データ構造と アルゴリズム I

第10回

予定 (II)

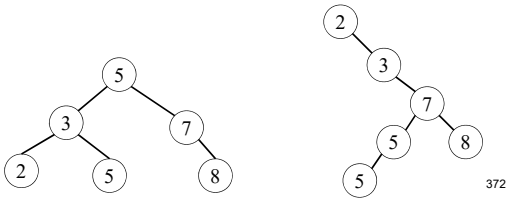
6/9: 第6回：ヒープソート
6/16: 第7回：クイックソート
6/23: 休講
6/30: 小テスト
7/7: 第8回：線形時間ソーティング
7/14:第9回：ハッシュ表
7/24 (月):第10回：2分探索木
7/28: 定期試験　工学部第三講義室

12. 2分探索木

- 探索木: SEARCH, MINIMUM, MAXIMUM, PREDECESSOR, SUCCESSOR, INSERT, DELETE等が利用できる動的集合用データ構造
- 辞書やプライオリティーキューとして利用できる
- 基本操作は木の高さに比例した時間がかかる
  - ランダムに構成された2分探索木の高さ: O(lg *n*)
  - 最悪時: O(*n*)
- 最悪時でも O(lg *n*) に改良できる (2色木)

12.1 2分探索木とは何か?

- 各節点は key, left, right, p フィールドを持つ
- 2分探索木条件 (binary-search-tree property)
  - 節点 *y* が *x* の左部分木に属する⇒ key[*y*]≤key[*x*]
  - 節点 *y* が *x* の右部分木に属する⇒ key[*x*]≤key[*y*]



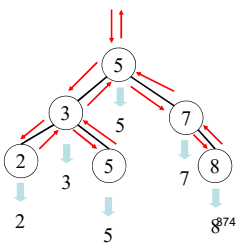
比較:
ヒープ条件 (Heap Property)

- 根以外の任意の節点 i に対して
  $A[\mathrm{PARENT}(i)] \geq A[i]$
- つまり，節点の値はその親の値以下
- ヒープの最大要素は根に格納される

木の中間順巡回

- 木の中間順巡回 (通りがけ順, inorder tree walk)
  - 根の左部分木に出現するキー集合
  - 根のキー
  - 右部分木に出現するキー集合の順にキーを出力
- 木の根から辿ると，全てのキーをソートされた順序で出力できる
- Θ(*n*) 時間

```
INORDER_TREE_WALK(node *x)
{
  if (x != NIL) {
    INORDER_TREE_WALK(left(x));
    cout << key(x) << endl;
    INORDER_TREE_WALK(right(x));
  }
}
```



演習：中間順巡回

- この手続きにより，キーがソートされた順序で印刷されることを証明せよ
  - ヒント：木の深さに関する帰納法

```
INORDER_TREE_WALK(node *x)
{
   if (x != NIL) {
      INORDER_TREE_WALK(left(x));
      cout << key(x) << endl;
      INORDER_TREE_WALK(right(x));
   }
}
```


その他の巡回法

- 先行順巡回 (行きがけ順, preorder tree walk): 根節点を先に出力し，次に左右の部分木を出力
- 後行順巡回 (帰りがけ順, postorder tree walk): 先に左右の部分木を出力し，最後に根節点を出力

```
PREORDER_TREE_WALK(node *x)
{
  if (x != NIL) {
     cout << key(x) << endl;
     PREORDER_TREE_WALK(left(x));
     PREORDER_TREE_WALK(right(x));
  }
}
```

```
POSTORDER_TREE_WALK(node *x)
{
   if (x != NIL) {
      POSTORDER_TREE_WALK(left(x));
      POSTORDER_TREE_WALK(right(x));
      cout << key(x) << endl;
   }
}
```


12.2. 2分探索木に対する質問

- 質問操作は高さに比例した時間で終了する
- 探索: 2分探索木の中から，ある与えられたキーを持つ節点のポインタを求める(存在しなければNIL)

```
TREE_SEARCH(node *x, data k)
{
   if (x == NIL || k == key(x)) return x;
   if (k < key(x)) return TREE_SEARCH(left(x),k);
   else  return TREE_SEARCH(right(x),k);
}
```


探索の正当性

- キー k が見つかったら探索を終了する
- k が key(x) より小さい場合
  - 2分探索木条件より，k は x の右部分木にはない
  - 左部分木に対して探索を続行する
- k が key(x) より大きい場合
  - 右部分木に対して探索を続行する
- 探索する節点は根からのパスになる
  - 実行時間は O(h) (h: 木の高さ)

最小値と最大値

- 最小/最大のキーを持つ要素のポインタを返す
- O(h) 時間

```
TREE_MINIMUM(node *x)
{
   if (left(x) == NIL) return x;
   else return
    TREE_MINIMUM(left(x));
}
```

```
TREE_MAXIMUM(node *x)
{
   if (right(x) == NIL) return x;
   else return
   TREE_MAXIMUM(right(x));
}
```


演習：最小値

- この手続きにより，最小値が得られることを証明せよ
  - ヒント：　木の深さに関する帰納法

```
TREE_MINIMUM(node *x)
{
   if (left(x) == NIL) return x;
   else return
    TREE_MINIMUM(left(x));
}
```

次節点と前節点

- 2分探索木のある節点が与えられたとき，木の中間順 (inorder) で次/前の節点を求める
- O(*h*) 時間

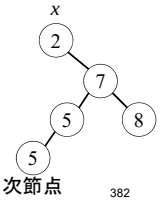
```
TREE_SUCCESSOR(node *x)
{
   node *y;
   if (right(x) != NIL) return TREE_MINIMUM(right(x));
   y = p(x);
   while (y != NIL && x == right(y)) {
      x = y;
      y = p(y);
   }
   return y;
}
```


x が右部分木を持つ場合

- *x* の次節点は，*x* 以上の要素で最小

⇒ *x* の次節点は，*x* の右部分木の最小要素

= TREE_MINIMUM(right(x))



演習：次節点

- ノード *x* が右節点を持たない場合，*x*の次節点 *y* は (存在するなら) *x* の祖先であり，*x* を左部分木に持つことを証明せよ

ヒント：まずyが祖先であることを証明

背理法と場合分け

- *y*が*x*の祖先でないなら，(i) *y*は*x*の子孫となるか，(ii) *x*, *y* は異なる部分木にある．
- 両方の場合で矛盾を示す
- 次に，yがxの祖先であることを仮定して，yがxを右部分木に持つと仮定して矛盾を示す．

演習：次節点

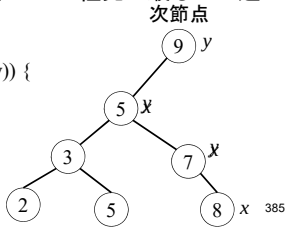
解答：

- 背理法と場合分け
- *y*が*x*の祖先でないなら，(i) *y*は*x*の子孫となるか，(ii) *x*, *y* は異なる部分木にある．
- (i)の場合，*x*は右節点を持たないので，*x*は*y*を左部分木に持つことになり，*y*は*x*の次節点になり得ない．
- (ii)の場合，*x*と*y*の共通の先祖で最も*x*,*y*に近いものを*z*とする．*y*が*x*の次節点であることから，*z*から見て*x*が左で*y*が右にある必要がある．よって，$x \leqq z \leqq y$となるので，*y*が次節点という仮定に反する．
- したがって*y*は*x*の祖先
- この場合，*y*が*x*を右部分木に持つとすると$y \leqq x$となり，*y*が次節点という仮定に矛盾．
- よって*y*は*x*を左部分木に持つ．

x が右部分木を持たない場合

- 次節点は必ず *x* の祖先
- 次節点の候補である *x* の祖先を *y* とする (初期値は*x*の親)
- *x* が，*y*の右の子ならば，*y*は *x* 以下でダメ
- *y* は，*x* を左部分木に持つ *x* の祖先で最も *x* に近いもの

```
y = p(x);
while (y != NIL && x == right(y)) {
   x = y;
   y = p(y);
}
return y;
```



定理 12.2 高さ *h* の2分探索木上の動的集合演算 SEARCH, MAXIMUM, MINIMUM, SUCCESSOR, PREDECESSOR は O(*h*) 時間で実行できる

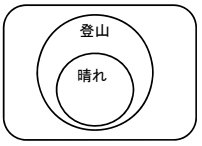
背理法と対偶

- 背理法：ある命題Aを証明するのに，¬Aを仮定して矛盾を導く．
- 対偶：p⇒qを証明するのに，それと論理的に同値な¬q⇒¬pを証明する．
- 全く独立な手法であるが，同時に用いられることもある
  - p⇒qを証明するのに，その否定であるp∧¬qを仮定して矛盾を導く
  - ¬qという前提から¬pがでてくれば矛盾が導ける（実質的には対偶を示してる）

387

対偶

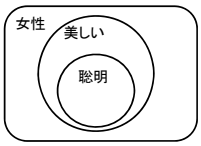
- 論理的には，a⇒bは，¬a∨bと等価
- “明日晴れたら登山に行こう”と言う場合，論理的には，雨のときは登山に行っても行かなくてもどちらでも良い
- a⇒bに対して，¬b⇒¬aを，元の式の対偶と呼ぶ．論理的には等価（のはず）



388

対偶が直感に合わない例

- 推論規則： 聡明な女性は美しい
- 100%正しいかは分からないが，そんなに違和感はない．政治家が言っても問題にはならない
  - 同様な規則：夢を追っている男性はかっこいい
- 対偶を考えると，美しくない女性は聡明ではない＝不細工はバカばっかり --- 違和感がある，政治家が言ったら間違いなく叩かれる



389
389

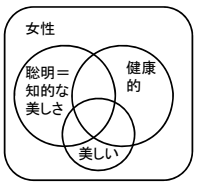
直感に合わない理由 (1)

- 同じことを言うのでも，否定的な言い方は引っかかるのは事実：この治療法で4割の人が助かる vs. この治療法では6割の人が死ぬ
- でも，この例は，二つのことが，同じことを言っているという気がしない

390

直感に合わない理由 (2)

- “美しい”という言葉には多義性があり，かつ，状況によって意味が振れやすい
  - 知的な美しさ，健康的な美しさ，…
- 「聡明な女性は美しい」と言った場合，現実に意味しているのは，「聡明な女性には，知的な美しさがある」ということ．これは当たり前なので誰も文句は言わない
- デフォルトの美しさは，重なりはあるが同じではない
- デフォルトの美しさを満たさないが，聡明な美しさを満たす可能性があるので論理的には間違い



391

論理クイズ

- 凶弾に倒れたケネディの魂が天に登る途中で，白い翼の天使からこんなことを教わった．
- 「あなたはもうすぐ，別れ道につくでしょう．一つの道は天国に，もう一つの道は地獄に行く道です．そこには二人の人が立っています．二人とも天使の姿になっているのであなたには区別できないでしょうが，そこに立っているのはチャーチルとヒットラーです．チャーチルはいつでも本当のことを言いますが，ヒットラーは必ず嘘をつきます．あなたはどちらかの人に一回だけ，yesかnoで答えられる質問ができます．では御成功を祈ります．」
- ケネディはどんな質問をすれば良いか？

392

解答

- 電気情報工学的に考えれば，チャーチルは質問の解答の値をそのまま返す回路で，ヒットラーは質問の解答の値を必ず反転して返す回路．
- これらの回路は二回通せば，どちらも解答の値そのままを返す回路になる．
- 質問の例：「あなたは左の道が天国へ行く道だと聞かれたらyesと答えますか？」
- 別解：これらの回路の両方を一回ずつ通せば，値を反転する回路になる
- 質問の例：もう一人を指さしながら 「あの人に左の道が天国へ行く道だと聞かれたらyesと答えますか?」

論理クイズ：スターリン

- みごとに難関をクリアしたケネディが天国への道を急いでいると，再び別れ道があり，三人の天使の姿をした人が立っていた．白い翼の天使がいうことには，
- 「一つの道は天国に，もう一つの道は地獄に行く道です．そこに立っているのはチャーチルとヒットラーとスターリンです．チャーチルはいつでも本当のことを言いますが，ヒットラーは必ず嘘をつきます．スターリンは本当のことをいうこともあれば，嘘をつくこともあります．あなたは2人の人に対して，それぞれ一回だけ，yesかnoで答えられる質問ができます．では御成功を祈ります．」
- ケネディは途方に暮れた．「もし俺が最初に質問する相手がスターリンだったとしたら，何の情報も得ることができない．どうやって二回で意味のある答えを得れば良いんだ?」

論理クイズ：トランシルバニアの吸血鬼

- トランシルバニアには，人間と見かけでは区別できないが，吸血鬼が暮らしている
- トランシルバニアの正気の人間は真実のみを述べ，狂気の人間は嘘のみを述べる
- 一方，正気の吸血鬼は嘘のみを述べ，狂気の吸血鬼は真実のみを述べる
- ここに，見た目は全く同じである，正気の人間と狂気の吸血鬼がいる
- どちらか片方に，一回だけyesかnoで答えられる質問をすることで，どちらが人間で，どちらが吸血鬼か判別できるか?
- 電気工学的は，どちらも質問の解答をそのまま返す回路で，どんな質問にも同じ答えを返す→区別するのは原理的に不可能?

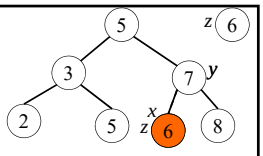
解答

- 電気工学的は，どちらも質問の解答をそのまま返す回路で，どんな質問にも同じ答えを返す→区別するのは原理的に不可能?
- しかし，質問中のパラメータの値が異なるなら，当然，答えは異なる可能性がある
  - 本人に関する質問をすれば良い！
- 実はダイレクトに，「あなたは人間ですか?」と聞けば良い
  - 正気の人間はyes，狂気の吸血鬼はno.

12.3 挿入と削除

- 要素を挿入/削除したあとも2分探索木条件が満たされる必要がある
- 挿入は比較的簡単
- 削除は複雑
- どちらも O(*h*) 時間

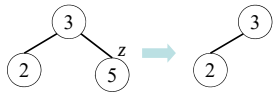
挿入



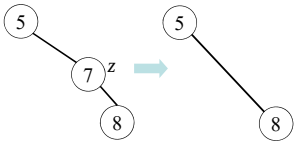

```
TREE_INSERT(node *z)
{
   node *x,*y;
   y = NIL;
   x = root;
   while (x != NIL) {                  // z を挿入する場所 x を決める
      y = x;
      if (key(z) < key(x)) x = left(x);          挿入場所は必ず葉
      else x = right(x);
   }                                   // y は x の親
   p(z) = y;                           // z の親を y にする
   if (y == NIL) root = z;             // T が空なら z が根節点
   else if (key(z) < key(y)) left(y) = z; // y の子を z にする
        else right(y) = z;
}
```

削除

- 探索木から節点 z を削除する
- z が子を持たない場合
  - z の親 $p(z)$ を変更する



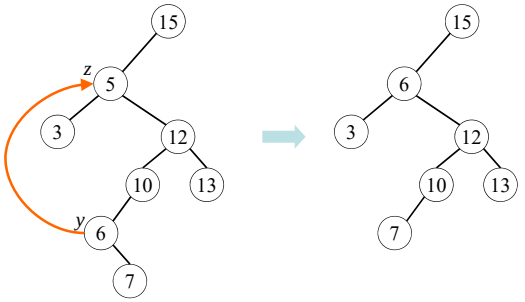
- z が子を1つ持つ場合
  - z の親と z の子を結ぶ



399

削除: z が子を2つ持つ場合

- z の次節点yは左の子を持たない
- z の場所に y を入れ，元の y を削除する



400

```
TREE_DELETE(node *z)
{
   node *x, *y;
   if (left(z) == NIL || right(z) == NIL) y = z;  // z の子の数が1以下
   else y = TREE_SUCCESSOR(z);                    // z は2つの子を持つ
   if (left(y) != NIL) x = left(y);  else x = right(y); //  x は y の子
   if (x != NIL) p(x) = p(y);                     // y を削除する
   if (p(y) == NIL) root = x;                     // y が根なら x を根に
   else if (y = left(p(y))) left(p(y)) = x;       // y の親と子をつなぐ
        else right(p(y)) = x;
   if (y != z) {
      key(z) = key(y);                            // y の内容を z に移動
      // y の付属データを z にコピー
   }
}
```

401

12.4 ランダムに構成された2分探索木

- 2分探索木上の基本操作は O(h) 時間で実行可
- 要素の挿入削除を繰り返すと探索木の高さ h は変化する
- n 個の相異なるキーをランダムな順序で挿入した2分探索木の高さを解析する
- 高さの期待値は O(lg n)
- 厳密な証明は教科書を参照

402

定期試験

- 4〜5問程度
- 現在 作成中
- 場所は工学部第三講義室

403

最低限できて欲しいこと

- 関数の増加のオーダを比較して大小関係を答える
- 電気情報工学科の卒業生として最低限の常識
  - 例: $n, \lg n, n^2, 2^n, \sqrt{n}, n!$
  - 解き方，多分大きいと思われる方で小さいと思われる方を割って，nを無限大にしたら0に収束するかをチェックする．lgをとるのも有効．
  - 例: n^2と2^nを比較，$n^2 / 2^n$ のlgをとると，$2\lg n / n \lg 2$, 明らかにnが無限大なら0に収束

404

最低限できて欲しいこと

- O記法，Ω記法，Θ記法の意味を理解し，様々なアルゴリズムに関して，その計算のオーダを判断できる

405

3.1 漸近記号

Θ-記法

- ある関数 $g(n)$ に対し，$\Theta(g(n))$ は次のような性質を持つ関数の集合と定義する

$\Theta(g(n)) = \{f(n)$:ある正の定数c_1, c_2, n_0 が存在し，
全ての $n \geq n_0$に対して
$0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$ が成立$\}$

$$\Theta(g(n)) = \{ f(n) : \exists c_1 > 0, \exists c_2 > 0, \exists n_0 > 0, \forall n \geq n_0 \quad 0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n) \}$$

- $f(n) = \Theta(g(n))$ は $f(n) \in \Theta(g(n))$ を意味する
- $2n^2+3n+1 = 2n^2 + \Theta(n)$ という表現も用いる

406

限量子の順序

- 順序によって意味が違う

$\forall x \exists y\ loves(x, y)$:
誰にでも愛する人が存在
yはxに依存して異なる値を取ってよい

$\exists y \forall x\ loves(x, y)$:
すべての人から愛されている人が存在
yはxによらない同じ値

407

O-記法

- ある関数 $g(n)$ に対し，$O(g(n))$ は次のような集合と定義する

$O(g(n)) = \{f(n)$:正の定数c, n_0 が存在し，
全ての $n \geq n_0$に対して
$0 \leq f(n) \leq c\, g(n)$ が成立$\}$

$$O(g(n)) = \{ f(n) : \exists c > 0, \exists n_0 > 0, \forall n \geq n_0 \quad 0 \leq f(n) \leq cg(n) \}$$

- $f(n) = \Theta(g(n))$ ならば $f(n) = O(g(n))$
- $n = O(n^2)$ も正しい表現

408

- 挿入ソートの実行時間は$O(n^2)$…正解
- 挿入ソートの実行時間は$\Theta(n^2)$… 間違い
  - ソートされた入力に対しては$\Theta(n)$
- 実行時間が$O(n^2)$であるという場合，最悪実行時間について言っている
  - 実行時間は入力データ数のみでなく，入力データそのものに依存する
  - 最悪実行時間はデータ数 n のみに依存

409

Ω-記法

- ある関数 $g(n)$ に対し，$\Omega(g(n))$ は次のような集合と定義する

$\Omega(g(n)) = \{f(n)$: ある正の定数c, n_0 が存在し，
全ての $n \geq n_0$に対して
$0 \leq c\, g(n) \leq f(n)$ が成立$\}$

$$\Omega(g(n)) = \{ f(n) : \exists c > 0, \exists n_0 > 0\ \forall n \geq n_0 \quad 0 \leq c\, g(n) \leq f(n) \}$$

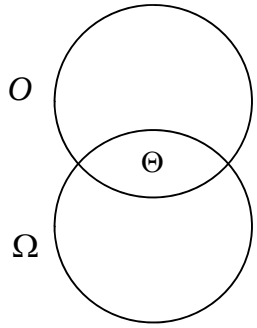
- $f(n)$, $g(n)$ に対して $f(n) = \Theta(g(n))$ であるための必要十分条件は
  $f(n) = O(g(n))$ かつ $f(n) = \Omega(g(n))$

410

- Ω-記法は下界を表す
- 挿入ソートの実行時間が$\Omega(n)$とは
  - このアルゴリズムではどのような入力に対しても少なくともn に比例した時間が必要という意味
  - 最良の実行時間について言っている
- アルゴリズムの実行時間が$\Omega(n^2)$とは
  - どのような入力に対しても少なくとも n^2 に比例する時間がかかるという意味

411

Θ ,O, Ωの関係



412

課題：正しいものを選べ

1. 挿入ソートの実行時間は$O(n^2)$
2. 挿入ソートの実行時間は$O(n)$
3. 挿入ソートの実行時間は$\Omega(n^2)$
4. 挿入ソートの実行時間は$\Omega(n)$
5. 挿入ソートの実行時間は$\Theta(n^2)$
6. 挿入ソートの最悪の実行時間は$\Theta(n^2)$
7. 挿入ソートの最良の実行時間は$\Theta(n)$

1,3が両方とも真なら5は真，逆も成立
6が真なら1は真（逆はダメ）
7が真なら4は真（逆はダメ）
2が真なら1は真（逆はダメ）
3が真なら4は真（逆はダメ）
正解は1, 4, 6, 7

413

最低限できて欲しいこと

- 漸化式を解いて計算のオーダを求める.

414

課題

- 以下の漸化式　$T(n) = T(n/2) + n$
- 帰納法／置き換え法で次の性質を示せ：$T(n) = O(n)$
  - $O(n)$ の定義に従って忠実に証明すること
    - 十分大きなすべてのnについて，ある定数cが存在し，以下が成立する：$T(n) \le c \cdot n$
  - 境界条件は無視して良い
  - nは2の累乗であることを仮定してよい：$n = 2^m$

    mで$T(2^m) \le c \cdot 2^m$が成立すると仮定し，
    $T(2^{m+1}) \le c \cdot 2^{m+1}$を示す.

415

証明

$T(n) = T(n/2) + n$

mで$T(2^m) \le c \cdot 2^m$が成立すると仮定し，
$T(2^{m+1}) \le c(2^{m+1})$を示す.

$$T(2^{m+1}) = c2^m + 2^m = \frac{(c+1)}{2} 2^{m+1} \le c \cdot 2^{m+1}$$

$c \ge 1$で成立

cに関する条件をなるべくタイトに示す!

416

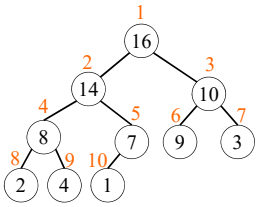
最低限できて欲しいこと

- 様々なソーティングアルゴリズム（挿入ソート，マージソート，ヒープソート，クイックソート，基数ソート等）の内容，動作を理解する

417

6.1 ヒープ

- ヒープ：完全2分木とみなせる配列
- 木の各節点は配列の要素に対応
- 木は最下位レベル以外の全てのレベルの点が完全に詰まっている
- 最下位のレベルは左から順に詰まっている



1	2	3	4	5	6	7	8	9	10
16	14	10	8	7	9	3	2	4	1

418

ヒープ条件 (Heap Property)

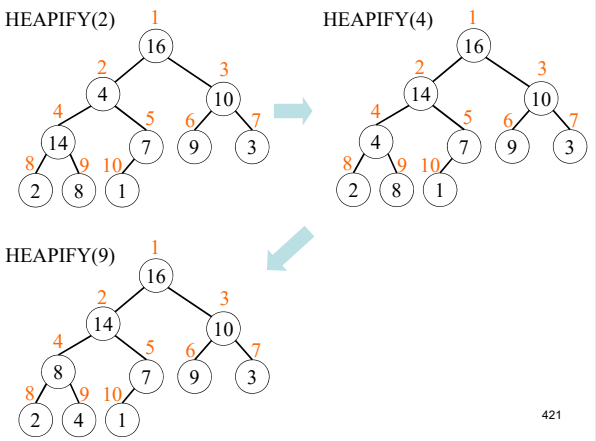
- 根以外の任意の節点 i に対して
  $A[\mathrm{PARENT}(i)] \geq A[i]$
- つまり，節点の値はその親の値以下
- ヒープの最大要素は根に格納される

419

ヒープの操作

- HEAPIFY: ヒープ条件を保持する (根節点が子供より大きいとは限らないが，両側の部分木ではヒープ条件が満たされていることを仮定)．$O(\lg n)$
- BUILD_HEAP: 入力の配列からヒープを構成する．線形時間．
- EXTRACT_MAX: ヒープの最大値を取り除き，残りがヒープ条件を満たすようにする．$O(\lg n)$ 時間．
- HEAPSORT: 配列をソートする．$O(n \lg n)$ 時間．
  - BUILD_HEAPとEXTRACT_MAXから構成される．
- INSERT: ヒープに値を追加する．$O(\lg n)$ 時間．

420



421

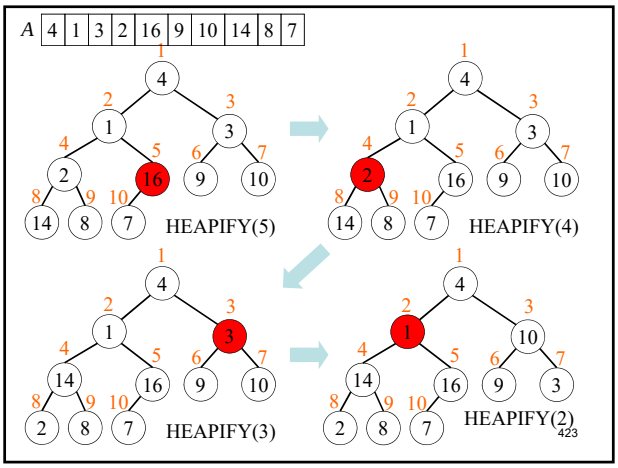
6.3 ヒープの構成

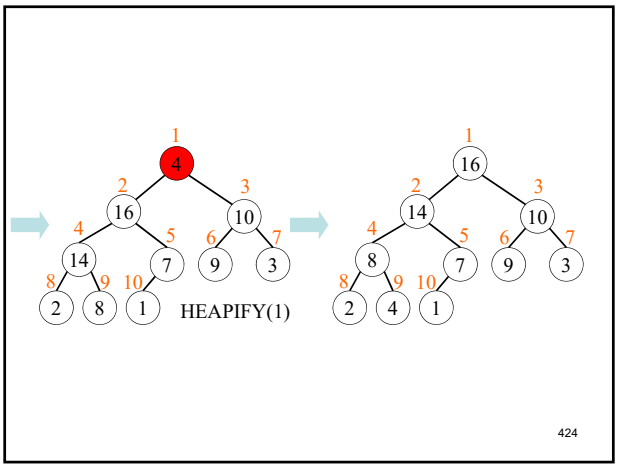
- HEAPIFYでは左右の部分木がヒープである必要がある
- 全体をヒープにするには，2分木の葉の方からヒープにしていけばいい

```
void BUILD_HEAP(int n, data D[])
{
  int i;
  heap_size = n;
  A=D;

  for (i = n/2; i >= 1; i--) {
    HEAPIFY(i);
  }
}
```

422



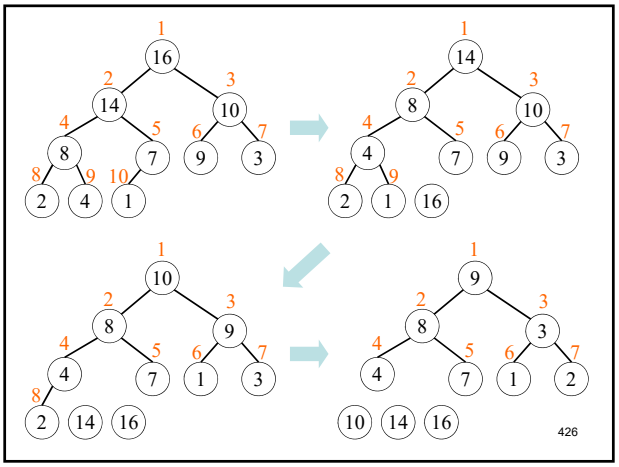


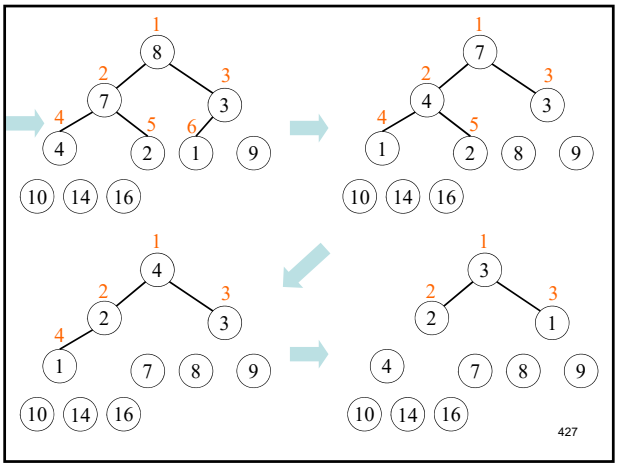
6.4 ヒープソート

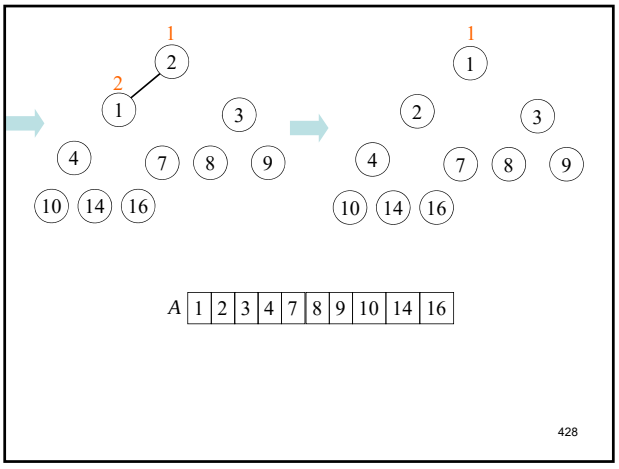
- まずヒープを作る
- すると最大要素が $A[1]$ に入る
- $A[1]$と$A[n]$を交換すると，最大要素が$A[n]$に入る
- ヒープのサイズを1つ減らしてヒープを維持する

```
void HEAPSORT(int n, data D[])
{int i;
 data tmp;
  BUILD_HEAP(n,D);
  for (i = n; i >= 2; i--) {
    tmp = A[1];  A[1] = A[i];  A[i] = tmp;
     // 根と最後の要素を交換
    heap_size = heap_size - 1;
    HEAPIFY(1);
  }}
```

425

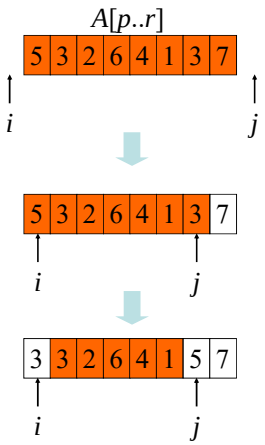






クイックソート

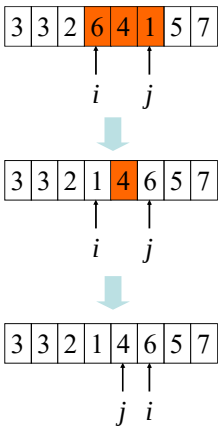
- n 個の数に対して最悪実行時間$\Theta(n^2)$のソーティングアルゴリズム
- 平均実行時間は$\Theta(n \log n)$
- Θ記法に隠された定数も小さい
- in-place (一時的な配列が必要ない)



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?

最低限できて欲しいこと

- キュー，スタック，リスト等のデータ構造を理解し，基本的なアルゴリズムを設計できる

最低限できて欲しいこと

- ハッシュ表の考え方を理解する
- 例： 3, 6, 11, 25, 49, 73のデータが順に到着したとき，ハッシュ関数としてh(x)=x mod 7を用いた場合のハッシュ表の状態を示せ．衝突はチェイン法を用いるものとする．

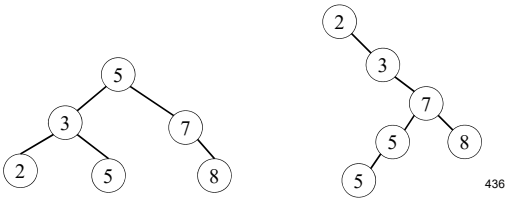
最低限できて欲しいこと

- 2分探索木の構造，基本的な操作について理解する.

435

12.1 2分探索木とは何か?

- 各節点は key, left, right, p フィールドを持つ
- 2分探索木条件 (binary-search-tree property)
  - 節点 *y* が *x* の左部分木に属する⇒ key[*y*]≤key[*x*]
  - 節点 *y* が *x* の右部分木に属する⇒ key[*x*]≤key[*y*]



436

最小値と最大値

- 最小/最大のキーを持つ要素のポインタを返す
- O(*h*) 時間

```
TREE_MINIMUM(node *x)
{
   if (left(x) == NIL) return x;
   else return
    TREE_MINIMUM(left(x));
}
```

```
TREE_MAXIMUM(node *x)
{
   if (right(x) == NIL) return x;
   else return
    TREE_MAXIMUM(right(x));
}
```

437

次節点と前節点

- 2分探索木のある節点が与えられたとき，木の中間順 (inorder) で次/前の節点を求める
- O(*h*) 時間

```
TREE_SUCCESSOR(node *x)
{
   node *y;
   if (right(x) != NIL) return TREE_MINIMUM(right(x));
   y = p(x);
   while (y != NIL && x == right(y)) {
      x = y;
      y = p(y);
   }
   return y;
}
```

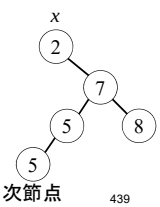
438

x が右部分木を持つ場合

- *x* の次節点は，*x* 以上の要素で最小

⇒ *x* の次節点は，*x* の右部分木の最小要素

= TREE_MINIMUM(right(x))

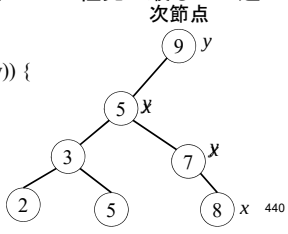


439

x が右部分木を持たない場合

- 次節点は必ず *x* の祖先
- 次節点の候補である *x* の祖先を *y* とする (初期値は*x*の親)
- *x* が，*y*の右の子ならば，*y*は *x* 以下でダメ
- *y* は，*x* を左部分木に持つ *x* の祖先で最も *x* に近いもの

```
y = p(x);
while (y != NIL && x == right(y)) {
   x = y;
   y = p(y);
}
return y;
```



440