アルゴリズムとデータ構造

コンピュータサイエンスコース
知能コンピューティングコース

第8回

アルゴリズムの設計
（分割統治法，動的計画法）
連結リストに関する補足説明

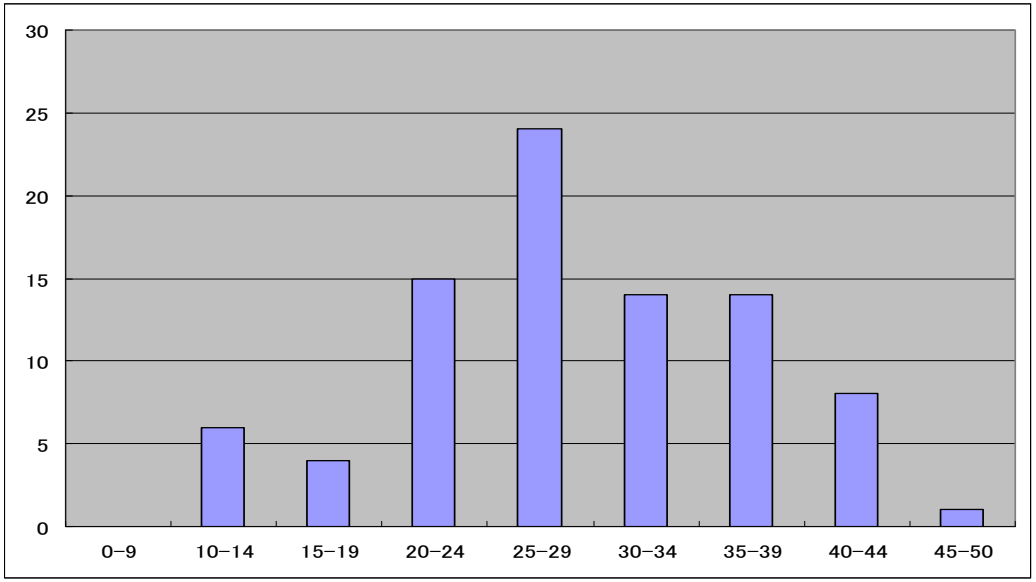
塩浦昭義

情報科学研究科 准教授

shioura@dais.is.tohoku.ac.jp
http://www.dais.is.tohoku.ac.jp/~shioura/teaching

中間試験の結果について

- 平均点 28.8 (配点:10, 10, 10, 12, 8)
- 50点満点, 24点以下は不合格
  - 20点以上24点以下の場合は追試レポートで救済可能
  - 追試レポート: 中間試験問題を全て解いて6／18講義前までに提出. 得点が45点以上ならば合格.
  - 19点以下の場合は応相談

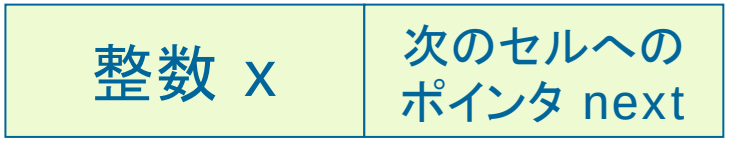


連結リストを用いたプログラム例

プログラムの説明 (list.c)

```
struct llist {
  int x;
  struct llist *next;
};
```

llist はセルを表す構造体




```
struct llist *newcell(int x)
{
  struct llist *pt;

  pt = (struct llist *) malloc(sizeof(struct llist));
  pt->x = x;
  return(pt);
}
```

newcell は新しいセルを作り出す関数
引数 x はセルの要素となる
新しく作ったセルへのポインタを出力

- セルの分の記憶領域を確保
- そのアドレスを pt に代入

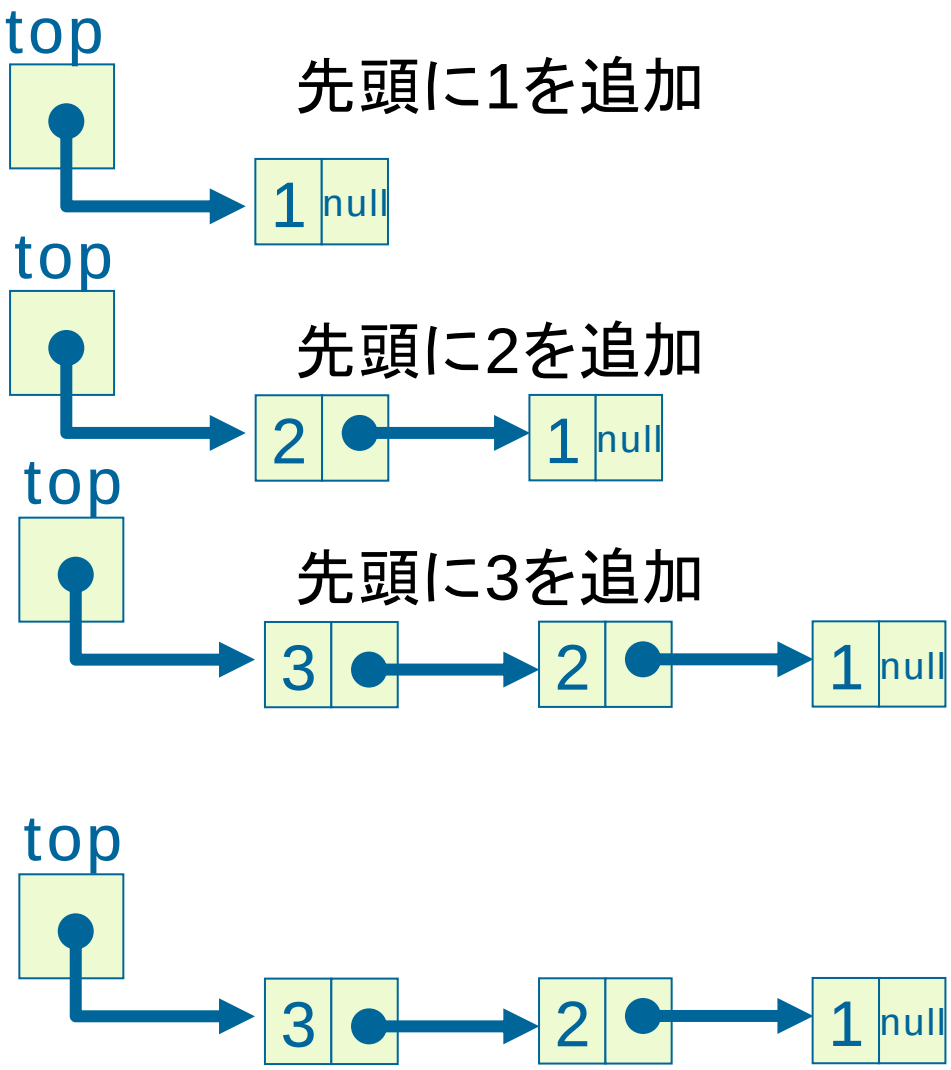
```
main()
{
  struct llist *top, *p;

  p = newcell(1);
  p->next = NULL; top = p;

  p = newcell(3);
  p->next = top; top = p;

  p = newcell(2);
  p->next = top; top = p;

  p = top;
   do {
     printf("%d ", p->x);
     p = p->next;
   } while (p != NULL);
   printf("¥n");
}
```



先頭から順番に，各セルの要素を表示

分割統治法 (divide-and-conquer method)

- アルゴリズム設計法のひとつ
- 以下の手順からなる
  1. 元の問題を小規模な部分問題に分割
  2. 部分問題を再帰的に解く
  3. 部分問題の解を統合することにより，元の問題の解を得る

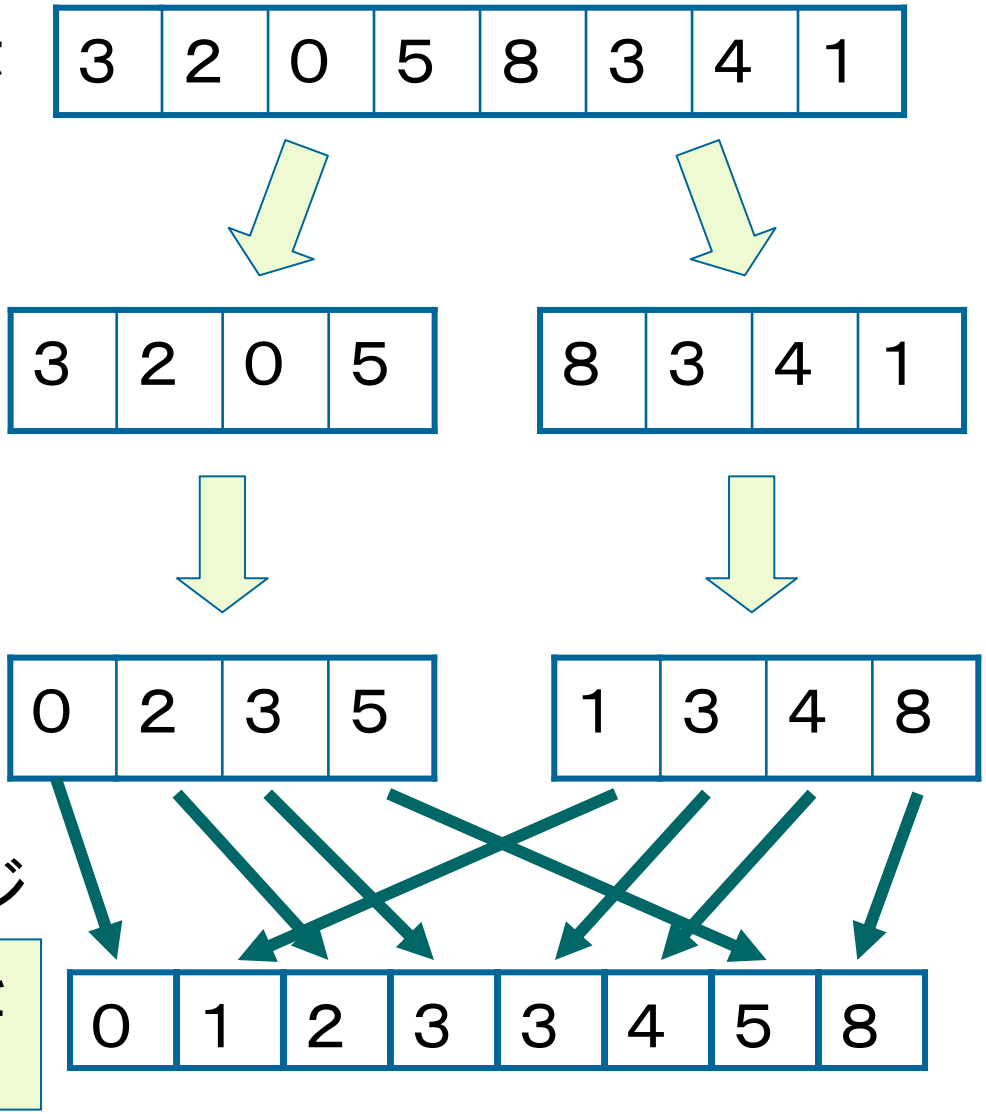
分割

統治

- これまでに出てきた例：マージソート，クイックソート
- 新しい例：長大数のかけ算

マージソートと分割統治法

- マージソートは分割統治法に基づくアルゴリズム



① 与えられた配列を2分割

元の問題を小規模な部分問題に分割

② 2分割された配列をそれぞれ再帰的にソート

部分問題を再帰的に解く

③ ソートされた2つの配列をマージ

部分問題の解を統合することにより，元の問題の解を得る

クイックソートと分割統治法

- クイックソートは分割統治法に基づくアルゴリズム
（と見ることも出来る）

①軸要素を選んで，軸要素未満の要素とそれ以外に分割

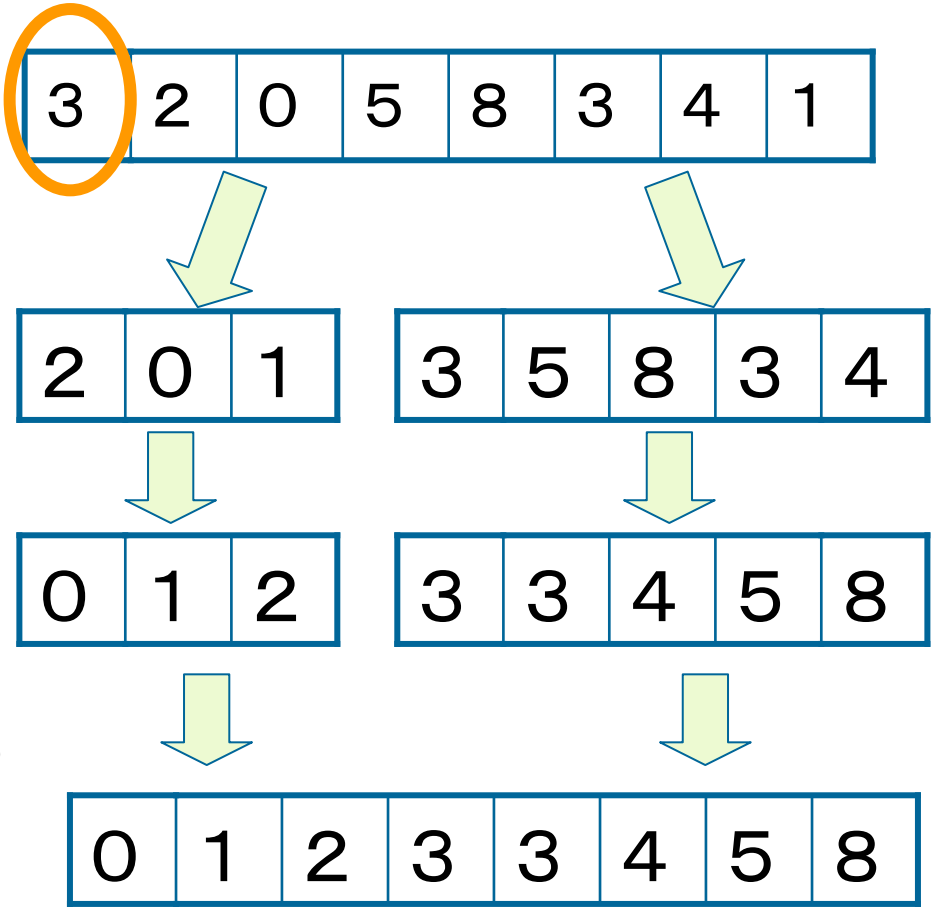
元の問題を小規模な部分問題に分割

②2分割された配列をそれぞれ再帰的にソート

部分問題を再帰的に解く

③ソートされた2つの配列をつなげる

部分問題の解を統合することにより，元の問題の解を得る



長大数のかけ算

- 10進数のn桁の数 $x=x_1x_2\ldots x_n$ と $y=y_1y_2\ldots y_n$ のかけ算を，1桁同士のかけ算を使って計算
- 普通の計算方法：
  - 1桁同士のかけ算がn^2回必要
  - さらに繰り上げの計算，足し算，桁シフトが必要
  - 桁シフト：234 ➔ 234000 のように桁を移動

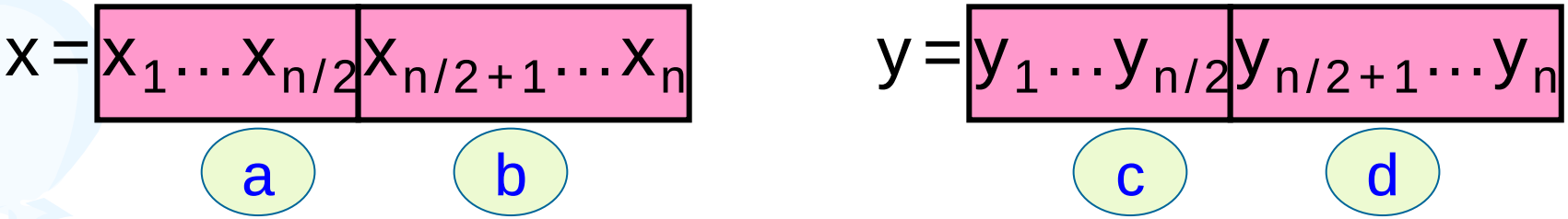
注意：かけ算は他の演算に比べて時間がかかる

- 分割統治に基づく方法：
  - 1桁同士のかけ算を $n^{\log_2 3} \simeq n^{1.59}$ に減らすことが可能

```
  583
×174
------
   12 = 3x4
  32  = 8x4
20    = 5x4
   21 = 3x7
      :
      :
```

長大数のかけ算に対する分割統治法

- 以下，説明を簡単にするために $n = 2^k$ と仮定
- x と y を n/2 桁ごとに2分割する



$$
\begin{aligned}
xy &= (a \cdot 10^{n/2} + b)(c \cdot 10^{n/2} + d) \\
&= ac \cdot 10^{n} + (ad + bc) \cdot 10^{n/2} + bd \\
&= ac \cdot 10^{n} + [(ac + bd) - (a - b)(c - d)] \cdot 10^{n/2} + bd
\end{aligned}
$$

ac, bd, (a-b)(c-d) のかけ算の結果がわかれば，
あとはたし算と桁シフトのみを使って計算可能

n桁同士のかけ算➔n/2桁同士のかけ算3回　に分割

長大数のかけ算の時間計算量

- T(n): n桁同士のかけ算の際に必要な，
  1桁同士のかけ算の回数
- 明らかに T(1) = 1

n桁同士のかけ算➔n/2桁同士のかけ算3回　に分割

T(n) = 3 T(n/2)

$$T(n) = 3 \cdot T(\tfrac{n}{2}) = 3^2 \cdot T(\tfrac{n}{2^2}) = 3^3 \cdot T(\tfrac{n}{2^3}) = \cdots$$

$$= 3^{\log_2 n} \cdot T(1) = n^{\log_2 3}$$

※nが2のべき乗でない場合も，同様のやり方で同様の結果が得られる

※一般に r 進数に対しても，同様の結果が成り立つ．

部分和問題 (subset-sum problem)

- 入力: n+1 個の正整数 (a_0, a_1, …, a_{n-1}; b)
- 出力: $a_0x_0+a_1x_1+\ldots+a_{n-1}x_{n-1}=b$
  を満たす0-1ベクトルが存在する➔yes, しない➔no

（別の見方: a_0,a_1, …,a_{n-1}の幾つかを足し合わせて bに一致する➔yes, どう足し合わせても一致しない➔no）

例1: $a_0=5$, $a_1=3$, $a_2=2$, b =7 のとき,
$a_0+a_2=b$ が成り立つ ➔ yes

例2: a0=5, a1=3, a2=2, b =6 のとき,
a0, a1, a2をどのように足し合わせても b に一致しない
➔ no

部分和問題の解の性質

部分和問題 $(a_0, a_1, \ldots, a_{n-2}, a_{n-1}; b)$ は

$x_{n-1}=1$ を満たす解をもつ

⇔部分和問題 $(a_0, a_1, \ldots, a_{n-2}; b-a_{n-1})$ は解をもつ

$x_{n-1}=0$ を満たす解をもつ

⇔部分和問題 $(a_0, a_1, \ldots, a_{n-2}; b)$ は解をもつ

元の問題が解をもつ⇔部分問題は解をもつ

部分和問題 $(a_0, a_1, \ldots, a_{n-2}, a_{n-1}; b)$ の

部分問題 $(a_0, a_1, \ldots, a_{k-1}, a_k; p)$

ただし，$0 \leqq k \leqq n-1$, $0 \leqq p \leqq b$

※ $k=n-1$, $p=b$ のとき，元の問題に一致

部分和問題の解法

解の性質を使うと，部分問題を繰り返し解くことで
元の問題の解を得ることが可能

部分問題(a_0; p) を 0≦ p ≦b について解く

部分問題(a_0, a_1; p) を 0≦ p ≦b について解く

部分問題(a_0,…, a_{n-2}; p) を 0≦ p ≦b について解く

部分問題(a_0,…, a_{n-2}, a_{n-1}; p) を0≦ p ≦bについて解く

一つの部分問題は定数時間で解ける➔時間計算量O(nb)

部分和問題の解法の例

部分和問題(3,7,5,8,2; 11) を解く

部分問題(3; p) を 0≦ p ≦11 について解く

k＼p	0	1	2	3	4	5	6	7	8	9	10	11
0	○	×	×	○	×	×	×	×	×	×	×	×
1												
2												
3												
4												

p＝0 のときは
常に
解をもつ

p＝a_0 のときは
解をもつ
それ以外は
解をもたない

部分和問題の解法の例

部分和問題(3,7,5,8,2; 11) を解く

部分問題(3, 7; p) を 0≦ p ≦11 について解く

k＼p	0	1	2	3	4	5	6	7	8	9	10	11
0	○	×	×	○	×	×	×	×	×	×	×	×
1	○	×	×	○	×	×	×	○	×	×	○	×
2												
3												
4												

(a_0; p)が解をもつならば,
(a_0,a_1; p)も解をもつ

(a_0; p-a_1)が解をもつならば,
(a_0,a_1; p)も解をもつ

部分和問題の解法の例

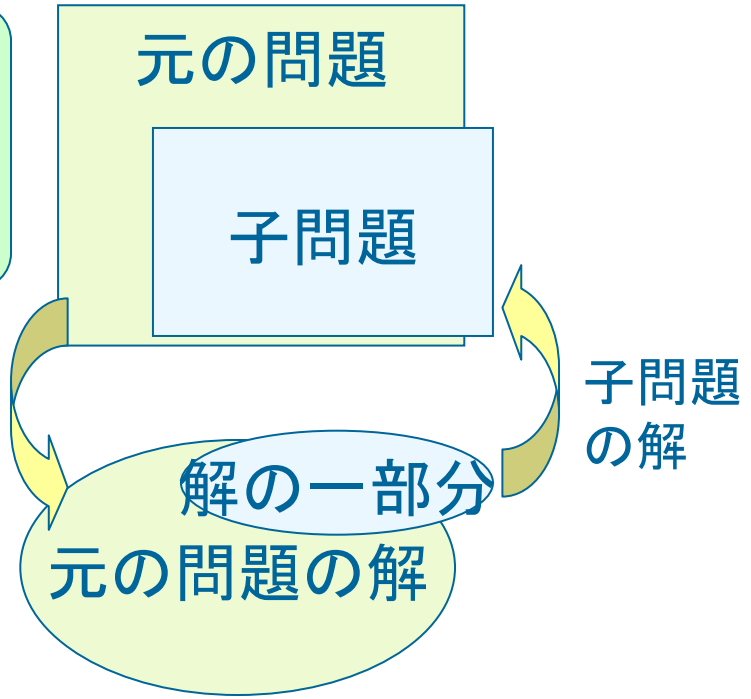
部分和問題(3,7,5,8,2; 11) を解く

k＼p	0	1	2	3	4	5	6	7	8	9	10	11
0	○	×	×	○	×	×	×	×	×	×	×	×
1	○	×	×	○	×	×	×	○	×	×	○	×
2	○	×	×	○	×	○	×	○	○	×	○	×
3	○	×	×	○	×	○	×	○	○	×	○	○
4	○	×	○	○	×	○	×	○	○	○	○	○

動的計画法

「最適性の原理」
元の問題の解の部分解は，
元の問題の部分問題の解である

再帰的なアルゴリズム



このアプローチ，もしくはこのアプローチにより得られる
再帰的アルゴリズムを動的計画法という

0-1ナップサック問題と部分問題

- 0-1ナップサック問題（入力は全て正の整数）

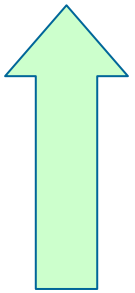
$$\text{最大化} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{条件} \quad \sum_{j=1}^{n} a_j x_j \leq b$$

$$x_j \in \{0,1\} \ (j = 1,2,\ldots,n)$$

元の問題の最適値

$f_n(b)$



- 部分問題（r = 1, 2,..., n, λ = 0, 1, ..., b)

$$(P_r(\lambda)) \quad \text{最大化} \quad \sum_{j=1}^{r} c_j x_j$$

$$\text{条件} \quad \sum_{j=1}^{r} a_j x_j \leq \lambda$$

$$x_j \in \{0,1\} \ (j = 1,2,\ldots,r)$$

この部分問題の最適値

$f_r(\lambda)$

0-1ナップサック問題の最適解の性質

- 部分問題(r = 1, 2,..., n, λ = 0, 1, ..., b)

$$(P_r(\lambda)) \quad \text{最大化} \quad \sum_{j=1}^{r} c_j x_j$$

$$\text{条件} \quad \sum_{j=1}^{r} a_j x_j \leq \lambda$$

$$x_j \in \{0,1\} \ (j = 1,2,\ldots,r)$$

この部分問題の最適値

$$f_r(\lambda)$$

- $P_r(\lambda)$の最適解 x* において $x^*_r = 0$ ならば
  $(x^*_1, \ldots, x^*_{r-1})$は$P_{r-1}(\lambda)$の最適解

$$f_r(\lambda) = f_{r-1}(\lambda)$$

- $P_r(\lambda)$の最適解 x* において $x^*_r = 1$ ならば
  $(x^*_1, \ldots, x^*_{r-1})$は$P_{r-1}(\lambda - a_r)$の最適解

$$f_r(\lambda) = f_{r-1}(\lambda - a_r) + c_r$$

0-1ナップサック問題に関する再帰式

$$x_r^* = 0 \Longrightarrow f_r(\lambda) = f_{r-1}(\lambda)$$
$$x_r^* = 1 \Longrightarrow f_r(\lambda) = f_{r-1}(\lambda - a_r) + c_r$$

$$f_r(\lambda) = \max\{f_{r-1}(\lambda), f_{r-1}(\lambda - a_r) + c_r\}$$

ただし $f_1(\lambda) = \begin{cases} 0 & (0 \le \lambda < a_1) \\ c_1 & (a_1 \le \lambda \le b) \end{cases}$

動的計画法

上記の再帰式をつかって,

$f_2(0), f_2(1), \ldots, f_2(b),\ f_3(0), f_3(1), \ldots, f_3(b),$
$\ldots,\ f_n(0), f_n(1), \ldots, f_n(b)$
を順に計算 $\Longrightarrow$ 元問題の最適値が得られる

計算時間
O(nb)

0-1ナップサック問題の最適解の計算

- 最適解も再帰的に計算可能

$$f_r(\lambda) = \max\{f_{r-1}(\lambda), f_{r-1}(\lambda - a_r) + c_r\}$$

ここで

$$x_r^* = 0 \iff f_r(\lambda) = f_{r-1}(\lambda)$$
$$x_r^* = 1 \iff f_r(\lambda) = f_{r-1}(\lambda - a_r) + c_r$$

よって,

$f_r(\lambda) = f_{r-1}(\lambda)$
ならば $P_r(\lambda)$ の最適解は $(x_1^*, \ldots, x_{r-1}^*, 0)$
ここで $(x_1^*, \ldots, x_{r-1}^*)$ は $P_{r-1}(\lambda)$ の最適解

$f_r(\lambda) = f_{r-1}(\lambda - a_r) + c_r$
ならば $P_r(\lambda)$ の最適解は $(x_1^*, \ldots, x_{r-1}^*, 1)$
ここで $(x_1^*, \ldots, x_{r-1}^*)$ は $P_{r-1}(\lambda - a_r)$ の最適解

レポート問題その1

この0－1ナップサック問題を
動的計画法で解きなさい

最大化　$10x_1 + 7x_2 + 25x_3 + 24x_4$

条件　$2x_1 + x_2 + 6x_3 + 5x_4 \leq 7$

$x_j \in \{0, 1\}\ (j = 1, 2, 3, 4)$

λ	0	1	2	3	4	5	6	7
f_1	0	0	10	10	10	10	10	10
f_2								
f_3								
f_4								

レポート問題その2

- 正しくない命題「$1+2+\ldots+(n-1)+n=O(n)$」に対する以下の証明のうち，どこが間違っているか説明しなさい.

証明:

n に関する数学的帰納法により証明する.

まず，$n=1$ のときは $1 = O(1)$ なので成立する.

次に，$n=k$ で成り立つと仮定して，$n=k+1$のときに成り立つことを証明する.

仮定より，$1+2+\ldots+(n-1)=O(n-1)$

したがって，$1+2+\ldots+(n-1)+n = O(n-1)+n=O(n)$

以上より，命題が証明された（証明終わり）

list.c

```
#include <stdio.h>

struct llist {
  int x;
  struct llist *next;
};

struct llist *newcell(int x)
{
  struct llist *pt;

  pt = (struct llist *) malloc(sizeof(struct llist));
  pt->x = x;
  return(pt);
}

main()
{
  struct llist *top, *p;

  p = newcell(1);
  p->next = NULL; top = p;

  p = newcell(3);
  p->next = top; top = p;

  p = newcell(2);
  p->next = top; top = p;

  p = top;
  do {
    printf("%d ", p->x);
    p = p->next;
  } while (p != NULL);
  printf("¥n");
}
```

プログラミングレポート課題1
リストの中身を表示した後，先頭のセルを削除し，その結果を表示するようにプログラムlist.cを修正せよ．

プログラミングレポート課題3
現在のリストの中身(2,3,1)を表示した後，上手にポインタを変更して，逆順(1,3,2)に並べ替えるようにプログラムlist.c，dlist.c を共に修正せよ．ただし，新しいセルは作ってはならない．

dlist.c

```
#include <stdio.h>

struct llist {
  int x;
  struct llist *next;
  struct llist *prev;
};

struct llist *newcell(int x)
{
  struct llist *pt;

  pt = (struct llist *) malloc(sizeof(struct llist));
  pt->x = x;
  return(pt);
}

main()
{
  struct llist *top, *bottom, *p;

  p = newcell(1);
  p->next = NULL; p->prev = NULL; top = p; bottom = p;

  p = newcell(3);
  p->next = top; top->prev = p; top = p;

  p = newcell(2);
  p->next = top; top->prev = p; top = p;

  p = top;
  do {
    printf("%d ", p->x);
    p = p->next;
  } while (p != NULL);
  printf("¥n");
}
```

プログラミングレポート課題2
リストの中身を先頭のセルから順に表示した後，逆に一番後ろのセルから順に表示するようにプログラムdlist.cを修正せよ．

参考文献：阿曽，他「Cによる情報処理入門」，昭晃堂，1997，第11章