アルゴリズム論
Theory of Algorithms

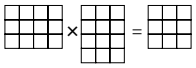
第10回講義
動的計画法(3)

アルゴリズム論
Theory of Algorithms

Lecture #10
Dynamic Programming (3)

問題P25:（連鎖行列積）
n個の行列の系列<A_1,A_2, ... , A_n>が与えられたとき，行列積
$A_1 \times A_2 \times \ \ldots \ \times A_n$
を計算するのに，演算回数を最小にする行列積の順序を求めよ．

p行×q列の行列とq行×r列の行列の行列積を計算するとp行×r列の行列が得られる．このときの演算（乗算と加算）の回数はp×q×r．

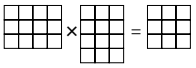


3行×4列　4行×3列　3行×3列

例: A_1=10行×20列，A_2=20行×5列，A_3=5行×25列のとき，
$((A_1 \times A_2) \times A_3)$の順だと，(10×20×5)+(10×5×25)=2250
$(A_1 \times (A_2 \times A_3))$の順だと，(10×20×25)+(20×5×25)=7500
なので，前者の方が演算回数は少ない．

Problem P25:（Chained Matrix Product）
Given a sequence of n matrices <A_1,A_2, ... , A_n>, find an order of matrix products to minimize the number of operations to compute the matrix product $A_1 \times A_2 \times \ \ldots \ \times A_n$.

Product of a p×q matrix and q×r matrix is a p×r matrix using p×q×r operations (multiplication and additioN).



3×4　4×3　3×3

Example: A_1=10×20 matrix, A_2=20×5 matrix, A_3=5×25 matrix.
$((A_1 \times A_2) \times A_3)$ require (10×20×5)+(10×5×25)=2250 ops.
$(A_1 \times (A_2 \times A_3))$ requires (10×20×25)+(20×5×25)=7500 ops.
Thus, the former needs less operations.

4個の行列の積だと，何通りも計算順序がある．
$((A_1 \times (A_2 \times A_3)) \times A_4)$
$(((A_1 \times A_2) \times A_3) \times A_4)$
$((A_1 \times A_2) \times (A_3 \times A_4))$
$(A_1 \times ((A_2 \times A_3) \times A_4))$
$(A_1 \times (A_2 \times (A_3 \times A_4)))$

正確には $\frac{1}{n+1}\binom{2n}{n}$

これらすべての計算順序について演算回数を求めればよい．

演習問題E10-1：括弧のつけ方は$O(4^n/n^{3/2})$通りあることを証明せよ．これはCatalan数として知られているものである．
ヒント：括弧のつけ方がP(n)通りあるとする．任意の系列においてk番目とk+1番目の間で分割してそれぞれの部分列に対して独立に括弧をつけることができる．よって，次の漸化式を得る．
P(1) = 1
$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$

For product of four matrices, there are many orders for their product.
$((A_1 \times (A_2 \times A_3)) \times A_4)$
$(((A_1 \times A_2) \times A_3) \times A_4)$
$((A_1 \times A_2) \times (A_3 \times A_4))$
$(A_1 \times ((A_2 \times A_3) \times A_4))$
$(A_1 \times (A_2 \times (A_3 \times A_4)))$

Precisely, $\frac{1}{n+1}\binom{2n}{n}$

It suffices to obtain the number of operations for all of them.

Exercise E10-1: Prove that there are $O(4^n/n^{3/2})$ ways for parenthesizations. This is known as the Catalan number.
Hint: Suppose there are P(n) ways for parenthesization. In each sequence we can parenthesize it by dividing it between its k-th and (k+1)-st position into subsequences independently. Thus, we have
P(1) = 1
$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$

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．

4個の行列の積の場合
$((A_1 \times (A_2 \times A_3)) \times A_4)$　最後は(A_1,A_2,A_3)とA_4の積
$(((A_1 \times A_2) \times A_3) \times A_4)$　最後は(A_1,A_2,A_3)とA_4の積
$((A_1 \times A_2) \times (A_3 \times A_4))$　最後は(A_1,A_2)と(A_3,A_4)の積
$(A_1 \times ((A_2 \times A_3) \times A_4))$　最後はA_1と(A_2,A_3,A_4)の積
$(A_1 \times (A_2 \times (A_3 \times A_4)))$　最後はA_1と(A_2,A_3,A_4)の積
部分系列に対する最適な計算順序が分かっていれば，
$((A_1,A_2,A_3), A_4)$, $((A_1,A_2), (A_3,A_4))$, $(A_1,(A_2,A_3,A_4))$
の3通りの分割を調べればよい．

一般には，最初にどこで分けるかが問題．
$((A_1, ...,A_k), (A_{k+1}, ... , A_n))$　k=1, 2, ... , n-1
それぞれの部分系列に対する最適な計算順序が分かっていれば
全体の最適な計算順序もわかる．

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

To compute the product of 4 matrices
$((A_1 \times (A_2 \times A_3)) \times A_4)$　last is the product of (A_1,A_2,A_3) and A_4
$(((A_1 \times A_2) \times A_3) \times A_4)$　last is the product of (A_1,A_2,A_3) and A_4
$((A_1 \times A_2) \times (A_3 \times A_4))$　last is the product of (A_1,A_2) and (A_3,A_4)
$(A_1 \times ((A_2 \times A_3) \times A_4))$　last is the product of A_1 and (A_2,A_3,A_4)
$(A_1 \times (A_2 \times (A_3 \times A_4)))$　last is the product of A_1 and (A_2,A_3,A_4)
If we know an optimal orders for subsequences, it suffices to check the three ways of partitions.
$((A_1,A_2,A_3), A_4)$, $((A_1,A_2), (A_3,A_4))$, $(A_1,(A_2,A_3,A_4))$

Generally, the problem is the place for the first partition.
$((A_1, ...,A_k), (A_{k+1}, ... , A_n))$　k=1, 2, ... , n-1
If we know an optimal order for computation for each subsequence, then an optimal order for computation is obtained.

各行列のサイズをp_i行q_i列とすると，行列積が定義されるためには，
$q_1=p_2, q_2=p_3, \cdots , q_n=p_{n+1}$
でなければならない．
したがって，入力では，$p_1, p_2, \cdots ,p_n, p_{n+1}$ だけを指定する．
また，A_iからA_jまでの行列の積をとると，p_i行$q_j=p_{j+1}$列の行列が得られる．
A_iからA_jまでの行列の積の計算に必要な最小の演算回数を M[i,j]とする．この積を計算するのに，kをiからjまで変化させて A_iからA_kまでの積とA_{k+1}からA_jまでの積をすべて評価すればよい．
A_iからA_kまでの積はp_i行p_{k+1}列の行列であり，A_{k+1}からA_jまでの積はp_{k+1}行p_{j+1}列の行列だから，それらの行列積に
$p_i p_{k+1} p_{j+1}$
回の演算が必要である．したがって，M[i,j]を求める漸化式は
$M[i, j] = \min\{M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1}, k=i,i+1, \cdots , j-1\}$
となる．

Let the size of each matrix be $p_i \times q_i$. Then, only if we have
$q_1=p_2, q_2=p_3, \cdots , q_n=p_{n+1}$
the product of those matrices is defined. Thus, we only specify $p_1, p_2, \cdots , p_n, p_{n+1}$ for input.
If we take the product of matrices from A_i to A_j then the $p_i \times q_j=p_{j+1}$matrix is obtained.
M[i,j] = the smallest number of computations to calculate the product of matrices from A_i to A_j. For the computation it suffices to evaluate all possible productions of matrices from A_i to A_k and those from A_{k+1} to A_j for each k between i and j.
The product for A_i through A_k is a $p_i \times p_{k+1}$ matrix, and that for A_{k+1} through A_j is a $p_{k+1} \times p_{j+1}$ matrix.
Thus, the number of operations we need to compute them is
$p_i p_{k+1} p_{j+1}$.
Therefore, the recurrence equation for M[i,j] is
$M[i, j] = \min\{M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1}, k=i,i+1, \cdots , j-1\}$.

アルゴリズムP25-A0:
入力：n個の行列のサイズ(p_1行p_2列),(p_2行p_3列), ... ,(p_n行p_{n+1}列).

```
for(i=1; i<=n; i++)
    M[i,i] = 0;
for(d=1; d<=n; d++)
  for(i=1; i<=n-d; i++)
    j=i+d;
    msf = M[i,i]+M[i+1,j]+p_i p_{i+1} p_{j+1};
    for(k=i; k<j; k++)
      if( M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1} < msf)
        msf =  M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1};
    M[i,j] = msf;
  }
return M[1,n];
```

演習問題E10-2：上のアルゴリズムでは最適解の値しか分からない．最適な計算順序も求められるようにアルゴリズムを変更せよ．

algorithm P25-A0:
input：matrix sizes ($p_1$rows p_2 columns),(p_2, p_3), ... ,(p_n, p_{n+1}).

```
for(i=1; i<=n; i++)
    M[i,i] = 0;
for(d=1; d<=n; d++)
  for(i=1; i<=n-d; i++)
    j=i+d;
    msf = M[i,i]+M[i+1,j]+p_i p_{i+1} p_{j+1};
    for(k=i; k<j; k++)
      if( M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1} < msf)
        msf =  M[i,k]+M[k+1,j]+p_i p_{k+1} p_{j+1};
    M[i,j] = msf;
  }
return M[1,n];
```

Exercise E10-2：The above algorithm only finds the value of an optimal solution. Modify it so that an optimal order of computation is also obtained.

問題P26:（ナップサック問題）
n個の荷物o_i(i=1, ... , n)に対する重さw_iと価値v_i，ナップサックの制限重量Cが与えられたとき，荷物の合計の重さがCを超えないような荷物の詰め込み方の中で価値が最大となるものを求めよ．

入力をI = {w_1, ... , w_n; v_1, ... , v_n; C}とする．解は{1,2,...,n}の部分集合Sで表現できる．最適解は，
容量制約　$\Sigma_{i\in S} w_i \leqq C$
を満たすSの中で
価値の総和　$\Sigma_{i\in S} v_i$
を最大にするものである．

仮定：どの荷物についても，その重さはCを超えないものとする．Cを超える荷物は決して選ばれることがないからである．

Problem P26:（Knapsack Problem）
Given n objects o_i (i=1, ... , n) and their weights w_i , prices v_i, and the capacity (or weight limit) C of a knapsack, find an optimal way of packing objects into the knapsack to meet the capacity constraint in such a way that the total price is maximized.

Input: I = {w_1, ... , w_n; v_1, ... , v_n; C}. A solution is represented by a subset S of {1,2,...,n}.
An optimal solution is such a set S satisfying the
Capacity constraint　$\Sigma_{i\in S} w_i \leqq C$
and maximizing
total sum of prices　$\Sigma_{i\in S} v_i$.

Assumption: Assume that weight of any object does not exceed the capacity C because any object with weight exceeding C is never selected.

例題:(w_1, ..., w_5)=(2,3,4,5,6), (v_1, ..., v_5)=(4,5,8,9,11), C=10の場合を考えよう．
V[k] = k番目までの荷物だけを対象にしたときの最適解の値とすると，定義より明らかに
V[1]≦V[2] ≦･･･≦V[n]
が成り立つ．この例では，次のようになる．
V[1] = v_1=4, w_1=2≦C,
V[2] = v_1+v_2=4+5=9, w_1+w_2=2+3≦C,
V[3] = v_1+v_2+v_3=4+5+8=17, w_1+w_2+w_3=2+3+4≦C,
V[4] = v_1+v_2+v_4=4+5+9=18, w_1+w_2+w_4=2+3+5≦C,
V[5] =v_3+v_5=8+11=19, w_3+w_5=4+6≦C.
ここで，{1,2,3,4}は解ではない．なぜなら，重さの合計が容量10を超過してしまうからである．

この例では，**部分問題に対する解が最適解に含まれない**．したがって，上のような順序で解を求めるのに動的計画法は適用できない．

Example: Consider the case in which (w_1, ..., w_5)=(2,3,4,5,6), (v_1, ..., v_5)=(4,5,8,9,11), C=10.
V[k] = value of an optimal solution for objects up to the k-th one.
Then, by the definition
V[1]≦V[2] ≦･･･≦V[n].
In this example, we have
V[1] = v_1=4, w_1=2≦C,
V[2] = v_1+v_2=4+5=9, w_1+w_2=2+3≦C,
V[3] = v_1+v_2+v_3=4+5+8=17, w_1+w_2+w_3=2+3+4≦C,
V[4] = v_1+v_2+v_4=4+5+9=18, w_1+w_2+w_4=2+3+5≦C,
V[5] =v_3+v_5=8+11=19, w_3+w_5=4+6≦C.
Here, {1,2,3,4} is not a solution since the total weight exceeds the capacity 10.

In this example, an optimal solution to a subproblem may not be included in an optimal solution. Thus, we cannot apply Dynamic Programming to find a solution in the above order.

では，荷物の選び方をすべて列挙して調べるという方法はどうだろう？

それぞれの荷物について，選ぶか選ばないか2通りある．
=>荷物の選び方は全部で2^n通り
すべての選び方を調べると指数時間かかってしまう．

動的計画法を適用するためには，**部分問題に対する解が最適解に含まれるように最適解を再帰的に定義しなければならない**．

D[i,j] = 荷物1,...,iの中から適当に荷物を選んで重さの和がjとなる荷物の組合せの中での価値の最大値，
ただし，重さの和がちょうどjとなる組合せがなければ0とする．

荷物1,...,i-1に関する最適解が分かっているとき，それぞれの解に荷物iを加える場合と加えない場合の良い方をとればよいから，
D[i,j] = max{D[i-1, j], D[i-1,j-w_i]+v_i}
これは**部分構造の最適性**が成り立つことを示している．

Then, what about a method to examine all possible ways of choosing objects?

For each object there are two ways, to choose or not to choose.
⇒there are 2^n ways to choose objects.
It takes exponential time if we examine all possible cases.

To apply Dynamic Programming, an optimal solution must be defined recursively so that it includes a solution to a subproblem.

D[i,j] = the largest total price among all possible ways to choose objects from objects 1, ... , i so that the total weight is j.
It is 0 if there is no way to choose them so that the total weight is j.

If an optimal solutions for objects 1,...,i-1 is known, we just consider two cases, to add an object i and not to add it. Thus, we have
D[i,j] = max{D[i-1, j], D[i-1,j-w_i]+v_i}
This implies the property of **Optimal Substructure**.

例題: $(w_1, ..., w_5)=(2,3,4,5,6)$, $(v_1, ..., v_5)=(4,5,8,9,11)$, C=10の場合
i=1のとき，荷物1を選ぶか選ばないかの2通りだけだから，
$D[1,w_1]=D[1,2]=v_1=4$, $D[1,j]=0$, $j \neq 2$,
i=2のとき，{}, {1}, {2}, {1,2}の組合せがあるから，
$D[2,2]=4$, $D[2,3]=5$, $D[2,5]=9$, $D[2,j]=0$ $j \neq 2,3,5$

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4								
2	4	5		9					
3	4	5	8	9	12	13		17	
4	4	5	8	9	12	13	14	17	18
5	4	5	8	9	12	13	15	17	19

は新たに得られた解を示す

$w_1=2, v_1=4$
$w_2=3, v_2=5$
$w_3=4, v_3=8$
$w_4=5, v_4=9$
$w_5=6, v_5=11$

重量制約C=10を超える重さになる組合せは無視してよい.

Example: Let $(w_1, ..., w_5)=(2,3,4,5,6)$, $(v_1, ..., v_5)=(4,5,8,9,11)$, C=10.
i=1➔only two ways to choose object 1 or not choose it:
$D[1,w_1]=D[1,2]=v_1=4$, $D[1,j]=0$, $j \neq 2$,
i=2➔there are four cases: {}, {1}, {2}, {1,2}
$D[2,2]=4$, $D[2,3]=5$, $D[2,5]=9$, $D[2,j]=0$ $j \neq 2,3,5$

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4								
2	4	5		9					
3	4	5	8	9	12	13		17	
4	4	5	8	9	12	13	14	17	18
5	4	5	8	9	12	13	15	17	19

indicates a new solution

$w_1=2, v_1=4$
$w_2=3, v_2=5$
$w_3=4, v_3=8$
$w_4=5, v_4=9$
$w_5=6, v_5=11$

We can ignore a set of objects if their total weight exceeds 10.

アルゴリズムP26-A0:
入力：n個の荷物o_i(i=1, ... , n)の重さw_iと価値v_i，制限重量C

```
for(i=1; i<=C; i++)
    D[0,i] = 0;
for(k=1; k<=n; k++)
  for(i=1; i<=C; i++)
    if(i＜w_i) D[k,i] = D[k-1,i];
    else {
      if(D[k-1,i-w_i]+v_i > D[k-1,i])
        D[k,i] = D[k-1,i-w_i]+v_i;
      else
        D[k,i] = D[k-1, i];
    }
max=0;
for(i=1; i<=C; i++)
  if(D[n,i]>max) max = D[n,i];
return max;
```


Algorithm P26-A0:
Input：n objects o_i(i=1, ... , n): weight w_i and price v_i, capacity C.

```
for(i=1; i<=C; i++)
    D[0,i] = 0;
for(k=1; k<=n; k++)
  for(i=1; i<=C; i++)
    if(i＜w_i) D[k,i] = D[k-1,i];
    else {
      if(D[k-1,i-w_i]+v_i > D[k-1,i])
        D[k,i] = D[k-1,i-w_i]+v_i;
      else
        D[k,i] = D[k-1, i];
    }
max=0;
for(i=1; i<=C; i++)
  if(D[n,i]>max) max = D[n,i];
return max;
```


最適解の値だけでなく，最適解も構成したい.

D[i,j]の表だけでなく，D[i,j]の値を与える荷物の組合せも記憶するようにする.
$D[i,j] = \max\{D[i, j-1], D[i-w_j,j-1]+v_i\}$
$T[i,j] = j$ $D[i,j]=D[i-w_j,j-1]+v_j$のとき
$T[i,j] = 0$ $D[i,j]=D[i,j-1]$のとき
このように決めると最適解を与えるD[i, n]から逆に辿ることにより最適解を求めることができる.

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4/1								
2	4/0	5/2		9/2					
3	4/0	5/0	8/3	9/0	12/3	13/3		17/3	
4	4/0	5/0	8/0	9/0	12/0	13/0	14/4	17/0	18/4
5	4/0	5/0	8/0	9/0	12/0	13/0	15/5	17/0	19/5

D[i,j]/T[i,j]の値

Want to construct an optimal solution with the value of optimal solution.

We maintain not only the table D[i,j] but also the combination to give the value of D[i,j].
$D[i,j] = \max\{D[i, j-1], D[i-w_j,j-1]+v_j\}$
$T[i,j] = j$ if $D[i,j]=D[i-w_j,j-1]+v_j$
$T[i,j] = 0$ if $D[i,j]=D[i,j-1]$
Then, we can construct an optimal solution by tracing back the value of D from D[i,n] giving the optimal solution.

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4/1								
2	4/0	5/2		9/2					
3	4/0	5/0	8/3	9/0	12/3	13/3		17/3	
4	4/0	5/0	8/0	9/0	12/0	13/0	14/4	17/0	18/4
5	4/0	5/0	8/0	9/0	12/0	13/0	15/5	17/0	19/5

values of D[i,j]/T[i,j]

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4/1								
2	4/0	5/2		9/2					
3	4/0	5/0	8/3	9/0	12/3	13/3		17/3	
4	4/0	5/0	8/0	9/0	12/0	13/0	14/4	17/0	18/4
5	4/0	5/0	8/0	9/0	12/0	13/0	15/5	17/0	19/5

D[i,j]/T[i,j]の値

最適解の値はD[5,10]=19
D[5,10]=19, T[5,10]=5≠0, 品物5を出力.
w_5=6だから，その前はD[4,10-6]=D[4,4],
D[4,4]=8, T[4,4]=0, 何も出力しない．その前はD[3,4]=8
D[3,4]=8, T[3,4]=3≠0, 品物3を出力.
w_3=4だから，その前はD[2,4-4]=D[2,0].
重量の和が0になったので，ここで終わり.
結局，最適解を構成する品物の集合は{3,5}.

k	2	3	4	5	6	7	8	9	10
0	0	0	0	0	0	0	0	0	0
1	4/1								
2	4/0	5/2		9/2					
3	4/0	5/0	8/3	9/0	12/3	13/3		17/3	
4	4/0	5/0	8/0	9/0	12/0	13/0	14/4	17/0	18/4
5	4/0	5/0	8/0	9/0	12/0	13/0	15/5	17/0	19/5

D[i,j]/T[i,j]の値

The value of an optimal solution is given by D[5,10]=19.
D[5,10]=19, T[5,10]=5≠0, output object 5.
Since w_5=6, its predecessor is D[4,10-6]=D[4,4],
D[4,4]=8, T[4,4]=0, output nothing. The predecessor is D[3,4]=8.
D[3,4]=8, T[3,4]=3≠0, output object 3.
Since w_3=4, its predecessor is D[2,4-4]=D[2,0].
Now the total weight becomes 0, and thus this is the end.
After all, the set of objects for an optimal solution is {3,5}.

アルゴリズムP26-A1:
入力：n個の荷物o_i(i=1, ... , n)の重さw_iと価値v_i，制限重量C

```
for(i=0; i<=C; i++)
    D[0,i] = T[0,i]=0;
for(k=1; k<=n; k++)
  for(i=1; i<=C; i++)
    if(i＜w_k){ D[k,i] = D[k-1,i]; T[k,i]=0;}
    else {
      if(D[k-1,i-w_k]+v_k > D[k-1,i])
        {D[k,i] = D[k-1,i-w_k]+v_k; T[k,i]=k;}
      else
        {D[k,i] = D[k-1, i]; T[k,i]=0;}
    }
k=0;
for(i=1; i<=C; i++)
  if(D[n,i]>D[n, k]) k = i;
for(i=n; i>0 && k>0; i--)
  if( T[i,k] > 0) {
    T[i,k]を出力; k = k - w_i;
  }
```


Algorithm P26-A1:
Input：n objects o_i(i=1, ... , n): weight w_i and price v_i, capacity C.

```
for(i=0; i<=C; i++)
    D[0,i] = T[0,i]=0;
for(k=1; k<=n; k++)
  for(i=1; i<=C; i++)
    if(i＜w_k){ D[k,i] = D[k-1,i]; T[k,i]=0;}
    else {
      if(D[k-1,i-w_k]+v_k > D[k-1,i])
        {D[k,i] = D[k-1,i-w_k]+v_k; T[k,i]=k;}
      else
        {D[k,i] = D[k-1, i]; T[k,i]=0;}
    }
k=0;
for(i=1; i<=C; i++)
  if(D[n,i]>D[n, k]) k = i;
for(i=n; i>0 && k>0; i--)
  if( T[i,k] > 0) {
    Output T[i,k]; k = k - w_i;
  }
```


計算時間の解析

アルゴリズムの構造より，計算時間は明らかに
O(nC)
である.
(1) 重量制約Cが荷物の個数nの多項式程度のとき
⇒ この計算時間はnに関する多項式となる.
(2) Cの値がnに比べて非常に大きいとき
Cの値自身はlog Cビットで表現可能
⇒入力の指数関数に比例する時間となる.
擬似多項式時間アルゴリズムと呼ぶ.

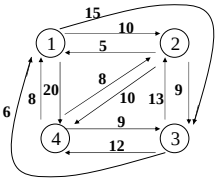
演習問題E10-3：アルゴリズムP26-A1では2次元配列を2つ用いているが，そのうちの一方は1次元配列にできることを示せ.

Analysis of Computation Time

From the structure of the algorithm the computation time is given by
O(nC).
(1) If the capacity C is polynomial in the number n of objects
⇒ this computation time is a polynomial in n.
(2) If C is much larger than n.
The value C itself can be represented by log C bits.
⇒Time is proportional to an exponential function in input size.
It is called **a pseudo-polynomial time algorithm.**

Exercise E10-3：Algorithm P26-A1 uses two 2-dimensional arrays. Show that one of them cab be replaced by a one-dimensional array.

問題P27:（行商人問題）
n都市の相互を結ぶ道路網を表す重みつきグラフが与えられたとき，すべての都市を訪れて元の都市に戻ってくる周遊路の中で最短のものを求めよ．

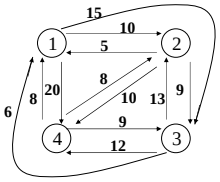


$$L=\begin{pmatrix}0 & 10 & 15 & 20\\ 5 & 0 & 9 & 10\\ 6 & 13 & 0 & 12\\ 8 & 8 & 9 & 0\end{pmatrix}$$

都市iとjの間の距離をL[i,j]とする．

周遊路(1,2,3,4,1)： 長さ=10+9+12+8=39,
周遊路(1,2,4,3,1)： 長さ=10+10+9+6=35,
周遊路(1,3,2,4,1)： 長さ=15+13+10+20=58,
周遊路(1,3,4,2,1)： 長さ=15+12+8+5=40, 等々

Problem P27:（Travelling-Salesperson Problem）
Given a weighted graph for a road network interconnecting n cities, find a shortest closed tour starting from a city and coming back to it after visiting every city.



$$L=\begin{pmatrix}0 & 10 & 15 & 20\\ 5 & 0 & 9 & 10\\ 6 & 13 & 0 & 12\\ 8 & 8 & 9 & 0\end{pmatrix}$$

L[i,j] is the distance between city i and city j.

tour(1,2,3,4,1)： length=10+9+12+8=39,
tour(1,2,4,3,1)： length=10+10+9+6=35,
tour(1,3,2,4,1) : length =15+13+10+20=58,
tour(1,3,4,2,1) : length =15+12+8+5=40, etc.

周遊路は全部で何通りあるだろう．

どの2都市間にも道があるなら，(n-1)!通りの周遊路が存在．
Stirlingの公式によると
$n! \fallingdotseq \sqrt{(2\pi n)}(n/e)^n$
これは大雑把にはn^nという指数関数．

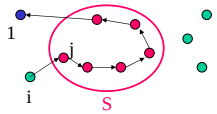
都市を1, 2, ..., nと番号づけ，必ず都市1から出発して都市1に戻ってくるものとする．
全都市の集合をN={1, 2, ... , n}とし，その部分集合をSとする．
都市1を含まない部分集合Sに対して，
g(i, S) = 都市iから出発して，Sの都市をすべて通って都市1に戻る経路の中での最短路の長さ，
ただし，iはSに属さないこと，
と定めると，求める最短周遊路の長さは
g(1, {2, 3, ... , n})
として与えられることになる．

How many tours are there in total?

If there is a road between any two cities, then there are (n-1)! tours.
Using the Stirling's formula, we have
$n! \fallingdotseq \sqrt{(2\pi n)}(n/e)^n$.
This is roughly an exponential function n^n.

Numbering the cities as 1, 2, ... , n, and assume that we start at city 1 and come back to it.
The set of all cities: N={1, 2, ... , n}. S is a subset of N.
For a subset S not containing city 1,
g(i, S) = the length of a shortest path from city i coming back to city 1 through every city of S.
Note that i does not belong to S.
Then, the length of the shortest tour is given as
g(1, {2, 3, ... , n}).

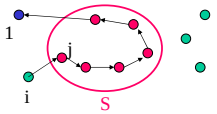
g(i, S)を再帰的に定義しよう



動的計画法を適用するためには，**部分問題に対する解が最適解に含まれるように最適解を再帰的に定義しなければならない．**

Sの中で都市iの次となる都市の候補をjとしたとき，
都市1までの最適な経路の長さは
g(j, S-{j})
で与えられる．よって，g(i, S)に対する漸化式として
$g(i, S) = \min_{j\in S}\{L[i,j] + g(j, S-\{j\})\}$
を得る．ただし，iはSの要素ではない．
L[i,j]は都市i,j間の距離．

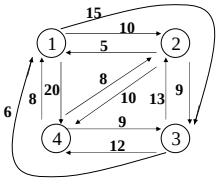
Let's define g(i, S) recursively!



To apply Dynamic Programming, an optimal solution must be defined recursively so that it includes a solution to a subproblem.

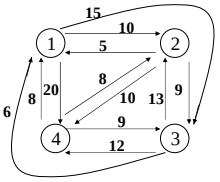
If j is a candidate among S for the city after city i, the length of an optimal path to city 1 is given by
g(j, S-{j}).
Thus, the recurrence equation for g(i, S) becomes
$g(i, S) = \min_{j\in S}\{L[i,j] + g(j, S-\{j\})\}$,
where i is not an element of S.
L[i,j] denotes the distance between city i and city j.

|S|=0
g(2, Φ)=L[2,1]=5
g(3, Φ)=L[3,1]=6
g(4, Φ)=L[4,1]=8
|S|=1
g(2, {3})=L[2,3]+g(3, Φ)=15
g(2, {4})=L[2,4]+g(4, Φ)=18
g(3, {2})=L[3,2]+g(2, Φ)=18
g(3, {4})=L[3,4]+g(4, Φ)=20
g(4, {2})=L[4,2]+g(4, Φ)=13
g(4, {3})=L[4,3]+g(3, Φ)=15



|S|=2
g(2, {3,4})=min{L[2,3]+g(3, {4}), L[2,4]+g(4,{3})}=min{29,25}=25
g(3, {2,4})=min{L[3,2]+g(2, {4}), L[3,4]+g(4,{2})}=min{31,25}=25
g(4, {2,3})=min{L[4,2]+g(2, {3}), L[4,3]+g(3,{2})}=min{23,27}=23
|S|=3
g(1, {2,3,4})=min{L[1,2]+g(2, {3,4}), L[1,3]+g(3, {2,4}),L[1,4]+g(4, {2,3})}
= min{35, 40, 43} = 35.
よって，最短周遊路の長さは35である.

|S|=0
g(2, Φ)=L[2,1]=5
g(3, Φ)=L[3,1]=6
g(4, Φ)=L[4,1]=8
|S|=1
g(2, {3})=L[2,3]+g(3, Φ)=15
g(2, {4})=L[2,4]+g(4, Φ)=18
g(3, {2})=L[3,2]+g(2, Φ)=18
g(3, {4})=L[3,4]+g(4, Φ)=20
g(4, {2})=L[4,2]+g(4, Φ)=13
g(4, {3})=L[4,3]+g(3, Φ)=15



|S|=2
g(2, {3,4})=min{L[2,3]+g(3, {4}), L[2,4]+g(4,{3})}=min{29,25}=25
g(3, {2,4})=min{L[3,2]+g(2, {4}), L[3,4]+g(4,{2})}=min{31,25}=25
g(4, {2,3})=min{L[4,2]+g(2, {3}), L[4,3]+g(3,{2})}=min{23,27}=23
|S|=3
g(1, {2,3,4})=min{L[1,2]+g(2, {3,4}), L[1,3]+g(3, {2,4}),L[1,4]+g(4, {2,3})}
= min{35, 40, 43} = 35.
Therefore, the length of an optimal tour is 35.

アルゴリズムP27-A0:
入力：n都市間の距離を表すグラフ
U={1, 2, ... , n}とする.
for(i=2; i<=n; i++)
g(i, Φ)=L[1,i];
for(k=1; k<n; k++){
for 1を含まないサイズkのすべての部分集合S {
for Sに含まれないすべてのiについて
g(i, S) = $\min_{j \in S}${L[i,j] + g(j, S-{j})}
}
return g(1, {2, 3, ... , n});

演習問題E10-4: 上記のアルゴリズムの計算時間と記憶量をnの関数で表せ.

Algorithm P27-A0:
Input: A graph representing distances among cities.
U={1, 2, ... , n}.
for(i=2; i<=n; i++)
g(i, Φ)=L[1,i];
for(k=1; k<n; k++){
for each subset S of size k not containing 1 {
for each i not contained in S
g(i, S) = $\min_{j \in S}${L[i,j] + g(j, S-{j})}
}
return g(1, {2, 3, ... , n});

Exercise E10-4: Express the computation time and amount of storage of the above algorithm as functions of n.