11. 動的計画法と擬多項式時間アルゴリズム

これまでは、主に、「問題がいかに難しいか」を理論的に証明する方法について学んできた。このような難しい問題として、「計算不可能な問題」や「NP完全問題」があった。

計算不可能な問題は計算機では原理的に解決不可能な問題であり、この問題の存在は計算機の限界を示している。

また、「NP完全問題」は、原理的には計算機で解決可能であるが、経験的に実用的な解決が望めない問題である。ここで、実用的の基準は、「多項式時間アルゴリズムの存在するか？」であった。NP完全問題に対する多項式時間の存在は今だに未解決であるが、これまでの数学者や計算機科学者の努力の結果から、NP完全問題に対する多項式時間アルゴリズムは存在しないという見方をしている研究者が多い。

このような、状況ではあるが、ここからは困難な問題に対する解決の糸口をいくつか紹介する。

まず、NP完全問題に対して、擬多項式時間アルゴリズムと呼ばれるものを紹介する。

11－1. 部分和問題(SUBSET SUM Problem)

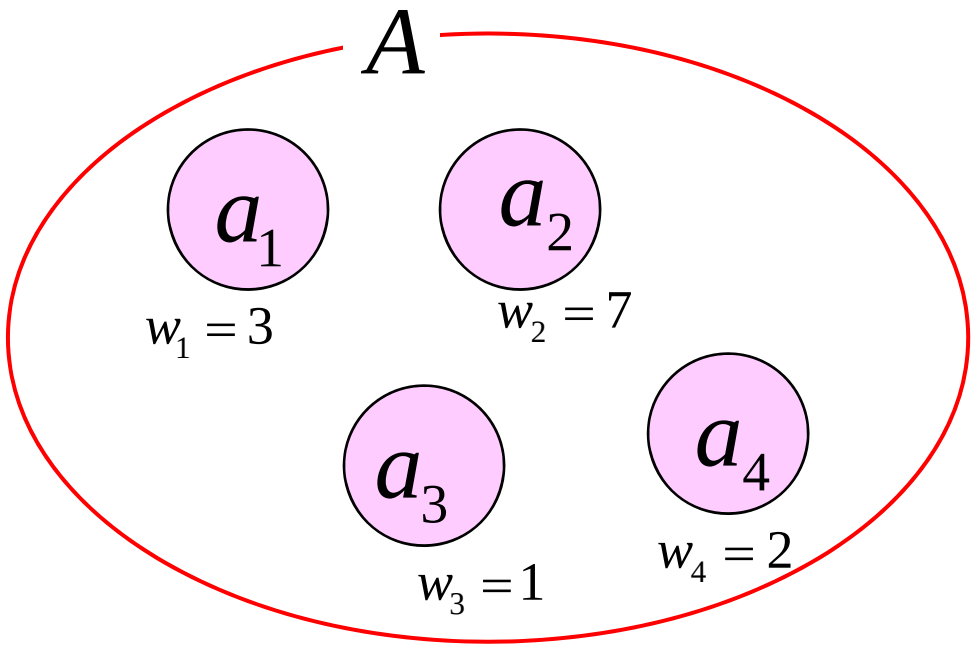
まず、NP完全問題である部分和問題を示す。
(NP完全性の証明は行わない。文献参照。)

名称:部分和問題(SUBSET SUM)
インスタンス:有限集合 $A = \{a_1, a_2, \cdots, a_n\}$ 、
重み $W = (w_1, w_2, \cdots, w_n)$ 、
目標値 K

問: $\sum_{a_i \in A'} w_i = K$ となるような部分集合 $A' \subseteq A$ が
存在するか？

添え字が同じ集合要素の重みを表す。

インスタンス例



$K = 9$

$A = \{a_1, a_2, a_3, a_4\}$

$W = (3, 7, 1, 2)$

$K = 9$

入力サイズ

インスタンスは、

$$A = \{a_1, a_2, \cdots, a_n\} \quad W = (w_1, w_2, \cdots, w_n) \quad K$$

であるので、これをTMのテープ上へ表現することを考える。

集合Aは要素数だけが問題であるので、$\log n$の長さで表現可能。

Wは、各重みを表現しなければならないので、
長さ $\sum_{i=1}^{n} \log w_i$ が必要。

最後に目標値Kの表現に、長さ $\log K$ が必要。

$\log n$	$\log w_1$	$\log w_2$	$\cdots$	$\log w_n$	$\log K$	

よって、入力サイズは、$\log n + \sum_{i=1}^{n} \log w_i + \log K$

ここで、$w_{\max} = \max_{1 \le i \le n} w_i$ とする。

$$\log n + \sum_{i=1}^{n} \log w_i + \log K \le \log n + n \log w_{\max} + \log K$$

なので、 $n \log w_{\max} + \log K$ を入力サイズとみなす。

部分和問題の指数時間アルゴリズム

問題を計算機に解かせるためには、
問題を数理的に記述する必要がある。
ここでは、その練習もかねて、指数時間アルゴリズムを構築していく。

次のような特徴ベクトルを用意する。

$$\boldsymbol{x} = (x_1, x_2, \cdots, x_n) \in \{0,1\}^n$$

ここで、次のように意味づけする。

$i = 1, 2, \cdots, n$

$$x_i = 1 \Rightarrow a_i \in A'$$
$$x_i = 0 \Rightarrow a_i \notin A'$$

これにより、特徴ベクトル $\boldsymbol{x}$ と部分集合 A' が一対一に対応することがわかる。

したがって、次のようなアルゴリズムが得られる。

アルゴリズムEXP__SUBSETSUM

1. x を2進数とみなし、
   $(0,0,\cdots,0)$ から $(1,1,\cdots,1)$ まで以下を繰り返す。
2. 2進数 x の i ビットが1であるようなものに対して
   w_i の総和を計算する。
   すなわち、$S_x = \sum_{x_i=1} w_i$ を計算する。
3. もし、$S_x = K$ であればYESを出力して終了する。
   そうでなければステップ1に戻る。
4. 1－3の繰り返しがすべて終了したら、NOを出力する。

このアルゴリズムの計算量は、1-3を $O(2^n)$ 回繰り返しており、
２．において $O(n \log w_{\max})$ 時間で総和を計算できるので、

$O(n2^n \log w_{\max})$ 時間

のアルゴリズムである。（多項式時間ではない。）

部分和問題の擬多項式時間アルゴリズム

擬多項式時間アルゴリズムは、
「動的計画法」(Dynamic　Programming、DP)
というアルゴリズム設計技法に基づいている。
まず、DP法の設計方針を書く。

〇部分問題の解を「表」として保存しておく。
〇部分問題の解を組あわせてより大きい問題の解を
　生成する。
〇これらを繰り返して、ボトムアップ的に問題を解決していく。
〇なお、部分問題の解は表として保存してあるので、
　一度計算したものを繰り返して計算する必要がない。

方針

○部分問題としては、添え字の小さい方だけの集合に対する
　部分和問題とする。
○現段階で実現可能性をすべて表として保存する。
○これらの表を、ϕ からはじめて、$\{a_1\},\{a_1,a_2\}$
　と一づつ要素を増やしながら、ボトムアップで大きくしていく。

ここでは、次のインスタンス例に対して、
実際に表を構成していく。

$$A=\{a_1,a_2,a_3,a_4\}$$
$$W=(3,7,1,2)$$
$$K=9$$

表の概形
（アルゴリズムの基礎にあたる。）

重み	0	1	2	3	4	5	6	7	8	9
ϕ	T	F	F	F	F	F	F	F	F	F
$\{a_1\}$										
$\{a_1, a_2\}$										
$\{a_1, a_2, a_3\}$										
$\{a_1, a_2, a_3, a_4\}$										

T:実現可能（True）
F:実現不可能（False)

要素　a_1　に注目した表の更新。

$w_1 = 3$ に注意する。

重み	0	1	2	3	4	5	6	7	8	9
ϕ	T	F	F	F	F	F	F	F	F	F
$\{a_1\}$	T	F	F	T	F	F	F	F	F	F

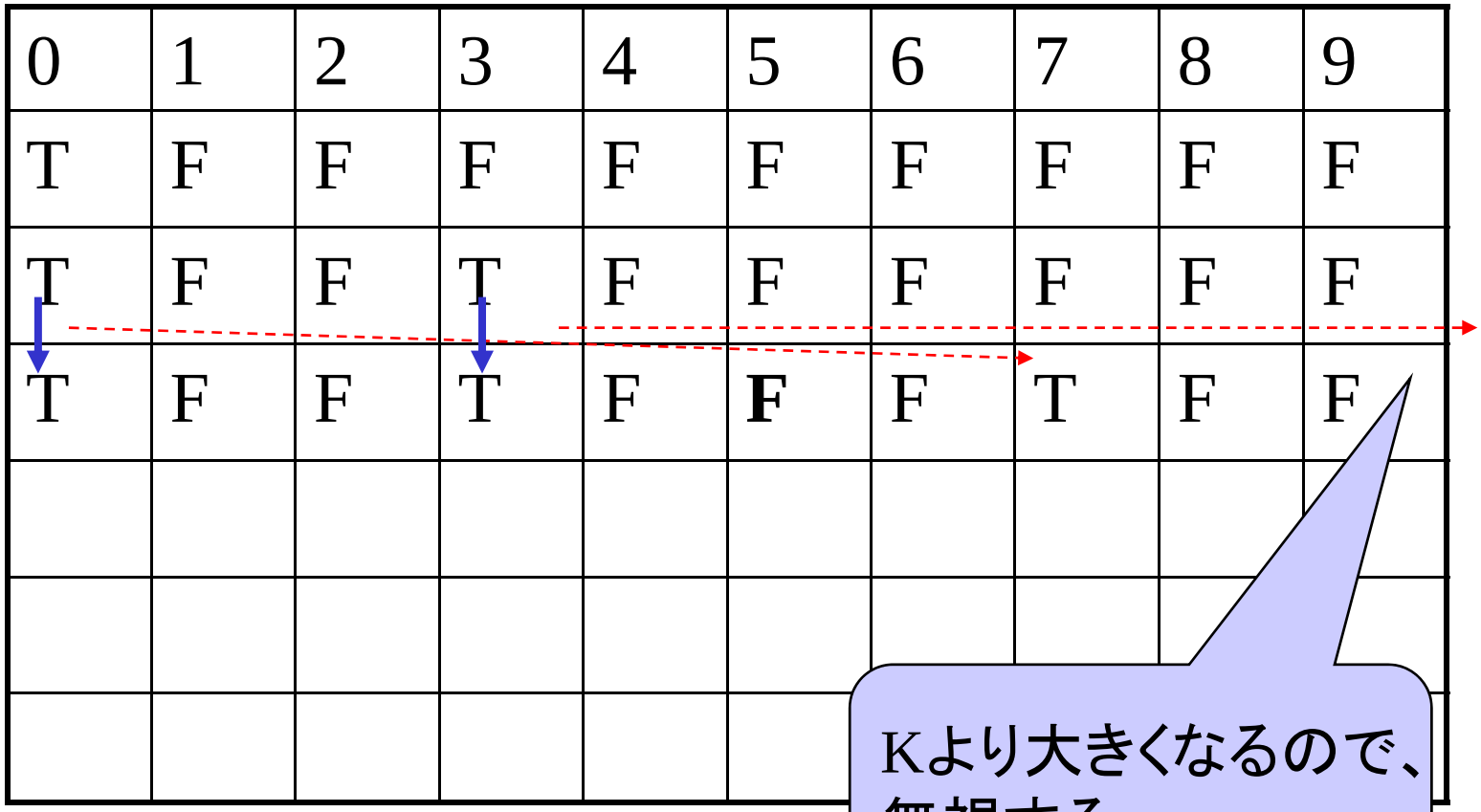
↓　上段で実現可能であるものは、下段でも実現可能。

↘　上段で実現可能であるものを、
w_i 分ずらしたものも実現可能。

要素　a_2　に注目した表の更新。

$w_2 = 7$ に注意する。

重み	0	1	2	3	4	5	6	7	8	9
ϕ	T	F	F	F	F	F	F	F	F	F
$\{a_1\}$	T	F	F	T	F	F	F	F	F	F
$\{a_1, a_2\}$	T	F	F	T	F	**F**	F	T	F	F



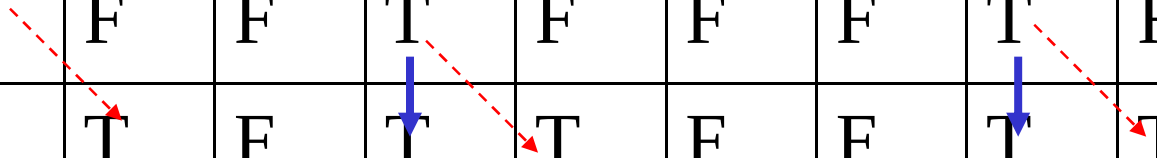
Kより大きくなるので、
無視する。

T:実現可能（True）
F:実現不可能（False)

要素　a_3　に注目した表の更新。

$w_3 = 1$ に注意する。

重み	0	1	2	3	4	5	6	7	8	9
ϕ	T	F	F	F	F	F	F	F	F	F
$\{a_1\}$	T	F	F	T	F	F	F	F	F	F
$\{a_1, a_2\}$	T	F	F	T	F	F	F	T	F	F
$\{a_1, a_2, a_3\}$	T	T	F	T	T	F	F	T	T	F



T:実現可能（True）

F:実現不可能（False)

要素　a_4　に注目した表の更新。

$w_4 = 2$ に注意する。

重み	0	1	2	3	4	5	6	7	8	9
ϕ	T	F	F	F	F	F	F	F	F	F
$\{a_1\}$	T	F	F	T	F	F	F	F	F	F
$\{a_1, a_2\}$	T	F	F	T	F	F	F	T	F	F
$\{a_1, a_2, a_3\}$	T	T	F	T	T	F	F	T	T	F
$\{a_1, a_2, a_3, a_4\}$	T	T	T	T	T	T	T	T	T	T

T:実現可能（True）

F:実現不可能（False)

以上より、インスタンス

$$I = (\{a_1, a_2, a_3, a_4\}, (3, 7, 1, 2), 9)$$

は肯定のインスタンスであることがわかった。
つまり、言語を

$$L_{SUBSETSUM} = \{w \mid w\text{は部分和問題の肯定のインスタンス}\}$$

とすると、

$$I \in L_{SUBSETSUM}$$

である。

練習

次のインスタンスが、肯定のインスタンスであるか、
否定のインスタンスであるかを決定せよ。

（1）

$$A = \{a_1, a_2, a_3, a_4, a_5\}$$
$$W = (7, 3, 6, 13, 5)$$
$$K = 17$$

（2）

$$A = \{a_1, a_2, a_3, a_4, a_5, a_6\}$$
$$W = (7, 3, 6, 13, 5, 14)$$
$$K = 25$$

ここでは、DP法に基づく、部分問題を解くアルゴリズムの計算量を解析する。

なお、一見すると表を埋めていくだけなので、多項式時間アルゴリズムかのように思える。しかし、入力サイズに基づいて考察すると、指数時間であることがわかる。このようなことから、このアルゴリズムは擬多項式時間アルゴリズムであるといわれる。

まず、i 行目、重みw のセルに対して、そのセルに割り当てる論理値を意味する変数を t_{iw} で表す。なお、t_{0w} で空集合に対する表のセルに対する変数を表すものとする。

このとき、セルへの割り当ては次のように決定される。

$i=0$ のとき

$$\begin{cases} t_{00} = T \\ t_{0w} = F \quad w = 1, 2, \cdots, K \end{cases}$$

$i>0$ のとき

$$t_{iw} = t_{(i-1)w} \vee t_{(i-1)(w-w_i)}$$

このこから、各セルを計算するのに必要な計算時間は、次のようになる。

RAMモデル　O(1)時間(定数時間)
TM　O(K)時間

RAMモデルは、任意位置に定数時間でアクセスできる。

ヘッドの移動分

表の要素数は、$(n+1)K = O(nK)$ 個あるので、
RAMモデルでは、

$O(nK)$ 時間

で実行でき、
TMでは、

$O(nK^2)$ 時間

で実行できる。
これは多項式なのであろうか？

もう一度入力サイズをおもいだすと、入力サイズは、

$$n \log w_{\max} + \log K$$

であった。したがって、要素数 n に対しては多項式であるが、目標値 K の入力サイズ $\log K$ に対しては多項式ではない。

$s = \log K$ とすると、

$$K = 2^{\log K} = 2^{s}$$

である。

擬多項式時間アルゴリズムの定義

入力における数値がすべて一進数で与えられるものとしたときに、
その入力サイズの多項式時間で動作するアルゴリズムのことを擬多項式時間アルゴリズムという。

部分和問題における入力サイズは、
数を一進数で表すとすると、
$O(nw_{\max} + K)$
である。

なお、RAMモデルの1ステップをTMで多項式時間でシミュレートできるので、多項式時間という範疇では両者ともに等価である。

11－2. ナップザック問題（KNAPSACK Problem)

ここでは、もう一つのNP完全問題にたいして、
動的計画法を用いた擬多項式時間アルゴリズムを示す。
すなわち、NP完全問題であるナップザック問題を解く
動的計画法に基づいた解法を示す。
（NP完全性の証明は行わない。文献参照。）

名称：ナップザック（KNAPSACK)
インスタンス：有限集合 $A = \{a_1, a_2, \cdots, a_n\}$、
大きさ $S = (s_1, s_2, \cdots, s_n)$、
価値 $V = (v_1, v_2, \cdots, v_n)$
バッケット B
目標値 K
問：以下の条件を満たす部分集合 $A' \subseteq A$ が
存在するか？

$$\sum_{a_i \in A'} s_i \leq B \quad AND \quad \sum_{a_i \in A'} v_i \geq K$$

バケットの大きさによる制限

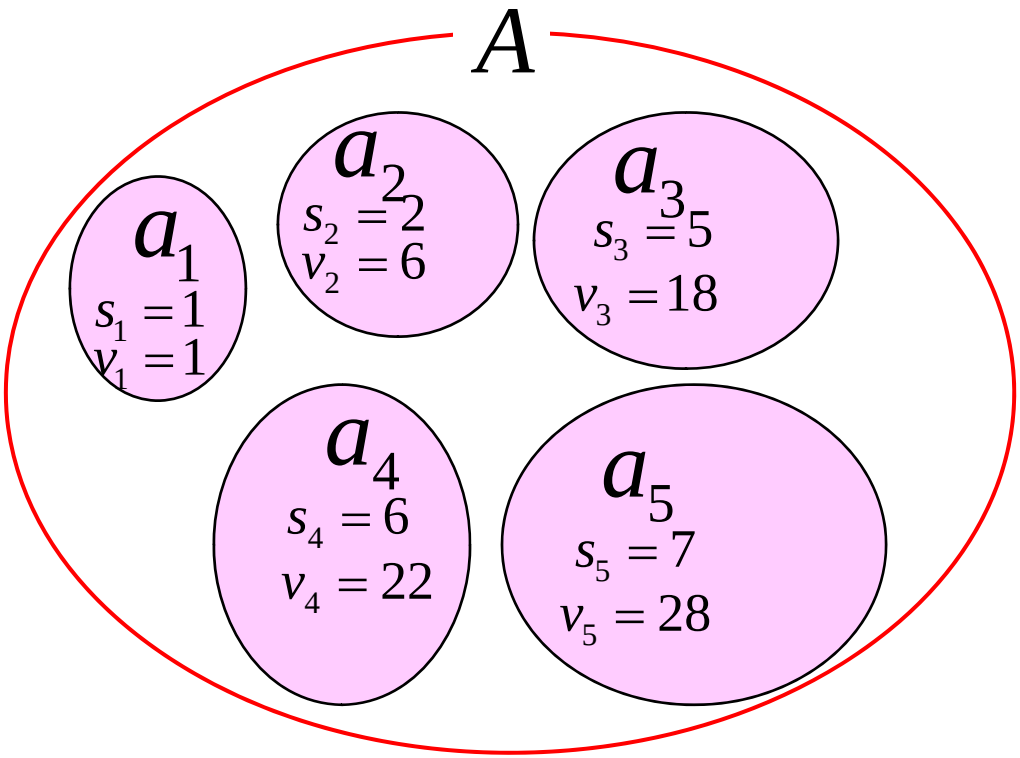
できるだけ、価値を大きくしたい。

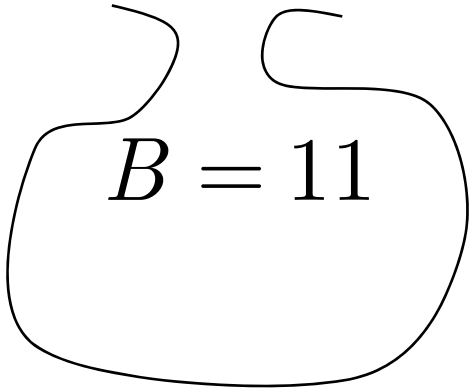
KNAPSACKを解く擬多項式時間アルゴリズム

SUBSET　SUMの擬多項式時間アルゴリズムを少し変更することによって、KNAPSACKを解く擬多項式時間アルゴリズムが得られる。（また、動的計画法に基づくアルゴリズムである。）

まず、次のインスタンスに対して、アルゴリズムの動作をみていく。

インスタンス例





$K = 40$

$A = \{a_1, a_2, a_3, a_4, a_5\}$

$S = (1, 2, 5, 6, 7)$

$V = (1, 6, 18, 22, 28)$

$B = 11$

$K = 40$

方針

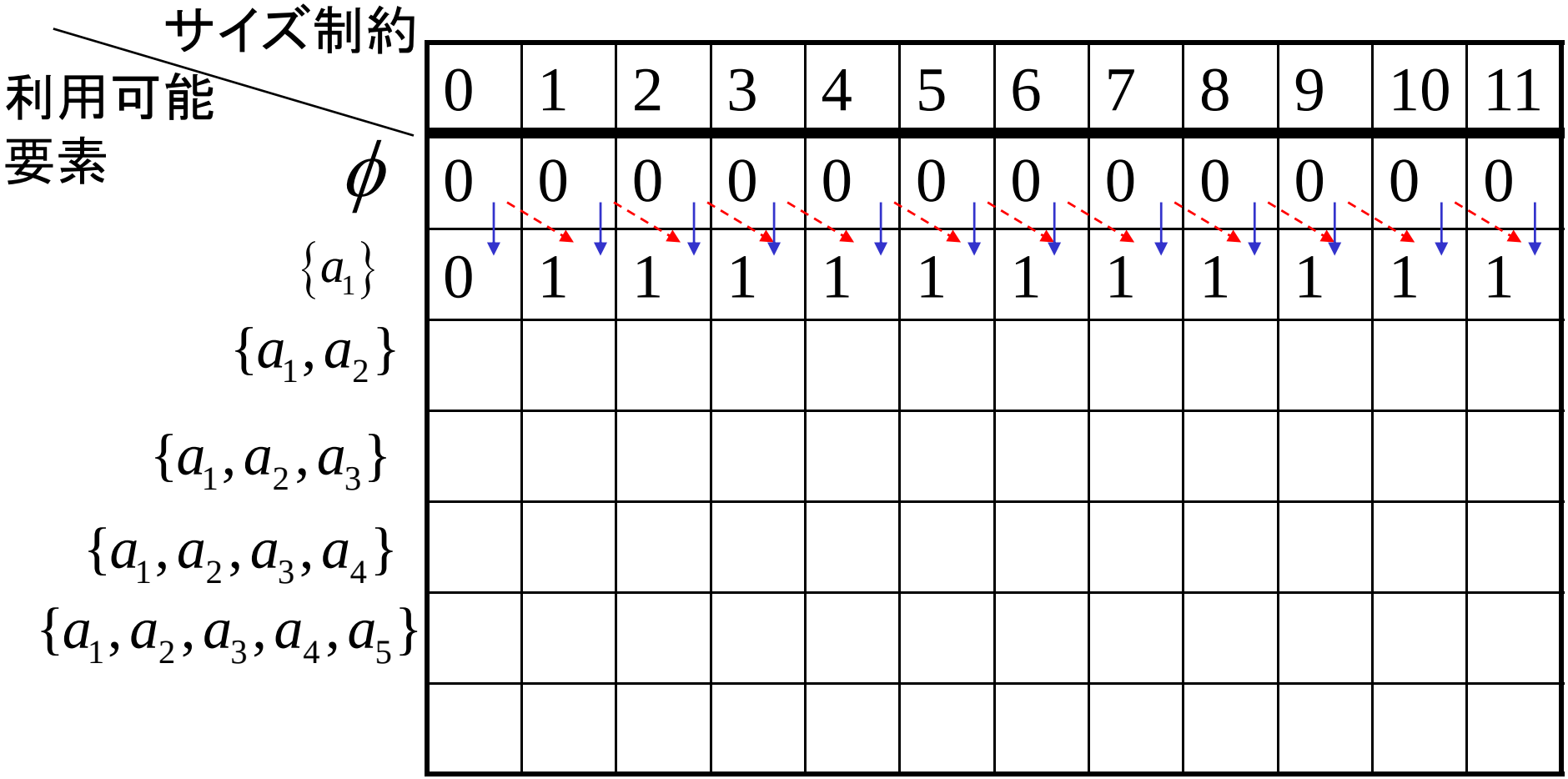
〇部分問題としては、添え字の小さい方だけの集合に対して、
　ナップザックで実現できる最大の価値を求める。
〇現段階で実現可能な最大値をすべて表として保存する。
〇これらの表を、ϕ からはじめて、$\{a_1\}, \{a_1, a_2\}$
　と一づつ要素を増やしながら、ボトムアップで大きくしていく。

表の概形
（アルゴリズムの基礎にあたる。）

利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0											
$\{a_1, a_2\}$	0											
$\{a_1, a_2, a_3\}$	0											
$\{a_1, a_2, a_3, a_4\}$	0											
$\{a_1, a_2, a_3, a_4, a_5\}$	0											

セル内の数値：最大の価値

要素　a_1 に注目した更新　$s_1 = 1$　$v_1 = 1$



セル内の数値：最大の価値

要素　a_2 に注目した更新 $s_2 = 2$　$v_2 = 6$

利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{a_1, a_2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{a_1, a_2, a_3\}$												
$\{a_1, a_2, a_3, a_4\}$												
$\{a_1, a_2, a_3, a_4, a_5\}$												

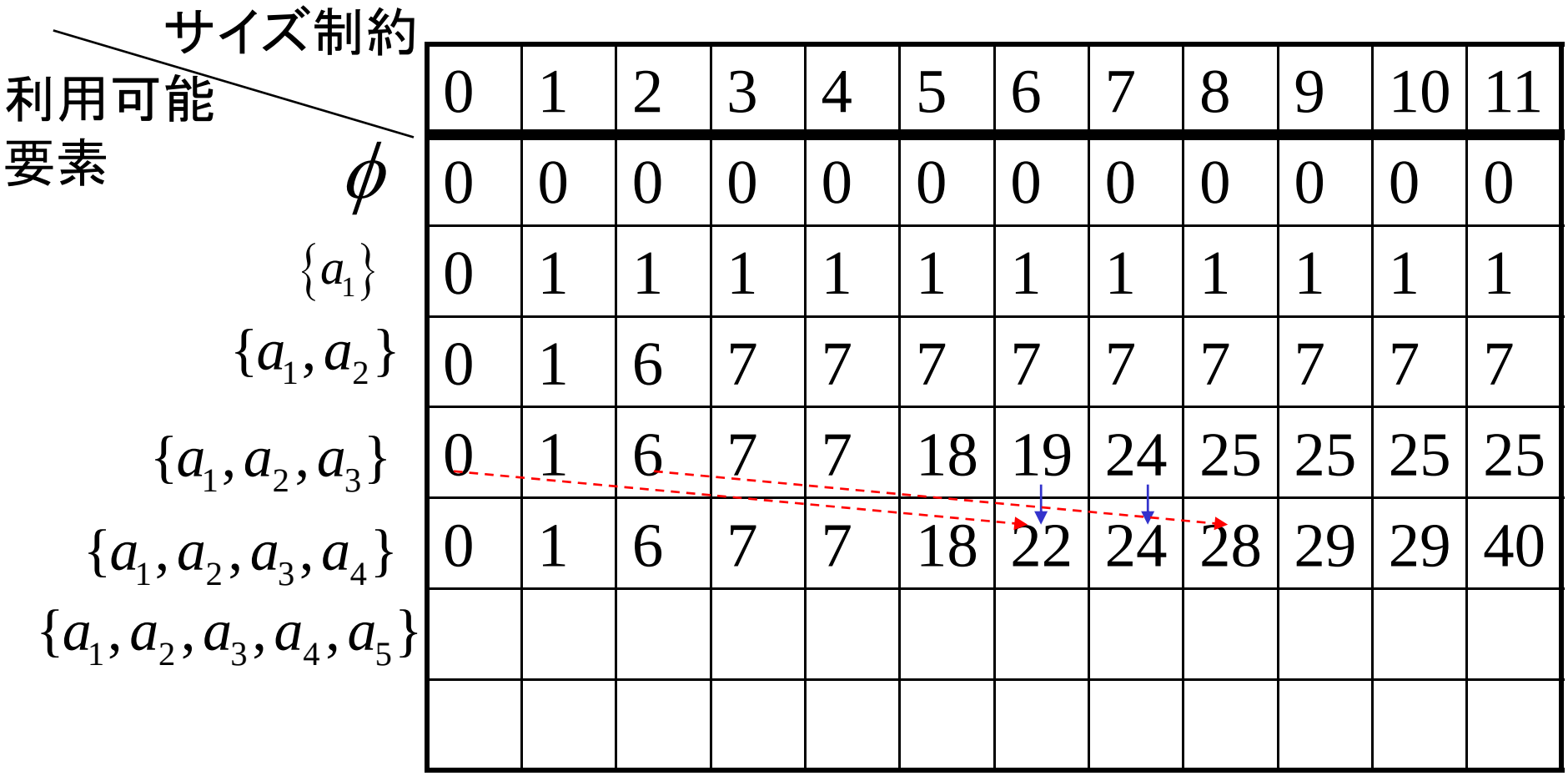
セル内の数値：最大の価値

要素　a_3 に注目した更新 $s_3 = 5$ $v_3 = 18$

利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{a_1, a_2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{a_1, a_2, a_3\}$	0	1	6	7	7	18	19	24	25	25	25	25
$\{a_1, a_2, a_3, a_4\}$												
$\{a_1, a_2, a_3, a_4, a_5\}$												

セル内の数値：最大の価値

要素　a_4 に注目した更新 $s_4 = 6$ $v_4 = 22$



利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{a_1, a_2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{a_1, a_2, a_3\}$	0	1	6	7	7	18	19	24	25	25	25	25
$\{a_1, a_2, a_3, a_4\}$	0	1	6	7	7	18	22	24	28	29	29	40
$\{a_1, a_2, a_3, a_4, a_5\}$												

セル内の数値：最大の価値

要素　a_5 に注目した更新 $s_5 = 7$ $v_5 = 28$

利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{a_1, a_2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{a_1, a_2, a_3\}$	0	1	6	7	7	18	19	24	25	25	25	25
$\{a_1, a_2, a_3, a_4\}$	0	1	6	7	7	18	22	24	28	29	29	40
$\{a_1, a_2, a_3, a_4, a_5\}$	0	1	6	7	7	18	22	28	29	34	35	40

セル内の数値：最大の価値

要素の決定。 $s_5 = 7 \ v_5 = 28$

利用可能要素 ＼ サイズ制約	0	1	2	3	4	5	6	7	8	9	10	11
ϕ	0	0	0	0	0	0	0	0	0	0	0	0
$\{a_1\}$	0	1	1	1	1	1	1	1	1	1	1	1
$\{a_1, a_2\}$	0	1	6	7	7	7	7	7	7	7	7	7
$\{a_1, a_2, a_3\}$	0	1	6	7	7	18	19	25	25	25	25	25
$\{a_1, a_2, a_3, a_4\}$	0	1	6	7	7	18	22	25	28	29	29	40
$\{a_1, a_2, a_3, a_4, a_5\}$	0	1	6	7	7	18	22	28	29	34	35	40

最大値を設定している要素を逆にたどることにより、
容易に求められる。

$\{a_3, a_4\}$

練習

次のKNAPSACKのインスタンスに対して、
肯定か否定かを決定せよ。

（1）

$$A = \{a_1, a_2, a_3, a_4, a_5\}$$

$$S = (1, 2, 5, 6, 7)$$

$$V = (1, 6, 18, 22, 28)$$

$$B = 12$$

$$K = 50$$

KNAPSACKを解くアルゴリズムの計算量を解析する。

なお、ここではRAMモデルに基づいて解析する。

まず、i 行目、サイズs のセルに対して、そのセルに割り当てる整数値を意味する変数を t_{is} で表す。なお、t_{0s} で空集合に対する表のセルに対する変数を表すものとする。

このとき、セルへの割り当ては次のように決定される。

$i = 0$ のとき

$$t_{0s} = 0$$

$s = 0$ のとき

$$t_{i0} = 0$$

$s < 0$ のとき

$$t_{is} = -\infty$$

便宜上、
負の無限大とする。

$i > 0 \quad AND \quad s > 0$ のとき

$$t_{is} = \max\{t_{(i-1)s}, (t_{(i-1)(s-s_i)} + v_i)\}$$

RAMモデルでは、各セルの計算は明らかに定数時間で行える。

したがって、このアルゴリズムは、セル数に比例する。つまり、
$O(Bn)$ 時間のアルゴリズムである。

なお、KNAPSACKの入力サイズは、

$$O(\log n + \sum_{i=1}^{n} \log s_i + \sum_{i=1}^{n} \log v_i + \log B + \log K)$$

であるので多項式時間ではなく、
擬多項式時間アルゴリズムである。

NP完全問題に対するこのような擬多項式時間アルゴリズムは、インスタンスによっては十分実用に耐えられることもある。SUBSET SUMやKNAPSACKにおいて、部分集合の組合せを全て列挙していないことが重要である。集合要素が多くてバケットのサイズが小さいときなどは、十分高速に動作する。インスタンスの性質を見極めて、擬多項式時間アルゴリズムを利用すると効果的である。

１１－３．Pの問題に対するDP法

　　ここでは、動的計画法がNP完全問題に対してだけでなく、P中の問題に対しても有効であることを見ていく。
具体的には、行列積演算の最適化問題を考える。この問題自体はクラスPに属するが、力ずくのアルゴリズムでは指数時間アルゴリズムになってしまう。
（この問題のように、クラスPの中にも巧妙な技法を用いなければ、指数時間アルゴリズムが必要となってしまう問題もある。）

行列積の演算最適化

名称：行列積最適化（MATRIX-CHAIN)
インスタンス：行列のサイズの列

$$P = (p_0, p_1, p_2, \cdots, p_n)$$

演算目標値　K

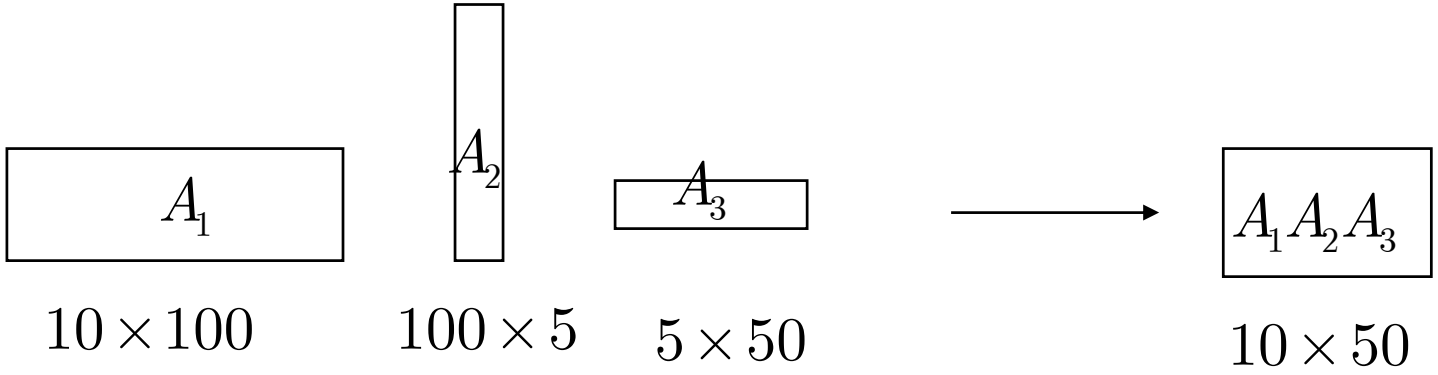
問い：$1 \leq i \leq n$ に対して$p_{i-1} \times p_i$行列を A_i と表す。このとき、$A_1 \bullet A_2 \bullet \cdots \bullet A_n$ を計算する総演算回数が K 以下となるような演算順序が存在するか？

最小化の問題とみなせる。

演算順序によって、演算回数が異なることに注意する。ただし、演算結果は等しい。

演算順序による演算数の相違

ここでは、演算の順序によって、演算数が異なることをみていく。



演算回数

$((A_1A_2)A_3)$ の場合

$$10\times100\times5+10\times5\times50=7500$$

$(A_1(A_2A_3))$ の場合

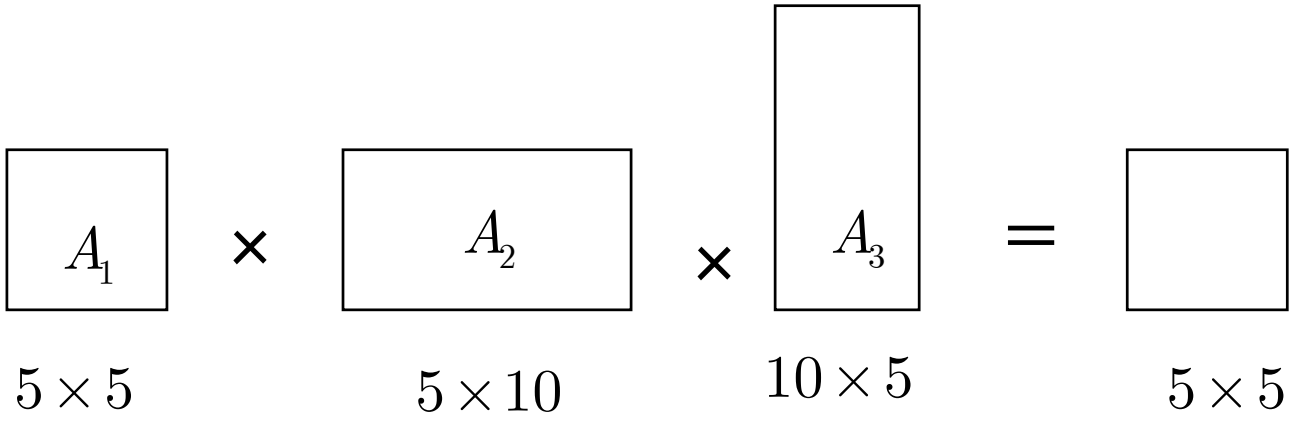
$$100\times5\times50+10\times100\times50=75000$$

10倍の相違

練習

次のインスタンスが、MATRIX－CHAINの肯定のインスタンスか否定のインスタンスかを決定せよ。

$$P = (5, 5, 10, 5) \qquad K = 400$$



このように、行列の積の順序によっては、演算回数が劇的に異なることがある。したがって、行列演算における最適な演算順序を求めることは、意味のある問題である。

しかし、この問題は単純な解法では、多項式時間にさえならない。まず、単純なアルゴリズムから調べていく。

アルゴリズムEXP__MATRIX__CHAIN

1. $A_1 A_2 \cdots A_n$ に対して、全ての括弧の付けを行う。
2. 1. の各括弧付けの中で、演算数が最小のものを出力する。
   （決定問題では、演算回数とKを比較して、YES、NOを判定する。）

このアルゴリズムの計算量を解析しよう。

n 個の行列の系列に対する括弧のつけ方が、$P(n)$ 通りあるとする。このとき、任意の $k = 1, 2, \cdots, n-1$ に対して、 n 個の行列を k 番目の $k+1$ 番目に分割してそれぞれの部分列に対して独立して括弧をつけることができる。したがって、$P(n)$ は次の漸化式を満たす。

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

このような漸化式の解は、組み合わせ論によって、カタラン数 $C(n)$ で表せることがしられている。

カタラン数は、

$$C(n) = \frac{1}{n+1}\binom{2n}{n} = \frac{1}{n+1} \bullet \frac{(2n)!}{n!n!} = \Omega(2^n)$$

と計算できるが、　n に関する多項式ではない。

このC(n)を用いて、括弧のつけ方の総数 $P(n)$ は、

$$P(n) = C(n-1)$$

と表される。

以上より、括弧のつけ方をしらみつぶしで調べる力づくのアルゴリズムでは、多項式時間で最適な行列演算順序を求めることができない。

問題の考察(部分問題の同定)

$i \leq j$ に対して、積の一部を次のように表記する。

$$A_{ij} \equiv A_i \cdots A_j$$

このとき、A_{1n} が求めたい積である。

一番最後の演算が k 番目の演算であるとすると、

$$A_{1n} = A_{1k} A_{(k+1)n}$$

と表せる。

ここで、次の事実に注目する。

A_{1n}を求めるために、最適な括弧のつけ方は、
A_{1k} および $A_{(k+1)n}$ を求めるための最適な括弧のつけ方でもある。

このように最適解が、その部分問題に対しても最適性を示すこと
が動的計画法を可能にしている。

漸化式

ここで、 A_{ij} を求めるための最適な演算数が満たすべき漸化式を示す。 A_{ij} を求めるための最適な演算数を $m[i,j]$ と表す。このとき、次の式を満たす。

積の演算を行わないので、演算数は0

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left\{ m[i,k] + m[k,j] + p_{i-1} p_k p_j \right\} & i < j \end{cases}$$

分割の可能性のなかで、最小値を求める。

表

これらの考察をもとに次のような表を構成できる。
$l \equiv j - i$ とする。

$l = 0$	$m[1,1]$	$m[2,2]$	$m[3,3]$	$m[4,4]$	$m[5,5]$
$l = 1$	$m[1,2]$	$m[2,3]$	$m[3,4]$	$m[4,5]$	
$l = 2$	$m[1,3]$	$m[2,4]$	$m[3,5]$		
$l = 3$	$m[1,4]$	$m[2,5]$			
$l = 4$	$m[1,5]$				

例えば、$m[1,4]$ を計算するには、

$$m[1,4] = \min \begin{cases} m[1,1] + m[2,4] + p_0 p_1 p_4 \\ m[1,2] + m[3,4] + p_0 p_2 p_4 \\ m[1,3] + m[4,4] + p_0 p_3 p_4 \end{cases}$$

の3式を計算する必要があるが、右辺の全ては
既に計算済みであることに注意する。

インスタンス例

次のような行列のサイズを考える。

行列	サイズ
A_1	30×35
A_2	35×15
A_3	15×5
A_4	5×10
A_5	10×20
A_6	20×25

このとき表を次のように構成できる。

l	i = 1	2	3	4	5	6
$l = 0$	0	0	0	0	0	0
$l = 1$	$30 \times 35 \times 15 = 15750$	$35 \times 15 \times 5 = 2625$	$15 \times 5 \times 10 = 750$	$5 \times 10 \times 20 = 1000$	$10 \times 20 \times 25 = 5000$	

l \ i	1	2	3	4	5	6
$l = 0$	0	0	0	0	0	0
$l = 1$	15750	2625	750	1000	5000	
$l = 2$	7875	4375	2500	3500		

$$m[1,3] = \min \begin{cases} m[1,1] + m[2,3] + p_0 p_1 p_3 = 0 + 2625 + 30 \times 35 \times 5 = 7875 \\ m[1,2] + m[3,3] + p_0 p_2 p_3 = 15750 + 0 + 30 \times 15 \times 5 \end{cases}$$

$$= 7875$$

l \ i	1	2	3	4	5	6
$l = 0$	0	0	0	0	0	0
$l = 1$	15750	2625	750	1000	5000	
$l = 2$	7875	4375	2500	3500		
$l = 3$	9375	7125	5375			

$$m[2,5] = \min \begin{cases} m[2,2] + m[3,5] + p_1 p_2 p_5 = 0 + 2500 + 35 \bullet 15 \bullet 20 = 13000 \\ m[2,3] + m[4,5] + p_1 p_3 p_5 = 2625 + 1000 + 35 \bullet 5 \bullet 20 = 7125 \\ m[2,4] + m[5,5] + p_1 p_4 p_5 = 4374 + 0 + 35 \bullet 10 \bullet 20 = 11375 \end{cases}$$

$$= 7125$$

	i 1	2	3	4	5	6
$l = 0$	0	0	0	0	0	0
$l = 1$	15750	2625	750	1000	5000	
$l = 2$	7875	4375	2500	3500		
$l = 3$	9375	7125	5375			
$l = 4$	11875	10500				
$l = 5$	15125					

このアルゴリズムの計算量を解析する。

まず、表のセルの総数は、$O(n^2)$ である。
また、一つのセルに対して、$O(n)$ 時間の計算で
値を決定することができる。
以上より、

$O(n^3)$ 時間

のアルゴリズムである。

１１－４．再帰アルゴリズムと動的計画法

漸化式に基づく計算法に、再帰アルゴリズムがあった。
しかし、再帰アルゴリズムを用いてしまうと、
計算時間が指数時間必要になってしまう。
このことを確かめるために、再帰アルゴリズムの
計算量を解析する。

基になる漸化式を再掲する。

$$m[i,j] = \begin{cases} 0 & i = j \\ \min\limits_{i \le k < j} \left\{ m[i,k] + m[k,j] + p_{i-1} p_k p_j \right\} & i < j \end{cases}$$

漸化式を基にすると、次のように、直接再帰アルゴリズムが得られる。

アルゴリズムREC-M(i,j)

```
if(i==j)
  {
        return(0);
   }
else
   {
        m[i,j]=∞;
        for(k=I;k<j;k++)
          {
                q＝REC-M(i,k)+REC-M(k,j)+ p_{i-1}p_k p_j;
                if(q<m[i,j])m[i,j]=q;
          }
 }
```

再帰アルゴリズムの計算量

再帰アルゴリズムの計算量を $T(n)$ と書くと、次の漸化式を満たす。

$$\begin{cases} T(1) \geq 1 & n=1 \\ T(n) \geq 1+\sum_{k=1}^{n-1}(T(k)+T(n-k)+1) & n>1 \end{cases}$$

ここで、

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

に注意すると、次式が成り立つことがわかる。

$$T(n) \geq 2\sum_{i=1}^{n-1} T(i) + n$$

ここで、 $T(n) \geq 2^{n-1}$ を帰納法によって示す。

基礎　$n = 1$

$$左辺 = T(1) = 1 \qquad 右辺 = 2^{1-1} = 2^0 = 1$$

よって、成り立つ。

帰納 $n > 1$

$1 \leq k < n$ の全てで成り立つと仮定する。

$$T(n) \geq 2\sum_{i=1}^{n-1} 2^{i-1} + n$$

$$= 2\sum_{i=0}^{n-2} 2^{i} + n$$

$$= 2(2^{n-1} - 1) + n$$

$$= 2^n + n - 2$$

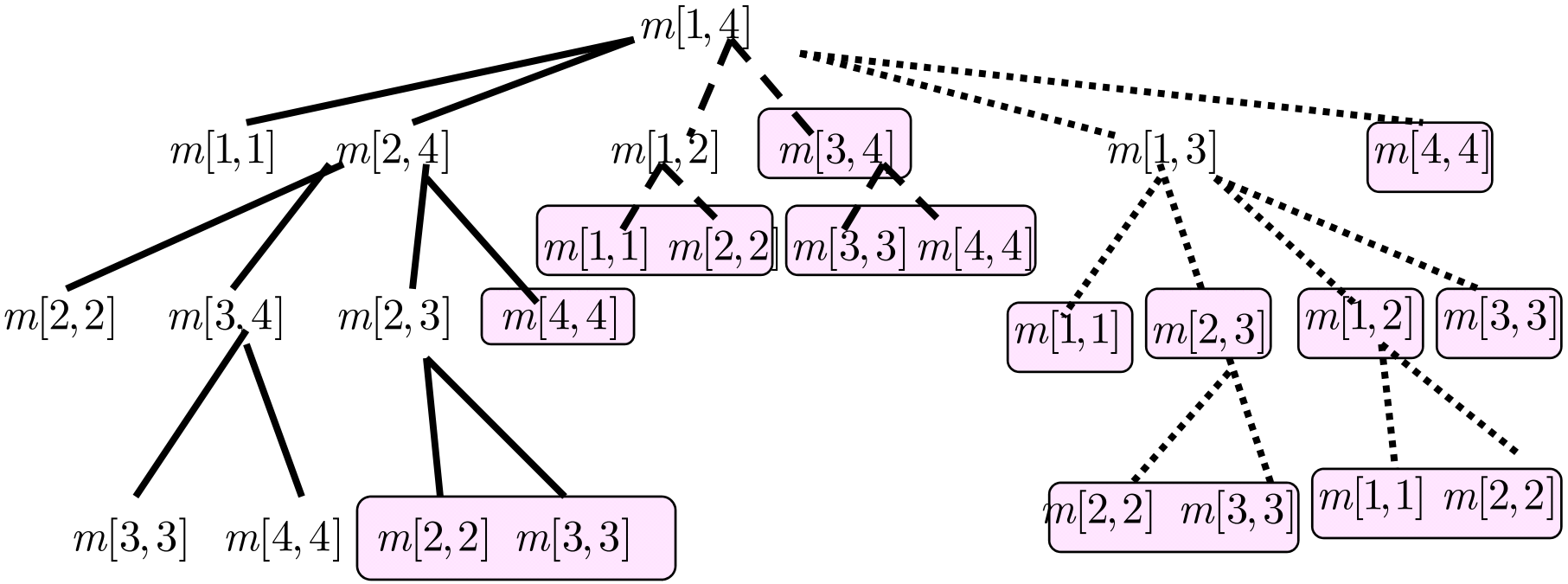
$$\geq 2^{n-1}$$

以上より、REC一M(1,n)は指数時間アルゴリズムであることがわかる。

部分問題の重複性

ここでは、再帰における計算過程を調べることにより、
再帰アルゴリズムとDPの計算時間の差が、どの原理から
導かれているかを調べる。

再帰における計算の様子は次のように木構造で示すことができる。



: 再計算している部分問題

１１－５．動的計画法の一般的性質

一般的に動的計画法を用いた場合に高速化される問題の特徴を挙げる。

- 部分構造の最適性を満たす。

  ある問題の最適解が、その内部の部分問題の最適解を含んでいる。

- 部分問題に重複性がある。

  ある問題に対する再帰アルゴリズムが、同じ問題を繰り返して解くような場合。

問題に対して、これらの性質をみつけることができたならば、動的計画法に基づくアルゴリズムの設計を試みると良い。