Dynamic Programming

1. Matrix chain Multiplication.

$$\begin{pmatrix} x & x \\ x & x \\ x & x \end{pmatrix} \begin{pmatrix} y & y & y & y \\ y & y & y & y \end{pmatrix} \begin{pmatrix} z \\ z \\ z \\ z \end{pmatrix} = \begin{pmatrix} \cdot \\ \cdot \\ \cdot \end{pmatrix}$$

$3 \times 2 \qquad 2 \times 4 \qquad 4 \times 1 \qquad 3 \times 1$

$$\begin{pmatrix} xy+xy & xy+xy & xy+xy & xy+xy \\ xy+xy & \cdot & \cdot & \cdot \\ xy+xy & \cdot & \cdot & \cdot \end{pmatrix}$$

$3.2.4 = 24$ multiplications

$$\begin{pmatrix} (xy+xy)z + (xy+xy)z + (xy+xy)z + (xy+xy)z \\ \cdot \\ \cdot \end{pmatrix}$$

$3.4.1 = 12$ multiplications

Total
$24+12=36$
But, there is another order requiring
$8+6=14$ mult.

<u>Matrix chain problem</u> Given matrices $A_1 A_2 \cdots A_n$ of size $p_0 \times p_1,\ p_1 \times p_2, \cdots, p_{n-1} \times p_n$, find a parenthesization that minimizes the number of multiplications.

Claim. There are $\frac{1}{n}\binom{2n-2}{n-1}$ different parenthesizations of n matrices.

$((((x\,x)(x\,x))(x\,x))$ or $(((x(x\,x))(x\,x))x)$ ... etc.

So, it is not efficient to try all parenthesizations.

A recursive solution

Let m_{ij} be the min. number of multiplication needed for $A_i A_{i+1} \cdots A_j$. We assume that pq-matrix times qr-matrix takes pqr mult.

```
function M(i,j)
  if i=j then return M:=0
    else  min := ∞;
      for k:= i to j-1 do
          mult:= M(i,k) + M(k+1, j) + p[i-1]*p[k]*p[j];
          if mult < min then min:= mult endif
      endfor
  endif
```

Assume array p[0...n] contains matrix sizes

Analysis.

$$T(n) = \left(\sum_{k=1}^{n-1} (T(k) + T(n-k)) \right) + O(1)$$

$$= 2 \sum_{k=1}^{n-1} T(k) + O(1)$$

$$\geq 2T(n-1) + O(1)$$

$$\geq \sum_{i=0}^{n-1} 2^i$$

$$= 2^n - 1$$

The main reason for this blow-up in the time complexity is that the algorithm repeatedly solves the same problem recursively. (Think about it why).

Solution. Store the results of the subproblems once computed and use them as needed.

Dynamic Programming Approach

The algorithm uses a tables M[1...n, 1...n] and S[1...n, 1...n] to store optimal solutions to subproblems. It works bottom-up.

```
procedure Matrixchain (p);
  for i:=1 to n do M[i,i]:= 0 endfor
  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
           mult:= M[i,k]+M[k+1, j]
                      + p[i-1]* p[k] * p[j];
           if (mult < M[i,j]) then
                M[i,j]:= mult;
                S[i,j]:= k
           endif
        endfor
     endfor
  endfor
```

Example.

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

p =

0	1	2	3	4
4	2	5	1	3

M:

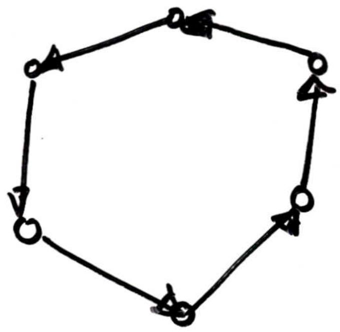
	1	2	3	4
1	0	40	18	30
2		0	10	16
3			0	15
4				0

S:

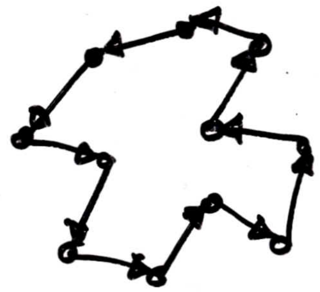
	1	2	3	4
1		1	1	3
2			2	3
3				3
4				

```
Print-Parent (S, i, j)
  if i=j then print 'A'
     else print '('
        print-Parent (S, i, S[i,j])
        print-Parent (S, S[i,j]+1, j)
        print ')'
  endif
```

Polygon Triangulation



Convex



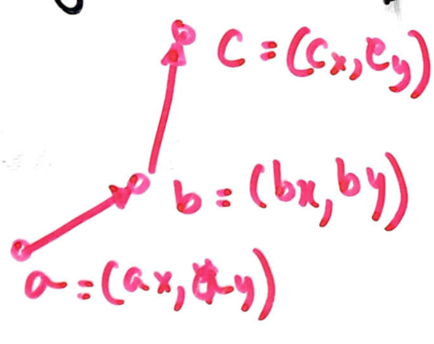
non-convex

A polygon can be represented with the counterclockwise sequence of its vertices.

type point = record x, y end
        polygon = array [0,...n-1] of point

var. P : polygon

a. Left-turns. A polygon is convex iff any 3 consequentive points form a left-turn



Claim abc is a left-turn iff

$$\det \begin{bmatrix} a_x & a_y & 1 \\ b_x & b_y & 1 \\ c_x & c_y & 1 \end{bmatrix} > 0.$$

Existence of a Polygon Triangulation

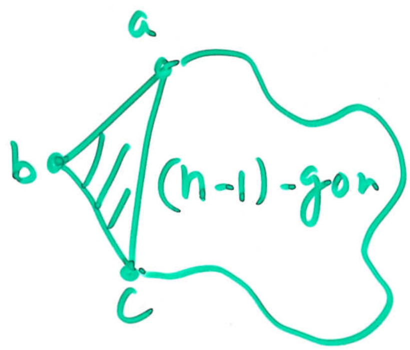
A triangulation is a decomposition of the polygon's interior into triangles whose vertices are the vertices of the polygon.

Claim. Every polygon can be triangulated.

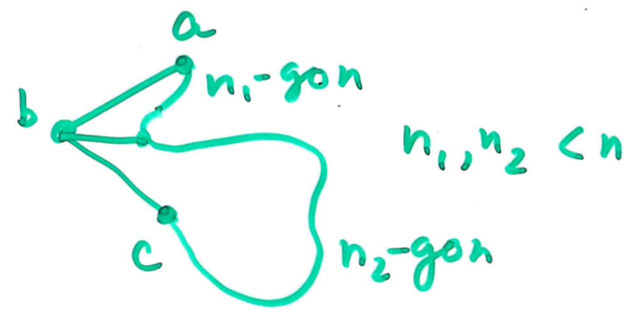
Proof: by Induction.

Let b be the leftmost point and a and c be its predecessor and successor.

Case 1.



Case 2.



The inductive argument can be used to show that the number of triangles is $\underline{n-2}$ and the number of chords is $\underline{n-3}$.

$t(n) = 1 + t(n-1)$ $\qquad$ $c(n) = 1 + c(n-1)$ and

$t(n) = t(n_1) + t(n_2)$ $\qquad$ $c(n) = 1 + c(n_1) + c(n_2)$

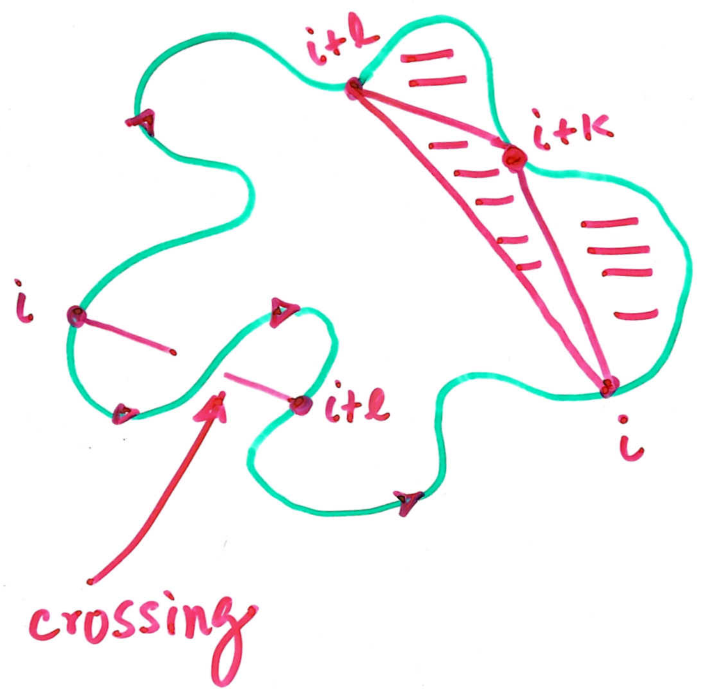
$n_1 + n_2 = n+2$ $\qquad$ $n_1 + n_2 = n+2$

Construction of Optimal Triangulation

It is similar to Matrix chain multiplication.

- Uses arrays $T[0 \ldots n-1, \phi \ldots n-1]$ and $V[0 \ldots n-1, 1 \ldots n-1]$.

where $T[i, l]$ stores weight of the best triangulation $P[i, i+L]$ and $V[i, l]$ stores the vertex $P[i+k]$ so that $p[i]\,P[i+k]\,p[i+l]$ is a triangle in the optimal triangulation.



$P[i \ldots i+l]$ is a polygon iff there is a k with $1 \le k \le l-1$, s.t.

$p[i \ldots i+k]$ is non-intersecting

$p[i+k \ldots i+l]$ " " "

$p[i]\,P[i+k]\,p[i+l]$ is a lt-turn

Assume $w(a,b,c)$ is the weight of a triangle (a,b,c) which is non-negative.

```
for i:=0 to n-1 do T[i,1]:=0 endfor;
for l:=2 to (n-1) do
    for i:= 0 to (n-1) do
        T[i,l] := ∞
        for k:= 1 to l-1 do
            if Left-turn(i, i+k, i+l) then
                t := T[i,k] + T[i+k, l-k] + w(i,i+k, i+l)
                if t < T[i,l] then
                        T[i,l]:=t; V[i,l]:=i+k
                endif
            endif
        endfor
    endfor
endfor.
```

$O(n^3)$ time

$O(n^2)$ space.