アルゴリズムとデータ構造入門
ソーティング・サーチ
Sorting and Hashing

16年前の17日
阪神・淡路大震災
犠牲者死者だけで6434人
天災は忘れた頃にやってくる （寺田寅彦）
天才は忘れた頃にやってくる （奥乃博）

奥 乃 博
大学院情報学研究科知能情報学専攻
知能メディア講座 音声メディア分野
http://winnie.kuis.kyoto-u.ac.jp/~okuno/Lecture/11/IntroAlgDs/
okuno@i.kyoto-u.ac.jp

試験は1月24日（火）3限 2号館101・102

1月17日・本日のメニュー

1. Internal Sorting（内部整列）
   1. 挿入ソート（insertion sort）
   2. クイックソート（quick sort）
2. vector（ベクタ）によるSorting
   1. ヒープソート（heap sort）
   2. バブルソート（bubble sort）
   3. マージソート（merge sort）
3. Searching（探索）
   1. 二分探索（binary search）
   2. ハッシュ法（hashing）

バブルソート（bubble sort）

```
(define (bubble-sort records . args)
  (let ((pred (if (null? args) >= (car args)))
        (size (vector-length records)) )
    (do ((i 0 (+ i 1)))
        ((>= i size) records)
      (do ((j (- size 1) (- j 1))
           (data nil) )
          ((<= j i))
        (set! data (vector-ref records j))
        (cond ((pred data
                 (vector-ref records (- j 1)) )
              (vector-set! records j
                (vector-ref records (- j 1)) )
              (vector-set! records (- j 1)
                data) ))))))
```


バブルソート（bubble sort）実行トレース

9 1 8 2 5 3 0 7 4

9 8 1 7 2 5 3 0

9 8 7 1 5 2 4 3 0

9 8 7 1 5 2 4 3 0

9 8 7 5 1 4 2 3 0

9 8 7 5 4 1 3 2 0

9 8 7 5 4 3 1 2 0

9 8 7 5 4 3 2 1 0

バブルソート(bubble sort)の計算量

1回ごとに敷居(|)が1つずつ減る

$$\sum_{i=1}^{n}(i-1)=\frac{1}{2}n(n-1)$$

$$\Theta(n^2)$$

シェルソート(Shell sort)

- バブルソート: 隣接データを比較
  - $h=1$
- 飛び飛び(h)に比較
  - $h_k = 3h_{k-1} + 1, \ldots, 1$ の時
  - e.g., 40, 13, 4, 1

$$\Theta(n^{1.25})$$

挿入ソート(insert sort)の計算量

1. 最悪の場合(*worst case*) (predが≧とする)
   - 小さいものから順に入ってくる
   - 比較回数は $\sum_{i=1}^{n}(i-1)=\frac{1}{2}n(n-1)$ $\Theta(n^2)$
2. 最良の場合(*best case*)
   - 大きなものから順に入ってくる
   - 比較回数は $\sum_{i=2}^{n}1=(n-1)$ $\Theta(n)$
3. 平均の場合(*average case*)
   - すでに入っているデータの半分だけ比較
   - 比較回数は $\sum_{i=1}^{n}\frac{1}{2}(i-1)=\frac{1}{4}n(n-1)$ $\Theta(n^2)$

マージソート(併合ソート、merge sort)

- ソート済みのデータを前からマージ(併合)
- リスト版
- ベクタ版
- 計算量は両方のデータのスキャンのみ
- m 個のデータと n 個のデータとすると

$$\Theta(m+n)$$

- 空間計算量も余分に $\Theta(m+n)$

1月17日・本日のメニュー

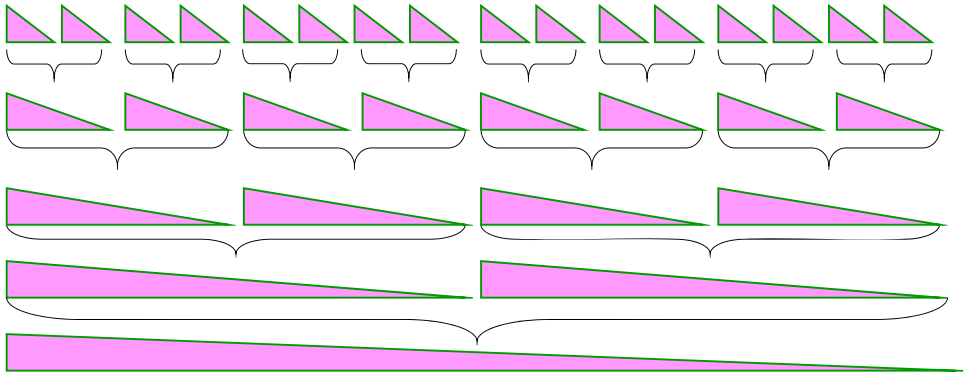
NoStudent Left Behind

1. Internal Sorting（内部整列）
   1. 挿入ソート（insertion sort）
   2. クイックソート（quick sort）
2. vector（ベクタ）によるSorting
   1. ヒープソート（heap sort）
   2. バブルソート（bubble sort）
   3. **マージソート（merge sort）**
3. **Searching（探索）**
   1. 二分探索（binary search）
   2. ハッシュ法（hashing）

内部マージソート（In-place merge sort）

NoStudent Left Behind

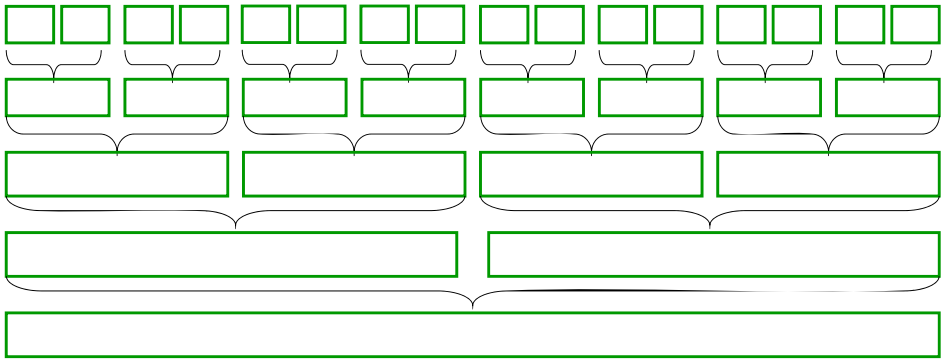
分割統治型



ラン（run）と言う　$\Theta(n \log n)$

内部マージソート（In-place merge sort）

分割統治型（ベクタの場合）



$\Theta(n \log n)$

NoStudent Left Behind

Sorting（整列）のまとめ

NoStudent Left Behind

- 選択ソート　(selection sort)　$\Theta(n^2)$
- 挿入ソート (insertion sort)　定$\Theta(n^2)$
- シェルソート (Shell sort)　$\Theta(n^{1.25})$
  - $h_k = 3h_{k-1} + 1, \ldots, 1$　の時
- クイックソート (quick sort),分割統治法(divide and conquer)
  - 平均　$\Theta(n \log n)$　最悪　$\Theta(n^2)$
- ヒープソート (heap sort)　常時　$\Theta(n \log n)$
- マージソート (merge sort) 常時　$\Theta(n \log n)$

1月17日・本日のメニュー

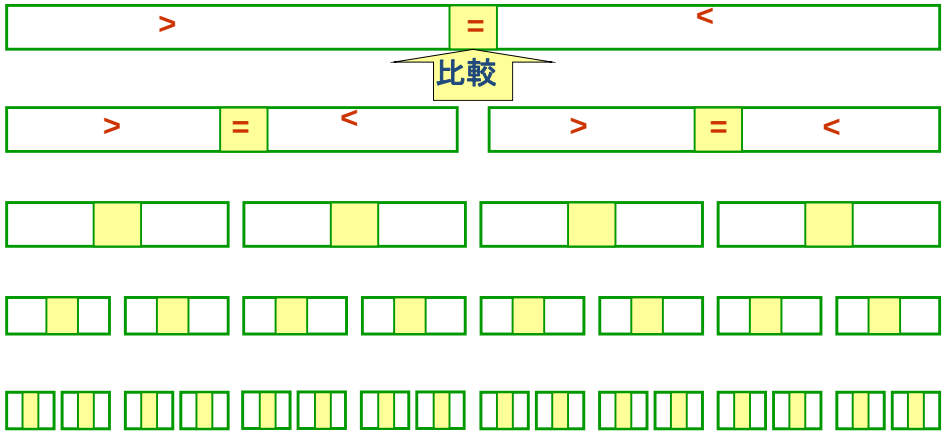
NoStudent Left Behind

1. Internal Sorting（内部整列）
   1. 挿入ソート（insertion sort）
   2. クイックソート（quick sort）
2. vector（ベクタ）によるSorting
   1. ヒープソート（heap sort）
   2. バブルソート（bubble sort）
   3. マージソート（merge sort）
3. **Searching（探索）**
   1. **二分探索（binary search）**
   2. **ハッシュ法（hashing）**

Sort（整列）済ベクタの探索

NoStudent Left Behind

二分探索（binary search）



二分探索の計算量 $\Theta(\log n)$

1月17日・本日のメニュー

NoStudent Left Behind

1. Internal Sorting（内部整列）
   1. 挿入ソート（insertion sort）
   2. クイックソート（quick sort）
2. vector（ベクタ）によるSorting
   1. ヒープソート（heap sort）
   2. バブルソート（bubble sort）
   3. マージソート（merge sort）
3. Searching（探索）
   1. 二分探索（binary search）
   2. **ハッシュ法（hashing）**

ハッシュ法（hashing）

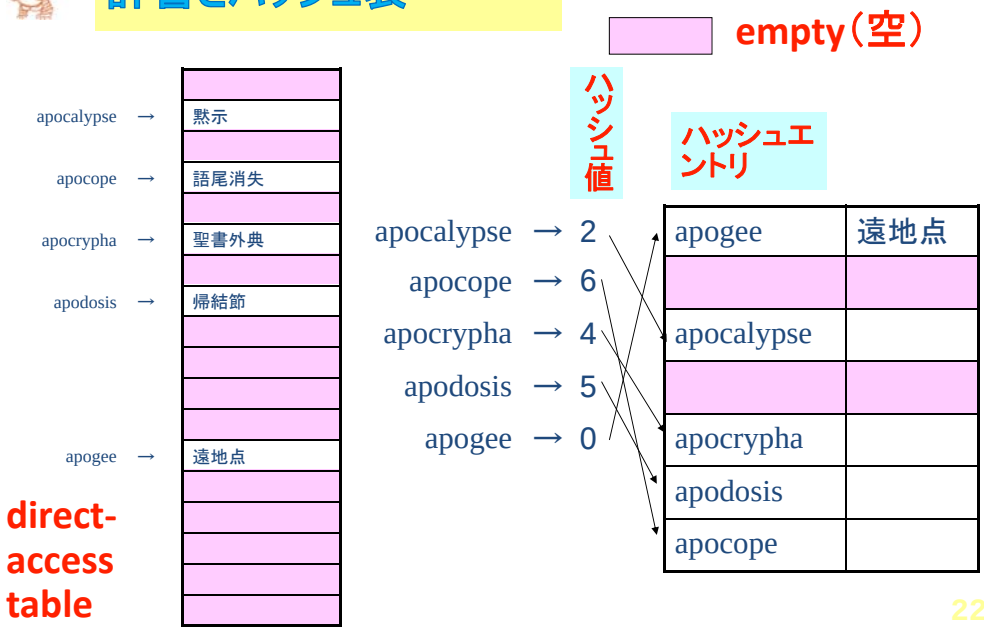
NoStudent Left Behind

- 探したいデータの範囲膨大
- 例： 最大10文字の単語
- 50文字とすると組合わせの数は 50^{10}

$$\log(50^{10}) = 10(2 - \log 2) \cong 17 \qquad 10^{17}$$

- ところが実際の単語数は高々 10^{6}
- ベクタ（配列）で単語を管理すると疎
- 　すかすかの配列
- ハッシュ法（hashing）を使用

辞書とハッシュ表



ハヤシライスを知っていますか

- hashed beaf
- 語源は同じ

ハッシュ法（hashing）

- キーの値の探索なしにアクセス
- ハッシュ関数（hash function）
- キー ⇒ ハッシュ値（整数）
- ハッシュ表（hash table）、サイズ *M*
- 占有率（load factor）α、データ量 *N*

$$\alpha = N/M$$

- 異なるキーに対してハッシュ値が同じ
- ハッシュ値の衝突（collision）

ハッシュ関数（hash function）

- 設計の指針： ランダム性を有するもの。
- キー： $x = a_1 a_2 \cdots a_n$ $key(x) = m$
- 例1：キーから $h_1(x) \equiv m \mod M$
- 例2： 文字列から整数への写像

$$h_2(x) \equiv \sum_{i=1}^{n} code(a_i) \mod M$$

- 例3： m^2の中央部分の log$M$ 桁分を使用

$$h_3(x) \equiv \lfloor m^2/K \rfloor \mod M$$

where constant K such that $MK^2 \cong N^2$

ハッシュ法の基本手続き

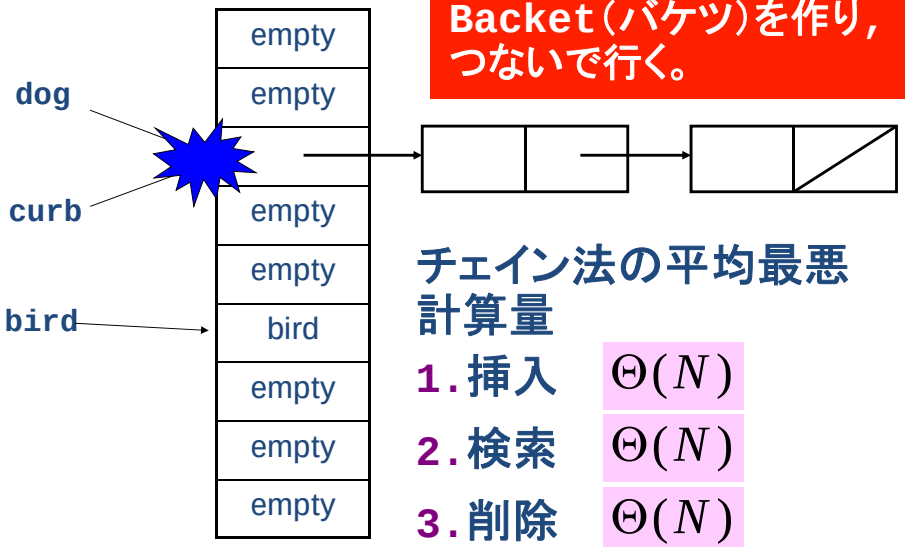
- **挿入（insert）**
  hash表にkeyを持つデータを挿入
- **検索（member）**
  hash表からkeyでデータを検索
- **削除（delete）**
  hash表からkeyを持つデータを削除

ハッシュ値衝突（collision）対処法

- **チェイン法**（chaining, separate chaining, 連鎖法、内部ハッシュ法）
- **開番地法**（open addressing, オープン法、外部ハッシュ法）
  1. **線形走査法**（linear probing）
  2. **万能ハッシュ法**（universal hashing）
  3. **２重ハッシュ法**（double hashing）
     h, gとすると、
     $h(x), h(x)+g(x), h(x)+2g(x), h(x)+3g(x), \ldots$

チェイン法

ハッシュ値の衝突
Backet（バケツ）を作り，つないで行く。



チェイン法の平均最悪計算量

1. 挿入 $\Theta(N)$
2. 検索 $\Theta(N)$
3. 削除 $\Theta(N)$

内部ハッシュ法（internal hashing）

- ハッシュ関数 h_i
- 占有率 α
- エントリに**状態**を導入
  empty/deleted/key（データ）

$$\alpha = N/M \quad <1$$

1. 挿入： empty/deleted というフラグのあるエントリに入れる
2. 検索：
   emptyまで探す
3. 削除：
   deletedというフラグを立てる。

empty	
empty	
empty	
empty	
deleted	
empty	
empty	
キー1	データ1
empty	
empty	
empty	
キー2	データ2
empty	
empty	
empty	
empty	

線形走査法(linear probing)

- ハッシュ関数 h
- 衝突発生時
- $h_i \equiv h+i \bmod M$
- 挿入
  - empty/deleted というフラグのあるエントリに入れる
- 検索
  - emptyまで探す。
- 削除
  - deletedというフラグを立てる。

empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	

万能ハッシュ法(universal hashing)

- ハッシュ関数 $h, \cdots$
- *ハッシュ関数をランダムに選択*
- 挿入
  - empty/deleted というフラグのあるエントリに入れる
- 検索
  - emptyまで探す。
- 削除
  - deletedというフラグを立てる。

empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	

2重ハッシュ法(double hashing)

- ハッシュ関数 h, g
- 衝突発生時
  $h_i \equiv h+ig \bmod M$
- 挿入
  - empty/deleted というフラグのあるエントリに入れる
- 検索
  - empty/deletedまで探す。
- 削除
  - deletedというフラグを立てる。

empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	
empty	

線形走査法(linear probing)の例

h(dog)=2

h(Kyoto)=4

h(Univ)=0

h(Informatics)=2

h(SICP)=3

h(test)=8

empty	
empty	
empty	
empty	

線形走査法の挿入の計算量

1. n個のデータが格納済、$n+1$個目のデータ挿入時に$h_i(x)$が$i-1$回目まですべて詰まっている確率

$$\frac{n}{M}\frac{n-1}{M-1}\frac{n-2}{M-2}\cdots\frac{n-i+1}{M-i+1}$$

2. 空きセルを見つけるまでの比較回数

$$1+\sum_{i=1}^{M-1}\frac{n(n-1)\cdots(n-i+1)}{M(M-1)\cdots(M-i+1)}\cong 1+\sum_{i=1}^{\infty}\left(\frac{n}{M}\right)^i=\frac{M}{M-n}$$

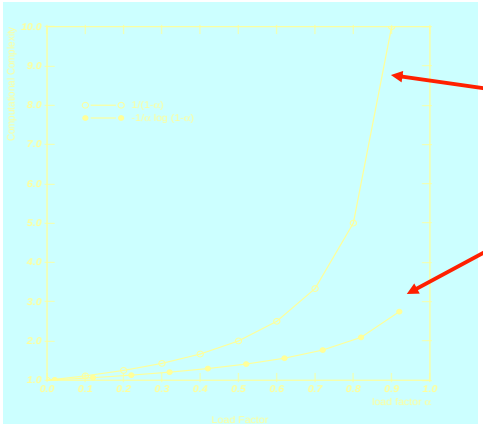
3. ハッシュ表にN個のデータを挿入する手間は

$$\sum_{n=0}^{N}\frac{M}{M-n}\cong\int_0^N\frac{M}{M-x}dx=M\log_e\frac{M}{M-N+1}$$

線形走査法の挿入の計算量（続）

4. 1回あたりの平均の挿入の手間は

$$\frac{M}{N}\log_e\frac{M}{M-N+1}\cong-\frac{1}{\alpha}\log_e(1-\alpha)$$



$$\frac{1}{1-\alpha}$$

$$-\frac{1}{\alpha}\log_e(1-\alpha)$$

線形走査法の検索の計算量

1. deleted　はないものと仮定
2. 表にキーがない時は、$n=N$ の挿入と同じ

$$\frac{M}{M-N}=\frac{1}{1-\alpha}$$

3. 表にキーがある時

$$\frac{M}{N}\log_e\frac{M}{M-N+1}\cong-\frac{1}{\alpha}\log_e(1-\alpha)$$

4. 削除も検索と同じ
5. 上記の解析は、一様ハッシュ（uniform hashing）を仮定：　キーの探索列ランダム

線形走査法の削除の計算量

1. deleted　はないものと仮定
2. 表にキーがない時は、$n=N$ の挿入と同じ

$$\frac{M}{M-N}=\frac{1}{1-\alpha}$$

3. 表にキーがある時

$$\frac{M}{N}\log_e\frac{M}{M-N+1}\cong\frac{1}{\alpha}\log_e(1-\alpha)$$

期末テスト，必修課題，随意課題

- 期末テスト、健闘を期待します。
- 必修課題2の締切は2月11日13時。
- 随意課題の提出でランクアップを。

期末テスト：1月24日（火）3限　2号館　101・201

随意課題：2月22日午後24時締切

挑戦的な作品を期待します
- 図形，フラクタル，音楽，Lego
- アルゴリズム可視化，

プログラムはメイルで

okuno@i.kyoto-u.ac.jp