平成１８年度　春期　ソフトウェア開発技術者　午後Ⅱ問題

問１　大きなけた数の整数の演算操作に関する次の記述を読んで，設問 1～3 に答えよ。

与えられた n けたの整数に対し，数字を並べ替えてできる最大の整数から，同様にしてできる最小の整数を引いて n けたの整数を求める操作を，カプレカ操作と呼ぶ。求めた結果が n けたより小さいときは，上位けたに不足けた数分の 0 を補い，常に n けたの整数として扱う。n けたの整数は有限個しか存在しないので，与えられた整数から始めて，カプレカ操作を繰り返すと，いつかは同じ数が現れる。

例えば，4 けたの整数 1234 から作ることのできる最大数 4321 から，最小数 1234 を引くと 3087 となる。このあと，カプレカ操作を繰り返すと次のように変化する。

8730 － 0378 ＝ 8352
8532 － 2358 ＝ 6174
7641 － 1467 ＝ 6174

6174 に対するカプレカ操作の結果は 6174 となり，これ以上操作を繰り返しても値は変わらない。

すべてのけたが同じ整数は 1 回のカプレカ操作で 0 になり，以後変化しないことは明らかである。それ以外は，どの 4 けたの整数もカプレカ操作を何回か繰り返すと 6174 になることが分かっている。このように，4 けたの場合には一つの定数になるが，4 けた以外の場合には定数になるとは限らない。例えば，12345 は 54321 － 12345 ＝ 41976 となり，次のように変化し，四つの数を循環するようになる。このような状態をサイクルという。

97641 － 14679 ＝ 82962
98622 － 22689 ＝ 75933
97533 － 33579 ＝ 63954
96543 － 34569 ＝ 61974
97641 － 14679 ＝ 82962

どの整数もカプレカ操作を繰り返すと，定数かサイクルに到達する。

整数型の変数では表現できないような大きなけた数の整数に対して，定数かサイクルになるまでカプレカ操作を繰り返して，途中経過を表示するプログラムを次のように設計した。

（1）整数の表現法 1

整数型の変数では表現できない大きなけた数の整数を扱うので，1 けたずつ整数の配列に入れる。けたの順序は意味がないので，常に昇順にソートしておく。こうすることでサイクルの確認も容易になる。扱うことのできる整数の上限は，配列の大きさで決まる。

（2）カプレカ操作

(1)の表現に対して演算を行う。既に昇順にソートされている整数の数字を逆順にすることで，降順にソートした整数を作り，その差を求める。結果を昇順にソートする。

（3）サイクルの検出

与えられた整数とカプレカ操作で求められた整数はすべてリストに記憶し，新たに求めた整数が

既に出現していないかを調べるのに使う。

〔プログラムの概略〕

（1）整数を配列Nに読み込み，各けたの数字を昇順にソート
（2）リストLを空リストに初期化
（3）LにNがない間
　（3-1）LにNを追加
　（3-2）Nにカプレカ操作を行い，結果をRに入れ，Rを表示
　（3-3）RをNに移動

カプレカ操作の関数kaprekarとサイクルの検出に使う比較関数compareを,図1に示す。ここで,number，number2，result，d1，d2は整数を表現する配列で，添字は0から始まるものとする。また，けたは下位けたから順に0，1，2，…と数える。

関数kaprekarは配列numberを受け取って，カプレカ操作の結果を配列resultに返す。nは受け取った整数のけた数である。関数compareは二つの整数を配列d1，d2で受け取って，二つの整数が同じなら1を，異なっていれば0を返す。nは受け取った整数のけた数である。

```
kaprekar(number, result, n)
{
   for i=0 to n-1     /*ループA(number2に最小数を入れる)*/
     number2[   a   ] = number[i]
   borrow = 0
   for i=0 to n-1     /*ループB(nけたの引き算)*/
   {
      result[i]= [   b   ]      /*iけた目を計算*/
      if (result[i]>=0 )  borrow = 0
      else
      {
         result[i]= [   c   ]
         borrow = 1
       }
    }
    for i=0 to n-2     /*ループC(選択法で昇順に並べ替える)*/
    {
        k = i
        for j=i+1 to n-1
            if(   d   ) k = j
        w = result[i]
        result[i] = result[k]
        result[k] = w
     }
}
compare (d1, d2, n)
{
    for i=0 to n-1
         if(not(d1[i]=d2[i]))  return 0
    return 1
}
```

図１　関数 kaprekar（その 1）と関数 compare

配列の要素に数字を 1 けたずつ入れる表現法 1 は，けた数が大きくなるとメモリの利用効率が極端に悪くなる。また，事前に想定したけた数を超えると，配列の大きさを変えなければならない。そこで，プログラムの設計を根本から見直し，新たに次のような整数の表現法 2 を用い，処理の高速化と記憶領域の縮小を図った。

〔整数の表現法 2〕

カプレカ操作では，各数字が何個あるかが分かれば計算できるので，各数字が何個出てきたかを要素数 10 の整数の配列 d で表現する。

この表現法で各数字の出現頻度が，

```
d[0] = d[1] = 2,
d[2] = d[3] = d[4] = d[5] = 1,
d[6] = d[7] = 0,
d[8] = 3,
d[9] = 1
```

であったとすると，これによって表される数は 12 けたの整数で，

最大数 ＝ 988854321100,

最小数 ＝ 001123458889

となる。

表現法 2 を使って，n けたの整数のカプレカ操作を行うプログラムを考える。ここで，i けた目の計算で，最大数の i けた目の数字を DR，最小数の i けた目の数字を DL とする。また，BR には，DR でない数字が初めて現れるけた位置を設定する。BL も同様に DL でない数字が現れるけた位置を設定する。最後の数字の場合は，数字がなくなる位置（n）を設定する。

表現法 2 で与えられた上記のデータに対して，実際にカプレカ操作を試してみると，次のようになる。

（1）0 けた目の計算（図 2）

最大数の 0 けた目は 0 なので DR は 0，最小数の 0 けた目は 9 なので DL は 9 となる。0 は 2 個あるので BR は 2，9 は 1 個なので BL は 1 になる。0 けた目の計算は，DR － DL ＝ 0 － 9 ＝ －9 となるので，上位のけたからの繰下りが起き，0 けた目は 1 となる。

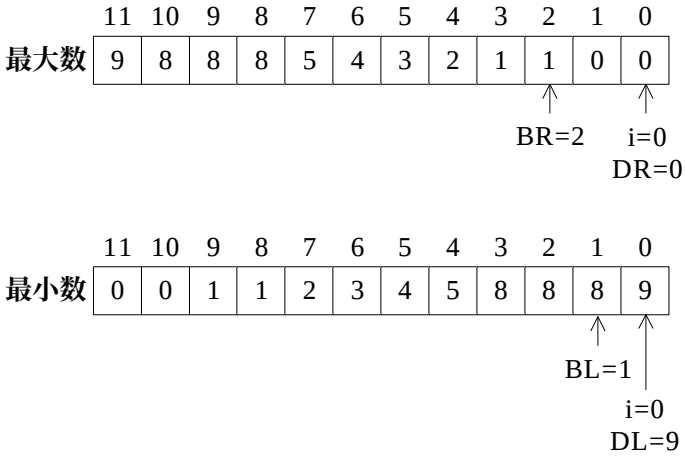

図２　0 けた目の計算

（2）1 けた目（i＝1）の計算（図 3）

i＜BR なので，BR，DR は変わらない。

i＝BL なので，DL は 8 に変わる。DL が変わると BL も更新する必要がある。8 は 3 個あるので，次に DL が変わるのは 4 けた目である。したがって，BL は 4 となる。

1けた目の計算は，下位のけたに繰下りがあり，DR－DL－1 ＝ 0－8－1 ＝ －9となるので，上位のけたからの繰下りが起き，1けた目は1となる。

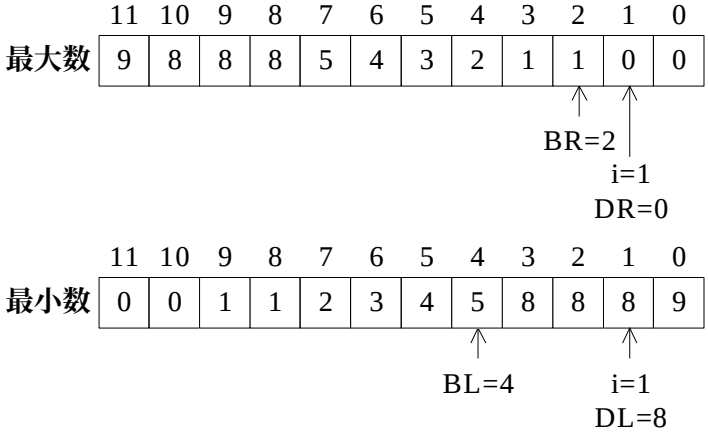

図３　1けた目の計算

（3）2～n－1けた目（i ＝ 2～n－1）に対して，同様の操作を繰り返す。

設問１　図1中の［　a　］～［　d　］に入れる適切な式を答えよ。

設問２　図1に示した関数kaprekarと関数compareを利用し，一つの整数に対してサイクルを検出するまでカプレカ操作を繰り返すプログラムの平均時間計算量に関する次の記述中の［　e　］～［　k　］に入れる適切な式を，m，nを用いて答えよ。ここで，整数のけた数をn，サイクルを検出するまでにリストに入れる整数の平均個数をmとする。

ループA，Bの計算量はO（［　e　］），ループCの計算量はO（［　f　］）である。したがって，関数kaprekarの計算量はO（［　g　］）となる。サイクルを検出するまでこれを繰り返すので，カプレカ操作に要する計算量はO（［　h　］）となる。

各カプレカ操作の後で行うサイクルの検出は，求めた整数とリスト中のすべての整数とを比較して，一致するものがあるかを探す操作である。関数compareの1回当たりの計算量はO（［　i　］）である。サイクルを検出するまでの関数compareの実行回数は［　j　］に比例するので，サイクルの検出にする計算量はO（［　k　］）となる。

プログラム全体の計算量は，カプレカ操作の計算量とサイクル検出の計算量のうち，大きい方の値になる。

設問３　整数の表現法 2 に合わせて書き換えた関数 kaprekar を，図 4 に示す。ここで，関数 kaprekar は，カプレカ操作の結果を整数の表現法 2 によって配列 result に返す。図 4 中の [l] ～ [s] に入れる適切な式を答えよ。

```
kaprekar(d, result, n)
{
    for i=0 to 9
        result[i] = 0
    DR = 0
    DL = [ l ]
    BR = d[DR]
    BL = d[DL]
    borrow = 0
    for i=0 to n-1
    {
        while(i>=BR)
        {
            DR = [ m ]
            BR = [ n ]
        }
        while(i>=BL)
        {
            DL = [ o ]
            BL = [ p ]
        }
        r = DR – DL - borrow
        borrow = 0
        if(r<0)
        {
            r = [ q ]
            borrow = [ r ]
        }
        result[r] = [ s ]
    }
}
```

図４　関数 kaprekar（その 2）