情報知能学科「アルゴリズムとデータ構造」試験の解答

出題者: 白井英俊　　日時: 2009 年 7 月 28 日 16:00–17:30 (822 教室)

注意: これは**教科書のみ持ち込み可**の試験である。解答は問題番号を明記して解答用紙に書くこと。解く順番は任意でよい。

問題 1: 以下は配列の要素をソートするアルゴリズムの一つである。このアルゴリズムを実現する Ruby のプログラムを作れ (関数名を isort とせよ)。また、入力の配列の要素数を n としたとき、このアルゴリズムの計算量のオーダはいくらと考えられるかを説明せよ。

入力: 正整数 n と、n 個の数を要素とする配列 a
出力: 要素が昇順にソートされた配列
手続き:

for $1 \leq i < n$ **do**　　# i が 1 から $n-1$ まで
  begin
    $j \leftarrow i$;
    while $(j > 0)$ and $(\mathrm{a[j-1]} > \mathrm{a[j]})$ **do**　　# $j > 0$ かつ $(\mathrm{a[j-1]} > \mathrm{a[j]})$ である限り
      begin
        a[$j-1$] と a[j] とを入れ替える (swap);
        $j \leftarrow j - 1$;
      end
  end
return a

注: 「a[$j-1$] と a[j] とを入れ替える (swap)」とは、例えば a が配列 [10,20,30,40,50] で、$j = 2$ とすると「a[$j-1$] と a[j] とを入れ替え」た結果、a の値が [10,30,20,40,50] となるようにすることである。

問 1 の解答

Ruby のプログラムは以下:

```
def isort(n,a)              # 引数は配列 a と要素数 n
   for i in 1...n           # for 1 <= i < n do
     j = i                                  # j <- i
     while (j > 0) and (a[j-1] > a[j])      #  while (j > 0) and (a[j-1] > a[j])
        a[j-1],a[j] = a[j],a[j-1]           #    # a[j-1] と a[j] とを swap
        j = j-1                             #    #  j <- j-1
     end # while                            # end ... while
   end # for                # end ... for
   return a                 # return a
end # def

# 以下は検証用
a = [30,10,20,60,50,40]
p isort(a.size,a)
```

計算量は $O(n^2)$。

一番外側の for では i は 1 から n までの値を取る。

その内側の while ループでは j は i から 1 までの値をとる。つまり i 回、「swap」が行われる。この swap の計算量は一定である。

よって、全体の計算量は $\sum_{i=1}^{n} i = \frac{1}{2}n^2$ であり、これは $O(n^2)$ に等しい。

問題 2: グラフ探索のアルゴリズム (Graph-Search, 教科書 p.62) では「頂点を入れるための入れ物 A,B を用意」というようなことが書かれていた。このように、アルゴリズムやプログラムでは「入れ物」として使えるデータ構造はとても重要である。

本講義「アルゴリズムとデータ構造」で学んだデータ構造のうち、「入れ物」として使えるデータ構造を「配列」を含めて少なくとも 4 種類あげ、それぞれ、どういう用途にはそのデータ構造を使うのがよいか、という特徴を述べよ。ただし、ここでは「変数」は「入れ物」として考えない (そもそもデータ構造ではないので)。

問 2 の解答

入れ物として使えるデータ構造として学んだのは、配列、ハッシュ、リスト、スタック、キュー、ヒープ、2 分探索木などである。ここでは配列、ハッシュ、スタック、キューをとりあげる。

- 配列は添字 (インデックス) という数によって要素にアクセスする。一般的な「入れ物」として使われるデータ構造。このアクセスはインデックスが指定されれば配列で記憶されている要素数によらず一定の手間で行える。また必要とする記憶領域は配列で記憶される要素数に比例した領域が必要。なお、要素の探索には要素数に比例した手間が必要。

- ハッシュは、記憶すべき要素そのものをキーとして、それに関連した値にアクセスする。要素の探索や追加、削除は要素数によらず一定の手間で行える。また必要とする記憶領域は、少なくとも記憶すべき要素数の倍の領域が必要とされ、配列と比べて大きな記憶領域を必要とする。

- スタックとキューはともに要素の取出しと追加が記憶領域の特定の場所で行われるデータ構造である。要素のアクセスは記憶されている要素数によらず一定の手間で行われる。記憶された要素を場所やその値によって探索する、という用途には向いていない。記憶された要素を特定の順番で取出すことが特徴である。記憶領域としては実装において配列を用いることが多い。

問題 3: ヒープについて問う。

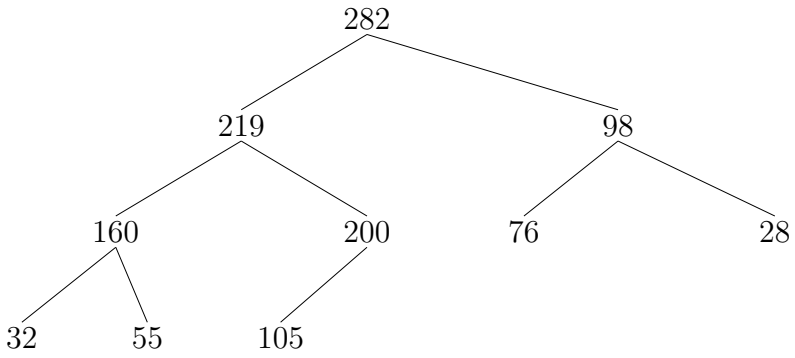
(問 3-1): 次の数を入力として構成されるヒープを示せ (数をこの順に読み込んで作るものとする)。ただし、２進木として図示すること。

32, 76, 105, 200, 282, 98, 28, 160, 55, 219

問 3-1 の解答

配列として書けば: `282, 219, 98, 160, 200, 76, 28, 32, 55, 105`

二進木として図示すると:



(問 3-2): この木の高さを答えよ。

問 3-2 の解答: 3

(問 3-3): この木の根の頂点の値 (ラベル) を答えよ。

問 3-3 の解答: 282

(問 3-4): この木の葉の頂点の値 (ラベル) をすべて答えよ。

問 3-4 の解答: 32, 55, 105, 76, 28

問題 4: 次の逆ポーランド記法でかかれた数式をスタックを用いて計算する過程を図示せよ (つまり、スタックの状態の変化がわかるように書くこと)。ただし、スタックの底と頂点を明示すること。

3 4 + 6 2 / * 8 1 − /

問 4 の解答:
以下では左側が底、右側が頂点。

(1) [3]

(2) [3 4]

(3) [7] $\cdots (= 3 + 4)$

(4) [7 6]

(5) [7 6 2]

(6) [7 3] $\cdots (= 6/2)$

(7) [21] $\cdots (= 7 * 3)$

(8) [21 8]

(9) [21 8 1]

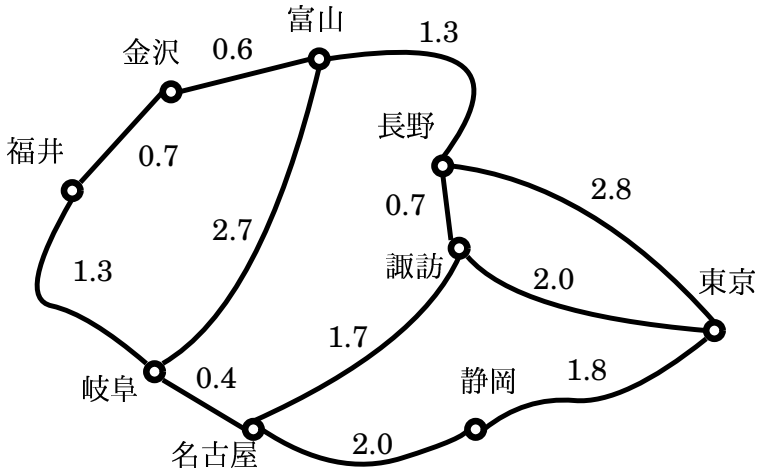
(10) [21 7] $\cdots (= 8 - 1)$

(11) [3] $\cdots (= 21/7)$

問題 5: 下図は日本の中部地方を中心とした道路地図の一部である。頂点は都市の名前、辺に書かれた数値は都市の間の所要時間を表しているものとする。

これをグラフとみなし (つまり、頂点集合は { 名古屋, 岐阜, 福井, 金沢, 富山, 長野, 諏訪, 静岡, 東京 })、Shortest-Path アルゴリズム (教科書 p.71、下に示す) を改変して、下の図に表されたグラフが与えられ、かつ東京を始点とした時に、東京と他のすべての市との最短所要時間を求められるようにせよ。つまり、どこをどのように改変するかを述べよ。

また、その改変したアルゴリズムに基づいて東京を始点として他のすべての市との最短所要時間を計算する過程を、アルゴリズムで用いられる d の値の変化に注目して記述し、最終結果を明示せよ。



Shortest-Path アルゴリズム
入力: グラフ G=(V,E), 隣接頂点関数 T, 辺の長さ関数 l, 始点 v_s, 終点 v_t
出力: 始点 v_s から終点 v_t までの最短距離
手続き:

1. $d(v_s) \leftarrow 0$
2. V $-$ $\{v_s\}$ に属すすべての頂点 v に対して $d(v) \leftarrow \infty$
3. $A \leftarrow \{v_s\}$
4. A の中の頂点のうち d の値が最小のもの v を取出す
5. $v = v_t$ なら、$d(v_t)$ を出力して終了
6. $T(v)$ に属す各頂点 w に対して
   if $d(w) = \infty$ **then** w を A に追加し、$d(w) \leftarrow d(v) + l(v, w)$
   elseif $d(w) > d(v) + l(v, w)$ **then** $d(w) \leftarrow d(v) + l(v, w)$
7. 4 へ進む

問 5 の解答: 改変するのは、アルゴリズムの 4, 5 番目である。そこを以下のようにする:

4. A が空なら、d を出力して終了
5. A の中の頂点のうち d の値が最小のもの v を取出す

別解: 5 を削除し、7 を「A が空なら d を出力して終了、さもなくば 4 へ進む」とする。

計算過程: (d のみを記述する. 括弧は A から取り除かれていることを意味)

	東京	静岡	諏訪	長野	名古屋	富山	岐阜	金沢	福井
Step 1–3	0	∞	∞	∞	∞	∞	∞	∞	∞
6	(0)	1.8	2.0	2.8	∞	∞	∞	∞	∞
6	(0)	(1.8)	2.0	2.8	3.8	∞	∞	∞	∞
6	(0)	(1.8)	(2.0)	2.7	3.7	∞	∞	∞	∞
6	(0)	(1.8)	(2.0)	(2.7)	3.7	4.0	∞	∞	∞
6	(0)	(1.8)	(2.0)	(2.7)	(3.7)	4.0	4.1	∞	∞
6	(0)	(1.8)	(2.0)	(2.7)	(3.7)	(4.0)	4.1	4.6	∞
6	(0)	(1.8)	(2.0)	(2.7)	(3.7)	(4.0)	(4.1)	4.6	5.4
6	(0)	(1.8)	(2.0)	(2.7)	(3.7)	(4.0)	(4.1)	(4.6)	5.3
6→4	(0)	(1.8)	(2.0)	(2.7)	(3.7)	(4.0)	(4.1)	(4.6)	(5.3)