アルゴリズム論
Theory of Algorithms

第8回講義
動的計画法(1)

アルゴリズム論
Theory of Algorithms

Lecture #8
Dynamic Programming (1)

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,

$C(n, k) = C(n-1, k-1) + C(n-1, k)$, 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);
}
```

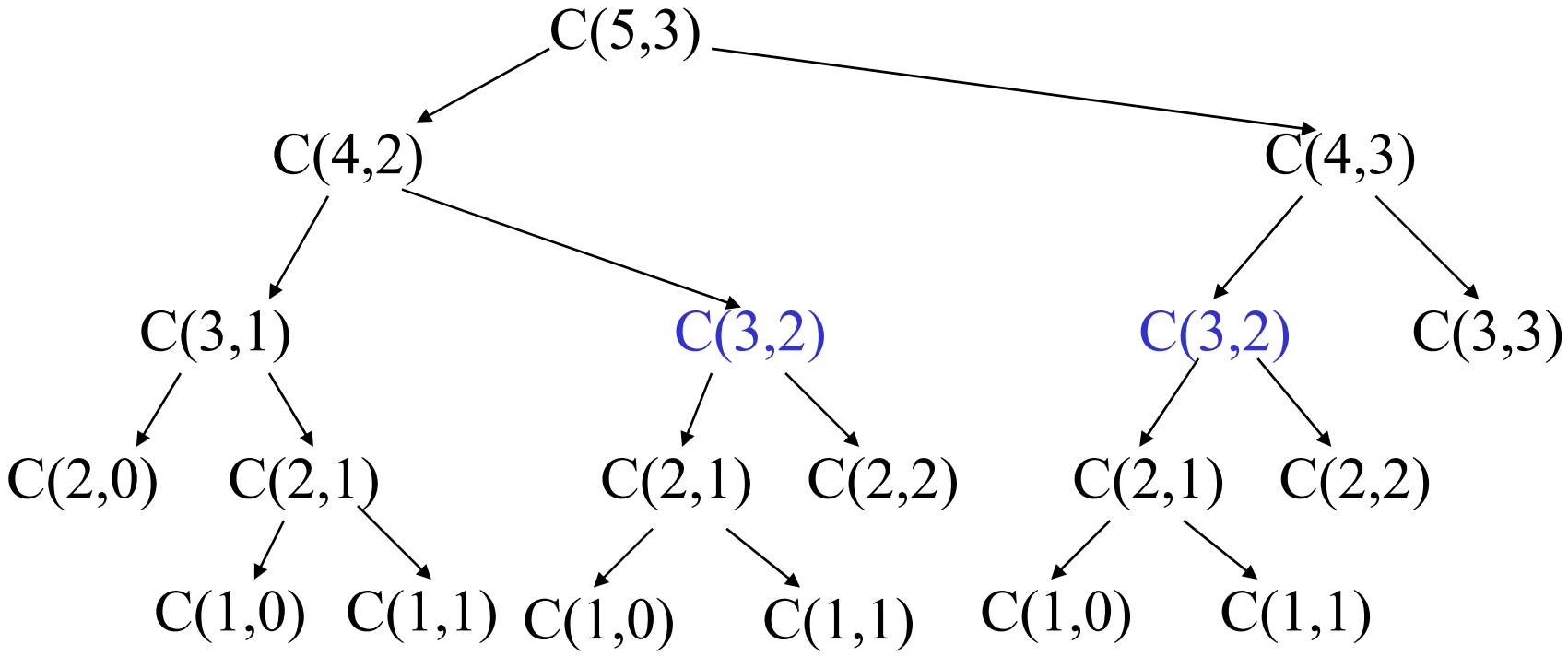
When you implement the program in practice, you will find that it takes much time. Why does it take time?

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！



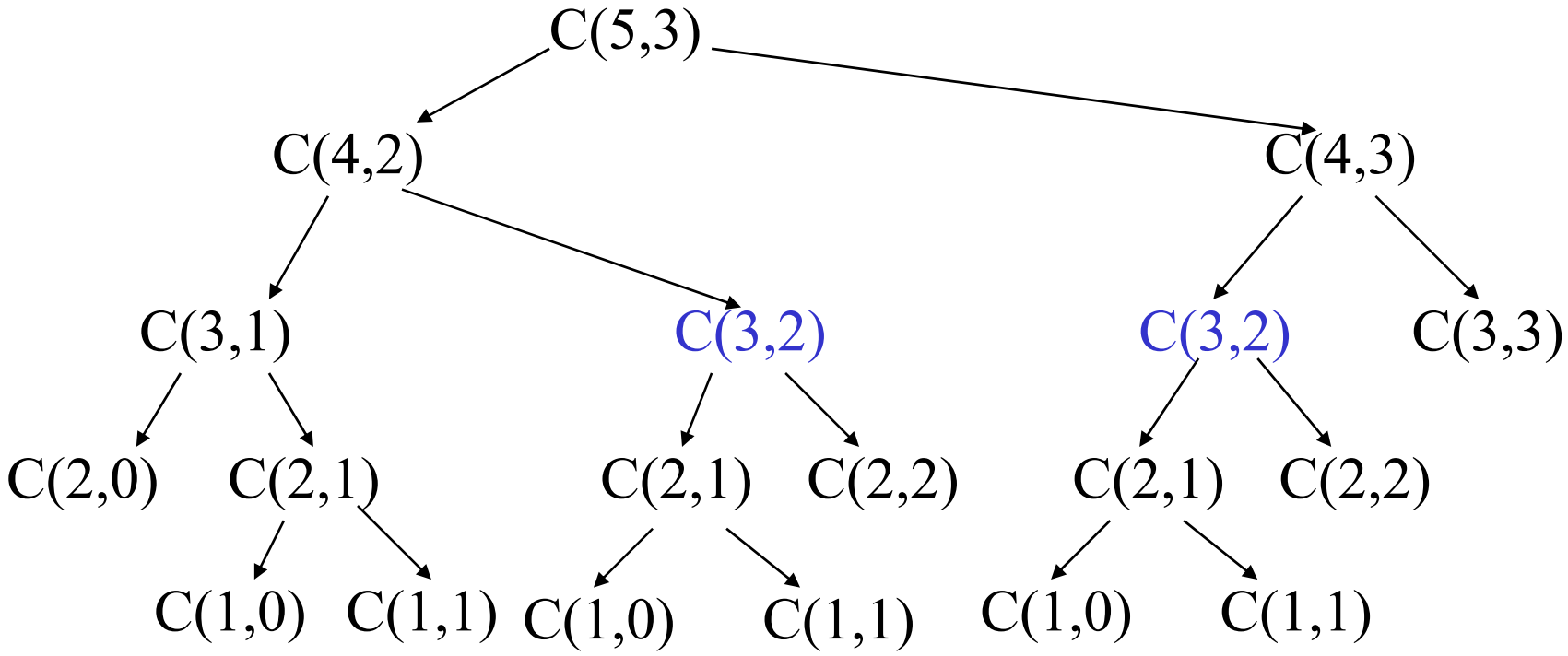
関数の呼び出しの様子

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

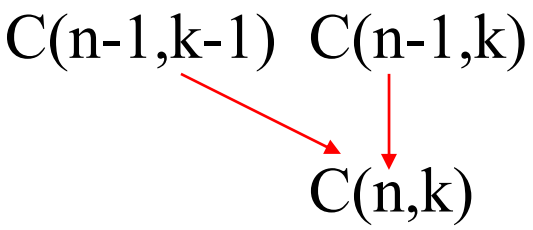
➔よって，1行目から順に埋めていけばよい．

	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



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)

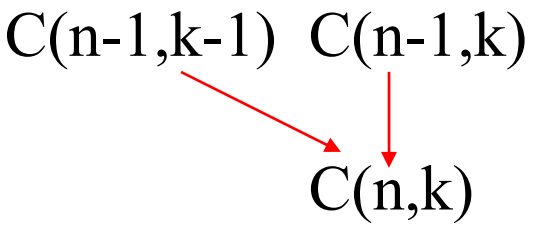
If the values in the (n-1)-st row are available, C(n,k) is easily computed. ➔Thus, we should fill in the table from the 1st row.

	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



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:

$C(n,k) = \dfrac{n!}{(n-k)!\ k!}$ 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 $C(n,k) = \frac{n!}{(n-k)!\ k!}$, 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

$F(n) = F(n-1) + F(n-2)$, 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) $a_i = b_j$ → L[i,j] = L[i-1,j-1]+1
(2) $a_i \neq b_j$ → 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*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) $a_i = b_j$ → L[i,j] = L[i-1,j-1]+1
(i-1, j-1)を記憶
(2) $a_i \neq b_j$ → 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]

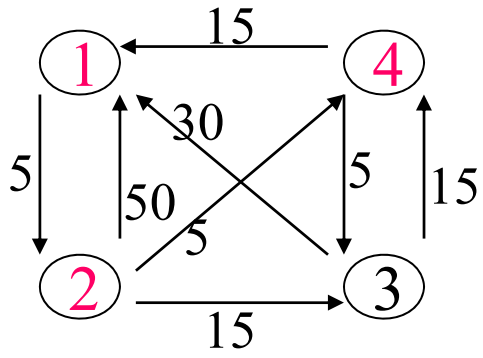
a[8]b[12] a[6]b[11] a[4]b[10] a[3]b[8] a[2]b[5] a[1]b[2]

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.



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

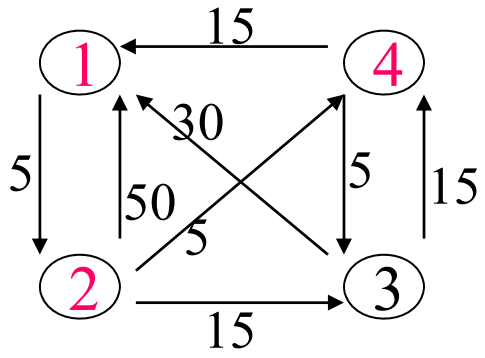
距離行列

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 $O(n^3)$.

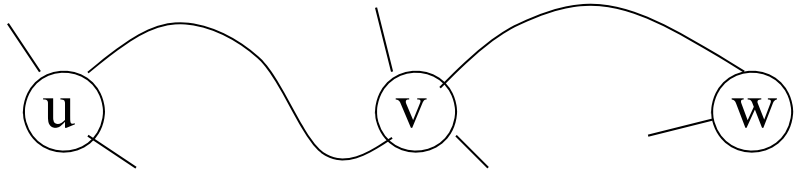
Since the number m of edges is $O(n^2)$, it takes $O(n^3)$ in the worst case.

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．

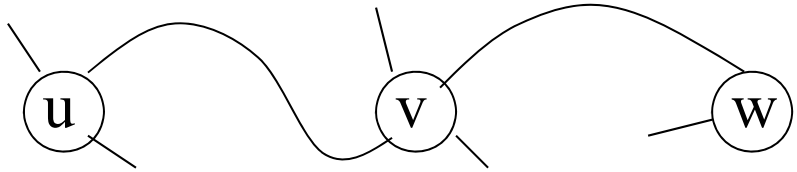


理由：uからvまでの部分経路が最短でなければ，
この部分経路を最短経路で置きかえると，uからw
へのより短い経路が得られる．これは矛盾．

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] }

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

Case 2: if the shortest path passes through vertex k

➔shortest path from i to k＋that from k to j

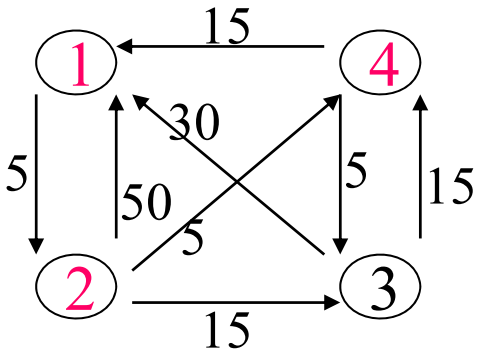
$\mathbf{D_k[i,j] = \min\{ D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] \}}$

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.

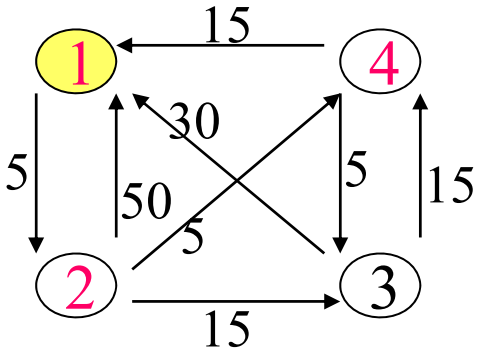
Algorithm P23-A0:

```
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] }
```



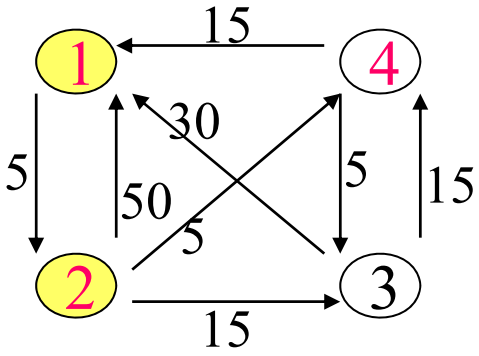
$$D_0=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

距離行列



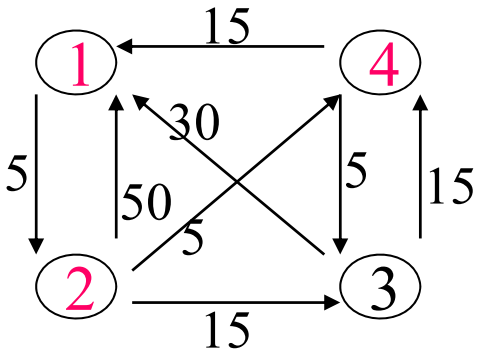
$$D_1=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

頂点1を途中に
通ってもよい.



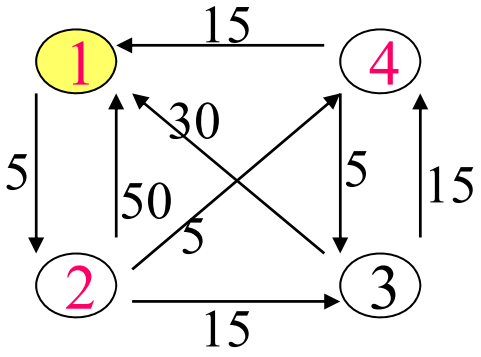
$$D_2=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

頂点1，2を途中に
通ってもよい.



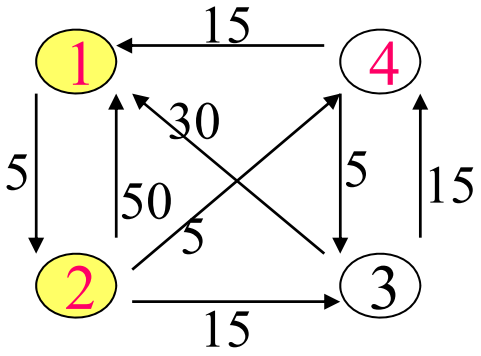
$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

distance matrix



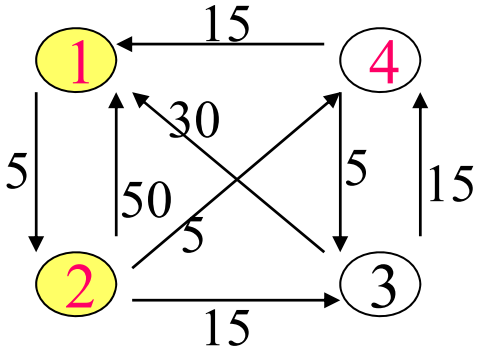
$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertex 1 can be passed through



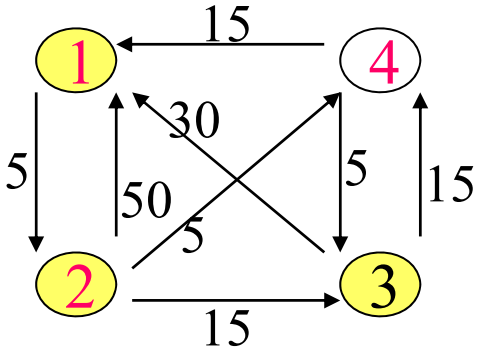
$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertices 1 and 2 can be passed through



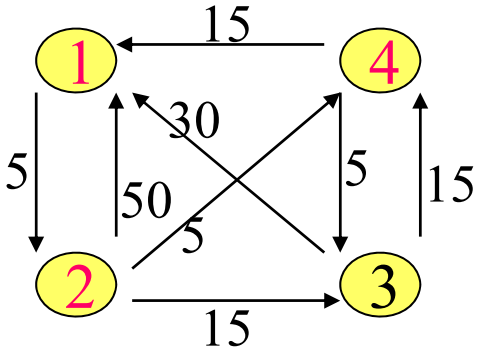
$$D_2=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

頂点1，2を途中に通ってもよい.



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

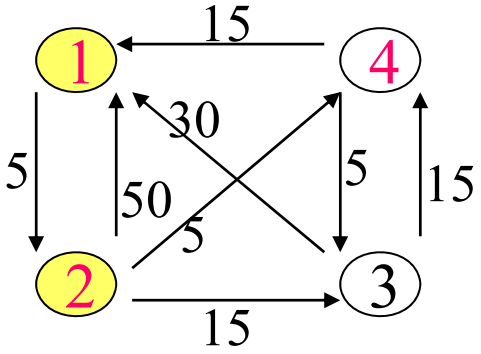
頂点1，2，3を途中に通ってもよい.



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

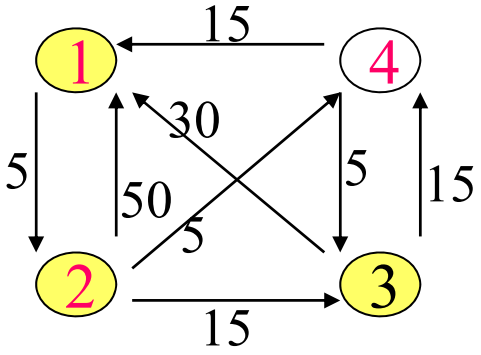
頂点1，2，3，4を途中に通ってもよい

最終的な距離行列



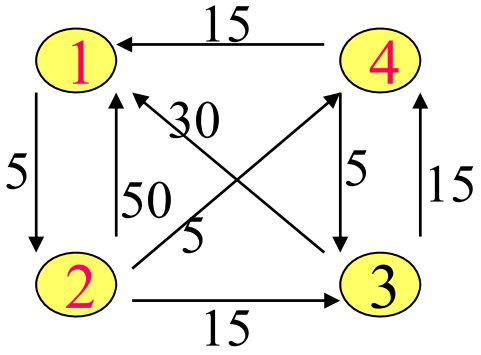
$$D_2=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertices 1 and 2 can be passed through



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

vertices 1, 2, and 3 can be passed through



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

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, ... , 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}.

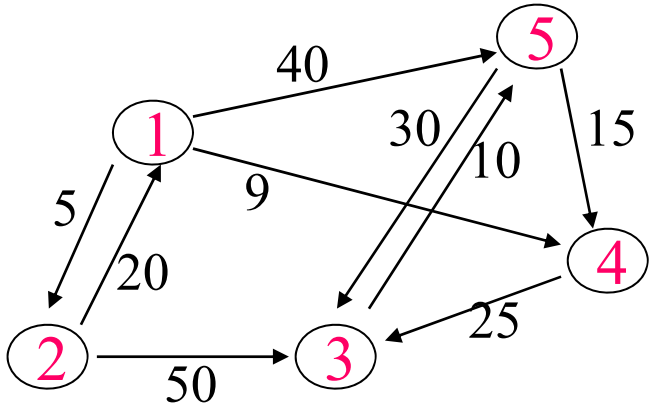
Hence, only one array is sufficient.

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.