to:
- Disadvantage racial/ethnic/religious groups

It can be used to:
- Disadvantage political parties
VA 5th District
VA 5th District

2018 Election

<table>
<thead>
<tr>
<th>Votes</th>
<th>Pct</th>
</tr>
</thead>
<tbody>
<tr>
<td>165,339</td>
<td>53.3%</td>
</tr>
<tr>
<td>145,040</td>
<td>46.7</td>
</tr>
</tbody>
</table>
Gerrymandering Today
Gerrymandering Today

Computers make it very effective

Is this even contiguous?
How Does it Work?

- States are broken into precincts
- All precincts have the same number of people
- We know voting preferences of each precinct
- Group precincts into districts to maximize the number of districts won by my party

Overall: R:217 D:183

100 voters per precinct

<table>
<thead>
<tr>
<th></th>
<th>R:65</th>
<th>D:35</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>R:45</td>
<td>D:55</td>
</tr>
<tr>
<td></td>
<td>R:47</td>
<td>D:53</td>
</tr>
</tbody>
</table>

(R) vs. (D)

Each district should have roughly the same number of people
How Does it Work?

• States are broken into precincts
• All precincts have the same number of people
• We know voting preferences of each precinct
• Group precincts into districts to maximize the number of districts won by my party

Overall: R:217 D:183

100 voters per precinct

Each district should have roughly the same number of people
Gerrymandering Problem Statement

Given:

- A list of precincts: \( p_1, p_2, \ldots, p_n \)
- Each precinct contains exactly \( m \) voters

Output districts \( D_1, D_2 \subset \{ p_1, p_2, \ldots, p_n \} \) where:

- \( |D_1| = |D_2| \)
- \( R(D_1), R(D_2) > \frac{mn}{4} \) where
  \( R(D_i) \) is the number of “Regular Party” voters in \( D_i \)

If no such assignment is possible, output \textit{impossible}
Dynamic Programming

Requires **optimal substructure**
- Solution to larger problem contains the solutions to smaller ones

**General Blueprint:**

1. Identify recursive structure of the problem
   - What is the “last thing” done?
2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest
3. Save solution to each subproblem in memory
Consider the Last Precinct

**Observation:** succeed if there is a way to assign \( k \) precincts from \( \{p_1, \ldots, p_{n-1}\} \) to \( D_1 \) with \( x \) voters in \( D_1 \) and \( y \) voters in \( D_2 \) such that either

- \( k + 1 = n/2 \) and \( x + R(p_n) > mn/4 \) and \( y > mn/4 \); or
- \( k = n/2 \) and \( x > mn/4 \) and \( y + R(p_n) > mn/4 \)

After assigning the first \( n - 1 \) precincts \( p_1, \ldots, p_{n-1} \)

- \( k \) precincts \( x \) voters for \( R \)
- \( n - k - 1 \) precincts \( y \) voters for \( R \)

**District** \( D_1 \)

- assign \( p_1 \) to \( D_1 \)

**District** \( D_2 \)

- assign \( p_2 \) to \( D_2 \)

**District** \( D_2 \)

- \( n - k \) precincts \( y + R(p_n) \) voters for \( R \)

Valid gerrymandering if:

- \( n - k = n/2 \), \( x, y + R(p_n) > mn/4 \)
Define Recursive Structure

**Observation:** succeed if there is a way to assign \( k \) precincts from \( \{p_1, \ldots, p_{n-1}\} \) to \( D_1 \) with \( x \) voters in \( D_1 \) and \( y \) voters in \( D_2 \) such that either
- \( k + 1 = n/2 \) and \( x + R(p_n) > mn/4 \) and \( y > mn/4 \); or
- \( k = n/2 \) and \( x > mn/4 \) and \( y + R(p_n) > mn/4 \)

**Recursive substructure:** can we achieve a specific split of the precincts?

\[
S(j, k, x, y) = \text{True } \text{if from among the first } j \text{ precincts:} \]

- \( k \) are assigned to \( D_1 \)
- exactly \( x \) vote for \( R \) in \( D_1 \)
- exactly \( y \) vote for \( R \) in \( D_2 \)

**Goal:** see if there exists \( x, y > mn/4 \) such that \( S(n, n/2, x, y) \) is true
Define Recursive Structure

**Observation:** succeed if there is a way to assign $k$ precincts from $\{p_1, \ldots, p_{n-1}\}$ to $D_1$ with $x$ voters in $D_1$ and $y$ voters in $D_2$ such that either

- $k + 1 = n/2$ and $x + R(p_n) > mn/4$ and $y > mn/4$; or
- $k = n/2$ and $x > mn/4$ and $y + R(p_n) > mn/4$

**Recursive substructure:** can we achieve a specific split of the precincts?

$S(j, k, x, y) = \text{True}$ if from among the first $j$ precincts:

- $k$ are assigned to $D_1$
- exactly $x$ vote for $R$ in $D_1$
- exactly $y$ vote for $R$ in $D_2$

4-dimensional dynamic programming!

Size of the memory?

$n \times n \times mn \times mn$
Identify Recursive Structure

\[ S(j, k, x, y) = \text{True if from among the first } j \text{ precincts:} \]
\[ \begin{align*}
& \text{k are assigned to } D_1 \\
& \text{exactly } x \text{ vote for } R \text{ in } D_1 \\
& \text{exactly } y \text{ vote for } R \text{ in } D_2
\end{align*} \]

Two possibilities: assign \( p_j \) to \( D_1 \) or assign \( p_j \) to \( D_2 \)

Case 1: assign \( p_j \) to \( D_1 \)

\[ S(j, k, x, y) \text{ is true if we can assign } k - 1 \text{ out of the first } j - 1 \text{ precincts to } D_1 \text{ such that:} \]
\[ \begin{align*}
& \text{exactly } x - R(p_j) \text{ vote for } R \text{ in } D_1 \\
& \text{exactly } y \text{ vote for } R \text{ in } D_2
\end{align*} \]

\[ S(j - 1, k - 1, x - R(p_j), y) \]

Case 2: assign \( p_j \) to \( D_2 \)

\[ S(j, k, x, y) \text{ is true if we can assign } k \text{ out of the first } j - 1 \text{ precincts to } D_1 \text{ such that:} \]
\[ \begin{align*}
& \text{exactly } x \text{ vote for } R \text{ in } D_1 \\
& \text{exactly } y - R(p_j) \text{ vote for } R \text{ in } D_2
\end{align*} \]

\[ S(j - 1, k, x, y - R(p_j)) \]
Identify Recursive Structure

\[ S(j, k, x, y) = \text{True if from among the first } j \text{ precincts:} \]
\[ k \text{ are assigned to } D_1 \]
\[ \text{exactly } x \text{ vote for } R \text{ in } D_1 \]
\[ \text{exactly } y \text{ vote for } R \text{ in } D_2 \]

Two possibilities: assign \( p_j \) to \( D_1 \) or assign \( p_j \) to \( D_2 \)

\[ S(j, k, x, y) = S(j - 1, k - 1, x - R(p_j), y) \text{ OR } S\left(j - 1, k, x, y - R(p_j)\right) \]

Base Case: \( S(0,0,0,0) = \text{True} \)
\[ S(0, k, x, y) = \text{False for all } k, x, y \]
Dynamic Programming

Requires **optimal substructure**

- Solution to larger problem contains the solutions to smaller ones

**General Blueprint:**

1. Identify recursive structure of the problem
   - What is the “last thing” done?

2. Select a good order for solving subproblems
   - “Top Down:” Solve each problem recursively
   - “Bottom Up:” Iteratively solve each problem from smallest to largest

3. Save solution to each subproblem in memory
Find a Good Ordering

\[ S(j, k, x, y) = \text{True if from among the first } j \text{ precincts:} \]
\[ \text{k are assigned to } D_1 \]
\[ \text{exactly } x \text{ vote for } R \text{ in } D_1 \]
\[ \text{exactly } y \text{ vote for } R \text{ in } D_2 \]

Two possibilities: assign \( p_j \) to \( D_1 \) or assign \( p_j \) to \( D_2 \)

\[ S(j, k, x, y) = S(j - 1, k - 1, x - R(p_j), y) \text{ OR } S(j - 1, k, x, y - R(p_j)) \]

Base Case: \( S(0,0,0,0) = \text{True} \)
\( S(0, k, x, y) = \text{False for all } k, x, y \)

Observation: Values with \( j \) only depend on values with \( j - 1 \) (start with first component and fill in rest in order)
**Final Algorithm**

\[ S(j, k, x, y) = S(j - 1, k - 1, x - R(p_j), y) \lor S(j - 1, k, x, y - R(p_j)) \]

initialize \( S[0, 0, 0, 0] = \text{True} \) and \( \text{False} \) elsewhere

for \( j = 1, \ldots, n \):
  
  for \( k = 1, \ldots, n \):
    
    for \( x = 0, \ldots, mn \):
      
      for \( y = 0, \ldots, mn \):
        
        \[ S[j, k, x, y] = \]
        
        \[ S[j - 1, k - 1, x - R[j], y] \lor S[j - 1, k, x, y - R[j]] \]

return \( \text{True} \) if exists \( x > mn/4, y > mn/4 \) where

\[ S[n, n/2, x, y] = \text{True} \]
Running Time

\[ S(j, k, x, y) = S(j-1, k-1, x-R(p_j), y) \lor S(j-1, k, x, y-R(p_j)) \]

initialize \( S[0, 0, 0, 0] = \text{True} \) and False elsewhere

for \( j = 1, \ldots, n \):
    for \( k = 1, \ldots, n \):
        for \( x = 0, \ldots, mn \):
            for \( y = 0, \ldots, mn \):
                \[
                S[j, k, x, y] =
                S[j-1, k-1, x-R[j], y] \lor
                S[j-1, k, x, y-R[j]]
                \]

return True if exists \( x > mn/4, y > mn/4 \) where \( S[n, n/2, x, y] = \text{True} \)

Overall Running Time: \( O(m^2n^4) \)
Running Time

Is this an efficient algorithm?

Efficient = “polynomial time”

Inputs to algorithm: \( R(p_1), \ldots, R(p_n), m \)

Length of inputs: \( O(n \log m) \)

Running time is exponential in length of input

In fact: Gerrymandering is NP-complete

Overall Running Time: \( O(m^2 n^4) \)

To be efficient, running time would have to be of the form \( n^s (\log m)^t \) for constants \( s, t \). But

\[
m^2 = (\log m)^{2 \log \log m}.
\]

We call this a “pseudo-polynomial” time algorithm.
Given access to an unlimited number of pennies, nickels, dimes, and quarters, give an algorithm which gives change for a target value $x$ using the fewest number of coins.
Change Making Algorithm

**Given:** target value $x$, list of coins $C = [c_1, \ldots, c_n]$

(in this case $C = [1, 5, 10, 25]$)

Repeatedly select the largest coin less than the remaining target value:

while $x > 0$:

let $c = \max(c_i \in \{c_1, \ldots, c_n\} \mid c_i \leq x)$

add $c$ to list $L$

$x = x - c$

output $L$

Example of a **greedy algorithm:**

always choose the “optimal” choice
Mental Stretch

Suppose we added a new coin worth 11 cents. In conjunction with pennies, nickels, dimes, and quarters, find the minimum number of coins needed to give 90 cents of change.
Greedy Solution

90 cents
Optimal Solution

When can we use the greedy solution?

90 cents
Greedy Algorithms

Requires **optimal substructure**
- Solution to larger problem contains the solution to a smaller one
- Only a **single** subproblem to consider

**General Blueprint:**
1. Identify a greedy **choice property**
   - Show that this choice is **guaranteed** to be included in some optimal solution
2. Repeatedly apply the choice property until no subproblems remain
Dynamic Programming:
  • Require optimal substructure
  • Optimal choice can be one of multiple smaller subproblems

Greedy:
  • Require optimal substructure
  • Only a single choice and a single subproblem
Largest coin less than or equal to target value must be part of some optimal solution (for standard U.S. coins)

To be continued...