

Computer Science & Engineering 423/823

Design and Analysis of Algorithms

Lecture 09 — Dynamic Programming (Chapter 15)

Stephen Scott

(Adapted from Vinodchandran N. Variyam)

Spring 2010

- Dynamic programming is a technique for solving optimization problems
 - Key element: Decompose a problem into **subproblems**, solve them recursively, and then combine the solutions into a final (optimal) solution
 - Important component: There are typically an exponential number of subproblems to solve, but many of them overlap
- ⇒ Can re-use the solutions rather than re-solving them
- Number of distinct subproblems is polynomial

- A company has a rod of length n and wants to cut it into smaller rods to maximize profit
- Have a table telling how much they get for rods of various lengths: A rod of length i has price p_i
- The cuts themselves are free, so profit is based solely on the prices charged for of the rods
- If cuts only occur at integral boundaries $1, 2, \dots, n - 1$, then can make or not make a cut at each of $n - 1$ positions, so total number of possible solutions is 2^{n-1}

Example: Rod Cutting (2)

CSCE423/823

Introduction

Rod Cutting

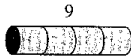
Recursive
Algorithm
Dynamic
Programming
Algorithm
Reconstructing a
Solution

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

i	1	2	3	4	5	6	7	8	9	10
p_i	1	5	8	9	10	17	17	20	24	30



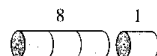
(a)



(b)



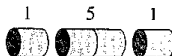
(c)



(d)



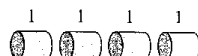
(e)



(f)



(g)



(h)

Example: Rod Cutting (3)

CSCE423/823

Introduction

Rod Cutting

Recursive
Algorithm
Dynamic
Programming
Algorithm
Reconstructing a
SolutionMatrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
Trees

- Given a rod of length n , want to find a set of cuts into lengths i_1, \dots, i_k (where $i_1 + \dots + i_k = n$) and $r_n = p_{i_1} + \dots + p_{i_k}$ is maximized
- For a specific value of n , can either make no cuts (revenue = p_n) or make a cut at some position i , then optimally solve the problem for lengths i and $n - i$:

$$r_n = \max(p_n, r_1 + r_{n-1}, r_2 + r_{n-2}, \dots, r_i + r_{n-i}, \dots, r_{n-1} + r_1)$$

- Notice that this problem has the **optimal substructure property**, in that an optimal solution is made up of optimal solutions to subproblems
 - Can find optimal solution if we consider all possible subproblems
- Alternative formulation: Don't further cut the first segment:

$$r_n = \max_{1 \leq i \leq n} (p_i + r_{n-i})$$

```

1  if  $n == 0$  then
2      return 0
3   $q = -\infty$ 
4  for  $i = 1$  to  $n$  do
5       $q = \max(q, p[i] + \text{CUT-ROD}(p, n - i))$ 
6  end
7  return  $q$ 
    
```

Algorithm 1: Cut-Rod(p, n)

What is the time complexity?

Time Complexity

CSCE423/823

Introduction

Rod Cutting

Recursive
AlgorithmDynamic
Programming
Algorithm
Reconstructing a
SolutionMatrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
Trees

- Let $T(n)$ be number of calls to CUT-ROD
- Thus $T(0) = 1$ and, based on the **for** loop,

$$T(n) = 1 + \sum_{j=0}^{n-1} T(j) = 2^n$$

- Why exponential? CUT-ROD exploits the optimal substructure property, but repeats work on these subproblems
- E.g. if the first call is for $n = 4$, then there will be:
 - 1 call to CUT-ROD(4)
 - 1 call to CUT-ROD(3)
 - 2 calls to CUT-ROD(2)
 - 4 calls to CUT-ROD(1)
 - 8 calls to CUT-ROD(0)

Time Complexity (2)

CSCE423/823

Introduction

Rod Cutting

Recursive
Algorithm

Dynamic
Programming
Algorithm

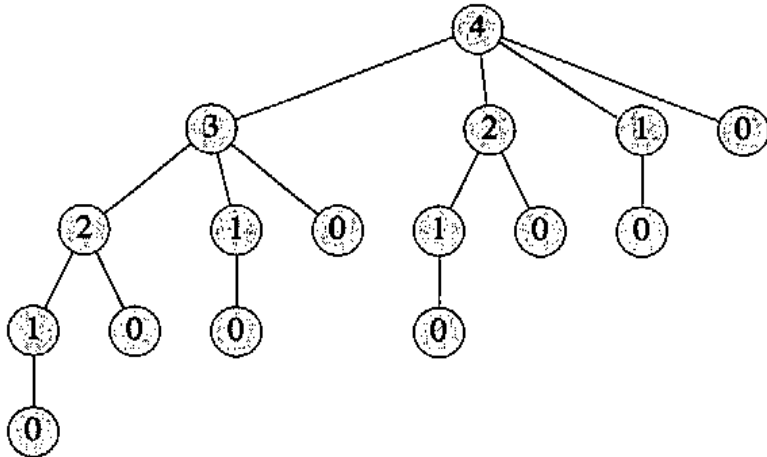
Reconstructing a
Solution

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

Recursion Tree for $n = 4$



- Can save time dramatically by remembering results from prior calls
- Two general approaches:
 - ① **Top-down with memoization:** Run the recursive algorithm as defined earlier, but before recursive call, check to see if the calculation has already been done and **memoized**
 - ② **Bottom-up:** Fill in results for “small” subproblems first, then use these to fill in table for “larger” ones
- Typically have the same asymptotic running time

Top-Down with Memoization

CSCE423/823

Introduction

Rod Cutting

Recursive
Algorithm

Dynamic
Programming
Algorithm

Reconstructing a
Solution

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

```

1  if  $r[n] \geq 0$  then
2      return  $r[n]$                 //  $r$  initialized to all  $-\infty$ 
3  if  $n == 0$  then
4       $q = 0$ 
5  else
6       $q = -\infty$ 
7      for  $i = 1$  to  $n$  do
8           $q =$ 
               $\max(q, p[i] + \text{MEMOIZED-CUT-ROD-AUX}(p, n - i, r))$ 
9      end
10      $r[n] = q$ 
11 return  $q$ 
    
```

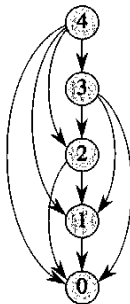
Algorithm 2: Memoized-Cut-Rod-Aux(p, n, r)

```
1 Allocate  $r[0 \dots n]$ 
2  $r[0] = 0$ 
3 for  $j = 1$  to  $n$  do
4      $q = -\infty$ 
5     for  $i = 1$  to  $j$  do
6          $q = \max(q, p[i] + r[j - i])$ 
7     end
8      $r[j] = q$ 
9 end
10 return  $r[n]$ 
```

Algorithm 3: Bottom-Up-Cut-Rod(p, n)

First solves for $n = 0$, then for $n = 1$ in terms of $r[0]$, then for $n = 2$ in terms of $r[0]$ and $r[1]$, etc.

Subproblem graph for $n = 4$



Both algorithms take linear time to solve for each value of n , so total time complexity is $\Theta(n^2)$

Reconstructing a Solution

CSCE423/823

Introduction

Rod Cutting

Recursive
AlgorithmDynamic
Programming
AlgorithmReconstructing a
SolutionMatrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
Trees

- If interested in the set of cuts for an optimal solution as well as the revenue it generates, just keep track of the choice made to optimize each subproblem
- Will add a second array s , which keeps track of the optimal size of the first piece cut in each subproblem

Reconstructing a Solution (2)

CSCE423/823

Introduction

Rod Cutting

Recursive
Algorithm

Dynamic
Programming
Algorithm

Reconstructing a
Solution

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

```

1  Allocate  $r[0 \dots n]$  and  $s[0 \dots n]$ 
2   $r[0] = 0$ 
3  for  $j = 1$  to  $n$  do
4       $q = -\infty$ 
5      for  $i = 1$  to  $j$  do
6          if  $q < p[i] + r[j - i]$  then
7               $q = p[i] + r[j - i]$ 
8               $s[j] = i$ 
9      end
10      $r[j] = q$ 
11 end
12 return  $r, s$ 
    
```

Algorithm 4: Extended-Bottom-Up-Cut-
Rod(p, n)

Reconstructing a Solution (3)

CSCE423/823

Introduction

Rod Cutting

Recursive
Algorithm

Dynamic
Programming
Algorithm

Reconstructing a
Solution

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

```

1  (r, s) = EXTENDED-BOTTOM-UP-CUT-ROD(p, n)
2  while n > 0 do
3      print s[n]
4      n = n - s[n]
5  end
    
```

Algorithm 5: Print-Cut-Rod-Solution(p, n)

Example:

i	0	1	2	3	4	5	6	7	8	9	10
$r[i]$	0	1	5	8	10	13	17	18	22	25	30
$s[i]$	0	1	2	3	2	2	6	1	2	3	10

If $n = 10$, optimal solution is no cut; if $n = 7$, then cut once to get segments of sizes 1 and 6

Matrix-Chain Multiplication

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Characterizing
Structure
Recursive
Definition
Computing
Optimal Value
Constructing
Optimal Solution
Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees

- Given a chain of matrices $\langle A_1, \dots, A_n \rangle$, goal is to compute their product $A_1 \cdots A_n$
- This operation is associative, so can sequence the multiplications in multiple ways and get the same result
- Can cause dramatic changes in number of operations required
- Multiplying a $p \times q$ matrix by a $q \times r$ matrix requires pqr steps and yields a $p \times r$ matrix for future multiplications
- E.g. Let A_1 be 10×100 , A_2 be 100×5 , and A_3 be 5×50
 - ① Computing $((A_1 A_2) A_3)$ requires $10 \cdot 100 \cdot 5 = 5000$ steps to compute $(A_1 A_2)$ (yielding a 10×5), and then $10 \cdot 5 \cdot 50 = 2500$ steps to finish, for a total of 7500
 - ② Computing $(A_1 (A_2 A_3))$ requires $100 \cdot 5 \cdot 50 = 25000$ steps to compute $(A_2 A_3)$ (yielding a 100×50), and then $10 \cdot 100 \cdot 50 = 50000$ steps to finish, for a total of 75000

Matrix-Chain Multiplication (2)

CSCE423/823

Introduction

Rod Cutting

**Matrix-Chain
Multiplication**

Characterizing
Structure
Recursive
Definition
Computing
Optimal Value
Constructing
Optimal Solution
Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees

- The **matrix-chain multiplication problem** is to take a chain $\langle A_1, \dots, A_n \rangle$ of n matrices, where matrix i has dimension $p_{i-1} \times p_i$, and fully parenthesize the product $A_1 \cdots A_n$ so that the number of scalar multiplications is minimized
- Brute force solution is infeasible, since its time complexity is $\Omega(4^n/n^{3/2})$
- Will follow 4-step procedure for dynamic programming:
 - ① Characterize the structure of an optimal solution
 - ② Recursively define the value of an optimal solution
 - ③ Compute the value of an optimal solution
 - ④ Construct an optimal solution from computed information

Characterizing the Structure of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationCharacterizing
StructureRecursive
DefinitionComputing
Optimal Value
Constructing
Optimal Solution
Overlapping
SubproblemsLongest
Common
SubsequenceOptimal
Binary Search
Trees

- Let $A_{i\dots j}$ be the matrix from the product $A_i A_{i+1} \cdots A_j$
- To compute $A_{i\dots j}$, must split the product and compute $A_{i\dots k}$ and $A_{k+1\dots j}$ for some integer k , then multiply the two together
- Cost is the cost of computing each subproduct plus cost of multiplying the two results
- Say that in an optimal parenthesization, the optimal split for $A_i A_{i+1} \cdots A_j$ is at k
- Then in an optimal solution for $A_i A_{i+1} \cdots A_j$, the parenthesization of $A_i \cdots A_k$ is itself optimal for the subchain $A_i \cdots A_k$ (if not, then we could do better for the larger chain)
- Similar argument for $A_{k+1} \cdots A_j$
- Thus if we make the right choice for k and then optimally solve the subproblems recursively, we'll end up with an optimal solution
- Since we don't know optimal k , we'll try them all

Recursively Defining the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Characterizing
Structure

Recursive
Definition

Computing
Optimal Value
Constructing
Optimal Solution
Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees

- Define $m[i, j]$ as minimum number of scalar multiplications needed to compute $A_{i...j}$
- (What entry in the m table will be our final answer?)
- Computing $m[i, j]$:
 - 1 If $i = j$, then no operations needed and $m[i, i] = 0$ for all i
 - 2 If $i < j$ and we split at k , then optimal number of operations needed is the optimal number for computing $A_{i...k}$ and $A_{k+1...j}$, plus the number to multiply them:

$$m[i, j] = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$$

- 3 Since we don't know k , we'll try all possible values:

$$m[i, j] = \begin{cases} 0 & \text{if } i = j \\ \min_{i \leq k < j} \{m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j\} & \text{if } i < j \end{cases}$$

- To track the optimal solution itself, define $s[i, j]$ to be the value of k used at each split

Computing the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationCharacterizing
Structure
Recursive
DefinitionComputing
Optimal ValueConstructing
Optimal Solution
Overlapping
SubproblemsLongest
Common
SubsequenceOptimal
Binary Search
Trees

- As with the rod cutting problem, many of the subproblems we've defined will overlap
- Exploiting overlap allows us to solve only $\Theta(n^2)$ problems (one problem for each (i, j) pair), as opposed to exponential
- We'll do a bottom-up implementation, based on chain length
- Chains of length 1 are trivially solved ($m[i, i] = 0$ for all i)
- Then solve chains of length 2, 3, etc., up to length n
- Linear time to solve each problem, quadratic number of problems, yields $O(n^3)$ total time

Computing the Value of an Optimal Solution (2)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Characterizing
Structure
Recursive
Definition

Computing
Optimal Value

Constructing
Optimal Solution
Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees

```

1  allocate  $m[1 \dots n, 1 \dots n]$  and  $s[1 \dots n, 1 \dots n]$ 
2  initialize  $m[i, i] = 0 \forall 1 \leq i \leq n$ 
3  for  $\ell = 2$  to  $n$  do
4      for  $i = 1$  to  $n - \ell + 1$  do
5           $j = i + \ell - 1$ 
6           $m[i, j] = \infty$ 
7          for  $k = i$  to  $j - 1$  do
8               $q = m[i, k] + m[k + 1, j] + p_{i-1}p_kp_j$ 
9              if  $q < m[i, j]$  then
10                  $m[i, j] = q$ 
11                  $s[i, j] = k$ 
12             end
13         end
14     end
15 return  $(m, s)$ 

```

Algorithm 6: Matrix-Chain-Order(p, n)

Computing the Value of an Optimal Solution (3)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

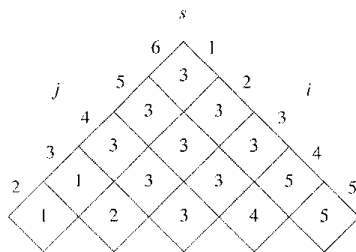
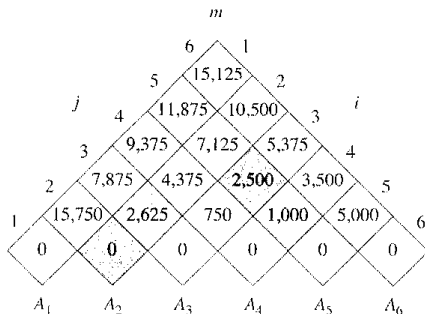
Characterizing
Structure
Recursive
Definition

Computing
Optimal Value

Constructing
Optimal Solution
Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees



matrix	A_1	A_2	A_3	A_4	A_5	A_6
dimension	30×35	35×15	15×5	5×10	10×20	20×25

Constructing an Optimal Solution from Computed Information

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationCharacterizing
StructureRecursive
DefinitionComputing
Optimal ValueConstructing
Optimal SolutionOverlapping
SubproblemsLongest
Common
SubsequenceOptimal
Binary Search
Trees

- Cost of optimal parenthesization is stored in $m[1, n]$
- First split in optimal parenthesization is between $s[1, n]$ and $s[1, n] + 1$
- Descending recursively, next splits are between $s[1, s[1, n]]$ and $s[1, s[1, n]] + 1$ for left side and between $s[s[1, n] + 1, n]$ and $s[s[1, n] + 1, n] + 1$ for right side
- and so on...

Constructing an Optimal Solution from Computed Information (2)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationCharacterizing
StructureRecursive
DefinitionComputing
Optimal ValueConstructing
Optimal SolutionOverlapping
SubproblemsLongest
Common
SubsequenceOptimal
Binary Search
Trees

```
1  if  $i == j$  then
2      print " $A$ " $i$ 
3  else
4      print "("
5      PRINT-OPTIMAL-PARENS( $s, i, s[i, j]$ )
6      PRINT-OPTIMAL-PARENS( $s, s[i, j] + 1, j$ )
7      print ")"
```

Algorithm 7: Print-Optimal-Parens(s, i, j)

Example of How Subproblems Overlap

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Characterizing
Structure

Recursive
Definition

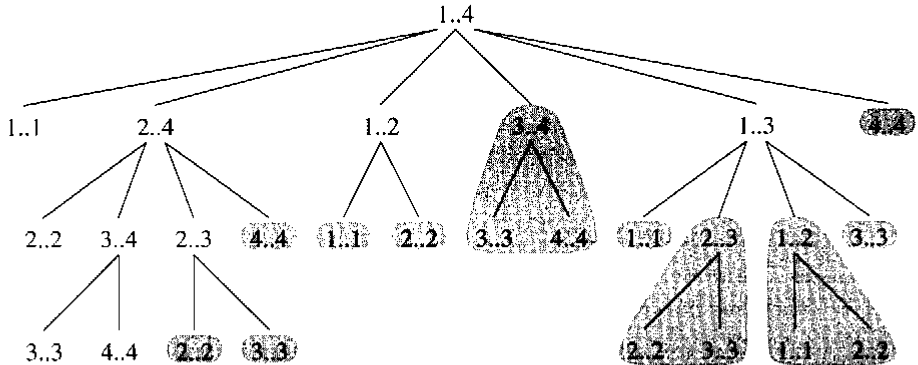
Computing
Optimal Value
Constructing
Optimal Solution

Overlapping
Subproblems

Longest
Common
Subsequence

Optimal
Binary Search
Trees

Entire subtrees overlap:



See Section 15.3 for more on optimal substructure and overlapping subproblems

Longest Common Subsequence

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Characterizing
Structure
Recursive
Definition
Computing
Optimal Value
Constructing
Optimal Solution

Optimal
Binary Search
Trees

- Sequence $Z = \langle z_1, z_2, \dots, z_k \rangle$ is a **subsequence** of another sequence $X = \langle x_1, x_2, \dots, x_m \rangle$ if there is a strictly increasing sequence $\langle i_1, \dots, i_k \rangle$ of indices of X such that for all $j = 1, \dots, k$, $x_{i_j} = z_j$
- I.e. as one reads through Z , one can find a match to each symbol of Z in X , in order (though not necessarily contiguous)
- E.g. $Z = \langle B, C, D, B \rangle$ is a subsequence of $X = \langle A, B, C, B, D, A, B \rangle$ since $z_1 = x_2$, $z_2 = x_3$, $z_3 = x_5$, and $z_4 = x_7$
- Z is a **common subsequence** of X and Y if it is a subsequence of both
- The goal of the **longest common subsequence problem** is to find a maximum-length common subsequence (LCS) of sequences $X = \langle x_1, x_2, \dots, x_m \rangle$ and $Y = \langle y_1, y_2, \dots, y_n \rangle$

Characterizing the Structure of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Characterizing
Structure

Recursive
Definition
Computing
Optimal Value
Constructing
Optimal Solution

Optimal
Binary Search
Trees

- Given sequence $X = \langle x_1, \dots, x_m \rangle$, the i th **prefix** of X is $X_i = \langle x_1, \dots, x_i \rangle$
- **Theorem** If $X = \langle x_1, \dots, x_m \rangle$ and $Y = \langle y_1, \dots, y_n \rangle$ have LCS $Z = \langle z_1, \dots, z_k \rangle$, then
 - ① $x_m = y_n \Rightarrow z_k = x_m = y_n$ and Z_{k-1} is LCS of X_{m-1} and Y_{n-1}
 - If $z_k \neq x_m$, can lengthen Z , \Rightarrow contradiction
 - If Z_{k-1} not LCS of X_{m-1} and Y_{n-1} , then a longer CS of X_{m-1} and Y_{n-1} could have x_m appended to it to get CS of X and Y that is longer than Z , \Rightarrow contradiction
 - ② If $x_m \neq y_n$, then $z_k \neq x_m$ implies that Z is an LCS of X_{m-1} and Y
 - If $z_k \neq x_m$, then Z is a CS of X_{m-1} and Y . Any CS of X_{m-1} and Y that is longer than Z would also be a longer CS for X and Y , \Rightarrow contradiction
 - ③ If $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}
 - Similar argument to (2)

Recursively Defining the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence
Characterizing
Structure

Recursive
Definition

Computing
Optimal Value
Constructing
Optimal Solution

Optimal
Binary Search
Trees

- The theorem implies the kinds of subproblems that we'll investigate to find LCS of $X = \langle x_1, \dots, x_m \rangle$ and $Y = \langle y_1, \dots, y_n \rangle$
- If $x_m = y_n$, then find LCS of X_{m-1} and Y_{n-1} and append x_m ($= y_n$) to it
- If $x_m \neq y_n$, then find LCS of X and Y_{n-1} and find LCS of X_{m-1} and Y and identify the longest one
- Let $c[i, j] = \text{length of LCS of } X_i \text{ and } Y_j$

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

Computing the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest

Common

Subsequence

Characterizing
Structure

Recursive
Definition

Computing
Optimal Value

Constructing
Optimal Solution

Optimal

Binary Search
Trees

```

1  allocate  $b[1 \dots m, 1 \dots n]$  and  $c[0 \dots m, 0 \dots n]$ 
2  initialize  $c[i, 0] = 0$  and  $c[0, j] = 0 \forall 0 \leq i \leq m$  and  $0 \leq j \leq n$ 
3  for  $i = 1$  to  $m$  do
4      for  $j = 1$  to  $n$  do
5          if  $x_i == y_j$  then
6               $c[i, j] = c[i - 1, j - 1] + 1$ 
7               $b[i, j] = "\searrow"$ 
8          else if  $c[i - 1, j] \geq c[i, j - 1]$  then
9               $c[i, j] = c[i - 1, j]$ 
10              $b[i, j] = "\uparrow"$ 
11          else
12               $c[i, j] = c[i, j - 1]$ 
13               $b[i, j] = "\leftarrow"$ 
14          end
15      end
16  return  $(c, b)$ 
    
```

Algorithm 8: LCS-Length(X, Y, m, n)

What is the time complexity?

Computing the Value of an Optimal Solution (2)

CSCE423/823

$$X = \langle A, B, C, B, D, A, B \rangle, Y = \langle B, D, C, A, B, A \rangle$$

		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
i									
0	x_i		0	0	0	0	0	0	0
1	A		0	↑	↑	↑	↖		↖
2	B		0	1	↖	↖	↑	↖	↖
3	C		0	↑	↑	2	↖	↑	↑
4	B		0	↖	↑	↑	↑	3	↖
5	D		0	↑	↖	↑	↑	↑	↑
6	A		0	↑	↑	↑	↖	↑	4
7	B		0	↖	↑	↑	↑	↖	4

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Characterizing
Structure
Recursive
Definition

Computing
Optimal Value

Constructing
Optimal Solution

Optimal
Binary Search
Trees

Constructing an Optimal Solution from Computed Information

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common

Subsequence

Characterizing
StructureRecursive
DefinitionComputing
Optimal ValueConstructing
Optimal SolutionOptimal
Binary Search
Trees

- Length of LCS is stored in $c[m, n]$
- To print LCS, start at $b[m, n]$ and follow arrows until in row or column 0
- If in cell (i, j) on this path, when $x_i = y_j$ (i.e. when arrow is “↖”), print x_i as part of the LCS
- This will print LCS backwards

Constructing an Optimal Solution from Computed Information (2)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common
SubsequenceCharacterizing
Structure
Recursive
Definition
Computing
Optimal ValueConstructing
Optimal SolutionOptimal
Binary Search
Trees

```
1 if  $i == 0$  or  $j == 0$  then  
2     return  
3 if  $b[i, j] == \nwarrow$  then  
4     PRINT-LCS( $b, X, i - 1, j - 1$ )  
5     print  $x_i$   
6 else if  $b[i, j] == \uparrow$  then  
7     PRINT-LCS( $b, X, i - 1, j$ )  
8 else PRINT-LCS( $b, X, i, j - 1$ )
```

Algorithm 9: Print-LCS(b, X, i, j)

What is the time complexity?

Constructing an Optimal Solution from Computed Information (3)

CSCE423/823

$X = \langle A, B, C, B, D, A, B \rangle$, $Y = \langle B, D, C, A, B, A \rangle$, prints "BCBA"

		j	0	1	2	3	4	5	6
		y_j		B	D	C	A	B	A
i									
0	x_i		0	0	0	0	0	0	0
1	A		0	0	0	0	1	1	1
2	B		0	1	1	1	1	2	2
3	C		0	1	1	2	2	2	2
4	B		0	1	1	2	2	3	3
5	D		0	1	2	2	2	3	3
6	A		0	1	2	2	3	3	4
7	B		0	1	2	2	3	4	4

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Characterizing
Structure
Recursive
Definition
Computing
Optimal Value

Constructing
Optimal Solution

Optimal
Binary Search
Trees

Optimal Binary Search Trees

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
TreesCharacterizing
Structure
Recursive
Definition
Computing
Optimal Value

- Goal is to construct binary search trees such that most frequently sought values are near the root, thus minimizing expected search time
- Given a sequence $K = \langle k_1, \dots, k_n \rangle$ of n distinct keys in sorted order
- Key k_i has probability p_i that it will be sought on a particular search
- To handle searches for values not in K , have $n + 1$ *dummy keys* d_0, d_1, \dots, d_n to serve as the tree's leaves
- Dummy key d_i will be reached with probability q_i
- If $\text{depth}_T(k_i)$ is distance from root of k_i in tree T , then expected search cost of T is

$$1 + \sum_{i=1}^n p_i \text{depth}_T(k_i) + \sum_{i=0}^n q_i \text{depth}_T(d_i)$$

- An **optimal binary search tree** is one with minimum expected search cost

Optimal Binary Search Trees (2)

CSCE423/823

Introduction

Rod Cutting

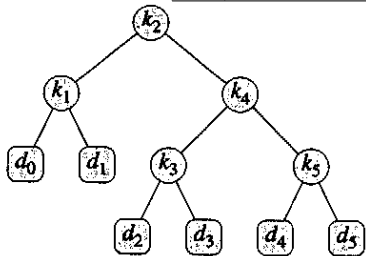
Matrix-Chain Multiplication

Longest Common Subsequence

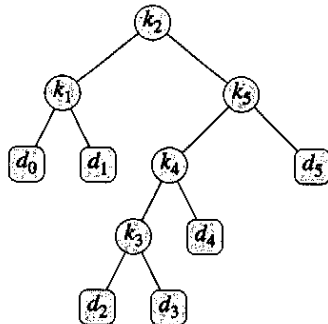
Optimal Binary Search Trees

- Characterizing Structure
- Recursive Definition
- Computing Optimal Value

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10



expected cost = 2.80



expected cost = 2.75 (optimal)

Characterizing the Structure of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
TreesCharacterizing
StructureRecursive
Definition
Computing
Optimal Value

- Observation: Since K is sorted and dummy keys interspersed in order, any subtree of a BST must contain keys in a contiguous range k_i, \dots, k_j and have leaves d_{i-1}, \dots, d_j
- Thus, if an optimal BST T has a subtree T' over keys k_i, \dots, k_j , then T' is optimal for the subproblem consisting of only the keys k_i, \dots, k_j
 - If T' weren't optimal, then a lower-cost subtree could replace T' in T ,
 \Rightarrow contradiction
- Given keys k_i, \dots, k_j , say that its optimal BST roots at k_r for some $i \leq r \leq j$
- Thus if we make right choice for k_r and optimally solve the problem for k_i, \dots, k_{r-1} (with dummy keys d_{i-1}, \dots, d_{r-1}) and the problem for k_{r+1}, \dots, k_j (with dummy keys d_r, \dots, d_j), we'll end up with an optimal solution
- Since we don't know optimal k_r , we'll try them all

Recursively Defining the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

Characterizing
Structure

Recursive
Definition

Computing
Optimal Value

- Define $e[i, j]$ as the expected cost of searching an optimal BST built on keys k_i, \dots, k_j
- If $j = i - 1$, then there is only the dummy key d_{i-1} , so $e[i, i - 1] = q_{i-1}$
- If $j \geq i$, then choose root k_r from k_i, \dots, k_j and optimally solve subproblems k_i, \dots, k_{r-1} and k_{r+1}, \dots, k_j
- When combining the optimal trees from subproblems and making them children of k_r , we increase their depth by 1, which increases the cost of each by the sum of the probabilities of its nodes
- Define $w(i, j) = \sum_{\ell=i}^j p_{\ell} + \sum_{\ell=i-1}^j q_{\ell}$ as the sum of probabilities of the nodes in the subtree built on k_i, \dots, k_j , and get

$$e[i, j] = p_r + (e[i, r - 1] + w(i, r - 1)) + (e[r + 1, j] + w(r + 1, j))$$

Recursively Defining the Value of an Optimal Solution (2)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
TreesCharacterizing
StructureRecursive
DefinitionComputing
Optimal Value

- Note that

$$w(i, j) = w(i, r - 1) + p_r + w(r + 1, j)$$

- Thus we can condense the equation to

$$e[i, j] = e[i, r - 1] + e[r + 1, j] + w(i, j)$$

- Finally, since we don't know what k_r should be, we try them all:

$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1 \\ \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w(i, j)\} & \text{if } i \leq j \end{cases}$$

- Will also maintain table $root[i, j] = \text{index } r \text{ for which } k_r \text{ is root of an optimal BST on keys } k_i, \dots, k_j$

Computing the Value of an Optimal Solution

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
MultiplicationLongest
Common
SubsequenceOptimal
Binary Search
TreesCharacterizing
StructureRecursive
DefinitionComputing
Optimal Value

```
1  allocate  $e[1 \dots n + 1, 0 \dots n]$ ,  $w[1 \dots n + 1, 0 \dots n]$ , and  
    $root[1 \dots n, 1 \dots n]$   
2  initialize  $e[i, i - 1] = w[i, i - 1] = q_{i-1} \forall 1 \leq i \leq n + 1$   
3  for  $\ell = 1$  to  $n$  do  
4      for  $i = 1$  to  $n - \ell + 1$  do  
5           $j = i + \ell - 1$   
6           $e[i, j] = \infty$   
7           $w[i, j] = w[i, j - 1] + p_j + q_j$   
8          for  $r = i$  to  $j$  do  
9               $t = e[i, r - 1] + e[r + 1, j] + w[i, j]$   
10             if  $t < e[i, j]$  then  
11                  $e[i, j] = t$   
12                  $root[i, j] = r$   
13             end  
14         end  
15     end  
16 return  $(e, root)$ 
```

Algorithm 10: Optimal-BST(p, q, n)

What is the time complexity?

Computing the Value of an Optimal Solution (2)

CSCE423/823

Introduction

Rod Cutting

Matrix-Chain
Multiplication

Longest
Common
Subsequence

Optimal
Binary Search
Trees

Characterizing
Structure

Recursive
Definition

Computing
Optimal Value

i	0	1	2	3	4	5
p_i		0.15	0.10	0.05	0.10	0.20
q_i	0.05	0.10	0.05	0.05	0.05	0.10

e w

