Data and Algorithm Analysis

Chapter 15 — Dynamic Programming

Lenwood S. Heath

Outline

Table of Contents

Optimization Problems

Find a solution that maximizes or minimizes some objective function.

Example — Traveling Salesman Problem

TRAVELING SALESMAN PROBLEM (TSP)

INSTANCE: Complete undirected graph $G = (V, E)$; weight function $w : E \to \mathbb{Z}$.

SOLUTION: A permutation $v_1, v_2, \ldots, v_n$ of V such that

$$w(v_n, v_1) + \sum_{i=1}^{n-1} w(v_i, v_{i+1})$$

is minimized.

- Solution is any permutation of V.
- Objective function to minimize is the given sum.

Other Examples

- Shortest path in a graph
- Minimum spanning tree
- Edit distance between strings
- Pattern matching
- Scheduling
- Maximum flow in a flow network
- Others

Table of Contents

Rod Cutting

- A company buys rods of length $n \in \mathbb{N}$.
- It cuts rods into integer-length pieces, which it sells.
- A rod of length i gets a price p_i.

ROD CUTTING

INSTANCE: Rod length n and prices $p_1, p_2, \ldots, p_n$.

SOLUTION: Positive integer rod lengths $i_1, i_2, \ldots, i_k$ such that

$$n = i_1 + i_2 + \cdots + i_k$$

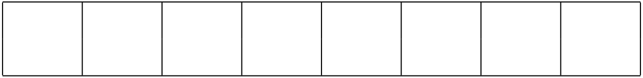
and

$$\sum_{j=1}^{k} p_{i_j}$$

is maximized.

Rod Cutting — Solution Space

Suppose the length of the initial rod is $n = 8$.



There are $n - 1 = 7$ places where a cut can occur.

So, how many different ways can this rod be cut into pieces?

For a general length n, what formula expresses the number of different ways to cut the length n rod into pieces?

Rod Cutting — Example Instance

Figure 15.1 contains this example of an instance for $n = 10$:

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

Optimal Substructure

Without loss of generality, assume that cuts are made left to right. For $0 \leq j \leq n$, let r_j be the optimal revenue for cutting a rod of length j. Suppose the first cut is at position i, where $1 \leq i \leq n$. Then, the optimal revenue for n is

$$p_i + r_{n-i}.$$

Since we do not know what i should be, we have

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

Recurrence

More generally, we get this recurrence for r_j, where $0 \le j \le n$:

$$r_j = \begin{cases} 0 & \text{if } j = 0; \\ \max\limits_{1 \le i \le j} p_i + r_{j-i} & \text{otherwise.} \end{cases}$$

By iterating from $j = 0$ to n, we can compute r_n.

The essence of dynamic programming is a recurrence that computes the optimal objective value in terms of optimal objective values for smaller instances.

Naive Recursive Implementation

NAIVE-CUT-ROD(n, p)

```
    // n is the initial rod length.
    // p is an array of p_i values.
    // Returns r_n.
    if n == 0
        return 0
    r_n = p_n
    for j = 1 to n - 1
        r_n = max(r_n, p_j + NAIVE-CUT-ROD(n - j, p))
    return r_n
```

Time complexity: considers all 2^{n-1} cuts explicitly, so $T(n) = \Omega(2^n)$.
Improve by **memoizing** r_i values.

Dynamic Programming — Bottom-Up Version

```
BOTTOM-UP-CUT-ROD(n, p)
    // n is the initial rod length.
    // p is an array of p_i values.
    // Let r_0, r_1, ..., r_n be new variables.
    // Returns r_n.
    r_0 = 0
    for i = 1 to n
        r_i = p_i
        for j = 1 to i − 1
            r_i = max(r_i, p_j + r_{i−j})
    return r_n
```

Time complexity?

Table of Contents

Dynamic Programming Paradigm

1. Identify subproblems and optimal substructure.
2. Develop a recurrence to compute the optimal objective values for each subproblem.
3. Compute a table of these values using the recurrence.
4. Backtrace to find the actual optimal solution.

Rod Cutting — Example Instance

i	0	1	2	3	4	5	6	7	8	9	10
p_i	0	1	5	8	9	10	17	17	20	24	30
r_i											
c_i											

Recurrence:

$$r_i = \begin{cases} 0 & \text{if } i = 0; \\ \max_{1 \le j \le i} p_j + r_{i-j} & \text{otherwise.} \end{cases}$$

Rod Cutting — Computing the Table

i	0	1	2	3	4	5	6	7	8	9	10
p_i	0	1	5	8	9	10	17	17	20	24	30
r_i	0	1	5	8	10	13	17	18	22	25	30
c_i	0	1	2	3	2	2	6	1	2	3	10

Recurrence:

$$r_i = \begin{cases} 0 & \text{if } i = 0; \\ \max\limits_{1 \le j \le i} p_j + r_{i-j} & \text{otherwise.} \end{cases}$$

Table of Contents

Strings

A **string** is a sequence of characters from some finite alphabet Σ.

Example

DNA alphabet $\Sigma = \{\mathrm{A}, \mathrm{C}, \mathrm{G}, \mathrm{T}\}$. String

$$S = \mathrm{G}, \mathrm{G}, \mathrm{C}, \mathrm{A}, \mathrm{G}, \mathrm{T}, \mathrm{C}, \mathrm{T}$$

written

$$S = \mathrm{GGCAGTCT}$$

Length of S is 8. Empty string ϵ has length 0.

Substrings and Subsequences

A **substring** of a string $S = s_1 s_2 \cdots s_n$ is a string

$$S[i..j] \quad = \quad s_i s_{i+1} \cdots s_j.$$

A **subsequence** of S is a string

$$S' \quad = \quad s_{i_1} s_{i_2} \cdots s_{i_k},$$

where

$$1 \leq i_1 < i_2 < \cdots < i_k \leq n.$$

Example

Substrings of $S = \mathrm{GGCAGTCT}$ include S, ϵ, T, GCAG, but not GAT, which is a subsequence. Subsequences of S include all substrings plus CCT and GGGC.

Common Subsequence and LCS

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ be strings over Σ. The string $Z = z_1 z_2 \ldots z_k$ is a **common subsequence** of X and Y if it is a subsequence of both X and Y.
Z is a **longest common subsequence (LCS)** of X and Y if it is a common subsequence of maximum length.

Example

$$
\begin{aligned}
X &= \mathrm{GGCAGTCT} \\
Y &= \mathrm{TCTGATGC}
\end{aligned}
$$

TCT is a common subsequence of length 3.
What is an LCS of X and Y?

LCS Problem

LONGEST COMMON SUBSEQUENCE (LCS)

INSTANCE: Strings $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ over Σ.

SOLUTION: String $Z = z_1 z_2 \ldots z_k$ that is a common subsequence of X and Y of maximum length.

Optimal Substructure of LCS

Let $X_i = x_1 x_2 \cdots x_i$ be the prefix of X of length i.

Theorem

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$. Let $Z = z_1 z_2 \ldots z_k$ be an LCS of X and Y.

1. *If $x_m = y_n$, then $z_k = x_m = y_n$ and Z_{k-1} is an LCS of X_{m-1} and Y_{n-1}.*
2. *If $x_m \neq y_n$ and $z_k \neq x_m$, then Z is an LCS of X_{m-1} and Y.*
3. *If $x_m \neq y_n$ and $z_k \neq y_n$, then Z is an LCS of X and Y_{n-1}.*

Proof.

In the textbook or on the board. □

Subinstances of LCS

Let $X = x_1 x_2 \cdots x_m$ and $Y = y_1 y_2 \cdots y_n$ be an instance of LCS.

Subinstances are pairs of all prefixes of X and Y, that is,

$$\begin{aligned} X_i &= x_1 x_2 \cdots x_i \\ Y_j &= y_1 y_2 \cdots y_j, \end{aligned}$$

where $0 \leq i \leq m$ and $0 \leq j \leq n$.

For subinstance i, j, define the length of a longest common subsequence of X_i and Y_j to be $c(i, j)$.

Recurrence for LCS

Base cases occur when $i = 0$ or $j = 0$:

$$\begin{array}{rcll} c(i,0) & = & 0 & 0 \le i \le m; \\ c(0,j) & = & 0 & 0 \le j \le n. \end{array}$$

General cases are for $1 \le i \le m$ and $1 \le j \le n$:

$$c(i,j) = \max \begin{cases} c(i-1,j-1)+1 & \text{if } x_i = y_j; \\ \max\{c(i,j-1), c(i-1,j)\} & \text{if } x_i \ne y_j. \end{cases}$$

Recording the Choices for $c(i,j)$ with Arrows

$$c(i,j) = \max \begin{cases} c(i-1,j-1)+1 & \text{if } x_i = y_j; \\ \max\{c(i,j-1), c(i-1,j)\} & \text{if } x_i \neq y_j. \end{cases}$$

The value of $c(i,j)$ depends directly on three other c values. The values that actually lead to a particular $c(i,j)$ value can be recorded with arrows in the table box.

$$\begin{array}{c|c|c|} & j-1 & j \\ \hline i-1 & c(i-1,j-1) & c(i-1,j) \\ \hline i & c(i,j-1) & \begin{array}{c} \nwarrow \quad \uparrow \\ \leftarrow c(i,j) \end{array} \\ \hline \end{array}$$

Example — Empty Table

$c(i,j)$	0	G 1	A 2	C 3	G 4	C 5	A 6
0							
C 1							
A 2							
G 3							
A 4							
G 5							

Example — Base Cases

$c(i,j)$	0	G 1	A 2	C 3	G 4	C 5	A 6
0	0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0
C 1	$\uparrow$ 0						
A 2	$\uparrow$ 0						
G 3	$\uparrow$ 0						
A 4	$\uparrow$ 0						
G 5	$\uparrow$ 0						

Example — General Case — Row $i = 1$

$c(i,j)$	0	G 1	A 2	C 3	G 4	C 5	A 6
0	0	$\leftarrow 0$	$\leftarrow 0$	$\leftarrow 0$	$\leftarrow 0$	$\leftarrow 0$	$\leftarrow 0$
C 1	$\uparrow$ 0	$\uparrow$ $\leftarrow 0$	$\uparrow$ $\leftarrow 0$	$\nwarrow$ 1	$\leftarrow 1$	$\nwarrow$ $\leftarrow 1$	$\leftarrow 1$
A 2	$\uparrow$ 0						
G 3	$\uparrow$ 0						
A 4	$\uparrow$ 0						
G 5	$\uparrow$ 0						

Example — General Case — Complete Table

$c(i,j)$	0	G 1	A 2	C 3	G 4	C 5	A 6
0	0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0
C 1	$\uparrow$ 0	$\uparrow$ $\leftarrow$0	$\uparrow$ $\leftarrow$0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ $\leftarrow$ 1	$\leftarrow$ 1
A 2	$\uparrow$ 0	$\uparrow$ $\leftarrow$0	$\nwarrow$ 1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\nwarrow$ 2
G 3	$\uparrow$ 0	$\nwarrow$ 1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ $\leftarrow$2
A 4	$\uparrow$ 0	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ $\leftarrow$2	$\uparrow$ $\leftarrow$2	$\nwarrow$ 3
G 5	$\uparrow$ 0	$\nwarrow\uparrow$ 1	$\uparrow$ 2	$\uparrow$ $\leftarrow$2	$\nwarrow$ 3	$\leftarrow$ 3	$\uparrow$ $\leftarrow$3

Example — Backtrace — Get LCS AGA

$c(i,j)$	0	G 1	A 2	C 3	G 4	C 5	A 6
0	0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0	$\leftarrow$ 0
C 1	$\uparrow$ 0	$\uparrow$ $\leftarrow$0	$\uparrow$ $\leftarrow$0	$\nwarrow$ 1	$\leftarrow$ 1	$\nwarrow$ $\leftarrow$ 1	$\leftarrow$ 1
A 2	$\uparrow$ 0	$\uparrow$ $\leftarrow$0	$\nwarrow$ 1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\nwarrow$ 2
G 3	$\uparrow$ 0	$\nwarrow$ 1	$\uparrow$ $\leftarrow$1	$\uparrow$ $\leftarrow$1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ $\leftarrow$2
A 4	$\uparrow$ 0	$\uparrow$ 1	$\nwarrow$ 2	$\leftarrow$ 2	$\uparrow$ $\leftarrow$2	$\uparrow$ $\leftarrow$2	$\nwarrow$ 3
G 5	$\uparrow$ 0	$\nwarrow$ $\uparrow$ 1	$\uparrow$ 2	$\uparrow$ $\leftarrow$2	$\nwarrow$ 3	$\leftarrow$ 3	$\uparrow$ $\leftarrow$3

Dynamic Programming Paradigm — LCS

1. Identify subproblems and optimal substructure.
2. Develop a recurrence to compute the optimal objective values for each subproblem.
3. Compute a table of these values using the recurrence.
4. Backtrace to find the actual optimal solution.

Time complexity to fill the table: $\Theta(mn)$
Time complexity to find one LCS from the table: $\Theta(m + n)$

Table of Contents

Cost of Multiplying Matrices

Suppose A_1, A_2, and A_3 are 10×100, 100×2, and 2×3 matrices.
Multiplying $A_1 \times A_2$ requires $2 * 10 * 100 = 2000$ scalar multiplications.
Multiplying $(A_1 \times A_2) \times A_3$ requires $2000 + 10 * 2 * 3 = 2060$ scalar multiplications.
Changing the order of evaluation, multiplying $A_1 \times (A_2 \times A_3)$ requires $100 * 2 * 3 + 10 * 100 * 3 = 3600$ scalar multiplications.
Order of evaluation matters!

Optimization Problem

MATRIX CHAIN MULTIPLICATION

INSTANCE: Matrices $A_1, A_2, \ldots, A_n$ where A_i has dimensions $p_{i-1} \times p_i$.

SOLUTION: Parenthesization of $A_1 \times A_2 \times \cdots \times A_n$ that minimizes the number of scalar multiplications.

Solution could also be an **expression tree** — a binary tree with $\times$ at the internal nodes and matrices at the leaves.

Dynamic Programming Subinstances

Subinstance:

$$A_i \times A_{i+1} \times \cdots \times A_j,$$

where $1 \leq i \leq j \leq n$.
Define variable $m[i,j]$ to be the minimum number of scalar multiplications to compute $A_i \times A_{i+1} \times \cdots \times A_j$.

Dynamic Programming Recurrence

Base cases: $m[i,i] = 0$, for $1 \leq i \leq n$.

General case: for $1 \leq i < j \leq n$,

$$m[i,j] \quad = \quad \min_{i \leq k < j} m[i,k] + m[k+1,j] + p_{i-1}p_k p_j.$$

Table size: ?

Time complexity to fill in table: ?

Example — Figure 15.5

There are $n = 6$ matrices with dimensions

p_0	p_1	p_2	p_3	p_4	p_5	p_6
30	35	15	5	10	20	25

Example — Table to Fill In

$m[i,j]$	1	2	3	4	5	6
1						
2						
3						
4						
5						
6						

Except for the base cases, for each $m[i,j]$ value, there is a k value that gives the root of an expression tree for that $m[i,j]$ value. This needs to be put in the table as well.

Example — Base Cases

$m[i,j]$	1	2	3	4	5	6
1	0 —					
2		0 —				
3			0 —			
4				0 —		
5					0 —	
6						0 —

Example — General Case — Next Diagonal

$m[i,j]$	1	2	3	4	5	6
1	0 —	15750 1				
2		0 —	2625 2			
3			0 —	750 3		
4				0 —	1000 4	
5					0 —	5000 5
6						0 —

Example — General Case — Finish

$m[i,j]$	1	2	3	4	5	6
1	0 —	15750 1	7875 1	9375 3	11875 3	15125 3
2		0 —	2625 2	4375 3	7125 3	10500 3
3			0 —	750 3	2500 3	5375 3
4				0 —	1000 4	3500 5
5					0 —	5000 5
6						0 —

Dynamic Programming Paradigm — Matrix Chain Multiplication

1. Identify subproblems and optimal substructure.
2. Develop a recurrence to compute the optimal objective values for each subproblem.
3. Compute a table of these values using the recurrence.
4. Backtrace to find the actual optimal solution.

Use backtrace in the previous example.
Time complexity to fill the table: $\Theta(n^3)$
Time complexity to find expression tree from the table: $\Theta(n)$