データ構造と アルゴリズム I

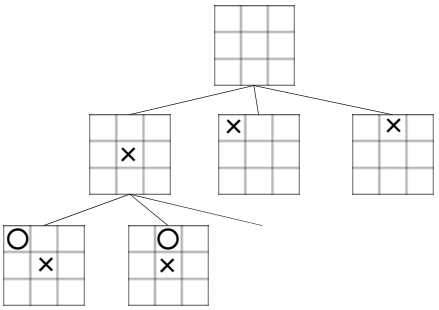
第7回

予定 (II)

6/9: 第6回：ヒープソート
6/16: 第7回：クイックソート
6/23: 休講
6/30: 小テスト
7/7: 第8回：線形時間ソーティング
7/14:第9回：ハッシュ表
7/21:第10回：2分探索木
7/28: 定期試験

木の利用例（ゲーム木）

- 自分の手番／相手の手番で分岐していく



例題

- 二人で交代に，1から順に25までの数を言う．
- 言う数の個数は，1個，2個，3個のいずれか好きなのを選んでよい
- 最後に25を言った方が負け

必勝法

- 24を言って，相手に順番を回せば絶対勝ち
- 一方，20を言って，相手に順番を回せば，相手が何個を選んでも，次に24を言える --- 絶対勝ち
- 同様に，16を言って，相手に回せば次に20を言える --- 絶対勝ち
- 同様に，12, 8, 4を言って回せば勝ち
- 先手が何を言おうと，後手は4を言って回せる
- 結局，後手が必勝

このゲームの性質

- 二人で交代に順番が回ってくる
- 自分の前の相手の行動／手は完全に観測できる
- 偶然の入る余地がない
- 多くのゲームは同様な性質を持つ
  - チェス，将棋，オセロ，囲碁，五目並べ，etc.
- 上記の性質を満たさないもの
  - バックギャモン：さいころ
  - ポーカー：相手の手は見えない
  - ブリッジ：プレイヤの協調

必勝法

- 二人，完全情報，決定的なゲームは，原理的には必勝法が存在する
  - 先手必勝／後手必勝／引き分け
- 先手／後手を決めた時点で勝負はついている (ゲームをするまでもない)!

257

必勝法 (続き)

- 簡単なゲームなら必勝法が分かる
  - ○×（三目並べ）　引き分け
  - 五目並べ　　先手必勝
  - 6x6 オセロ　　後手必勝
- 複雑なゲームでは分かっていない
  - 分かってしまえばゲームは終り?

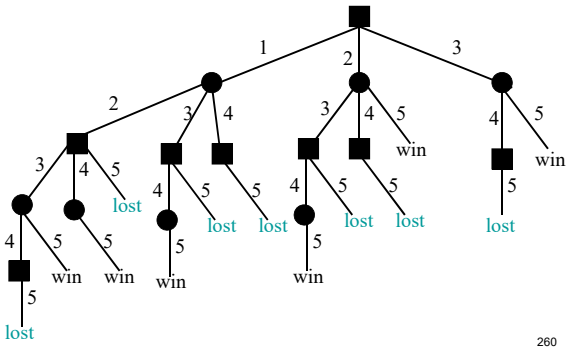
258

ゲームの木

- 状態／ノード: ゲームの可能な状態
- 状態の遷移／リンク: 正しい手により遷移可能な状態間を結ぶ（一方向）.
- 先手をMAXプレイヤ，後手をMINプレイヤ，先手の順番（手番）に対応する状態をMAXノード，後手の手番の状態をMINノードと呼ぶ.
- 勝ち負けが決まったノードを端点と呼ぶ

259

例:5を言ったら負け



260

ノードのラベル付け（考え方）

- お互いに自分が勝つようにベストを尽くす
- win/lostのラベルは先手（MAXプレイヤ）の立場
- MAXプレイヤは，絶対勝てる手があればそれを選び，後手（MINプレイヤ）は，MAXプレイヤを絶対負かすことができる手があれば，それを選ぶ

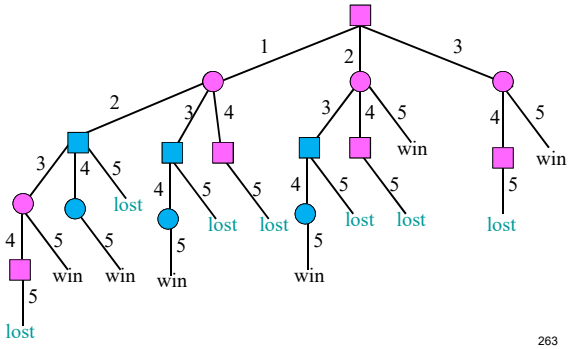
261

ノードのラベル付け

- 以下のように再帰的に定義
  - 端点に関して，そのままwin/lost
  - MAXノードに関しては，子ノードに少なくとも一つwinがあればwin，すべてlostならlost
  - MINノードに関して，子ノードに少なくとも一つlostがあればlost，すべてwinならwin
  - winを100，lostを-100とすると，上記の処理はMAXノードでは子ノードの最大値，MINノードでは最小値を取ることに対応

262

ノードのラベル付け



例題： ニム（コイン取り）

- コインが1個と6個の列
- 交互に，1個もしくは隣り合う2個を取る
- 最後に1個もしくは隣り合う2個を取った方が勝ち
- 先手必勝／必負?, 木を書いて確かめよう

状態／ノード

- 各列の個数の (小さい順に並べた) リストで表現：初期状態は (1, 6)
- 初期状態から遷移可能な状態:
  (6), (1, 5), (1, 4), (1, 1, 4), (1, 2, 3)…
- すべての木を展開するのは大変なので，とりあえず (1, 4) から木を展開してみよう

ゲーム木の展開

必勝法を見つけるためには

- 必ずしも木を完全に展開する必要はない
  - あるMAXノードに関して，子ノードに少なくとも一つのWINがあれば，そのMAXノードはWIN
    - 他の子ノードは展開しなくても良い
  - あるMINノードに関して，子ノードに少なくとも一つLOSTがあれば，そのMINノードはLOST
    - 他の子ノードは展開しなくて良い

ゲーム木のサイズ

- チェッカー　10の30乗　世界チャンピオン
- オセロ　10の60乗　世界チャンピオン
- チェス　10の120乗　世界チャンピオン
- 将棋　1 2013年A級プロ棋士に勝利！
- 囲碁　10の360 2016年アルファ碁がトッププロに勝利！

~~チェッカーでも必勝法は~~
2007年に引き分けであることが証明された

クイックソート

- n 個の数に対して最悪実行時間$\Theta(n^2)$のソーティングアルゴリズム
- 平均実行時間は$\Theta(n \log n)$
- Θ記法に隠された定数も小さい
- in-place (一時的な配列が必要ない)

7.1 クイックソートの記述

- 分割統治法に基づく
- 部分配列 $A[p..r]$ のソーティング

1. 部分問題への分割:
   - 配列 $A[p..r]$ を2つの部分配列 $A[p..q]$ と $A[q+1..r]$ に再配置する.
   - $A[p..q]$ のどの要素も $A[q+1..r]$ の要素以下にする.
   - 添字 q はこの分割手続きの中で計算する.
2. 部分問題の解決 (統治):
   - 部分配列 $A[p..q]$ と $A[q+1..r]$ を再帰的にソート
3. 部分問題の統合
   - $A[p..r]$ はすでにソートされているので何もしない

クイックソートのコード（章末問題7.1のHoareバージョン，）

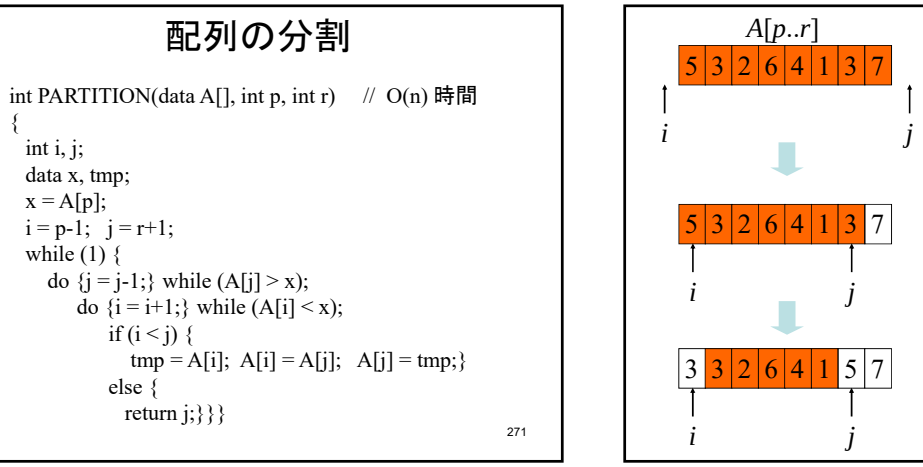
```
void QUICKSORT(data A[], int p, int r)  {
  int q;

  if (p < r) {
     q = PARTITION(A,p,r);
     QUICKSORT(A,p,q);
     QUICKSORT(A,q+1,r);
  }
}

main(){
    配列Dの初期化
    QUICKSORT(D,1,n);
}
```


配列の分割

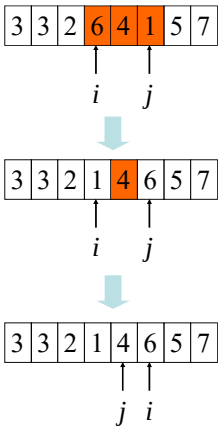
```
int PARTITION(data A[], int p, int r)     // O(n) 時間
{
  int i, j;
  data x, tmp;
  x = A[p];
  i = p-1;   j = r+1;
  while (1) {
     do {j = j-1;} while (A[j] > x);
        do {i = i+1;} while (A[i] < x);
           if (i < j) {
              tmp = A[i];  A[i] = A[j];   A[j] = tmp;}
           else {
             return j;}}}
```

初期状態:
i と j は配列の範囲外
$x = A[p] = 5$ によって分割
x: 枢軸 (pivot) と呼ぶ

$A[i] \geq x$
$A[j] \leq x$ となる最初の i, j
7 は正しい分割位置にある

$A[i]$ と $A[j]$ を交換
正しい分割位置になる



$A[i] \geq x$
$A[j] \leq x$ となる最初の i, j

$A[i]$ と $A[j]$ を交換

$i \geq j$ となったので
$q = j$ を返す
$A[p..q]$ は x 以下
$A[q+1..r]$ は x 以上

課題

- A=<5, 2, 6, 4, 1, 3>に対して，PARTITION(A,1, 6) を実行した結果を求めよ
- 上記の配列Aに対してQUICKSORTを実行した場合，手続きPARTITIONは何回呼び出されるか?

クイックソートの正しさ

- クイックソートの基本的なアイデアはきわめてシンプル
- 細かなプログラミングは結構 微妙
  - 自分で考えて作ろうとすると幾つかの落とし穴がある
    - 無限ループ!
    - qが配列から外れる!

275

課題

枢軸 (pivot) を最初の値A[p]でなく，最後の値A[r]にしたらどうなる?

A=<5, 2, 6, 4, 1, 3>なら? 他の値では?

```
int PARTITION(data A[], int p, int r)     // O(n) 時間
{
  int i, j;
  data x, tmp;
  x = A[r];
  i = p-1;   j = r+1;
  while (1) {
     do {j = j-1;} while (A[j] > x);
        do {i = i+1;} while (A[i] < x);
           if (i < j) {
              tmp = A[i];  A[i] = A[j];   A[j] = tmp;}
           else {return j;}}}
```

276

課題の解答

- 入力がすでにソートされている場合，例えばA=<1,2>の場合，PARTITION(A, 1, 2)が呼ばれる
- pivotのx=2, j=2, i=2で止まり，i<jではないので，j=2がPARTITIONの返り値となる
- よって，q=2なので，PARTITION(A, 1, 2)で無限ループ
- pivotにA[r]を使いたければ，どこを変更すべき?

```
void QUICKSORT(data A[], int p, int r)   {
  int q;
  if (p < r) {
     q = PARTITION(A,p,r);
     QUICKSORT(A,p,q);
     QUICKSORT(A,q+1,r);}}
```

277

PARTITIONの正当性

- PARTITIONの返り値を q とする
- $A[p..r]$ の分割後に満たされるべき条件
  - $A[p..q]$ はある pivot x 以下
  - $A[q+1..r]$ は x 以上
  - $p \le q < r$
- $q = r$ となると，QUICKSORTは停止しない（再帰呼び出しの引数が変わらない）.
- どんな A に対しても $q < r$ となることを保証する必要がある

278

PARTITIONの正当性 (続き)

q < r が成立する理由:

- 枢軸を最初の要素 A[p] にすることが重要，最後の要素 A[r] にしてしまうとダメ!
- 枢軸との比較で，iとjの両方で等号も含んで止まることもポイント
- iは必ず最初の要素 A[p] で止まる
- jが最後の要素で止まらなければOK，止まったとしても i=p との交換で先に進む

279

PARTITIONの正当性 (続き)

p ≤ q が成立する理由:

- 終了条件が i ≥ j であることがポイント
- 枢軸よりも小さい要素があれば，その先には j は進めない
- 枢軸が最小の要素であれば，外側のループで j=pまで進む
- i=p なので終了条件が成立して，j=pで終了

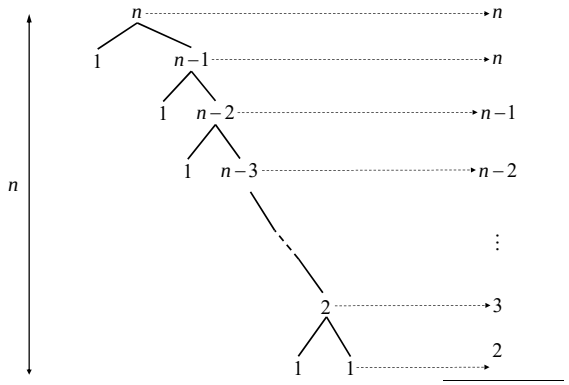
280

7.2 クイックソートの性能

- クイックソートの実行時間は分割が平均化されているかどうかに依存
- これはpivotの選択に依存
- 分割が平均化されていれば実行時間は漸近的にマージソートと同じ ($\Theta(n \log n)$)
- 最悪時は $\Theta(n^2)$

最悪の分割

- 最悪の場合は，PARTITIONによって領域が 1 要素と n-1 要素に分けられる場合
- 分割には $\Theta(n)$ 時間かかる
- 実行時間に対する漸化式は
  - $T(n) = T(n-1) + \Theta(n), \quad T(1) = \Theta(1)$
- $T(n) = \Theta(n^2)$
- 最悪実行時間は挿入ソートと同じ
- 入力が完全にソートされているときに起こる (挿入ソートなら $O(n)$ 時間)

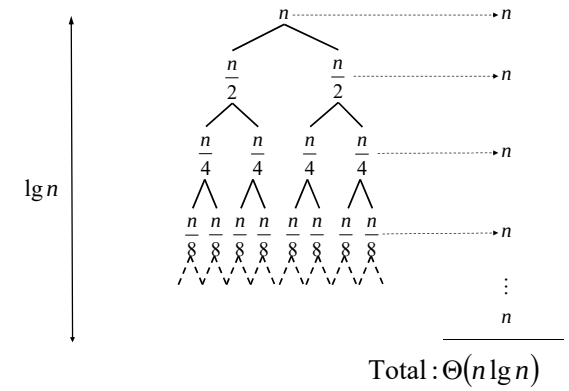


再帰木

$\text{Total} : \Theta(n^2)$

最良の分割

- クイックソートが最も速く動作するのは，PARTITIONによってサイズ $n/2$ の2つの領域に分割されるとき．
- $T(n) = 2T(n/2) + \Theta(n)$
- $T(n) = \Theta(n \lg n)$
- ちょうど半分に分割される場合が最速



$\text{Total} : \Theta(n \lg n)$

均等分割

- PARTITIONで常に 9 対 1 の比で分割されると仮定する
- $T(n) = T(9n/10) + T(n/10) + \Theta(n)$
- 再帰木の各レベルのコストは $O(n)$
- 再帰の深さは $\log_{\frac{10}{9}} n = \Theta(\lg n)$
- 漸近的には中央で分割するのと同じ
- 分割の比が 99 対 1 でも同じ (定数比なら良い)

課題

- A=<4, 1, 5, 6, 2, 3>に対して，PARTITION(A,1, 6) を実行した結果を求めよ
- 上記の配列Aに対してQUICKSORTを実行した場合，手続きPARTITIONは何回呼び出されるか?

287

乱択の強さ

- ランダム／行き当たりばったりは悪いこと?
  - 敵／意地の悪い自然の選択に対抗するには有効
  - 最悪の場合（相手に読まれてしまう場合）を回避
- 例：ジャンケンで絶対に負けない方法
  - 確率1/3でグー，チョキ，パーを混ぜる

288

7.3 乱択版クイックソート

- クイックソートの平均的な場合の実行時間を解析する場合，入力の頻度を仮定する必要がある.
- 通常は，すべての順列が等確率で現れると仮定
- しかし実際にはこの仮定は必ずしも期待できない
  - 最悪の場合がほとんど生じないとは断言できない
  - 最悪の場合が頻繁に生じるかもしれない
- この仮定が成り立たなくてもうまく動作するクイックソートの乱択版アルゴリズムを示す

289

乱択 (randomized) アルゴリズム

- 動作が入力だけでなく乱数発生器 (random-number generator)に依存するアルゴリズム
- 関数 RANDOM(a,b): a 以上 b 以下の整数を等確率で返すとする.
- プログラミング言語は擬似乱数発生器 (pseudorandom-number generator) を備える
- 擬似乱数: 統計的にはランダムに見えるが，決定的に作られる数(の列)

290

乱択アルゴリズム1

- クイックソートを行う前に入力配列の要素をランダムに並び替える
- 実行時間は入力順序に依存しなくなる
- アルゴリズムがうまく動作しないのは，乱数発生器によって運の悪い順列を作る場合のみ
- 最悪の実行時間は改善されない ($\Theta(n^2)$)
- 最悪の場合はほとんど起きない --- 入力に依存しない一定の小さい確率で生じる

291

乱択アルゴリズム2

- 配列を $A[p..r]$ を分割する前に，$A[p]$ と $A[p..r]$ からランダムに選んだ要素を交換
- pivotが $r–p+1$ 個の要素から等確率で選ばれることを保障する
- 分割が平均的にはバランスのとれたものになることが期待できる

292

```
int RANDOMIZED_PARTITION(data A[], int p, int r)
{
  int i;
  data tmp;
  i = RANDOM(p,r);
  tmp = A[i];  A[i] = A[p];   A[p] = tmp;    //  値の交換
  return PARTITION(A,p,r);
}

void RANDOMIZED_QUICKSORT(data A[], int p, int r)
{
  int q;
  if (p < r) {
      q = RANDOMIZED_PARTITION(A,p,r);
      RANDOMIZED_QUICKSORT(A,p,q);
      RANDOMIZED_QUICKSORT(A,q+1,r);
  }
}
```

293

7.4 平均的な場合の解析

- $T(n)$: サイズ n の入力に対するRANDOMIZED QUICKSORTの平均実行時間を考える
- すべての値は異なることを仮定
- 小さい方から数えてi番目の要素と，i+k番目の要素が比較される確率は2/(k+1)未満
  - 比較は必ずpivotとの間．i番目とi+k番目の間の要素が先にpivotに選ばれると決して比較されない．どちらかがk+1の要素中で最初にpivotに選ばれた時にのみ比較が行われる

$$\sum_{i=1}^{n-1}\sum_{k=1}^{n-i}\frac{2}{k+1} < \sum_{i=1}^{n-1} O(\lg n) = O(n \lg n)$$

294