計算機プログラミング'93　＃8

久野　靖*

1993.10.27

1　探索技法

計算機による「単純でない」問題解決の多くが探索という枠組みで捉えられる。例えば、

- 人物 A がある人の子孫であるかどうかを調べる。
- ある人のすべての子孫のリストを求める。
- チェスのある盤面から始めて勝つ経路を求める。

などなど。これらの問題は次のような基準で分類できる。

- 徹底探索か、即決探索か。
- 探索空間は予め与えられているか、探索とともに作り出されるか。
- 探索空間は木かより一般的なネットワークか。
- 探索方式は深さ優先か幅優先か。
- しらみつぶし探索か否か。

これらがどういう意味かは以下の例題を通じて分かっていただけると思う。

2　幅優先探索と深さ優先探索

まず、「ある人の子孫をすべて打ち出す」という操作を考えてみる。例題には前に使ったデータをちょっと増やしたのを用いる。

```
(setq *persons* '(alice john bob mary chris bud cathy james))
(setf (get 'bob 'children) '(donald robby))
(setf (get 'alice 'children) '(bob mary chris))
(setf (get 'john 'children) '(bob mary))
(setf (get 'bud 'children) '(james alice))
(setf (get 'cathy 'children) '(james alice))

(defun expand-children (x) (get x 'children))

(defun visit1 (node expand action &aux queue)
  (setq queue (list node))
  (loop (if (null queue) (return))
        (setq node (pop queue))
        (funcall action node)
        (setq queue (append queue (funcall expand node)))))

(defun print-all-children (who)
  (visit1 who #'expand-children #'print))
```

*筑波大学大学院経営システム科学専攻

```
>(print-all-children 'cathy)
CATHY
JAMES
ALICE
BOB
MARY
CHRIS
DONALD
ROBBY
NIL
```

このやり方では、まず指定した人が打ち出され、次にその子供がすべて打ち出され、次に孫がすべて打ち出され…という順序になる。(なぜか?) このような順序での探索を幅優先探索という。これに対し、次のように展開したものを queue の先頭に置くようにすると…

```
(defun visit2 (node expand action &aux queue)
  (setq queue (list node))
  (loop (if (null queue) (return))
        (setq node (pop queue))
        (funcall action node)
        (setq queue (append (funcall expand node) queue))))

>(print-all-children 'cathy)
CATHY
JAMES
ALICE
BOB
DONALD
ROBBY
MARY
CHRIS
NIL
```

今度はまず第 1 子の子孫が全部プリントされ、その後で第 2 子、…という順序に変わる。これを深さ優先の探索という。

ところで、関数 visit1/visit2 は特定の問題には依存していないことに注意。どんなデータでも、それが木構造であれば「子供を展開する」関数と「節を処理する」関数を与えることで動作する。では問題。

練習 24 次のような関数を書け。なおテストデータ類はすべてまとめて/ua/kuno/work/ex7data.lsp に入れてあります。

- 例題のようにプリントする代わりに、すべての子孫をリストとして返す関数 collect-all-children を書け。ヒント: 子孫のリストを集めるのに局所変数を使用し、action として渡す関数を lambda 式で書いてその中で局所変数に追加していく。
- 人名 A、B を渡して、A が B の子孫かどうか調べる関数 sison。

なお、2 番目の方は条件にかなうものが 1 つ見つかったら直ちに終りにすることができる。このような探索を即決探索という。これに対し全部の節を調べる場合には徹底探索という。

3 グラフの探索

さて、これまでは木だったので経路にループがなかった。しかし、より一般のグラフの場合はループもある。例として図のような鉄道の経路を考えてみる。これを次のようなデータ構造であらわす。

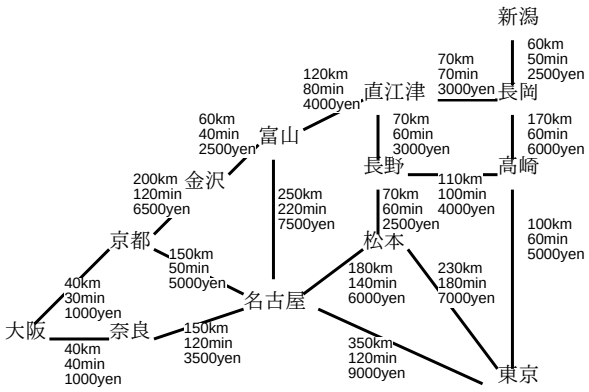

図 1: 鉄道の経路、距離、運賃

```
  (setq *cities* '(tokyo nagoya osaka kyoto nara kanazawa toyama
                         naoetsu nagaoka niigata nagano takasaki matsumoto))
  (setq *prices* nil)
  (setq *ntimes* nil)
  (setq *distances* nil)

  (dolist (c *cities*) (setf (get c 'adj) nil))
  (defun link (x y d m p)
     (push x (get y 'adj)) (push y (get x 'adj))
     (push (list x y d) *distances*) (push (list y x d) *distances*)
     (push (list x y m) *ntimes*) (push (list y x m) *ntimes*)
     (push (list x y p) *prices*) (push (list y x p) *prices*))

  (link 'tokyo 'nagoya 350 120 9000)
  (link 'nagoya 'nara 150 120 3500)
  (link 'nara 'osaka 40 40 1000)
  (link 'nagoya 'kyoto 150 50 5000)
  (link 'kyoto 'osaka 40 30 1000)
  ; とりあえず簡単のためこれだけ。

(defun distance (x y)
   (dolist (l *distances*)
           (if (and (eq (car l) x) (eq (cadr l) y))
               (return-from distance (caddr l)))))

(defun ntime (x y)
   (dolist (l *ntimes*)
           (if (and (eq (car l) x) (eq (cadr l) y))
               (return-from ntime (caddr l)))))

(defun price (x y)
   (dolist (l *prices*)
           (if (and (eq (car l) x) (eq (cadr l) y))
               (return-from price (caddr l)))))
```

これまずは、ある都市から始まる (堂々めぐりにならない) 経路をすべて打ち出す例をやってみよう。

```
  (defun gvisit1 (node expand action &aux queue path)
    (setq queue (list (list node)))
    (loop (if (null queue) (return))
          (setq path (pop queue))
          (funcall action path)
          (dolist (node (funcall expand (car path)))
```

```
                (if (not (member node path))
                    (push (cons node path) queue)))))

(defun print-all-path (city)
   (gvisit1 city #'(lambda (c) (get c 'adj)) #'print))
```

これを実行してみる。

```
>(print-all-path 'tokyo)

(TOKYO)
(NAGOYA TOKYO)
(NARA NAGOYA TOKYO)
(OSAKA NARA NAGOYA TOKYO)
(KYOTO OSAKA NARA NAGOYA TOKYO)
(KYOTO NAGOYA TOKYO)
(OSAKA KYOTO NAGOYA TOKYO)
(NARA OSAKA KYOTO NAGOYA TOKYO)
NIL
```

このように、経路のリストを持っていれば「前に来たからここから先は行っても仕方ない」という判断ができる。ループを含むグラフではこれが不可欠なわけである。

練習 25
- 上のプログラムは幅優先か? 深さ優先か? どちらにせよ、そうでない版を作成して動かせ。
- これを改造し、都市 x と y を指定し、x から y まで行く経路を 1 つ求める関数を書け。
- できるだけ短いパスでいく経路を求めるようにしてみよ。最短距離/時間/値段ではどうか。

4 小課題その 5 — ネットワークの探索

課題今回は少し役に立ちそうな問題として、上でやったようなデータを利用して何か役に立つ情報 (最短経路とか) を見つける関数をつくってみてください。なお、データファイルにはグラフ全体のデータが入っていますが、最初から全体で動かすとすごく大変でしょうから、ほとんどコメントになってます。うまく動くようになったら徐々に大きくしてみてください。

結果をレポートとして提出する場合には、必ず A4 版の紙を使用し、次のような順で構成し、全体を綴じて提出すること。

- 表紙。課題名 (1993 年度計算機プログラミング課題# 3)、学籍番号、氏名、提出日付) のみを記すこと。
- 方針。どのような考え方で課題を解こうと思ったか記す。
- 回答。作成したプログラムとその解説、実行例など。
- 考察。課題をやった結果どんなことがわかったか、何が問題か、など。(感想も入れてほしいが、感想ばかりでは内容として不足。)

〆切は一応次々回の授業 (10/13) の直前までとしますが、遅れても受理はします。小課題の提出は義務ではありません。