CS4102 Algorithms
Spring 2019

Warm up

Given access to unlimited quantities of pennies, nickels dimes, and quarters, (worth value 1, 5, 10, 25 respectively), provide an algorithm which gives change for a given value x using the fewest number of coins.

1

Change Making

43 cents

2

Change Making Algorithm

- Given: target value x, list of coins $C = [c_1, \dots, c_n]$
  (in this case $C = [1,5,10,25]$)
- Repeatedly select the largest coin less than the remaining target value:

while($x > 0$)
  let $c = \max(c_i \in \{c_1, \dots, c_n\} \mid c_i \le x)$
  print c
  $x = x - c$

3

Why does this always work?

- If $x < 5$, then pennies only
  - 5 pennies can be exchanged for a nickel
    Only case Greedy uses pennies!
- If $5 \leq x < 10$ we must have a nickel
  - 2 nickels can be exchanged for a dime
    Only case Greedy uses nickels!
- If $10 \leq x < 25$ we must have at least 1 dime
  - 3 dimes can be exchanged for a quarter and a nickel
    Only case Greedy uses dimes!
- If $x \geq 25$ we must have at least 1 quarter
  Only case Greedy uses quarters!

4

Today's Keywords

- Dynamic Programming
- Gerrymandering
- Greedy Algorithms
- Choice Function
- Change Making

5

CLRS Readings

- Chapter 15
- Chapter 16

6

Homeworks

- Homework 5 due Wednesday March 27 at 11pm
  - Seam Carving!
  - Dynamic Programming (implementation)
  - Java or Python
- Homework 6 out tonight, due Wednesday April 3 at 11pm
  - Dynamic Programming and Greedy Algorithms
  - Written (using Latex!)

7

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  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 recursively
     - "Bottom Up": Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

8

Generic Top-Down Dynamic Programming Soln

```
mem = {}
def myDPalgo(problem):
        if mem[problem] not blank:
                return mem[problem]
        if baseCase(problem):
                solution = solve(problem)
                mem[problem] = solution
                return solution
        for subproblem of problem:
                subsolutions.append(myDPalgo(subproblem))
        solution = OptimalSubstructure(subsolutions)
        mem[problem] = solution
        return solution
```

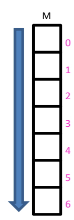
9

DP Algorithms so far

- $2 \times n$ domino tiling (Fibonacci)
- Log cutting
- Matrix Chaining
- Longest Common Subsequence
- Seam Carving

Domino Tiling

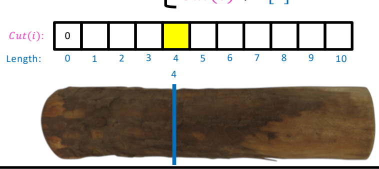
```
Tile(n):
    Initialize Memory M
    M[0] = 0
    M[1] = 0
    for i = 0 to n:
        M[i] = M[i-1] + M[i-2]
    return M[n]
```



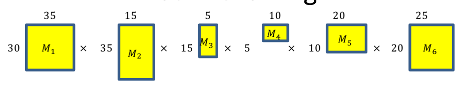
Log Cutting

Solve Smallest subproblem first

$$Cut(4) = \max \begin{cases} Cut(3) + P[1] \\ Cut(2) + P[2] \\ Cut(1) + P[3] \\ Cut(0) + P[4] \end{cases}$$



Matrix Chaining



$$Best(i,j) = \min_{k=1}^{j-1}\left(Best(i,k) + Best(k+1,j) + r_i r_{k+1} c_j\right)$$

$$Best(i,i) = 0$$

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

$$Best(1,6) = \min\begin{cases} Best(1,1) + Best(2,6) + r_1 r_2 c_6 \\ Best(1,2) + Best(3,6) + r_1 r_3 c_6 \\ Best(1,3) + Best(4,6) + r_1 r_4 c_6 \\ Best(1,4) + Best(5,6) + r_1 r_5 c_6 \\ Best(1,5) + Best(6,6) + r_1 r_6 c_6 \end{cases}$$

13

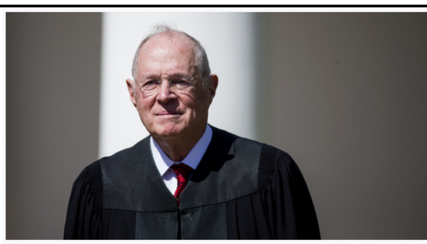
Longest Common Subsequence

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

$Y =$	$X =$	0	A 1	T 2	C 3	T 4	G 5	A 6	T 7
	0	0	0	0	0	0	0	0	0
T	1	0	0	1	1	1	1	1	1
G	2	0	0	1	1	1	2	2	2
C	3	0	0	1	2	2	2	2	2
A	4	0	1	1	2	2	2	3	3
T	5	0	1	2	2	3	3	3	4
A	6	0	1	2	2	3	3	4	4

To fill in cell (i,j) we need cells $(i-1,j-1), (i-1,j), (i,j-1)$
Fill from Top->Bottom, Left->Right (with any preference)

14



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.

16

Gerrymandering

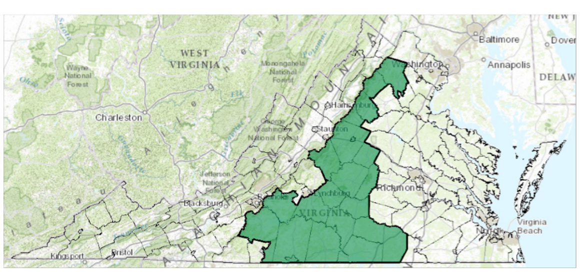
- Manipulating electoral district boundaries to favor one political party over others
- Coined in an 1812 Political cartoon
- Governor Gerry signed a bill that redistricted Massachusetts to benefit his Democratic-Republican Party

The Gerrymander

According to the Supreme Court

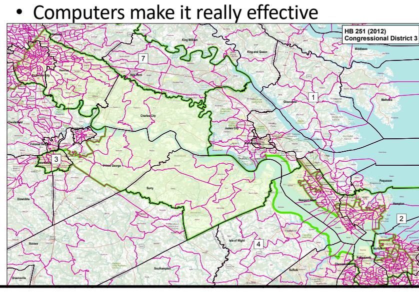
- Gerrymandering cannot be used to:
  - Disadvantage racial/ethnic/religious groups
- It can be used to:
  - Disadvantage political parties

VA 5th District



Gerrymandering Today

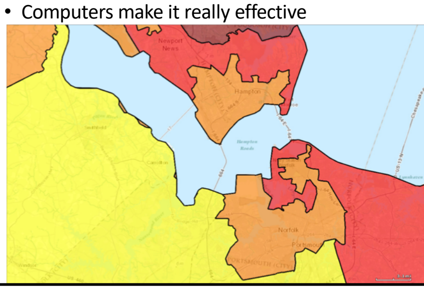
- Computers make it really effective



20

Gerrymandering Today

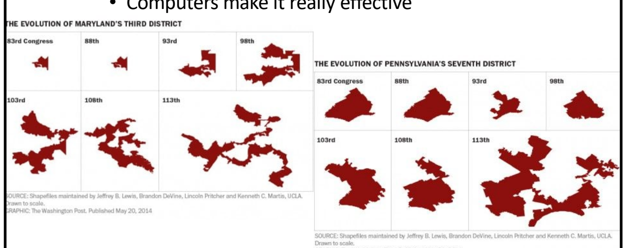
- Computers make it really effective



21

Gerrymandering Today

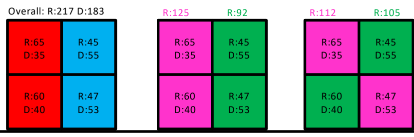
- Computers make it really effective



22

How does it work?

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



23

Gerrymandering Problem Statement

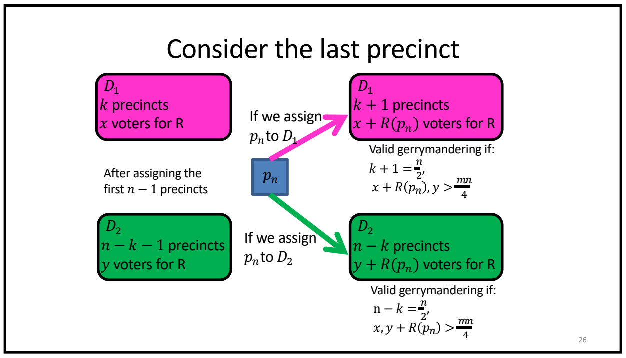
- Given:
  - A list of precincts: $p_1, p_2, \dots, p_n$ — precincts, n of them
  - Each containing m voters
- Output: ← m — not the voters, but how many
  - Districts $D_1, D_2 \subset \{p_1, p_2, \dots, p_n\}$
  - Where $|D_1| = |D_2|$
  - $R(D_1), R(D_2) > \frac{mn}{4}$ $\quad \frac{n}{2} \cdot m \cdot \frac{1}{2}$
    - $R(D_i)$ gives number of "Regular Party" voters in D_i
    - $R(D_i) > \frac{mn}{4}$ means D_i is majority "Regular Party"
  - "failure" if no such solution is possible

24

Dynamic Programming

- Requires Optimal Substructure
  - Solution to larger problem contains the solutions to smaller ones
- Idea:
  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 recursively
     - "Bottom Up": Iteratively solve smallest to largest
  3. Save solution to each subproblem in memory

25



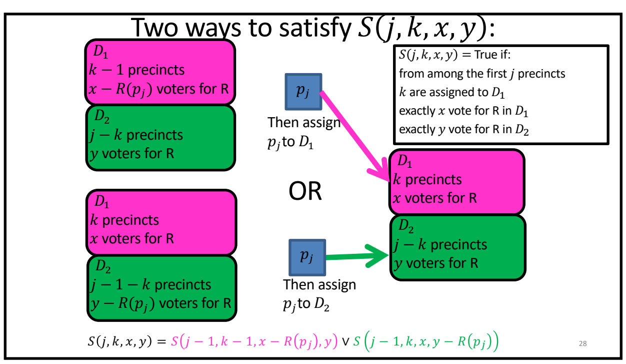
Define Recursive Structure

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

$n \times n \times mn \times mn$

4D Dynamic Programming!!!

27

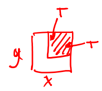


Final Algorithm

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

Initialize $S(0,0,0,0) = \text{True}$
for $j = 1, \ldots, n$:
  for $k = 1, \ldots, \min(j, \frac{n}{2})$:
    for $x = 0, \ldots, jm$:
      for $y = 0, \ldots, jm$:
        $S(j,k,x,y) =$
          $S(j-1,k-1,x-R(p_j),y)$
          $\lor S(j-1,k,x,y-R(p_j))$
Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

$S(j,k,x,y) = \text{True}$ 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



29

Run 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 $S(0,0,0,0) = \text{True}$
n for $j = 1, \ldots, n$:
$\frac{n}{2}$ for $k = 1, \ldots, \min(j, \frac{n}{2})$:
nm for $x = 0, \ldots, jm$:
nm for $y = 0, \ldots, jm$:
        $S(j,k,x,y) =$
          $S(j-1,k-1,x-R(p_j),y)$
          $\lor S(j-1,k,x,y-R(p_j))$
Search for True entry at $S(n, \frac{n}{2}, > \frac{mn}{4}, > \frac{mn}{4})$

$\Theta(n^4 m^2)$

30

$\Theta(n^4 m^2)$

- Runtime depends on the *value* of m, not *size* of m
- Run time is exponential in *size* of input
- Note: Gerrymandering is NP-Complete

31