Lecture 08 — Dynamic Programming (Part 2)

COMPSCI 308 — Design and Analysis of Algorithms

Xing Shi Cai https://newptcai.gitlab.io

Duke Kunshan University, Kunshan, China

Jan-Mar 2026

Summary

Assignments[1]

Practice makes perfect!

Introduction to Algorithms (ITA) 📖 Required Readings:

- Section 15.2.
- Section 15.3.

✏️ Required Exercises:

- Exercises 15.2 — 1–6. Exercises 15.3 — 1–5.

[1] 💣 Assignments will not be collected; however, quiz problems will be selected from them. (This includes both Readings and Exercises.)

ITA 15.2 Matrix-Chain Multiplication

Product of more than 2 matrices.

Let A and B be two matrices.

Let a_{ij} and b_{ij} be the entries of A and B respectively.

Let $C = A \cdot B$ and c_{ij} be the entries of C.

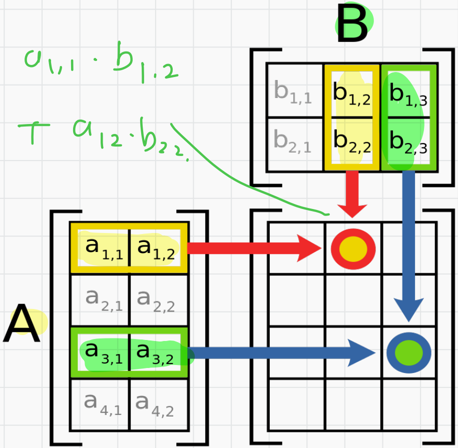
Then

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \cdot b_{kj}$$

This is called the row-column rule.

$a_{1,1} \cdot b_{1,2}$

$+ a_{12} \cdot b_{22}$



C

Figure 1: Matrix Multiplication

Given matrices A of size $m \times n$ and B of size $n \times p$,

$$A = \left.\underbrace{\begin{bmatrix} a_{11} & a_{12} & \cdots & a_{1n} \\ a_{21} & a_{22} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & a_{mn} \end{bmatrix}}_{n}\right\} m, \quad B = \left.\underbrace{\begin{bmatrix} b_{11} & b_{12} & \cdots & b_{1p} \\ b_{21} & b_{22} & \cdots & b_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ b_{n1} & b_{n2} & \cdots & b_{np} \end{bmatrix}}_{p}\right\} n$$

The product $C = A \times B$ is a matrix of size $m \times p$

$$C = \begin{bmatrix} c_{11} & c_{12} & \cdots & c_{1p} \\ c_{21} & c_{22} & \cdots & c_{2p} \\ \vdots & \vdots & \ddots & \vdots \\ c_{m1} & c_{m2} & \cdots & c_{mp} \end{bmatrix}$$

The total # of $\times$ is

$m \cdot n \cdot p$

where

$$c_{ij} = \sum_{k=1}^{n} a_{ik} \times b_{kj}$$

The Pseudocode

🍰 If A is of size $m \times n$ and B is of size $n \times p$, how many ✖ do we need to compute AB?

Matrix-Multiply(A, B)
let C be a new A.rows $\times$ B.columns matrix
for $i = 1$ to A.rows **do** — m
  for $j = 1$ to B.columns **do** — p.
    $C_{ij} = 0$
    for $k = 1$ to A.columns **do** — k.
      $C_{ij} = C_{ij} + A_{ik} \times B_{kj}$ ← m×p×k times.
return C

🍰 Let

$$\underbrace{A = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}}_{2},\ \ \ B = \begin{bmatrix} 0 \\ 2 \end{bmatrix},\ \ \ C = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

2×2   2×1

- We have —

$$AB = \begin{bmatrix} 3\cdot 0 + 0\cdot 2 \\ 5\cdot 0 + 4\cdot 2 \end{bmatrix} = \begin{bmatrix} 0 \\ 8 \end{bmatrix} \qquad (AB)C = \begin{bmatrix} 0 \\ 8 \end{bmatrix}\begin{bmatrix} 2 & 4 \end{bmatrix} = \begin{bmatrix} 0\cdot 2 & 0\cdot 4 \\ 8\cdot 2 & 8\cdot 4 \end{bmatrix} = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

row : i

colum : j

Associative Law of Multiplication

🍰 Let

$$A = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

- We have —

$$AB = \begin{bmatrix} 0 \\ 8 \end{bmatrix} \qquad (AB)C = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

Associative Law of Multiplication

🍰 Let

$$A = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

2×1 $\qquad 1 \times 2.$

$$\begin{bmatrix} 0 \cdot 2 & 0 \cdot 4 \\ 2 \cdot 2 & 2 \cdot 4 \end{bmatrix}.$$

- We have —

$$AB = \begin{bmatrix} 0 \\ 8 \end{bmatrix} \qquad (AB)C = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

- We also have —

$$BC = \begin{bmatrix} 0 & 0 \\ 4 & 8 \end{bmatrix} \qquad A(BC) = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}\begin{bmatrix} 0 & 0 \\ 4 & 8 \end{bmatrix} = \begin{bmatrix} \overset{0}{3\cdot 0 + 0 \cdot 4} & \overset{0}{3 \cdot 0 + 0 \cdot 8} \\ 5 \cdot 0 + 4 \cdot 4 & 5 \cdot 0 + 4 \cdot 8 \end{bmatrix}$$

2×2

$$= \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}.$$

Associative Law of Multiplication

🍰 Let

$$A = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

- We have —

$$AB = \begin{bmatrix} 0 \\ 8 \end{bmatrix} \qquad (AB)C = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

The same.

- We also have —

$$BC = \begin{bmatrix} 0 & 0 \\ 4 & 8 \end{bmatrix} \qquad A(BC) = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

Associative Law of Multiplication

🍰 Let

$$A = \begin{bmatrix} 3 & 0 \\ 5 & 4 \end{bmatrix}, \qquad B = \begin{bmatrix} 0 \\ 2 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 4 \end{bmatrix}$$

- We have —

$$AB = \begin{bmatrix} 0 \\ 8 \end{bmatrix} \qquad (AB)C = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

- We also have —

$$BC = \begin{bmatrix} 0 & 0 \\ 4 & 8 \end{bmatrix} \qquad A(BC) = \begin{bmatrix} 0 & 0 \\ 16 & 32 \end{bmatrix}$$

The Associative Law

Given any matrices A, B, C (of compatible sizes), we have —

$$A(BC) = (AB)C$$

💡 So we often drop the brackets and simply write ABC.

Fully Parenthesized Product

When computing a product like $A_1 A_2 \ldots A_n$, we must decide the order of the ✖.

Parentheses can be used to indicate that order —

$$A_1 A_2 = (A_1 A_2) \qquad 1$$

Fully Parenthesized Product

When computing a product like $A_1 A_2 \ldots A_n$, we must decide the order of the $\times$.

Parentheses can be used to indicate that order —

$$A_1 A_2 = (A_1 A_2)$$ 1

$$A_1 A_2 A_3 = (A_1(A_2 A_3)) = ((A_1 A_2)A_3).$$ 2

Fully Parenthesized Product

When computing a product like $A_1 A_2 \ldots A_n$, we must decide the order of the ✖.

Parentheses can be used to indicate that order —

$$A_1 A_2 = (A_1 A_2)$$ 1

$$A_1 A_2 A_3 = (A_1(A_2 A_3)) = ((A_1 A_2) A_3).$$ 2

$$\begin{aligned} A_1 A_2 A_3 A_4 &= (A_1(A_2(A_3 A_4))) = (A_1((A_2 A_3) A_4)) \\ &= ((A_1 A_2)(A_3 A_4)) = ((A_1(A_2 A_3)) A_4) \\ &= (((A_1 A_2) A_3) A_4). \end{aligned}$$ 5

🤪 In how many ways can we fully parenthesize $A_1 A_2 \cdots A_n$? $\frac{1}{n}\binom{2(n-1)}{n-1}.$

$$= \frac{1}{n} \cdot \frac{(2(n-1))!}{(n-1)!\,(n-1)!}$$

$\boxed{\text{Catalan number.}}$

Why Do Parentheses Matter?

Consider three matrices A_1, A_2, A_3 of sizes $10 \times 100, 100 \times 5, 5 \times 50$.

🍰 How many scalar ✖ do we need to compute

It Takes 100 x 5 x 50 "x"

= 25,000

$$(A_1(A_2A_3))$$

10 x 100

100 x 5 5 x 50

100 x 50

+

=

75,000 "x"

It Takes 10 x 100 x 50 = 50,000

Why Do Parentheses Matter?

Consider three matrices A_1, A_2, A_3 of sizes $10 \times 100, 100 \times 5, 5 \times 50$.

🍰 How many scalar ✖ do we need to compute

$$(A_1(A_2A_3))$$

🍰 What about

$$((A_1A_2)A_3)$$

Takes. $10 \times 100 \times 5 = 5,000$ "x"

10×5 5×50

Take $10 \times 5 \times 50$

$= 2,500$ "x"

+

= In Total.

7,500 "x"

Why Do Parentheses Matter?

Consider three matrices A_1, A_2, A_3 of sizes $10 \times 100, 100 \times 5, 5 \times 50$.

🍰 How many scalar ✖ do we need to compute

$$(A_1(A_2A_3))$$

🍰 What about

$$((A_1A_2)A_3)$$

🍰 Can you think of a case when any parenthesization of $A_1, A_2, \dots A_n$ uses the same amount of ✖?

when $A_1, \dots, A_n$ are all squared.

Let $P(n)$ be the number of ways to fully parenthesize $A_1 A_2 \ldots A_n$. Then

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n > 1. \end{cases}$$

$(A_1 \cdots A_k)(A_{k+1} \cdots A_n)$

Let $P(n)$ be the number of ways to fully parenthesize $A_1 A_2 \ldots A_n$. Then

$$P(n) = \begin{cases} 1 & \text{if } n = 1, \\ \sum_{k=1}^{n-1} P(k)P(n-k) & \text{if } n > 1. \end{cases}$$

In COMPSCI 203, we have seen that $P(n)$ is the famous Catalan number —

$$P(n) = \frac{1}{n}\binom{2n-2}{n-1} = \Theta\left(\frac{4^n}{n^{3/2}}\right) \tag{1}$$

There are simply too many ways to check exhaustively.

🧩 Try to prove (1).

The Structure of the Optimal Solution

If the optimal way to parenthesize $A_i A_{i+1} \dots A_j$ is to "break" the chain into two parts between A_k and A_{k+1}, i.e.,

$$((A_i \dots A_k)(A_{k+1} \dots A_j))$$

then the parenthesizations of both

$$A_i \dots A_k \qquad \text{and} \qquad A_{k+1} \dots A_j$$

must both be optimal.

Otherwise we could get even better parenthesization.

A Recursive Solution

Let $m[i,j]$ be the minimum number of ✖ needed to compute $A_i A_{i+1} \dots A_j$.

Let the size of A_i be $p_{i-1} \times p_i$.

Then by the previous observation

No computation needed.

$\leq j-1$

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

$$\underbrace{(A_i \cdots A_k)}_{p_{i-1} \times p_k} \underbrace{(A_{k+1} \cdots A_j)}_{p_k \times p_j}.$$

🍰 Why is $m[i,j] = 0$ when $i = j$?

To compute $m[i,j]$, we can fill up the optimal cost table and split table as shown below.

$m[3,5]$ $= \min \{ m[3,3] + m[4,5] + 15 \times 5 \times 20,$ $m[3,4] + m[5,5] + 15 \times 10 \times 20 \}$

$A_3 (A_4 A_5)$

$(A_3 A_4) \cdot A_5$

Bottom up

$m[1,2]$ = The min "x" needed for $A_1 \cdot A_2$.

30×35

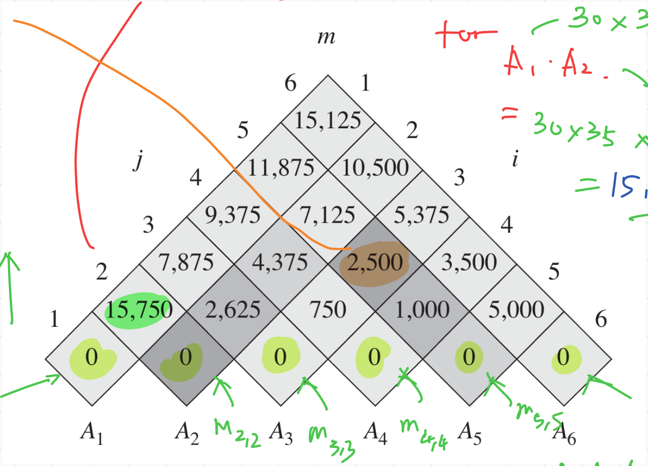
35×15

$= 30 \times 35 \times 15$

$= 15,750$

$A_3 : 15 \times 5$

$A_4 \cdot A_5 : 5 \times 20$



A_1 A_2 A_3 A_4 A_5 A_6

Figure 2: The matrix chain cost table for matrices with $\langle p_0, \dots p_6 \rangle = \langle 30, 35, 15, 5, 10, 20, 25 \rangle$

```
Matrix-Chain-Order(p)
Initialize m and s
for l = 2 to n do
    for i = 1 to n − l + 1 do
        j ← i + l − 1
        m[i,j] ← ∞
        for k = i to j − 1 do
            q ← m[i,k] + m[k+1,j] + p[i−1] × p[k] × p[j]
            if q < m[i,j] then
                m[i,j] ← q
                s[i,j] ← k
return m and s
```

$(A_i \cdots A_k)(A_{k+1} \cdots A_j)$.

POP

Print-Optimal-Parens(s, i, j)
if $i = j$ then
    print "A_i"
else
    print "("
    PRINT-OPTIMAL-PARENS$(s, i, s[i, j])$
    PRINT-OPTIMAL-PARENS$(s, s[i, j] + 1, j)$
    print ")"

🍰 What is the output when this is applied to the table on the right?

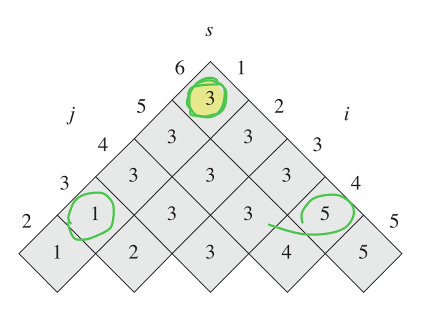


Figure 3: The matrix chain split table for matrices with $\langle p_0, \ldots p_6 \rangle = \langle 30, 35, 15, 5, 10, 20, 25 \rangle$

$(A_1 (A_2 A_3)) ((A_4 A_5) A_6)$

Consider matrices A_1, A_2, A_3 whose sizes are given by

$$\langle 4, 5, 7, 3 \rangle$$

What is $m(1,3)$?

$A_1 \; A_2 \; A_3$

$m[1,3]$

$A_1(A_2 A_3)$

$= \min\{ m[1,1] + m[2,3] + 4 \times 5 \times 3,$

$m[1,2] + m[3,3] + 4 \times 7 \times 3 \}.$

$= \min\{105 + 60,$

$140 + 84\}$

$=$

$(A_1 A_2) \cdot A_3$

$m[1,2]$

$= 4 \times 5 \times 7$

$m[2,3] = 5 \times 7 \times 3$

$= 105$

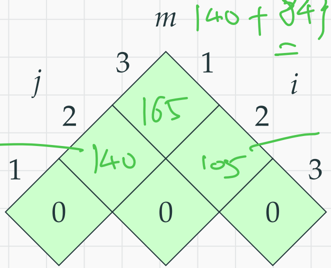


Figure 4: The matrix chain cost table

ITA 15.3 Elements of Dynamic Programming

★ Optimal Structure

A pattern in problems suitable for dynamic programming is —

1. A solution entails making a choice, e.g., an initial cut in a rod or a matrix split index. This choice results in subproblems.

★ Optimal Structure

A pattern in problems suitable for dynamic programming is —

1. A solution entails making a choice, e.g., an initial cut in a rod or a matrix split index. This choice results in subproblems.
2. Assume that you already have the choice leading to an optimal solution.

$$(A_1 \cdots A_k) \,\Big|\, (A_{k+1} \cdots A_j).$$

★ Optimal Structure

A pattern in problems suitable for dynamic programming is —

1. A solution entails making a choice, e.g., an initial cut in a rod or a matrix split index. This choice results in subproblems.
2. Assume that you already have the choice leading to an optimal solution.
3. Based on this choice, identify the subproblems and characterize the resulting space.

★ Optimal Structure

A pattern in problems suitable for dynamic programming is —

1. A solution entails making a choice, e.g., an initial cut in a rod or a matrix split index. This choice results in subproblems.
2. Assume that you already have the choice leading to an optimal solution.
3. Based on this choice, identify the subproblems and characterize the resulting space.
4. Prove subproblem solutions within an optimal solution must be optimal by contradiction — If a subproblem's solution isn't optimal, by replacing it with the optimal one, a better solution to the original problem is obtained.

Look Out for Pitfalls

In the directed graph below, the longest simple path from q to t is $q \to r \to t$.

🍰 What is the longest simple path from q to r and from r to t?

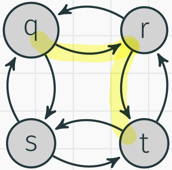


Figure 5: A directed graph

In the directed graph below, the longest simple path from q to t is $q \to r \to t$.

🍰 What is the longest simple path from q to r and from r to t?

↑

longest from q to r is

$q \to s \to t \to r$.

longest path from r to t is

$r \to q \to s \to t$.

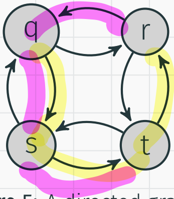


Figure 5: A directed graph

💣 Sometimes the optimal solution does not contain optimal solutions for subproblems.

🍰 Can you think of another problem like this?

Overlapping Subproblems

A sign that a problem is suitable for dynamic programming is that there are overlapping subproblems.

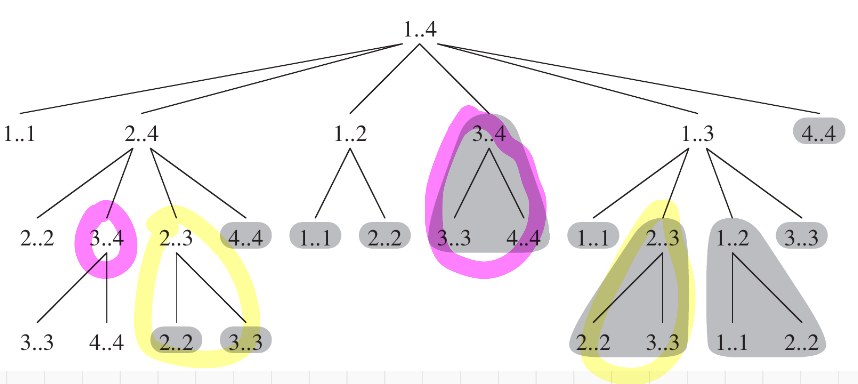


Figure 6: 📌 Example — The recursion tree for the matrix chain multiplication problem

Compute the Optimal Chain Naively

```
Recursive-Matrix-Chain(p, i, j)
if i = j then
    return 0
min ← ∞
for k = i to j − 1 do
    c ← Recursive-Matrix-Chain(p, i, k)
        + Recursive-Matrix-Chain(p, k + 1, j)
        + p[i − 1] × p[k] × p[j]
    if c < min then
        min ← c
return min
```

Compute the Optimal Chain Naively

```
Recursive-Matrix-Chain(p, i, j)
if i = j then
    return 0
min ← ∞
for k = i to j − 1 do
    c ← Recursive-Matrix-Chain(p, i, k)
        + Recursive-Matrix-Chain(p, k + 1, j)
        + p[i − 1] × p[k] × p[j]
    if c < min then
        min ← c
return min
```

Let $T(n)$ be the running time of this algorithm. Then we have

$$T(n) \geq 1 + \sum_{k=1}^{n-1} (T(k) + T(n-k) + 1) \quad \text{for } n > 1.$$

We can show by induction that

$$T(n) \geq 2^{n-1}.$$

More Examples in Dynamic Programming

More Examples in Dynamic Programming

The Change-Making Problem

📌 Example — Buying a Magic Mushroom[2]

Consider wanting to purchase a magic 🍄 priced at $\$99$.

However, the vendor only accepts 🪙 in denominations of

value

$$\$5, \$10, \$25, \$50.$$

Determine the fewest number of 🪙 required to make the purchase.

Figure 7: A magic 🍄 shop

[2]This is not in the textbook.

The Change-Making Problem

For a given set of 🪙 denominations $D = \{d_1, d_2, \dots, d_n\}$ and a specified amount A —

Determine the minimum count of coins, from the set, that combine to reach the amount A.

Let c_i represent the count of coins of denomination d_i.

Our objective is to minimize —

$$C(A) = \sum_{i=1}^{n} c_i$$

Subject to the constraint —

$$\sum_{i=1}^{n} c_i \cdot d_i = A$$

Dynamic Programming Solution

The following simple algorithm computes the optimal solution.

ChangeMaking(D, A)
  Initialize array $M[0 \dots A]$ with values set to ∞
  $M[0] = 0$
  for $i = 1$ to A **do**
    for each d in D **do**
      if $i - d \geq 0$ and $M[i-d] + 1 < M[i]$ **then**
        $M[i] = M[i-d] + 1$
  return $M[A]$

min # of coins needed to pay $i - d$.

What is the output when $D = \{5, 10, 25, 50\}$ and $A = 99$?

Not possible.

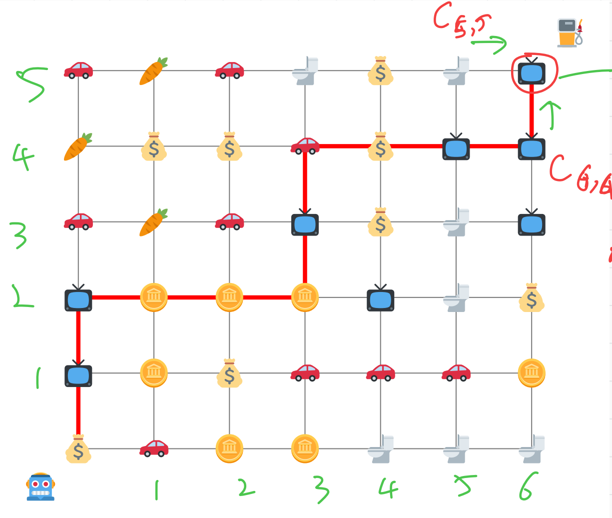
multiples of 5

The algorithm returns ∞

More Examples in Dynamic Programming

Even More Examples

Treasure Hunt Problem



$C_{6,5} = r_{6,5} + \max(C_{5,5}, C_{6,4})$

Figure 8: A treasure hunt on a grid

Treasure Hunt Problem

A 🤖 starts at coordinate $(0,0)$ on a $m \times n$ grid.

Each grid cell (i,j) holds a treasure with value t_{ij}.

The 🤖 can move either up or right one unit at a time.

🎯 Maximize the total treasure collected.

Let c_{ij} be the maximum treasure collected by the robot starting from coordinate (i,j).

🧩 Can you find a recursion for c_{ij}?

🧩 Can you write a dynamic programming algorithm in pseudocode for finding c_{mn}?

A palindrome is a string that reads the same forward and backward.

🤔 Given a string s, how can the longest palindromic substring in s be found? For example,

- $s =$ `"babad"` $\Rightarrow$ `"bab"` or `"aba"`
- $s =$ `"cbbd"` $\Rightarrow$ `"bb"`
- $s =$ `"a"` $\Rightarrow$ `"a"`

Let

$$P(i,j) = \begin{cases} \text{true}, & i = j, \\ (s_i = s_j), & j = i+1, \\ (s_i = s_j) \wedge P(i+1, j-1), & j > i+1. \end{cases}$$

$P(i,j)$ indicates whether $s[i \ldots j]$ is a palindrome. The longest palindromic substring is the valid (i,j) pair that maximizes $j-i+1$.

What are your main takeaways today? Any questions?