CS173
Designing DP algorithms and proving them correct

Tandy Warnow

CS 173

Designing DP algorithms and proving them correct

Tandy Warnow

DP algorithms you've already seen

September 27

- Fibonacci numbers
- Coin changing problem

September 29

- DP algorithm for computing the longest increasing substring
- DP algorithm for finding a longest increasing subsequence

October 2

- DP algorithm for computing All Pairs Shortest Paths in graph

Are these algorithms correct?
Can we prove these algorithms correct?

Dynamic Programming to compute $F(n)$

Let $F(n)$ denote the n^{th} Fibonacci number:
Input: n, positive integer
Output: F(n)

Fill in an array, FIB[1...n] as follows:

- $FIB[1] := 1$
- $FIB[2] := 1$
- For $i := 3$ up to n do:
  - $FIB[i] := FIB[i-1] + FIB[i-2]$
- Return $FIB[n]$

Recall we analyzed the running time and showed it was $O(n)$ to compute $FIB[n]$.

Let's prove that $FIB[n]$ is the same as $F(n)$, the n^{th} Fibonacci number.

Proving the DP algorithm correct

Let $F(n)$ be the n^{th} Fibonacci number, defined recursively by

- $F(1) = F(2) = 1$ and
- $F(n) = F(n-1) + F(n-2)$ for $n \geq 3$

We prove that $F(n) = FIB[n]$ by strong induction on n.

Let $P(N)$ denote the statement $\forall n \in \mathbb{Z}^+, n \leq N, FIB[n] = F(n)$.

By definition, $FIB[n] = F(n)$ for $n = 1, 2$, so the two base cases are true.

We have shown $P(1)$ and $P(2)$ is true (our base cases).

Our Strong Inductive Hypothesis is that $P(N)$ is true for some arbitrary $N \geq 2$.

We wish to prove that $P(N+1)$ is true.

In other words, we wish to prove that $FIB[N+1] = F(N+1)$.

Proving the DP algorithm for Fibonacci is correct

To prove that $FIB[N+1] = F(N+1)$, note that $N \geq 2$ so $N+1 \geq 3$.

Hence $FIB[N+1] = FIB[N] + FIB[N-1]$, by the DP algorithm.

By the inductive hypothesis $FIB[N] = F(N)$ and $FIB[N-1] = F(N-1)$, and so $FIB[N+1] = F(N) + F(N-1)$.

Hence, $FIB[N+1] = F(N+1)$, by the definition of the Fibonacci numbers.

Since N was arbitrary, by the Principle of Mathematical Induction, $FIB[N] = F(N)$ for all non-negative integers N.

Other applications of Dynamic Programming

We have already shown DP algorithms for some other problems, such as:

- Coin changing problem
- Computing the longest increasing substring in a sequence
- Finding the longest increasing subsequence in a sequence
- Finding all-pairs shortest paths in an edge-weighted graph

You can also find DP algorithms online for:

- Longest common subsequence of two sequences
- Minimum edit distance between two strings (where insertions, deletions, and substitutions each cost 1)
- Biology problem: optimal pairwise alignment between two DNA sequences (corresponds to minimum edit distance)
- Biology problem: maximum parsimony on a tree

Let's do DP for a two-person game.

DP algorithm for a two-person game

Suppose we have a two-person game, as follows.

- There are two piles of rocks.
- Each player picks a pile and then takes 1 or 2 two rocks off that pile.
- The person who takes the last rock off wins.

Use DP to determine which player has a winning strategy when the starting condition has x rocks on pile 1 and y rocks on pile 2.

DP algorithm for two-person game

Consider the starting condition (x, y) to mean that pile 1 has x rocks and pile 2 has y rocks.

Define a matrix $M[0...x, 0...y]$ by

- $M[0, 0] = 2$
- If $i + j > 0$ then $M[i, j]$ is 1 if and only if Player 1 has a winning strategy for starting condition (i, j).

Questions to class:

- What is $M[1, 0]$?
- What is $M[2, 0]$?
- What is $M[3, 0]$?
- What is $M[1, 1]$?

How should $M[i, j]$ be defined, algorithmically?

DP algorithm for two-person game

Key observation: Player 1 has a winning strategy if and only if she can move to a condition where player 2 has a winning strategy (because she becomes player 2 after she moves).

Remember that each player picks a pile and then takes 1 or 2 rocks off the pile.

Hence, we should set $M[i,j]$ to 1 if and only if at least one of the following is set to 2:

- $M[i-1,j]$
- $M[i-2,j]$
- $M[i,j-1]$
- $M[i,j-2]$

Of course, you need to make sure to check if these value are out of bound or not.

Finishing the DP algorithm

Given starting condition x, y with $0 \leq x, y$ and $x + y > 0$, we fill out the matrix $M[., .]$ as follows:

- We set $M[0, 0]$ to 2
- We set $M[1, 0], M[2, 0], M[0, 1]$, and $M[0, 2]$ all to 1 (these are the cases where Player 1 wins immediately).
- For all other pairs i, j with $i \leq x$ and $j \leq y$, we set $M[i, j]$ to 1 if and only if at least one of the following is set to 2:
    - $M[i - 1, j]$
    - $M[i - 2, j]$
    - $M[i, j - 1]$
    - $M[i, j - 2]$

  Otherwise, we set $M[i, j] = 2$.

Class exercise: Fill out the matrix for $x = 4, y = 3$.

Languages

A **language** is a set of strings over an alphabet Σ.

- The set of all finite-length strings over an alphabet Σ is denoted Σ^*.
- The set of all non-empty finite-length strings over Σ is denoted Σ^+
- The length of a string is the number of characters it has
- The empty string has zero length
- If x and y are strings, we write yx to denote the concatenation of the two strings. For example, if $x = 00$ and $y = 101$ then $xy = 00101$.

A recursively defined language, L

Let L be a set of strings over $\{0,1\}$ defined recursively by:

- $1 \in L$
- If $x \in L$ then $x10 \in L$
- If $x \in L$ then $x0 \in L$

Thus, L contains only those strings that can be derived using these rules.
Notes:

- L doesn't contain any infinite length strings!
- All strings in L of length two or more start with 1 and end with 0.

Question to class: does L contain every string that begins with 1 and ends with 0?

The set L of strings

Let L be a set of strings over $\{0, 1\}$ defined recursively by:

- $1 \in L$
- If $x \in L$ then $x10 \in L$
- If $x \in L$ then $x0 \in L$

Questions to class:

1. Is $0 \in L$?
2. Is $11 \in L$?
3. Is $10110 \in L$?
4. Find all strings of length up to 3 that are in L.
5. Give one string in L of length 10.

DP algorithm to determine if $x \in L$

Let's design a DP algorithm to determine if $x \in L$ where x is a binary string.

Let $x \in \{0, 1\}^+$ be given as input (so x is not the empty string).

We define the length of x to be the number of characters in x. For example, if $x = 011001$ then the length of x is 6.

We write $x[i]$ to denote the i^{th} letter of x and $x[1...i]$ to denote the prefix of x ending at $x[i]$.

For example, if $x = 011001$ then $x[4] = 0$ and $x[1..4] = 0110$.

DP algorithm to determine if $x \in L$, continued

If the length of x is at most 2, we return *True* if and only if $x \in \{1, 10\}$.
For all other strings x, we will compute an array $M[1...n]$ where n is the length of x, and where

$$M[i] = \textit{True} \text{ if and only if } x[1...i] \in L.$$

We will then return $M[n]$!

Basic challenge: how shall we calculate the array M?

DP algorithm to determine if $x \in L$

Computing the array $M[1...n]$ where $n > 2$ is the length of x:

- $M[1] := [x[1] = 1]$
- $M[2] := [(x[1] = 1) \wedge (x[2] = 0)]$
- For $i := 3$ up to n, we set $M[i] = \mathit{True}$ if and only if at least one of the following is *True*:
  - $M[i-1] \wedge (x[i] = 0)$
  - $M[i-2] \wedge (x[i] = 0) \wedge (x[i-1] = 1)$

What are the entries of M when $x = 110$? What about $x = 100$?

The DP algorithm

Input: $x \in \{0, 1\}^+$
Output: *True* or *False* (i.e., whether $x \in L$)

Algorithm:

- If $length(x) \leq 2$, Return $(x \in \{1, 10\})$
- Else compute $M[1...n]$, where $n = length(x)$, and Return (M[n])

Questions:

- Is this algorithm correct? Could you prove it correct?
- What is the running time?

Class exercise: Compute $M[1...6]$ for $x = 111000$ and $y = 1000100$