第１０回：１２月９日（水）

10　第１０回

10.1　動的計画法 dynamic programming

今回の講義では，前回同様，広汎な離散最適化問題を厳密に解くためのアルゴリズム手法である，**動的計画法**について説明する．

動的計画法の考え方も問題例をいくつかの小さい問題例に分け，これらを解けば元の問題例が解けるという問題例の置き換えに基づく．分枝限定法の分枝操作では与えられた問題例の実行可能領域を分割することで規模の小さな問題例を定義することが多い．一方，動的計画法が使われる場合には，一つの実行可能解そのものを分割して規模の小さな問題例を定義することが多い．そして動的計画法が（分枝限定法と比較して）特に効果を発揮するのは，対象とする問題が「最適性の原理」を有している場合である．

最適性の原理

最短経路問題を例にとって最適性の原理の考え方を説明する．

♠**「講義資料と課題」2 ページ: 最適性の原理**　このスライドでは地図上の A 点から B 点への最短経路を青色の線で表している．

- この最短経路の途中の点 C を適当に選び，青色の最短経路のうち A 点から C 点までの経路を赤色で示してある．
- ここで,「A 点から C 点への最短経路を求めよ」という問題があったとすると，A 点から C 点までの赤色の経路は，そのまま A 点から C 点への最短経路になっている[8]ことが簡単に示せる[9]．

最短経路問題のように，最適解があるとき，その部分構造も（適切に定義された）部分問題の最適解になっているという性質を**最適性の原理**と呼ぶ．

最短経路問題に関連して，最長経路問題に対しては最適性の原理を見出すことが困難である．

- A 点から B 点へ同じ点を二度と通らない経路で最長のものを選んだとする．

[8]最短経路は複数存在し得ることに注意せよ．赤色の経路は，数ある最短経路のうちの一つである．

[9]もし赤色の経路が A 点から C 点までの最短経路でなかったとすると，次のようにして青色の経路よりも短い A 点から B 点への経路を作ることができる: A 点から C 点までの最短経路（赤色の経路ではない）を通り，C 点から B 点は青色の経路をなぞる．これは青色の経路が A 点から B 点までの最短経路であることに矛盾．

- この A 点から B 点への最長経路の途中で D 点を選び，A 点から D 点までの経路を考えると，これは必ずしも「A 点から D 点への同じ点を二度と通らない経路で最長のもの」ではない．

動的計画法を用いたアルゴリズムの設計方針

部分問題を定義したときに最適性の原理が成り立っていると，規模の小さい部分問題を解いて，それらの最適解（最適値）から元の問題の最適解（最適値）を求めることができる．

動的計画法を適用して，アルゴリズムを設計する過程は以下のようになる．

1. 定義 ($\triangleq$)：問題の再帰的構造を見つけ，部分問題を定める．
2. 数学的性質 ($=$)：部分問題の最適値の間に成り立つ関係式（境界条件，再帰式）を導く．
3. 最適値の計算 ($:=$)：関係式に基づき，境界条件（自明に小さい問題）から順に積み上げていくアルゴリズムを設計する（再帰呼び出しを使うアルゴリズムではない）．
4. 最適解の計算 ($:=$)：最適値の計算の過程を記録しておき，これをバックトラックすることで一つの最適解を構築する．

10.2 ナップサック問題に対する動的計画法（サイズ版）

0,1-ナップサック問題の問題例をアイテムのサイズ $a_i,\ i = 1, 2, \ldots, n$，アイテムの利得 c_i, $i = 1, 2, \ldots, n$，ナップサックサイズ b とする．0,1-ナップサック問題でアイテムのサイズ a_i，ナップサックサイズ b が整数値である場合を考えよう．

部分問題の定義

整数 $j = 1, 2, \ldots, n,\ k = 0, 1, \ldots, b$ に対し，容量 k のナップサックに入るように $\{1, 2, \ldots, j\}$ からアイテム集合 S を選ぶときの利得和の最大値を $f^*(j, k)$ と定義する．式で表現すれば以下の通り．

$$f^*(j,k) \triangleq \max\Big\{\sum_{i\in S} c_i \mid S \subseteq \{1,2,\ldots,j\}, \sum_{i\in S} a_i \leq k\Big\}.$$

元の問題の最適値は $f^*(n, b)$ で与えらえる．

再帰式の導出

部分問題で最も規模の小さい問題例はすぐに答えが分かるので，**境界条件**という．この場合，最も規模の小さい問題例は $f^*(1, k), k = 0, 1, \ldots, b$ である．

境界条件：$k = 0, 1, \ldots, b$,

$$f^*(1,k) = \begin{cases} c_1, & b \geq a_1, \\ 0, & b < a_1. \end{cases}$$

$f^*(j,k), j \geq 2$ に対しては直接のその値を知ることはできないので，規模の小さい問題例の最適値を加工して表現することを考える．$f^*(j,k)$ の最適解 S はアイテム $1, 2, \ldots, j$ の中から選択するが，その最適解 S にアイテム j が使われていない場合と使われている場合に分けて考える．

- 前者の場合は，最適解 S はそのままアイテム $1, 2, \ldots, j-1$ の中からサイズ和が k 以下の制約下で選ぶ最適解となっている（最適性の原理）．よって，$f^*(j,k) = f^*(j-1,k)$ が成り立つ．

- 後者の場合は，$S' = S \setminus \{j\}$ もやはりアイテム $1, 2, \ldots, j-1$ の中からサイズ和が $k - a_j$ 以下の制約下で選ぶ最適解となっている．この場合は，c_j の利得を忘れずに足して，$f^*(j,k) = f^*(j-1, k-a_j) + c_j$ が成り立つ（最適性の原理）．

ただし，$k < a_j$ の場合は後者の選択はあり得ない．

以上から以下のように再帰式を得ることができる．

再帰式：$j = 2, 3, \ldots, n$,

$$f^*(j,k) = \begin{cases} \max\{f^*(j-1,k),\ f^*(j-1,k-a_j) + c_j\}, & k \geq a_j \\ f^*(j-1,k), & k < a_j. \end{cases}$$

アルゴリズムの設計

再帰式が得られたので再帰アルゴリズムと考える人もいるであろうが，この場合は，同じ部分問題が計算中に何度も現れるので計算効率が良くない．ここでは，地味に，境界条件の問題から解き始め，得られた最適値を保存し，順に大きな部分問題の最適値を再帰式に従って決定していく． 元の問題の最適値が得られたら，これを与えている部分問題をバックトラックして探索し，最適解を決定する．

次の例題 $(n = 2)$ を用いて説明する．

$$\begin{aligned} c_1 = 4, &\quad c_2 = 6, \\ a_1 = 2, &\quad a_2 = 4, \\ b = 7. & \end{aligned}$$

部分問題 $f^*(j,k)$ は 2 個のパラメータを持つので for ループを二重に持つ反復アルゴリズムの形で書くことができるが，計算の進行は，j, k を成分とする行列（表）のセルを埋めていく様子を見ると理解しやすい．

「講義資料と課題」スライドの 3 ページ以降を使って説明する．

- スライド 3 は境界条件に従って表の一段目を埋めた様子である．

- スライド 4 で示すように，再帰式の第一項は，上の段の値のコピーを取ることに相当する．
- 一方，再帰式の第二項は，上の段の値を $a_2 = 4$ だけ右へシフトさせたのち，$c_2 = 6$ を上乗せすることに相当する．スライド 5 参照．
- そして，第一項と第二項のうち大きな利得値を選ぶ．スライド 6 参照．この問題例の最適値 $f^*(2, b) = 10$ が得られる．

次に，最適解を求める (スライド 7)．

- これには表を埋めていく際に，再帰式において第一項と第二項のうちどちらが最大値を決めていたかを記録しておく．
- すると，最後に得られた最適値 $f^*(2, b) = 10$ から順に第一項と第二項の二択のうち最大のほうへさかのぼっていくことで，利得和が $f^*(2, b) = 10$ を達成しているアイテムの集合が得られる．
- すなわち，部分問題 $f^*(j, k)$ において第二項が最大だった場合は，アイテム j を使い，第一項が最大だった場合はアイテム j を使わない．
- この問題例の場合は，2 番目のアイテムを使うことが分かり，次に 1 番目のアイテムを使うことが判明する．

最適解は $z = (z_1 = 1, z_2 = 1)$ となる．

計算量の上界の解析

部分問題の個数は nb であり，これに比例する領域があれば計算が実行できるので，領域計算量は $O(nb)$ である．部分問題をひとつ解くためには，$f^*(j-1, k)$ や $f^*(j-1, k-a_j)+c_j$ を計算する必要があるが，いずれも 2 次元配列にデータを格納しておけば $O(1)$ 時間で計算できる．部分問題ひとつを $O(1)$ 時間で解けるので，部分問題の個数をかけて，$O(nb)$ がアルゴリズム全体の時間計算量の上界となる．この上界は正確である．

ところで，この計算量 $O(nb)$ は入力サイズの多項式ではない．整数値 b を入力するためには $\log b$ ビットあれば十分であるので，入力サイズ $\log b$ から見ると b はその指数関数になる．しかし，$O(nb)$ はデータの数値 b が小さければ十分に効率が良いので，典型的な指数時間 $O(2^n), O(n!)$ とは区別して，**疑多項式時間**と呼ばれる．

【課題１０－１】スライドで実演した計算例を参考にして，第 8 回で紹介した 0,1-ナップサック問題の例題 (3) に対して動的計画法アルゴリズム（サイズ版）を適用したときの計算の過程を表

にせよ．最適値だけでなくバックトラックにより最適解を求める過程も説明をつけること．問題例 (3) は以下の通り．

$$
\begin{array}{llll}
c_1 = 4, & c_2 = 6, & c_3 = 13, & c_4 = 15, \\
a_1 = 2, & a_2 = 4, & a_3 = 6, & a_4 = 7, \\
b = 11. & & &
\end{array}
$$

第１０回の講義は以上である．