アルゴリズムとデータ構造 プログラミング演習

森 立平 mori@c.titech.ac.jp

2016 年 7 月 12 日

今日の目標

- 動的計画法（「宝詰め込み問題」，最適解の探索も含む）のプログラムを書けるようになる．
- 「レポート」に必要なプログラムを書く．

今日の課題はなし

今日のワークフロー

1. 「レポート」に必要なプログラムを書く．
2. 余裕があれば「発展的な課題」をやる．
3. さらに余裕があれば「超発展的な課題」をやる．

1 宝詰め込み問題のプログラムの注意点

実行時エラー `Bus error`, `Segmentation fault` は配列のインデックスが配列のサイズ以上になっている場合に起き得る．また `Segmantation fault` は再帰呼び出しが無限に続いている場合にも起きる．

2 課題の提出期限と提出方法

元々のレポートと「発展的課題」については既に告知通りの提出期限、提出方法である。プログラムは OCW を通じて提出すること。「超発展的課題」は先週と変更したので提出期限を 7 月 19 日の演習の時間の始めとする。プログラムは OCW を通じて提出すること。

3 発展的な課題

A. 分割数の $O(n)$ 空間計算量のプログラムを参考に「宝詰め込み問題」の $O(W)$ 空間計算量のプログラムを書け．ただし，最適解の計算は必要なく最適値の計算だけでよい．プログラム名は `treasure_spaceW.c` とすること．

B. おそらく全員が $O(nW)$ 時間計算量のプログラムを書いていることと思う．補助関数の定義を変えることで $O(nP)$ 時間計算量のアルゴリズムが作れる．ここで P は「宝詰め込み問題」の最適値である．

   B.1 価値と重さの役割を逆にした補助関数を定義し，その補助関数の満たす再帰的な関係式を導出せよ．

   B.2 その新しい補助関数を用いてトップダウン法もしくはボトムアップ法で時間計算量が $O(nP)$ の動的計画法のプログラムを書け．プログラム名は `treasure_timeNP.c` とすること．

4 超発展的な課題: 実用的な高速アルゴリズム

4.1 アルゴリズムの概要

最悪時間計算量は $O(nW)$ だが実際には多くの場合効率的に動くアルゴリズムを紹介する（理論的な証明はない）。以下では最適値の計算だけを目標とする（最適解の計算もできるのだが C 言語での実装は面倒くさい）。

4.2 アルゴリズムの概要

「宝詰め込み問題」の $O(nW)$ 時間アルゴリズムの改善を考えよう。動的計画法の二次元配列はの各行で「値が変わる場所」だけ覚えておけばよい。この「値が変わる場所」というのは実際にある宝の詰め込み方をしたときの重さの合計、価値の合計に対応している。以下ではこの重さの合計 c と価値の合計 v のペア (c, v) の集合を保存して更新することを考えよう。以降ペア (c, v) のことを「**状態**」と呼ぶことにする。二つの状態 (c, v) と (c', v') について $c \leq c'$ と $v \geq v'$ が満たされるとき、「状態 (c, v) が状態 (c', v') を**支配する**」と言う。状態の集合 U_i を「$i-1$ 番目までの宝を使った詰め込み方に対応する状態の集合」と定義する（宝は 0 番目から数えることにする）。ここで U_i は高々 2^i 個の要素を持つ。また、状態の集合 L_i を「U_i の要素で U_i の他の要素に支配されていないものからなる集合」と定義する。**この集合 L_i の各要素が二次元配列の各行における「値が変わる場所」に対応している**。「宝詰め込み問題」の最適値は L_n を用いて $\max\{v_j \mid (c_j, v_j) \in L_n, c_j \leq W\}$ と表わされる。

以下ではこの集合 L_i を再帰的に計算することを考える。前提として状態の集合 L_i は順序付けされていて、L_i の j 番目の要素を (c_j, v_j) とすると、

$$c_1 < c_2 < c_3 < \dots$$
$$v_1 < v_2 < v_3 < \dots$$

が成り立っているものとする。このことは、L_i の要素達がお互いを支配していないことから一般性を失うことなく仮定できる。まず $L_0 = \big((0,0)\big)$ である。次に L_{i-1} を使って L_i を計算する方法を考える。$L'_{i-1} := \{(c + \texttt{C[i-1]}, v + \texttt{V[i-1]}) \mid (c, v) \in L_{i-1}\}$ とする。そして L_{i-1} と L'_{i-1} の和集合から支配されている要素を取り除くことで L_i が得られる。この L_{i-1}（と L'_{i-1}）から L_i を計算する操作は次のように実現できる。L_{i-1} は上述のように整列しているとする。L_{i-1} の各要素に $i-1$ 番目の重み、価値を足しながらコピーすることで L'_{i-1} を整列した状態で得ることができる。最後に L_{i-1} と L'_{i-1} を「マージ」して L_i を計算する。このときに得られた L_i は**整列しており、他の要素に支配されている要素を含まない**必要がある。この「マージ」操作がこのアルゴリズムにおいて重要なところである。

4.3 C 言語による実装

状態は構造体を用いて表現することにする。

```
struct state_t {
  int c;
  int v;
};
```

プログラムの雛形を与える。

```
#include <stdio.h>

#define N 20000
#define WMAX 100000
#define MAXSTATES WMAX

unsigned int C[N];
unsigned int V[N];

/*
```

```
状態の構造体
*/
struct state_t {
  int c;
  int v;
};

/*
  states は状態の整列した集合を表す配列
  states_tmp0 と states_tmp1 は状態の集合の更新のために使う配列
  ns は状態の個数とする
*/
struct state_t states[MAXSTATES];
struct state_t states_tmp0[MAXSTATES];
struct state_t states_tmp1[MAXSTATES];
unsigned int ns;

/*
states_tmp0 と states_tmp1 を「マージ」して states に保存する関数
c が W を超える状態は保存する必要がない
*/
void merge(int W){
}

int main(){
  int i, j;
  unsigned int n, W;

  if(scanf("%u", &n) != 1) return 1;
  if(scanf("%u", &W) != 1) return 1;
  if(n > N){
    puts("Too many items");
    return 1;
  }
  if(W > WMAX){
    puts("Too large bag");
    return 1;
  }
  for(i = 0; i < n; i++){
    if(scanf("%u", C+i) != 1) {
      puts("Error");
      return 1;
    }
  }
  for(i = 0; i < n; i++){
    if(scanf("%u", V+i) != 1) {
      puts("Error");
      return 1;
    }
  }
/*
  この時点で宝の数は n に，袋の限界容量は W に，
  宝の重さは配列 C に，宝の価値は配列 V に
  代入されているとしてよい
*/

/*
  重さ0，価値0 の状態がひとつある
*/
```

```
  states[0].c = 0;
  states[0].v = 0;
  ns = 1;

/*
  ここにプログラムを書く
*/

/*
  最適値と最終的な状態の数の出力（下2行は消さないこと）
*/
  printf("%d\n", states[ns-1]);
  printf("%d\n", ns);

  return 0;
}
```

このプログラムを完成させてほしい。整列した状態の集合 L_i は states に保存するとする。また状態の数は ns で表す（よって states[0] から states[ns-1] に状態が保存されている）。各 i について、states を states_tmp0 にそのままコピーし、states を states_tmp1 に C[i-1] と V[i-1] を足しながらコピーする。そのあと関数 merge で states_tmp0 と states_tmp1 を「マージ」したものを states に保存する。この時に**他の状態に支配されている状態は取り除く**必要がある。また、「マージ」した結果は**整列していなければならない**。

4.4 超発展的な課題

I. この状態を更新していくことで「宝詰め込み問題」の最適値を計算するアルゴリズムについて、プログラム全体の擬似コードを書いて説明せよ（関数 merge の擬似コードは書かなくてよい）。

II. 関数 merge の擬似コードを書いて説明せよ。

III. C 言語のプログラムを書け。プログラム名は treasure_pareto.c とすること.

IV. このアルゴリズムについて、もし高速化の工夫などがあれば書け（プログラムで未実装でもよい）。

V. このアルゴリズムを修正することで最適解の計算もできる。状態に線形リストを用いた「宝の詰め込み方」の表現を含めることにすれば、高々定数倍の実行時間の増加で済む。このプログラムの擬似コードを書け。

5 おまけ

先週出した「超発展的な課題」（取り下げ済み）をトップダウンではなく、状態を用いたアルゴリズムにするととても高速になる。「マージ」した後に「枝刈り」をすればよい。どのような上界を使って「枝刈り」をするべきかは少し考える必要がある（先週の資料にある「カットあり宝詰め込み問題」のアイデアを使うとよい）。興味がある人はやってみるといいかもしれない。

今日（7 月 12 日）発売の「数学セミナー」（理工系の学部生以上が対象の雑誌。生協に売っているかもしれない）に私が書いた記事が掲載されている。テーマは誤り訂正符号であるが、数学的な部分だけを簡単に説明する。実数 $\epsilon \in (0,1)$ に対して 2 つの写像

$$f_0(\epsilon) := 1 - (1-\epsilon)^2, \qquad f_1(\epsilon) := \epsilon^2.$$

を適用すると $\epsilon_0 := f_0(\epsilon)$、$\epsilon_1 := f_1(\epsilon)$ という 2 つの実数が得られる。さらに ϵ_0 と ϵ_1 のそれぞれに 2 つの写像を適用することで $\epsilon_{00} = f_0(\epsilon_0)$、$\epsilon_{10} = f_1(\epsilon_0)$、$\epsilon_{01} = f_0(\epsilon_1)$、$\epsilon_{11} = f_1(\epsilon_1)$ という 4 つの実数が得られる。この操作を n 回くり返すと 2^n 個の実数が得られる。非自明であるが n が大きいと 2^n 個の実数のうちの**ほとんどが 0 か 1 に近い**。そして 1 に近いものの割合は ϵ に収束する。分割統治法を使えば 2^n 個の実数を計算するプログラムが簡単に書ける。