データ構造と アルゴリズムI

第2回
アルゴリズムの設計

ケーキ分割問題

- 3人でケーキを分割する
- 人によって好みは異なるかも知れない (イチゴが欲しい，生クリームが好き，スポンジが好き)
- 目的は，公平な (fairな) 分割を達成すること
  - 誰もが，ケーキ全体の価値の1/3以上の価値を持つ分け前を得ることができる
  - 人によって価値観が異なるので，全員が1/3を超えることも可能

48

ケーキ分割問題（二人の場合）

- 二人の場合は簡単
- 二人をA, Bとする.
- Aがケーキを，自分の価値観で半分ずつに分ける.
- Bは，二つの好きな方を選ぶ.
- Aは残りを取る
- Bの分け前の（Bにとっての）価値は明らかに1/2以上，Aの分け前の価値は1/2.

49

ケーキ分割問題（3人の場合）

- A, B, Cの三人がいる場合，
- 二人の場合なら，それぞれが1/2以上を得る方法は分かっている
- これ（の変形版）をうまく部品として使えないか?
- 二人の間で，片方は2/3以上，もう片方は1/3以上を得るようにできる?

50

ケーキ分割問題（3人の場合）

- A, B, Cの三人がいる場合，
- まず，A, Bで，二人の場合のアルゴリズムでケーキを二分割する.
- A, Bは，それぞれ，自分の価値観で，自分の持っている半分を，ちょうど三等分する.
- Cは，Aの三等分，Bの三等分から，それぞれ一切れずつ取る.
- A, Bは，それぞれ自分の残りの二個を得る.

51

envy-freeな分割

- 二人の場合のアルゴリズムは，公平であると同時にenvy-free
  - 他者の取り分をうらやましく思うことはない
- 三人の場合のアルゴリズムはenvy-freeではない
  - Aはイチゴが好きなので，最初にイチゴを半分に分けて，自分の半分のイチゴをさらに三等分した
  - ところが，Bはイチゴを切らず，一つのピースに半分のイチゴが残っている
  - Cがこのピースを取って，Cは1/2+1/6=2/3のイチゴを手に入れた
  - 一方，Aは1/3のイチゴしか得られない
  - AはCがうらやましい！
- 3人の場合もenvy-freeにするにはどうしたらよい?

52

3人の場合のenvy-freeな分割

- Aが三等分
- Bが三つのピースを望ましい順に並べる (BにとってX1 ≧ X2 ≧ X3 とする)
- さらに，BはX1からEを切り取って，BにとってX1'=X1−E＝X2となるようにする.
- とりあえずEは脇によけておいて，C,B,Aの順に，X1', X2, X3から好きなのを取る. ただし，Bは自分の番でX1'が残っていたら取らないといけない.
- この時点で，Cは一番良いものを取っているのでenvy-free, Bにとって，X1'=X2≧X3で，X1'かX2のどちらかを取れるからenvy-free, Aは，自分としてはX2=X3>X1'で，X1'以外を取れるからenvy-free, さらに，AにとってX1'はE分の価値が低いことに注意.

53

3人の場合のenvy-freeな分割（続き）

- B, Cのどちらかは X1'を得ている. X1'を得た方をB', 得ていない方をC'とする.
- C'がEを三等分する. これをB', A, C'の順に取る
- C'は自分で三等分したので文句はない
- B'は最初に好きなのを選んでいるので文句はない
- 一般の場合だと，AがB'をうらやましいと思う可能性はあるが，B'はX1'を得ており，AにとってのX1'の価値は，自分のピースの価値よりE少ない. よってBがE中の価値を全部得ても合計ではうらやましくない.

54

卒研配属でのEnvy-Freeness

- 社会的な意思決定ではEnvy-freeを保証することは重要
- Envy-freeなら参加者は得られた結果に納得できる
- Gale-Shapleyを使った場合，卒研配属の結果はenvy-free?
- 当然，自分の行きたくて行けなかった研究室に配属された他の学生はうらやましい
- しかし，その学生の方が成績が良ければ，表立って文句は言えない
- 正当化されるようなenvyがないことは保証できる

55

1. アルゴリズム

- アルゴリズムとは
  - 入力 (input): ある値(の集合)
  - 出力(output): ある値(の集合)
  - 明確に定義された計算手続き
- 明確に定義された計算問題を解くための道具
- 問題の記述とは
  - 望むべき入出力関係を指定すること

56

1.1 ソーティング問題の形式的定義

- 入力: n 個の数の列 $\langle a_1, a_2, ..., a_n \rangle$
- 出力: $a'_1 \le a'_2 \le \ldots \le a'_n$ であるような入力列の置換 $\langle a'_1, a'_2, ..., a'_n \rangle$

- 入力例 (具体例, instance)
  - 入力 〈31, 41, 59, 26, 41,58〉
  - 出力 〈26, 31, 41, 41, 58, 59〉

57

ソーティング

- 計算機科学における基本的な操作
- 多くのアルゴリズムが開発されている
- 入力例によって優劣が異なる
  - ソートすべきデータの数
  - どの程度までデータがすでにソートされているか
  - 用いる記憶装置の種類 (主記憶, ハードディスク, CD-ROM/DVD, テープ)

58

アルゴリズムの正しさ／正当性

- アルゴリズムが正しい／正当 (correct)
  ⇒全ての具体例に対して正しい出力とともに停止する
  – 与えられた計算問題を解く (solve) という.
- 正しくないアルゴリズム
  – ある具体例に対して望ましい答えを出力せずに停止
  – ある具体例に対して全く停止しない
- 確率的アルゴリズムも存在
  – 高い確率で正しい答えを出す or 早く停止する

59

2.1 挿入ソートのアイデア

- 手に持っている複数のカード／トランプをソートすることを考える
- 自然なやり方：左から順に，一枚ずつ，正しい順番になるように，左側の (すでにソートされた列に) 挿入していく

60

講義でのプログラムの記述

- C++を用いる

```
#include <iostream>
using namespace std;

関数定義等
…

main () {
関数の呼び出し等
}
```

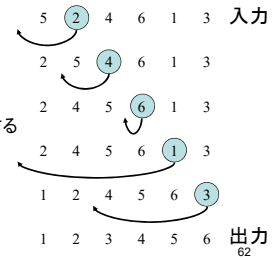
61

挿入ソート

- 入力: 長さ n の配列 A[1..n]
- 出力: ソートされた配列 A[1..n]

```
typedef int data; // プログラミング上は結構大事なポイント（薄皮を被せる）

void INSERTION-SORT(data A[], int n) {
  data key;
  int i, j;
  for (j=2; j <= n; j++) {
    key = A[j];
    //  A[j] をソート列 A[1..j-1] に挿入する
    i = j – 1;
    while (i > 0 && A[i] > key) {
        A[i+1] = A[i];
        i = i -1;}
    A[i+1] = key;}}
```



62

挿入ソート（続き）

```
main () {
  const int n=6;
  data D[n+1]={-1,5, 2, 4, 6, 1, 3};
  // D[0]はdummy
  INSERTION_SORT(D,n);
}
```

63

プログラム上の注意（C++）

- 配列の宣言と初期化：int D[3]={3, 2, 1};
- 配列の要素へのアクセス D[i]
  – D[0]が最初の要素だが，教科書との整合性のため，D[0]はなるべく使わず，D[1]から使うようにする
- 配列の受け渡し：
  – 関数側: void INSERTION-SORT(int A[], int n)
  – 呼び出し側：INSERTION-SORT(D, n)

64

演習

- 挿入ソートの動作を手でトレースする
- <5, 2, 4, 1, 3, 8, 6, 7>

65

アルゴリズムの正当性

ループ不変式：ループの繰り返しで常にある性質が成立することを証明する．具体的には以下の通り．

1. 初期条件：ループの実行開始直後ではループ不変式は真であることを示す．
2. ループ内条件：ループの何回目かの繰り返しの直前でループ不変式が成立すると仮定すると，次の繰り返しの直前でもループ不変式は真であることを示す
3. 終了時条件：1,2より，ループが終了した時にループ不変式が成立することから，アルゴリズムの正当性（もしくは証明を手助けする有力な性質）を導く

66

挿入ソートでのループ不変式

- forループの繰り返しが開始される時，A[1,…,j-1]には，入力時点でA[1,…,j-1]に入っていた要素がソートされて入っている
- 初期条件: j=2なので，A[1]にはA[1]に入っていた要素がソートされて入っている
- ループ内条件：A[1,…,j-1]にソートされた要素が入っているなら，A[j]を正しい場所に挿入するので成立
- 終了時条件：終了時はj=n+1なので，A[1,…,n]には入力時にA[1,…,n]に入っていた要素＝すべての要素がソートされている!

67

2.2. アルゴリズムの解析

- アルゴリズムの実行に必要な資源を予測したい
  - メモリ量
  - 通信バンド幅，論理ゲート数
  - 計算時間
- 解析を行うにはモデルを仮定する必要がある
- RAM (random access machine) モデル
  - 命令は1つずつ実行
  - どのメモリ番地も一定の時間で読み書き可
  - 解析を容易にするためにかなり簡単化したモデル

68

挿入ソートの解析

- INSERTION-SORTの手続きに要する時間は入力によって変わる（ソートするデータ数が多ければ，小さい場合より時間がかかるのは当たり前）．
- 一般に，プログラムの実行時間は入力のサイズの関数で表す．
- 入力サイズの定義
  - ソーティング，離散フーリエ変換など: データ数
  - 整数の積の計算など: 入力のビット数
  - グラフの問題: グラフの頂点と辺の数

69

実行時間の定義

- 実行される基本的な演算の回数
- プログラムの第 i 行の実行に c 時間かかるとする（c は定数，すべての行で同じとする）
- 注: サブルーチン呼び出しは定数時間ではない

70

```
void INSERTION-SORT(data A[], int n)        コスト    回数
{
   data key;
   int i, j;
   for (j=2; j <= n; j++) {                  c        n
      key = A[j];                            c        n-1
      //  A[j] をソート列 A[1..j-1] に挿入する
      i = j – 1;                             c        n-1

      while (i > 0 && A[i] > key) {          c        Σ_{j=2}^{n} t_j

         A[i+1] = A[i];                      c        Σ_{j=2}^{n} (t_j – 1)

         i = i -1;                           c        Σ_{j=2}^{n} (t_j – 1)
      }
      A[i+1] = key;                          c        n-1
   }
}
```

t_j: while ループが j の値に対して実行される回数

INSERTION-SORTの実行時間

$$T(n) = cn + c(n-1) + c(n-1) + c\sum_{j=2}^{n} t_j + c\sum_{j=2}^{n} (t_j - 1) + c\sum_{j=2}^{n} (t_j - 1) + c(n-1)$$

t_j の値は入力によって変化する.
最良の場合 = 配列が全てソートされている場合
$t_j = 1$

$$\begin{aligned} T(n) &= cn + c(n-1) + c(n-1) + c(n-1) + c(n-1) \\ &= 5cn - 4c \\ &= an + b \end{aligned}$$

n の線形関数 (linear function)

最悪の場合

- 配列が逆順にソートされている場合
  - $t_j = j$

$$\begin{aligned} T(n) &= cn + c(n-1) + c(n-1) + c\left(\frac{n(n+1)}{2} - 1\right) \\ &\quad + c\left(\frac{n(n-1)}{2} - 1\right) + c\left(\frac{n(n-1)}{2} - 1\right) + c(n-1) \\ &= an^2 + bn + c' \end{aligned}$$

n の2次関数 (quadratic function)

時間計算量 (Time Complexity)

- アルゴリズムの実行時間は入力に依存する
- アルゴリズムの解析では通常最悪時の実行時間を考える
  - 最悪時の実行時間は任意の入力に対する実行時間の上界
  - 最悪時の実行時間を保証する
  - 「最悪時」は頻繁に起きる (データベース検索など)
  - 平均時も最悪時と同じくらい悪いことが多い (挿入ソートでランダムな数を挿入した場合等)

増加のオーダ

- 実行時間の解析を容易にするための抽象化
  - 各行の実行時間 (コスト) を定数とする
  - 最悪の実行時間が an^2+bn+c だとする
  - 実行時間の増加率をみるには主要項 an^2 で十分
  - 定数係数も無視
  - $\Theta(n^2)$ と表す
- 挿入ソートは $\Theta(n^2)$ という最悪実行時間を持つ
- あるアルゴリズムが他より効率がよい
  ⇔最悪実行時間の増加率が低い

2.3 アルゴリズムの設計

- 挿入ソート: 逐次添加法 (incremental approach)
  - 部分的な解に, 要素を追加していく
- 分割統治法 (divide-and-conquer) に基づく方法
  →マージソート
  - 実行時間の解析が容易であることが多い

分割統治法

- 問題の再帰的な (recursive) 構造を利用
  分割：問題をいくつかの小さな部分問題に分割
  統治：各部分問題を再帰的に解く
  統合：それらの解を組合わせて元の問題の解を構成
- マージソートでは
  分割：n 要素の列を $n/2$ 要素の2つの部分列に分割
  統治：マージソートを用いて2つの部分列をソート
  統合：2つのソートされた部分列を統合して答を得る

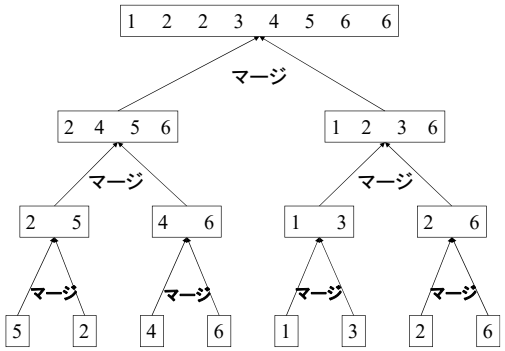
77

マージソート

```
void MERGE-SORT(data A[], int p, int r)    // A[p..r] をソート
{
   int q;
   if (p < r) {                              // p==r ならソートする必要なし
         q = (p+r)/2;
         MERGE-SORT(A, p, q);      //  A[p..q] を再帰的にソート
         MERGE-SORT(A, q+1, r); //  A[q+1..r] を再帰的にソート
         MERGE(A, p, q, r);
                  // ソートされた部分列 A[p..q] と A[q+1..r] を統合
   }
}

main ()
{
  初期化
   MERGE-SORT(D,1,n);
}
```

78



79

マージ

```
#include <climits>
#define infinity  INT_MAX
void MERGE(data A[], int p, int q, int r)
            // ソートされた部分列 A[p..q] と A[q+1..r] を統合
{ int n, n1, n2, i, j, k;
  n = r-p+1;
  data L[n+1];
  data R[n+1]; // 部分列を保存する一時的な配列
  n1=q-p+1; n2=r-q;
  for (i=1;i<=n1;i++) L[i]=A[p+i-1]; //一時的な配列に保存
  for (i=1;i<=n2;i++) R[i]=A[q+i]; //一時的な配列に保存
  L[n1+1]=infinity; R[n2+1]=infinity; //番兵
                                  注意：i++は副作用としてiの値を1増加
  i = 1; j = 1;                   させるが，返す値はiの増加する前の値.
  for (k=p; k<=r;k++)             ++iだと返す値はiの増加後の値.
    if (L[i] <= R[j]) A[k] =L[i++];  // 前半のほうが小さい
    else A[k] = R[j++];            //  後半のほうが小さい
}
```

80

番兵

- 本来なら，L[], R[]に関して，最後の要素まで A[]に移動済みかどうかのチェックが必要
- この処理を避けるため，L[], R[]の最後に，他の要素よりも必ず大きくなる値（番兵）を置く
- 番兵の先の要素にアクセスすることを回避できる

81

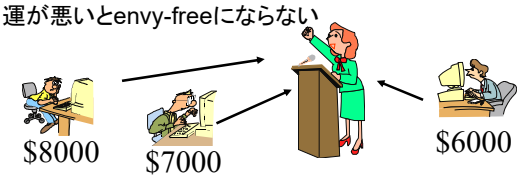
演習

- マージソートの動作を手でトレースする
- <5, 2, 4, 1, 3, 8, 6, 7>

82

Envy-freeな割当て

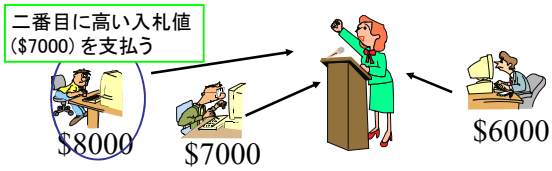
- ある財を参加者の誰か一人に割り当てる
- 通常の入札を使う場合，各参加者は，自分の評価値（それ以上は払いたくない，ぎりりぎりの値）から，いくらかディスカウントした値を入札
- 運が悪いとenvy-freeにならない



83

Vickrey入札

- 最高値の入札者が落札
- 支払う金額は二番目に高い入札値



84

Vickrey入札の性質

- 自分の支払う意思のあるぎりぎりの金額を入札するのが最適 (正直が最良の策).
- 他者の入札値を知っても利益にならない.
  - 自分の評価値が$8000の場合:

  (A) 他者の入札値の最高額が$8000未満: 何を入札しても支払額は同じ

  (B) $8000以上: 何を入札しても利益を得ることは不可能

全員が（自発的に）正直な入札をして，必ずenvy-freeな割当てが可能

85