情報科学基礎論： 「データ構造とマッチングアルゴリズム」

知能情報学専攻 奥乃 博 (okuno@i.kyoto-u.ac.jp)

OHP： http://winnie.kuis.kyoto-u.ac.jp/ ∼ okuno

/Lecture/04/DataStructure/informatics.html

1. 各種データ構造
   - 基本データ構造
   - 複雑なデータ構造
2. 探索アルゴリズム
   - 整列 (sorting) と探索 (searching)
   - マッチングアルゴリズム (matching algorithm)
3. レポート問題

アルゴリズム (Algorithm) とデータ構造 (Data Structure)

- 入力 $\longrightarrow$ **アルゴリズム**
有限回の計算ステップ $\longrightarrow$ 出力

- データ構造 — データの有限の集合, しかも, 動的に要素の数が変化する集合を扱うための表現方法
- オブジェクトベースド — (Ada)
データ構造と演算のカプセル化
- クラスベースド — インスタンスとクラスの区別 (CLU)
- オブジェクト指向 — クラス間にインヘリタンス (階層構造)

基本データ構造

1. リスト (list), 配列 (array), cyclic list, sentinel
2. 待ち行列 (キュー, queue) — FIFO (First-In First-Out)
3. スタック (stack) — LIFO (Last-In First-Out)
4. 木 (tree)
   - 頂点 (vertex), 節点 (node),
     枝 (edge, branch), 辺 (arc)
   - 根付き木 (rooted tree) — 根 (root), 葉 (leaf)
   - 節の深さ (depth) — 根からその節までの節数 (含: その節)
   - 木の高さ (height) — 最大の深さ

木 (Tree)

1. 有向木 (ordered tree) — 親 (parent), 子 (child), 兄弟 (sibling), 祖先 (ancestor), 孫 (descendent)
2. 順序木 (ordered tree) — 根付き木で子 (兄弟) が順序付け
3. 木の走査 (traversal) — 節点, 枝を訪問する順序
   - 前順走査 (pre-order traversal) 最初
   - 間順走査 (in-order traversal) 2 分木の場合, 左枝の次
   - 後順走査 (post-order traversal) 最後

探索 (Searching)

- **線形探索 (Linear search)**

  探索 $O(n)$**, 挿入削除** $O(n)$
- **2 分探索 (Binary search)**

  探索 $O(\log n)$**, 挿入削除** $O(n)$
- **2 分探索木 (Binary search tree)**
  - **最良: 完全 2 分木 (complete binary search tree)**

    探索, 挿入削除 $O(\log n)$
  - **最悪ケース:** $O(n)$
  - **平均:** $O(\log n)$
- O **記法 (上限),** Ω **記法 (下限),** Θ **記法 (上下限)**

平衡木

- **AVL 木 — 2 分木**

  どの頂点においても，その左部分木の高さと右部分木の高さの差が高々１.

  木の頂点数の増加に対して，木の高さの増加は対数オーダ

- **B 木 — 多分木 (Multiway tree, m-ary tree)**

  1. 各頂点 (葉以外) の子の数の最大は m

  2. 各頂点 (葉以外) の子の数の最小は $\lceil \frac{m}{2} \rceil$

  3. 根から葉までの深さはどの葉についても同じ.

  ファイルを用いた検索. ディスクアクセスを極力減らす.

  “2-3 木” (枝が 2 か 3), “2-3-4 木” (枝が 2 〜 4),

ハッシュ法 (hashing)

キーの値の探索なしにアクセス, e.g. 配列

ハッシュ関数 (hash function), ハッシュ表 (hash table)

占有率 (load factor): データサイズ / 配列サイズ

ハッシュ関数の衝突 (collision) への対処法

- チェイン法 (chaining, separate chaining 連鎖法)
- 開番地法 (open addressing, オープン法): 空地使用.
  1. 線形走査法 (linear probing)
  2. 均一ハッシュ法 (uniform hashing) 複数ハッシュ関数
  3. ２重ハッシュ法 (double hasing) — h, g

$$h(x), h(x)+g(x), h(x)+2g(x), \cdots$$

トライ (trie) ・ パトリシア木 (Patricia tree)

- トライ — キーが複数桁で表現 (例: 文字列)

  根からの分岐 — 最上位桁, 次の節点の分岐 — 次の桁

  探索, 挿入, 削除: キーの長さに依存. ×データの個数

- パトリシア木

  トライから無駄な桁のチェックを省く.

  節点にはビット位置を記述, 枝は 0 か 1

例 0001, 0011, 1010, 1100, 1101

整列 (Sorting) アルゴリズム

- **選択ソート (selection sort)** — $O(n^2)$
- **挿入ソート (insertion sort)** — $O(n^2)$ **定数小**
- **シェルソート (Shell sort)** — h **毎にソート**

  $h_k = 3h_{k-1} + 1, \cdots, 40, 13, 4, 1$ **のとき** $O(n^{1.25})$
- **クイックソート (quick sort)** — **分割統治法 (devide and conquer), 平均** $O(n \log n)$**, 最悪** $O(n^2)$
- **ヒープソート (heap sort)** — **常時** $O(n \log n)$

  部分順序付き木を配列で表現 (heap) し, 最大値を求める

  a[1]: root, a[i**] の左の子を a[**$2i$**], 右の子を a[**$2i+1$**]**
- **マージソート (merge sort)** — **常時** $O(n \log n)$

整列アルゴリズムと平均計算量

- **選択ソート** $O(n^2)$
- **挿入ソート** $O(n^2)$
- **シェルソート** $O(n^{1.25})$
- **バブルソート** $O(n \log n)$
- **クイックソート** $O(n \log n)$
- **ヒープソート** $O(n \log n)$
- **マージソート** $O(n \log n)$
- **基底ソート** $O(nm)$
- **分布ソート** $O(n)$
- **逆写像ソート** $O(n)$

文字列照合 (string pattern matching), 文字列の探索

テキストの長さ n, パターン (pattern) の長さ m

1. 素朴なアルゴリズム (naive) — 最悪の計算量 $O(nm)$
2. Knuth-Morris-Pratt (KMP) のアルゴリズム

   何文字か一致後の不一致 $\Longrightarrow$ 無駄な照合をスキップ

   (a) 前準備 — 位置 n で不一致, n を揃える位置 m を計算

パターン (0 始点)	a	b	a	b	b	a	a
不一致位置		1	2	3	4	5	6
再開位置							

   (b) 照合操作

   計算量 $O(n+m)$

文字列照合の続き Boyer-Moore (BM 法) のアルゴリズム

- パターンをずらす量の決め方 (パターンは右端から調べる)
  1. 不一致となった時点でのテキストの文字

テキストの文字	n	e	v	r	それ以外
ずらし量					

  ダメな場合 aaaaaaa...aaaa, baaaaaa $\Rightarrow O(mn)$

  2. 何文字か一致後の不一致, 一致情報を利用 (a là KMP)

パターン	a	b	c	b	c	a	b	c
ずらし量								

- 照合操作
- ずらし量は上記の 2 つから決める.

$O(n+m)$ うまくいくと $O(n/m)$

レポート問題

どれか一つのアルゴリズムを選んで，その動きを表示するプログラム (Java, X Windows, Matlab, ...) で作成しなさい.

- 御自身の研究に関係のあるアルゴリズム
- Boyer-Moore (BM 法) の文字列照合アルゴリズム
- その他

締切は 5 月 31 日 (月) 午後 13 時,

工学部 10 号館入口レポート提出箱