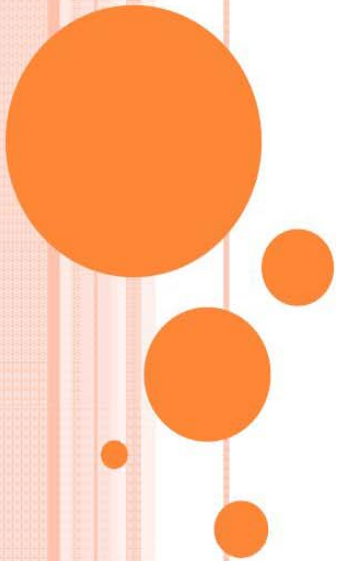
DYNAMIC PROGRAMMING



Dr. Zhenyu He

Autumn 2010

OUTLINE

- Introduction to Dynamic Programming
- Famous Examples
  - Matrix-chain Multiplication
  - Longest Common Subsequence
  - Triangle Decomposition of Convex
  - Polygon
  - The Optimal Binary Search Trees

Introduction to Dynamic Programming

Why?

What?

How?

WHY?

The problem of Divide-and-conquer

- Treat the subproblems independently;
- If they are dependent, we will calculate redundantly, which leads low efficiency.

Optimization Problem

- Given a group of constrains and the cost function, find *a* solution with *the* optimal value (min or max) in the solution space;
- Lots of the optimization can be divided into subproblems, which are dependent, so the solution of the subproblem can be reused.

WHY?

Those who cannot remember the past are doomed to repeat it.

-----George Santayana,
The life of Reason,
Book I: Introduction and
Reason in Common
Sense (1905)

What?

Dynamic Programming

- Divide into subproblem
- Solve each subproblem only once, store the solutions in a list, and access it when the solution is reused
- Bottom-up

WHAT?

Elements of dynamic programming

- Optimal substructure

  A problem exhibits *Optimal substructure* if an optimal solution to the problem is contained within its optimal solutions to subproblems.

- Overlapping subproblems

  When a recursive algorithm revisits the same problem over and over again, we say that the optimization problem has overlapping subproblems.

Optimal substructure

1) Show that a solution to the problem consists of making a choice.
2) Suppose the choices are known.
3) Determine which subproblems ensue.
4) Cut-and-paste.

How?

The step of dynamic programming:

- Characterize the structure of an optimal solution
- Recursively define the value of an optimal solution
- Compute the value of an optimal solution in a bottom-up fashion
- Construct an optimal solution from computed information

Matrix-chain Multiplication

PROBLEM DEFINITION

Inputs: Matrix chain $< A_1, A_2, \ldots, A_n >$

Outputs: $A_1 A_2 \ldots A_n$

If ***A*** is a $\boldsymbol{p \times q}$ matrix, and ***B*** a $\boldsymbol{q \times r}$ matrix, then normally we spend **O(pqr)** times to compute ***AB*.**

MOTIVATION

Matrix multiplication fulfill multiplication associativity.

Example:

$(A_1 A_2 A_3 A_4)$

$= (A_1 (A_2 (A_3 A_4)))$

$= ((A_1 A_2) (A_3 A_4))$

....

$= (((A_1 A_2) A_3) A_4)$

MULTIPLICATION ORDER

The complexity depends on the order

- Suppose A_1 a 10×100 matrix, A_2 a 100×5 matrix, A_3 a 5×50 matrix
- $T((A_1 A_2) A_3)$

  $=10 \times 100 \times 5 + 10 \times 5 \times 50 = 7500$
- $T(A_1 (A_2 A_3))$

  $= 100 \times 5 \times 50 + 10 \times 100 \times 50 = 750000$

SOLUTION SPACE

- Denote the number of different solutions as ***P(n),*** the recursive function of ***P(n)***

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

- ***P(n) is Catalan number***

$$P(n)=C(n-1)=\frac{1}{n}\binom{2n-2}{n-1}=\Omega(4^n/n^{3/2})$$

P(n) is so big to solute the problem through enumeration methods.

SOLUTION SPACE

- Denote the number of different solutions as ***P(n),*** the recursive function of ***P(n)***

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

P(n) is Catalan number

$$P(n)=C(n-1)=\frac{1}{n}\binom{2n-2}{n-1}=\Omega(4^n/n^{3/2})$$

Enumerating won't work

P(n) is so big to solute the problem through enumeration methods.

STEP 1: STRUCTURE OF OPTIMAL SOLUTION

- Denote the multiplication $A_i A_{i+1} \ldots A_j$ as A [i: j]
- Suppose we cut at k, and get a optimal order

$$(A_1 A_2 \ldots A_n) = ((A_1 \ldots A_k)(A_{k+1} \ldots A_n))$$

- A [1:n]=A [1:k] A [k+1:n]
- we can prove A [1:k] and A [k+1:n] are optimal order too

MULTIPLICATION ORDER

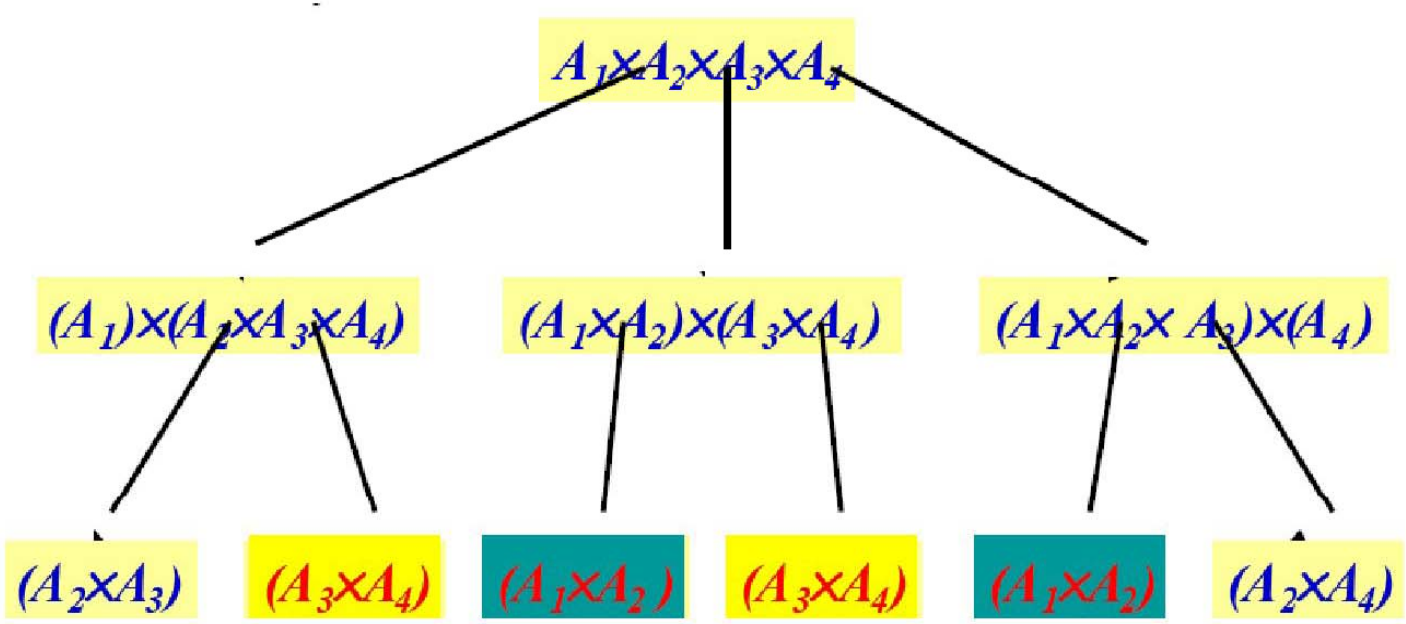
The complexity depends on the order

- Suppose A_1 a 10×100 matrix, A_2 a 100×5 matrix, A_3 a 5×50 matrix
- $T((A_1 A_2) A_3)$

  $=10 \times 100 \times 5 + 10 \times 5 \times 50=7500$
- $T(A_1 (A_2 A_3))$

  $= 100 \times 5 \times 50+10 \times 100 \times 50=750000$

OVERLAPPING



STEP 2: RECURSION

Denotation

m[i, j] ----the minimal cost(times) to calculate A [i:j]

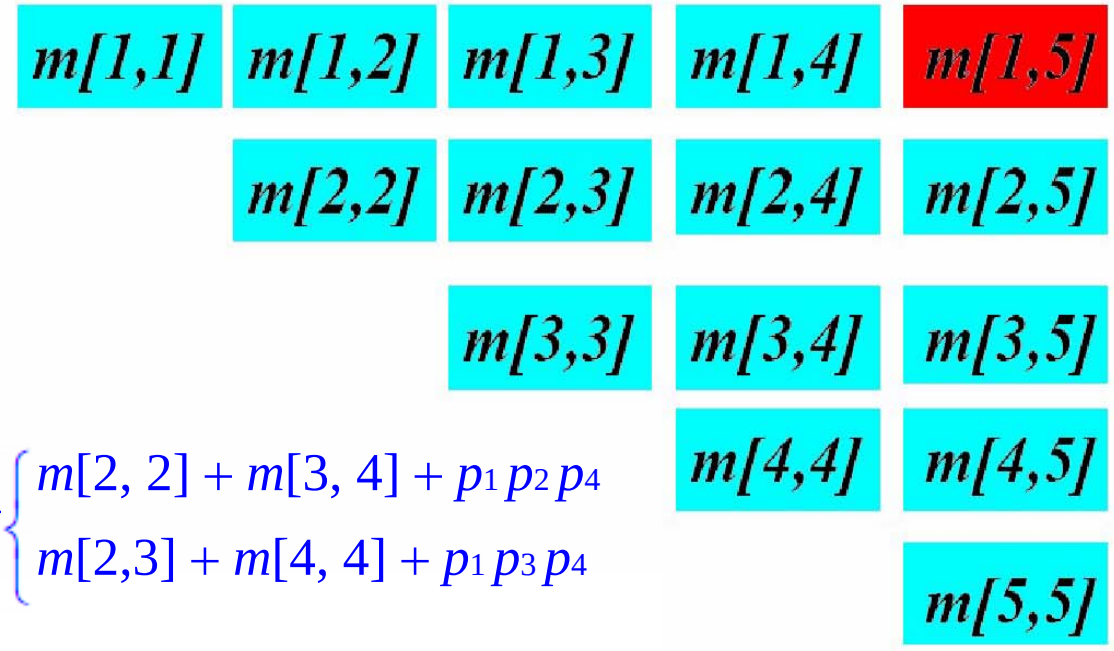
m[1, n] ----the minimal cost(times) to calculate A [1:n]

Recursion of cost

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

EXAMPLE

$$m[1,5] = \min_{1 \delta k <5} \{m[1, k\ ] + m[k + 1,5] + p_0 p_k p_5\ \}$$



$$m[2, 4] = \min \begin{cases} m[2, 2] + m[3, 4] + p_1 p_2 p_4 \\ m[2,3] + m[4, 4] + p_1 p_3 p_4 \end{cases}$$

STEP 3 BOTTOM-UP

```
public static void matrixChain(int [] p, int [][] m, int [][] s)
  {
    int n=p.length-1;
    for (int i = 1; i <= n; i++) m[i][i] = 0;
    for (int r = 2; r <= n; r++)
      for (int i = 1; i <= n - r+1; i++) {
        int j=i+r-1;
        m[i][j] = m[i+1][j]+ p[i-1]*p[i]*p[j];
        s[i][j] = i;
        for (int k = i+1; k < j; k++) {
          int t = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
          if (t < m[i][j]) {
            m[i][j] = t;
            s[i][j] = k;}
          }
        }
  }
```

STEP 3 BOTTOM-UP

```
public static voi
  {
    int n=p.lengt
    for (int i = 1;
    for (int r = 2
      for (int i = 1; i <= n - r+1; i++) {
        int j=i+r-1;
        m[i][j] = m[i+1][j]+ p[i-1]*p[i]*p[j];
        s[i][j] = i;
        for (int k = i+1; k < j; k++) {
          int t = m[i][k] + m[k+1][j] + p[i-1]*p[k]*p[j];
          if (t < m[i][j]) {
            m[i][j] = t;
            s[i][j] = k;}
          }
        }
  }
```

complexity:

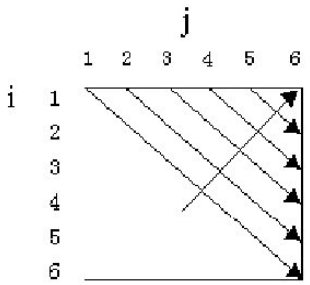
It is based on the ternary recurrence of ***r, i, k***。 So the time complexity is $O(n^3)$ while the space complexity is $O(n^2)$。

STEP 4 CONSTRUCTING

```
Print-Optimal-Parens(s, i, j)
IF j=i
THEN  Print "A"i;
ELSE   Print "("
          Print-Optimal-Parens(s, i, s[i, j])
          Print-Optimal-Parens(s, s[i, j]+1, j)
          Print  ")"
```

EXAMPLE

A1	A2	A3	A4	A5	A6
30×35	35×15	15×5	5×10	10×20	20×25



(a) 计算次序

i \ j	1	2	3	4	5	6
1	0	15750	7875	9375	11875	15125
2		0	2625	4375	7125	10500
3			0	750	2500	5375
4				0	1000	3500
5					0	5000
6						0

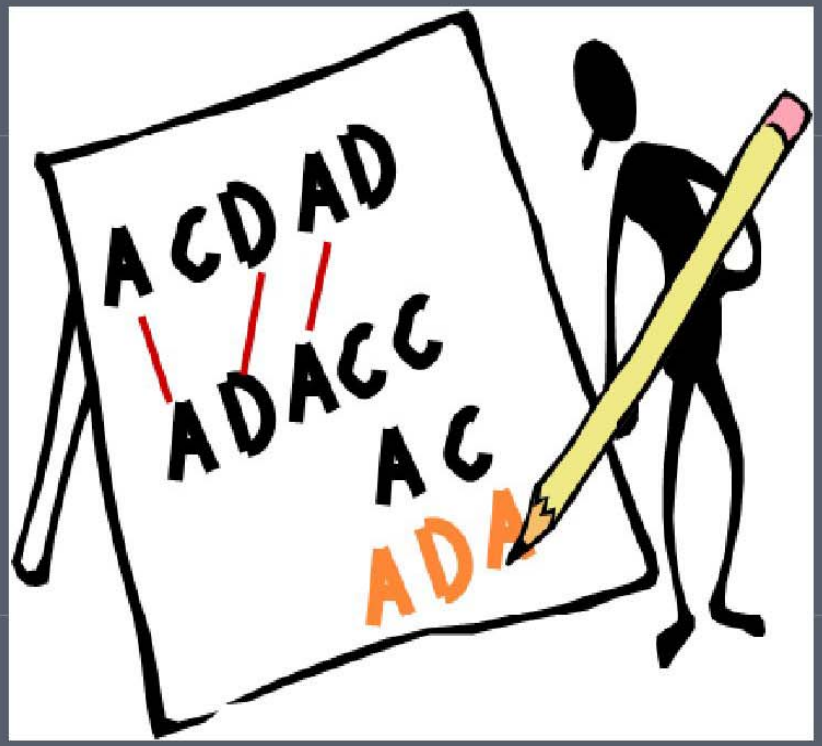
(b) m[i][j]

i \ j	1	2	3	4	5	6
1	0	1	1	3	3	3
2		0	2	3	3	3
3			0	3	3	3
4				0	4	5
5					0	5
6						0

(c) s[i][j]

$$m[2][5] = \min\begin{cases} m[2][2] + m[3][5] + p_1 p_2 p_5 = 0 + 2500 + 35 \times 15 \times 20 = 13000 \\ m[2][3] + m[4][5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \times 5 \times 20 = 7125 \\ m[2][4] + m[5][5] + p_1 p_4 p_5 = 4375 + 0 + 35 \times 10 \times 20 = 11375 \end{cases}$$

Longest Common Sequence



LONGEST COMMON SUSEQUENCE (LCS)

- Definition
- Characterizing LCS
- Recursive solution
- Bottom-up to computing the length of LCS
- Construct Optimal solution

DEFINITION

Subsequence

Definition:

A sequence $Z =< z_1 , z_2 ,..., z_k >$ **is a subsequence of** $X =< x_1 , x_2 ,..., x_m >$ **if there exists a strictly increasing sequence** $< i_1 , i_2 ,..., i_k >$ **of indices of** X **such that for all** $j=1,2,\ldots,k$, **we have** $x_{i_j} = z_j$

Example:

$Z=<A,B,C,D>$ **is a subsequence of** $X=<A,C,B,C,A,D>$ **with corresponding index sequence <1,3,4,6>.**

Common subsequence

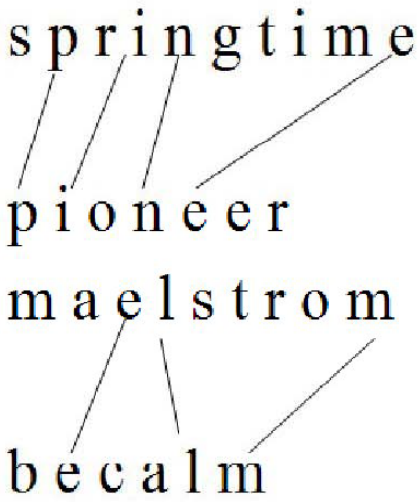
We say that a sequence *Z* is a common subsequence of *X* and *Y* if *Z* is a subsequence of both *X* and *Y*.

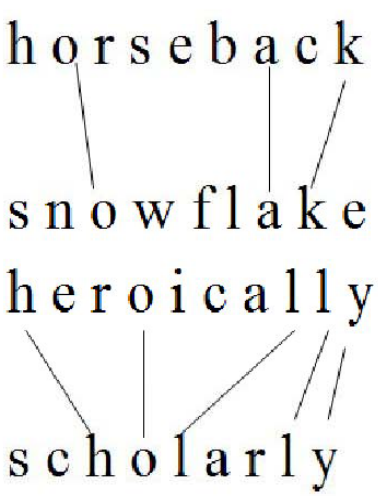
The longest-common-subsequence problem(LCS):

Input: $X =< x_1 , x_2 ,..., x_m >$ $Y =< y_1 , y_2 ,..., y_n >$

Output: the common subsequence *Z* that max(|Z|)

EXAMPLE


springtime
pioneer
maelstrom
becalm


horseback
snowflake
heroically
scholarly

EXAMPLE

Subsequence

Brute-force algorithm:
For every subsequence of X, check whether it's a subsequence of Y.
Time: $\Theta(n2^m)$

***Z*=<*A*,*B*,*C*,*D*> is a subsequence of *X*=<*A*,*C*,*B*,*C*,*A*,*D*> with corresponding index sequence <1,3,4,6>.**

Common subsequence

We say that a sequence *Z* is a common subsequence of *X* and *Y* if *Z* is a subsequence of both *X* and *Y*.

The longest-common-subsequence problem(LCS):

Input: $X =< x_1, x_2, ..., x_m >$ $Y =< y_1, y_2, ..., y_n >$
Output: the common subsequence *Z* that max(|Z|)

CHARACTERIZING LCS

The *ith prefix* of *X*----- X_i

Given $X = < x_1 , x_2 , \ldots , x_m >$, the *ith* prefix of *X* is

$$X_i = < x_1 , x_2 , \ldots , x_i >$$

Example

X=(A, B, D, C, A)

X1=(A), X2=(A, B), X3=(A, B, D)

CHARACTERIZING LCS

Optimal substructure of an LCS

Let $X = \langle x_1, x_2, ..., x_m \rangle$, $Y = \langle y_1, y_2, ..., y_n \rangle$ be sequences, and let $Z = \langle z_1, z_2, ..., z_k \rangle$ be any LCS of X and Y.

- 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}
- 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 $x_m \neq y_n$, then $z_k \neq y_n$ implies that Z is an LCS of X and Y_{n-1}

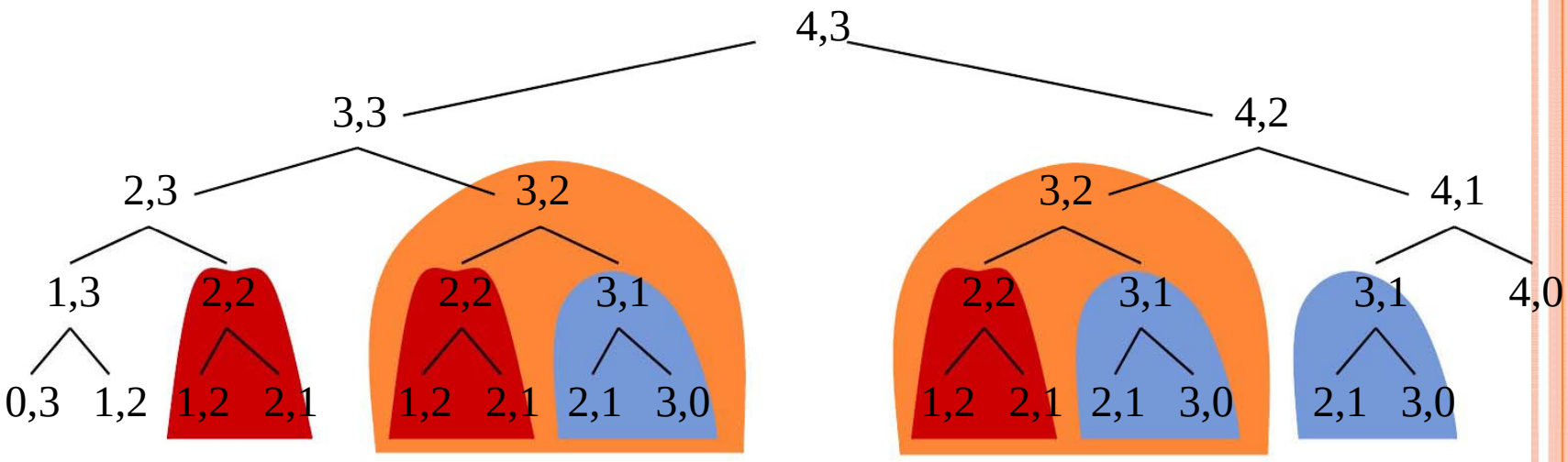
RECURSIVE SOLUTION

- Define ***c[i, j] as the length of an LCS of the sequences*** X_i and Y_j
- ***The recursive formula is***

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

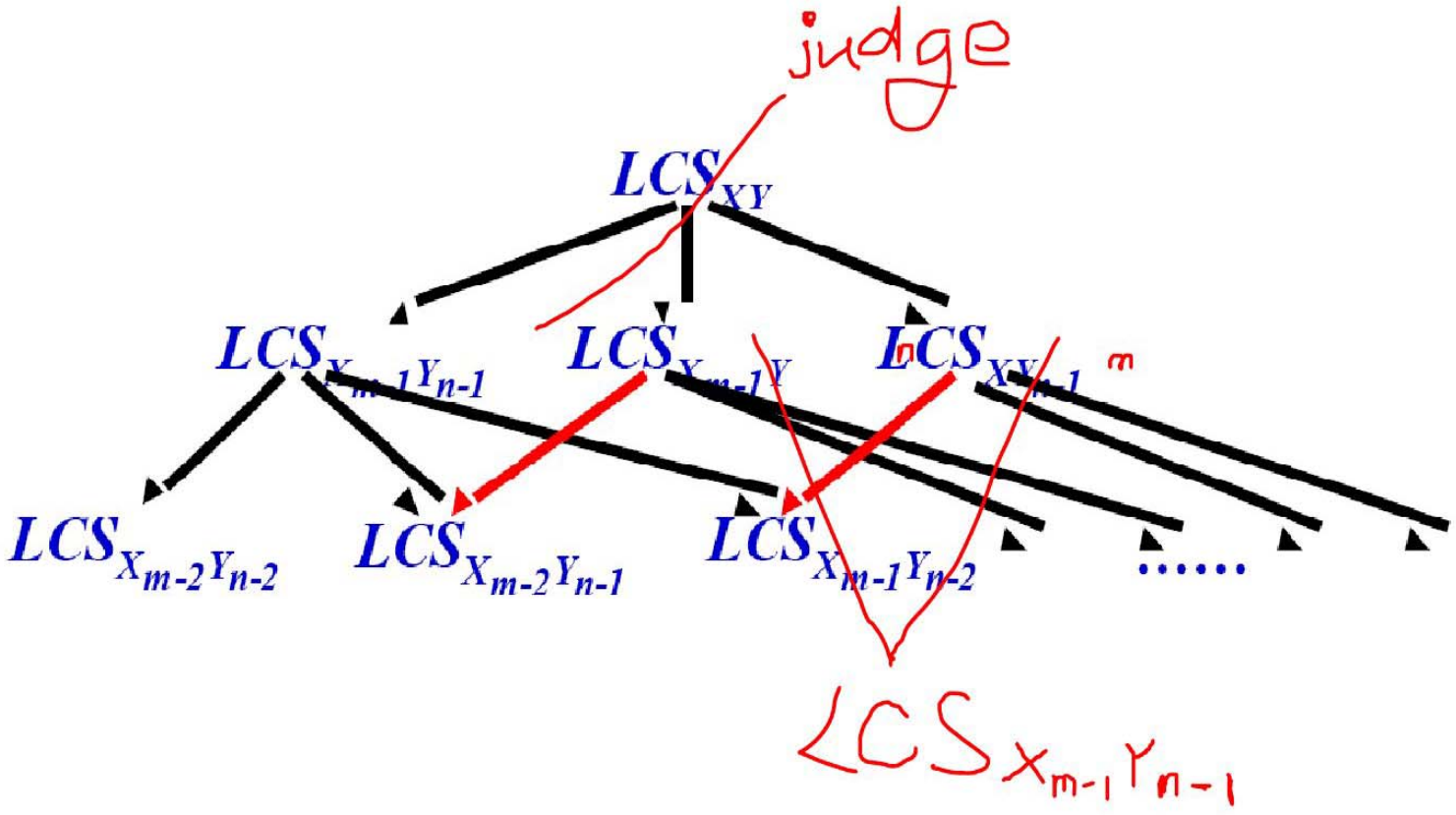
RECURSIVE SOLUTION

We could write a recursive algorithm based on this formulation.
Try with “bozo”, “bat”.



Lots of repeated subproblems.
Instead of recomputing, store in a table.

OVERLAPPING



COMPUTING THE LENGTH OF AN LCS

```
LCS-length(X, Y)
m←length(X);  n←length(Y);
For i←1 To m Do C[i,0]←0;
For j←1 To n Do C[0,j]←0;
For i←1 To m Do
    For j←1 To n Do
        If X_i = Y_j
        Then C[i,j]←C[i-1,j-1]+1;  B[i,j]←"↖";
        Else If C[i-1,j]≥C[i,j-1]
                Then C[i,j]←C[i-1,j]; B[i,j]←"↑";
                Else C[i,j]←C[i,j-1]; B[i,j]←"←";
Return C and B.
```

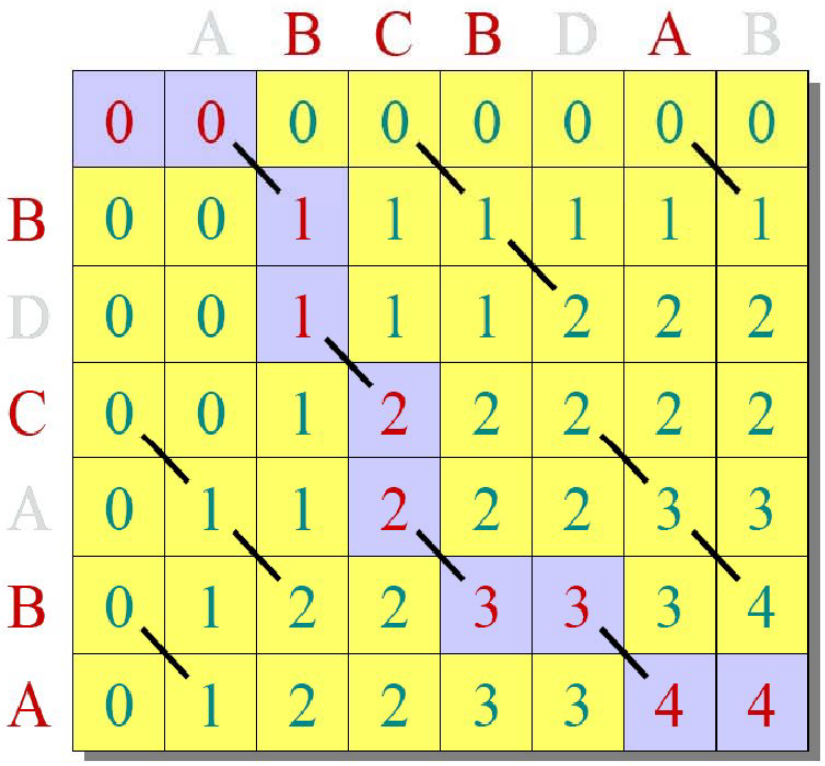
CONSTRUCTING AN LCS

- Initial call is PRINT-LCS (B, *X, m, n)*
- B[i, j] points to table entry whose subproblem we used in solving LCS of Xi and Y j .
- When *b[i, j]* = ↖ , we have extended LCS by one character. So longest common subsequence = entries With ↖ in them.

Print-LCS(*B, X, i, j*)
IF *i=0* or *j=0* THEN Return;
IF *B[i, j]*=" ↖ "
THEN Print-LCS(*B, X, i-1, j-1*); Print x_i;
ELSE If *B[i, j]*=" ↑ "
THEN Print-LCS(*B, X, i-1, j*);
ELSE Print-LCS(*B, X, i, j-1*).

EXAMPLE

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



		A	B	C	B	D	A	B
	0	0	0	0	0	0	0	0
B	0	0	1	1	1	1	1	1
D	0	0	1	1	1	2	2	2
C	0	0	1	2	2	2	2	2
A	0	1	1	2	2	2	3	3
B	0	1	2	2	3	3	3	4
A	0	1	2	2	3	3	4	4

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0										
2	p	0										
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0										
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
3	a	0										
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←
4	n	0										
5	k	0										
6	i	0										
7	n	0										
8	g	0										

Demonstration

i		j	0	1	2	3	4	5	6	7	8	9	10
			y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↖	
5	k	0											
6	i	0											
7	n	0											
8	g	0											

Demonstration

i	j	0	1	2	3	4	5	6	7	8	9	10
		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↖
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↑
6	i	0										
7	n	0										
8	g	0										

Demonstration

	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↖
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↑
6	i	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↖	3←	3↑
7	n	0										
8	g	0										

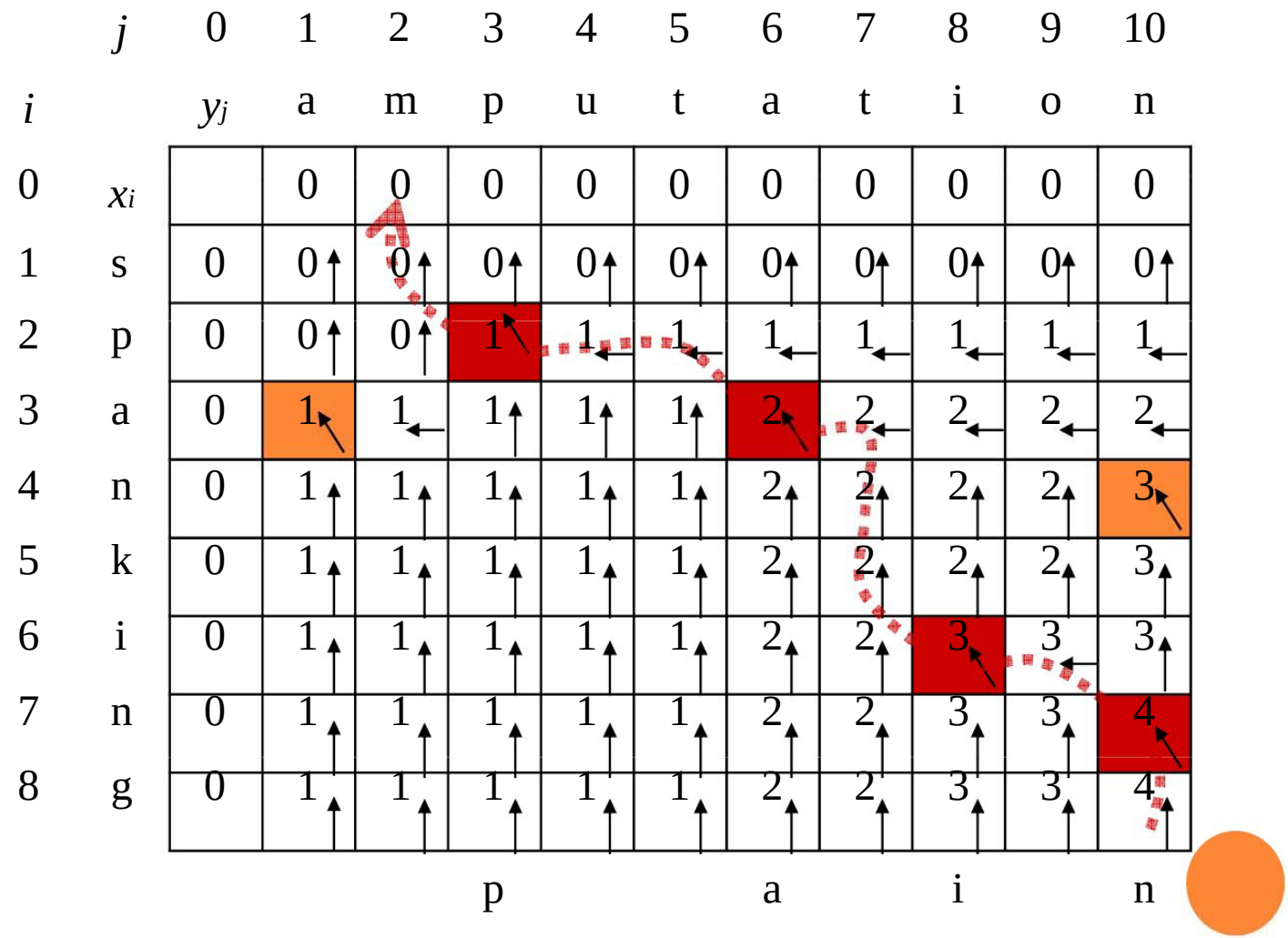
Demonstration

i		j	0	1	2	3	4	5	6	7	8	9	10
			y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↖	
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↑	
6	i	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↖	3←	3↑	
7	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	3↑	4↖	
8	g	0											

Demonstration

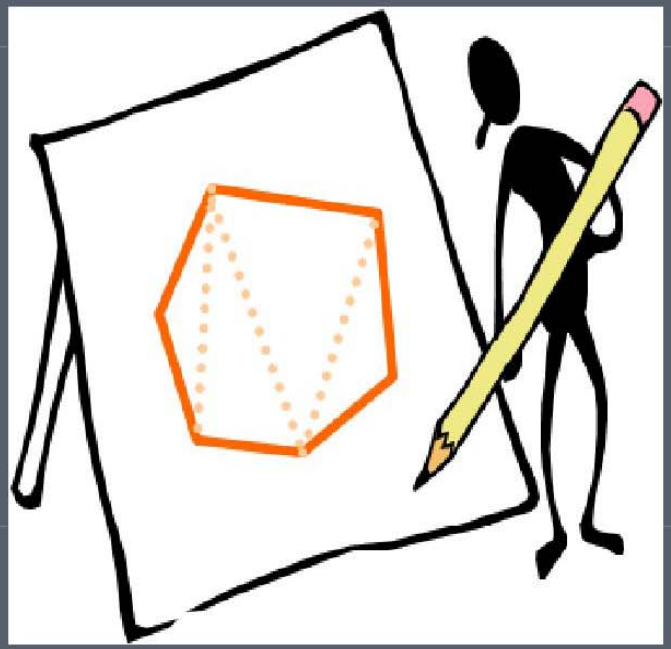
	j	0	1	2	3	4	5	6	7	8	9	10
i		y_j	a	m	p	u	t	a	t	i	o	n
0	x_i		0	0	0	0	0	0	0	0	0	0
1	s	0	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑	0↑
2	p	0	0↑	0↑	1↖	1←	1←	1←	1←	1←	1←	1←
3	a	0	1↖	1←	1↑	1↑	1↑	2↖	2←	2←	2←	2←
4	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↖
5	k	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	2↑	2↑	3↑
6	i	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↖	3←	3↑
7	n	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	3↑	4↖
8	g	0	1↑	1↑	1↑	1↑	1↑	2↑	2↑	3↑	3↑	4↑

Demonstration



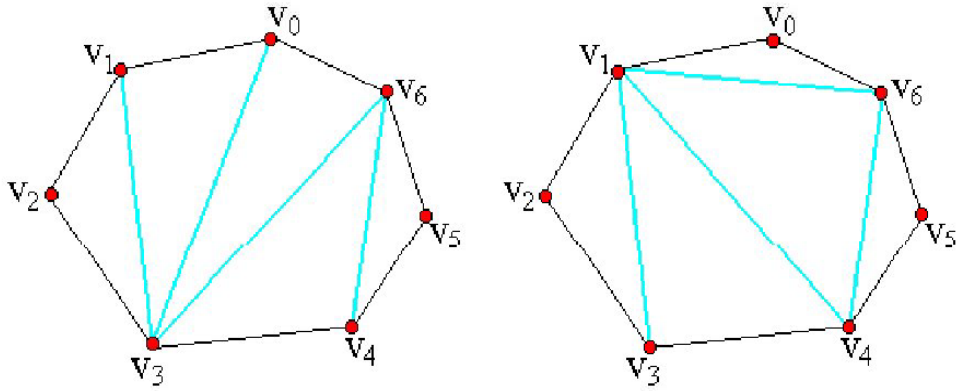
Time: $\cup$ (mn)

Triangle Decomposition of Convex Polygon



DEFINITION

- Convex polygon $P = \{v_0, v_1, ..., v_{n-1}\}$
- Triangle decomposition: chords set *T* that decompose polygon into disjoint triangle
- We can improve that in a triangle decomposition of a ***n*** vertices convex polygon, there happened to be n-1 chords and n-2 triangles.



DEFINITION

Weighting function w

for example:

$$w(v_i v_j v_k) = |v_i v_j| + |v_j v_k| + |v_i v_k|$$

where $|v_i v_j|$ is the Euclid distance between v_i and v_j

DEFINITION

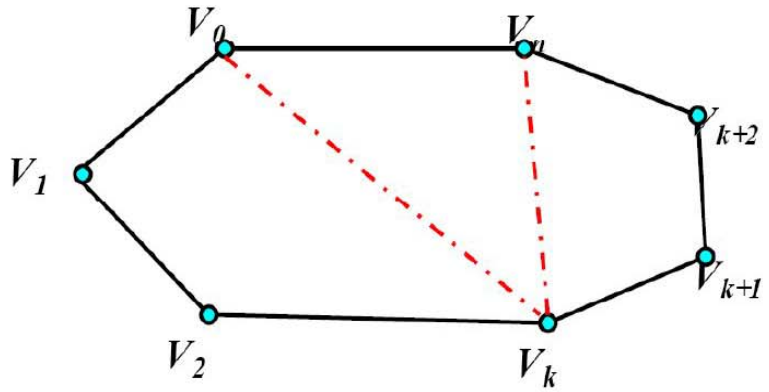
Optimal triangle decomposition

- Input: polygon P and weighting function W
- Output: triangle decomposition T, to minimize

$$\blacktriangleleft_{s \in S_T} W(s)$$

THE STRUCTURE OF OPTIMAL SOLUTION

- $P=(v0,v1,...vn)$*is* a $n+1$ vertices polygon
- T_P is an optimal triangle decomposition, v_k is the decomposition point.



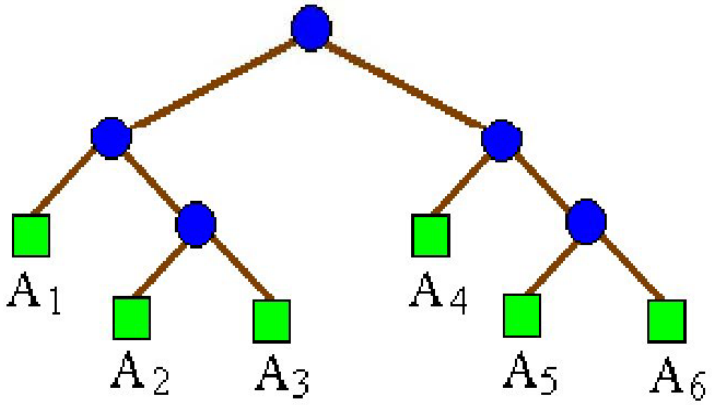
- The structure of optimal solution

$$T_P = T(v0, ..., v_k) \cup T(v_k, ..., v_n) \cup \{v0v_k, v_kv_n\}$$

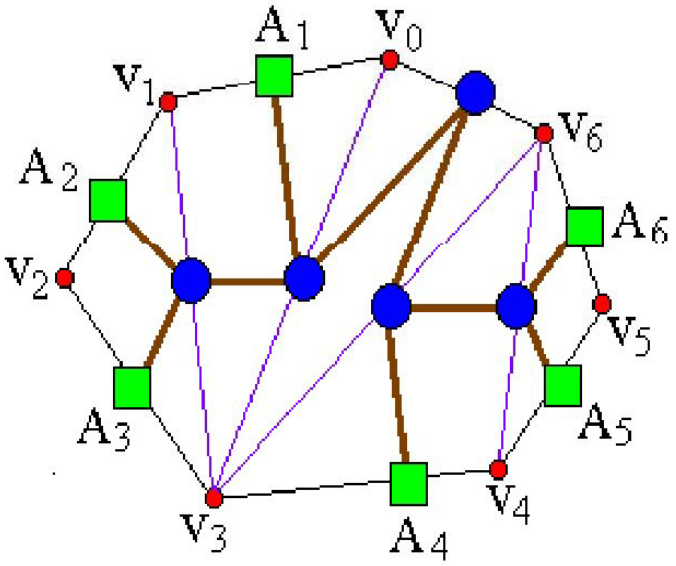
TRIANGLE DECOMPOSITION AND MATRIX-CHAIN-ORDER

- A matrix chain is corresponding to a complete binary tree. For example:

  ((A1(A2A3))(A4(A5A6))) is converted to a complete binary tree.



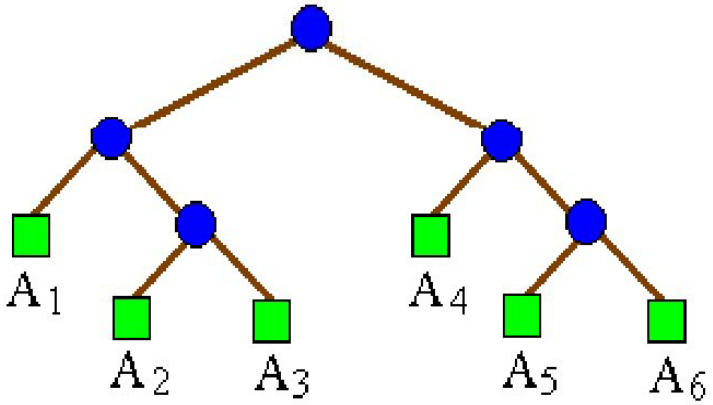
(a)



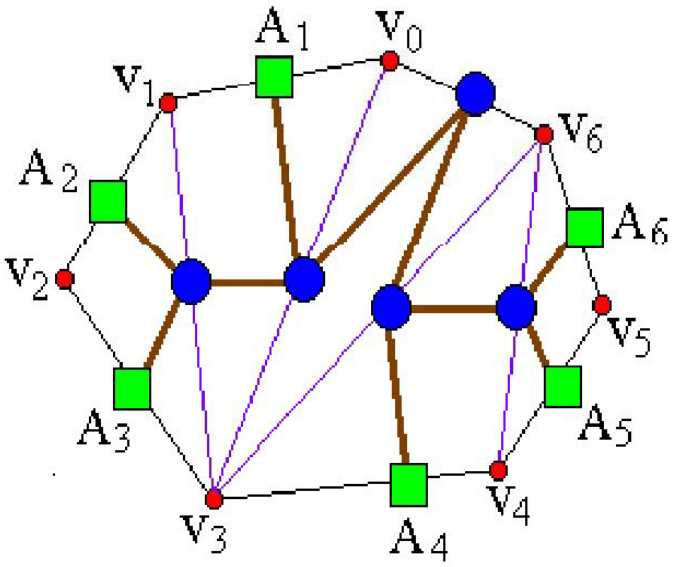
(b)

TRIANGLE DECOMPOSITION AND MATRIX-CHAIN-ORDER

Triangle decomposition of a *n-vertices* convex polygon is corresponding to a *n-1* leaves complete binary tree.



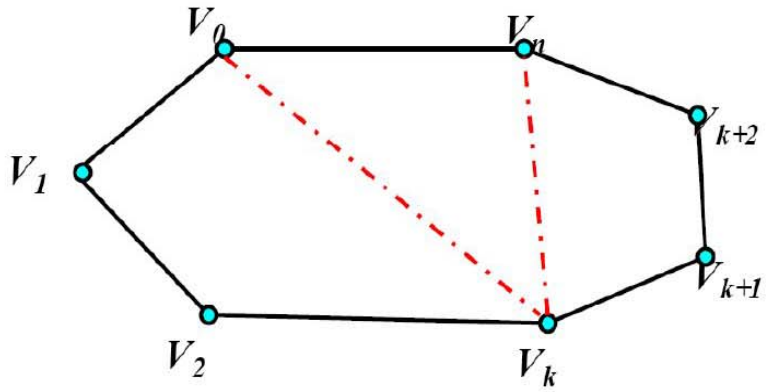
(a)



(b)

RECURSIVE SOLUTION

$$t[i][j] = \begin{cases} 0 & i = j \\ \min\{t[i][k] + t[k+1][j] + w(v_i v_{i-1} v_k v_j)\} & i < j \end{cases}$$



CONSTRUCT OPTIMAL SOLUTION

It is coordinate with the Matrix-chain-Order. So modify Matrix-chain-order, we can construct optimal triangle decomposition.