アルゴリズムとデータ構造1　　演習問題 (2)　　回答

下の漸化式の値を, $n = 2^k$ ($k = 1,2,\cdots$) なる各 k に対して, log-logのグラフに$T(n)$ の値をプロットするためのプログラム.

$$T(n) = \begin{cases} 2 \cdot T\left(\frac{n}{2}\right) + c \cdot n^2 & : \quad n > 1 \\ 2 \cdot c & : \quad n = 1 \end{cases}$$

```
main ( )
    /*  n = 2^k (k = 1, 2, ⋯, 20) なる各kに対して, 漸化式T(n) を計算し, 配列T[・] に入れる  */
    /*  ここで, c は入力する  */
{
        typedef  struct {
                        float     size ;        /*  欄 size は n = 2^k の値を実数で持つ  */
                        int       time ;        /*  欄 time は T(n) の値を持つ  */
                        float     ltime ;       /*  欄 ltime は log_{2}T(n) の値を実数で持つ  */
                } GCELL ;
        maxlength = 20 ;

        GCELL   T[1..maxlength] ;               /*  配列 T[・] の各要素は GCELL 型の構造体  */
        int     k ;                             /*  1 ～ maxlength の繰り返しに用いる制御変数  */
        int     n ;                             /*  n = 2^k の値  */
        int     c ;                             /*  漸化式に現れる定数 c  */

        正整数を一つ入力せよと指示を出す ;
        入力された正整数を c とする ;

        n  :=  2 ;
        for ( 1 ≦ k ≦ maxlength なる各 k.  k の昇順に  ) {
                T[k].time  :=  Recursion(  k,  n,  c  ) ;       /*  T(2^k) の計算  */
                T[k].ltime  :=  log( T[k].time )/ log( 2 ) ;    /*  組込み関数を用いた対数の計算  */
                T[k].size  :=  n ;
                n  :=  n + n ;                                  /*  次の k のための n を計算する  */
        }

        T[・].time の各値を log-log グラフに描画し, その横に T[・].size の値を印刷する ;
} /*  main  */
```

```
int   Recursion(  int  k,  int  n,  int  c  )
    /*  漸化式 T(n) = 2·T(n/2) + c·n^2 を計算する再帰的関数. n = 2^k である  */
{
    int     time ;                              /*  T(n) の値を計算するための局所変数  */

    if (  k = 0  )  time  :=  2*c ;
    else            time  :=  2*Recursion(  k–1,  n/2,  c  ) + c*n*n ;      /*  漸化式の計算  */
                    /*  if あるいは else の中で実行する文が一つの場合, { } を省略できるので, 省略している  */
    return  time ;
} /*  Recursion  */
```

このプログラムでは, *Recursion*(1, 2, *c*) が全体で *maxlength* 回も呼ばれているため, 無駄な計算が繰り返されている. そこで, 一度計算した *T*(2^*k*) の値を何回も計算しないようにすれば, プログラムの効率は上がる. この値は *T*[*k*].*time* に蓄えられているのであるから, それを利用すれば, 再帰的に何回も計算しなくてもよい. そのようなプログラムとして, 下記のものが考えられる.

```
main ( )
    /*  n = 2^k (k = 1, 2, ⋯, 20) なる各kに対して, 漸化式T(n) を計算し, 配列T[・] に入れる */
    /*  ここで, c は入力する  */
{
        typedef  struct {
                        float     size ;        /* 欄 size は n = 2^k の値を実数で持つ */
                        int       time ;        /* 欄 time は T(n) の値を持つ */
                        float     ltime ;       /* 欄 ltime は log_{2}T(n) の値を実数で持つ */
                } GCELL ;
        maxlength = 20 ;

        GCELL   T[1..maxlength] ;               /* 配列 T[・] の各要素は GCELL 型の構造体 */
        int     k ;                             /* 1 ～ maxlength の繰り返しに用いる制御変数 */
        int     n ;                             /* n = 2^k の値 */
        int     c ;                             /* 漸化式に現れる定数 c */

        正整数を一つ入力せよと指示を出す ;
        入力された正整数を c とする ;

        T[0].time  :=  2*c ;
        n  :=  2 ;
        for ( 1 ≦ k ≦ maxlength なる各 k.  k の昇順に ) {
                T[k].time  :=  2*T[k–1].time + c*n*n ;      /* T(2^k) の計算 */
                T[k].ltime  :=  log( T[k].time )/ log( 2 ) ;  /* 組込み関数を用いた対数の計算 */
                T[k].size  :=  n ;
                n  :=  n + n ;                              /* 次の k のための n を計算する */
        }

        T[・].time の各値を log-log グラフに描画し, その横に T[・].size の値を印刷する ;
} /*  main  */
```