

# 3-1 Dynamic Programming

## (Part I: Examples)

Hengfeng Wei

hfwei@nju.edu.cn

September 17, 2018



# Taolu



# Steps for Applying DP:

## Steps for Applying DP:

- (I) Define subproblems
- (II) Set the goal
- (III) Identify the recurrence
  - ▶ larger subproblem  $\leftarrow$  # smaller subproblems
  - ▶ init. conditions
- (IV) Write pseudo-code: filling in “tables” in some order
- (V) Analyze the time complexity
- (VI) Extract the optimal solution (optionally)

## Steps for Applying DP:

- (I) Define subproblems
- (II) Set the goal
- (III) Identify the recurrence
  - ▶ larger subproblem  $\leftarrow$  # smaller subproblems
  - ▶ init. conditions
- (IV) Write pseudo-code: filling in “tables” in some order
- (V) Analyze the time complexity
- (VI) Extract the optimal solution (optionally)

## 1D Subproblems

**Input:**  $x_1, x_2, \dots, x_n$  (array, sequence, string)

**Subproblems:**  $x_1, x_2, \dots, x_i$  (prefix/suffix)

**#:**  $\Theta(n)$

- Examples:**
- ▶ Rod cutting
  - ▶ Maximum-sum subarray
  - ▶ Longest increasing subsequence
  - ▶ Printing Neatly

## 2D Subproblems

(I) Input:  $x_1, x_2, \dots, x_m; y_1, y_2, \dots, y_n$

Subproblems:  $x_1, x_2, \dots, x_i; y_1, y_2, \dots, y_j$

#:  $\Theta(mn)$

Examples: Edit distance, Longest common subsequence

## 2D Subproblems

(I) Input:  $x_1, x_2, \dots, x_m; y_1, y_2, \dots, y_n$

Subproblems:  $x_1, x_2, \dots, x_i; y_1, y_2, \dots, y_j$

#:  $\Theta(mn)$

Examples: Edit distance, Longest common subsequence

(II) Input:  $x_1, x_2, \dots, x_n$

Subproblems:  $x_i, \dots, x_j$

#:  $\Theta(n^2)$

Examples: Matrix chain multiplication, Optimal BST

## 3D Subproblems

- ▶ Floyd-Warshall algorithm

$$d(i, j, k) = \min \left( d(i, j, k - 1), d(i, k, k - 1) + d(k, j, k - 1) \right)$$

## DP on Graphs

(I) On rooted tree

Subproblems: rooted subtrees

(II) On DAG

Subproblems: nodes after/before in the topo. order

## DP on Graphs

(I) On rooted tree

Subproblems: rooted subtrees

(II) On DAG

Subproblems: nodes after/before in the topo. order

## Knapsack Problem

Subset sum problem, Change-making problem

# And Others . . .

# And Others . . .



# How to identify the recurrence?

How to identify the recurrence?

G U E S S

Make Choices by asking yourself the right question



**Make Choices** by asking yourself the right question



(I) Binary choice

- ▶ whether ...

(II) Multi-way choices

- ▶ where to ...
- ▶ which one ...

# Rod Cutting



## Rod Cutting Problem

Rod of length  $n$



length	$i$	1	2	3	4	5	$\dots$
price	$p_i$	1	5	8	9	10	$\dots$

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

$$r_n = p_{i_1} + p_{i_2} + \dots + p_{i_k}$$

$R(i)$  : max revenue obtained from *cutting a rod of length  $i$*

$$R(n)$$

$R(i)$  : max revenue obtained from cutting a rod of length  $i$

$$R(n)$$

Where is the leftmost cut?

$R(i)$  : max revenue obtained from cutting a rod of length  $i$

$$R(n)$$

Where is the leftmost cut?

$$R(i) = \max_{1 \leq j \leq i} (p_j + R(i - j))$$

$R(i)$  : max revenue obtained from cutting a rod of length  $i$

$$R(n)$$

Where is the leftmost cut?

$$R(i) = \max_{1 \leq j \leq i} (p_j + R(i - j))$$

$$R(0) = 0$$

$R(i)$  : max revenue obtained from cutting a rod of length  $i$

$$R(n)$$

Where is the leftmost cut?

$$R(i) = \max_{1 \leq j \leq i} (p_j + R(i - j))$$

$$R(0) = 0$$

$$O(n^2) = \Theta(n) \cdot O(n)$$

## Rod Cutting Problem (Problem 15.1-3)

Each cut incurs a fixed cost of  $c$ .

## Rod Cutting Problem (Problem 15.1-3)

Each cut incurs a fixed cost of  $c$ .

$$R(i) = \max_{1 \leq j \leq i} (p_j - c + R(i - j))$$

## Rod Cutting Problem (Problem 15.1-3)

Each cut incurs a fixed cost of  $c$ .

$$R(i) = \max_{1 \leq j \leq i} (p_j - c + R(i - j))$$

$$R(i) = \max \left( p_i, \max_{1 \leq j < i} (p_j - c + R(i - j)) \right)$$

## Printing Neatly (Problem 15-4)

A sequence of  $n$  words of lengths  $l_1, l_2, \dots, l_n$

Line width  $M$

$$\text{extra}[i, j] = M - (j - i) - \sum_{k=i}^j l_k$$

## Printing Neatly (Problem 15-4)

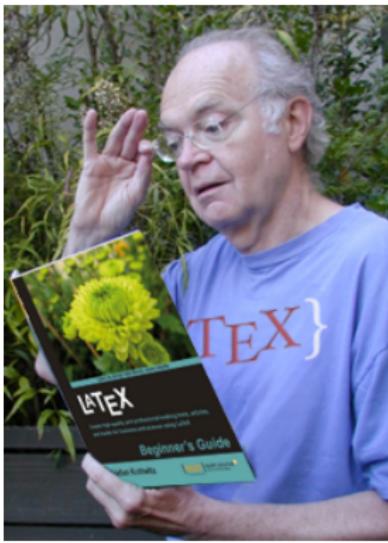
A sequence of  $n$  words of lengths  $l_1, l_2, \dots, l_n$

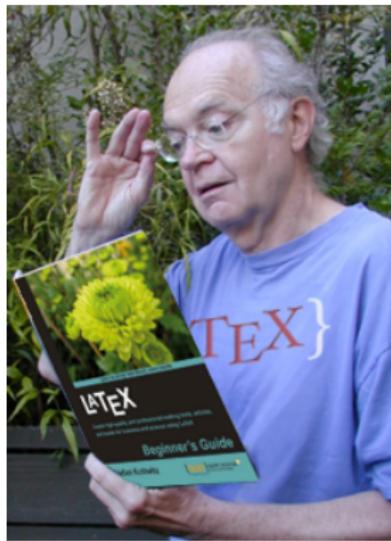
Line width  $M$

$$\text{extra}[i, j] = M - (j - i) - \sum_{k=i}^j l_k$$

To minimize the sum, over all lines *except the last*, of the **cubes** of the numbers of extra space characters at the ends of lines.

$$C(n) = \min_L \sum_{l_{[i,j]} \in L \wedge j \neq n} (\text{extra}[i, j])^3$$





$\text{\TeX}$  3.14159265  $\sim \pi$

A **sequence** of  $n$  words of lengths  $l_1, l_2, \dots, l_n$

A **sequence** of  $n$  words of lengths  $l_1, l_2, \dots, l_n$

$C(i)$  : min cost of neatly printing the *first*  $i$  words

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

$C(i, j)$  : min cost of neatly printing *words i through j*

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

*How many words to place on the *first* line?*

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

*How many words to place on the *first* line?*

$$C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ extra[i, i+k-1] \geq 0}} \left( extra[i, i+k-1] \right)^3 + C(i+k)$$

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

*How many words to place on the *first* line?*

$$C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ extra[i, i+k-1] \geq 0}} \left( extra[i, i+k-1] \right)^3 + C(i+k)$$

$$C(i) = 0, \quad \text{if } extra[i, n] \geq 0$$

$C(i)$  : min cost of neatly printing the *last* words  $i$  through  $n$

*How many words to place on the *first* line?*

$$C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ extra[i, i+k-1] \geq 0}} \left( extra[i, i+k-1] \right)^3 + C(i+k)$$

$$C(i) = 0, \quad \text{if } extra[i, n] \geq 0$$

$$O(nM)$$

---

```
1: procedure PRINTING-NEATLY( $n$ )
2:   for  $i \leftarrow n \downarrow 1$  do
3:     if  $\text{extra}[i, n] \geq 0$  then            $\triangleright$  put  $w_i$  through  $w_n$  on a line
4:        $C[i] \leftarrow 0$ 

6:   else
7:      $C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ \text{extra}[i, i+k-1] \geq 0}} \left( \text{extra}[i, i+k-1] \right)^3 + C(i+k)$ 
```

---

---

```
1: procedure PRINTING-NEATLY( $n$ )
2:   for  $i \leftarrow n \downarrow 1$  do
3:     if  $\text{extra}[i, n] \geq 0$  then            $\triangleright$  put  $w_i$  through  $w_n$  on a line
4:        $C[i] \leftarrow 0$ 
5:        $B[i] \leftarrow n$ 
6:     else
7:        $C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ \text{extra}[i, i+k-1] \geq 0}} (\text{extra}[i, i+k-1])^3 + C(i+k)$ 
8:        $B[i] = i + k$                        $\triangleright$  line break
```

---

---

```
1: procedure PRINTING-NEATLY( $n$ )
2:   for  $i \leftarrow n \downarrow 1$  do
3:     if  $\text{extra}[i, n] \geq 0$  then            $\triangleright$  put  $w_i$  through  $w_n$  on a line
4:        $C[i] \leftarrow 0$ 
5:        $B[i] \leftarrow n$ 
6:     else
7:        $C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ \text{extra}[i, i+k-1] \geq 0}} (\text{extra}[i, i+k-1])^3 + C(i+k)$ 
8:        $B[i] = i + k$                        $\triangleright$  line break
```

---

$B[1],$

---

```
1: procedure PRINTING-NEATLY( $n$ )
2:   for  $i \leftarrow n \downarrow 1$  do
3:     if  $\text{extra}[i, n] \geq 0$  then            $\triangleright$  put  $w_i$  through  $w_n$  on a line
4:        $C[i] \leftarrow 0$ 
5:        $B[i] \leftarrow n$ 
6:     else
7:        $C(i) = \min_{\substack{0 < k \leq (n-i+1) \\ \text{extra}[i, i+k-1] \geq 0}} (\text{extra}[i, i+k-1])^3 + C(i+k)$ 
8:        $B[i] = i + k$                        $\triangleright$  line break
```

---

$$B[1], \quad B[B[1] + 1], \quad \dots$$

*Neatness:*  $c[i, j] = extra[i, j]^1$

— Tianyun Zhang

*Neatness:  $c[i, j] = extra[i, j]$* <sup>1</sup>

— Tianyun Zhang



*Neatness:  $c[i, j] = extra[i, j]$* <sup>1</sup>

— Tianyun Zhang



Proof: Exchange Argument

## Longest Increasing Subsequence (Problem 15.4-5)

$A[1 \dots n]$

5, 2, 8, 6, 3, 6, 9, 7

Find (the length of) a longest increasing (non-decreasing) subsequence.

5, 2, 8, 6, 3, 6, 9, 7

$L(i)$  : the length of an LIS of  $A[1 \dots i]$

$L(n)$

$L(i)$  : the length of an LIS of  $A[1 \dots i]$

$L(n)$

Is  $A[i]$  in this LIS of  $A[1 \dots i]$ ?

$L(i)$  : the length of an LIS of  $A[1 \dots i]$

$L(n)$

Is  $A[i]$  in this LIS of  $A[1 \dots i]$ ?

$$L(i) = \max \left( \underbrace{L(i-1),}_{\text{NO}} \right)$$

$L(i)$  : the length of an LIS of  $A[1 \dots i]$

$L(n)$

Is  $A[i]$  in this LIS of  $A[1 \dots i]$ ?

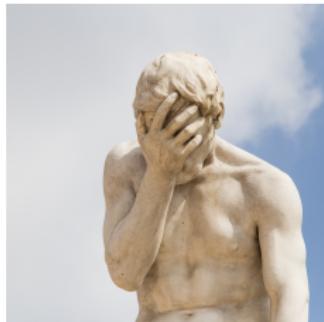
$$L(i) = \max \left( \underbrace{L(i-1)}_{\text{NO}}, \underbrace{1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)}_{\text{YES}} \right)$$

$L(i)$  : the length of an LIS of  $A[1 \dots i]$

$L(n)$

Is  $A[i]$  in this LIS of  $A[1 \dots i]$ ?

$$L(i) = \max \left( \underbrace{L(i-1)}_{\text{NO}}, \underbrace{1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)}_{\text{YES}} \right)$$



$L(i)$  : the length of an LIS *ending with*  $A[i]$

$$\max_i L(i)$$

$L(i)$  : the length of an LIS *ending with*  $A[i]$

$$\max_i L(i)$$

*What is the previous element?*

$L(i)$  : the length of an LIS *ending with*  $A[i]$

$$\max_i L(i)$$

*What is the previous element?*

$$L(i) = 1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)$$

$L(i)$  : the length of an LIS *ending with*  $A[i]$

$$\max_i L(i)$$

*What is the previous element?*

$$L(i) = 1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)$$

$$L(1) = 1$$

$L(i)$  : the length of an LIS *ending with*  $A[i]$

$$\max_i L(i)$$

*What is the previous element?*

$$L(i) = 1 + \max_{j < i \wedge A[j] \leq A[i]} L(j)$$

$$L(1) = 1$$

$$O(n^2) = \Theta(n) \cdot O(n)$$

$$\text{LIS}(A) = \text{LCS}\left(A, \text{SORT}(A)\right)$$

$$\text{LIS}(A) = \text{LCS}\left(A, \text{SORT}(A)\right)$$

$$O(n^2) = O(n \log n) + O(n^2)$$

## Longest Increasing Subsequence (Problem 15.4-6)

$O(n \log n)$

## Longest Increasing Subsequence (Problem 15.4-6)

$O(n \log n)$



## Longest Increasing Subsequence (Problem 15.4-6)

$O(n \log n)$



Answer by [Eugene Yarovoi](#) © Quora

$E[l] = \{all\ increasing\ subsequences\ of\ length\ l\}$

$$E[l] = \{ \text{all increasing subsequences of length } l \}$$

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset, \quad \forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow$ 
5:        $\{ \text{all inc. subseq. of length } l \}$ 
6:   return
```

---

$$E[l] = \{ \text{all increasing subsequences of length } l \}$$

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset, \quad \forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow$ 
5:        $\{ \text{all inc. subseq. of length } l \}$ 
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

$$E[l] = \{ \text{all increasing subsequences of length } l \}$$

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset, \quad \forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow \quad \triangleright \text{By extending each shorter subseq.}$ 
5:       {all inc. subseq. of length  $l$ }
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

$E[l] = \{ \text{all increasing subsequences of length } l \}$

$E[l] = \{ \text{all increasing subsequences of length } l \}$

$\langle 3 \quad 9 \quad 11 \quad 13 \rangle$

$\langle 4 \quad 6 \quad 10 \quad 15 \rangle$

$\langle 2 \quad 5 \quad 12 \quad 18 \rangle$

$E[l] = \{ \text{all increasing subsequences of length } l \}$

$$\langle 3 \quad 9 \quad 11 \quad 13 \rangle$$

$$\langle 4 \quad 6 \quad 10 \quad 15 \rangle$$

$$\langle 2 \quad 5 \quad 12 \quad 18 \rangle$$

$|E[l]| = 1 : \text{the one with the } \text{smallest} \text{ ending number}$

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset$ ,  $\forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow$   $\triangleright$  By extending each shorter subseq.
5:     the inc. subseq. of length  $l$  with the smallest ending number
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

$E[l] = \{all\ increasing\ subsequences\ of\ length\ l\}$

$|E[l]| = 1$  : the one with the **smallest** ending number

$E[l] = \{ \text{all increasing subsequences of length } l \}$

$|E[l]| = 1$  : the one with the **smallest** ending number

$$E[3] = \{\langle 2 \ 8 \ 15 \rangle\}$$

$$E[4] = \{\langle 4 \ 6 \ 11 \ 13 \rangle\}$$

$$E[5] = \{\langle 3 \ 9 \ 11 \ 12 \ 13 \rangle\}$$

$E[l] = \{ \text{all increasing subsequences of length } l \}$

$|E[l]| = 1$  : the one with the **smallest** ending number

$$E[3] = \{\langle 2 \ 8 \ 15 \rangle\}$$

$$E[4] = \{\langle 4 \ 6 \ 11 \ 13 \rangle\}$$

$$E[5] = \{\langle 3 \ 9 \ 11 \ 12 \ 13 \rangle\}$$

$\forall i < j : \text{the ending number of } E[i] < \text{the ending number of } E[j]$

---

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset$ ,  $\forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow$   $\triangleright$  By extending each shorter subseq.
5:     the inc. subseq. of length  $l$  with the smallest ending number
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset$ ,  $\forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow \triangleright$  Extending only one subseq using binary
   search
5:     the inc. subseq. of length  $l$  with the smallest ending number
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \emptyset$ ,  $\forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $\forall 1 \leq l \leq i : E[l] \leftarrow$   $\triangleright$  Extending only one subseq using binary search
5:     the inc. subseq. of length  $l$  with the smallest ending number
6:   return  $\max \{l \mid E[l] \neq \emptyset\}$ 
```

---

$$2, 5, 10, 16, 26 \quad \Leftarrow 12$$

# *Extending*

*Extending*  
*by looking at only the ending number of the inc. subseq*

## *Extending*

*by looking at only the ending number of the inc. subseq*

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $E[l] \leftarrow \triangleright$  Extending only one subseq using binary search
5:       the smallest ending number for the inc. subseq. of length  $l$ 
6:   return
```

---

## *Extending*

*by looking at only the ending number of the inc. subseq*

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \infty$    $\forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $E[l] \leftarrow$             $\triangleright$  Extending only one subseq using binary search
5:     the smallest ending number for the inc. subseq. of length  $l$ 
6:   return
```

---

## *Extending*

*by looking at only the ending number of the inc. subseq*

---

```
1: procedure LIS( $n$ )
2:    $E[l] \leftarrow \infty \quad \forall 1 \leq i \leq n$ 
3:   for  $i \leftarrow 1 \uparrow n$  do
4:      $E[l] \leftarrow \dots \triangleright$  Extending only one subseq using binary search
      the smallest ending number for the inc. subseq. of length  $l$ 
5:
6:   return  $\max \{l \mid E[l] < \infty\}$ 
```

---

# Matrix-chain Multiplication



$m[i, j]$  : min cost to compute the matrix  $A_i \cdots A_j$

$m[1, n]$

$m[i, j] : \min$  cost to compute the matrix  $A_i \cdots A_j$

$m[1, n]$

*Where is the last parentheses?*

$m[i, j] : \min$  cost to compute the matrix  $A_i \cdots A_j$

$m[1, n]$

*Where is the last parentheses?*

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

$m[i, j] : \min$  cost to compute the matrix  $A_i \cdots A_j$

$m[1, n]$

*Where is the last parentheses?*

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

$$m[i, i] = 0$$

$m[i, j] : \min$  cost to compute the matrix  $A_i \cdots A_j$

$m[1, n]$

*Where is the last parentheses?*

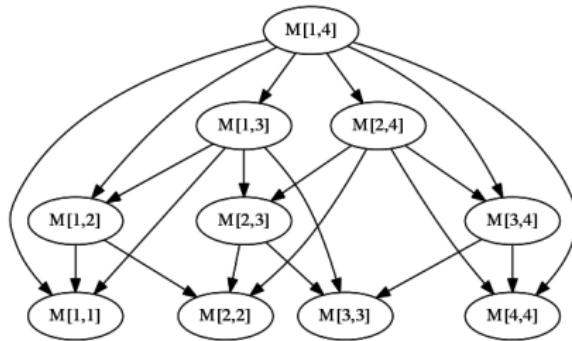
$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j)$$

$$m[i, i] = 0$$

$$O(n^3) = \Theta(n^2) \cdot O(n)$$

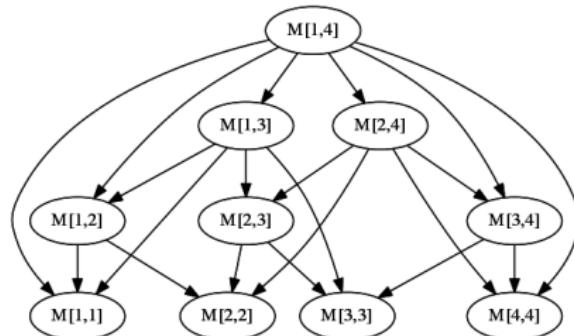
## Subproblem Graph for Matrix-chain Multiplication (Problem 15.2-4)

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



## Subproblem Graph for Matrix-chain Multiplication (Problem 15.2-4)

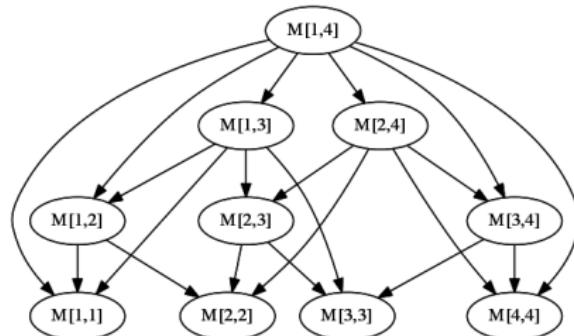
$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j)$$



$$\left| \{(i, j) | 1 \leq i \leq j \leq n\} \right| = \frac{n(n + 1)}{2}$$

## Subproblem Graph for Matrix-chain Multiplication (Problem 15.2-4)

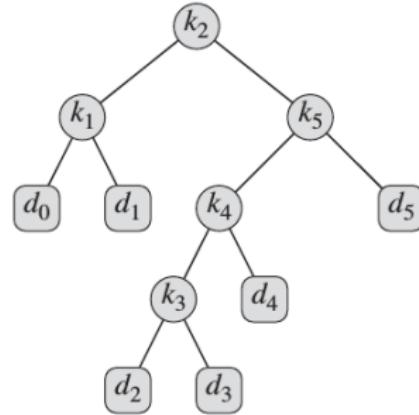
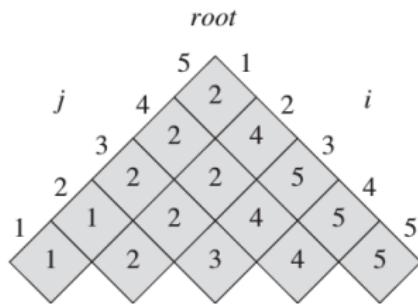
$$m[i, j] = \min_{i \leq k < j} (m[i, k] + m[k + 1, j] + p_{i-1} p_k p_j)$$



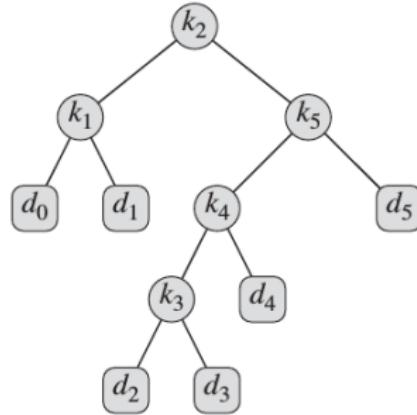
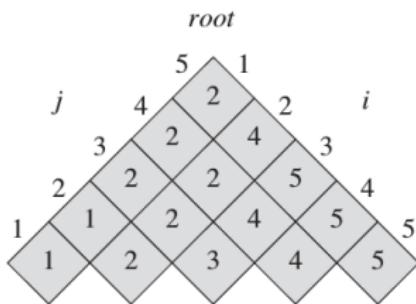
$$\left| \{(i, j) | 1 \leq i \leq j \leq n\} \right| = \frac{n(n+1)}{2}$$

$$\sum_{1 \leq i \leq j \leq n} 2(j - i) = \frac{n^3 - n}{3}$$

## CONSTRUCT-OPTIMAL-BST(root) (Problem 15.5-1)



## CONSTRUCT-OPTIMAL-BST(root) (Problem 15.5-1)



*What about  $d_0, d_1, \dots, d_n$ ?*

*How to distinguish left from right?*

## CONSTRUCT-OPTIMAL-BST(root) (Problem 15.5-1)

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

## CONSTRUCT-OPTIMAL-BST(root) (Problem 15.5-1)

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

---

```
1: procedure CONSTRUCT-OPTIMAL-BST(root, i, j)
2:   if j = i − 1 then
3:     return node with di-1
4:   rn ← node with key kroot[i,j]
5:   rn.left ← CONSTRUCT-OPTIMAL-BST(root, i, root[i,j] − 1)
6:   rn.right ← CONSTRUCT-OPTIMAL-BST(root, root[i,j] + 1, j)
7:   return rn
```

---

## CONSTRUCT-OPTIMAL-BST(root) (Problem 15.5-1)

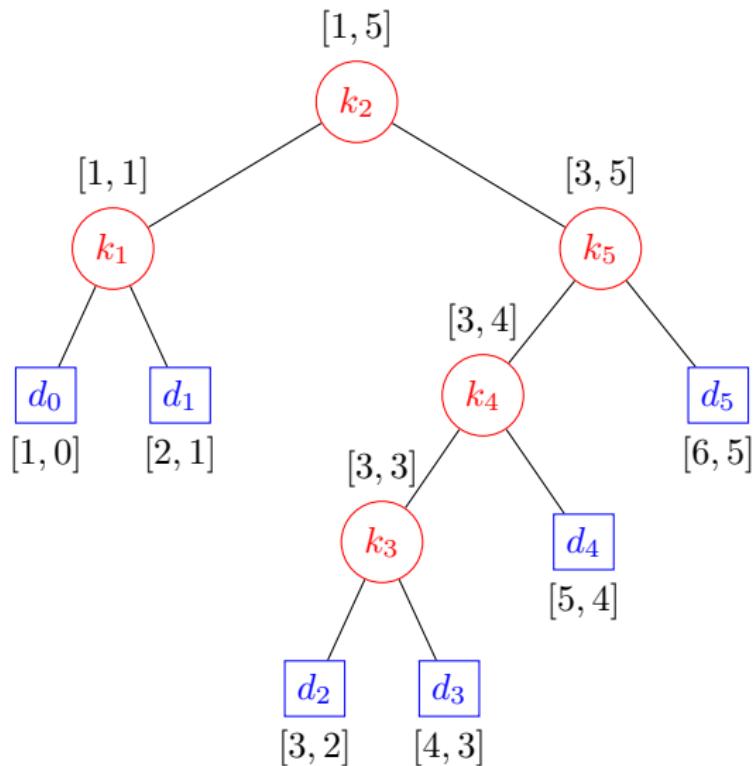
$$e[i, j] = \begin{cases} q_{i-1} & \text{if } j = i - 1, \\ \min_{i \leq r \leq j} \{e[i, r - 1] + e[r + 1, j] + w(i, j)\} & \text{if } i \leq j. \end{cases}$$

---

```
1: procedure CONSTRUCT-OPTIMAL-BST(root, i, j)
2:   if j = i − 1 then
3:     return node with di-1
4:   rn ← node with key kroot[i,j]
5:   rn.left ← CONSTRUCT-OPTIMAL-BST(root, i, root[i,j] − 1)
6:   rn.right ← CONSTRUCT-OPTIMAL-BST(root, root[i,j] + 1, j)
7:   return rn
```

---

CONSTRUCT-OPTIMAL-BST(*root*, 1, *n*)

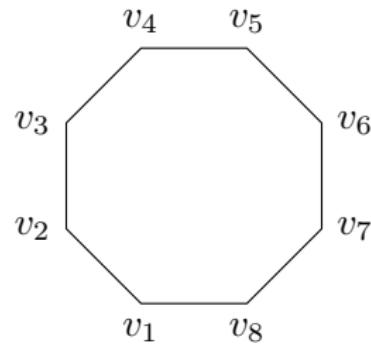


---

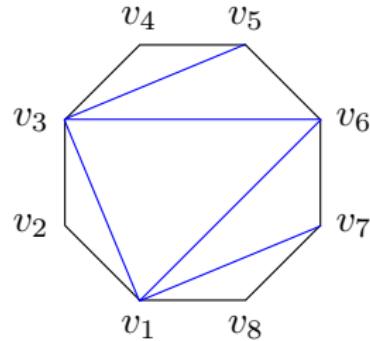
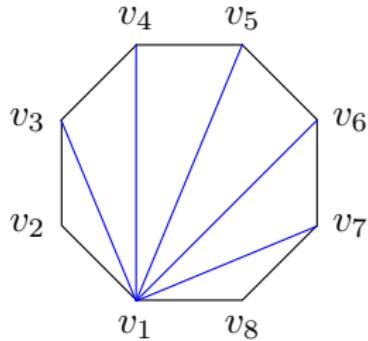
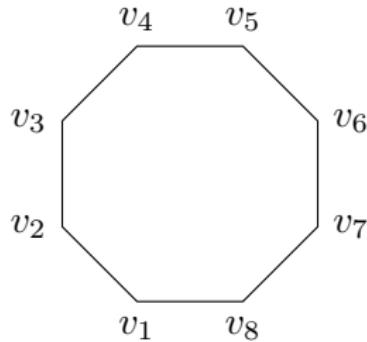
```
1: procedure CONSTRUCT-OPTIMAL-BST(root, i, j, r)    ▷ r : parent
2:   if r = i then
3:     print "di-1 is the left child of kr"    return node with di-1
4:   if r = j then
5:     print "di-1 is the right child of kr"   return node with di-1
6:   r' ← root[i, j]      rn ← node with key kr'
7:   if r = 0 then
8:     print "kr' is the root"
9:   else if j < r then
10:    print "kr' is the left child of kr"
11:   else if i > r then
12:     print "kr' is the right child of kr"
13:   rn.left ← CONSTRUCT-OPTIMAL-BST(root, i, r' - 1, r')
14:   rn.right ← CONSTRUCT-OPTIMAL-BST(root, r' + 1, j, r')
15:   return rn
```

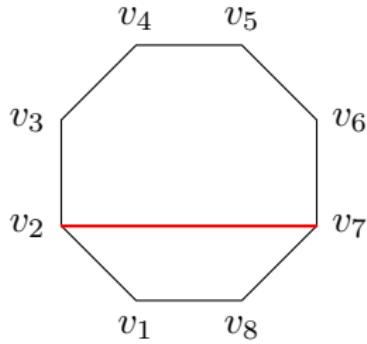
---

## Minimum Weight Triangulation

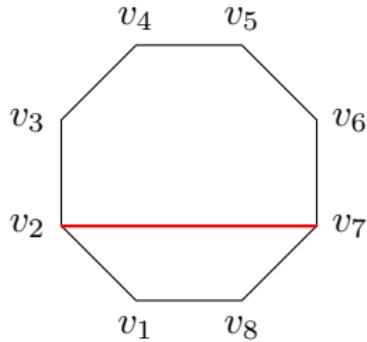


## Minimum Weight Triangulation



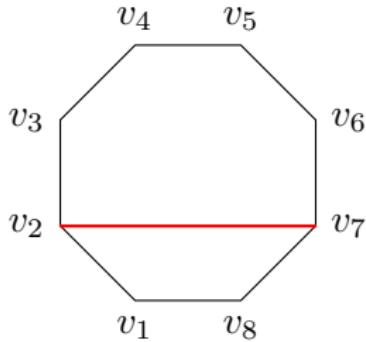


$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), } \textcolor{red}{\text{clockwise}}$



$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), clockwise}$

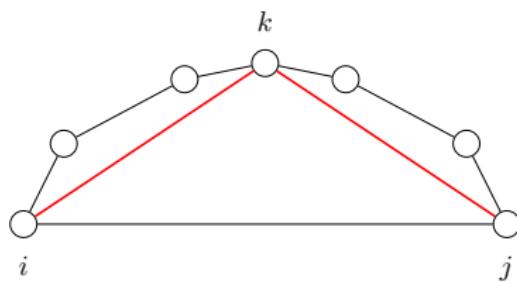
$$T(i, j) = \min_{\substack{i \leq k < l-1 \leq j \\ (k, l) \neq (i, j)}} \left( T[k, l] + T[?, ?] + d_{ij} \right)$$

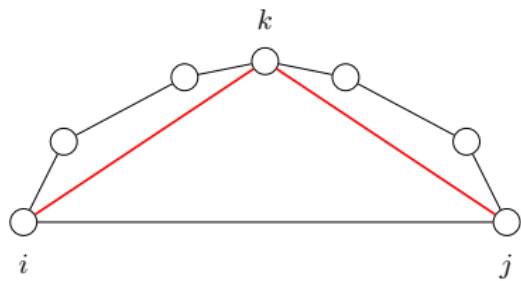


$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), clockwise}$

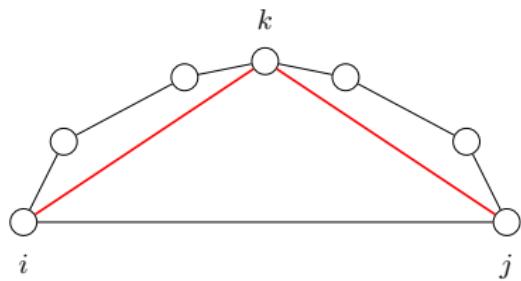
$$T(i, j) = \min_{\substack{i \leq k < l-1 \leq j \\ (k, l) \neq (i, j)}} \left( T[k, l] + T[?, ?] + d_{ij} \right)$$

$$O(n^4) = O(n^2) \cdot O(n^2)$$



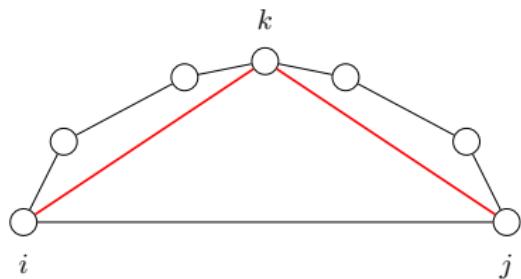


Which vertex  $k$  to pair with  $(i, j)$ ?



Which vertex  $k$  to pair with  $(i, j)$ ?

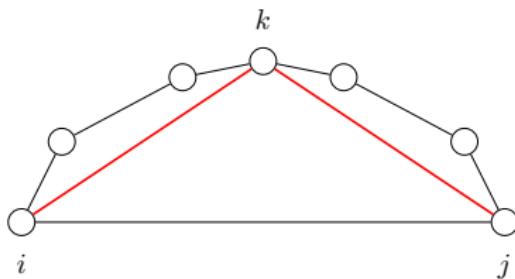
$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), } i < j$



Which vertex  $k$  to pair with  $(i, j)$ ?

$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), } i < j$

$$T(i, j) = \min_{i < k < j} (T(i, k) + T(k, j) + d_{ik} + d_{kj})$$

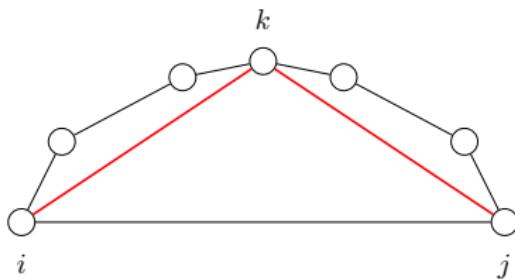


Which vertex  $k$  to pair with  $(i, j)$ ?

$T(i, j) : \min \text{ cost of triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), } i < j$

$$T(i, j) = \min_{i < k < j} (T(i, k) + T(k, j) + d_{ik} + d_{kj})$$

$$T[i, i+1] = 0, \quad 1 \leq i \leq n-1$$



Which vertex  $k$  to pair with  $(i, j)$ ?

$T(i, j) : \min \text{ cost of } \text{triangulating } v_i \cdots v_j \text{ (with } v_j - v_i\text{), } i < j$

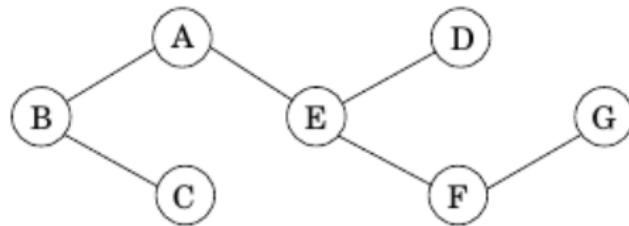
$$T(i, j) = \min_{i < k < j} (T(i, k) + T(k, j) + d_{ik} + d_{kj})$$

$$T[i, i+1] = 0, \quad 1 \leq i \leq n-1$$

$$O(n^3) = O(n^2) \cdot O(n)$$

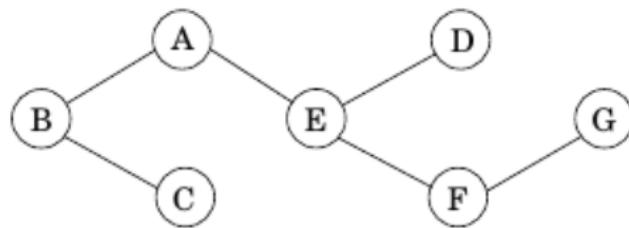
## Minimum Vertex Cover on Trees (Additional Problem)

- ▶ Undirected tree  $T = (V, E)$ ; *No designated root*
- ▶ Compute (the size of) a minimum vertex cover (MVC) of  $T$



## Minimum Vertex Cover on Trees (Additional Problem)

- ▶ Undirected tree  $T = (V, E)$ ; *No designated root*
- ▶ Compute (the size of) a minimum vertex cover (MVC) of  $T$



Rooted  $T$  at any node  $r$ .

$I(u)$  : the size of an MVC of subtree  $T_u$  rooted at  $u$

$$I(r)$$

$I(u)$  : the size of an MVC of subtree  $T_u$  rooted at  $u$

$$I(r)$$

Is  $u$  in  $MVC[u]$ ?

$I(u)$  : the size of an MVC of subtree  $T_u$  rooted at  $u$

$$I(r)$$

Is  $u$  in  $MVC[u]$ ?

$$I(u) = \min \left\{ \underbrace{1 + \sum_{v: \text{ children of } u} I(v), \# \text{ children of } u}_{\in} + \underbrace{\sum_{v: \text{ grandchildren of } u} I(v)}_{\notin} \right\}$$

$I(u)$  : the size of an MVC of subtree  $T_u$  rooted at  $u$

$$I(r)$$

Is  $u$  in  $MVC[u]$ ?

$$I(u) = \min \left\{ 1 + \underbrace{\sum_{v: \text{ children of } u} I(v), \# \text{ children of } u}_{\in} + \underbrace{\sum_{v: \text{ grandchildren of } u} I(v)}_{\notin} \right\}$$

$I(u) = 0$ , if  $u$  is a leave





*There is an MVC which contains no leaves.*



*There is an MVC which contains no leaves.*

Proof: Exchange Argument



*There is an MVC which contains no leaves.*

Proof: Inductive Exchange Argument

## Algorithms that use dynamic programming

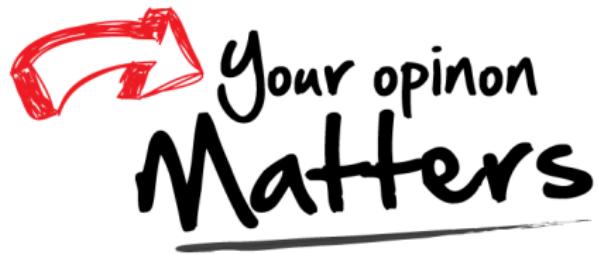
[edit | edit source]



This section does not cite any sources. Please help improve this section by adding citations to reliable sources. Unsourced material may be challenged or removed.

- Recurrent solutions to lattice models for protein-DNA binding
- Backward induction as a solution method for finite-horizon discrete-time dynamic optimization problems
- Method of undetermined coefficients can be used to solve the Bellman equation in infinite-horizon, discrete-time, discounted, time-invariant dynamic optimization problems
- Many string algorithms including longest common subsequence, longest increasing subsequence, longest common substring, Levenshtein distance (edit distance)
- Many algorithmic problems on graphs can be solved efficiently for graphs of bounded treewidth or bounded clique-width by using dynamic programming on a tree decomposition of the graph.
- The Cocke–Younger–Kasami (CYK) algorithm which determines whether and how a given string can be generated by a given context-free grammar
- Knuth's word wrapping algorithm that minimizes raggedness when word wrapping text
- The use of transposition tables and refutation tables in computer chess
- The Viterbi algorithm (used for hidden Markov models)
- The Earley algorithm (a type of chart parser)
- The Needleman–Wunsch algorithm and other algorithms used in bioinformatics, including sequence alignment, structural alignment, RNA structure prediction
- Floyd's all-pairs shortest path algorithm
- Optimizing the order for chain matrix multiplication
- Pseudo-polynomial time algorithms for the subset sum, knapsack and partition problems
- The dynamic time warping algorithm for computing the global distance between two time series
- The Selinger (a.k.a. System R) algorithm for relational database query optimization
- De Boor algorithm for evaluating B-spline curves
- Duckworth–Lewis method for resolving the problem when games of cricket are interrupted
- The value iteration method for solving Markov decision processes
- Some graphic image edge following selection methods such as the "magnet" selection tool in Photoshop
- Some methods for solving interval scheduling problems
- Some methods for solving the travelling salesman problem, either exactly (in exponential time) or approximately (e.g. via the bitonic tour)
- Recursive least squares method
- Beat tracking in music information retrieval
- Adaptive-critic training strategy for artificial neural networks
- Stereo algorithms for solving the correspondence problem used in stereo vision
- Seam carving (content-aware image resizing)
- The Bellman–Ford algorithm for finding the shortest distance in a graph
- Some approximate solution methods for the linear search problem
- Kadane's algorithm for the maximum subarray problem





Office 302

Mailbox: H016

hfwei@nju.edu.cn