アルゴリズム論（第6回）

2003.11.17
櫻井　彰人
algorithm@soft.ae.keio.ac.jp
http://www.sakurai.comp.ae.keio.ac.jp/

きょうの講義概要

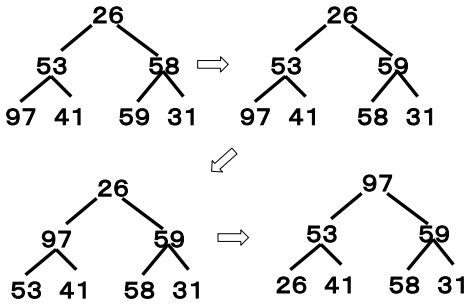
- ヒープソート（2）
- 整列のまとめ
- 探索（1）
  - 探索のためのデータ構造（配列、リスト、表、キュー、スタック、木、2分木）
  - 2分木の巡回
- 原理、アルゴリズム、計算量、正当性、実際問題での利用
- 今年度の研究室配属について

ヒープソートアルゴリズム

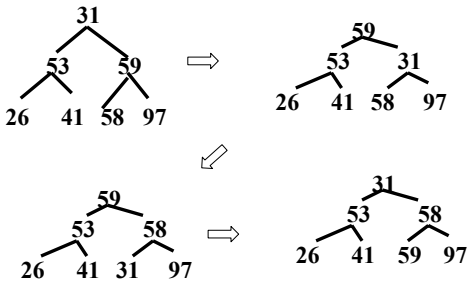
（1）与えられたn個のデータから木（実際には配列）を作り、葉に近い部分木からヒープ再構成アルゴリズムを適用して木全体をヒープに

（2）根と葉の最後の要素を入れ換え、対象とするデータの個数を一つずつ減らしながら、対象になるデータが2個になるまでヒープ再構成アルゴリズムを繰り返して適用する

ヒープを作る



ソートをする



アルゴリズム3.5 ヒープソート

- 関数heap(1,n)
- 入力と出力は配列a[1,2,3,…,n]に
- 内容
  - （1）n=1なら、何もしないで終了
  - （2）i = div(n,2) から1まで、arrangeheap(i,n) を繰り返す
  - （3）i = n から2まで以下を繰り返す
    - » (3.1) 1番目の要素と i 番目の要素とを入れ替える
    - » (3.2) arrangeheap(1,i-1)
- div(n,2) :nが偶数のときn/2、奇数のとき(n-1)/2

ヒープソートの実行

はじめのデータにheap(1,n)を

1	2	3	4	5	6	7
26	53	58	97	41	59	31

（2）でarrangeheap(3,7)，a-heap(2,7)，a-heap(1,7)を

1	2	3	4	5	6	7
97	53	59	26	41	58	31

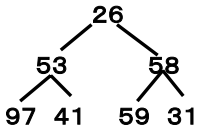
(3.1) を実行すると

1	2	3	4	5	6	7
31	53	59	26	41	58	97

ヒープソートの実行（2）

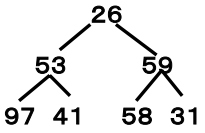
はじめのデータ

1	2	3	4	5	6	7
26	53	58	97	41	59	31



arrangeheap(3,7)

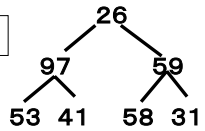
1	2	3	4	5	6	7
26	53	59	97	41	58	31
		k				r



ヒープソートの実行（3）

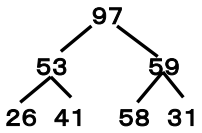
arrangeheap(2,7)

1	2	3	4	5	6	7
26	97	59	53	41	58	31
	k					r



arrangeheap(1,7)

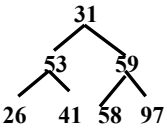
1	2	3	4	5	6	7
97	53	59	26	41	58	31
k						r



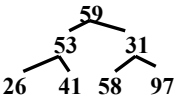
ヒープソートの実行（4）

１と７の要素を入れ換えa-heap(1,6)

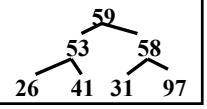
1	2	3	4	5	6	7
31	53	59	26	41	58	97
k					r	



1	2	3	4	5	6	7
59	53	31	26	41	58	97



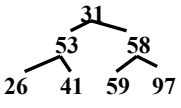
1	2	3	4	5	6	7
59	53	58	26	41	31	97



ヒープソートの実行（5）

１と６の要素を入れ換えa-heap(1,5)

1	2	3	4	5	6	7
31	53	58	26	41	59	97
k				r		



１と５を入れ替えてa-heap(1,4)，。。。。

1	2	3	4	5	6	7
26	31	41	53	58	59	97

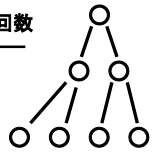
ヒープソートの演習

- ヒープソートを使って、3、8、6、4、7、5、1、2を小さい順に並べ替えなさい
  - プロセスが分かるように
  - 木の形でも表現

ヒープソートの計算量（1）

■（1）与えられたデータ全体で最初にヒープを作る ＋ （2）根のデータを最後の節と入れ替えながらヒープの再構成を繰り返す
- 説明は、データ（節）の数$n = 2^k - 1$（完全2分木）のケースで

節	個数	ヒープ再構成の最大比較回数
高さ0（葉）	$(n+1)/2$	0回 ＝ 2x0 回
高さ1	$(n+1)/2^2$	2回 ＝ 2x1 回
高さ2	$(n+1)/2^3$	4回 ＝ 2x2 回
高さ3	$(n+1)/2^4$	6回 ＝ 2x3 回
高さ4	$(n+1)/2^5$	8回 ＝ 2x4 回



ヒープソートの計算量（2）

◆（1）与えられたデータ全体で最初にヒープを作るときの比較回数（つづき）
- 上限は 2(n+1)

$$2x0 \times ((n+1)/2^1) + 2x1 \times ((n+1)/2^2) + 2x2 \times ((n+1)/2^3) + 2x3 \times ((n+1)/2^4) + \cdots\cdots\cdots$$
$$= (n+1)\{0 + 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + 5/2^5 + \cdots\cdots\}$$
$$= 2(n+1)$$

$$1/(1-x) = 1 + x + x^2 + x^3 + \cdots$$
$$1/(1-x)^2 = 0 + 1 + 2x + 3x^2 \cdots$$
$$x/(1-x)^2 = 0 + x + 2x^2 + 3x^3 \cdots$$

ヒープソートの計算量（3）

■（2）根のデータを最後の節と入れ替えながらヒープの再構成を繰り返す
- ■一度ヒープができれば、根と最後の節を入れ替えたあとで1度再構成するときの比較回数の最大値は
  - ■（節の高さ） × 2
  - ■ 一番高い節では、$\{\log_2 (n+1) - 1\} \times 2$
    - ■ このレベルで、$((n+1)/2) - 1$ 回の繰り返し
    - ■ $[\{\log_2 (n+1) - 1\} \times 2] \times \{((n+1)/4) - 1\}$
  - ■対象とするヒープの高さが一つ減った節では、$\{\log_2 (n+1) - 2\} \times 2$
    - ■ このレベルで、$(n+1)/4$ 回の繰り返し
    - ■ $[\{\log_2 (n+1) - 2\} \times 2] \times \{(n+1)/8\}$

ヒープソートの計算量（4）

◆（2）根のデータを最後の節と入れ替えながらヒープの再構成を繰り返す（つづき）
- 以上の操作の比較回数の上限（計算のため、一番高い節での繰り返し回数を(n+1)/4とおく）
  - » $2(n+1)\log_2 (n+1) - 4(n+1)$

$$\{\log_2 (n+1) -1\} \times 2 \times (n+1)/4 + \{\log_2 (n+1) -2\} \times 2 \times (n+1)/8 + \{\log_2 (n+1) -3\} \times 2 \times (n+1)/16 + \cdots\cdots$$
$$= (n+1)\log_2 (n+1)\{1 + 1/2 + 1/2^2 + 1/2^3 + 1/2^4 + \cdots\cdots\}$$
$$- (n+1)\{1 + 1/2 + 2/2^2 + 3/2^3 + 4/2^4 + \cdots\cdots\}$$

◆（1） ＋ （2）
- » $2n \log_2 (n+1)$
- » $\Theta(n \log_2 n)$

3.6 ソートアルゴリズムのまとめ

アルゴリズム名	計算量	実装の難度
最小（大）値法	$\Theta(n^2)$	最も簡単
挿入法	$\Theta(n) \sim \Theta(n^2)$	簡単
シェル	$\Theta(n(\log n)^2)$	やや簡単
クイック	$\Theta(n \log n) \sim \Theta(n^2)$	やや難
ヒープ	$\Theta(n \log n)$	難

4 探索

■ 探索（Search）
- 系統的にまたは試行錯誤して解を探す
  - » 解：初期状態から目標状態へ
- 蓄えられた情報を効率よく探し出す
  - » 情報：キーのついたレコード
    - ◆ 名前＋電話番号、英単語＋説明、キーワード＋ポインタ、その他
- 評価：見つかるまでの計算量（cf. 2分探索）

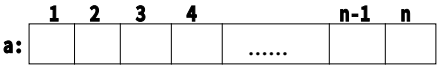
探索アルゴリズム

- **解の探索**（Winston, P. H.: Artificial Intelligence, Addison-Wesley, Pub., 1984, p. 88より）
  - とにかく解を
    - » 深さ優先探索、山登り法、幅優先探索、Beam、Best-first
  - 最適解
    - » 分岐限定法、ダイナミックプログラミング、A*アルゴリズム
  - ゲーム
    - » ミニマックス法、α--β 法、ヒューリスティックな枝がり、

4.1 探索のためのデータ構造

■ 配列（array）

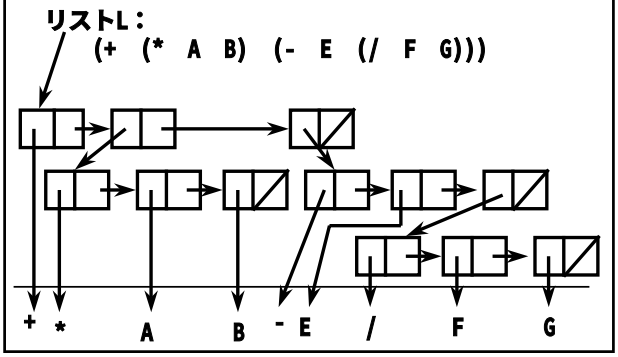
– 同一タイプのデータが1次元または2次元に並んだもの



■リスト（list）

–データの組が一定の順序で集まったもの（入れ子構造を許す）

探索のためのデータ構造（2）



探索のためのデータ構造（3）

■表（table）

–複数個の欄（field）で１つのレコード（record）

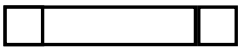
–各欄に名前がつき、レコードごとに欄に対応する値

–複数個のレコードの集合が表

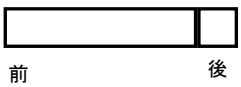
	欄1	欄2	欄3	…	欄n
1					
2					
…					
m					

探索のためのデータ構造（4）

- キュー（queue、待ち行列）：先入れ先出し
  - FIFO: First In First Out



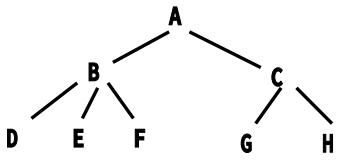
- スタック（stack、棚）：後入れ先出し
  - LIFO: Last In First Out, FILO: First In Last Out



探索のためのデータ構造（5）

■ 木（tree）

– 階層的な構造で情報を表現

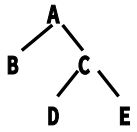


4.3 2分木の巡回（1）

- 2分木の節に蓄えられた情報の探索（2分木の再帰的巡回アルゴリズム）
  - Symmetric Order (In Order)
    - »（1）左の部分木をSymmetric Order で巡回
    - »（2）根を巡回
    - »（3）右の部分木を Symmetric Order で巡回
  - Preorder
    - »（1）根を巡回
    - »（2）左の部分木を Preorder で巡回
    - »（3）右の部分木を Preorder で巡回

2分木の巡回（2）

- 2分木の巡回（つづき）
  - Postorder
    - »（1）左の部分木を Postorder で巡回
    - »（2）右の部分木を Postorder で巡回
    - »（3）根を巡回



S-Order: B, A, D, C, E
Pre: A, B, C, D, E
Post: B, D, E, C, A

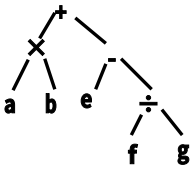
アルゴリズム4.1(Postorder)

- 入力：2分木（節ごとに子供の節へのポインタ）
- 出力：Postorderでの巡回結果
  - （1）根の節をスタックに入れる（プッシュダウン）
  - （2）スタックが空なら終了
  - （3）スタックの一番上にある節pに調べるべき子供の節がなければ、pをスタックから取り出し（ポップアップ）、出力エリアに送って（2）へ
  - （4）pに調べていない子供の節cがあれば、このcをプッシュダウンして（3）へ

数式の木の巡回とポーランド記法

- 数式：根を演算子、左右の部分木を被演算子とした2分木
  - Preorder がポーランド記法（前置記法）、Postorder が逆ポーランド記法（後置記法）に対応。括弧不要
  - In-order が中置記法に相当。括弧が必要。×と÷の方が＋と－より優先度が高いという順位を使って括弧の数を削減

例：a×b + (e - f÷g)
（1）Preorderによる巡回
+ × a b – e ÷ f g
（2）Postorederによる巡回
a b × e f g ÷ - +



ポーランド記法と英語
逆ポーランド記法と日本語

例：a×b + (e - f÷g)
(1) Preorder + × a b – e ÷ f g
sum of “product of a and b” and “difference of e and division of f by g”
(2) Postoreder a b × e f g ÷ - +
a と b との積と e と f と g の商の差の和
(a と b との積)と (e と f と g の商の差)の和

アルゴリズム4.2（逆ポーランド記法）

- 入力：カッコと単項演算子を含まない数式
- 出力：逆ポーランド記法の文字列
  - （1）i ←1
  - （2）i番目の入力を調べる
    - » (2.1) 変数なら出力エリアに送り、i を1増やして（2）へ
    - » (2.2) 入力の終了であれば、スタックにある演算子をポップアップしながら出力エリアに送って終了
    - » (2.3) 演算子のときは、
      - (2.3.1) スタックが空またはスタックの一番上にある演算子より優先度が高ければプッシュダウンし、i を1増やして（2）へ
      - (2.3.2) スタックの一番上の演算子の優先度より低いか同じときは、ポップアップして出力エリアに送り、この条件が成り立つ間この操作を繰り返して(2.3.1)へ

演習問題

- アルゴリズム4.1にならって、Symmetric OrderとPreorderのアルゴリズムを記述しなさい。
- 数式にカッコが使えたり、単項演算子を使えるように、アルゴリズム4.2を拡張しなさい。