A-025

ある単純な木表現法の応用について

Applications of a Simple Tree Representation

都倉信樹†

Nobuki TOKURA

1. はじめに

レベル木表現（Tree Representation with Levels, TRL と略す)と呼ぶ単純な木表現法について，その応用をのべる．木を用いたアルゴリズムの説明図をテキストに多数描かざるを得なくなり，ドロー系のソフトを使うが，これは木を手書きしているのと同様で，思い描いている木と同数の節点や枝などを描画しないといけない．労力は大きい．できるだけ簡便な手段でとりあえずの説明用の木(図 1 のようなグラフ）を描画するプログラムを作りたいと考えた．

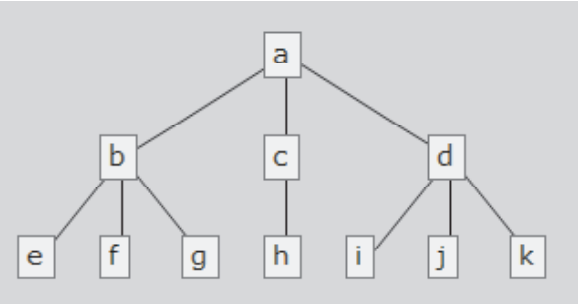

図 1　木グラフの例

(口は節点，根はこの場合 a.　枝は上から下へ向かう)

```
{(a,b),(a,c),(a,d),(b,e),(b,f),
 (b,g),(c,h),(d,i),(d,j),(d,k)}
```

図 2 枝リスト

```
{(a:(b,c,d)},(b:(e,f,g)),((c:(h)),(d:(i,j,k)}
```

図 3　節点と子節点リストのリスト

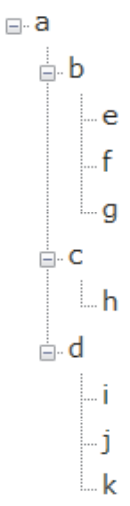

図 4　TreeView

```
a
    b
        e
        f
        g
    c
        h
    d
        i
        j
        k
```

図 5　Indent 方式

```
0,a
1,b
2,e
2,f
2,g
1,c
2,h
1,d
2,i
2,j
2,k
```

図 6 TRL

ところが，木のデータの入力法で行き詰まる．ラフスケッチから直接入力データに結びつくものが欲しかった．枝リスト(図2)，節点と子のリストのリスト(図3）を検討したが，これらはコンパクトではあるが，意外と入力しにくい．図4の TreeView は OS でディレクトリ構造を表示するのに使われており，よく見ているであろう．このコントロールを使い，節点の挿入や削除，名前などの変更などできるプログラムは容易に作れる．しかし，一から一つの木を作るには，ドローソフトと同等の労力を要する．

そこで気がついたのが，Python などのプログラム言語で使われているインデント方式である．図5にその例を示すが，これは入力量も非常に少なくラフスケッチを見ながら，さっさと入力できる．しかし，ひとつ問題がある．タブというホワイトスペースの数が基本になるが，木が非常に深くなって折り返すとか，ページが変わるととたんに戸

† 大阪電気通信大学

Osaka Electro-Communication University

惑う事態になる．そんな大きな木は扱わないということならそれでもいい．しかし，見えないタブの数に頼ることで，ときどきトラブルを経験した．そこで，タブの数を明示的に先頭におき，その後に木の情報をならべ，1 行で一つの節点を表す，図6のような TRL に至った．

以下，２でTRLについて説明する．

2. 木と TRL

ここで扱う木 T(V,E,r)で，V は節点の有限集合，E ⊆ V×V は有向枝の集合（ただし，図では上から下に向きがあるとして，矢印にはしない），r ∈ V は根である．もうひとつ条件があり，ここで考える木は順序木(ordered tree)でもある．つまり，各節点において子節点は(図示するときは左から）順序が付いていると考える．その順序は0始まりのインデクスで参照する．なお，節点 v のレベル L(v)は，L(r)=0, v の子節点の集合を Child(v)とするとき，各 c ∈ Child(v)について，L(c)=L(v)+1 と再帰的に定義される．

2.1 TRL

TRL はコメント行なども含めることがあるが，それを除けば，T の節点を前順序で走査した順に，1 行に 1 節点の情報を記入したテキストである．その各行は，その節点 v のレベル L(v)と節点の名前，その他節点に付随する属性情報を表す文字列をカンマなどのセパレータで区切った形の文字列である．ある節点 v を根とする部分木 S(v)に対応する行は，v に対応する行を先頭に，レベルが L(v)より大きいものの連続した行である．このように，部分木が連続してまとまっているので，削除や挿入作業は非常に簡単に行える．

2.2 他の表現法の関係

XML，JSON 等は完全括弧つき表現（fully parenthesized notation)に相当し，親節点の括弧の中に子節点の情報がすべて含まれる．こういう完全括弧つきの表現から離れたものが，有名なポーランド記法である．たとえば，前置記法はオペレータの後にそれの作用する部分式を並べる．しかし，オペレータの数 arity は固定されていて，オペレータを左の括弧として，右の括弧は次のレベルの部分式が arity 個並んだあとにあると解釈することで，完全括弧付き表現と同等の能力をもつ．また，Landin[2]によって提唱された字下げ表現(indent representation)は，Python 等のプログラム言語に採用されてよく知られるようになった．これもレベルが戻るところで，レベル差の閉じ括弧を補えば完全括弧付き表現が得られる．本論のレベル木表現は，インデントする代わりにそのレベルを併記する．インデントの数を数えれば，レベルが分かるし，レベルがわかれば，インデントも付けられる．ここではレベルはその数で表現しているが，前レベルとの差分を使うとか種々の変形は考えられる．これらの表現の中では TRL は比較的コンパクトに木構造を直感的に把握しやすく表現し，他表現との変換，あるいは，木構造の変形操作において，アルゴリズムが簡単になると感じた．

3. 応用例

3.1 TreeEditor

入力は TRL を含むテキストファイルで，これから木構造を決定し，描画する． また，比較用に，TreeView でも表示する．図1程度の品質の木が楽に描ける．

また, 描画色や節点の形状等を別に設定する機能, XML, JSON 等に変換する機能も用意している．

3.2 ラムダ簡約の可視化

ラムダ計算の簡約について，テキスト形式での可視化を行った．このとき，代入される部分式を強調表示するために RichTextBox コントロールを用いた．その変形の履歴を RichText 形式で出力し，簡約例をテキストにそのまま挿入できるようにした．従来手計算で簡約して，それを入力していたのが大幅に省力化できた．また，ラムダ表現を木で表現する１つの方法をつくり，それを TRL で内部的にもち, 変数名の書き換えや部分木の代入の操作を可視化した．これは順々に木が変形していく過程を見せるもので学習者の理解を助けるであろう．

このように，木構造の変形を伴うアルゴリズムを扱う際に，外部表現だけでなく，内部でも使うのは有用である．

3.3 数独プログラムの中断再開処理

数独（ナンプレ）というパズルがある．これを紙上で解くかわりに問題を画面に表示し，マウスで指定した数字を

指定した場所に書き込んだり，訂正したりする機能をつけたプログラムを作った．こういうゲームの場合，中断の対応を考えたい．そこで，用いるデータ構造の内容をすべてシリアル化してテキストファイルに格納し，それを逆シリアル化して続きを楽しめるものとした．たとえば，9×9の配列，UNDO用のスタック，初期配列等各種のデータをシリアル化した．このとき，シリアル化の定番とされている XML だと非常にプログラムも面倒である．XML でなく，TRL を使うことで，ごく簡単にシリアル化，逆シリアル化ができる．その詳細は省略するが，その際，汎用の拡張メソッドをいくつか作った．以下，4 で数例をあげる

4. シリアル化, 逆シリアル化

4.1 単純変数のシリアル化拡張メソッド Serialize

これは次のように作れば良い．

```
public static string Serialize(this int a,
        int level, string name)
        => $"{level + 1} : {name}, {a}";
```

図 7　int 用 serialize メソッド　(C#)

第 1 引数の this int で，int 型の拡張メソッドとなる．ラムダ演算子 => に続いて，補間表現で 1 行のデータを作っている．子ノードとして，level を 1 増やし，名前と整数値をセパレータ ”:”. ”, ” で区切って出力する．逆シリアル化の拡張メソッドも示す．out 引数で結果を得る．

```
public static void DeSerialize(this string s, out int a)
   =>   a = int.Parse((s.Split(':', ',')[2].Trim()));
```

図8　図7に対応する逆シリアル化拡張メソッド　(C#)

4.2 ジェネリックなリスト　List<T>のシリアル化

数独では，配列，スタックなどのデータ構造を用いた．セルの状態は構造体に持たせており，これは内部では List<CellInfo>などとして管理した．ここではジュネリックなリストのシリアル化の拡張メソッドを作った例を示そう．

List<T>のシリアル化

```
1  public static string[] Serialize<T>
2   (this List<T> S1, int level, string name, int size)
3  {
4      string[] SA = new string[size + 1];
5      SA[0] =$"{level} :   {name}, {size}";
6      int i = 1;
7      T t = S1[1];
8      switch (t)
9      {
10         case int _:
11             foreach (object s in S1)
12             {
13                 SA[i] = ((int)s).Serialize(level, "int");
14                 i++;
15             }
16             return SA;
17         case Double _:    略
18         case string _:    略
19         略
20         default:     return SA;
21     }
22 }
```

図 9 ジェネリック List のシリアル化拡張メソッド例　(C#)

説明

L.1-2　汎用リスト型 List,T>に対する拡張メソッドで，引数は int level, string name, int size を受け取り，文字列配列を結果として与える．Size にはリストに含まれる要素数を与える．

L.4　文字列配列 SA を size+1 個の文字列が入るとする．

L.4　L.4 で+1 したのは，この行をこのグループのヘッダとして追加するため．この行は level そのものを使う．

L.7　T t=S1[1] 渡されたリストの先頭の要素を t に入れる．これは次で利用する．

L.8-20　t で型スイッチの機能を使う．T の型名でなく，T のインスタンスを用いるので，t を使った．空リストの対応は試作版では省いている．

L.10-16　t が int 型の場合．S1 の各要素について，先に作った int 用のシリアル化メソッドを呼び出して，結果を配

列に入れていく．すべて終われば，結果の配列を return で戻してこのメソッドを抜ける．

L.17-18　int の場合とほぼ同様．

L.19　T に構造体やタプルなどにも適用できる．

4.3 実行例

```
List<int> SA = new List<int> { 2, 3, 7, 18 };
string[] SB = SA.Serialize<int>(1, "SA", SA.Count);
```

図 10　シリアル化の例　(C#)

L.1　で，整数リスト SA を定義している．

L.2　で，SA の Serialize メソッドをレベル 1 からになるように生成することを指示する．

これを実行すると，SB には，シリアル化した結果が得られる．

```
1 : SA, 4
2 : int, 2
2 : int, 3
2 : int, 7
2 : int, 18
```

図 11　SB 内に得られる TRL 表現

1行目にはどういうデータをシリアル化したかの情報がレベル1で示されている．その子ノードにリストの各要素をシリアル化した情報をレベル2として生成している．このように生成されたシリアル化の結果が容易に読めるということも大きなメリットである．

5. むすび

[1]は，プログラミング入門教育に続く教育の教案の例示提案であるが，ここでの検討の発端になった木を描画するプログラム，TreeEditor が主要な例題となっている．その実行画面では，左ペーンに TRL(図 6)を読込み，中央ペーンに TreeView(図 4)を表示し，右ペーンにグラフ(図 1)を表示する．このようなプログラムを作ることを意図したことから，TRL にたどり着いたが，いろいろの木を描画することで，当初のラフスケッチからの入力という点では TRL は適した表現であると感じている．

また，ラムダ簡約プログラムでは内部表現として，名前の変更，木に対する部分木の切り出しとコピーなどの操作が非常に簡単にできる．これは内部で，TRL の要素ごとに LIST にすることで，LINQ 機能を利用できたことが大きい．内部表現と外部表現が近いということも時としてメリットになるであろう．

また，数独プログラムでは，中断再開処理に，TRL へのシリアル化，逆シリアル化を行って，ごく短時間に実現できた．通常シリアル化というと XML，あるいは JSON が使われているが，XML についてはいろいろ実務家から批判はある．TRL はある程度答えるものとなるのではないかと感じる．

最後に参考までに，先の図 1 の例を XML で表現してみたものを挙げておく．

```
<?xml version="1.0" encoding="utf-8"?>
<TreeView>
  <node text="a">
      <node text="b" >
          <node text="e"/>
          <node text="f"/>
          <node text="g"/>
      </node>
      <node text="c">
          <node text="h"/>
      </node>
      <node text="d">
          <node text="i"/>
          <node text="j"/>
          <node text="k" />
      </node>
  </node>
</TreeView>
```

図 12　図 1 の木構造の XML シリアライズ

参考文献

[1]都倉信樹．入門プログラミング教育につづく科目案，SSS2019,情報処理学会，採録決定 2019.8.

[2] P.J.Landin.”The next 700 programming languages”，C.ACM vol.9,No3, pp.157–166 (1966 March).