平成 21 年 5 月更新(前回は 2 月)

【 正　誤　表 】

「C で学ぶ データ構造とアルゴリズム」

西原清一（著), オーム社, 第 1 版第 1 刷, 2008 年 1 月発刊

- 今回の新たな更新箇所は 2 箇所です。通番の直後に‘*’を付してあります。例：(17*)
- 下記のように、本書中の誤り(または不適切な部分)を訂正(または改訂)させていただきます。多くの訂正・改訂をお願い致しておりますことをお詫び致します。
- 動作確認を行ったプログラムのソースリストは下記に置いてあります。どうぞご参照ください。

http://npal.cs.tsukuba.ac.jp/~nishihara/

……ページ, 行……	誤り(または不適切)……		正（改訂後）……
(1)　3, 11	E が引用している	→	E を引用している
(2)　14, 4	を修正する	→	を見つけて修正する
(3)　15, 8	有用性が	→	有用性
(4)　34, 8	格納する．あとは	→	格納する．p->age はポインタ p の指す構造体のメンバーage を表す．あとは
(5)　35, 下 6	list;	→	list();
(6)　39, 図 2･13, 4 行目	f-r+m == 1)	→	f-r+M == 1)
(7)　41, 下 4	待ち行列	→	スタック
(8)　42, 図 2･16	（下図と置き換えてください）		

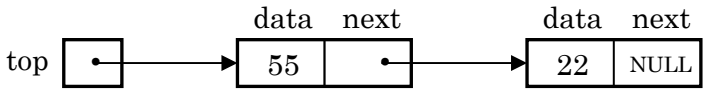

図 2･16　データ 22, 55 を追加後のスタック

(9)　46, 下 11	ために	→	という気持ちで
(10)　60, 1	i=k-4	→	k=i+4
(11)　74, 3	{"UVWXYZ"};	→	"UVWXYZ";
(12)　74, 図 3･17, 4 行目	{"ABCDEF"},	→	"ABCDEF",
(13)　74, 図 3･17, 4 行目	{"UVWXYZ"};	→	"UVWXYZ";
(14)　84, 図 4･9(a), 下 1 行目	}	→	};
(15)　99, 図 4･24（下記の 3 ヵ所）	=	→	==
16 行目： == BLACK	21 行目： if (t==0||		22 行目： sw == 0
(16)　99, 図 4･24, 下 5 行目	group(t,w[k]);	→	group(t,w[k]);}
(17*) 119, 下 3	ばばらない.	→	ばならない.
(18) 120, 8	こと）と	→	のこと）と
(19*) 123, 図 5･22, 3 行目	void strong_connect() {	→	void strong_connect() {

(20) 126,図 5･26,下 4 行目　　k = d[u]+w[u][x]　　→　　(k = d[u]+w[u][x])

(21) 127, 8　　(C 言語でどのように実現するか, 考えよ).

→

(具体的には, 最初 s[0],…,s[N-1]をすべて 0 に初期化しておき, ①では s[p]=1; を実行するとよい).

(22) 131,下 7　　p[i,j]　　→　　p[i][j]

(23) 131,下 5 (2 か所)　　parent[i,j]　　→　　parent[i][j]

(24) 132,図 5･32,1 行目　　int m;　　→　　int m,

(25) 132,図 5･32,下 3 行目　　″ => END　　→　　″ END

(26) 149,下 8　　代償　　→　　大小

(27) 150,図 7･2　　a[1] a[2] a[3] a[4] a[5]

→　　a[0] a[1] a[2] a[3] a[4]

(28) 150,下 4　　多い. しかし　　→　　多い. *n* はキーの個数. しかし

(29) 153, 4　　①では, 前半では,　　→　　①では,

(30) 153, 6　　繰り返される.　　→

繰り返される.(配列をヒープへ仕立てる方法としては, 他に, a[2]から a[N]に至るそれぞれのキーについて, 親との比較を行いながら根に向かって交換を試みる方法もある.)

(31) 153,下 5　　示す.　　→

示す. その場整列となっていることに注意されたい.

(32) 155, 14-15　　配列データ　　→　　入力キー列

(33) 156, 下 5　　a[n]　　→　　a[N]

(34) 161, 11　　ない.　しかし　　→　　ない. *n* はキーの個数. しかし

(35) 162,2～3 行　　ここでは, 入力キーの … 番人を置くところである.

→

ここでは, 入力キーの個数を N とし, 配列 a[1]～a[N]に格納されているものとし, 直感的な説明を行う.

(36) 163,図 7･15 を下記のプログラムで置き換えてください:

```
void quicksort(int p, int q)
{   int i,j,s,w;
    if (q-p < 1) return;                          ①
    i=p; j=q; s=a[p];
    while (i <= j) {
        for (; a[i] < s; i++);
        for (; a[j] > s; j--);                    } ②
        if (i <= j) {
            w = a[i]; a[i] = a[j]; a[j] = w;      } ③
            i++; j--;
        }
    }
```

```
    quicksort(p, j);                                    ④
    quicksort(i, q);                                    ⑤
}
```

(37) 163, 6～164, 下 10 の文章（全部で 35 行）を下記で置き換えてください:

簡単のため，入力キーは整数値とし，外部宣言された配列 a[]の a[p]～a[q]の部分に格納されているものとする．関数 quicksort(p, q) は，再帰的プログラムになっており，基準となるキーの値 s として先頭の a[p]を選んだ後，入力キー列を s 以下のキーと s 以上のキーとに仕分ける作業を再帰的に繰り返す．

まず，①では，キーの総数が 1 個以下であればただちに処理を終える．

本プログラムの主要な処理は，左端の a[p]および右端の a[q]から走査を開始し，②に示すようにそれぞれ s 以上のキーと s 以下のキーを順に探していき，見つかったらそれらを③において交換するという操作である．これを i≦j である限り繰り返す．

実際，変数 i は，それより左のキーはすべて s 以下であることを保証するもので，処理の進行につれて大きくなっていく．変数 j はちょうどその逆の働きをする．

最終的に，a[p]～a[j]には s 以下の値を持つキーが入り，a[i]～a[q]には s 以上の値を持つキーが入る．したがって，④において，これら二つに分割されたそれぞれの部分並びに対して quicksort()が再帰的に適用される．

なお，入力キーに同じ値のものが含まれている場合でも正しく動作することを確認されたい．とくに，基準値 s を持つキーが複数個ある場合は，それらは分割後の二つの部分並び分散されることがある．それでも処理は正しく行われる（これも確認されたい）．

【問題 7.11】

図 7･15 の②の右向き走査と左向き走査が一致するのはどのような場合か．

（**解答**）例えば, 図 7･14 において a[6]のキーが（15 ではなく）75 だったらどうなるであろうか．また，この場合，④と⑤の quicksort()において，（j-i=1 ではなく）j-i=2 となることにも注意されたい．詳細省略．

図 7・15 では，基準値 s を与えるキーとして，単純に a[p]を用いている．しかしクイックソートでは，キー列をなるべく均等に分割するように基準値を選ぶことが重要である．たとえば，ランダムに 3 つ(以上)のキーを選び，その中央値を採用するなどの方法がある．このような場合，まず選ばれたキーと a[p]とを交換してから，図 7･15 を施すとよい．

また，キーの個数が少ないとき（たとえば 10 個以下）は，単純な整列法（例えば単純挿入法など）を呼ぶほうがむしろ効率がよい．このようなときは，①の替わりに，

```
        if (q-p < 10) return straight_sort(p, q);
```

とするとよい．ここに，straight_sort は単純な整列法を指す．

(38) 164, 下 8　　$n \log n$　　→　　$O(n \log n)$

(39) 177, 図 7･30　を下記のプログラムで置き換えてください:

```
void selection(int p, int q, int k)
{   int i, j, s, w;
    if (q-p < 1) return;
    i=p; j=q; s=a[p];
    while (i <= j) {
        for (; a[i] < s; i++);
```

```
        for (; a[j] > s; j--);
        if (i <= j) {
            w = a[i]; a[i] = a[j]; a[j] = w;
            i++; j--;
        }
    }
    if (k <= j) selection(p,j,k);
        else    selection(i,q,k);          } ①
}
```

(40) 177, 下13　　[k-1]：前半部分列　→　[j]：前半部分列

(41) 177, 下12　　(この行、削除)

(42) 177, 下11　　a[k+1],…　→　a[i],…

(43) 177, 下2－下1　　また，部分列　……　処理する。選択処理では，
→　選択処理では，

(44) 181, 12　　最も単純な　→　最も素朴な

(45) 198, 9　　である.　→
である．P を指していたポインタも更新する必要があることに注意されたい.

(46) 209, 1　　1：g＝1：(1+g)　→　1：g ＝ g：(1+g)

(47) 213,解図 5･A,,下 1 行目　　{M}　→　[M]

(48) 213,下 2 行目　　MAX　→　M

(49) 214,解図 5･B　を下記のプログラムで置き換えてください:

```
void breadth_first()
{   int v,w;
    v = dequeue();
    if (v != -2)
    {   printf(" %d =>",v);
        for (w = 0; w < N; w++) {
            if (graph[v][w] == 1 && t[w] == NON)
                {t[w]=v; enqueue(w);}
        }
        breadth_first();
    }
}
```

(50) 230,右　　(最下行として，下記の行を追加して下さい)

-> *30, 34*

以　上