トレーニング・フォア・データライブラリアン

天野晃

May 4, 2019

Contents

Part I

導入

Chapter 1

対象読者

本書はもともと、筑波大学情報学群の学部生を対象とした授業テキストであったものを、ライブラリアンのトレーニング向けの内容に改編したものである。本書が対象とする読者は、現職の図書館員、または、相当の業務 (研究支援等) を行う職員の中で、さらにデータライブラリアンとしての専門性を目指す人々である。広くは研究所や大学の情報及び図書に関わる部門において研究データを取り扱う業務に従事する人々を対象としている。前提としている知識・技術としては、すでに図書館の業務に十分に慣れており、IT 技術水準は、コマンドラインシェルを扱え、簡単なプログラム (スクリプト) が組める程度である。例えば、UNIX ユーザーなら bash、Windows ユーザーなら Powershell を利用

し、複数のプログラムを呼び出し、パイプライン化を行うようなことがすでに身についていることを前提としている。また、データベースの知識があることが望ましい (利用経験はなくて良い)。そのような読者にとっては、本書が、すでに身につけているデータ処理技術をさらに伸ばすトレーニングのきっかけとなるであろう。

本書は「一冊で終わる」本ではない。エッセンスのみを凝縮させたような、いわば、学習の起点となるような本を目指した。その目的から言えば深い理解を目指しているわけでもなく、研究者とデータについて語り合える最低限の計算機とデータの知識・技術を提示しているにすぎない。しかしながら、あるいはそれだけに、バランスの良い入門書としての存在を目指した。

トレーニングである以上、本書では各所で読者に実践を勧める記述がある。

> ⇒ このような box があるときは本書を離れ実践してみよう。

スキルを身につけるためには、ぜひ、実践していただきたい。当然、アプリケーションのインストールなどの基本的な PC のスキルは身についているものとして話を進めるので、必要なツールを判断し、適宜インストールして欲しい。特に本書では C 言語を引き合いに出すことが多いので、C 言語にはある程度慣れていただきたい。

Chapter 2

図書館員を取り巻く状況

2.1 データサイエンティストとデータライブラリアン

データライブラリアンと同様にデータに深く関わる職としてデータサイエンティストと言われるものがある。近年、研究におけるデータ管理・公開の重要性が強く認識されるようになり、これらの職種に対して注目と期待が集まっているが、一つの差別化の考えとしては、データサイエンティストはあくまでも研究者であり、研究対象がたまたまデータであるに過ぎないと考えることができる。この意味において、他の研究者と同様、常にライブラリアンの助けを必要としている。

2.2 想定するデータライブラリアンの仕事

データライブラリアンは、研究者が研究活動に専念できるように様々なサポートを行なうべき存在である。雑誌契約やレファレンス、図書受け入れ作業などと同格な業務として、データ利用契約、データ検索、登録データのチェック、公開データの構造化、これらに関わるシステムの設計、ツールの作成などといった作業を行うことになるであろうことが予想され、業務の目的としては何ら今までのライブラリアンの仕事と変わらない。

ただ、技術面に注目すれば、このような作業を行うには、各分野におけるデータの性質に関する知識があることに加え、データチェックのための技能、データを構造化するための知識やデータ構造の解析技術などが身についている必要がある。また、このような様々な形態のデータを処理するにあたり、毎回対応するスクリプティングやプログラミングを行うことは非効率的であり、これらのデータを効率的に扱うことができる高機能なツールは必須である。しかしながら、逆説的ではあるが、高機能なツールでは処理が重すぎて利用が困難な場合もある。つまり上記の業務を効率的に行うには、いずれの場合にも対応できるように、それぞれのツールに対する知識が必要となる。本書によって、これらのツールや機能を使いこなすための、データに対する知識とデータ処理の基本的な考え方が身につくことを期待している。

Part II

基礎技術編

Chapter 3

計算機とデータの知識

3.1 データとは

データとは、計算機が扱うことのできる情報の形式と考えることができる。この意味では、データを理解するためには計算機の構造をある程度理解する必要があることを述べておく。このことは「データ処理とは」の章でもう少し詳しく解説するが、本書では端的には「電子計算機が扱うことのできるバイト列またはビット列」を指す。

⇒ 電子計算機以外にどのようなタイプの計算機があるか調べてみよう。

3.1.1 データ型

一般的にデータ型とはデジタル電子計算機が扱うデータフォーマットのことと考えてよい。データ型には幾つかの抽象化レベルがあるが、最下層としては単なるビット列をどのように解釈するかである。すなわち、例えば以下のようなものである。

- 整数型
- 小数 (浮動小数点数) 型
- 文字型

⇒ 上記以外にどのようなデータ型があるか調べてみよう。

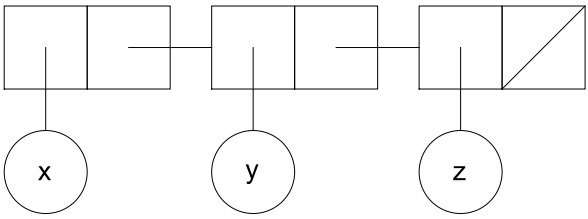
3.1.2 文字と文字列

ここで、文字と数字、文字列について述べておこう。文字は文字コードによって解釈されるが、文字コードはさらにエンコードされている。UTF8 とは、ユニコードをエンコードする一つの方法である。数字とは、数を表すための文字であり、数値とは異なる。数値とは前述したように、整数や小数という概念そのもの、あるいは計算機上で数値として取り扱うビット列のことである。文字列とは文字コードまたは文字エンコードの並らびである。数値は、数字列に

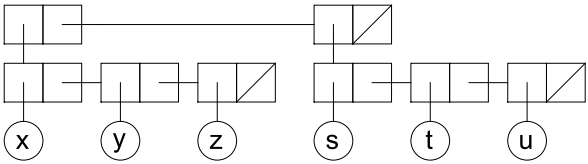
よって表現されることが一般的である。直接バイナリ記述される場合もある。

⇒ このことを C のsscanf()などで確認しよう。
⇒ リトルエンディアンとは何か調べてみよう。

3.1.3 集合、リスト、ツリー、グラフ

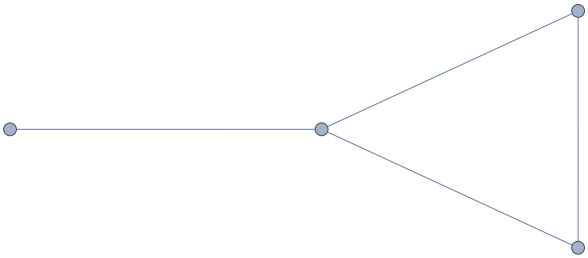


リスト (x,y,z) で表されるリスト構造



リスト ((x,y,z),(s,t,u)) で表されるリスト構造 (行列)

Figure 3.1: [リスト構造]



グラフのグラフィクス表現

$$\begin{pmatrix} 0 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 0 \end{pmatrix}$$

グラフの隣接行列表現

Figure 3.2: [グラフ構造]

一般的にデータを取り扱う際にはデータ単体 (datum) の単位では扱わず、データ集合として扱う。ここで言う集合は一般的な集合ではなく、一種の配列、つまり、要素の並び順や重複に意味があるものである。リストとは、要素を並べただけのものから、より高位の階層構造などを表すために用いられる表現である。最も代表的な構造は Cons Cell であり、文字列による表現ではカッコ () によって階層を表し、内部実

装としてはポインタによるセルの連結である [Figure:3.1]。近年よく用いられる JSON 形式は一種のリスト表現である。ツリーは基本的にはリストと同じで、分岐のあるリストを特にツリーと称する。この場合も内部実装は Cons Cell で現すことができる。グラフはより高位の構造であり、抽象レベルの表現、内部実装ともに複雑になるが、これらを表現する幾つかの方法がある。その一つは隣接行列を利用することである [Figure:3.2]。

グラフよりもさらに高次な構造も当然ある。グラフの均質化やラベル付与は、より複雑なネットワーク構造を表す [1]。いずれの構造を表わすにおいても、前述のリスト構造がデータ構造における基本と考えてよい。

⇒ 配列とポインタ参照

プログラミング言語ではよくポインタが使用されるが、Java のような近代的な言語はその効果が隠蔽されており、解り難い。C 言語においてすらポインタ参照と配列参照はカッコ [] と引数で表現され、違いがない。しかし、次のようなコードでその違いを確認できる。

==========Start of code==========

```
/* データサイズを print する */
#include <stdio.h>
#include <stdlib.h>

int main(){
        printf("int             \t:%ld:\tbyte\n",sizeof(int));
          int a[2][2];
```

```
        printf("int 2D:2x2      \t:%ld:\tbyte\n",sizeof(a));
        printf("pointer int*    \t:%ld:\tbyte\n",sizeof(int*));
        return(0);
}
```

==========End of code==========

出力は以下となる。

==========Start of print==========

```
int             :4: byte
int 2D:2x2      :16: byte
pointer int*    :8: byte
```

==========End of print==========

3.2 データ処理とは

データ処理とは、計算機が扱うことのできる情報の変換である。より詳細には、処理対象と処理命令の定義が記述された表現を、その表現以外の知識 (ルール) を用い、表現が変化しなくなるまで評価を続けることであり、すなわち、端的には「表現評価系を用いた表現の変換」と言える。この解釈では「処理命令」(=プログラム) と「処理対象」(=データ) は分離が曖昧であり、計算機科学としてはどちらも「処理対象」とすることがふさわしい。このことは、より抽象的

かつ厳密に万能チューリングマシンとして定義可能である。

3.2.1 チューリングマシン

チューリングマシンとは、アラン・チューリングによる仮想機械の仕組みであり、現実の計算機はこれを物理的に実装したものであり、次の要素から成る。

- 状態 (有限) を記憶するメモリ
- 記号 (有限) 列 (無限) を記録するテープ
- メモリとテープを読み書きするデバイス

なお、チューリングマシンの論理的表現の要素は以下となる。

- M: 状態を表す有限集合
- T: 記号集合
- δ: $\delta(m,t) = (m',t',h)$: 遷移関数、ただし、$m, m' \ni M$、$t, t' \ni T$、h: テープの移動方向

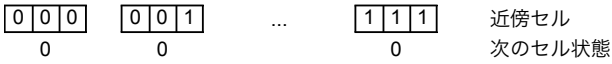
つまり、マシンは現在の状態 (m、t) を読み、遷移関数の定義に従い次の状態 (m'、t') を作り、どちらかに移動し、これを繰り返す。ただし、以上は最も簡単な構造であり、より複雑な多次元テープと複数デバイスのモデルも構築することができる。実際の計算機は万能チューリングマシン、つまり、他のチューリングマシンをエミュレート可能なチュー

リングマシンとして設計実装されており、基本的には上記の考えを拡張したものである。

⇒ WolframAlpah を利用してチューリングマシンの動きを確かめよう。

3.2.2 セルオートマトン

一方、局所規則のみで自己組織的に駆動する構造がセルオートマトンである。一部の規則群がチューリング完全である。つまり、適切な規則を定義することによりチューリングマシンとして機能する。セルオートマトンは、n 次元のセル



セルオートマトン (1 次元 2 近傍 2 状態) の変化規則の一つ

Figure 3.3: [セルオートマトン]

構造とセルの取り得る状態とセル状態の変化規則からなる。例えば、「1 次元 3 近傍 2 状態」とは、一次元に並ぶ有限または無限のセルと、変化規則が適応される 3 つの連続したセル (うち真ん中の一つは自分自身 = 状態変化の対象) のパターンからなる。このとき、8 通り ($= 2^3$) の状態からの遷移を規定しなければならず、すべての変化規則は $256(= 2^8)$ 通りとなる [Figure:3.3]。

⇒ 2次元9近傍2状態の変化規則が何通りか考えよう。

3.2.3 プログラミングとは

プログラミングとは、計算機に対しての処理命令をまとめたものであり、プログラミング言語はそれを行うための仕組みの一つである。広義には、人間または計算機が、計算機に指示をするための様々なレベルの様々な仕組みであり、簡単なバッチ処理スクリプトや、たった一つの機械語命令もこれに含まれる。しかし、より狭義には次のような特徴を持つプログラミング言語を用いて行うものが、プログラミングであると考えるのが妥当であろう。

- 抽象化
- 繰り返しまたは再帰
- 条件分岐

抽象化とは、変数を利用可能であることである。ポインタの利用も抽象化の一種である。繰り返しと再帰は、本質的に同等であり、相互に書き換えが可能であるとされる。条件分岐は、最も直感的に必要であると考え得る機能のはずである。

⇒ マークアップ言語 (HTML や YML) がプログラミング言語であり得るか、考えてみよう。

3.3 データ処理の基本技術

3.3.1 文字列処理

文字列処理の中心はパターンマッチおよび文字 (列) 書き換えの技術である。パターンとは、特定の文字列または文字列集合を表すもので、多くの場合、このパターンそのものが文字列として表現される。とくに、文字列集合を一つの文字列で表す方式を正規表現または正則表現と言う。現在では、正規表現はライブラリ (エンジン) として纏められ、共通ライブラリが複数のユーティリティーで利用されている。Perl 互換正規表現ライブラリ (PCRE) と鬼車 (Oniguruma) は代表的な正規表現ライブラリである。

一方で、バッカス・ナウア記法 (BNF) は、文法定義のためのメタ言語である。すなわち与えられた文字列がその言語において受け入れられるかを決定するための表現である。

さらに、マークアップ言語もマークアップ言語に対する定義を表すスキーマ言語も、メタ言語として捉えることが可能である。

上記のいずれの表現も、文字列パターン定義および文字列処理定義に利用可能である。

⇒ 正規表現、BNF(文法定義記法)、マークアップ言語、に対するユーティリティーに何があるか調べてみよう。

3.3.2 数値処理

3.3.3 数式処理

⇒ テスト用。

Chapter 4

統計または解析の技術

Chapter 5

図表表示の技術

Part III

実践編

Chapter 6

サービス構築

6.1 データベース

6.2 HTTP サーバー

Chapter 7

データ解析

Bibliography

[1] Fujiwara Yuzuru. 情報学基礎論の現状と展望. **情報知識学会誌**, 9(1):13, 1999.