アルゴリズムとデータ構造

第22週目

担当　情報システム部門　徳光政弘
2025年11月25日

今日の内容

- グリーディ法
- 動的計画法

グリーディ法

- 貪欲に目先の最適解を探していく方法
- 大局的な視点にかける(もしかしたら近い条件でさらによい解があるかもしれない場合を見過ごす)

グリーディ法

- 常に最短距離を走ろうとする、だけど少し遠回りしたほうが早く着く場合も

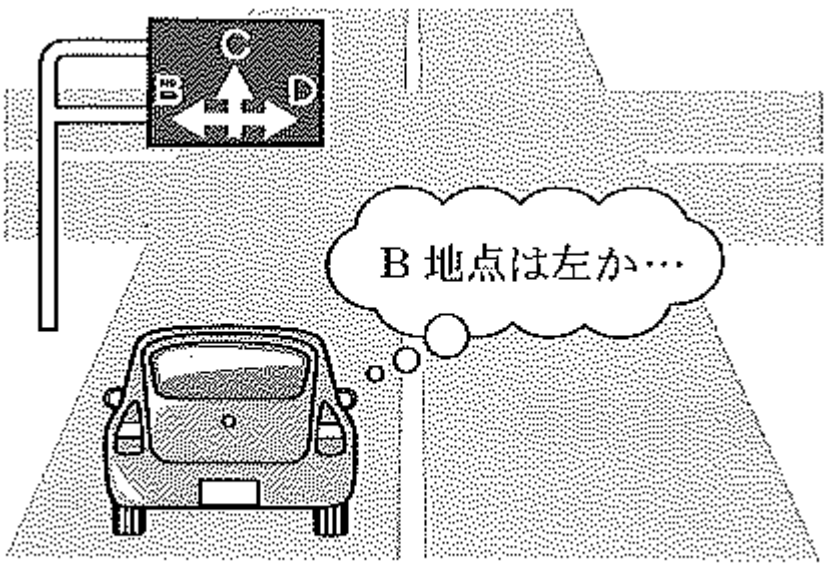


図 8.1　グリーディ法のアイデアによる車の運転

例 コインの両替問題

100 円，50 円，10 円，5 円，1 円の硬貨がそれぞれ十分大量にあるものとする．この硬貨を用いて，整数 x が与えられたときに，x 円となるような枚数最小の硬貨の組合せを求めよ．

この問題の入力として，$x = 138$ が与えられたとすると，この問題の解は，

$$138\text{円} = 100\text{円} \times 1 + 50\text{円} \times 0 + 10\text{円} \times 3 + 5\text{円} \times 1 + 1\text{円} \times 3$$

という 8 枚の硬貨の組合せである．

思いつく考え方

① x 円を超えないように 100 円硬貨でできるだけ支払う.

② 残りの金額に対して，50 円硬貨でできるだけ支払う.

③ 残りの金額に対して，10 円硬貨でできるだけ支払う.

④ 残りの金額に対して，5 円硬貨でできるだけ支払う.

⑤ 残りの金額に対して，1 円硬貨ですべて支払う.

反例　うまくいかない場合

- 50円、40円、1円の硬貨があるとする。
- 120円を表す組み合わせを考える。
- 40円　3枚
- 50円　2枚、1円　20枚

ナップサック問題

[問題 8.2] ナップサック問題

1 から n までの番号のついた n 個の荷物があり，番号が i の荷物の重さと価値がそれぞれ w_i, v_i であるとする．また，荷物を入れるナップサックがあり，このナップサックには重さの和が c までならいくらでも荷物を入れられるものとする．このとき，ナップサックの中の荷物の価値の和が最大になるようなナップサックに入れる荷物の組合せを求めよ．

価値の総和： $\sum_{i=1}^{n} x_i v_i \quad \rightarrow$ 最大

重さの制約条件： $\sum_{i=1}^{n} x_i w_i \leq c$

x_i 重さ
v_i 価値
重さの条件を満たしつつ、
価値の最大化をする

問題設定

表 8.1　ナップサック問題の問題例 1

番号	種類	重さ	価格
1	モカ	2kg	2000 円
2	キリマンジャロ	1kg	3000 円
3	コロンビア	4kg	3000 円
4	ブレンド	5kg	4000 円

ナップサックの容量を5kgとする。

分割ナップサック問題

荷物の重さを分割できる場合

また，ナップサックの容量を 5kg とする．この問題の入力を式で表すと，以下のようになる．

$$w_1 = 2, \quad w_2 = 1, \quad w_3 = 4, \quad w_4 = 5,$$

$$v_1 = 2000, \quad v_2 = 3000, \quad v_3 = 3000, \quad v_4 = 4000, \quad c = 5$$

この問題を分割ナップサック問題とした場合，価値が最大となる解は，

$$x_1 = 1, \quad x_2 = 1, \quad x_3 = 0, \quad x_4 = 0.4$$

である．つまり，モカとキリマンジャロをすべてナップサックに入れて，ブレンドを $\frac{2}{5}$ だけ入れた場合がもっとも価値が高くなり，価値の和は，$2000+3000+0.4\times4000 = 6600$ 円である．

0-1ナップサック問題

荷物の重さを分割できない場合

つぎに，この問題を 0-1 ナップサック問題だと考えた場合の解は，

$$x_1 = 0, \qquad x_2 = 1, \qquad x_3 = 1, \qquad x_4 = 0$$

であり，キリマンジャロとコロンビアをナップサックに入れた場合がもっとも価値が高くなり，価値の和は $3000 + 3000 = 6000$ 円である．

分割ナップサック問題の解法

- グリーディ手法　単位重さあたりの価値v_i / w_i が大きい順にナップサックに入れる

$$\frac{v_1}{w_1} = 1000, \qquad \frac{v_2}{w_2} = 3000, \qquad \frac{v_3}{w_3} = 750, \qquad \frac{v_4}{w_4} = 800$$

分割ナップサック問題の解法

$$\frac{v_1}{w_1} = 1000, \qquad \frac{v_2}{w_2} = 3000, \qquad \frac{v_3}{w_3} = 750, \qquad \frac{v_4}{w_4} = 800$$

表 8.1　ナップサック問題の問題例 1

番号	種類	重さ	価格
1	モカ	2kg	2000 円
2	キリマンジャロ	1kg	3000 円
3	コロンビア	4kg	3000 円
4	ブレンド	5kg	4000 円

アルゴリズム **8.1**

分割ナップサック問題を解くグリーディ法によるアルゴリズム

入力：荷物の重さを格納する配列 W[1], W[2], ..., W[n], 荷物の価値を格納する配列 V[1], V[2], ..., V[n], およびナップサック容量を表す定数 C

```
for (i=1; i<=n; i=i+1) { Z[i]=V[i]/W[i]; }    //単位重さあたりの価値を計算
Z[1],Z[2],...,Z[n]を荷物の番号とともに昇順にソート;
j=1; sum=0;
while (sum<C) {
  Z[j]からZ[j]=V[k]/W[k]を満たすような荷物の番号k を得る;
  if (W[k]<=C-sum) { X[k]=1; sum=sum+W[k]; }
  else { X[k]=(C-sum)/W[k]; sum=C; }
  j=j+1;
}
X[1],X[2],...,X[n]を出力;
```

動的計画法

- 問題を部分問題に分割して、先に求めた部分問題の解を活用して解く方法

車の運転例

過去に通ったことがある経路データを使って、移動する（手法のアイデアを理解してもらう例え話）

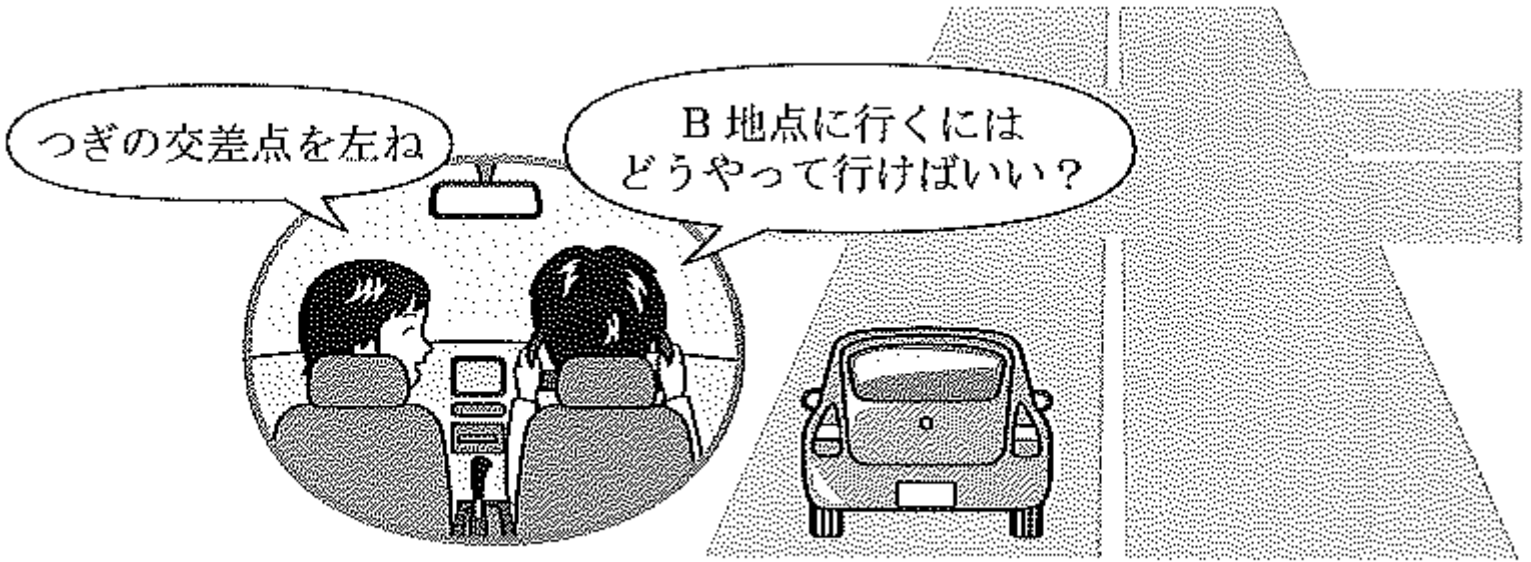


図 8.2　動的計画法のアイデアによる車の運転

べき乗の計算

アルゴリズム **8.2**　べき乗を求める基本的なアルゴリズム

```
p=x;
for (i=2; i<=n; i=i+i) { p=p*x; }
p を出力;
```

$$x^n = \begin{cases} x^{\frac{n}{2}} \times x^{\frac{n}{2}} & (n \text{が偶数の場合}) \\ x \times x^{\frac{n-1}{2}} \times x^{\frac{n-1}{2}} & (n \text{が奇数の場合}) \end{cases}$$

べき乗の計算

関数の再帰を使っているのは漸化式と親和性よく説明するため、実際は効率が悪い

アルゴリズム 8.3　べき乗を求める動的計画法を用いたアルゴリズム

```
power(x,n) {
  if (n==1) return x;
  else {
    if (n が偶数) { p=power(x,n/2); return p*p;}
    else { p=power(x,(n-1)/2); return x*p*p }
  }
}
```

$$x^n = \begin{cases} x^{\frac{n}{2}} \times x^{\frac{n}{2}} & (n\text{が偶数の場合}) \\ x \times x^{\frac{n-1}{2}} \times x^{\frac{n-1}{2}} & (n\text{が奇数の場合}) \end{cases}$$

べき乗の計算

$$T(n) = \begin{cases} \underbrace{T\left(\frac{n}{2}\right)}_{\texttt{power(x,n/2)}\ \text{の計算}} + c & (n\ \text{が}\ 2\ \text{以上の偶数の場合}) \\ \underbrace{T\left(\frac{n-1}{2}\right)}_{\texttt{power(x,(n-1)/2)}\ \text{の計算}} + c & (n\ \text{が}\ 3\ \text{以上の奇数の場合}) \\ c & (n = 1\ \text{の場合}) \end{cases}$$

べき乗の計算

$$T(n) = \begin{cases} \underbrace{T\left(\frac{n}{2}\right)}_{\text{power(x,n/2) の計算}} + c & (n \text{ が } 2 \text{ 以上の偶数の場合}) \\ \underbrace{T\left(\frac{n-1}{2}\right)}_{\text{power(x,(n-1)/2) の計算}} + c & (n \text{ が } 3 \text{ 以上の奇数の場合}) \\ c & (n = 1 \text{ の場合}) \end{cases}$$

$$T(n) \leq T\left(\frac{n}{2}\right) + c = T\left(\frac{n}{4}\right) + c + c = \cdots = \underbrace{c + c + \cdots + c}_{\log_2 n \text{ 個}}$$

$T(n) = O(\log n)$　　これまでの木の再帰と同じ考え方で求まる

0-1ナップサック問題への活用

ある荷物iを入れる　x_i = 1、x_i = 0
荷物の入れ方をn組で考える

(x_1, x_2, x_3, x_4) = (入れる、入れない、入れる、入れない) = (1, 0, 1, 0)

$$(0,0,0,0),\ (0,0,0,1),\ (0,0,1,0),\ (0,0,1,1),\ (0,1,0,0),\ (0,1,0,1),$$

$$(0,1,1,0),\ (0,1,1,1),\ (1,0,0,0),\ (1,0,0,1),\ (1,0,1,0),\ (1,0,1,1),$$

$$(1,1,0,0),\ (1,1,0,1),\ (1,1,1,0),\ (1,1,1,1)$$

0-1ナップサック問題への活用

アルゴリズム 8.4　0-1 ナップサック問題を解く基本的なアルゴリズム

入力：荷物の重さを格納する配列 W[1], W[2], ..., W[n]，荷物の価値を格納する配列
　　V[1], V[2], ..., V[n]，およびナップサック容量を表す定数 C

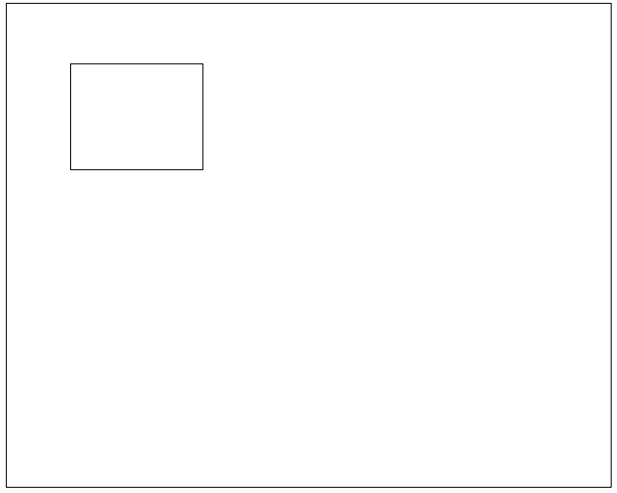
```
2^n個の変数割当を作成;
    //作成されたk番目の変数割当をZ[k][1], Z[k][2], ..., Z[k][n]と表す
vmax=0;
for (i=1; i<=2^n, i=i+1)  {
  wsum=0; vsum=0;
  for (j=1; j<=n; j=j+1)  {      //ナップサック中の重さと価値の和を計算
    wsum=wsum+Z[i][j]*W[j]; vsum=vsum+Z[i][j]*V[j];
```

0-1ナップサック問題への活用

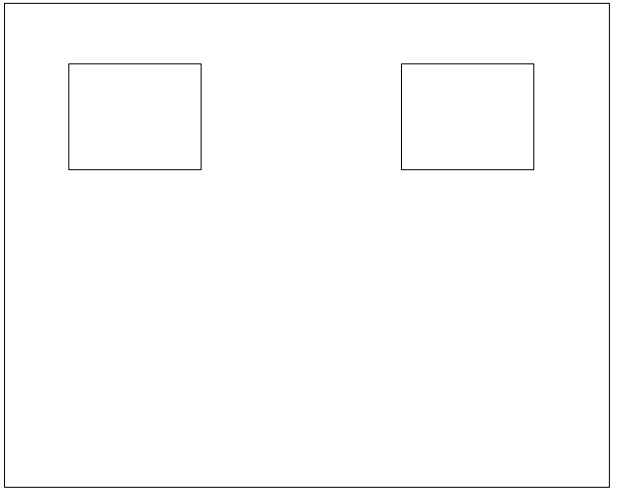
```
    for (j=1; j<=n; j=j+1) {  X[j]=Z[i][j];  } //割り当てを配列Xに記録
  }
}
X[1], X[2], ..., X[n]を出力;
```

単純な方法で求める効率が悪い　$O(n2^n)$

0-1ナップサック問題解法の考え方



荷物1を入れる・入れない



荷物2を入れる・入れない

アプローチ　荷物の重さの「合計」に対する価値を表で整理する

0-1ナップサック問題解法の考え方

重さ

種類

	0	1	2	3	4	5
1	$(0, \phi)$		$(2000, \{1\})$			

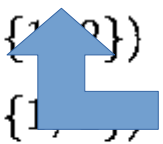
	0	1	2	3	4	5
1	$(0, \phi)$	$(0, \phi)$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$

	0	1	2	3	4	5
1	$(0, \phi)$	$(0, \phi)$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$
2	(V_0, L_0)	(V_1, L_1)	(V_2, L_2)	(V_3, L_3)	(V_4, L_4)	(V_5, L_5)

0-1ナップサック問題解法の考え方

重さ

種類	0	1	2	3	4	5
1	$(0, \phi)$	$(0, \phi)$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$
2	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$
3	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$	$(6000, \{2, 3\})$
4	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000$	$\{1$	$(6000, \{2, 3\})$



前のステップを見ている

- 漸化式をコンピュータで前進的に計算している
- ステップ前の計算結果を使用する。
- 一度、手計算で確認してみるとよい。
- アルゴリズムのfor文と照らし合わせてみる。

表 8.1　ナップサック問題の問題例 1

番号	種類	重さ	価格
1	モカ	2kg	2000 円
2	キリマンジャロ	1kg	3000 円
3	コロンビア	4kg	3000 円
4	ブレンド	5kg	4000 円

0-1ナップサック問題解法の考え方

重さ

種類	0	1	2	3	4	5
1	$(0, \phi)$	$(0, \phi)$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$	$(2000, \{1\})$
2	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$
3	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$	$(6000, \{2, 3\})$
4	$(0, \phi)$	$(3000, \{2\})$	$(3000, \{2\})$	$(5000, \{1, 2\})$	$(5000, \{1, 2\})$	$(6000, \{2, 3\})$

- 種類3に対して計算することを考える。種類2の計算結果である(3000, {2})を適用するときに、価値が6000、{2, 3}で重さ5kgの制約を満たして価値が最大となる。

表 8.1　ナップサック問題の問題例 1

番号	種類	重さ	価格
1	モカ	2kg	2000 円
2	キリマンジャロ	1kg	3000 円
3	コロンビア	4kg	3000 円
4	ブレンド	5kg	4000 円

0-1ナップサック問題解法の考え方

	0	1	2	3	4	5
1	$(0, \phi)$	$(0, \phi)$	(2000, {1})	(2000, {1})	(2000, {1})	(2000, {1})
2	$(0, \phi)$	(3000, {2})	(3000, {2})	(5000, {1, 2})	(5000, {1, 2})	(5000, {1, 2})
3	$(0, \phi)$	(3000, {2})	(3000, {2})	(5000, {1, 2})	(5000, {1, 2})	(6000, {2, 3})
4	$(0, \phi)$	(3000, {2})	(3000, {2})	(5000, {1, 2})	(5000, {1, 2})	(6000, {2, 3})

- 重さ未満でもナップサック問題の解となる

表 8.1　ナップサック問題の問題例 1

番号	種類	重さ	価格
1	モカ	2kg	2000 円
2	キリマンジャロ	1kg	3000 円
3	コロンビア	4kg	3000 円
4	ブレンド	5kg	4000 円

アルゴリズム 8.5

0-1ナップサック問題を解く動的計画法を用いたアルゴリズム

入力：荷物の重さを格納する配列W[1], W[2], ..., W[n], 荷物の価値を格納する配列V[1], V[2], ..., V[n], およびナップサック容量を表す定数C

2次元配列Tを準備し，Tのすべての値を (0, φ) に初期化;
//アルゴリズムの実行のため，表は0行目も準備する

```
for (i=1; i<=n; i=i+1) {
  for (j=1; j<=C; j=j+1) {
    if (j>=w[i])  {
      T[i-1][j-W[i]] に格納されている値を取り出し，その値を(v1,S1)とする;
      T[i-1][j] に格納されている値を取り出し，その値を(v2,S2)とする;
      if (v1+V[i]>v2) { T[i][j]=(v1+V[i],S1にiを追加; }
      else { T[i][j]=(v2,S2) }
    }
  }
}
```

for文のところに注目すると、計算のオーダーが小さくなっていることがわかる(工夫がうまい！)

動的計画法の応用例

- 数学の計算（フィボナッチ数列、アッカーマン関数）
- 最短経路問題
- ナップサック問題（今回）
- 文字列の編集距離計算
- 行列の積の順番決定（物理計算、AIで大事ですね！）
- 音声認識の計算（これは確率の計算）