Today:
Dynamic Programming
Longest Common Subsequence

COSC 581, Algorithms

January 28, 2014

Reading Assignments

- Today's class:
  - Chapter 15.3-4
- Reading assignment for next class:
  - Chapter 15.5

Dynamic Programming: Recall

- Motivation of dynamic programming
  - Solving each subproblem only once
- Steps of dynamic programming
  - Characterize the structure of an optimal solution
  - Recursively define the value of an optimal solution
  - Compute the value of an optimal solution
  - Construct an optimal solution from computed information
- Dynamic programming implementation
  - Top-down method with memoization
  - Bottom-up approach

Dynamic Programming (Cont'd)

- When to use dynamic programming? **Two Elements:**
  - Problem exhibits **optimal structure**
    - Optimal solutions to a problem incorporate optimal solutions to subproblems
    - Subproblems can be solved independently
  - Problem has **overlapping subproblems**

DP Element 1: Optimal Structure

- Discover optimal structure
  - Show that a solution to a problem consists of making a choice
    - Rod-cutting: choosing an initial cut a rod
    - Matrix chain multiplication: choosing an index to split a matrix chain
  - Suppose the choice leads to an optimal solution
  - Given the choice, determine which subproblems to use
    - Rod-cutting: a subproblem is also a rod-cutting problem with a smaller size
    - Matrix-chain multiplication: a subproblem is also a matrix-chain multiplication problem with new start and end indices
  - Show that the solutions to the subproblems must themselves be optimal by using a “cut-and-paste” technique

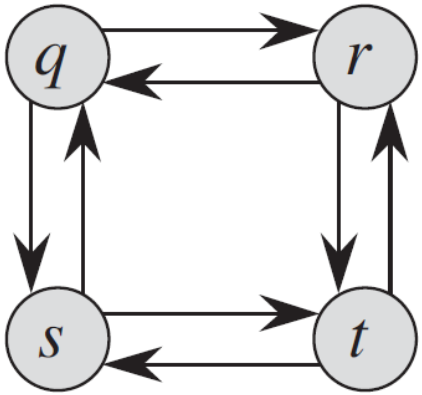
DP Element 1: Optimal Structure

- Optimal structure varies across problem domains
  - Number of subproblems
    - Rod-cutting: one
    - Matrix chain multiplication: two
  - Number of choices to determine which sub problem to use
    - Rod-cutting: for cutting up a rod of size *n*, we must consider n choices
    - Matrix-chain multiplication: for the subchain $A_i A_{i+1} \cdots A_j$, we must consider $(j - i)$ choices; that is to choose k to split the chain $A_i A_{i+1} \cdots A_k$ and $A_{k+1} A_{k+2} \cdots A_j$.
- Runtime of dynamic programming generally depends on:

  overall number of subproblems x number of choices
  - Can be visualized using the subproblem graph

DP Element 1: Optimal Structure

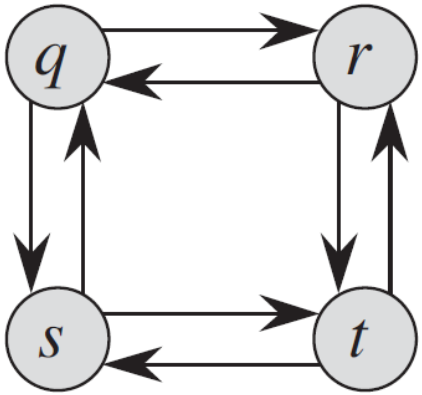
- Be careful with optimal structure: **Not** always apply!
  - Given a directed graph: $G = (V, E)$
  - Unweighted shortest path
  - Unweighted longest path: consider $q \rightarrow r \rightarrow t$



- What makes optimal structure no longer apply?

DP Element 1: Optimal Structure

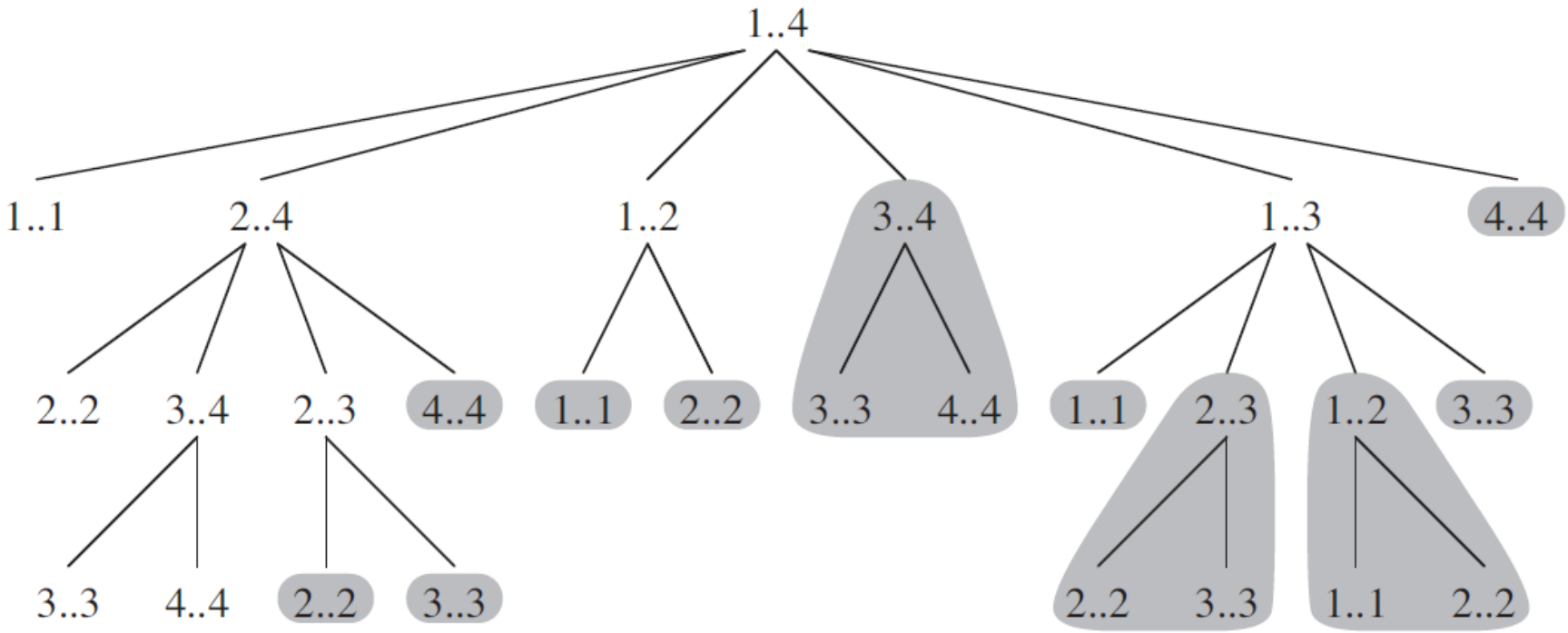
- Be careful with optimal structure: **Not** always apply!
  - Given a directed graph: $G = (V, E)$
  - Unweighted shortest path
  - Unweighted longest path: consider $q \rightarrow r \rightarrow t$



- What makes optimal structure no longer apply?

Subproblems are not independent

DP Element 2: Overlapping Subproblems



Example: Matrix-chain multiplication (4 matrix in the chain)

Dynamic Programming—Optimal Strategy for a Game

Consider a row of n coins of values $v1 \ldots vn$ where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin.

Determine the maximum possible amount of money we can definitely win (for the entire game) if we move first. We have no control over the opponent's strategy, so we must presume that the opponent always chooses coins that are good for the opponent.

Let $W(i,j)$ be the maximum value we can definitely win if it is our turn and only coins $i..j$, with values $vi \ldots vj$, remain. We can calculate two sets of base cases: $W(i, i)$ and $W(i, i+1)$, for all i. Note that these base cases correspond to all possible series of coins of length 1 and 2, respectively.

1. Write a recursive expression for the value of W(i,j), including the base cases. [Remember that the value of W(i,j) is always from our perspective, never from the opponent's perspective. So, the opponent's possible choices have to be folded into the recursive expression.]

Dynamic Programming—Optimal Strategy for a Game

Consider a row of n coins of values $v1 \ldots vn$ where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin.

Determine the maximum possible amount of money we can definitely win (for the entire game) if we move first. We have no control over the opponent's strategy, so we must presume that the opponent always chooses coins that are good for the opponent.

Let $W(i,j)$ be the maximum value we can definitely win if it is our turn and only coins $i..j$, with values $vi \ldots vj$, remain. We can calculate two sets of base cases: $W(i, i)$ and $W(i, i+1)$, for all i. Note that these base cases correspond to all possible series of coins of length 1 and 2, respectively.

2. If you were to implement this as a dynamic programming problem, in what order would the W(i,j) table be filled in?

Dynamic Programming—Optimal Strategy for a Game

Consider a row of n coins of values $v1$ ... vn where n is even. We play a game against an opponent by alternating turns. In each turn, a player selects either the first or last coin from the row, removes it from the row permanently, and receives the value of the coin.

Determine the maximum possible amount of money we can definitely win (for the entire game) if we move first. We have no control over the opponent's strategy, so we must presume that the opponent always chooses coins that are good for the opponent.

Let $W(i,j)$ be the maximum value we can definitely win if it is our turn and only coins $i..j$, with values $vi \ldots vj$, remain. We can calculate two sets of base cases: $W(i, i)$ and $W(i, i+1)$, for all i. Note that these base cases correspond to all possible series of coins of length 1 and 2, respectively.

3. How many distinct subproblems are there in this problem?

4. What would be the runtime of a dynamic programming implementation of this recursive solution?

Longest Common Subsequence

Slides adapted from version found online

Subsequences

Suppose you have a sequence

$X = < x_1, x_2, \ldots, x_m >$

of elements over a finite set S.

A sequence $Z = < z_1, z_2, \ldots, z_k >$ over S is called a subsequence of X if and only if it can be obtained from X by deleting elements.

Put differently, there exist indices $i_1 < i_2 < \ldots < i_k$ such that

$$z_a = x_{i_a}$$

for all a in the range $1 <= a <= k$.

Common Subsequences

Suppose that X and Y are two sequences over a set S.

We say that Z is a common subsequence of X and Y if and only if

- Z is a subsequence of X
- Z is a subsequence of Y

The Longest Common Subsequence Problem

Given two sequences X and Y over a set S, the longest common subsequence problem asks to find a common subsequence of X and Y that is of maximal length.

Naïve Solution

Let X be a sequence of length m,

and Y a sequence of length n.

Check for every subsequence of X whether it is a subsequence of Y, and return the longest common subsequence found.

There are 2^m subsequences of X. Testing a sequences whether or not it is a subsequence of Y takes O(n) time. Thus, the naïve algorithm would take $O(n2^m)$ time.

Dynamic Programming

Let us try to develop a dynamic programming solution to the LCS problem.

Prefix

Let $X = < x_1, x_2, \ldots, x_m >$ be a sequence.

We denote by X_i the sequence

$X_i = < x_1, x_2, \ldots, x_i >$

and call it the i^{th} prefix of X.

LCS Notation

Let X and Y be sequences.

We denote by LCS(X, Y) the set of longest common subsequences of X and Y.

Optimal Substructure

Let $X = < x_1, x_2, \ldots, x_m >$

and $Y = < y_1, y_2, \ldots, y_n >$ be two sequences.

Let $Z = < z_1, z_2, \ldots, z_k >$ is any LCS of X and Y.

a) If $x_m = y_n$ then certainly $x_m = y_n = z_k$

and Z_{k-1} is in LCS(X_{m-1} , Y_{n-1})

Optimal Substructure (2)

Let $X = < x_1, x_2, \ldots, x_m >$

and $Y = < y_1, y_2, \ldots, y_n >$ be two sequences.

Let $Z = < z_1, z_2, \ldots, z_k >$ is any LCS of X and Y.

b) If x_m != y_n then x_m != z_k implies that Z is in $LCS(X_{m-1}, Y)$

c) If x_m != y_n then y_n != z_k implies that Z is in $LCS(X, Y_{n-1})$

Overlapping Subproblems

If $x_m = y_n$ then we solve the subproblem to find an element in LCS(X_{m-1} , Y_{n-1}) and append x_m

If x_m != y_n then we solve the two subproblems of finding elements in LCS(X_m , Y_{n-1}) and LCS(X_{m-1} , Y_n) and choose the longer one.

Recursive Solution

Let X and Y be sequences.

Let c[i,j] be the length of an element in LCS(X_i, Y_j).

$$c[i,j] = \begin{cases} 0 & \text{if } i=0 \text{ or } j=0 \\ c[i-1,j-1]+1 & \text{if } i,j>0 \text{ and } x_i = y_j \\ \max(c[i,j-1],c[i-1,j]) & \text{if } i,j>0 \text{ and } x_i \neq y_j \end{cases}$$

Dynamic Programming Solution

To compute length of an element in LCS(X,Y) with X of length m and Y of length n, we do the following:

- Initialize first row and first column of the array c with 0.
- Calculate c[1,j] for 1 <= j <= n,
  c[2,j] for 1 <= j <= n …
- Return c[m,n]
- Complexity O(mn).

Dynamic Programming Solution (2)

How can we get an actual longest common subsequence?

Store in addition to the array c an array b pointing to the optimal subproblem chosen when computing c[i,j].

Example

	y_j	B	D	C	A
x_j	0	0	0	0	0
A	0	0	0	0	1
B	0	1	1	1	1
C	0	1	1	2	2
B	0	1	1	2	2

Start at b[m,n]. Follow the arrows. Each diagonal array gives one element of the LCS.

$$c[i,j] = \begin{cases} 0 & \text{if } i = 0 \text{ or } j = 0, \\ c[i-1, j-1] + 1 & \text{if } i, j > 0 \text{ and } x_i = y_j \\ \max(c[i, j-1], c[i-1, j]) & \text{if } i, j > 0 \text{ and } x_i \neq y_j \end{cases}$$

Reading Assignments

- Today's class:
  - Chapter 15.3-4
- Reading assignment for next class:
  - Chapter 15.5