3-1 Dynamic Programming

(Part I: Examples)

Hengfeng Wei

hfwei@nju.edu.cn

September 17, 2018

Taolu

Steps for Applying DP:

(I) Define subproblems Define subproblems

(II) Set the goal

(III) Identify the recurrence 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: $x_1, x_2, \ldots, x_n$ (array, sequence, string)

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

#: $\Theta(n)$

Examples:

- Rod cutting
- Maximum-sum subarray
- Longest increasing subsequence
- Printing Neatly

2D Subproblems

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

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

#: $\Theta(mn)$

Examples: Edit distance, Longest common subsequence

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

Subproblems: $x_i, \ldots, x_j$

#: $\Theta(n^2)$

Examples: Matrix chain multiplication, Optimal BST

3D Subproblems

- Floyd-Warshall algorithm

$$\mathsf{d}(i,j,k) = \min\Big(\mathsf{d}(i,j,k-1), \mathsf{d}(i,k,k-1) + \mathsf{d}(k,j,k-1)\Big)$$

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 . . .

How to identify the recurrence?

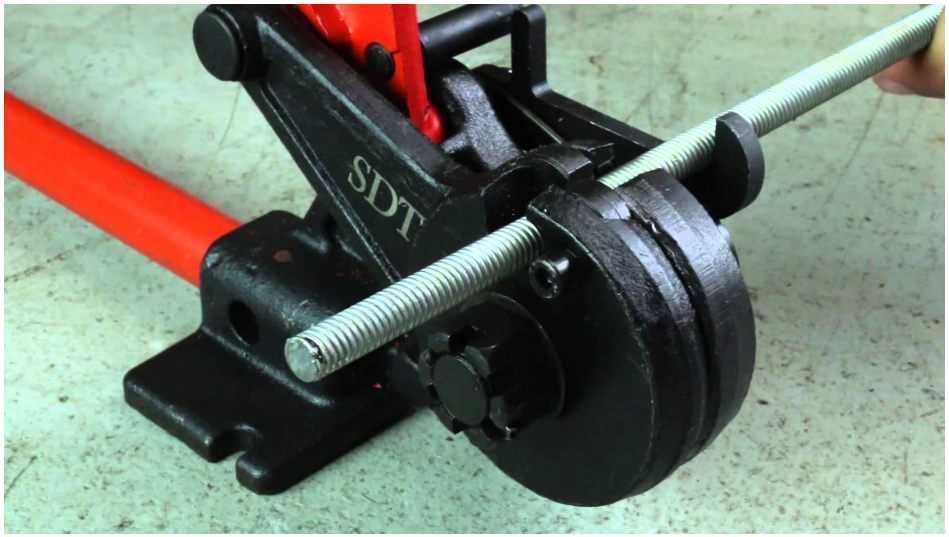
GUESS

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	$\cdots$
price p_i	1	5	8	9	10	$\cdots$

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

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

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

$$R(n)$$

Where is the leftmost cut?

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

$$R(0) = 0$$

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

Rod Cutting Problem (Problem 15.1-3)

Each cut incurs a fixed cost of c.

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

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

Printing Neatly (Problem 15-4)

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

Line width M

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

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.

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

TEX 3.14159265 $\sim \pi$

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

$C(i)$: 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

How many words to place on the first line?

$$C(i) = \min_{\substack{0 < k \le (n-i+1) \\ extra[i, i+k-1] \ge 0}} \Big(extra[i, i+k-1]\Big)^3 + C(i+k)$$

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

$$O(nM)$$

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

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

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

— *Tianyun Zhang*

Proof: Exchange Argument

Longest Increasing Subsequence (Problem 15.4-5)

$$A[1 \ldots n]$$

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

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

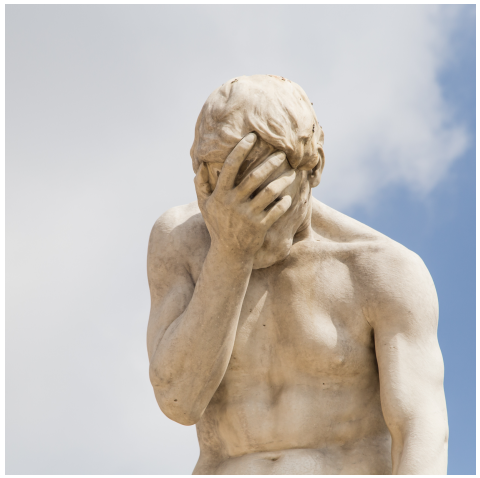
$$5,\ 2,\ 8,\ 6,\ 3,\ 6,\ 9,\ 7$$

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

$L(n)$

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

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



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

$$\max_i L(i)$$

What is the previous element?

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

$$L(1) = 1$$

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

$$\mathsf{LIS(A)} = \mathsf{LCS}\Big(A, \textsc{Sort}(A)\Big)$$

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

Longest Increasing Subsequence (Problem 15.4-6)

$$O(n \log n)$$

Answer by Eugene Yarovoi @ Quora

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

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

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

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

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

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

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

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: $\forall 1 \leq l \leq i : E[l] \leftarrow$ ▷ *By extending each shorter subseq.*

5: *the inc. subseq. of length* l *with the smallest ending number*

6: ***return*** $\max\{l \mid E[l] \neq \emptyset\}$

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

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

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

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

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

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

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: $\forall 1 \leq l \leq i : E[l] \leftarrow$ ▷ *By extending each shorter subseq.* ▷

Extending only one subseq using binary search

5: *the inc. subseq. of length l with the smallest ending number*

6: ***return*** $\max\{l \mid E[l] \neq \emptyset\}$

$$2,\ 5,\ 10,\ 16,\ 26 \qquad \Longleftarrow 12$$

Extending

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

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

Matrix-chain Multiplication

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

$$m[1,n]$$

Where is the last parentheses?

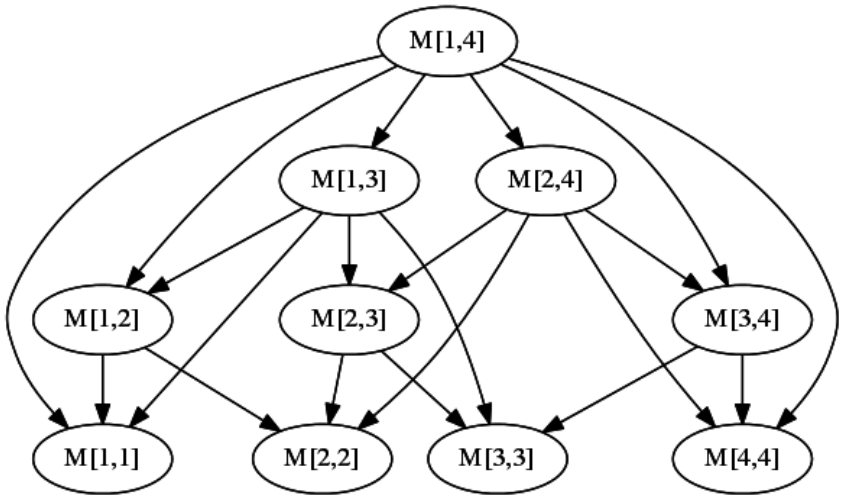
$$m[i,j] = \min_{i \le k < j} \left(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \right)$$

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

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

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

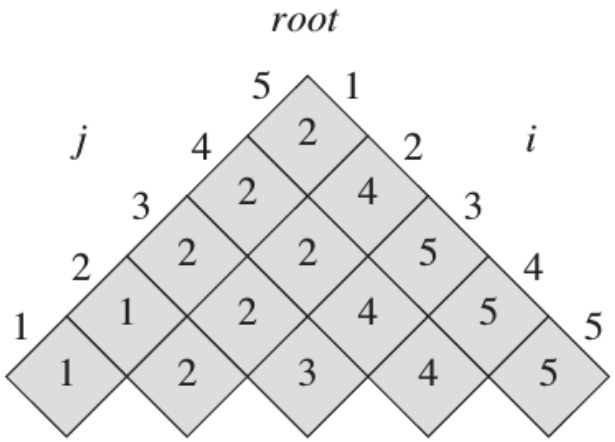
$$m[i,j] = \min_{i \le k < j} \Big(m[i,k] + m[k+1,j] + p_{i-1} p_k p_j \Big)$$

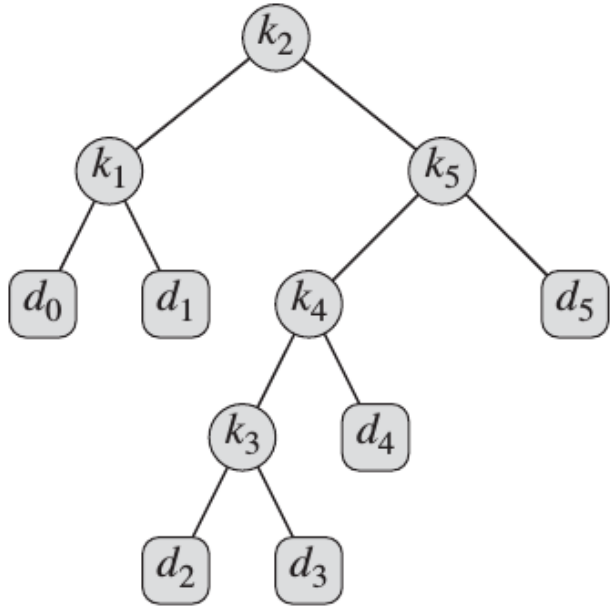


$$\Big| \{(i,j) | 1 \le i \le j \le n\} \Big| = \frac{n(n+1)}{2}$$

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

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





What about $d_0, d_1, \cdots, d_n$?

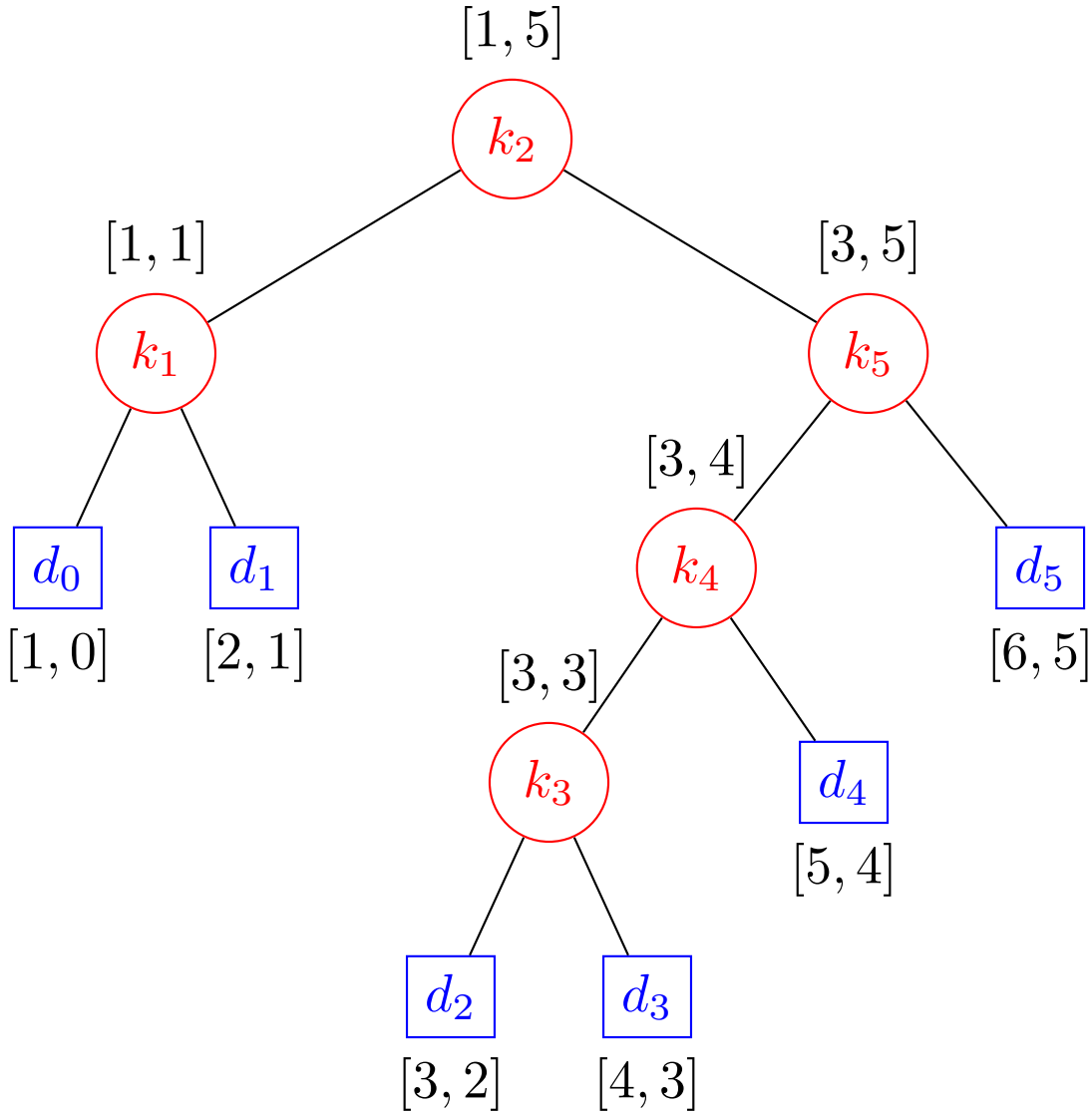
How to distinguish left from right?

Construct-Optimal-BST(root) (Problem 15.5-1)

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

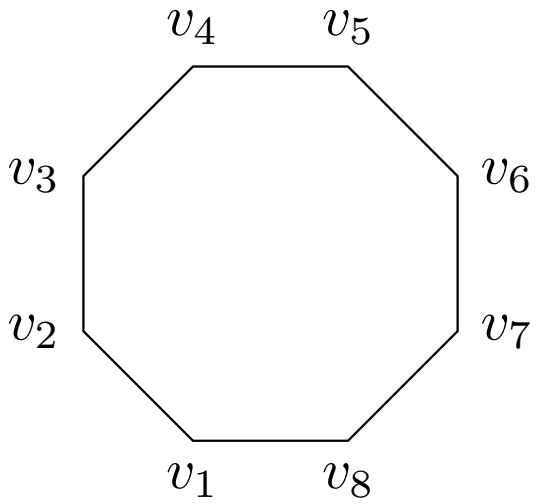
1: **procedure** Construct-Optimal-BST($root$, i, j)
2: **if** $j = i - 1$ **then**
3: **return** node with d_{i-1}
4: $rn \leftarrow$ node with key $k_{root[i,j]}$
5: $rn.left \leftarrow$ Construct-Optimal-BST$(root, i, root[i,j]-1)$
6: $rn.right \leftarrow$ Construct-Optimal-BST$(root, root[i,j]+1, j)$
7: **return** rn

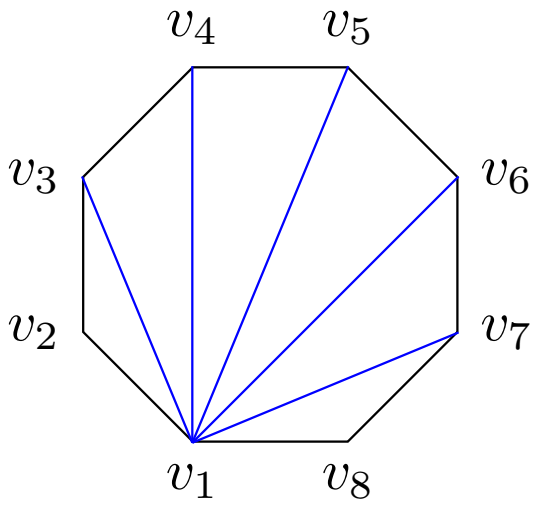
Construct-Optimal-BST$(root, 1, n)$

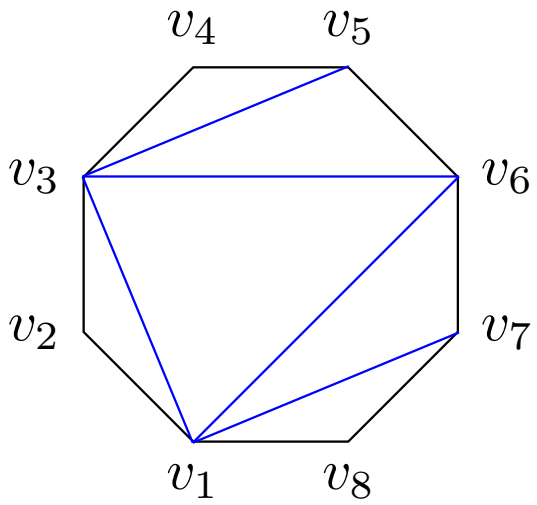

[1, 5]
k_2
[1, 1]
k_1
[3, 5]
k_5
[3, 4]
k_4
d_0
[1, 0]
d_1
[2, 1]
d_5
[6, 5]
[3, 3]
k_3
d_4
[5, 4]
d_2
[3, 2]
d_3
[4, 3]

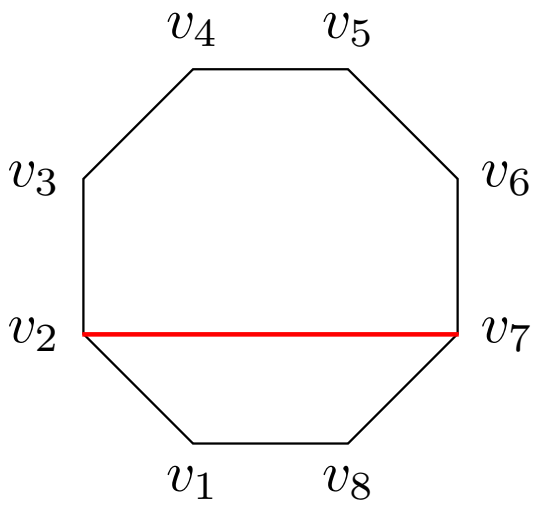
```
1:  procedure CONSTRUCT-OPTIMAL-BST(root, i, j, r)      ▷ r : parent
2:      if r = i then
3:          print "d_{i-1} is the left child of k_r"    return node with d_{i-1}
4:      if r = j then
5:          print "d_{i-1} is the right child of k_r"   return node with d_{i-1}
6:      r' ← root[i, j]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





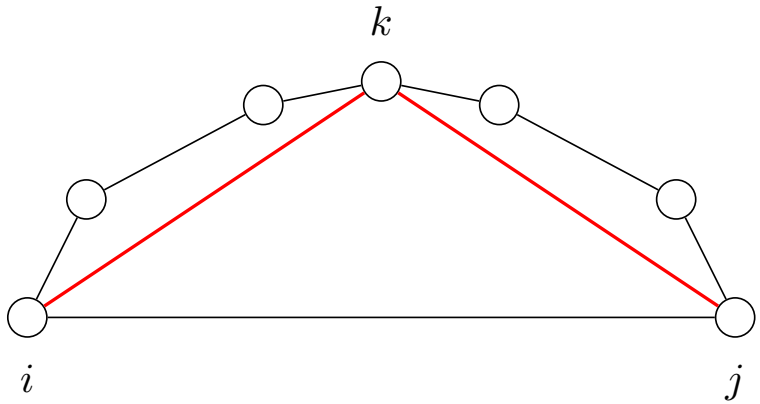




$T(i,j)$: min cost of triangulating $v_i \cdots v_j$ (with $v_j - v_i$), *clockwise*

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

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



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

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

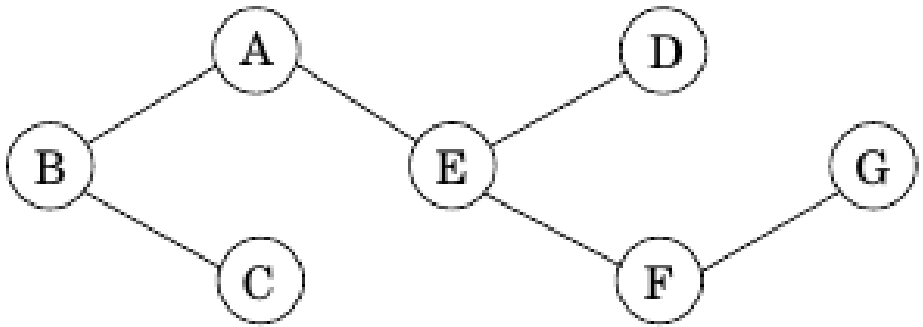
$$T(i, j) = \min_{i<k<j} \Big(T(i, k) + T(k, j) + d_{ik} + d_{kj} \Big)$$

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

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

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



Rooted T at any node r.

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

$$I(r)$$

Is u in MVC$[u]$?

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

$$I(u) = 0, \text{ if } u \text{ is a leave}$$

There is an MVC which contains no leaves.

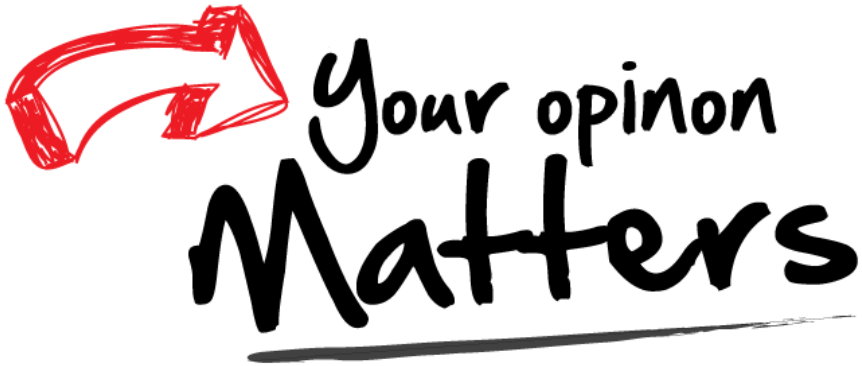
Proof: Inductive Exchange Argument

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 ar
how and when to remove this template message)

- 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


THANK YOU



Office 302

Mailbox: H016

hfwei@nju.edu.cn