Dynamic Programming

CS16: Introduction to Data Structures & Algorithms
Spring 2019
Outline

- Dynamic Programming
- Examples
  - Fibonacci
  - Seamcarve
This is Hard!

You will understand it...

The element of CONFUSION
Struggle is a prelude to success.
Fibonacci
What is Dynamic Programming?

- Algorithm design paradigm/framework
  - Design efficient algorithms for optimization problems

- Optimization problems
  - “find the best solution to problem X”
  - “what is the shortest path between u and v in G”
  - “what is the minimum spanning tree in G”

- Can also be used for non-optimization problems
When is Dynamic Programming Applicable?

- **Condition #1**: sub-problems
  - The problem can be solved recursively
  - Can be solved by solving sub-problems

- **Condition #2**: overlapping sub-problems
  - Same sub-problems need to be solved many times

- **Core idea**
  - Solve each sub-problem once and store the solution
  - Use stored solution when you need to solve sub-problem again
Steps to Solving a Problem w/ DP

- What are the **sub-problems**?
- What is the “**magic**” step?
  - Given solution to a sub-problem…
  - …how do I get solution to the problem?
- Which **topological order** on sub-problems can I use?
  - so that solutions to sub-problems available when needed
- Design iterative **algorithm**
  - that solves sub-problems in order and stores their solution
Topological Order?

- Valid topological orderings
  - 15, 16, 22, 141
  - 22, 15, 16, 141
  - 15, 22, 16, 141
Topological Order on Sub-Problems!
Fibonacci
Fibonacci

base cases:

$$F(n) = F(n - 1) + F(n - 2)$$

$$\begin{array}{c}
F(0) = 1 \\
F(1) = 1 \\
\end{array}$$
Fibonacci (Recursive)

- Defined by the recursive relation
  - $F_0 = 0$, $F_1 = 1$
  - $F_n = F_{n-1} + F_{n-2}$
- We can implement this with a recursive function

```python
function fib(n):
    if n = 0:
        return 0
    if n = 1:
        return 1
    return fib(n-1) + fib(n-2)
```

Q: Is any work happening?
Recursion & Clones

- At each intersection
  - clone yourself twice and send one Left and one Right
  - wait for clones to report a path to exit (if it exists) and its length
  - pick direction that gets you to exit the fastest
Fibonacci (Recursive)

Big-O runtime of recursive \texttt{fib} function?
Fibonacci (Recursive)

Big-O runtime of recursive \texttt{fib} function?
Fibonacci (Recursive)

Activity #1

Big-O runtime of recursive `fib` function?
Fibonacci (Recursive)

How many times does \( \text{fib} \) get called for \( \text{fib}(4) \)?

- 8 times

At each level it makes twice as many recursive calls as last

- For \( \text{fib}(n) \) it makes approximately \( 2^n \) recursive calls

Algorithm is \( O(2^n) \)

```python
function fib(n):
    if n = 0:
        return 0
    if n = 1:
        return 1
    return fib(n-1) + fib(n-2)
```
Fibonacci (Recursive)

- How many times does `fib(1)` get computed?
- Instead of recomputing Fibonacci numbers over and over again
- Compute them **once** and store them for later

```python
function fib(n):
    if n = 0:
        return 0
    if n = 1:
        return 1
    return fib(n-1) + fib(n-2)
```
Fibonacci (Dynamic Programming)

- Given $n$ compute
  - $Fib(n) = Fib(n-1) + Fib(n-2)$
  - with base cases $Fib(0) = 0$ and $Fib(1) = 1$
- What are the **sub-problems**?
  - $Fib(n-1), Fib(n-2), \ldots, Fib(1), Fib(0)$
- What is the **magic** step?
  - $Fib(n) = Fib(n-1) + Fib(n-2)$

Magic step is usually not provided!!
Fibonacci (Dynamic Programming)

- Which **topological order** should I use?
  - $\text{Fib}(0), \text{Fib}(1), \ldots, \text{Fib}(n-1), \text{Fib}(n)$
Fibonacci (Dynamic Programming)

- Design iterative **algorithm**

```python
function Fib(n):
    fibs = []
    fibs[0] = 0
    fibs[1] = 1
    for i from 2 to n:
        fibs[i] = fibs[i-1] + fibs[i-2]
    return fibs[n]
```
Fibonacci (Dynamic Programming)

- What’s the runtime of `dynamicFib()`?
  - Calculates Fibonacci numbers from 0 to n
  - Performs $O(1)$ ops for each one
  - Runtime is clearly $O(n)$

- We reduced runtime of algorithm
  - From exponential to linear
  - with dynamic programming!
Seams
Finding Low Importance Seams

- **Idea:** remove **seams** not columns
  - (vertical) seam is a path from top to bottom
  - that moves left or right by at most one pixel per row
Finding Low Importance Seams

‣ How many seams in a \( c \times r \) image?
  ‣ At each row the seam can go Left, Right or Down
  ‣ It chooses 1 out of 3 dirs at all but last row \( r \)
  ‣ So about \( 3^{r-1} \) seams from some starting pixel
  ‣ There are \( c \) starting pixels so total number of seams is
    ‣ about \( c \times 3^{r-1} \)

‣ For square \( n \times n \) image
  ‣ there are about \( n3^{n-1} \) possible seams
Finding Low Importance Seams

- Brute force algorithm
  - Try every possible seam & find least important one
- What is running time of brute force algorithm?
  - If image is $n \times n$ brute force takes about $n^3 \cdot 3^{n-1}$
  - So brute force is $\Omega (2^n)$ (i.e., exponential)
Seamcarve

- What is the runtime of Seamcarve?
- The algorithm
  - Iterate over all pixels from bottom to top
  - Populate `costs` and `dirs` arrays
  - Create seam by choosing minimum value in top row and tracing downward
- How many operations per pixel?
  - A constant number of operations per pixel (4)
- Constant number of operations per pixel means algorithm is linear
  - $O(n)$ where $n$ is number of pixels
- Also could have counted # of nested loops in pseudocode...
Seam carve

- How can we possibly go from
  - exponential running time with brute force
  - to linear running time with Seam carve?
- What is the secret to this magic trick?

Dynamic Programming!
Designing Seamcarve

- What are the subproblems?
  - lowest cost seam (LCS) starting at is
    \[ \text{min} \left( \text{LCS}(\quad), \text{LCS}(\quad), \text{LCS}(\quad) \right) \]

- Are they overlapping?
  - Yes!
  - ex: LCS(\quad) is subproblem of LCS(\quad) and LCS(\quad)
Designing Seamcarve

- What is the magic step?

  \[ \min( \text{LCS}(\text{green}), \text{LCS}(\text{purple}), \text{LCS}(\text{yellow})) \]

- Which topological order should I use?
  - to solve LCS problem at cell \((i,j)\)
  - we need to have solved problem at cells below
Designing Seamcarve

- What is the magic step?
  \[ \min( \text{LCS}(\text{cell}), \text{LCS}(\text{cell}), \text{LCS}(\text{cell})) \]

- Which topological order should I use?
  - to solve LCS problem at cell \((i,j)\)
  - we need to have solved problem at cells below
Designing Seamcarve

- Algorithm
  - compute cost of LCS for each cell going bottom up
  - store cost of LCS in an auxiliary 2D array...
  - ...so we can reuse them

\[
\text{Cost}(\text{cell}) = \text{Val}(\text{cell}) + \min(\text{Cost}(\text{cell}_1), \text{Cost}(\text{cell}_2), \text{Cost}(\text{cell}_3))
\]
Designing Seamcarve

- Problem
  - Costs array only gives us cost of LCS at cell
  - We need the seam. What happened?
  - We used

\[
\text{Cost(\textcolor{red}{\text{red}})} = \text{Val(\textcolor{red}{\text{red}})} + \min(\text{Cost(\textcolor{green}{green})}, \text{Cost(\textcolor{purple}{purple})}, \text{Cost(\textcolor{yellow}{yellow})})
\]

- But recall that at “seam level” we had

\[
\text{LCS(\textcolor{red}{\text{red}})} = \textcolor{red}{\text{red}} \mathbin{\|} \min(\text{LCS(\textcolor{green}{green})}, \text{LCS(\textcolor{purple}{purple})}, \text{LCS(\textcolor{yellow}{yellow})})
\]
Designing Seamcarve

- It’s OK!
  - We can keep track of minimum LCS
  - at each step in auxiliary structure Dirs