アルゴリズム論
Theory of Algorithms

第8回講義
動的計画法

アルゴリズム論
Theory of Algorithms

Lecture #8

Dynamic Programming

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)

組合せの数の計算

問題P20: n個の異なるものからk個取り出す組合せの数C(n,k)を計算せよ.

公式によると,

$C(n, k) = C(n-1, k-1) + C(n-1, k)$, $0<k<n$のとき,

$C(n, 0) = C(n, n) = 1$,

であるから，直ちに次のプログラムを得る.

```
int C(int n, int k){
  if(k==0 || k==n) return 1;
  return C(n-1, k-1) + C(n-1, k);
}
```

左のプログラムを実際に実行してみると，かなり時間がかかることがわかる.
時間がかかる原因は？

$$\left(\frac{n}{k}\right)^k \le \binom{n}{k} \le \left(\frac{ne}{k}\right)^k$$

計算時間の解析:

C(n,k)を計算するのに要する時間をT(n,k)とすると,

$T(n,k)=T(n-1,k-1)+T(n-1,k)$が成り立つ．よって，$T(n,k)=C(n,k)$.

これは指数関数時間.

Number of combinations

Problem P20: Calculate the number $C(n, k)$ of combinations to choose k items from n different items.

Using the formula,

$$C(n, k) = C(n-1, k-1) + C(n-1, k), \quad \text{if } 0<k<n,$$

$$C(n, 0) = C(n, n) = 1.$$

Therefore, we have the following program.

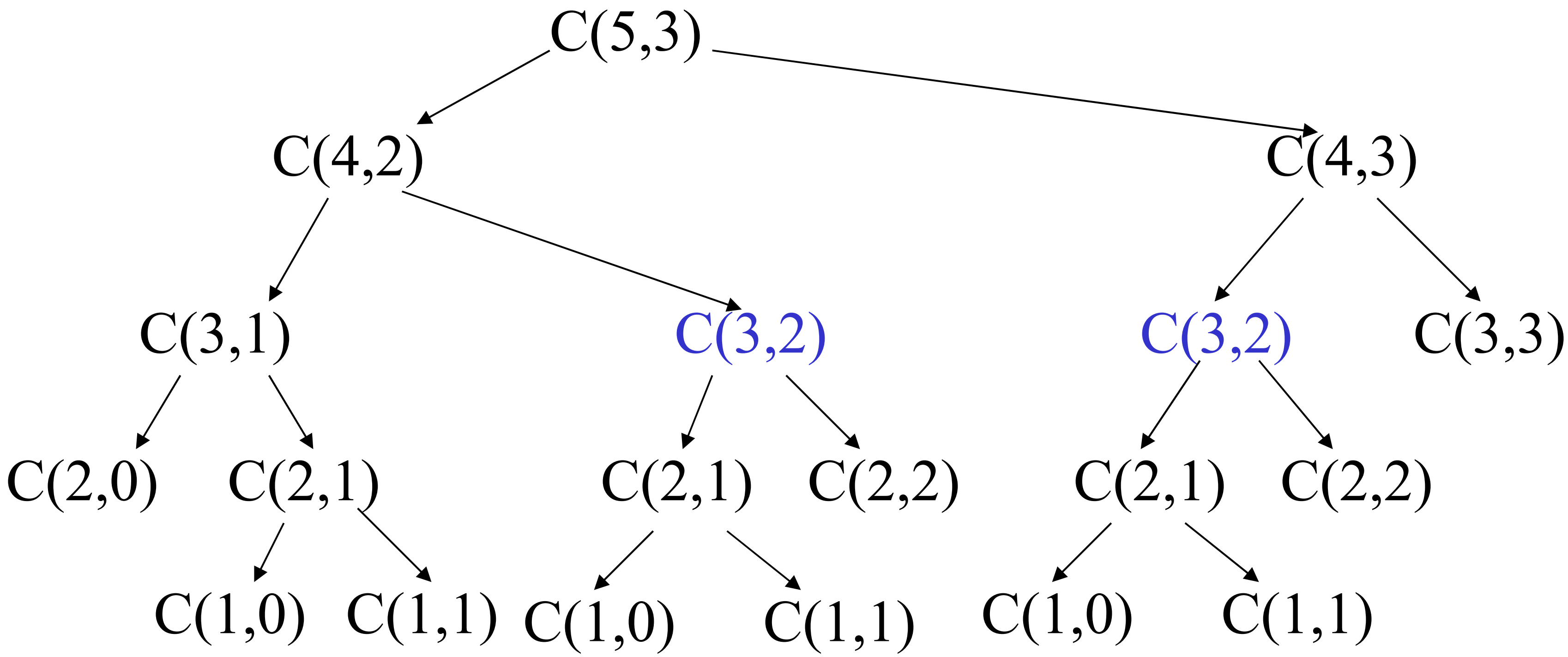
```
int C(int n, int k){
  if(k==0 || k==n) return 1;
  return C(n-1, k-1) + C(n-1, k);
}
```

When you implement the program in practice, you will find that it takes much time. Why does it take time?

Analysis of computation time:

Let $T(n,k)$ be time to compute $C(n,k)$. Then, we have $T(n,k)=T(n-1,k-1)+T(n-1,k)$. Thus, $T(n,k)=C(n,k)$. This is an exponential function.

プログラムの動作を調べてみよう！



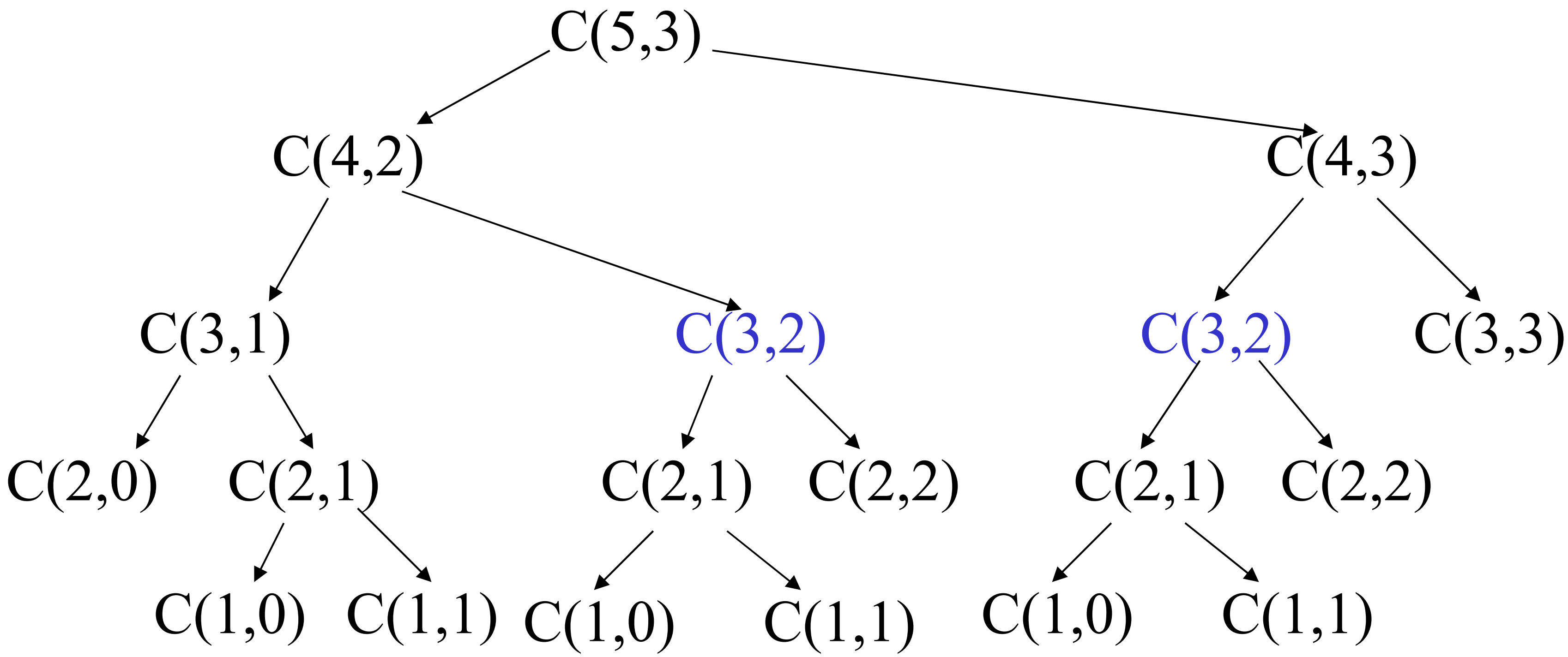
関数の呼び出しの様子

同じ値に対して関数が何度も呼び出されている.

例：C(3,2)は2回呼び出されている➔無駄

初めて(n,k)の組についてC(n,k)の値を計算するとき，その値を配列の(n,k)要素として蓄えておくと，同じ要素は2度と再計算しない． 要するに表を埋めればよい！

Let's analyze behavior of the program!



How is the function called

The function is called many times for the same value.

Example : C(3,2) is called twice. ➔redundant

If we store the value C(n,k) as the (n,k) element of an array when it is first computed, then the same value is never computed twice. Basically, it suffices to fill in the table.

C(n,k)の表を埋めよう！

公式：$C(n,k) = C(n-1,k-1) + C(n-1,k)$

n-1行目の値が分かっていれば，簡単にC(n,k)が計算可能

➔よって，1行目から順に埋めていけばよい.

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3						
4						
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4						
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$C(n-1,k-1)$ $C(n-1,k)$

$C(n,k)$

表の各要素は定数時間で計算可能.

よって，全体の計算時間は

$O(n^2)$時間

Fill in the table C(n,k)!

Formula : $C(n,k) = C(n-1,k-1) + C(n-1,k)$

If the values in the (n-1)-st row are available, C(n,k) is easily computed. ➔Thus, we should fill in the table from the 1st row.

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3						
4						
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4						
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5						

	0	1	2	3	4	5
0	1					
1	1	1				
2	1	2	1			
3	1	3	3	1		
4	1	4	6	4	1	
5	1	5	10	10	5	1

$$\begin{array}{cc} C(n-1,k-1) & C(n-1,k) \\ & \searrow \quad \downarrow \\ & C(n,k) \end{array}$$

Each element of the table can be computed in constant time.
Thus, the total time is
$O(n^2)$.

C言語のプログラムは下の通り:

```
int C(int n, int k){
   C[0][0]=1;
   for(i=1; i<=n; i++){
      C[i][0]=1; C[[i][i]=1;
      for(j=1; j<i; j++)
         C[i][j] = C[i-1][j-1] + C[i-1][j];
   }
   return C[n][k];
}
```

演習問題 E8-1: 素朴な方法2で、C(n,k)がオーバーフローしないならば途中でもオーバーフローしないような計算方法を考えてみよ。

素朴な方法2:

$$C(n,k) = \frac{n!}{(n-k)!\ k!}$$ であることを用いると, O(n)時間で計算可能.

ただし, この方法では整数のオーバーフローに注意.

C program is as follows:

```
int C(int n, int k){
  C[0][0]=1;
  for(i=1; i<=n; i++){
    C[i][0]=1; C[[i][i]=1;
    for(j=1; j<i; j++)
      C[i][j] = C[i-1][j-1] + C[i-1][j];
  }
  return C[n][k];
}
```

Exercise E8-1: Consider an algorithm that computes C(n,k) based on the Naïve idea such that it computes C(n,k) correctly if C(n,k) itself does not overflow.

Naive Algorithm 2:

Using the formula $C(n,k) = \dfrac{n!}{(n-k)!\ k!}$, it can be computed in O(n).

Here, note that this algorithm may suffer from numerical overflow.

フィボナッチ数の計算

問題P21: 次式で定義されるフィボナッチ数F(n)を求めよ.

$F(n) = F(n-1) + F(n-2)$, $n>1$のとき,

$F(0) = F(1) = 1$.

フィボナッチ数:

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

演習問題 E8-2: 問題P20と同様の議論を展開せよ.

フィボナッチ数F(n)は, 黄金比ϕを用いて,

$F(n) = O(\phi^n)$

と表すことができる.

$\phi=(1+\sqrt{5})/2 \fallingdotseq 1.61803$

Computation of Fibonacci number

Problem P21: Compute the Fibonacci number F(n) defined by

$F(n) = F(n-1) + F(n-2), \quad \text{if } n>1,$

$F(0) = F(1) = 1.$

Fibonacci number :

1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, ...

Exercise E8-2 : Have an argument similar to that in Problem P20.

Using the golden ration ϕ, the Fibonacci number F(n) can be represented as

$F(n) = O(\phi^n)$

where $\phi=(1+\sqrt{5})/2 \fallingdotseq 1.61803.$

最長共通部分文字列

問題P22: 長さnとmの2つの文字列AとBが与えられたとき，両方の文字列に共通する部分文字列で最長のものを求めよ.

例: A＝G A A T T C A G T T A , B= G G A T C G Aのとき, GATCAは共通部分文字列である.

A=　GAATTC　AGTTA

B=GGA　T　CGA

文字列Aの任意の部分文字列A'が文字列Bの部分文字列かどうかは，A'の文字が文字列Bにおいて同じ順に出現するかどうかを調べればよい. ➔線形時間で判定可能

演習問題 E8-3: 2つの文字列を入力して，最初の文字列が2番目の文字列の部分文字列になっているかどうかを線形時間で判定するプログラムを書け.

Longest Common Subsequence

Problem P22: Given two strings A and B of lengths n and m, find the longest substring common to both of them.

Example: For A = G A A T T C A G T T A and B= G G A T C G A, the longest common substring is GATCA.

A= GAATTC AGTTA

B=GGA T CGA

Any substring A' of A is a substring of B if characters of A' appear in the same order in the string B.

➔It can be determined in linear time.

Exercise E8-3: Write a program to determine whether the first string of two input strings is a substring of the second string in linear time.

アルゴリズムP22-A0:（腕力法）
文字列Aのすべての部分文字列A'について，A'が文字列Bの部分文字列になっているかどうかを確かめ，最も長い共通文字列を最後に出力する.

計算時間の解析:
- 長さnの文字列の部分文字列は全部で2^n通りある.
- この文字列が文字列Bより長いときは，明らかに文字列Bの部分文字列ではない.
- そうでないとき，それぞれに$O(m)$の時間がかかる.
- よって，全体の計算時間は，$O(2^n m)$時間となる.

高速化は可能か？
多項式時間のアルゴリズムは存在するか？

Algorithm P22-A0:（Brute-Force Algorithm）

For each substring A' of a string A, determine whether A' is a substring of a string B, and finally output the longest common substring.

Analysis of computation time：

- There are 2^n different substrings of a string of length n.
- If this substring is longer than the string B, obviously it is not a substring of B.
- Otherwise, each test takes O(m) time.
- Thus, the total time is $O(2^n m)$ time.

Is it possible to have faster algorithm?

Is there any polynomial-time algorithm?

アルゴリズムP22-A1:

$A= a_1a_2...a_n$, $B= b_1b_2... b_m$
$L[i,j] = a_1a_2...a_i$と$b_1b_2... b_j$の最長共通部分文字列の長さ

観察:
(0) $i=0$または$j=0$のとき, $L[i,j]=0$.
(1) $a_i = b_j \rightarrow L[i,j] = L[i-1,j-1]+1$
(2) $a_i \neq b_j \rightarrow L[i,j] = \max\{ L[i,j-1], L[i-1, j]\}$

A=GAATTC AGTTA
B= GGATC GA
$a_i = b_j$のとき

A=GAATTC AGTTA
B=GGATCG A
$a_i \neq b_j$のとき

したがって, 今度も表L[i,j]を順に埋めていけばよい！
表のサイズはn×mだから, 計算時間もO(nm)時間.

Algorithm P22-A1:

$A = a_1a_2...a_n$, $B = b_1b_2... b_m$

$L[i,j]$ = the length of the longest substring common to $a_1a_2...a_i$ and $b_1b_2... b_j$

Observation:

(0) if $i=0$ or $j=0$, $L[i,j]=0$.

(1) $a_i = b_j \rightarrow L[i,j] = L[i-1,j-1]+1$

(2) $a_i \neq b_j \rightarrow L[i,j] = \max\{ L[i,j-1], L[i-1, j]\}$

A=GAATTC AGTTA B= GGATC GA when $a_i = b_j$	A=GAATTC AGTTA B=GGATCG A when $a_i \neq b_j$

Therefore, it suffices to fill in the table $L[i,j]$ in order.

Since the table size is $n \times m$, it takes $O(nm)$ time.

アルゴリズムP22-A1:

```
for(i=0; i<=n; i++)
  L[i][0]=0;
for(j=0; j<=m; j++)
  L[0][ j] = 0;
for(i=1; i<=n; i++)
  for(j=1; j<=m; j++)
    if( a[i] == b[j]) L[i][j] = L[i-1][j-1]+1;
    else L[i][j] = max{ L[i][j-1], L[i-1][j] };
return L[n][m];
```

A=XYXXZXYZXY,
B=ZXZYYZXXYXXZ
のときの動作例

A= X Y XXZXYZXY,
B=ZXZYYZXXYXXZ

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3	3	3
4	0	0	1	1	2	2	2	3	4	4	4	4	4
5	0	1	1	2	2	2	3	3	4	4	4	4	5
6	0	1	2	2	2	2	3	4	4	4	5	5	5
7	0	1	2	2	3	3	3	4	4	5	5	5	5
8	0	1	2	3	3	3	4	4	4	5	5	5	6
9	0	1	2	3	3	3	4	4	5	5	6	6	6
10	0	1	2	3	4	4	4	5	5	6	6	6	6

Algorithm P22-A1:

```
for(i=0; i<=n; i++)
   L[i][0]=0;
for(j=0; j<=m; j++)
   L[0][ j] = 0;
for(i=1; i<=n; i++)
   for(j=1; j<=m; j++)
      if( a[i] == b[j]) L[i][j] = L[i-1][j-1]+1;
      else L[i][j] = max{ L[i][j-1], L[i-1][j] };
return L[n][m];
```

Example for the case
A=XYXXZXYZXY and
B=ZXZYYZXXYXXZ

A= X Y XXZXYZXY,
B=ZXZYYZXXYXXZ

	0	1	2	3	4	5	6	7	8	9	10	11	12
0	0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	1	1	1	1	1	1	1	1	1	1	1
2	0	0	1	1	2	2	2	2	2	2	2	2	2
3	0	0	1	1	2	2	2	3	3	3	3	3	3
4	0	0	1	1	2	2	2	3	4	4	4	4	4
5	0	1	1	2	2	2	3	3	4	4	4	4	5
6	0	1	2	2	2	2	3	4	4	4	5	5	5
7	0	1	2	2	3	3	3	4	4	5	5	5	5
8	0	1	2	3	3	3	4	4	4	5	5	5	6
9	0	1	2	3	3	3	4	4	5	5	6	6	6
10	0	1	2	3	4	4	4	5	5	6	6	6	6

最適解の構成

最適解の値は表を埋めることにより求められるが，その値を達成する最適解はどのように構成すればよいか？

最長共通部分文字列の問題では最長の共通部分文字列の長さ（最適解の値）だけでなく，文字列そのもの（最適解）も求めたい．

表を埋めるとき，L[i][j]の値が同じ表のどの値によって決まったかを記録する:

(1) $a_i = b_j \rightarrow L[i,j] = L[i-1,j-1]+1$
(i-1, j-1)を記憶
(2) $a_i \neq b_j \rightarrow L[i,j] = \max\{ L[i,j-1], L[i-1, j]\}$
L[i,j-1]>L[i-1,j]のとき，(i, j-1)を記憶，
そうでないとき，(i-1, j)を記憶

Construction of an optimal solution

The value of an optimal solution is obtained by filling in the table. How can we construct an optimal solution achieving the value?

In the problem of finding the longest common substring, we want to find not only the length (value of an optimal solution) but also the longest such substring (optimal solution) itself.

When we fill in the table, we memorize which table element determined L[i][j].

(1) $a_i = b_j \rightarrow L[i,j] = L[i-1,j-1]+1$
(i-1, j-1) is memorized

(2) $a_i \neq b_j \rightarrow L[i,j] = \max\{ L[i,j-1], L[i-1, j]\}$
if $L[i,j-1] > L[i-1,j]$ then (i, j-1) is memorized, and
otherwise, (i-1, j) is memorized.

具体的なプログラム

```
for(i=1; i<n; i++)
  for(j=1; j<m; j++){
    if( A[i] == B[j] ){
      L[i][j] = L[i-1][j-1] + 1;
      B1[i][j] = i-1; B2[i][j] = j-1;
    } else {
      L[i][j] = max2(L[i][j-1], L[i-1][j]);
      if( L[i][j-1] > L[i-1][j] ){
        L[i][j] = L[i][j-1];
        B1[i][j] = B1[i][j-1]; B2[i][j] = B2[i][j-1];
      } else {
        L[i][j] = L[i-1][j];
        B1[i][j] = B1[i-1][j]; B2[i][j] = B2[i-1][j];
      }
    }
  }
```

演習問題 E8-4:実際にプログラムを作って動作を確認せよ.

Concrete program

```
for(i=1; i<n; i++)
  for(j=1; j<m; j++){
    if( A[i] == B[j] ){
      L[i][j] = L[i-1][j-1] + 1;
      B1[i][j] = i-1; B2[i][j] = j-1;
    } else {
      L[i][j] = max2(L[i][j-1], L[i-1][j]);
      if( L[i][j-1] > L[i-1][j] ){
        L[i][j] = L[i][j-1];
        B1[i][j] = B1[i][j-1]; B2[i][j] = B2[i][j-1];
      } else {
        L[i][j] = L[i-1][j];
        B1[i][j] = B1[i-1][j]; B2[i][j] = B2[i-1][j];
      }
    }
  }
```

Exercise E8-4: Write a program in practice to see its behavior.

A=XYXXZXYZXY，B=ZXZYYZXXYXXZの場合

バックトラック用の表

	1	2	3	4	5	6	7	8	9	10	11	12
1	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)	(0,6)	(0,7)	(0,7)	(0,9)	(0,10)	(0,10)
2	(0,0)	(0,1)	(0,1)	(1,3)	(1,4)	(1,4)	(1,4)	(1,4)	(1,8)	(1,8)	(1,8)	(1,8)
3	(0,0)	(2,1)	(0,1)	(1,3)	(1,4)	(1,4)	(2,6)	(2,7)	(2,7)	(2,9)	(2,10)	(2,10)
4	(0,0)	(3,1)	(0,1)	(1,3)	(1,4)	(1,4)	(3,6)	(3,7)	(3,7)	(3,9)	(3,10)	(3,10)
5	(4,0)	(3,1)	(4,2)	(1,3)	(1,4)	(4,5)	(3,6)	(3,7)	(3,7)	(3,9)	(3,10)	(4,11)
6	(4,0)	(5,1)	(4,2)	(1,3)	(1,4)	(4,5)	(5,6)	(5,7)	(3,7)	(5,9)	(5,10)	(4,11)
7	(4,0)	(5,1)	(4,2)	(6,3)	(6,4)	(4,5)	(5,6)	(5,7)	(6,8)	(5,9)	(5,10)	(4,11)
8	(7,0)	(5,1)	(7,2)	(6,3)	(6,4)	(7,5)	(5,6)	(5,7)	(6,8)	(5,9)	(5,10)	(7,11)
9	(7,0)	(8,1)	(7,2)	(6,3)	(6,4)	(7,5)	(8,6)	(8,7)	(6,8)	(8,9)	(8,10)	(7,11)
10	(7,0)	(8,1)	(7,2)	(9,3)	(9,4)	(7,5)	(8,6)	(8,7)	(9,8)	(8,9)	(8,10)	(7,11)

L[10][12]から逆に辿ると

L[10][12] →L[7][11]→L[5][10]→L[3][9]→L[2][7]→L[1][4]→L[0][1]

a[8]b[12]　a[6]b[11]　a[4]b[10]　a[3]b[8]　a[2]b[5]　a[1]b[2]

ゆえに，最長共通部分文字列は

123456789A　123456789ABC

XYXXZXYZXY　ZXZYYZXXYXXZ　　XYXXXZ

For the case: A=XYXXZXYZXY, B=ZXZYYZXXYXXZ

Table for backtrack

	1	2	3	4	5	6	7	8	9	10	11	12
1	(0,0)	(0,1)	(0,1)	(0,1)	(0,1)	(0,1)	(0,6)	(0,7)	(0,7)	(0,9)	(0,10)	(0,10)
2	(0,0)	(0,1)	(0,1)	(1,3)	(1,4)	(1,4)	(1,4)	(1,4)	(1,8)	(1,8)	(1,8)	(1,8)
3	(0,0)	(2,1)	(0,1)	(1,3)	(1,4)	(1,4)	(2,6)	(2,7)	(2,7)	(2,9)	(2,10)	(2,10)
4	(0,0)	(3,1)	(0,1)	(1,3)	(1,4)	(1,4)	(3,6)	(3,7)	(3,7)	(3,9)	(3,10)	(3,10)
5	(4,0)	(3,1)	(4,2)	(1,3)	(1,4)	(4,5)	(3,6)	(3,7)	(3,7)	(3,9)	(3,10)	(4,11)
6	(4,0)	(5,1)	(4,2)	(1,3)	(1,4)	(4,5)	(5,6)	(5,7)	(3,7)	(5,9)	(5,10)	(4,11)
7	(4,0)	(5,1)	(4,2)	(6,3)	(6,4)	(4,5)	(5,6)	(5,7)	(6,8)	(5,9)	(5,10)	(4,11)
8	(7,0)	(5,1)	(7,2)	(6,3)	(6,4)	(7,5)	(5,6)	(5,7)	(6,8)	(5,9)	(5,10)	(7,11)
9	(7,0)	(8,1)	(7,2)	(6,3)	(6,4)	(7,5)	(8,6)	(8,7)	(6,8)	(8,9)	(8,10)	(7,11)
10	(7,0)	(8,1)	(7,2)	(9,3)	(9,4)	(7,5)	(8,6)	(8,7)	(9,8)	(8,9)	(8,10)	(7,11)

If we trace the table from L[10][12] in reverse order,

L[10][12] →L[7][11]→L[5][10]→L[3][9]→L[2][7]→L[1][4]→L[0][1]

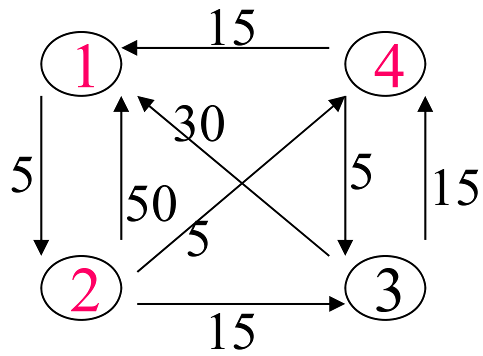
a[8]b[12] a[6]b[11] a[4]b[10] a[3]b[8] a[2]b[5] a[1]b[2]

Thus, the longest common substring is

123456789A 123456789ABC

XYXXZXYZXY ZXZYYZXXYXXZ XYXXXZ

問題P23:（全点対間最短経路問題）
重みつきのグラフが与えられたとき，全ての頂点対について
それらの間の最短経路の長さを求めよ．



$$L=\begin{array}{c c} & \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{array}{c} 1 \\ 2 \\ 3 \\ 4 \end{array} & \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

距離行列

ダイクストラ法

入力：n個の頂点とm本の辺からなる重みつきグラフ

出力：任意の１点から残りのすべての頂点への距離

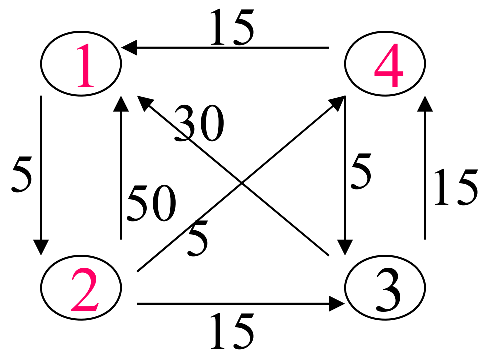
計算時間：$O(m + n \log n)$時間，または$O(n^2)$時間

したがって，各頂点を始点としてダイクストラ法を適用すると，
計算時間は$O(nm + n^2 \log n)$または$O(n^3)$となる．
辺の本数mは$O(n^2)$になるから，最悪の場合は，$O(n^3)$である．

高速化は可能か？

Problem P23: (All-pairs shortest path problem)

Given a weighted graph, for every pair of vertices find the length of a shortest path between them.



$$L = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

distance matrix

Dijkstra's algorithm

Input: Weighted graph consisting of n vertices and m edges

Output: Distances to all the vertices from one arbitrary vertex.

Computation time: $O(m + n \log n)$ time, or $O(n^2)$ time.

Hence, applying the Dijkstra's algorithm for each vertex, computation time is $O(nm + n^2 \log n)$ or $O(n^3)$.

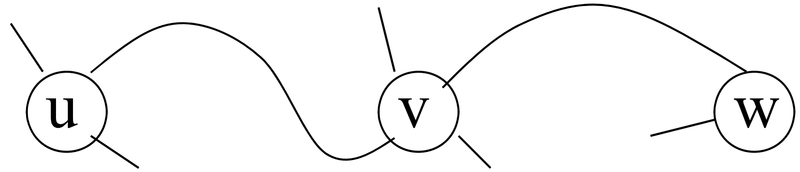
Since the number m of edges is $O(n^2)$, it takes $O(n^3)$ in the worst case.

Any faster algorithm?

最短経路の性質

v ： 頂点uから頂点wまでの最短経路上の任意の頂点

➔ uからvまでの部分経路と，vからwまでの部分経路は それぞれ，uからvまでと，vからwまでの最短経路である．

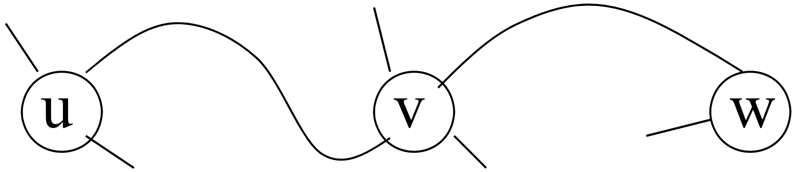


理由：uからvまでの部分経路が最短でなければ，この部分経路を最短経路で置きかえると，uからwへのより短い経路が得られる．これは矛盾．

Property of a shortest path

v : Any vertex on a shortest path from vertex u to vertex w.

➔ subpath from u to v and subpath from v to w are both shortest paths from u to v and from v to w, respectively.



Why: If the subpath from u to v is not shortest,
we can shorten the path from u to w by replacing its subpath by the shorter one, which is a contradiction.

$D_k[i,j]$ = 集合{1,2,...,k}に属する頂点だけを経由して頂点iから頂点jに至る最短経路の長さ

ケース1：最短経路が頂点kを経由しない➔ $D_{k-1}[i,j]$と同じ
ケース2：最短経路が頂点kを経由する
➔iからkまでの最短経路＋kからjまでの最短経路

$$D_k[i,j] = \min\{ D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] \}$$

ただし，$D_0[i,j] = L[i,j]$　（直接の距離=辺の長さ）

$D_0, D_1, D_2, \ldots, D_n$ の順に計算を行えばよい.

アルゴリズムP23-A0:
距離行列をD0とする.
for(p=1; p<=n; p++)
　すべての(i,j)について
　　D[p,i,j] = min{ D[p-1,i,j], D[p-1,i,k]+D[p-1,k,j] }

$D_k[i,j]$ = the length of a shortest path from vertex *i* to vertex *j* only through the vertices in the set {1,2,...,k}.

Case 1 : if the shortest path does not pass through vertex k
➔ same as **$D_{k-1}[i,j]$**

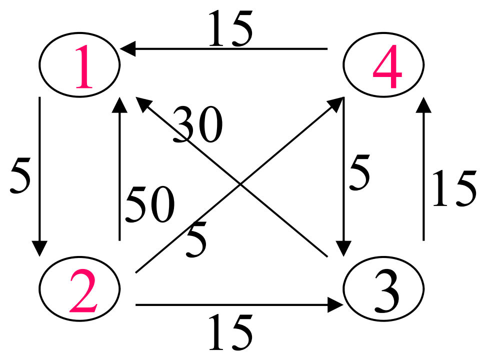
Case 2: if the shortest path passes through vertex k
➔shortest path from i to k＋that from k to j

$D_k[i,j] = \min\{ D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] \}$
where, $D_0[i,j] = L[i,j]$ （direct distance=edge length）

$D_0, D_1, D_2, \ldots, D_n$ should be computed in this order.

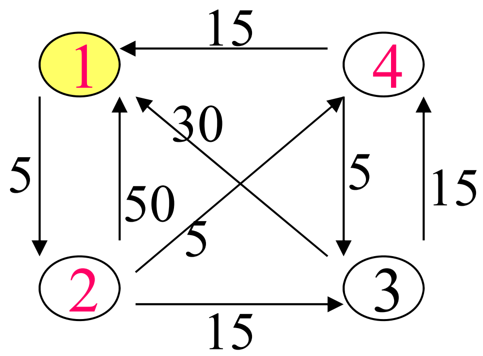
Algorithm P23-A0:

```
Let D0 be the distance matrix.
for(p=1; p<=n; p++)
   for each (i,j)
        D[p,i,j] = min{ D[p-1,i,j], D[p-1,i,k]+D[p-1,k,j] }
```



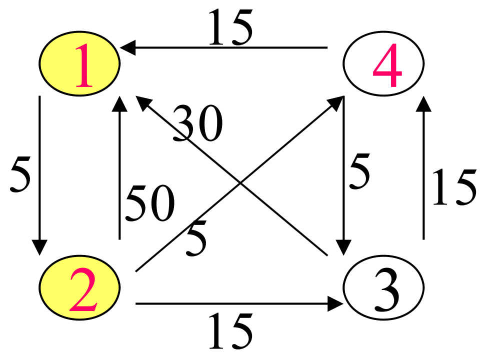
$$D_0=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \left(\begin{array}{cccc} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{array}\right) \end{array}$$

距離行列



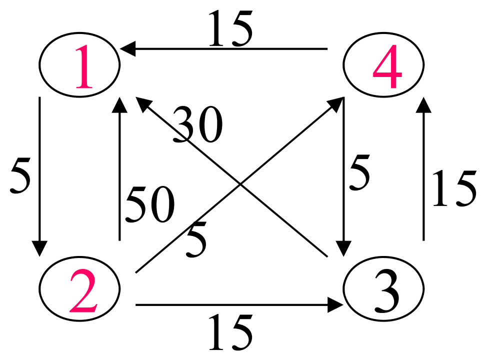
$$D_1=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \left(\begin{array}{cccc} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \color{red}{35} & 0 & 15 \\ 15 & \color{red}{20} & 5 & 0 \end{array}\right) \end{array}$$

頂点1を途中に
通ってもよい.



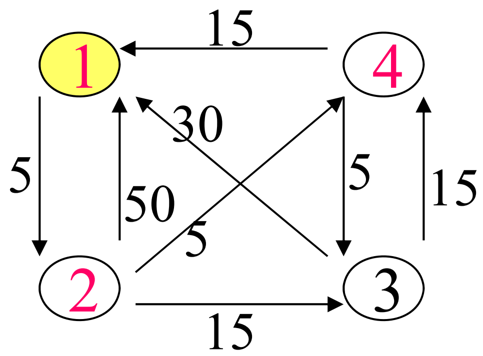
$$D_2=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \left(\begin{array}{cccc} 0 & 5 & \color{red}{20} & \color{red}{10} \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{array}\right) \end{array}$$

頂点1，2を途中に
通ってもよい.



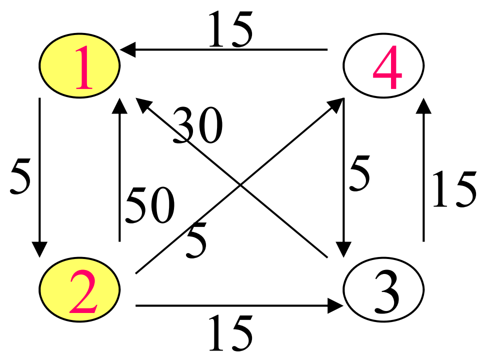
$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

distance matrix



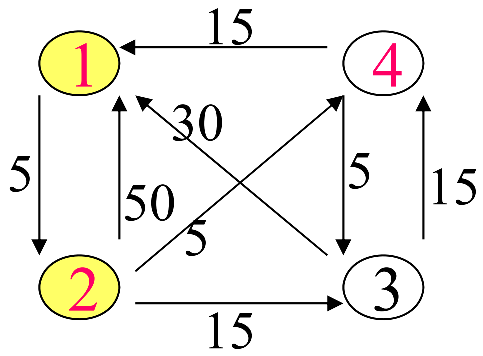
$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertex 1 can be passed through



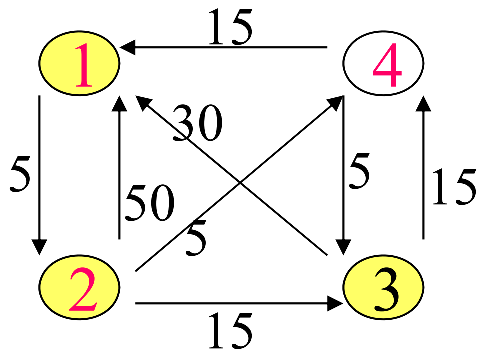
$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertices 1 and 2 can be passed through



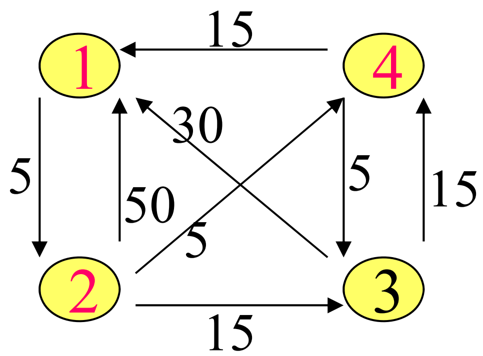
$$D_2=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{pmatrix} 0&5&20&10\\ 50&0&15&5\\ 30&35&0&15\\ 15&20&5&0 \end{pmatrix}\end{array}$$

頂点1，2を途中に通ってもよい．



$$D_3=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{pmatrix} 0&5&20&10\\ 45&0&15&5\\ 30&35&0&15\\ 15&20&5&0 \end{pmatrix}\end{array}$$

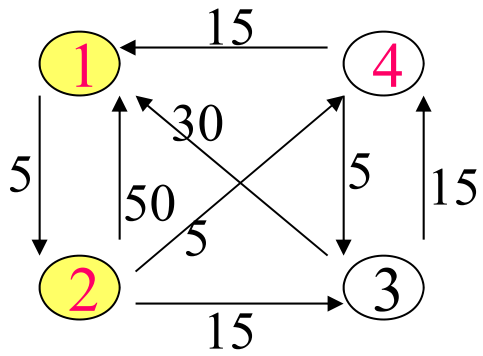
頂点1，2，3を途中に通ってもよい．



$$D_4=\begin{array}{c} \\ 1\\2\\3\\4 \end{array}\begin{array}{c} \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{pmatrix} 0&5&15&10\\ 20&0&10&5\\ 30&35&0&15\\ 15&20&5&0 \end{pmatrix}\end{array}$$

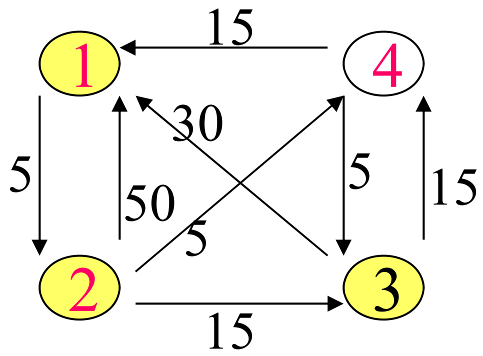
頂点1，2，3，4を途中に通ってもよい

最終的な距離行列



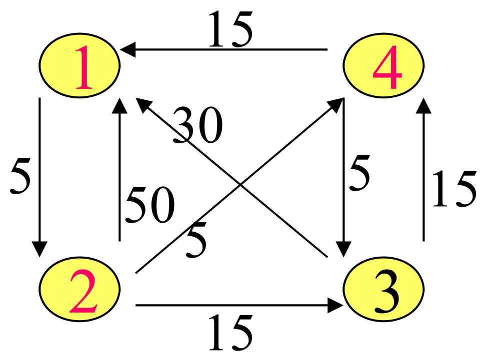
$$D_2=\begin{array}{cc} & \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{pmatrix}0&5&20&10\\50&0&15&5\\30&35&0&15\\15&20&5&0\end{pmatrix}\end{array}$$

vertices 1 and 2 can be passed through



$$D_3=\begin{array}{cc} & \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{pmatrix}0&5&20&10\\45&0&15&5\\30&35&0&15\\15&20&5&0\end{pmatrix}\end{array}$$

vertices 1, 2, and 3 can be passed through



$$D_4=\begin{array}{cc} & \begin{array}{cccc}1&2&3&4\end{array}\\ \begin{array}{c}1\\2\\3\\4\end{array} & \begin{pmatrix}0&5&15&10\\20&0&10&5\\30&35&0&15\\15&20&5&0\end{pmatrix}\end{array}$$

vertices 1, 2, 3, and 4 can be passed through

Final distance matrix

記憶スペースの問題

先の方法では3次元の配列が必要になる.
➔D_kの計算が終わったときD_{k-1}は必要がない.
よって，１つの配列だけで十分.

最短経路の長さだけでなく，最短経路も求めたい

$P_k[i,j] = \{1, 2, \ldots , k\}$を経由するiからjへの最短経路上で頂点jの直前の頂点の番号

これを用いると，終点から始点まで経路を戻ることが可能.

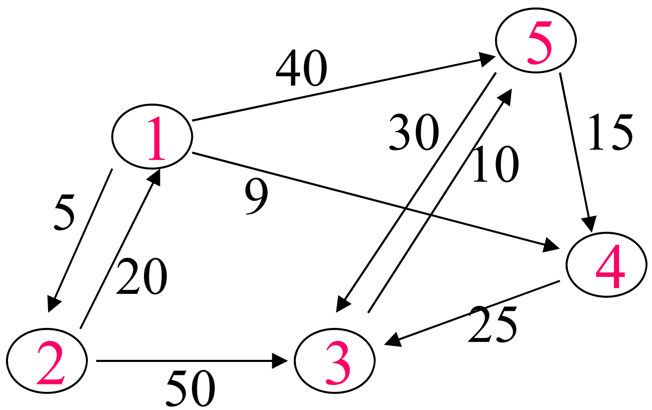
Problem on Space Requirement

The previous algorithm requires a 3-dimensional array.
➔when we have computed D_k, we don't need to store D_{k-1}.
Hence, only one array is sufficient.

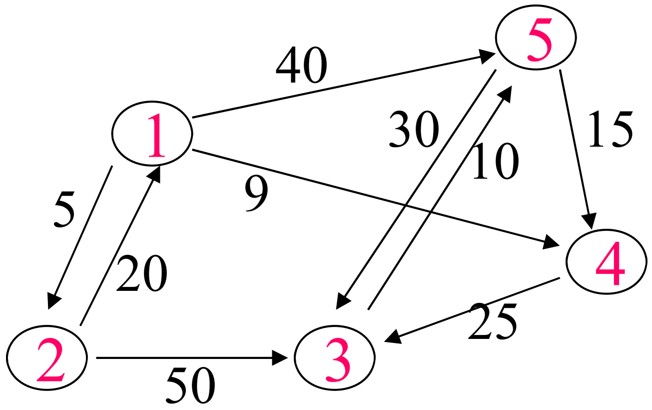
We want to compute not only the lengths of shortest paths but also shortest paths themselves.

$P_k[i,j]$ = the vertex number immediately before the vertex j on the shortest path from i to j through {1, 2, ... , k}.
Using it, we can trace back from the terminal vertex to the initial vertex.

演習問題 E8-5: 下のグラフについてアルゴリズムP23-A0の動作を記述せよ.



Exercise E8-5：Describe the behavior of the algorithm P23-A0 for the graph below.



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： $S = \{a_1, a_2, \dots, a_n\}, a_1 \leqq a_2 \leqq \cdots \leqq a_n$
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：

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

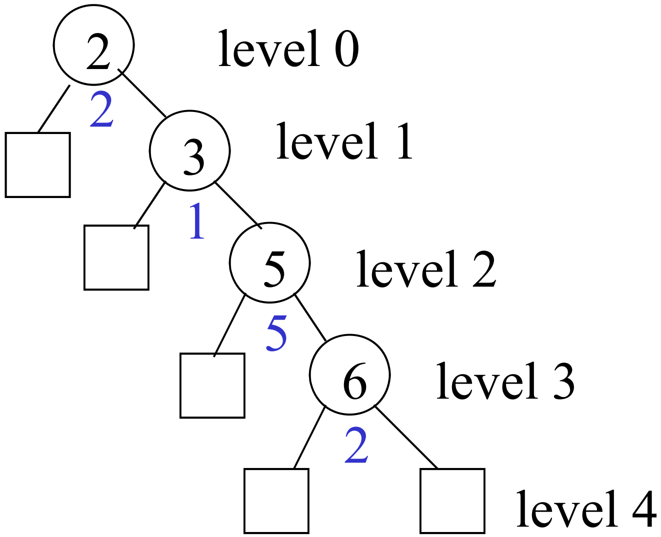
Problem P24: (Construction of an optimal binary search tree)
When probability that each element is asked is given, store n data in a binary search tree so that the expected number of comparisons to locate a query in the tree is minimized.

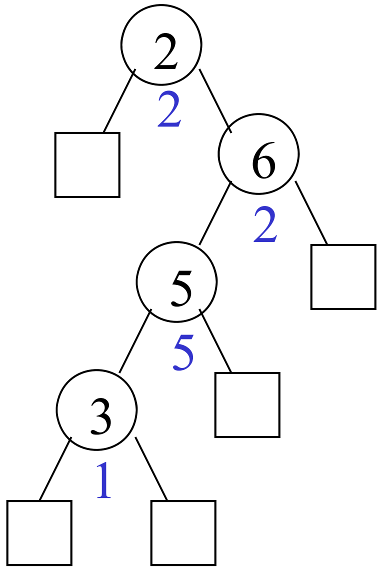
Data to be stored: $S = \{a_1, a_2, \ldots, a_n\}$, $a_1 \leqq a_2 \leqq \cdots \leqq a_n$
A priori knowledge: Assume that only elements of S are retrieved.
probability for Find(a_i, S) is p_i
When S is stored in a binary search tree,
let the level of a node a_i containing an element of S be level(a_i).
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



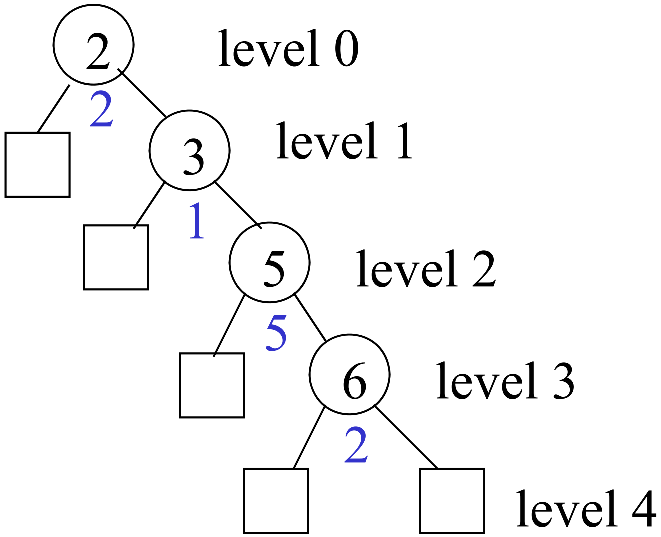


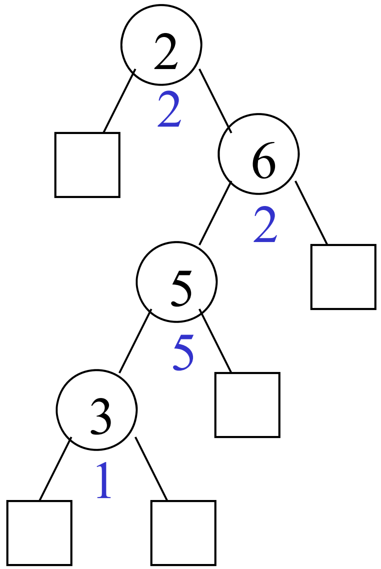
コスト=(2*1+1*2+5*3+2*4)/10
=2.7

コスト=(2*1+2*2+5*3+1*4)/10
= 2.5

Examlpe:

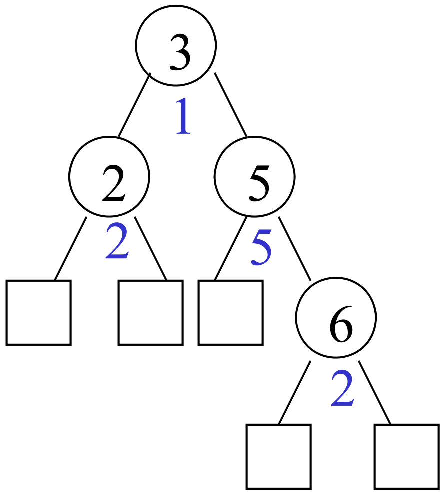
	a[1]	a[2]	a[3]	a[4]
S	2	3	5	6
	2/10	1/10	5/10	2/10
	p_1	p_2	p_3	p_4

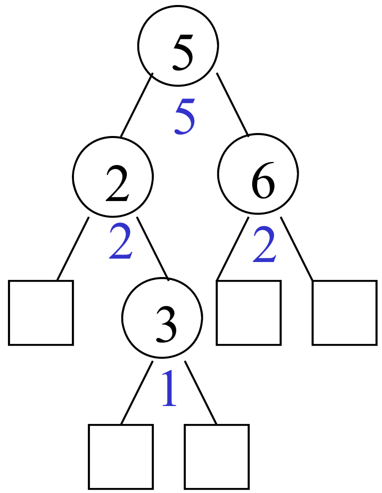




cost=(2*1+1*2+5*3+2*4)/10
=2.7

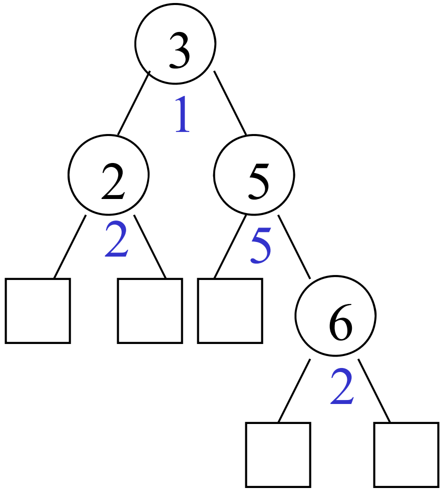
cost=(2*1+2*2+5*3+1*4)/10
= 2.5



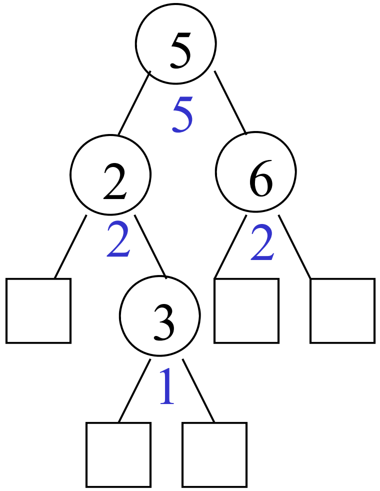


コスト=(1*1+(2+5)*2+2*3)/10 =2.1　　コスト=(5*1+(2+2)*2+1*3)/10 = 1.6

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

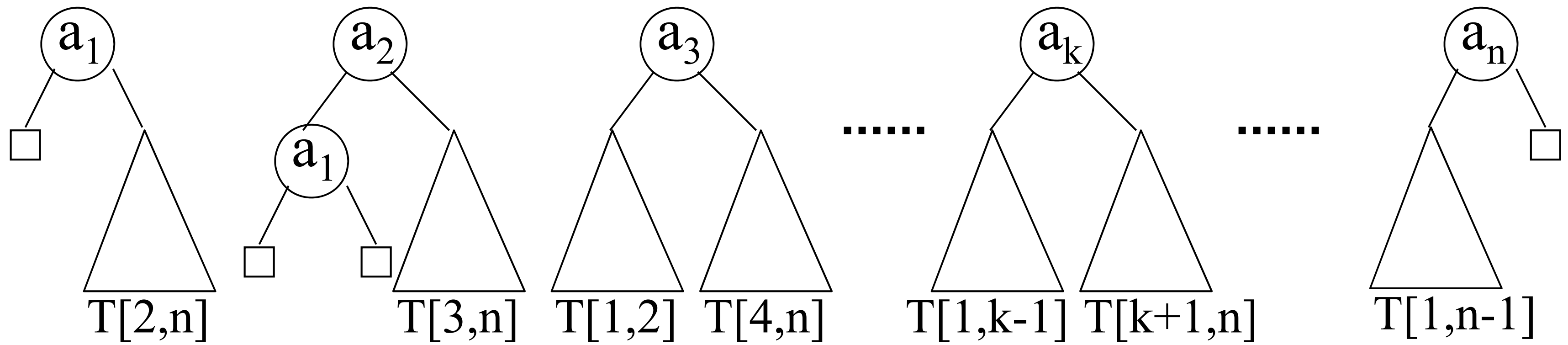
If we enumerate all search trees and compute their costs, then we can find an optimal search tree. But it is not efficient to enumerate all of them.

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
$i=1, \ldots, n,\ j=i, i+1, \ldots, n$

すべての可能性を列挙すると



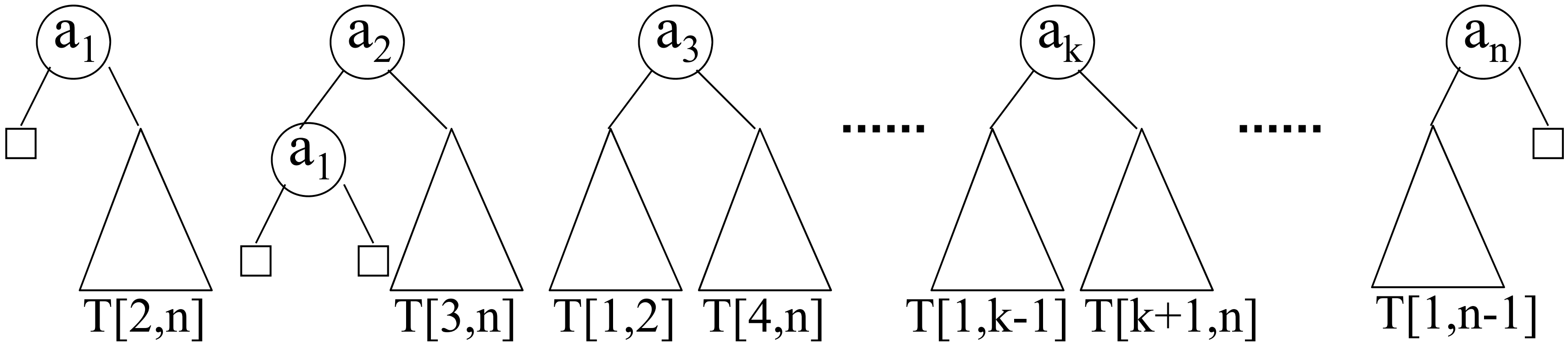
T[2,n], T[3,n], ... , T[1,2], T[4,n], ... , T[1,n-1]がすべて
求まっていれば，上のそれぞれの木のコストを計算できる．
最小コストの木を選べば，その根a_kを決めることが可能．

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 $\{a_i, a_{i+1}, \ldots, a_j\}$
$i=1, \ldots, n, \quad j=i, i+1, \ldots, n$

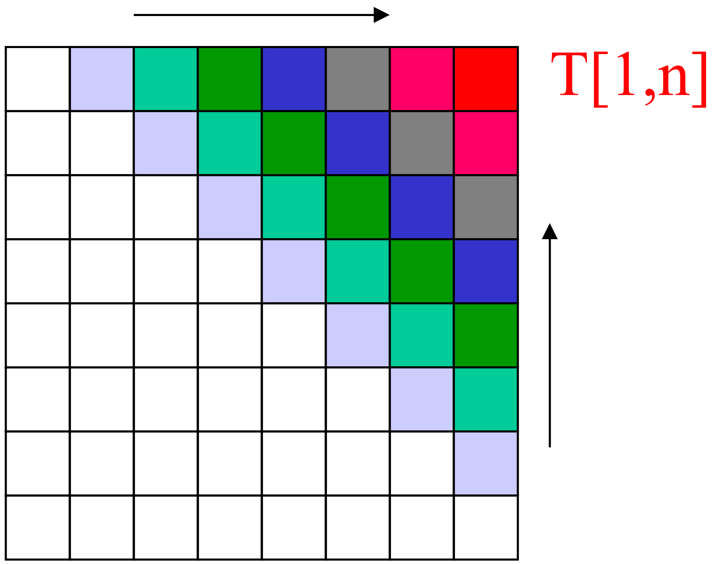
Enumerating all possibilities:



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.

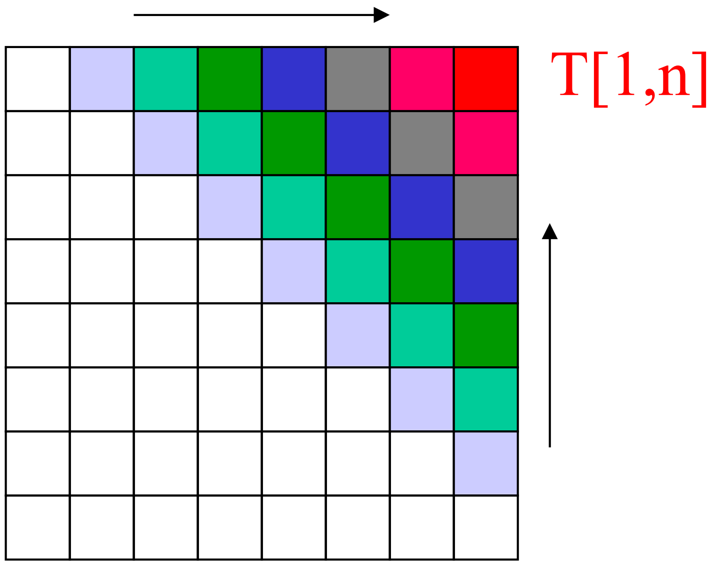
ボトムアップの形式で最適解の値を求める.

{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.



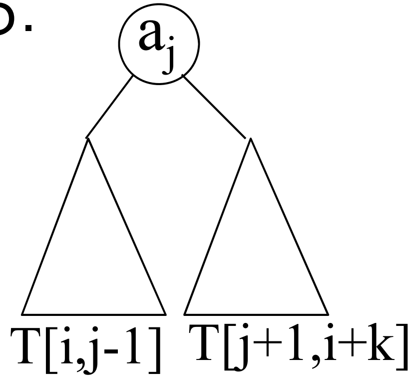
T[i, i+k]の求め方

T[i,i+k] = 部分集合$\{a_i, a_{i+1}, \dots , a_{i+k}\}$に対する最小コスト木だから，その根は，$a_i, a_{i+1}, \dots , a_{i+k}$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．



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

レベルを1だけ増やすと，

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

つまり，T[i,j-1]に$p_i + p_{i+1} +\dots + p_{j-1}$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：

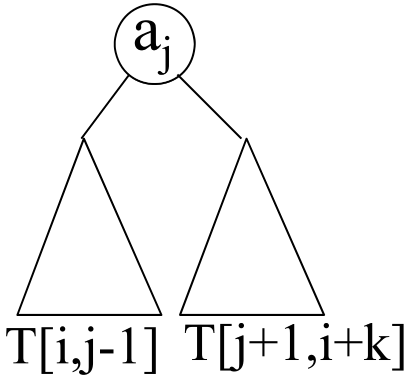
$$p_j+T'[i,j-1]+T'[j+1,i+k]$$
$$= T[i,j-1]+T[j+1,i+k]+ p_i + p_{i+1} +\dots + p_{j+k}$$

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:

$$C[i,j-1]+C[j+1,i+k]+W[i,i+k]$$

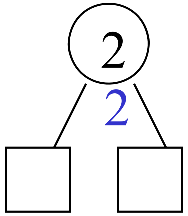
$C[i,i+k]$ is obtained by taking the minimum value while varying j.
Considering the cases where a_i and a_{i+k} are roots, we have

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

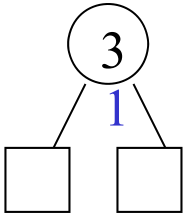
$k=1, 2, \ldots, n-i$

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

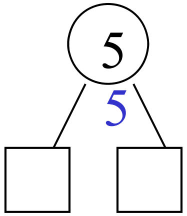
$W[i,j] = p_i + p_{i+1} + \ldots + p_j$



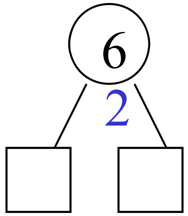
T[1,1]
C[1,1]=0.2



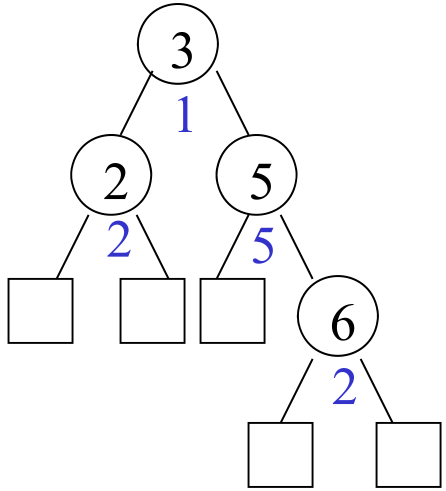
T[2,2]
C[2,2]=0.1

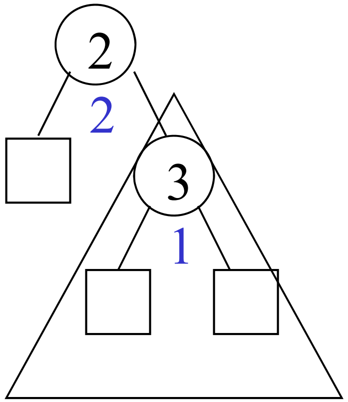


T[3,3]
C[3,3]=0.5

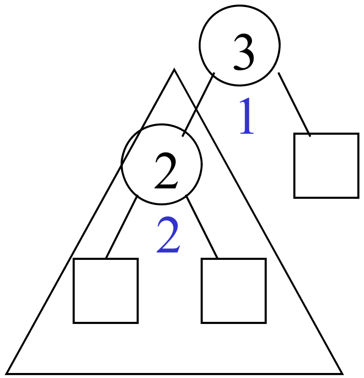


T[4,4]
C[4,4]=0.2





T[2,2]

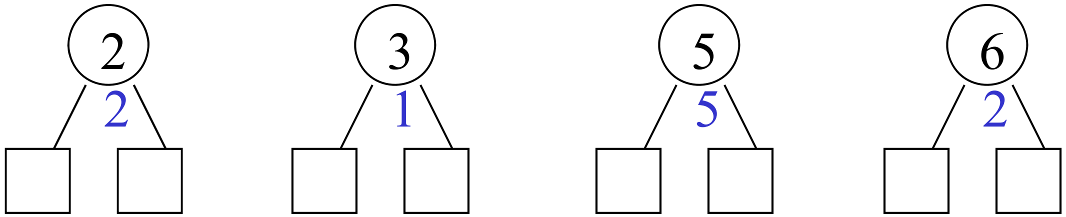


T[1,1]

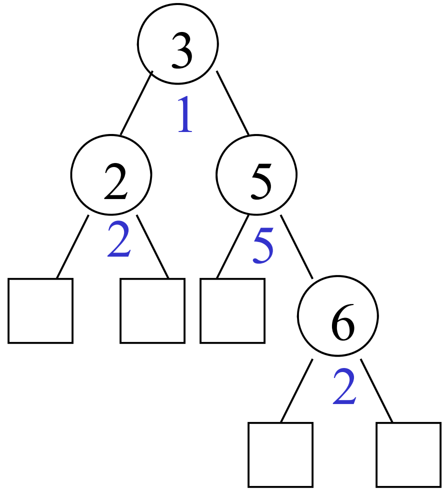
コスト
=0.2+C[2,2]+W[2,2]
=0.2+0.1+0.1=0.4

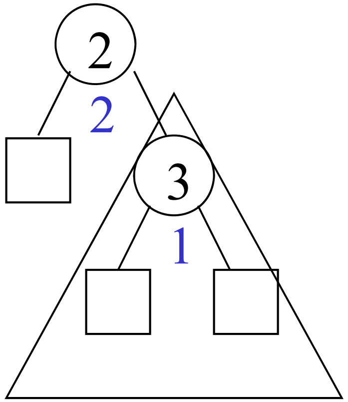
コスト
=0.1+C[1,1]+W[1,1]
=0.1+0.2+0.2=0.5

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

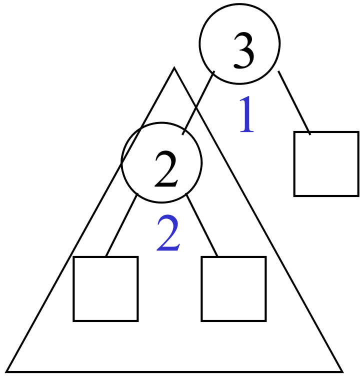


T[1,1]	T[2,2]	T[3,3]	T[4,4]
C[1,1]=0.2	C[2,2]=0.1	C[3,3]=0.5	C[4,4]=0.2





T[2,2]



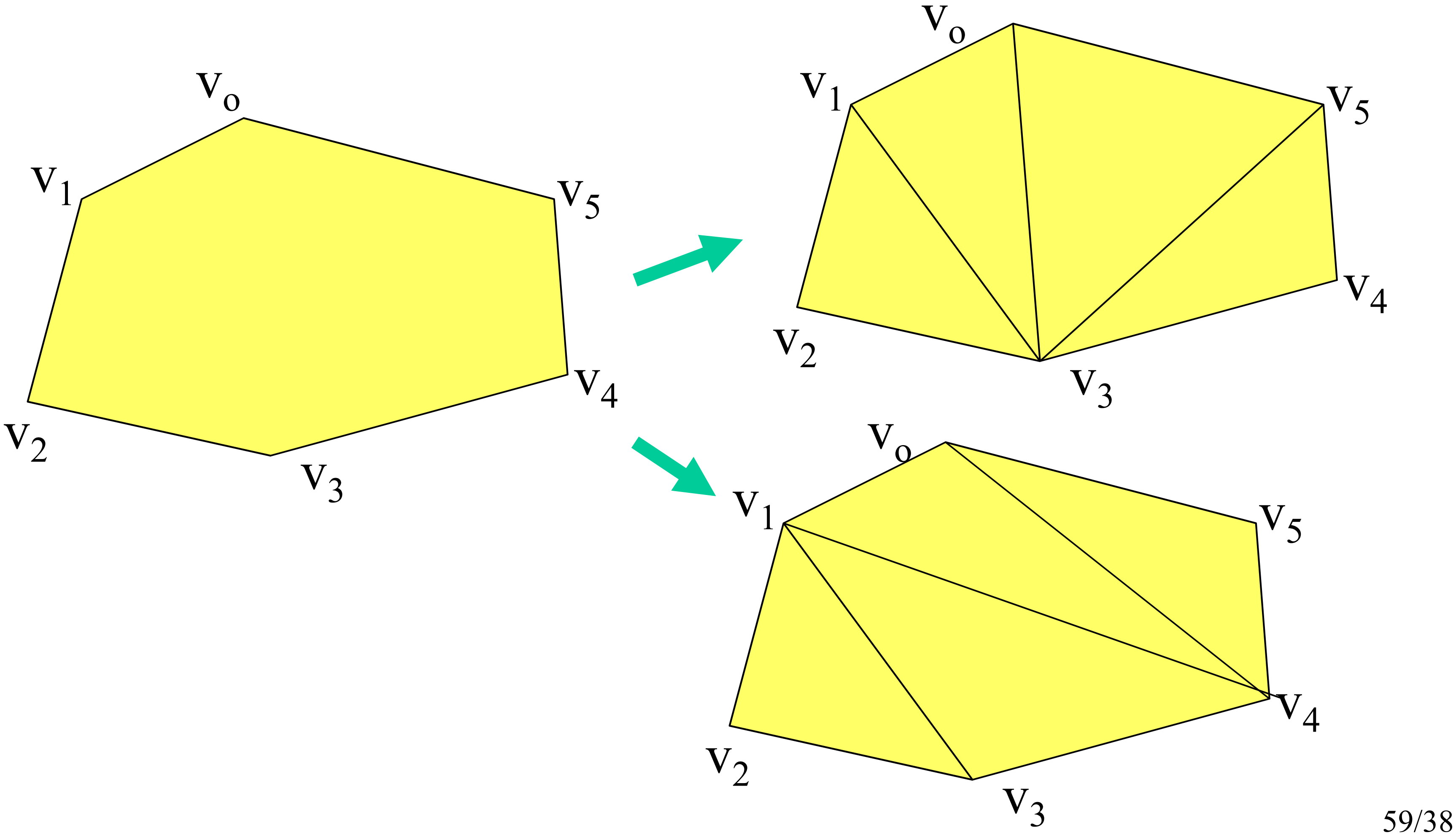
T[1,1]

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

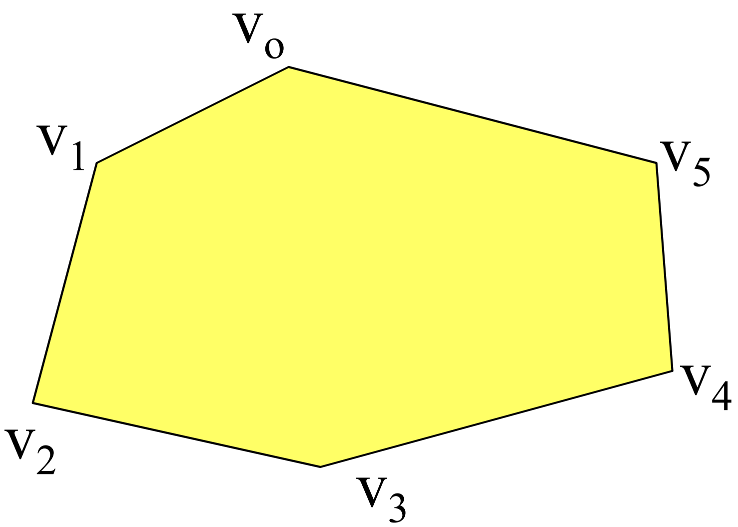
問題P24:（最適三角形分割）

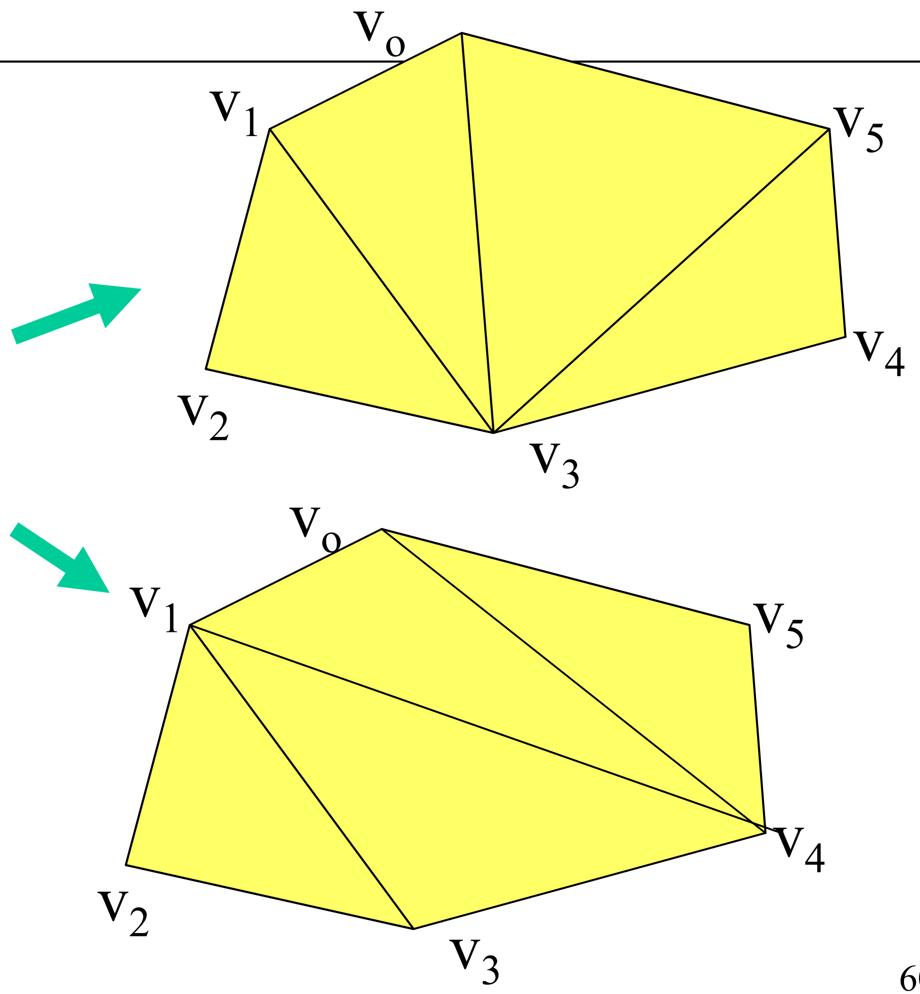
凸多角形が頂点の系列として与えられたとき，2つの頂点の間に弦を引くことにより多角形の内部を三角形に分割することができるが，弦の長さの総和を最小にする三角形分割を求めよ.



Problem P24:（Optimal triangulation）

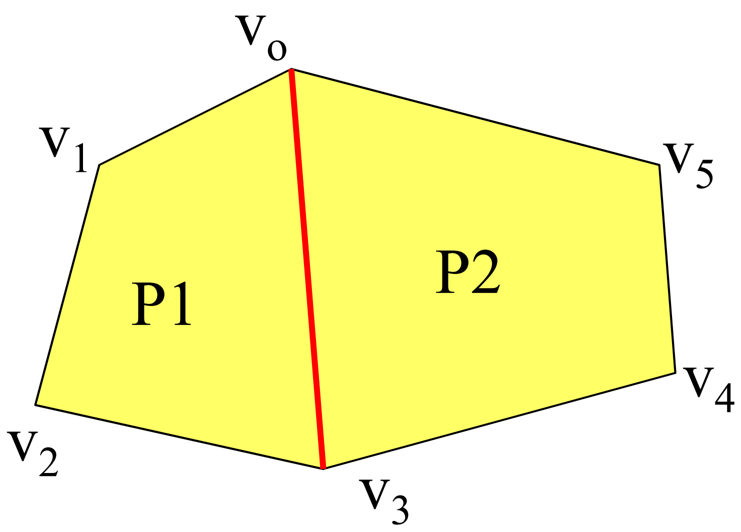
Given a convex polygon as a vertex sequence, we can partition its interior into triangles by drawing chords between two vertices. Find a triangulation so that the total sum of lengths of chords is minimized.





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．

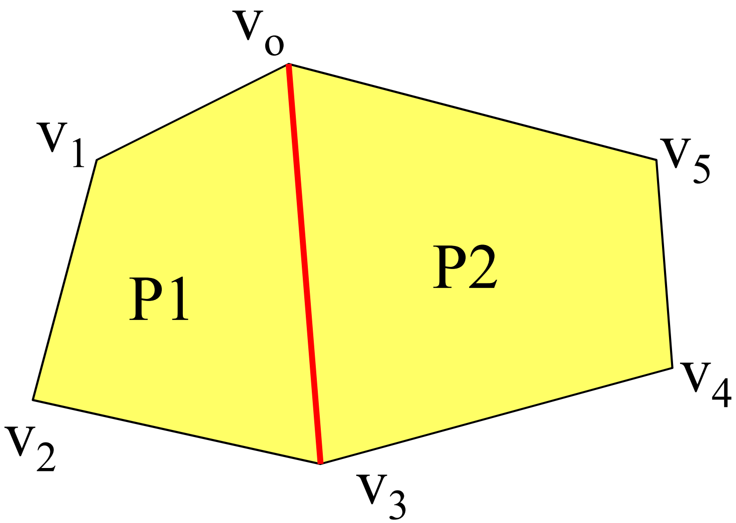
弦を１本引くことによって生じる部分多角形はやはり凸であり，頂点数はn未満だから，すべての部分多角形について最適解を求めておけば，元の問題の最適解が得られる．

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, ... , 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), ... , (v_0,v_{n-2})が考えられる.

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

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

弦$(v_0,v_4)\rightarrow$5角形$(v_0,v_1,v_{2,},v_3,v_4)$ + (n-3)角形$(v_0,v_4,v_5,\ldots,v_{n-1})$

:

弦$(v_0,v_{n-2})\rightarrow$(n-1)角形$(v_0,v_1,v_2,\ldots,v_{n-2})$ + 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})\rightarrow$3角形(v_0,v_1,v_{n-1}) + (n-1)角形$(v_1,v_2,v_3,\ldots,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,

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

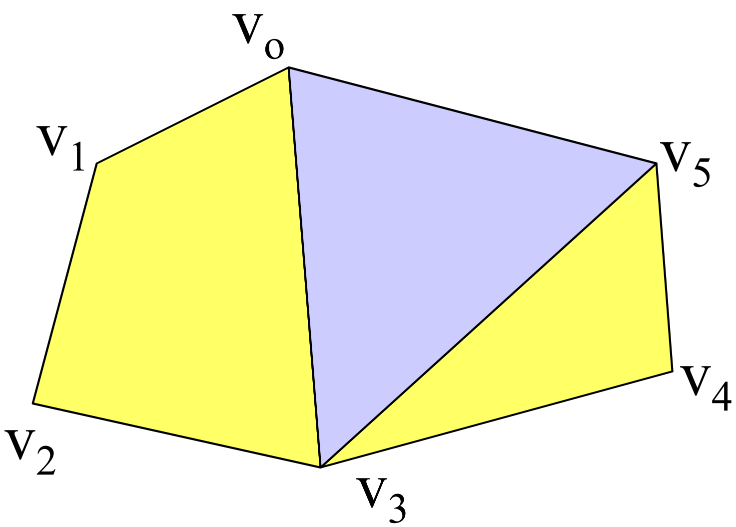
$f(3) = 1$.

$g(n)=2g(n-1)$, $g(3)=1$ 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.

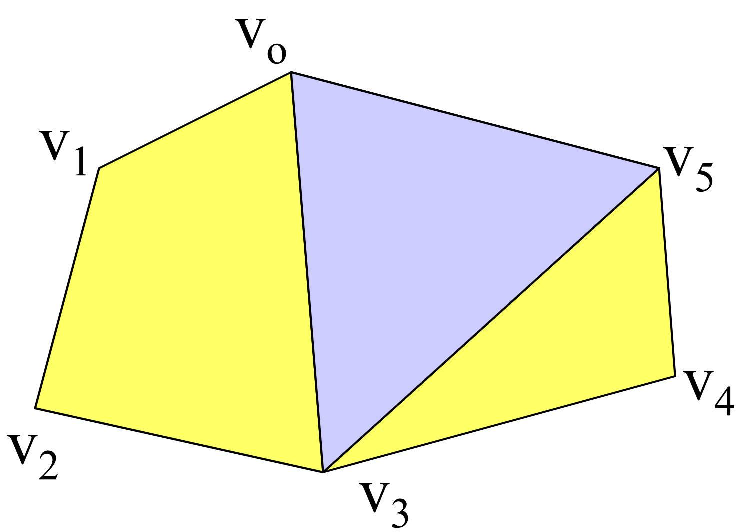


三角形(v_o, v_3, v_5)の場合, 残りの部分は2つの凸多角形に分割される.

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！

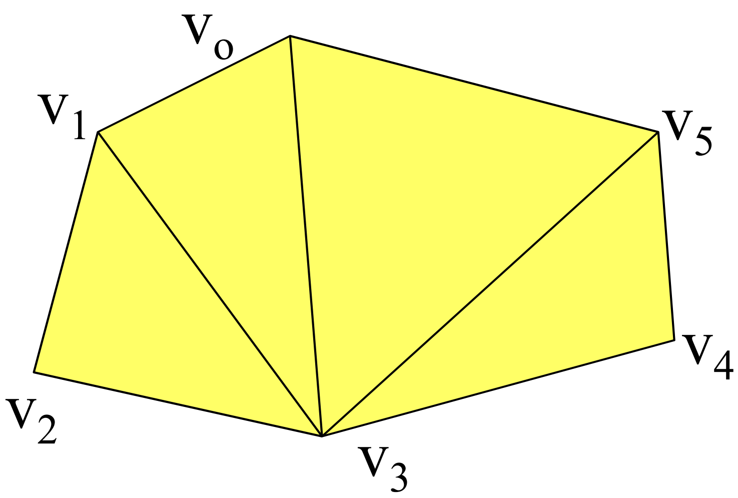
Recursive representation of a solution in a different way

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).

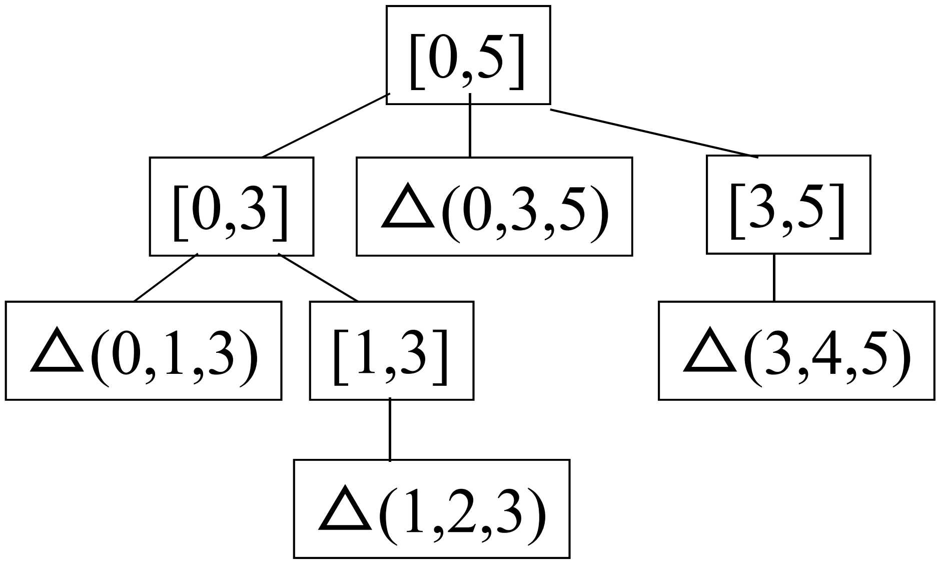


For the triangle (v_o, v_3, v_5), the remaining part is partitioned into two convex polygons.

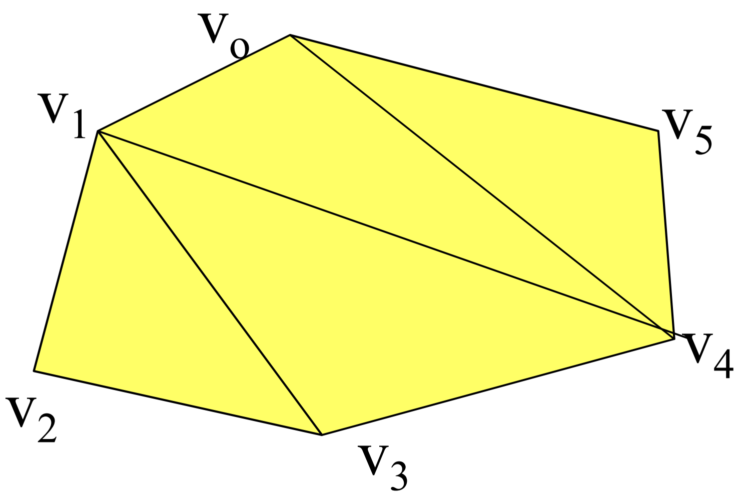
In general, when we partition the polygon P(v_o, ..., v_{n-1}) by a triangle containing the edge (v_o, v_{n-1}), we have two convex polygons: P(v_o,v_1, ..., v_k) and P(v_k,v_{k+1}, ..., v_{n-1}). This corresponds to a partition of an interval [0,n-1] into [0,k] and [k,n-1]. Thus, the number of different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



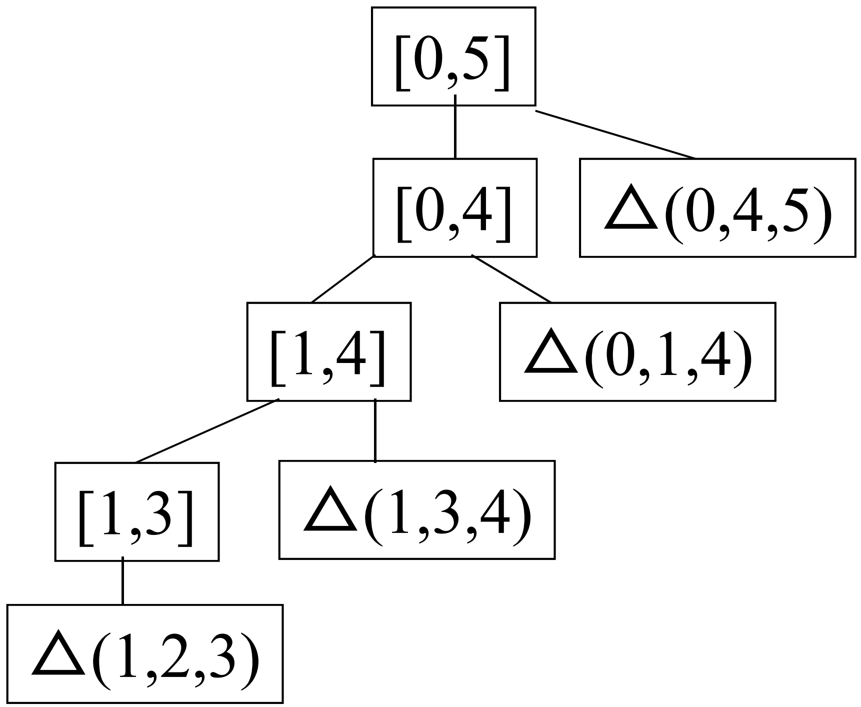
v_o
v_1
v_5
v_4
v_2
v_3



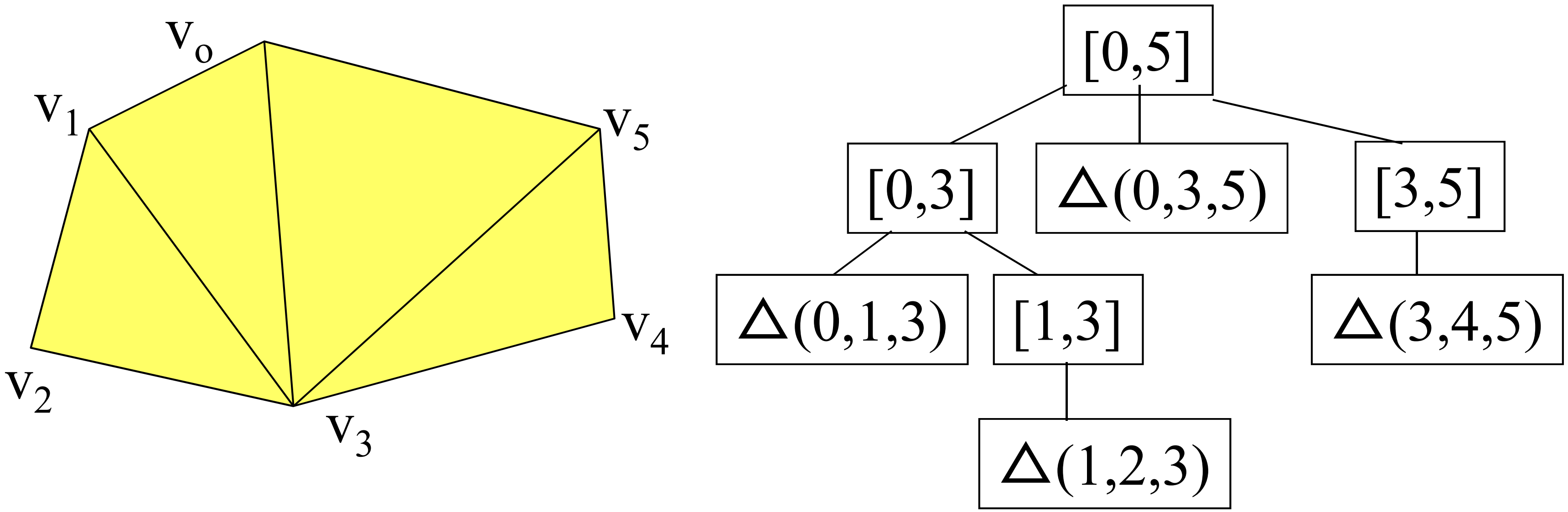
[0,5]
[0,3]
$\Delta(0,3,5)$
[3,5]
$\Delta(0,1,3)$
[1,3]
$\Delta(3,4,5)$
$\Delta(1,2,3)$



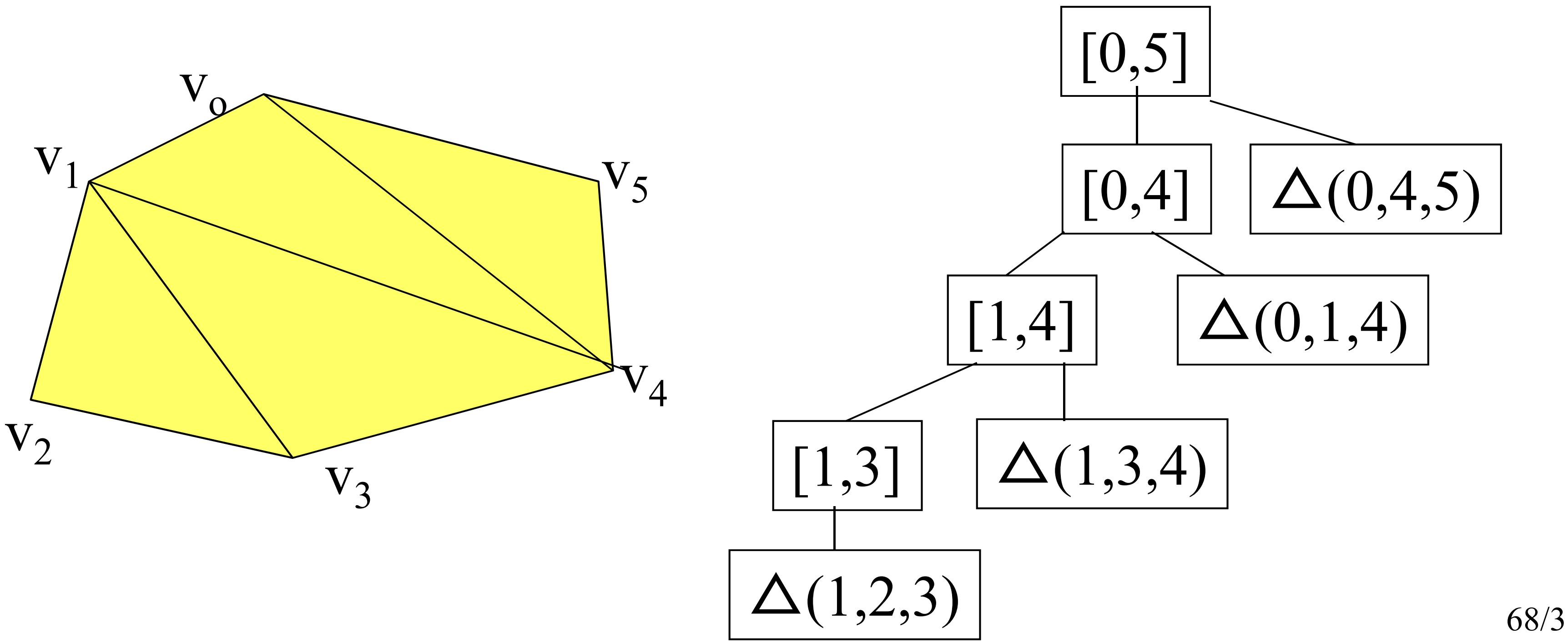
v_o
v_1
v_5
v_4
v_2
v_3



[0,5]
[0,4]
$\Delta(0,4,5)$
[1,4]
$\Delta(0,1,4)$
[1,3]
$\Delta(1,3,4)$
$\Delta(1,2,3)$

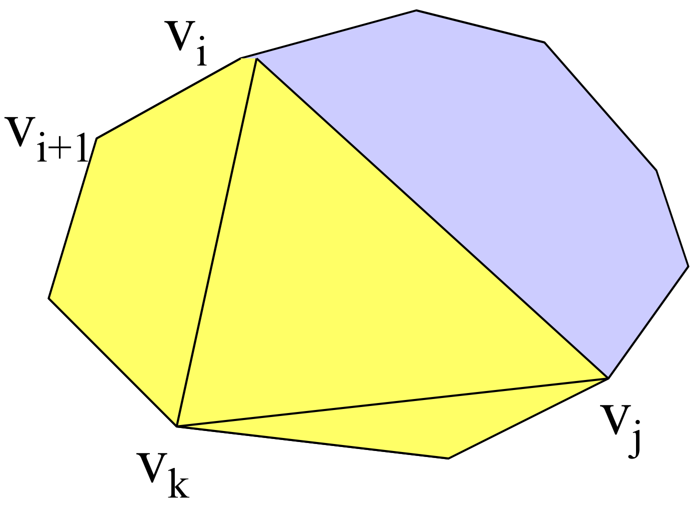


v_o
v_1
v_5
v_4
v_2
v_3
[0,5]
[0,3]
$\triangle(0,3,5)$
[3,5]
$\triangle(0,1,3)$
[1,3]
$\triangle(3,4,5)$
$\triangle(1,2,3)$



v_o
v_1
v_5
v_4
v_2
v_3
[0,5]
[0,4]
$\triangle(0,4,5)$
[1,4]
$\triangle(0,1,4)$
[1,3]
$\triangle(1,3,4)$
$\triangle(1,2,3)$

[0,n-1]の部分区間[i,j]に対応する多角形の分割

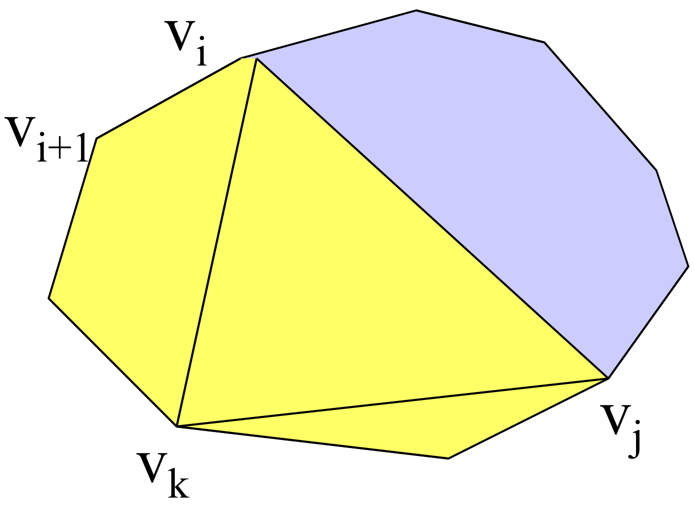


凸多角形P(v_i,v_{i+1}, ..., v_j)を
三角形(v_i, v_k, v_j),
凸多角形(v_i,v_{i+1}, ..., v_k)
凸多角形(v_k,v_{k+1}, ..., v_j)
に分割.
ただし，k=i+1のときとk=j-1のときは
三角形と残りの多角形の2つに分割.

L[i,j] = 頂点列(v_i,v_{i+1}, ..., 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

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

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



Partition a convex polygon
$P(v_i, v_{i+1}, \ldots, v_j)$ into
  a triangle (v_i, v_k, v_j),
  a convex polygon$(v_i, v_{i+1}, \ldots, v_k)$ and
  a convex polygon$(v_k, v_{k+1}, \ldots, v_j)$,
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 $(v_i, v_{i+1}, \ldots, v_j)$.
$w[i,j]$ = 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, ..., 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viと頂点vj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 $P(v_0, v_1, \ldots, v_{n-1})$

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.

おまけ:

Z. Abel, E. D. Demaine, M. Demaine, H. Ito, J. Snoeyink and R. Uehara:
Bumpy Pyramid Folding,
The 26th Canadian Conference on Computational Geometry (CCCG 2014),
pp. 258-266, 2014/08/11-2014/08/13, Halifax, Canada.

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?

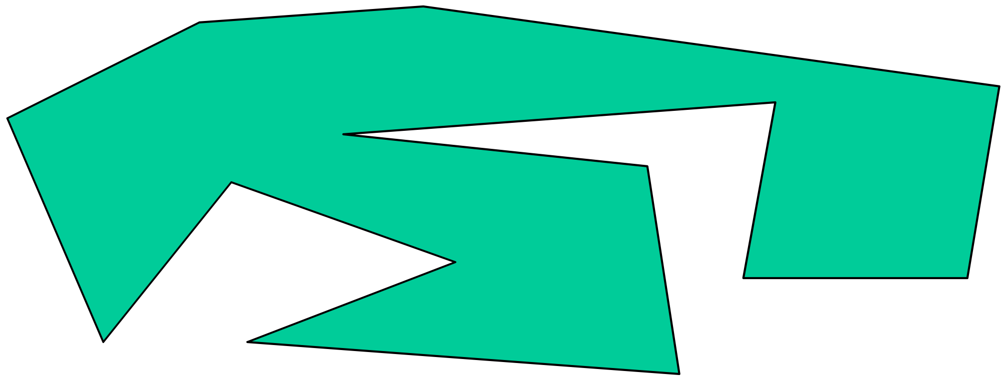
Exercise E9-5 : Modify the algorithm so that not only the value of an optimal solution but also an optimal solution itself (optimal triangulation) is obtained.

Add. :

Z. Abel, E. D. Demaine, M. Demaine, H. Ito, J. Snoeyink and R. Uehara: Bumpy Pyramid Folding, *The 26th Canadian Conference on Computational Geometry (CCCG 2014)*, pp. 258-266, 2014/08/11-2014/08/13, Halifax, Canada.

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:

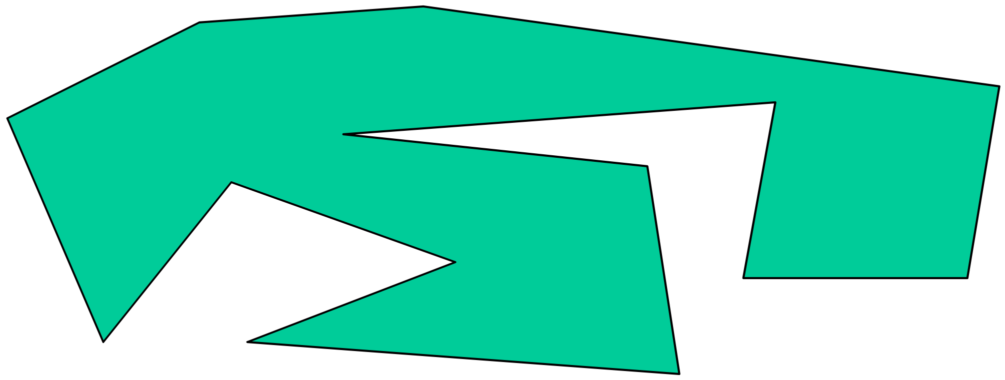
頂点v_iと頂点v_jを結ぶ線分が多角形の内部だけを通るときその長さをw[i,j]とし，そうでないときは，∞とする.

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.