実践的アルゴリズム理論
Theory of Advanced Algorithms

8. 動的計画法(2)

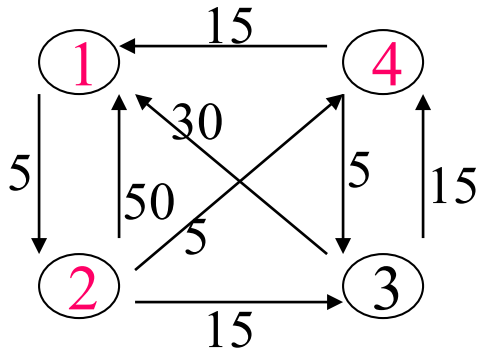
担当:上原隆平

Theory of Advanced Algorithms
実践的アルゴリズム理論

8. Dynamic Programming (2)

Ryuhei Uehara

問題P24:（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

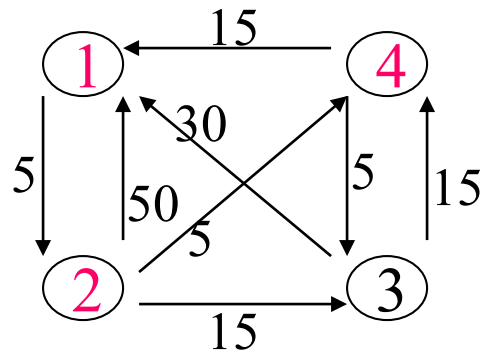
入力：n個の頂点とm本の辺からなる重みつきグラフ
出力：任意の1点から残りのすべての頂点への距離
計算時間：$O(m + n \log n)$時間，または$O(n^2)$時間

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 P24: (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 $O(n^3)$.

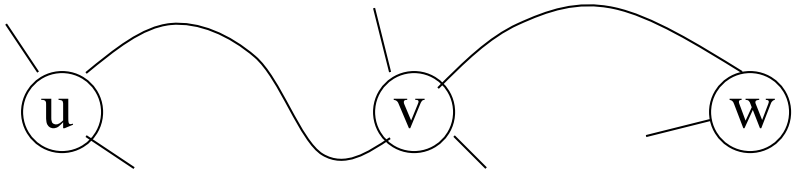
Since the number m of edges is $O(n^2)$, it takes $O(n^3)$ in the worst case.

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．

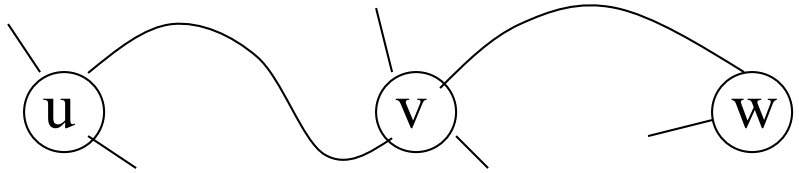


理由：uからvまでの部分経路が最短でなければ，
この部分経路を最短経路で置きかえると，uからw
へのより短い経路が得られる．これは矛盾．

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P24-A0:
距離行列をD[0,i,j]とする.
for(k=1; k<=n; k++)
　すべての(i,j)について
　　D[k,i,j] = min{ D[k-1,i,j], D[k-1,i,k]+D[k-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

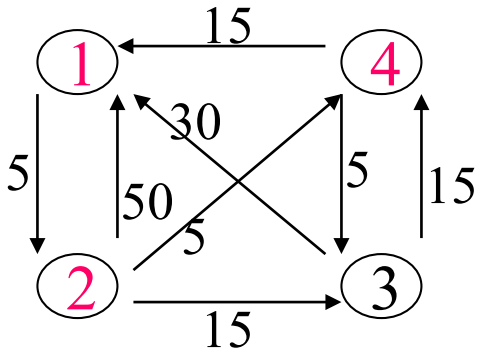
$$D_k[i,j] = \min\{ D_{k-1}[i,j], D_{k-1}[i,k] + D_{k-1}[k,j] \}$$

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.

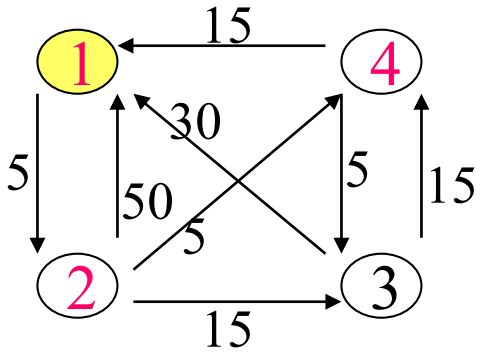
Algorithm P24-A0:

```
Let D[0,i,j] be the distance matrix.
for(k=1; k<=n; k++)
    for each (i,j)
        D[k,i,j] = min{ D[k-1,i,j], D[k-1,i,k]+D[k-1,k,j] }
```



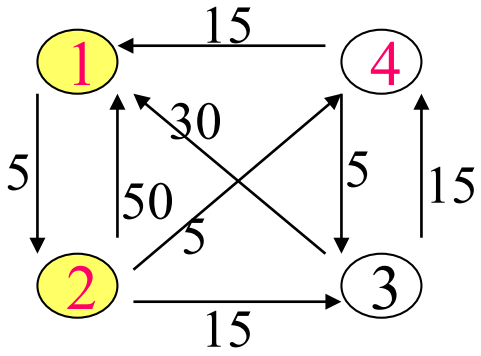
$$D_0=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

距離行列



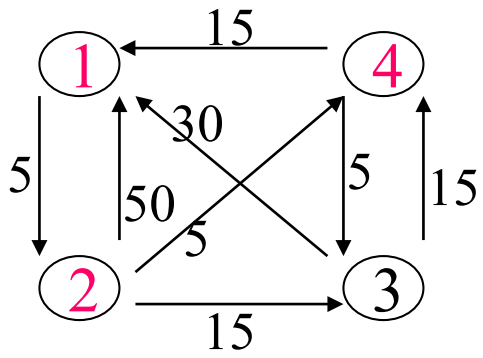
$$D_1=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

途中で頂点1を
通ってもよい.



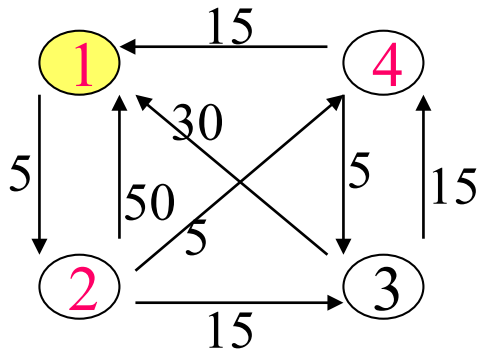
$$D_2=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

途中で頂点1,2を
通ってもよい.



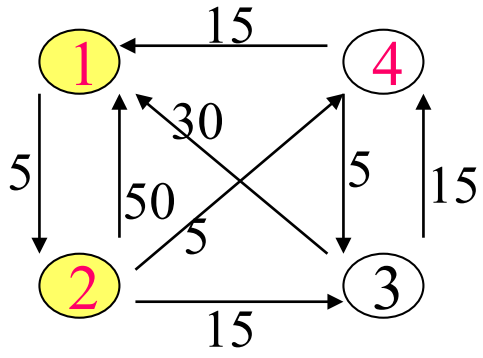
$$D_0 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & \infty & 0 & 15 \\ 15 & \infty & 5 & 0 \end{pmatrix} \end{array}$$

distance matrix



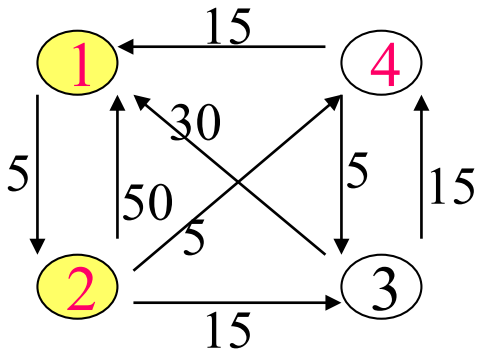
$$D_1 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & \infty & \infty \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertex 1 can be passed through



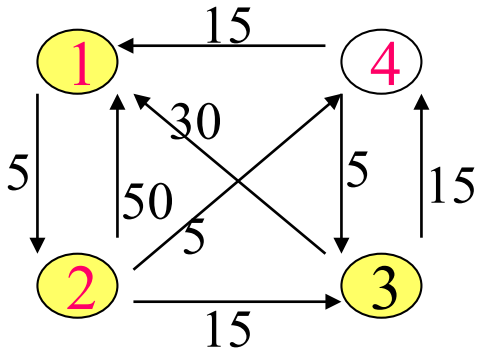
$$D_2 = \begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array} \begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

vertices 1 and 2 can be passed through



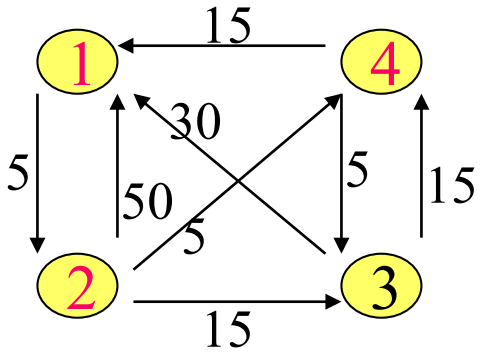
$$D_2=\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \end{array}\begin{array}{c} \begin{array}{cccc} 1 & 2 & 3 & 4 \end{array} \\ \begin{pmatrix} 0 & 5 & 20 & 10 \\ 50 & 0 & 15 & 5 \\ 30 & 35 & 0 & 15 \\ 15 & 20 & 5 & 0 \end{pmatrix} \end{array}$$

途中で頂点1,2を
通ってもよい.



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

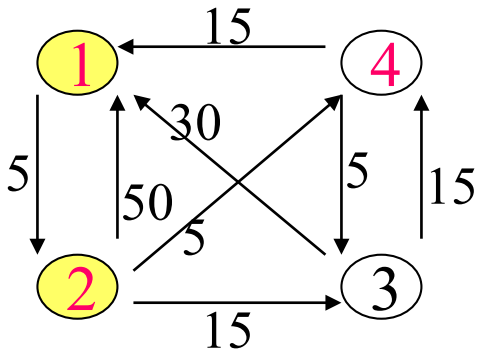
途中で
頂点1,2,3を
通ってもよい.



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

途中で
頂点1,2,3,4を
通ってもよい

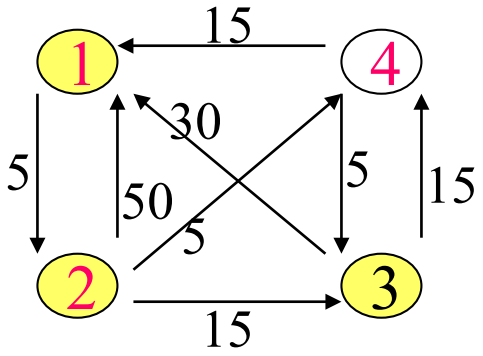
最終的な距離行列



$D_2=$

	1	2	3	4
1	0	5	20	10
2	50	0	15	5
3	30	35	0	15
4	15	20	5	0

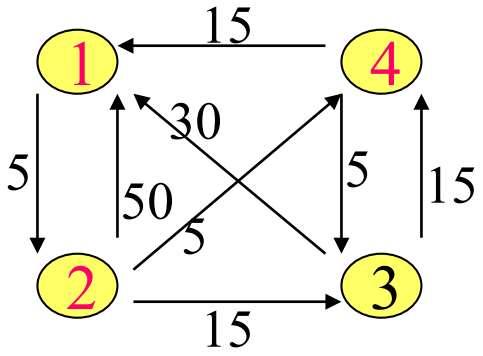
vertices 1 and 2 can be passed through



$D_3=$

	1	2	3	4
1	0	5	20	10
2	45	0	15	5
3	30	35	0	15
4	15	20	5	0

vertices 1, 2, and 3 can be passed through



$D_4=$

	1	2	3	4
1	0	5	15	10
2	20	0	10	5
3	30	35	0	15
4	15	20	5	0

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.

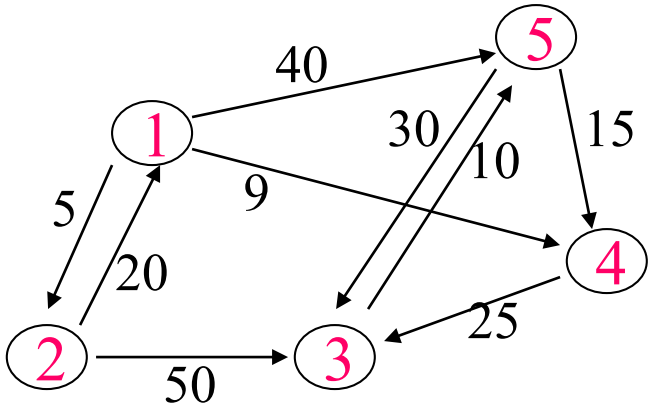
➔when we have computed D_k, we don't need to store D_{k-1}. Hence, only one array is sufficient.

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

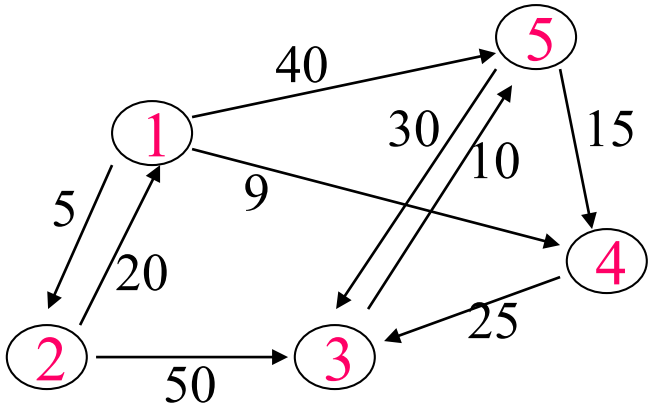
$P_k[i,j]$ = the vertex number immediately before the vertex j on the shortest path from i to j through {1, 2, ... , k}.

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.



問題P25:（最適2分探索木の構成）
n個のデータを2分探索木に蓄えるのに，各データに対する検索の確率（頻度）が予め予想できるとき，探索のための比較回数（の期待値）が最小になるように2分探索木を構成せよ.

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

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

Problem P25: (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

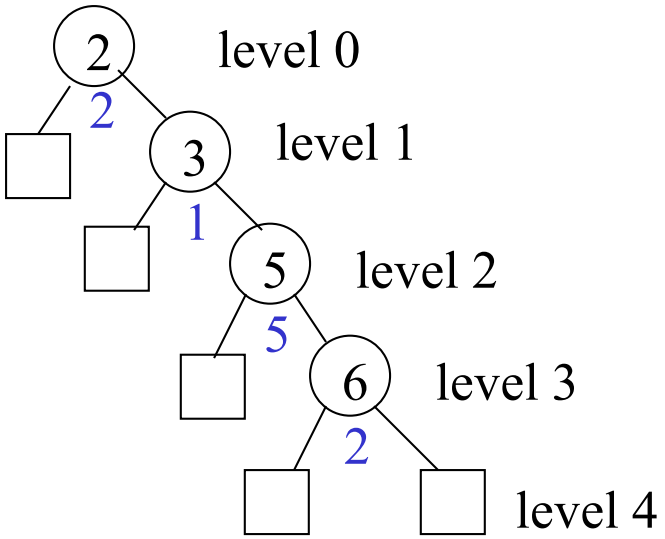
(assuming the level of the root node is 0)

Therefore, the cost of a search tree (expected number of comparisons) is given by

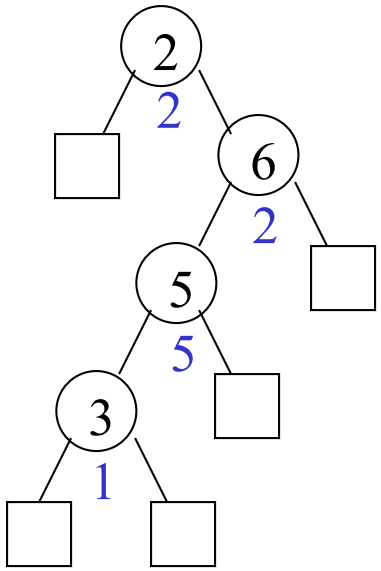
$$\text{Cost of search tree} = \sum p_i \times [\text{level}(a_i) + 1]$$

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



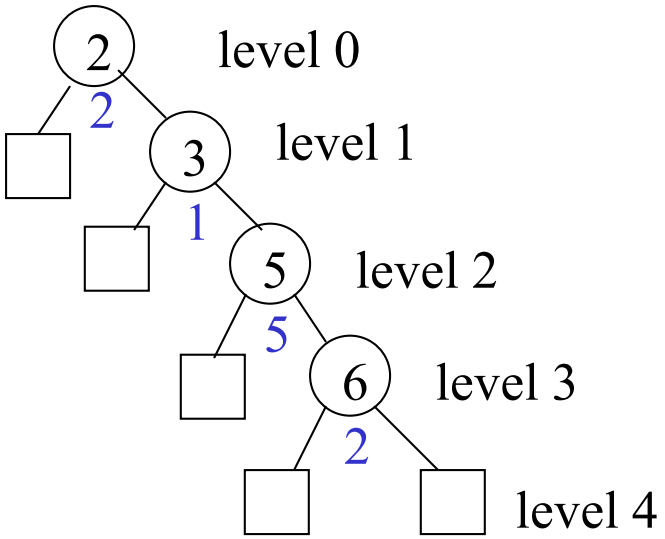
コスト=(2*1+1*2+5*3+2*4)/10
=2.7



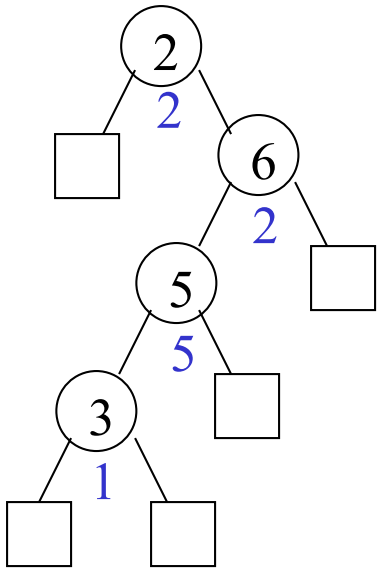
コスト=(2*1+2*2+5*3+1*4)/10
= 2.5

Examlpe:

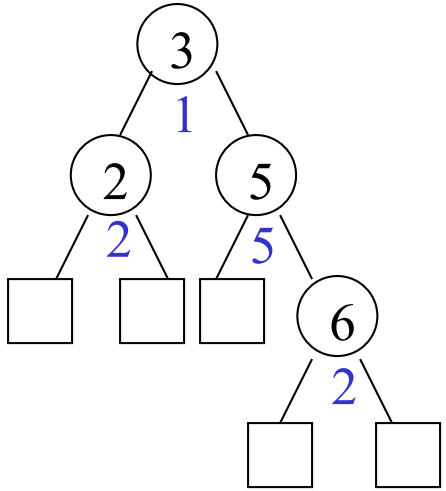
	a[1]	a[2]	a[3]	a[4]
S	2	3	5	6
	2/10	1/10	5/10	2/10
	p_1	p_2	p_3	p_4

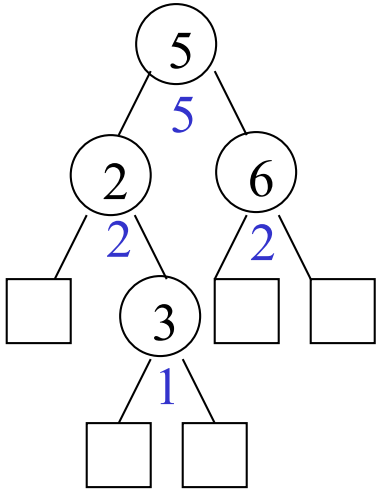


cost=(2*1+1*2+5*3+2*4)/10
=2.7



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

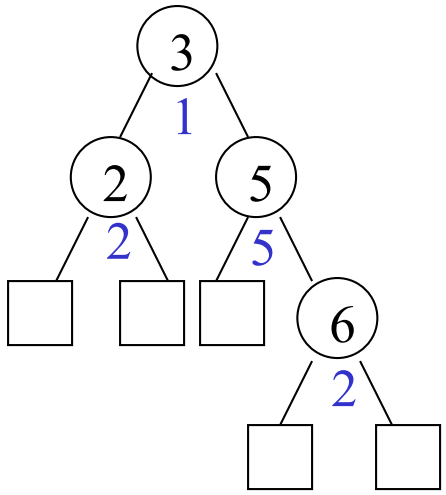


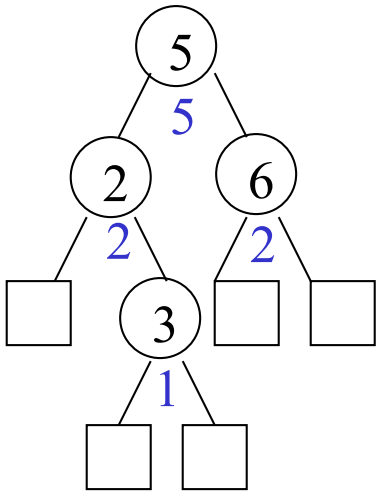


コスト=(1*1+(2+5)*2+2*3)/10 =2.1

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

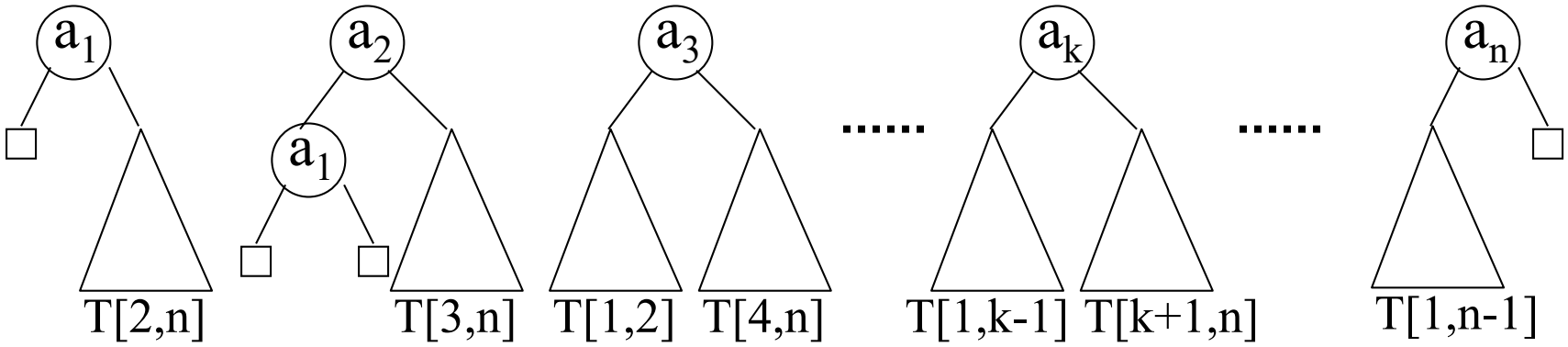
If we enumerate all search trees and compute their costs, then we can find an optimal search tree. But it is not efficient to enumerate all of them.

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



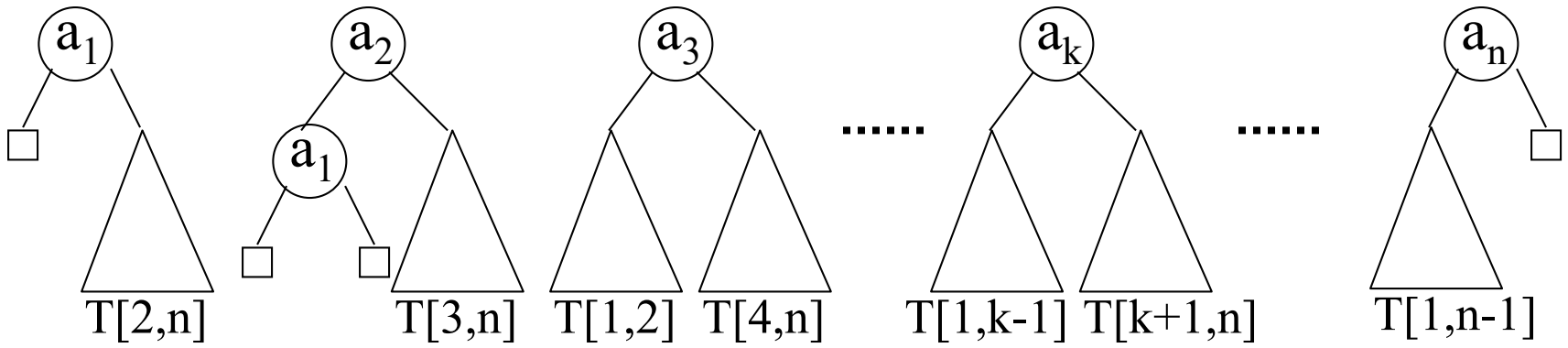
$T[2,n], T[3,n], \ldots, T[1,2], T[4,n], \ldots, T[1,n-1]$がすべて
求まっていれば，上のそれぞれの木のコストを計算できる．
最小コストの木を選べば，その根a_kを決めることが可能．

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

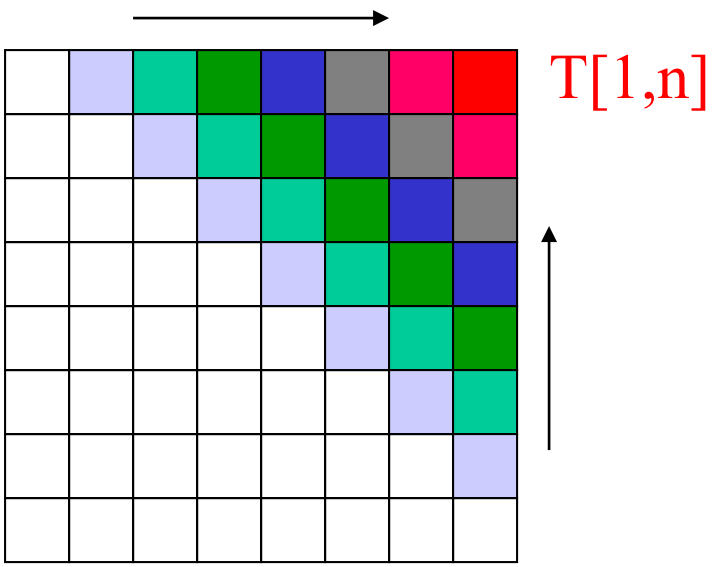
Enumerating all possibilities:



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.

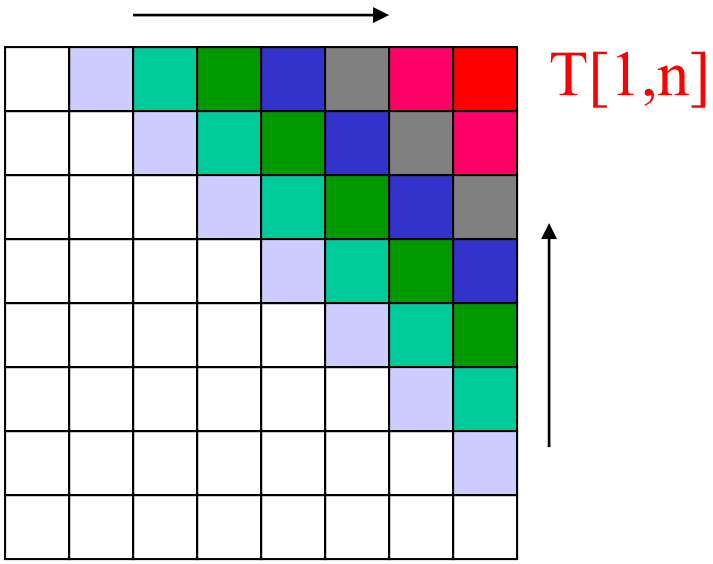
ボトムアップの形式で最適解の値を求める.

{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.



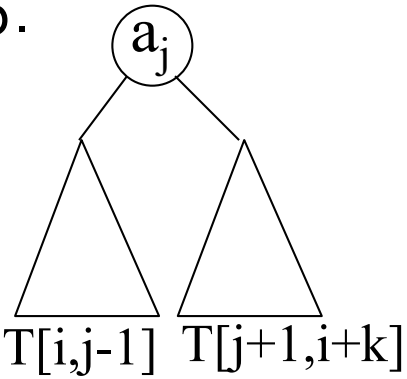
Computing the value of optimal solution in a bottom-up fashion

{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$\{a_i, a_{i+1}, \ldots, a_{i+k}\}$に対する最小コスト木
だから，その根は，$a_i, a_{i+1}, \ldots, a_{i+k}$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．



$T[i,j-1] = \sum p_m \times [\text{level}(a_m) + 1]$
レベルを1だけ増やすと，
$T'[i,j-1] = \sum p_m \times [\text{level}(a_m) + 2] = T[i,j-1] + \sum p_m$
つまり，T[i,j-1]に$p_i + p_{i+1} + \ldots + p_{j-1}$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：

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

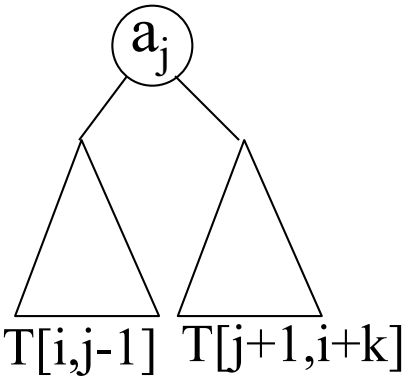
How to compute T[i, i+k]

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}, ... , 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, ... , 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,

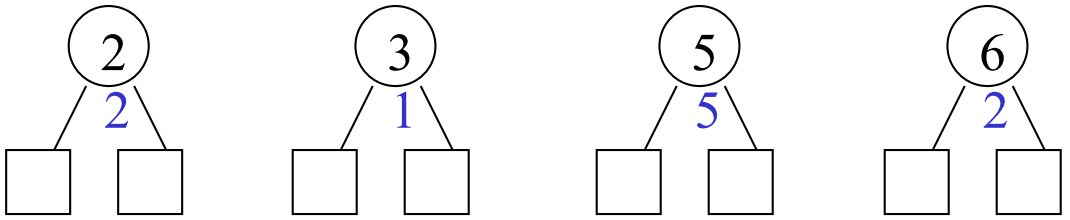
the cost when a_j is the root is given by the following:
C[i,j-1]+C[j+1,i+k]+ W[i, i+k]

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

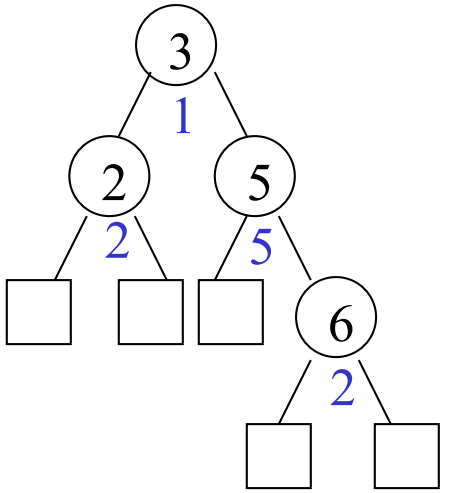
$$C[i,i+k] = \min\{ C[i+1,i+k]+W[i,i+k],$$
$$\min\{C[i,j-1]+C[j+1,i+k]+W[i,i+k],\ j=i+1, \ldots, i+k-1\},$$
$$C[i,i+k-1]+W[i,i+k]\}$$
$$k=1, 2, \ldots, n-i$$

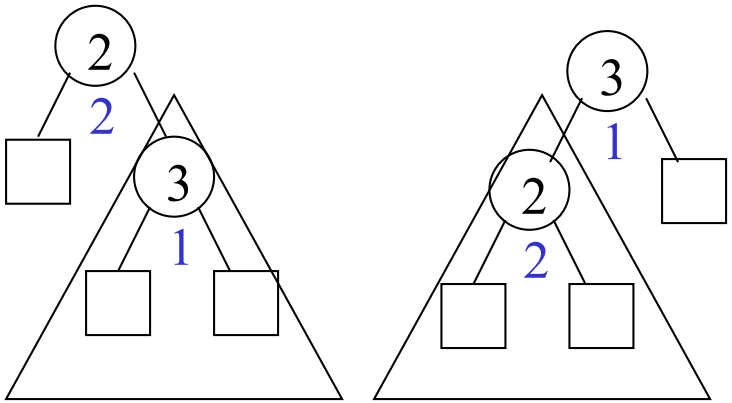
$C[i,j] = \{a_i, a_{i+1}, \ldots, a_j\}$に対する最小木T[i,j]のコスト

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



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]

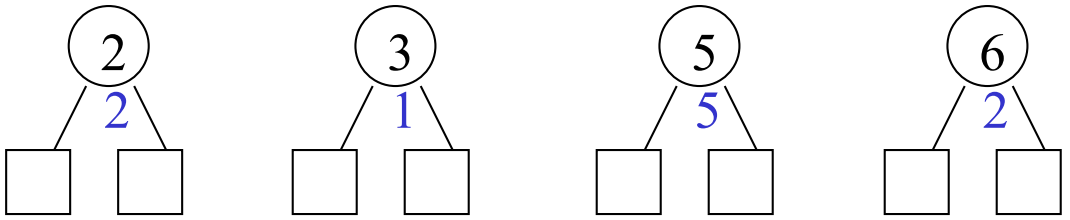
コスト
=0.2+C[2,2]+W[2,2]
=0.2+0.1+0.1=0.4

T[1,1]

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

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

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

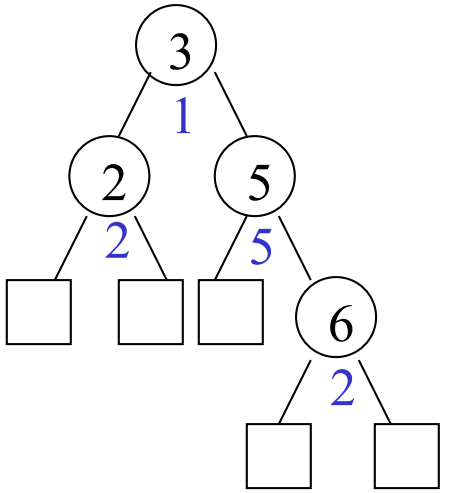


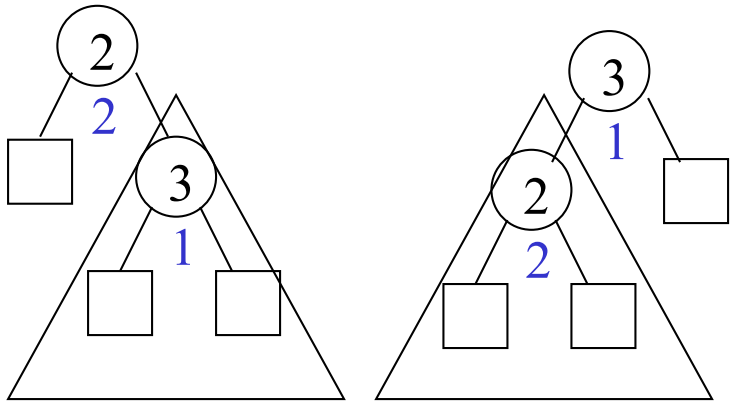
T[1,1] C[1,1]=0.2

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

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

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





T[2,2]

コスト

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

=0.2+0.1+0.1=0.4

T[1,1]

コスト

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

=0.1+0.2+0.2=0.5

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$I = \{w_1, \dots, w_n; v_1, \dots, v_n; C\}$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 $\sum_{i \in S} w_i \leqq C$

を満たすSの中で

価値の総和 $\sum_{i \in S} v_i$

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.

ナップサック問題はNP完全
問題だったはずでは…？

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: $I = \{w_1, \ldots, w_n; v_1, \ldots, v_n; C\}$. A solution is represented by a subset S of {1,2,...,n}.

An optimal solution is such a set S satisfying the

Capacity constraint $\sum_{i \in S} w_i \leqq C$

and maximizing

total sum of prices $\sum_{i \in S} v_i$.

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: $(w_1, \ldots, w_5)=(2,3,4,5,6)$, $(v_1, \ldots, v_5)=(4,5,8,9,11)$, $C=10$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

$V[1] \leqq V[2] \leqq \cdots \leqq V[n]$

が成り立つ. この例では, 次のようになる.

$V[1] = v_1=4, w_1=2 \leqq C$,

$V[2] = v_1+v_2=4+5=9, w_1+w_2=2+3 \leqq C$,

$V[3] = v_1+v_2+v_3=4+5+8=17, w_1+w_2+w_3=2+3+4 \leqq C$,

$V[4] = v_1+v_2+v_4=4+5+9=18, w_1+w_2+w_4=2+3+5 \leqq C$,

$V[5] = v_3+v_5=8+11=19, w_3+w_5=4+6 \leqq C$.

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

$V[1] \leqq V[2] \leqq \cdots \leqq V[n]$.

In this example, we have

$V[1] = v_1=4$, $w_1=2 \leqq C$,

$V[2] = v_1+v_2=4+5=9$, $w_1+w_2=2+3 \leqq C$,

$V[3] = v_1+v_2+v_3=4+5+8=17$, $w_1+w_2+w_3=2+3+4 \leqq C$,

$V[4] = v_1+v_2+v_4=4+5+9=18$, $w_1+w_2+w_4=2+3+5 \leqq C$,

$V[5] = v_3+v_5=8+11=19$, $w_3+w_5=4+6 \leqq C$.

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.
$\Rightarrow$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: $(w_1, \ldots, w_5)=(2,3,4,5,6)$, $(v_1, \ldots, v_5)=(4,5,8,9,11)$, 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＜wi) D[k,i] = D[k-1,i];
      else {
         if(D[k-1,i-wi]+vi > D[k-1,i])
            D[k,i] = D[k-1,i-wi]+vi;
         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．

$D[i,j] = \max\{D[i, j-1], D[i-w_j,j-1]+v_j\}$

$T[i,j] = j$　$D[i,j]=D[i-w_j,j-1]+v_j$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.

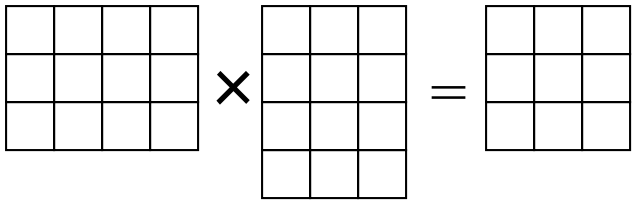
問題P27:（連鎖行列積）

n個の行列の系列$<A_1, A_2, \ldots, A_n>$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，$(10 \times 20 \times 5)+(10 \times 5 \times 25)=2250$

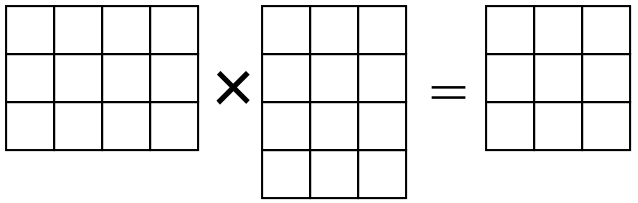
$(A_1 \times (A_2 \times A_3))$の順だと，$(10 \times 20 \times 25)+(20 \times 5 \times 25)=7500$

なので，前者の方が演算回数は少ない.

Problem P27: (Chained Matrix Product)

Given a sequence of n matrices $<A_1, A_2, \ldots, A_n>$, 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 $p \times q$ matrix and $q \times r$ matrix is a $p \times r$ matrix using $p \times q \times r$ operations (multiplication and additioN).



3×4 $\quad$ 4×3 $\quad$ 3×3

Example: $A_1 = 10 \times 20$ matrix, $A_2 = 20 \times 5$ matrix, $A_3 = 5 \times 25$ matrix.

$((A_1 \times A_2) \times A_3)$ require $(10 \times 20 \times 5) + (10 \times 5 \times 25) = 2250$ ops.

$(A_1 \times (A_2 \times A_3))$ requires $(10 \times 20 \times 25) + (20 \times 5 \times 25) = 7500$ 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．

$P(1) = 1$

$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$

Richard P. Stanley によれば，Catalan数には207通りもの解釈がある！
(http://www-math.mit.edu/~rstan/ec/)

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

$P(1) = 1$

$P(n) = \sum_{k=1}^{n-1} P(k)P(n-k)$

Due to Richard P. Stanley, the Catalan number has 207 representations!
(http://www-math.mit.edu/~rstan/ec/)

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 matrixes

$((A_1 \times (A_2 \times A_3)) \times A_4)$ 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

$q_1=p_2$, $q_2=p_3$, ... , $q_n=p_{n+1}$

the product of those matrices is defined. Thus, we only specify p_1, p_2, ... , 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

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

アルゴリズムP27-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 P27-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.