プログラミング創造演習・2022 年度問題

解答期限: 2022 年 4 月 13 日 (水) 17:00

以下の問題に答えよ．本や Web サイトで調べたりプログラムを書いて確かめたりしてもよいが，各自自力で解くこと．

1　ある再帰関数

a をある定数として、以下で定義される関数 $f\colon \mathbb{Z} \to \mathbb{Z}$ を考える（ただし、$\mathbb{Z}$ は整数全体の集合をあらわす)。

$$f(n) = \begin{cases} n - a & (n > a^2) \\ f(f(n + a + 1)) & (n \leq a^2) \end{cases}$$

また、$f(n)$ の値を上記の f の定義に従って計算する場合の f の呼び出し回数を $g(n)$ とおく。たとえば、$n > a^2$ のとき $g(n) = 1$ であり、また $g(a^2) = 3$ である。

このとき、$a = 101$ として以下の問いに答えよ。

問 1　$f(a^2)$ を求めよ。

問 2　$f(n) = f(0)$ を満たす整数 $n \geq 0$ の個数を求めよ。

問 3　$g(0)$ を求めよ。

問 4　$g(-1234567890)$ を求めよ。

1.1　入出力例

なお、問 1 から問 4 において $a = 10$ とした場合の回答は以下の通りである。

問 1　91

問 2　102

問 3　203

問 4　2469135983

2　二分木の走査

図 1 のような二分木を表す C のデータ構造を考えよう．ここでは二分木の頂点をコード 1 に示す構造体 bnode で，辺（有向辺）を構造体間のポインタとして表すことにする．構造体 bnode のフィールド v は頂点のラベル（頂点に付与されるデータ）を，lc および rc は左右の子を表す構造体へのポインタをそれぞれ格納する．図 1 の例では 8 個の頂点それぞれに整数値 1〜8 がラベルとして付与されている．以降，ラベル n が付与された頂点を頂点 n と呼ぶ．頂点 1〜8 を表す構造体 bnode 型の変数 n1〜n8 はコード 2 のように定義できる．例えば n1 の定義では，構造体のフィールド v は 1 に，lc および rc はそれぞれ n2 および n3 へのポインタに初期化される．また，n4〜n8 では lc や rc の値を NULL とすることで子の不在を表している．

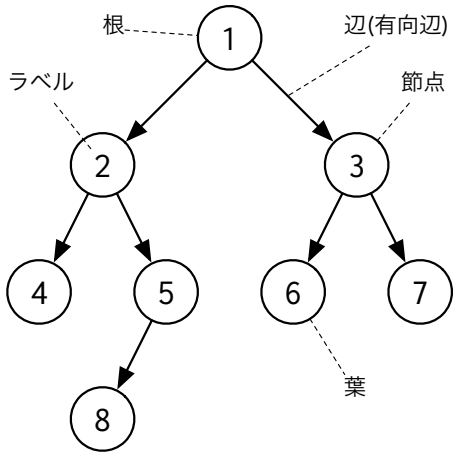

図 1　二分木の例

コード 1　頂点を表す構造体

```
struct bnode {
    int v;                  // ラベル
    struct bnode *lc, *rc;  // 左右の子
};
```

コード 2　図 1 の頂点 1〜8 を表す変数の定義

```
struct bnode n8 = { 8, NULL, NULL };
struct bnode n7 = { 7, NULL, NULL };
struct bnode n6 = { 6, NULL, NULL };
struct bnode n5 = { 5, &n8,  NULL };
struct bnode n4 = { 4, NULL, NULL };
struct bnode n3 = { 3, &n6,  &n7  };
struct bnode n2 = { 2, &n4,  &n5  };
struct bnode n1 = { 1, &n2,  &n3  };
```

2.1　再帰による二分木の走査

木の頂点を順番に訪れて何らかの処理をすることを木の走査 (traversal) という．木の頂点全てを検査して特定の条件をみたすものを探し出したり，頂点に付与された値（ラベル）の総和を求めるといった処理は走査の例である．ここでは簡単な例として，頂点のラベルを順に出力する処理を扱う．

コード 3　行きがけ順の走査

```
1 void trav_pre(struct bnode *t) {
2     if (t != NULL) {
3         printf("%d", t->v);
4         trav_pre(t->lc);
5         trav_pre(t->rc);
6     }
7 }
```

コード 4　通りがけ順の走査

```
1 void trav_in(struct bnode *t) {
2     if (t != NULL) {
3         trav_in(t->lc);
4         printf("%d", t->v);
5         trav_in(t->rc);
6     }
7 }
```

コード 3 は再帰呼び出しを使った二分木を走査する関数の例である．ここで定義されている関数 trav_pre

は，引数として与えられた頂点（正確には頂点を表す bnode 構造体へのポインタ）のラベルを最初に出力したのち，その頂点の左の子と右の子についてそれぞれ再帰的に処理する．このような走査の順番を行き掛け順 (pre-order) と呼ぶ．一方，コード 4 で定義されている関数 trav_in の場合，最初に引数として与えられた頂点の左の子について再帰的に処理した後，引数として与えられた頂点のラベルを出力し，最後にその頂点の右の子について再帰的に処理する．このような走査の順番は通りがけ順 (in-order) と呼ばれる．

問 5　`trav_pre(&n1);` を実行したときの出力を示せ．

問 6　`trav_in(&n1);` を実行したときの出力を示せ．

問 7　`trav_pre(&n1);` を実行したときの関数 trav_pre の呼び出し回数を示せ．

2.2　再帰を使わない二分木の走査

再帰を使わずに二分木の走査はできないだろうか．コード 3 は 4 行目と 5 行目で計 2 回再帰呼び出しをおこなっているので，関数の動作を変えずにこれらを除去してみよう．

2.2.1　末尾再帰の除去

最初にコード 3 の 5 行目の再帰呼び出しについて考えてみる．この再帰呼び出しは関数 trav_pre の最後の処理（trav_pre の呼び出しから戻る直前の処理）である．このような再帰は末尾再帰 (tail recursion) と呼ばれる．末尾再帰は関数の先頭にジャンプすることで代用可能であり，したがって容易に除去できる．コード 5 が末尾再帰を除去した結果であり，コード 3 の 5 行目の再帰呼び出しの引数 t->rc を引数 t に代入して関数の先頭に **goto** 文[*1]でジャンプしている．コード 5 を **goto** 文を使わないよう整理したものがコード 6 である．

コード 5　2 番目の再帰を除去（末尾再帰の除去）

```
1 void trav_pre(struct bnode *t) {
2 start:
3     if (t != NULL) {
4         printf("%d", t->v);
5         trav_pre(t->lc);
6         t = t->rc;
7         goto start;
8     }
9 }
```

コード 6　コード 5 を整理

```
1 void trav_pre(struct bnode *t) {
2     while (t != NULL) {
3         printf("%d", t->v);
4         trav_pre(t->lc);
5         t = t->rc;
6     }
7 }
```

問 8　コード 4 で定義されている関数 trav_in の末尾再帰を除去して，コード 6 のような（**goto** 文を使わない）形にして示せ．

2.2.2　スタックを用いた再帰の除去

コード 6 で定義されている関数 trav_pre にはまだ再帰呼び出しが残っているので，これも除去したいが，これは末尾再帰ではないので前節でやったようにはいかない．どうするか．

[*1] コードのスパゲティ化の原因のひとつとして忌み嫌われる **goto** 文であるが，この例のようなコードの機械的変形を行う際にはしばしば登場する．

とりあえずコード 6 の 4 行目を，コード 5 の 6,7 行目と同様に変形してみる．具体的には t に t->lc を代入したのちに関数の先頭（ラベル start をつけた部分）に **goto** 文でジャンプするようにする．ただしこれは末尾再帰ではないので，コード 6 の 5 行目にある代入文 t = t->rc; の位置に戻って来る必要がある．そこでこの代入文の直前に resume というラベルを付けて戻って来られるようにする．ところがこの位置に戻ってきたとして，そのときの t の値は上で t->lc を代入する前の値でなければならない．そこで t に t->lc を代入する前に t の値をどこかにセーブしておくことになる．本のコード（コード 6）で再帰が行われる度にこの処理が必要なので，セーブしておく場所は単なる変数ではなくスタックにする必要がある．そこで，t と同じデータ型を要素の型とする配列 stack を用意と，push/pop する位置を表す変数 sp を用意する．t の値をスタックに push したいときは stack[sp++] = t; を，スタック上の値を pop して t に戻したいときは t = stack[--sp]; をそれぞれ実行すればよい．**while** ループをぬけて関数から戻る際，スタックにまだ値が積まれていたら pop して t に戻し，ラベル resume の位置にジャンプする．以上を実現したのがコード 7 であり，それを整理して **goto** 文を除去したものがコード 8 である．

コード 7　スタックを用いた再帰の除去

```
1  void trav_pre(struct bnode *t) {
2      struct bnode* stack[STACK_SIZE];
3      int sp = 0;
4  start:
5      while (t != NULL) {
6          printf("%d", t->v);
7          stack[sp++] = t; // PUSH
8          t = t->lc;
9          goto start;
10 resume:
11         t = t->rc;
12     }
13     if (sp > 0) {
14         t = stack[--sp]; // POP
15         goto resume;
16     }
17 }
```

コード 8　コード 7 を整理

```
1  void trav_pre(struct bnode *t) {
2      struct bnode* stack[STACK_SIZE];
3      int sp = 0;
4      for (;;) {
5          while (t != NULL) {
6              printf("%d", t->v);
7              stack[sp++] = t; // PUSH
8              t = t->lc;
9          }
10         if (sp == 0) return;
11         t = stack[--sp];     // POP
12         t = t->rc;
13     }
14 }
```

問 9　コード 8 で定義された関数 trav_pre について，trav_pre(&n1); の実行中にに変数 sp が取り得る値の最大値を示せ．

問 10　問 8 で作成した，末尾再帰を除去した trav_in について，コード 8 のようにスタックを用いて再帰呼び出しを除去した（**goto** 文を使わない）形にして示せ．

問 11*　コード 9 に示すのは，帰りがけ順 (post-order) による二分木の走査を再帰を用いて定義した関数 trav_post である．この関数をスタックを用いて再帰を含まない形にせよ（**goto** 文は含んでいても構わない）．

コード 9　帰りがけ順の走査

```
1  void trav_post(struct bnode *t) {
2      if (t != NULL) {
3          trav_pre(t->lc);
4          trav_pre(t->rc);
5          printf("%d", t->v);
6      }
7  }
```

2.3 再帰もスタックも使わない二分木の走査

やや特殊なやり方であるが，コード 10 のようにすると再帰もスタックも用いずに二分木の走査ができる．

コード 10　行き掛け順の走査

```
1  void trav_pre(struct bnode *t) {
2      while (t != NULL) {
3          if (t->lc == NULL) {
4              printf("%d", t->v);
5              t = t->rc;
6          }
7          else {
8              struct bnode *rm = t->lc;
9              while (rm->rc != NULL && [ A ] ) {
10                 rm = rm->rc;
11             }
12             if (rm->rc == NULL) {
13                 printf("%d", t->v);
14                 rm->rc = t;
15                 t = t->lc;
16             }
17             else {
18                 rm->rc = NULL;
19                 t = t->rc;
20             }
21         }
22     }
23 }
```

問 12* コード 10 の [A] に入る式を示せ．

問 13* コード 10 と同様にして通りがけ順で走査を行う関数 `trav_in` を示せ．