アルゴリズム論
Theory of Algorithms

第９回講義
動的計画法(2)

アルゴリズム論
Theory of Algorithms

Lecture #9
Dynamic Programming (2)

動的計画法に基づくアルゴリズムの開発

最適化問題が対象：
制約を満たす解の中で定められた基準で最適なものを求めるのが問題.

動的計画法による問題解決

1. 最適解の構造を特徴づける.
2. 最適解の値を再帰的に定義する.
   （部分問題の解を用いて問題の解を構成する）
3. ボトムアップの形式で最適解の値を求める.
   （表を埋めていく方式）
4. 計算で求められた情報から1つの最適解を構成する.
   （最適解の値だけでなく，表を辿ることで最適解を構成）

Developing Algorithms based on Dynamic Programming

Objects: optimization problems

problem of finding an optimal solution among those satisfying given constraints.

Problem solving by dynamic programming

1. Characterize a structure of an optimal solution.
2. Define an optimal solution recursively.
   (construct a solution using solutions to subproblems)
3. Compute a value of an optimal solution in a bottom-up manner
   (in the way to fill in a table)
4. Construct an optimal solution using information obtained.
   (not only finding a value of an optimal solution but also constructing an optimal solution by following in the table)

問題P24:（最適2分探索木の構成）
n個のデータを2分探索木に蓄えるのに，各データに対する検索の確率（頻度）が予め予想できるとき，探索のための比較回数（の期待値）が最小になるように2分探索木を構成せよ.

蓄えるべき値： $S = \{a_1, a_2, \ldots, a_n\}$, $a_1 \leqq a_2 \leqq \cdots \leqq a_n$
事前知識：必ずSの要素だけが検索されるものと仮定.
　Find(a_i, S)の出現確率はp_i
Sを2分探索木に蓄えたとき，
　Sの要素a_iを含む節点のレベルをlevel(a_i)とする.
　　a_iの探索に必要な比較回数はlevel(a_i) +1回（根のレベルは0）
したがって，探索木のコスト（比較回数の期待値）は，次式で与えられる：

$$\text{探索木のコスト} = \sum p_i \times [\text{level}(a_i) + 1]$$

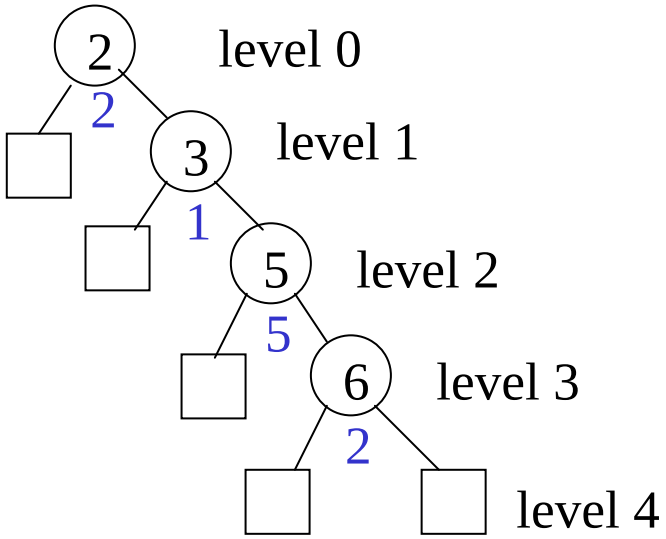
Problem P24: (Construction of an optimal binary search tree)
When probability that each element is asked is given, store n data in a binary search tree so that the expected number of comparisons to locate a query in the tree is minimized.

Data to be stored: $S = \{a_1, a_2, \ldots, a_n\}$, $a_1 \leqq a_2 \leqq \cdots \leqq a_n$
A priori knowledge: Assume that only elements of S are retrieved.
probability for Find(a_i, S) is p_i
When S is stored in a binary search tree,
let the level of a node a_i containing an element of S be level(a_i).
the number of comparisons for searching a_i is level(a_i) +1
(assuming the level of the root node is 0)
Therefore, the cost of a search tree (expected number of comparisons) is given by

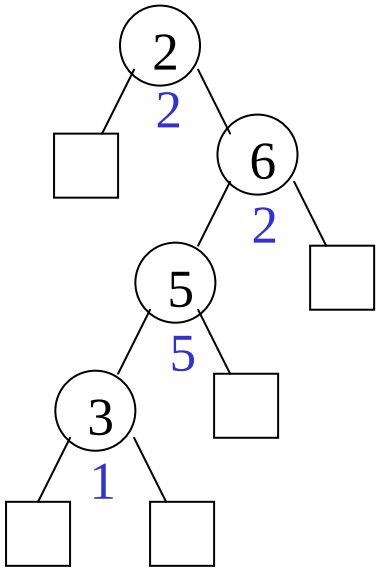
$$\text{Cost of search tree} = \sum p_i \times [\text{level}(a_i) + 1]$$

例題:

	a[1]	a[2]	a[3]	a[4]
S	2	3	5	6
	2/10	1/10	5/10	2/10
	p_1	p_2	p_3	p_4



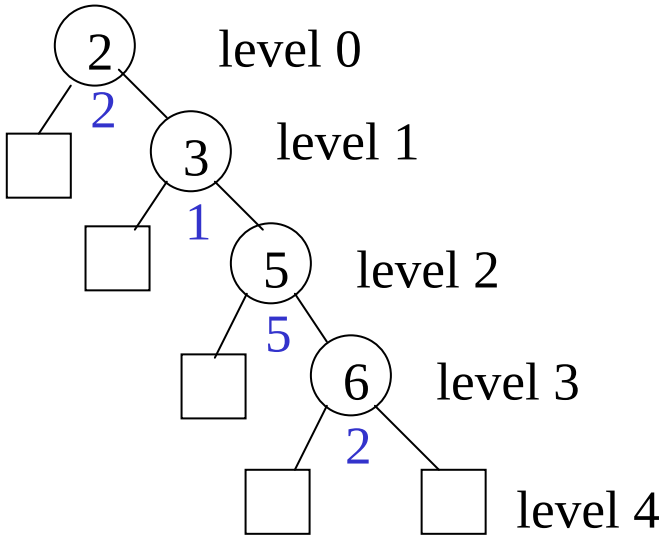
コスト=(2*1+1*2+5*3+2*4)/10
=2.7



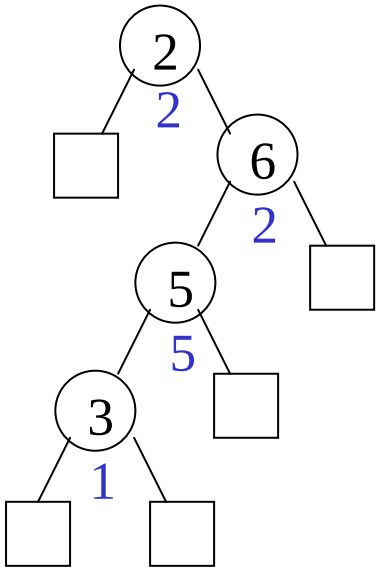
コスト=(2*1+2*2+5*3+1*4)/10
= 2.5

Examlpe:

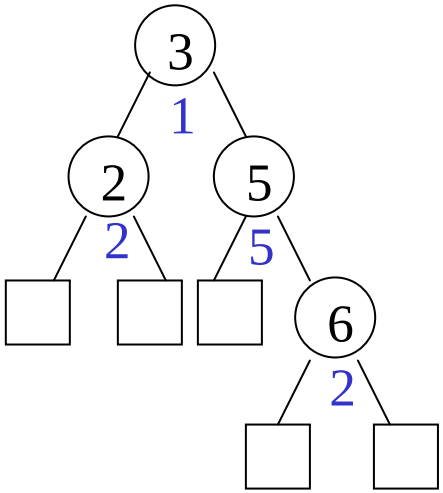
	a[1]	a[2]	a[3]	a[4]
S	2	3	5	6
	2/10	1/10	5/10	2/10
	p_1	p_2	p_3	p_4

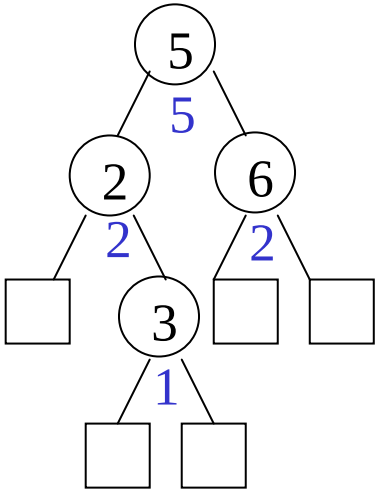


cost=(2*1+1*2+5*3+2*4)/10
=2.7



cost=(2*1+2*2+5*3+1*4)/10
= 2.5

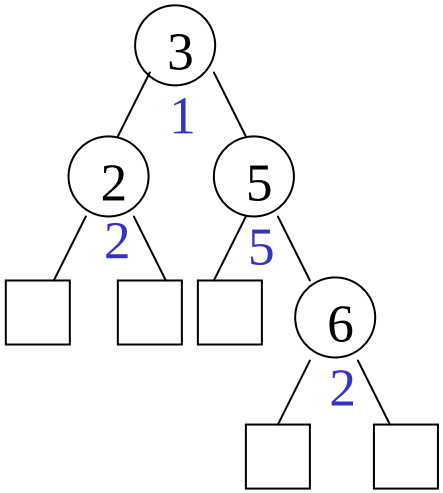


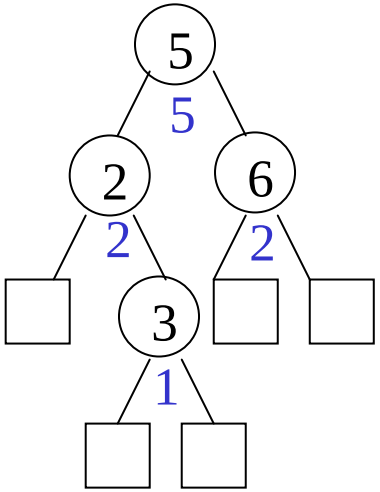


コスト=(1*1+(2+5)*2+2*3)/10 =2.1

コスト=(5*1+(2+2)*2+1*3)/10 = 1.6

すべての探索木を列挙してコストを計算すれば最適な探索木が求まるが，すべてを列挙するのは効率が悪い．





cost=(1*1+(2+5)*2+2*3)/10 =2.1

cost=(5*1+(2+2)*2+1*3)/10 = 1.6

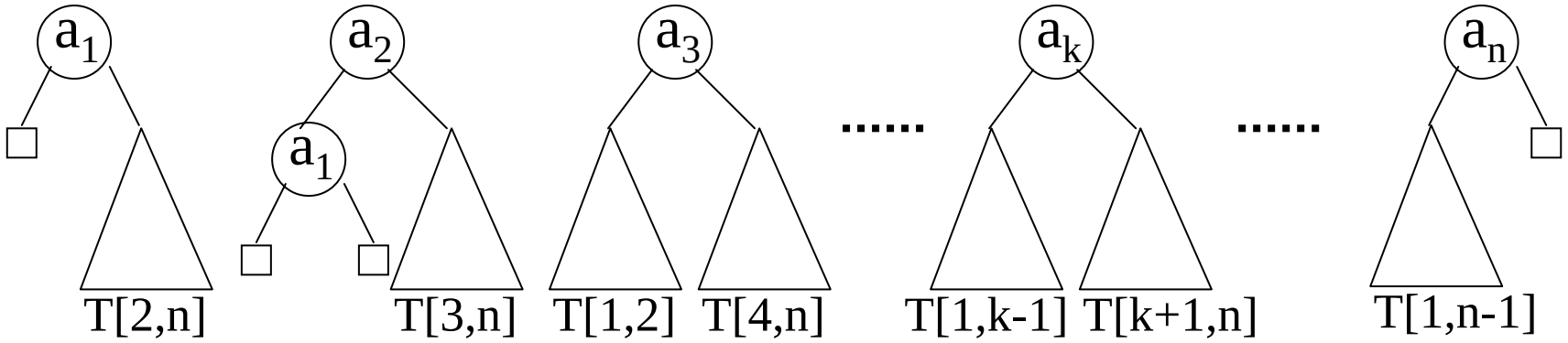
If we enumerate all search trees and compute their costs, then we can find an optimal search tree. But it is not efficient to enumerate all of them.

最適な2分探索木の構成

最適解の構造を特徴づけ，最適解の値を再帰的に定義する．

T[i,j] = 部分集合$\{a_i, a_{i+1}, \ldots, a_j\}$に対する最小コスト木
i=1, ... ,n, j=i, i+1, ... , n

すべての可能性を列挙すると



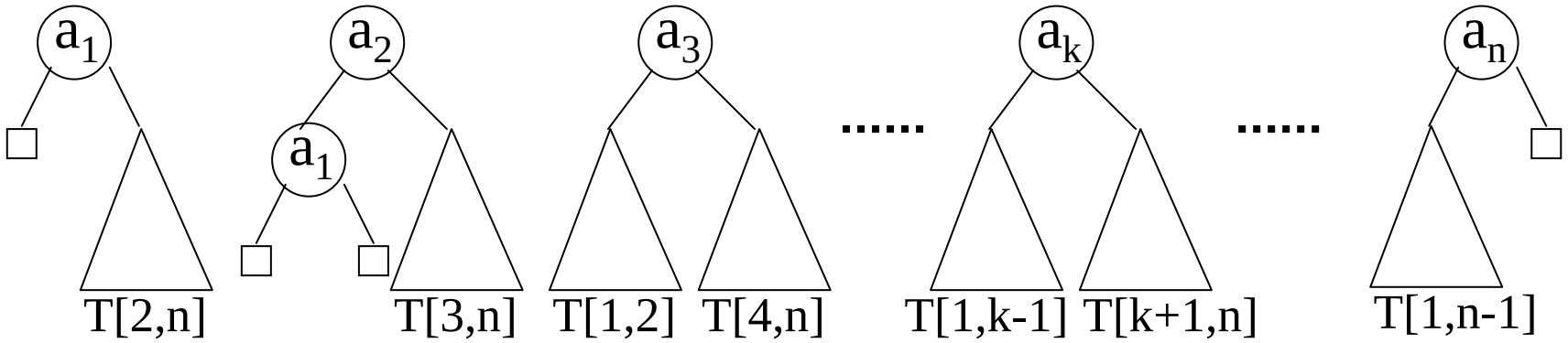
T[2,n], T[3,n], ... , T[1,2], T[4,n], ... , T[1,n-1]がすべて
求まっていれば，上のそれぞれの木のコストを計算できる．
最小コストの木を選べば，その根a_kを決めることが可能．

Construction of an optimal binary search tree

Characterize structure of an optimal solution and define the value of an optimal solution recursively.

T[i,j] = minimum-cost tree for a subset $\{a_i, a_{i+1}, \ldots, a_j\}$
$i=1, \ldots, n, \quad j=i, i+1, \ldots, n$

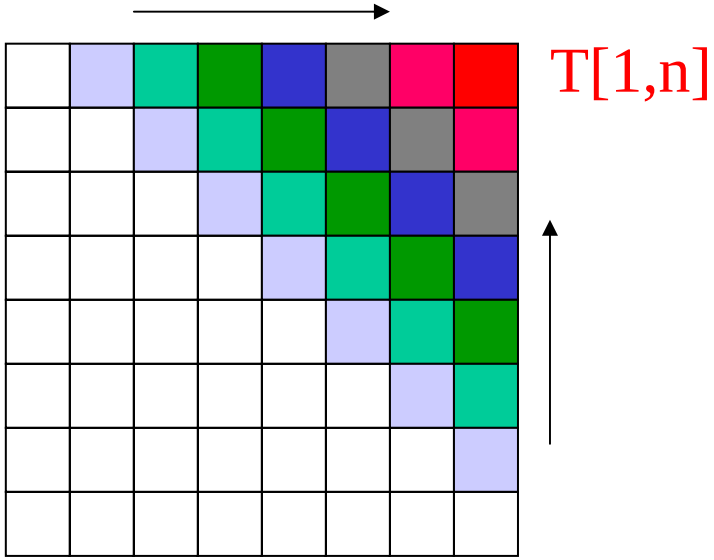
Enumerating all possibilities:



If T[2,n], T[3,n], ... , T[1,2], T[4,n], ... , T[1,n-1] are all available, costs of those trees can be computed. If we choose the minimum-cost tree, we can determine its root a_k.

ボトムアップの形式で最適解の値を求める.

{T[i, i+1], i=1, 2, ... , n-1}を求める・・・・・差1
{T[i, i+2], i=1, 2, ... , n-2}を求める・・・・・差2
:
{T[i, i+k], i=1, 2, ... , n-k}を求める・・・・・差k
:
最後にT[1, n]が求まれば，これが最適解の値.



Computing the value of optimal solution in a bottom-up fashion

{T[i, i+1], i=1, 2, ... , n-1} is computed⋯⋯difference 1

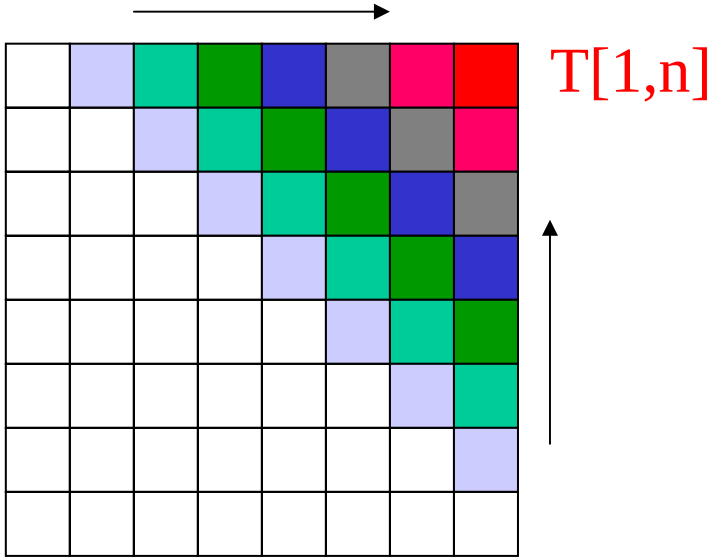
{T[i, i+2], i=1, 2, ... , n-2} is computed⋯⋯difference 2

:

{T[i, i+k], i=1, 2, ... , n-k} is computed⋯⋯difference1 k

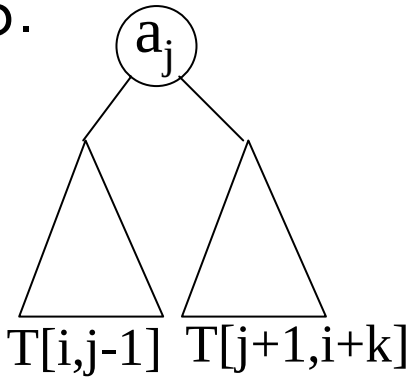
:

Finally, we compute T[1, n], which is the value of optimal solution.



T[i, i+k]の求め方

T[i,i+k] = 部分集合{a_i, a_{i+1}, ... , a_{i+k}}に対する最小コスト木
だから，その根は，a_i, a_{i+1}, ... , a_{i+k}のk+1通りある．
a_jを根として選んだとき，
　左部分木をT[i,j-1]，右部分木をT[j+1,i+k]
　とするのが最適．
　T[i,j-1]とT[j+1,i+k]でのコストの計算よりも
　レベル1分だけ増えていることに注意．



$T[i,j-1] = \sum p_m \times [level(a_m) + 1]$
レベルを1だけ増やすと，
$T'[i,j-1] = \sum p_m \times [level(a_m) + 2] = T[i,j-1] + \sum p_m$
つまり，T[i,j-1]に$p_i + p_{i+1} + ... + p_{j-1}$を加えれば
1レベル下げた値が求まる．T'[j+1,i+k]についても同じ．
よって，a_jを根とするときのコストは次式で与えられる：

$$p_j + T'[i,j-1] + T'[j+1,i+k]$$
$$= T[i,j-1] + T[j+1,i+k] + p_i + p_{i+1} + ... + p_{j+k}$$

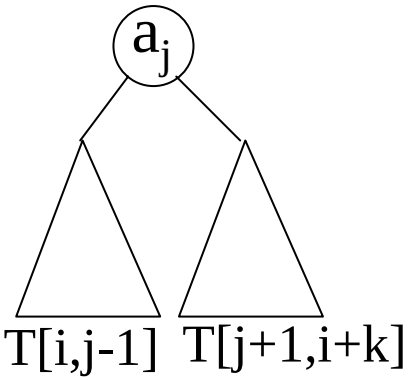
How to compute T[i, i+k]

T[i,i+k] = min-cost tree for a subset $\{a_i, a_{i+1}, \ldots, a_{i+k}\}$. Thus, k+1 different roots $a_i, a_{i+1}, \ldots, a_{i+k}$ are possible.

If we choose a_j as a root,

an optimal solution has T[i,j-1] as its left subtree and T[j+1,i+k] as right subtree.

Note that one level is increases than when computing the costs for T[i,j-1] and T[j+1,i+k].



$T[i,j-1] = \sum p_m \times [\text{level}(a_m) + 1]$

If we increase the level by one,

$T'[i,j-1] = \sum p_m \times [\text{level}(a_m) + 2] = T[i,j-1] + \sum p_m$

That is, we have the value one level down by adding $p_i + p_{i+1} + \ldots + p_{j-1}$ to T[i,j-1]. Same for T'[j+1,i+k].

Thus, the cost with a_j at the root is given by:

$p_j + T'[i,j-1] + T'[j+1,i+k]$

$= T[i,j-1] + T[j+1,i+k] + p_i + p_{i+1} + \ldots + p_{j+k}$

T[i, i+k]の求め方

$C[i,j] = \{a_i, a_{i+1}, \ldots, a_j\}$に対する最小木T[i,j]のコスト
$W[i,j] = p_i + p_{i+1} + \ldots + p_j$
とすると

a_jを根とするときのコストは次式で与えられる
$C[i,j-1]+C[j+1,i+k]+ W[i, i+k]$

jを変化させて上記の値の最小値を取ればC[i,i+k]が求まる.
a_iとa_{i+k}が根となる場合も考慮すると，次の式を得る:

$$C[i,i+k] = \min\{ C[i+1,i+k]+W[i,i+k],$$
$$\min\{C[i,j-1]+C[j+1,i+k]+W[i,i+k],\ j=i+1, \ldots, i+k-1\},$$
$$C[i,i+k-1]+W[i,i+k]\}$$

$k=1, 2, \ldots, n-i$
として順に求めることができる.

How to compute T[i, i+k]

C[i,j] = cost of the minimum-cost tree T[i,j] for $\{a_i, a_{i+1}, \ldots, a_j\}$
$W[i,j] = p_i + p_{i+1} + \ldots + p_j$
Then,

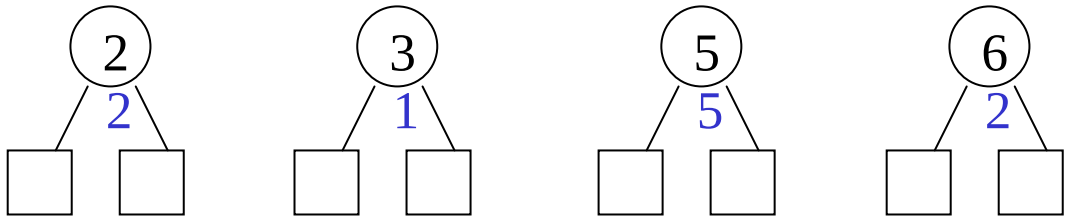
the cost when a_j is the root is given by the following:
C[i,j-1]+C[j+1,i+k]+ W[i, i+k]

C[i,i+k] is obtained by taking the minimum value while varying j.
Considering the cases where a_i and a_{i+k} are roots, we have

$$C[i,i+k] = \min\{\ C[i+1,i+k]+W[i,i+k],$$
$$\min\{C[i,j-1]+C[j+1,i+k]+W[i,i+k],\ j=i+1, \ldots, i+k-1\},$$
$$C[i,i+k-1]+W[i,i+k]\}$$
$$k=1, 2, \ldots, n-i$$

C[i,j] = {a_i, a_{i+1}, ... , a_j}に対する最小木T[i,j]のコスト

W[i,j] = p_i + p_{i+1} +... + p_j

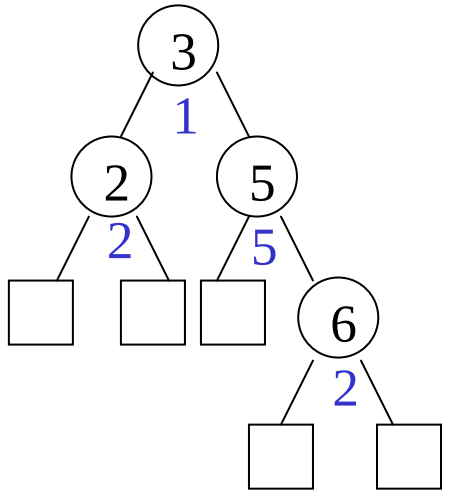


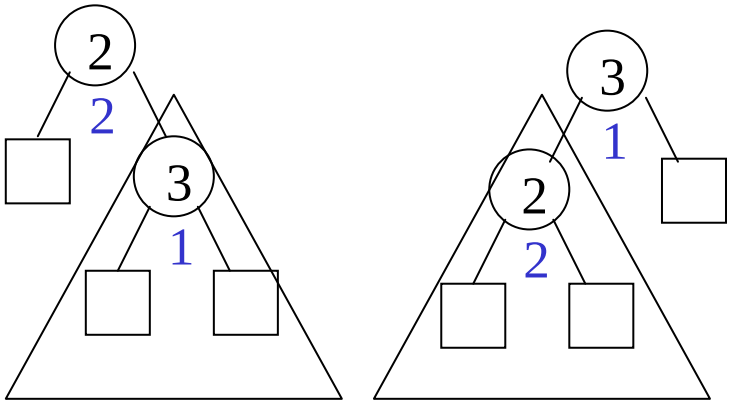
T[1,1] C[1,1]=0.2

T[2,2] C[2,2]=0.1

T[3,3] C[3,3]=0.5

T[4,4] C[4,4]=0.2





T[2,2]

コスト

=0.2+C[2,2]+W[2,2]

=0.2+0.1+0.1=0.4

T[1,1]

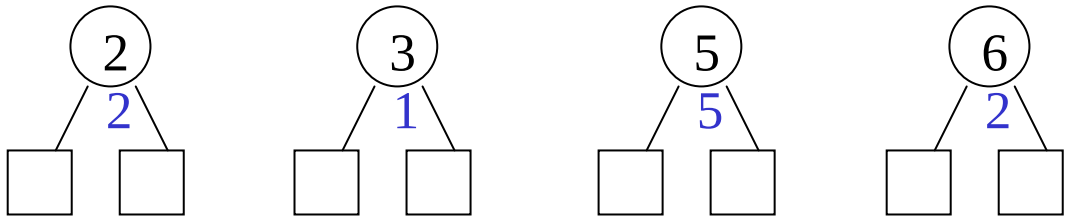
コスト

=0.1+C[1,1]+W[1,1]

=0.1+0.2+0.2=0.5

C[i,j] = cost of minimum-cost tree T[i,j] for $\{a_i, a_{i+1}, \ldots, a_j\}$

$W[i,j] = p_i + p_{i+1} + \ldots + p_j$

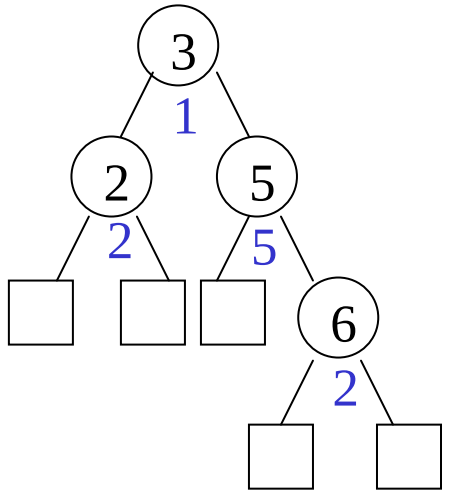


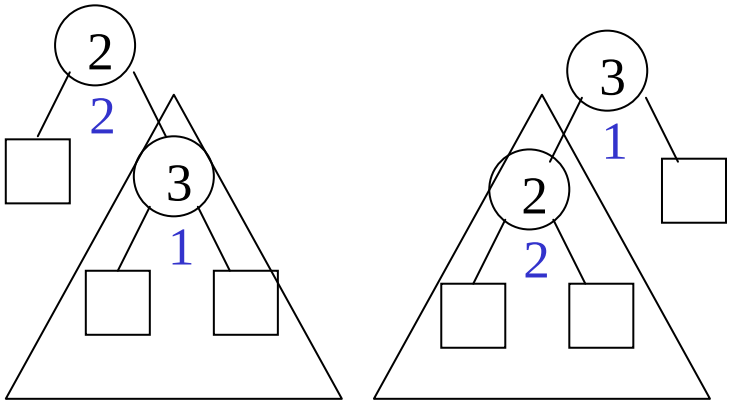
T[1,1] C[1,1]=0.2

T[2,2] C[2,2]=0.1

T[3,3] C[3,3]=0.5

T[4,4] C[4,4]=0.2





T[2,2]

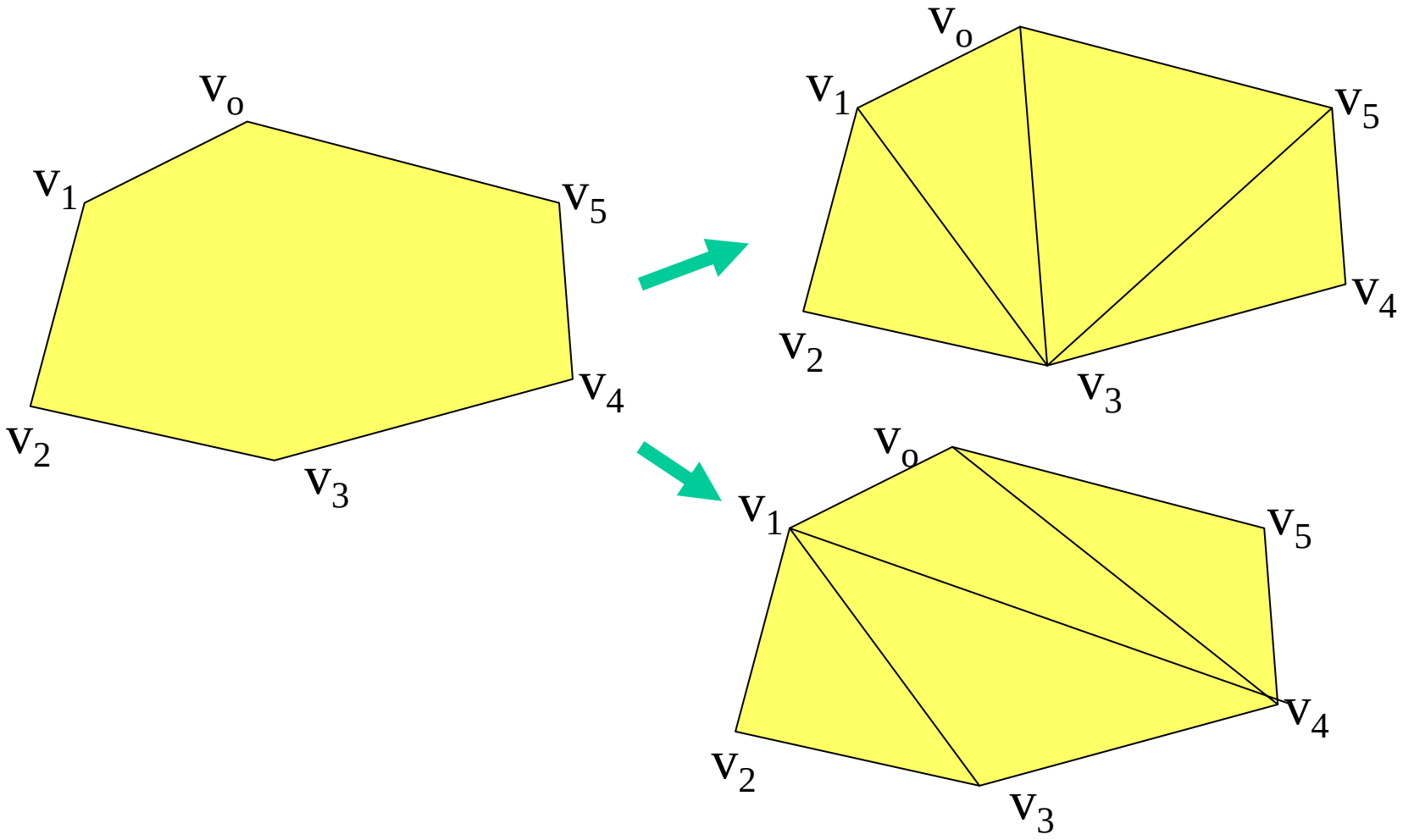
T[1,1]

コスト
=0.2+C[2,2]+W[2,2]
=0.2+0.1+0.1=0.4

コスト
=0.1+C[1,1]+W[1,1]
=0.1+0.2+0.2=0.5

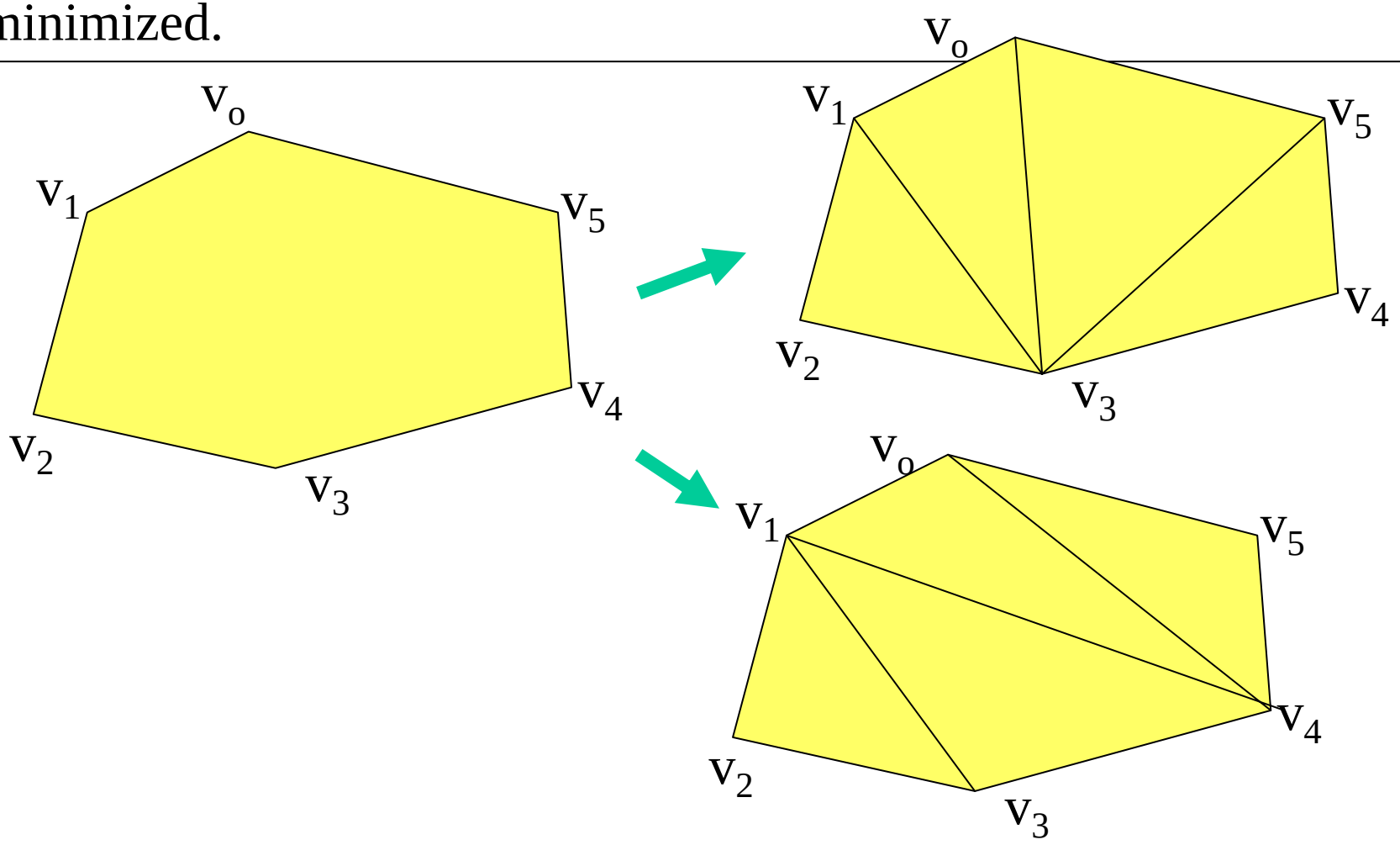
問題P24:（最適三角形分割）

凸多角形が頂点の系列として与えられたとき，2つの頂点の間に弦を引くことにより多角形の内部を三角形に分割することができるが，弦の長さの総和を最小にする三角形分割を求めよ.



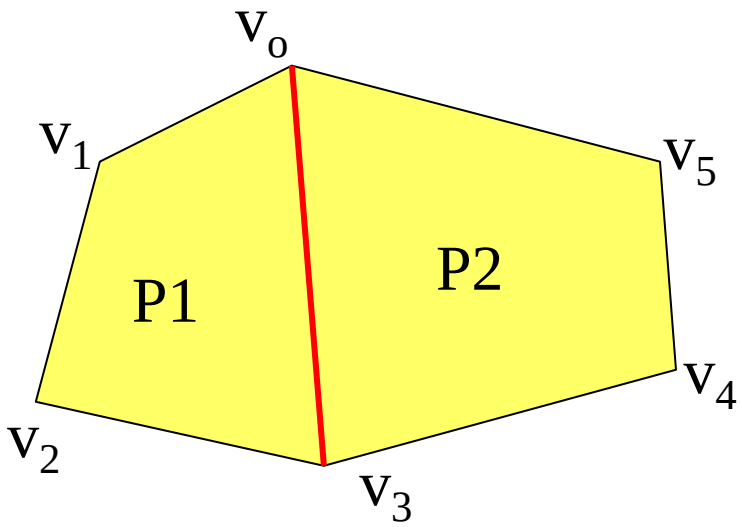
Problem P24:（Optimal triangulation）

Given a convex polygon as a vertex sequence, we can partition its interior into triangles by drawing chords between two vertices. Find a triangulation so that the total sum of lengths of chords is minimized.



最適解の構造を特徴づけ，最適解の値を再帰的に定義する．

凸多角形を$P(v_o, v_1, v_2, v_3, v_4, v_5)$のように頂点の系列で表現．
頂点v_oについて考えると，
　ケース１：v_oから別の頂点v_iにむけて弦を引く．
　ケース２：v_oにつながる弦はない．



演習問題E9-1：ケース２のとき，必ず両隣の頂点が弦で結ばれることを証明せよ．

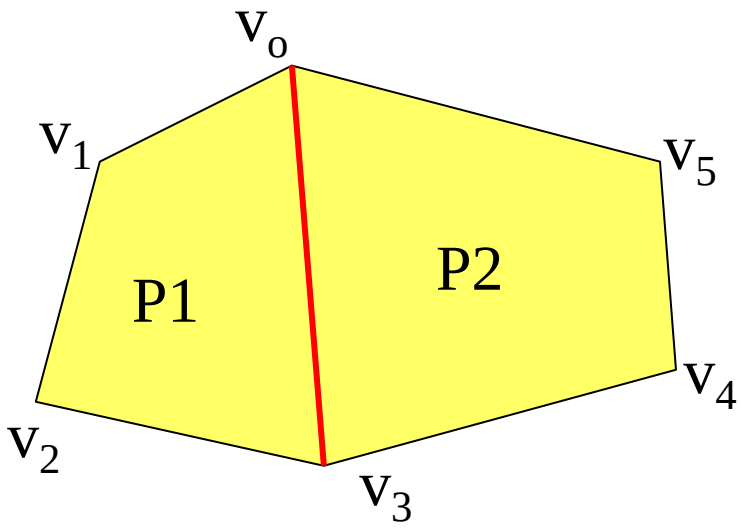
弦を１本引くことによって生じる部分多角形はやはり凸であり，頂点数はn未満だから，すべての部分多角形について最適解を求めておけば，元の問題の最適解が得られる．

Characterize structure of an optimal solution and define the value of an optimal solution recursively.

Represent a convex polygon as a vertex sequence like $P(v_o, v_1, v_2, v_3, v_4, v_5)$. Considering a vertex v_o,

Case 1:draw a chord from v_o to another vertex v_i.

Case 2: there is no chord incident to v_o.



Exercise E9-1 : Prove that two adjacent must be connected by a chord in Case 2.

When we draw a chord, two resulting subpolygons are convex. Since it has less than n vertices, if we have all optimal solutions for all subproblems, then we can obtain an optimal solution.

凸n角形を分割する仕方は何通りあるだろう？

凸n角形$(v_0, \dots, v_{n-1})$を分割する仕方がf(n)通りあるとする.

ケース1：v_oから別の頂点v_iにむけて弦を引く.

弦としては$(v_0,v_2), (v_0,v_3), \dots, (v_0,v_{n-2})$が考えられる.

弦(v_0,v_2)→3角形(v_0,v_1,v_2) + (n-1)角形$(v_0,v_2,v_3,\dots,v_{n-1})$

弦(v_0,v_3)→4角形$(v_0,v_1,v_2,,v_3)$ + (n-2)角形$(v_0,v_3,v_4,\dots,v_{n-1})$

弦(v_0,v_4)→5角形$(v_0,v_1,v_2,,v_3,v_4)$ + (n-3)角形$(v_0,v_4,v_5,\dots,v_{n-1})$

：

弦(v_0,v_{n-2})→(n-1)角形$(v_0,v_1,v_2,\dots,v_{n-2})$ + 3角形(v_0,v_{n-2},v_{n-1})

ケース2：v_oにつながる弦はない.

弦(v_1,v_{n-1})→3角形(v_0,v_1,v_{n-1}) + (n-1)角形$(v_1,v_2,v_3,\dots,v_{n-1})$

同じ三角形分割が何度も現れることを考えると

$f(n) \leqq 3f(n-1) + 2f(n-2) + 2f(n-3) + \cdots + 2f(4) + 3f(3)$

$f(3) = 1.$

$g(n)=2g(n-1)$, $g(3)=1$なら$g(n)=2^{n-3}$だから，f(n)も指数関数.

すなわち，この方法では多項式時間では解けない！

How many different triangulations of a convex polygon?

Suppose that there are f(n) ways to triangulate a convex polygon $(v_0, \dots, v_{n-1})$.

Case 1: Draw a chord from v_o to v_i.

possible chords are (v_0,v_2), (v_0,v_3), ... , (v_0,v_{n-2}).

(v_0,v_2)=>triangle (v_0,v_1,v_2) + (n-1)-gon$(v_0,v_2,v_3,\dots,v_{n-1})$

(v_0,v_3)=>quadrangle$(v_0,v_1,v_2,,v_3)$ + (n-2)-gon$(v_0,v_3,v_4,\dots,v_{n-1})$

(v_0,v_4)=>pentagon$(v_0,v_1,v_2,,v_3,v_4)$ + (n-3)-gon$(v_0,v_4,v_5,\dots,v_{n-1})$

:

(v_0,v_{n-2})=>(n-1)-gon$(v_0,v_1,v_2,\dots,v_{n-2})$ + triangle(v_0,v_{n-2},v_{n-1})

Case 2: there is no chord incident to v_o.

(v_1,v_{n-1})=>triangle (v_0,v_1,v_{n-1}) + (n-1)-gon$(v_1,v_2,v_3,\dots,v_{n-1})$

Considering duplicate appearance of triangles,

$f(n) \leqq 3f(n-1) + 2f(n-2) + 2f(n-3) + \cdots + 2f(4) + 3f(3)$

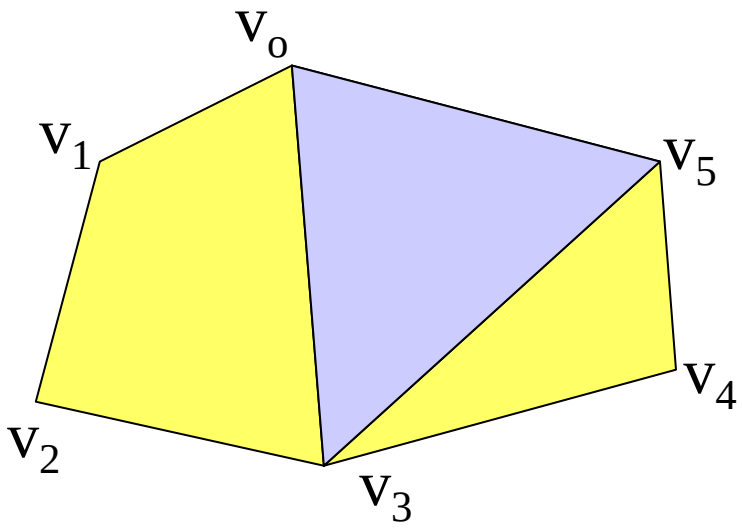
$f(3) = 1$.

$g(n)=2g(n-1)$, $g(3)=1$ then $g(n)=2^{n-3}$, thus f(n) is also exponential.

That is, this method does not lead to polynomial-time algorithm.

別の方法で再帰的に解を表現する.

凸多角形P(v_o, v_1, v_2, v_3, v_4, v_5)の辺(v_o, v_5)は必ずどれかの三角形に含まれなければならない. そのような三角形は(v_o, v_1, v_5),(v_o, v_2, v_5),(v_o, v_3, v_5),(v_o, v_4, v_5)の中の一つ.

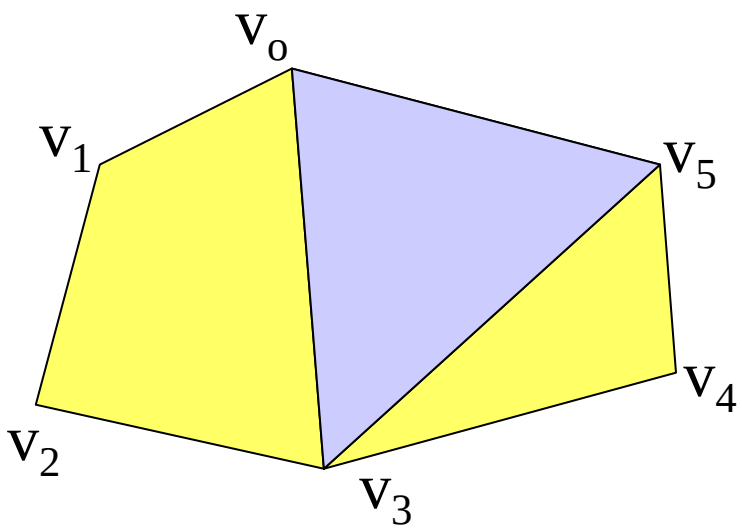


三角形(v_o, v_3, v_5)の場合,
残りの部分は2つの凸多角形に分割される.

一般に, 凸多角形P(v_o, ..., v_{n-1})を辺(v_o, v_{n-1})を含む三角形(v_o, v_k, v_{n-1})によって分割すると,
凸多角形P(v_o,v_1, ..., v_k)と凸多角形P(v_k,v_{k+1}, ..., v_{n-1})に分かれる.
これは, [0,n-1]という区間を[0,k]と[k,n-1]に分割することに対応.
よって, 分割の仕方は部分区間の数, $O(n^2)$通りしかない!

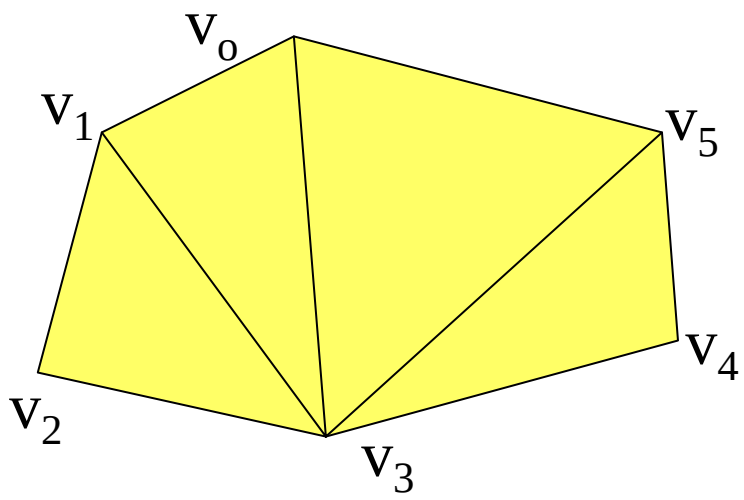
Recursive representation of a solution in a different way

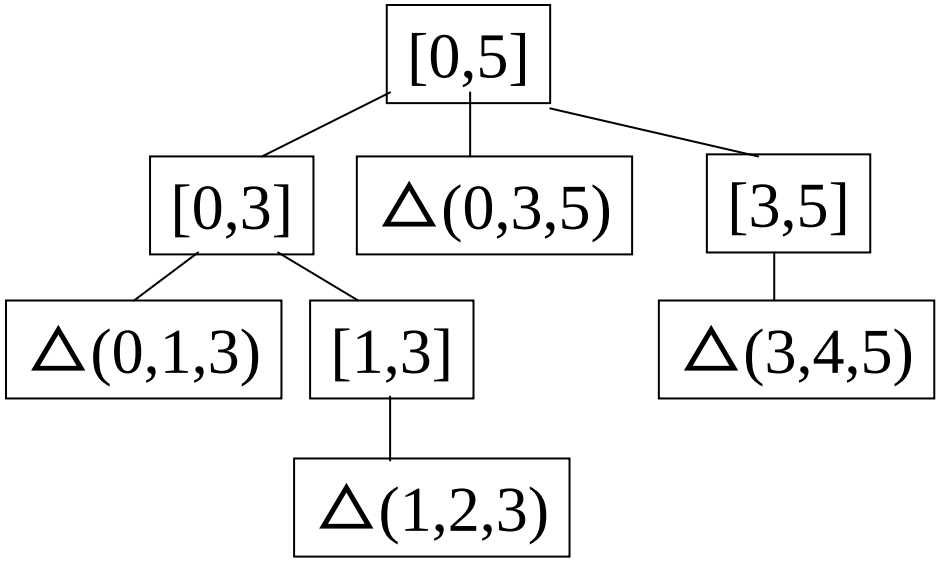
The edge (v_o, v_5) of the convex polygon $P(v_o, v_1, v_2, v_3, v_4, v_5)$ must be included in at least one triangle. Such a triangle is one of (v_o, v_1, v_5),(v_o, v_2, v_5),(v_o, v_3, v_5), and (v_o, v_4, v_5).

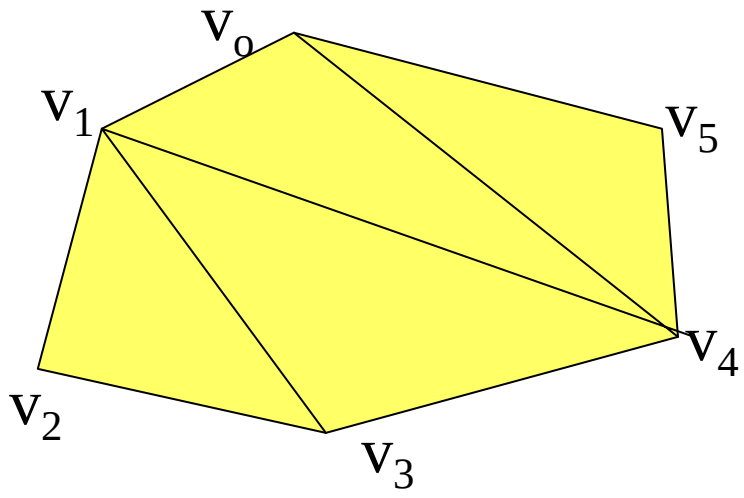


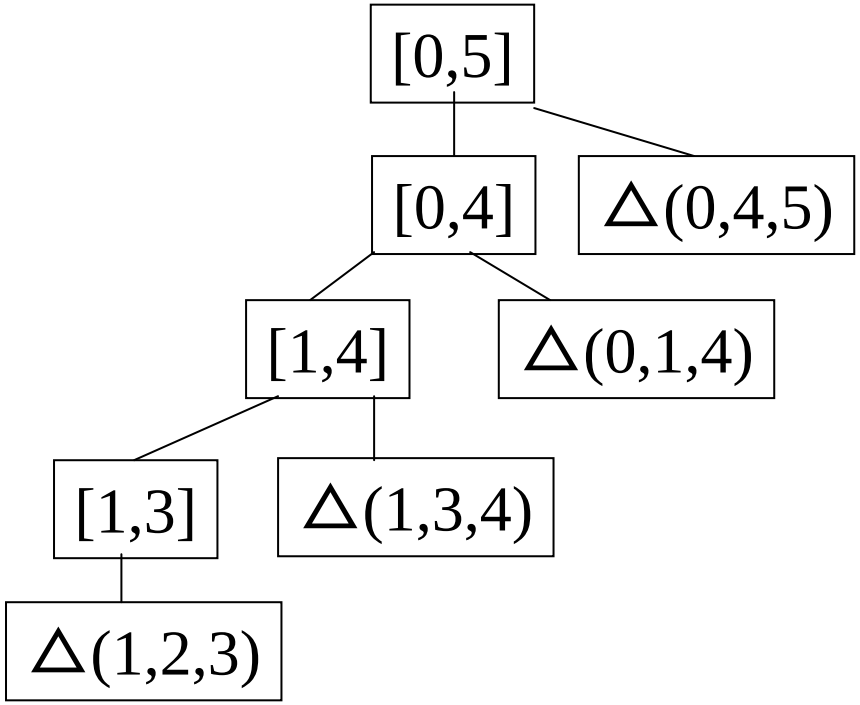
For the triangle (v_o, v_3, v_5), the remaining part is partitioned into two convex polygons.

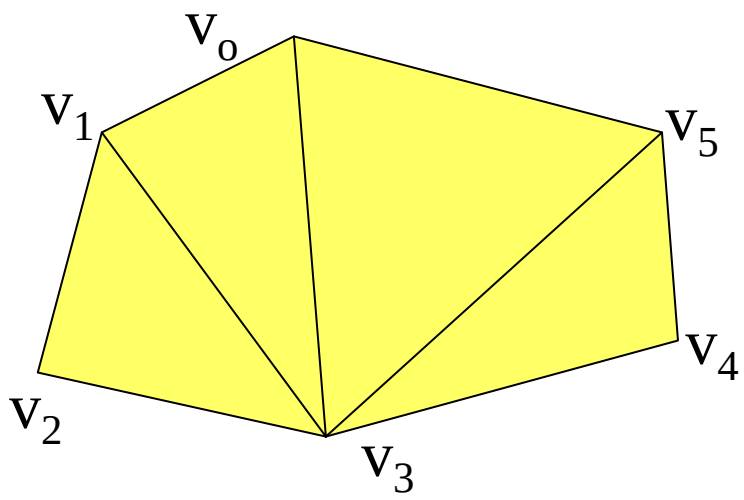
In general, when we partition the polygon $P(v_o, ..., v_{n-1})$ by a triangle containing the edge (v_o, v_{n-1}), we have two convex polygons: $P(v_o,v_1, ..., v_k)$ and $P(v_k,v_{k+1}, ..., v_{n-1})$. This corresponds to a partition of an interval [0,n-1] into [0,k] and [k,n-1]. Thus, the number of different partitions is that of different intervals, that is, $O(n^2)$.

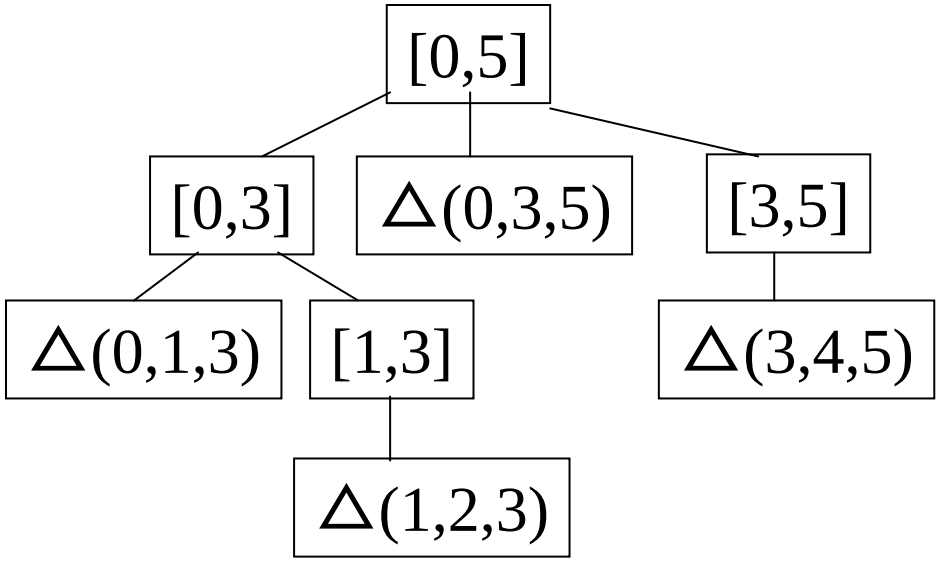

v_o
v_1
v_5
v_4
v_2
v_3

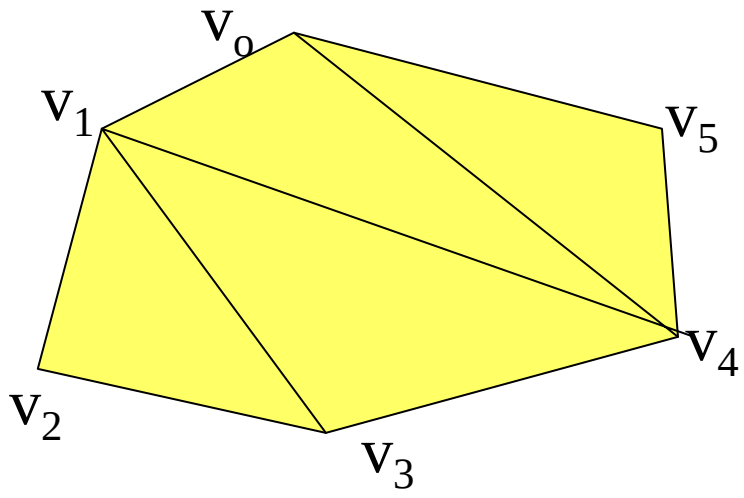

[0,5]
[0,3]
△(0,3,5)
[3,5]
△(0,1,3)
[1,3]
△(3,4,5)
△(1,2,3)

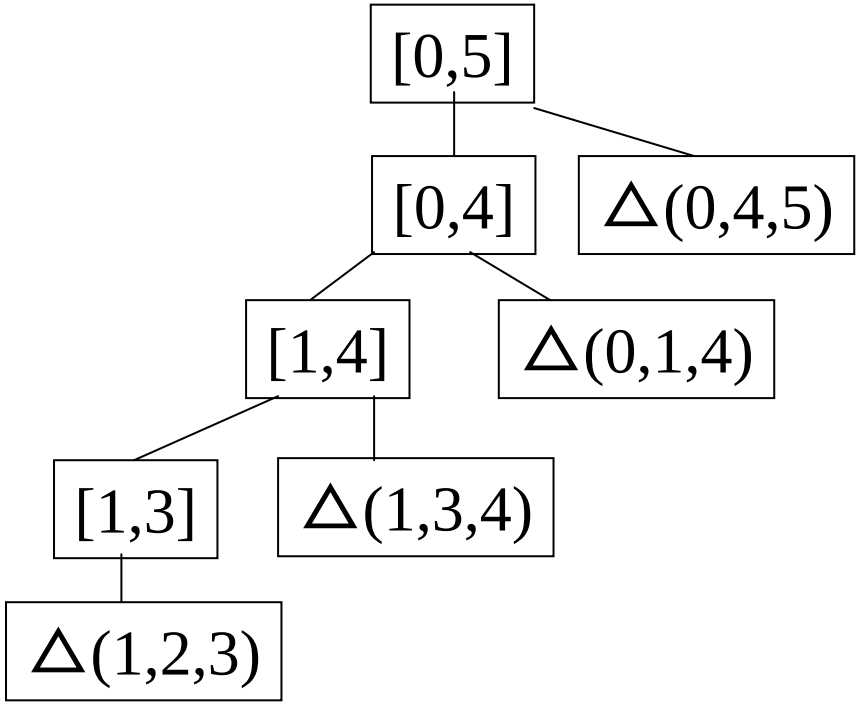

v_o
v_1
v_5
v_4
v_2
v_3


[0,5]
[0,4]
△(0,4,5)
[1,4]
△(0,1,4)
[1,3]
△(1,3,4)
△(1,2,3)

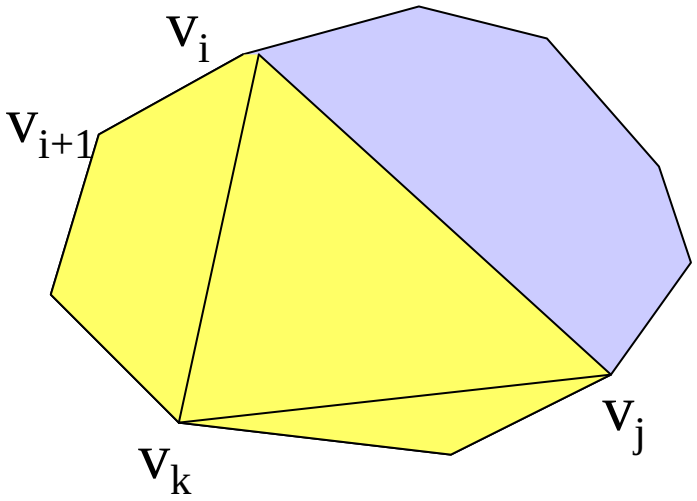

v_o
v_1
v_5
v_4
v_2
v_3


[0,5]
[0,3]
△(0,3,5)
[3,5]
△(0,1,3)
[1,3]
△(3,4,5)
△(1,2,3)


v_o
v_1
v_5
v_4
v_2
v_3


[0,5]
[0,4]
△(0,4,5)
[1,4]
△(0,1,4)
[1,3]
△(1,3,4)
△(1,2,3)

[0,n-1]の部分区間[i,j]に対応する多角形の分割

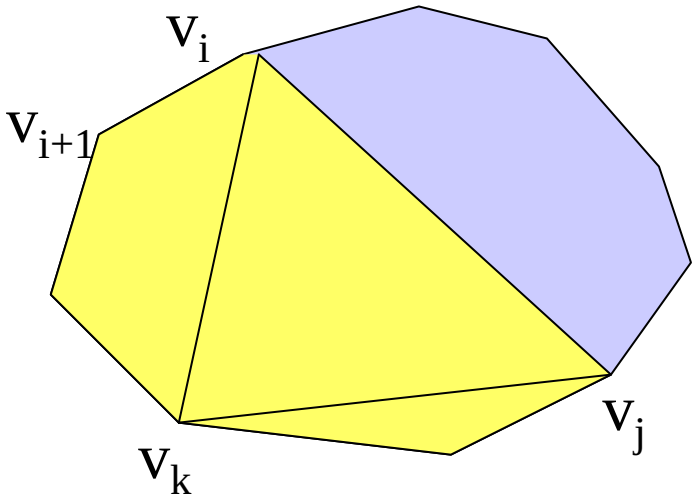


凸多角形$P(v_i, v_{i+1}, \ldots, v_j)$を
三角形(v_i, v_k, v_j),
凸多角形$(v_i, v_{i+1}, \ldots, v_k)$
凸多角形$(v_k, v_{k+1}, \ldots, v_j)$
に分割.
ただし, $k=i+1$のときと$k=j-1$のときは
三角形と残りの多角形の2つに分割.

$L[i,j]$ = 頂点列$(v_i, v_{i+1}, \ldots, v_j)$によって定まる部分多角形の内部に含まれる弦の長さの総和.
$w[i,j]$ = 頂点v_iと頂点v_jを結ぶ弦の長さ
とすると, 漸化式は

$$L[i,j] = \min\{ L[i+1,j]+w[i+1,j],$$
$$\min\{ L[i,k]+L[k,j]+w[i,k]+w[k,j] \mid k=i+2, \ldots, j-2\},$$
$$L[i,j-1]+w[i,j-1]\}$$

Partition of a polygon corresponding to a subinterval [i,j] of [0,n-1]



Partition a convex polygon $P(v_i, v_{i+1}, ..., v_j)$ into
  a triangle (v_i, v_k, v_j),
  a convex polygon$(v_i, v_{i+1}, ..., v_k)$ and
  a convex polygon$(v_k, v_{k+1}, ..., v_j)$,
if $k>i+1$ and $k<j-1$, and
  a triangle and a convex polygon
if $k=i+1$ or $k=j-1$.

L[i,j] = the sum of lengths of chords contained in the interior of the subpolygon determined by vertex sequence $(v_i, v_{i+1}, ..., v_j)$.
w[i,j] = the length of the chord between vertices v_i and v_j
Then, we have the following recurrence equation:

$$L[i,j] = \min\{ L[i+1,j]+w[i+1,j], \min\{ L[i,k]+L[k,j]+w[i,k]+w[k,j] \mid k=i+2, \ldots, j-2\}, L[i,j-1]+w[i,j-1]\}$$

アルゴリズムP24-A0:

弦の長さの総和を最小にする三角形分割

入力：凸多角形$P(v_0, v_1, \ldots, v_{n-1})$

出力：最適な三角形分割における弦の長さの総和

```
for(i=0; i<n; i++)
   for(j=i+2; j<n; j++){
      w[i,j] = 頂点v_iと頂点v_jを結ぶ弦の長さ;
      L[i,j] = 0;}
for(d=3; d<n; d++)
   for(i=0; i<n; i++)
      for(j=i+d; j<n; j++){
         msf = min( L[i+1,j]+w[i+1,j], L[i,j-1]+w[i,j-1]);
         for(k=i+2; k<=j-2; k++)
            if( L[i,k]+L[k,j]+w[i,k]+w[k,j] < msf)
               msf = L[i,k]+L[k,j]+w[i,k]+w[k,j];
      }
return L[0,n-1];
```

Algorithm P24-A0:

Triangulation to minimize the sum of lengths of chords

Input : convex polygon $P(v_0, v_1, ..., v_{n-1})$

Output : the sum of lengths of chords in an optimal triangulation

```
for(i=0; i<n; i++)
   for(j=i+2; j<n; j++){
      w[i,j] = the length of the chord between vertices v_i and v_j
L[i,j] = 0;}
for(d=3; d<n; d++)
   for(i=0; i<n; i++)
      for(j=i+d; j<n; j++){
         msf = min( L[i+1,j]+w[i+1,j], L[i,j-1]+w[i,j-1]);
         for(k=i+2; k<=j-2; k++)
            if( L[i,k]+L[k,j]+w[i,k]+w[k,j] < msf)
               msf = L[i,k]+L[k,j]+w[i,k]+w[k,j];
      }
return L[0,n-1];
```

演習問題E9-2：問題P24では弦の長さの総和を最小にする三角形分割を求めたが，弦の長さの2乗和を最小にする場合はどうか．

演習問題E9-3：各三角形の面積の2乗和を最小にする場合はどうか．

演習問題E9-4：各三角形の面積の和を最小にする場合はどうか．

演習問題E9-5：最適解の値だけではなく，最適解（最適三角形分割）そのものを求めるようにアルゴリズムを変更せよ．

Exercise E9-2: In Problem P24 we tried to find a triangulation to minimize the sum of chord lengths. What about the case where the sum of squared chord lengths is minimized?

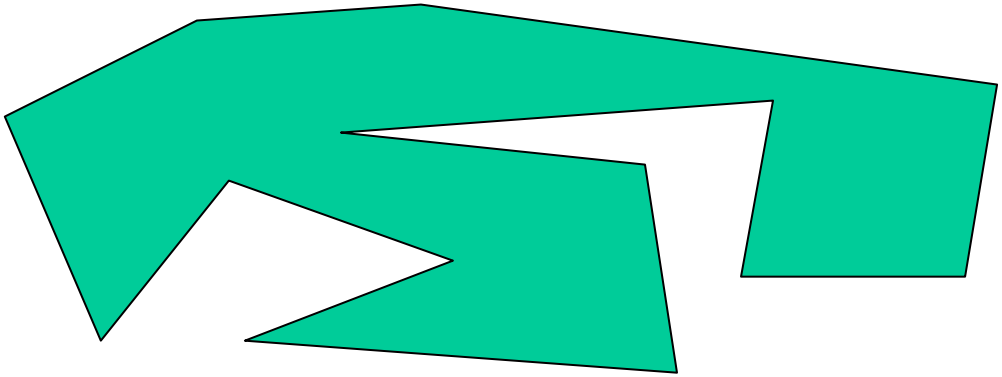
Exercise E9-3: What about the case where the sum of squared area of triangles?

Exercise E9-4: What about if we want to minimize the sum of area of triangles?

Exercise E9-5: Modify the algorithm so that not only the value of an optimal solution but also an optimal solution itself (optimal triangulation) is obtained.

凸多角形ではなく，一般の多角形の三角形分割の場合に拡張できるだろうか？

違いは，凸多角形の場合にはどの２点間にも弦を引けたが，一般の多角形では弦を引けないことがある．



w[i,j]の定義を変更する：

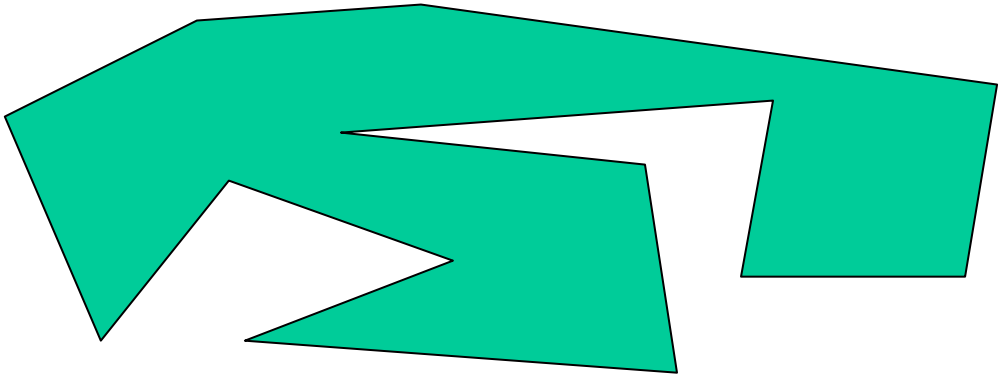
頂点v_iと頂点v_jを結ぶ線分が多角形の内部だけを通るときその長さをw[i,j]とし，そうでないときは，∞とする．

後は，まったく同じアルゴリズムで最適解を求めることができる．

演習問題**E9-6**：頂点v_iと頂点v_jを結ぶ線分が多角形の内部だけを通るかどうかを判定する方法を考えよ．

Is it possible to extend it to the case of triangulation of a general polygon instead of a convex polygon?

One difference is that we could connect any two vertices as chords in a convex polygon but a general polygon may not allow it.



Modify the definition of w[i,j] :

Only if the segment between vertices v_i and v_j is contained in the interior of the polygon, its length is defined to be w[i,j], Otherwise, w[i,j] is ∞.

Only with this modification we can find an optimal solution.

Exercise E9-6 : Devise a method for determining whether a segment between vertices v_i and v_j is contained in the interior of a polygon.