アルゴリズム論
Theory of Algorithms

第３回講義
アルゴリズムの設計と解析の基礎(3)

アルゴリズム論
Theory of Algorithms

Lecture #3

Foundation of Design and Analysis of Algorithms(3)

問題P5: n個のデータが配列a[]に与えられているとする.
実数w>0が与えられたとき，その中の要素の最大値と最小値の差が高々wであるような最長の区間[p,q]を求めよ.

max(p,q) = max{a[p], a[p+1], ... , a[q]}
min(p,q) = min{a[p], a[p+1], ... , a[q]}
とすると，上の条件は
max(p,q) - min(p, q) ≦w
と表すことができる.

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

w=23なら，[0,9]が最長区間
w=20なら，[0,7]が最長区間
w=12なら，[2,7]が最長区間

Problem P5: Suppose that n data are stored in an array a[].
Given a real number w>0, find a longest interval [p,q] in which the difference between the maximum and minimum is at most w.

max(p,q) = max{a[p], a[p+1], ... , a[q]}
min(p,q) = min{a[p], a[p+1], ... , a[q]}
Then, the condition is as follows:

$$\max(p,q) - \min(p, q) \leqq w$$

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

for w=23, [0,9] is the longest interval
for w=20, [0,7] is the longest interval
for w=12, [2,7]is the longest interval

腕力法
すべての区間を列挙して，幅wの条件を満たすかどうかを調べる

アルゴリズムP5-A0:

```
maxlength=0;
for(p=0; p<n-1; p++)
  for(q=p+1; q<n; q++){
    max=a[p]; min=a[p];
    for(i=p+1; i<=q; i++){
      if(a[i]>max) max=a[i];
      if(a[i]<min) min=a[i];
    }
    if(max-min<=w && q-p+1>maxlength)
      maxlength = q-p+1;
  }
cout << maxlength;
```

計算時間は
$O(n^3)$
どこまで改善できる？

Brute-force algorithm

Enumerate all intervals and check the condition on width w.

Algorithm P5-A0:

```
maxlength=0;
for(p=0; p<n-1; p++)
  for(q=p+1; q<n; q++){
    max=a[p]; min=a[p];
    for(i=p+1; i<=q; i++){
      if(a[i]>max) max=a[i];
      if(a[i]<min) min=a[i];
    }
    if(max-min<=w && q-p+1>maxlength)
      maxlength = q-p+1;
  }
cout << maxlength;
```

Computation time

$O(n^3)$

How much improvement is possible?

区間の左端pを固定してみよう.
　区間の右端qをどこまで延ばせるかを調べる.

アルゴリズムP5-A1:

```
maxlength=0;
for(p=0; p<n-1; p++){
  max=a[p]; min=a[p];
  for(q=p+1; q<n; q++){
    if(a[q]>max) max=a[q];
    if(a[q]<min) min=a[q];
    if(max-min>w) break;
  }
  if(q-p>maxlength) maxlength = q-p;
}
cout << maxlength;
```

計算時間は
$O(n^2)$

Fix the left endpoint p of an interval.

Extend the right endpoint q as far as possible.

Algorithm P5-A1:

```
maxlength=0;
for(p=0; p<n-1; p++){
   max=a[p]; min=a[p];
   for(q=p+1; q<n; q++){
      if(a[q]>max) max=a[q];
      if(a[q]<min) min=a[q];
      if(max-min>w) break;
   }
   if(q-p>maxlength) maxlength = q-p;
}
cout << maxlength;
```

Computation time

$O(n^2)$

分割統治法の利用

左半分で問題を解く
右半分で問題を解く
中央を含む区間で最長のものを求める.
上記3つの区間の中で最長のものを出力.

アルゴリズムP5-A2:
最初に，中央mから左右でw以内の最長区間を求める.
次に，q=m, m+1, ...について，区間[m, q]での最大値と最小値を求め，配列に記録しておく.
次に，中央mから左に区間を延ばしていく.
このとき，wの条件を満たさなくなれば，区間の右端qを適宜減らす．右端qが中央を越えるか，左端pが配列の端に到達すれば終了.
それまでに調べた区間の中で最長のものを出力する.

Algorithm based on divide-and-conquer

- solve the problem for the left half
- solve the problem for the right half
- find the longest interval containing the middle
- output the longest one among the above three

Algorithm P5-A2:

First, find a longest interval within w from the middle m to the right. At the same time find maximum and minimum in the interval [m, q] for q=m, m+1, ... and store them in arrays.

Next, extend the interval from the middle m to the left.

Then, if the width condition is not satisfied, the right endpoint q is decreased appropriately. If the right endpoint q exceeds the middle or the left endpoint p reaches the margin of the array, then stop.

Output the longest interval among ones found so far.

アルゴリズムP5-A2:

```
int findmaxlength(left, right){
  m = (left+right)/2;
  L1=findmaxlength(left, m-1);
  L2=findmaxlength(m+1, right);
  qmax[m]=qmin[m] = a[m];
  for(q=m+1; q<n; q++)
    qmax[q]=max(qmax[q-1], a[q]);
    qmin[q]=min(qmin[q-1], a[q]);
    if(qmax[q]-qmin[q]>w) break;
  }
  q=q-1;
  L3 = q-m+1;
  for(p=m-1; p>=0; p--){
    do{
      pmax=max(qmax[q], a[p]);
      pmin=min(qmin[q], a[p]);
      if(pmax-pmin>w)q=q-1;
    }while(pmax-pmin>w);
    if(q-p+1>L3) L3=q-p+1;
  }
  return max(L1, L2, L3);
}
main()
{
  ....
  cout << findmaxlength(0, n-1);
}
```

練習問題：計算時間はO(n log n)であることを証明せよ.

練習問題：実際にプログラムを作って動作を確かめよ.

Algorithm P5-A2:

```
int findmaxlength(left, right){
  m = (left+right)/2;
  L1=findmaxlength(left, m-1);
  L2=findmaxlength(m+1, right);
  qmax[m]=qmin[m] = a[m];
  for(q=m+1; q<n; q++)
    qmax[q]=max(qmax[q-1], a[q]);
    qmin[q]=min(qmin[q-1], a[q]);
    if(qmax[q]-qmin[q]>w) break;
  }
  q=q-1;
  L3 = q-m+1;
  for(p=m-1; p>=0; p--){
    do{
      pmax=max(qmax[q], a[p]);
      pmin=min(qmin[q], a[p]);
      if(pmax-pmin>w)q=q-1;
    }while(pmax-pmin>w);
    if(q-p+1>L3) L3=q-p+1;
  }
  return max(L1, L2, L3);
}
main()
{
  ....
  cout << findmaxlength(0, n-1);
}
```

Exercise : Prove that the computation time is O(n log n).

Exercise : Implement the algorithm as a program to see its behavior.

連結リストを用いたアルゴリズム

区間[p,q]がmax(p,q)-min(p,q)≦wの条件を満たすとき，この区間[p,q]は*feasible*(実行可能）であるという.
a[0]からa[n-1]に向けて走査して行き，最長の実行可能区間を求めることができる.

a[q]に注目しているとき，qを右端とする実行可能区間で最長のものを[p, q]とする．i.e.,
　[p,q]は実行可能，[p-1,q]は実行可能でない.
cur_min=min{a[p], a[p+1], ... , a[q]}
cur_max=max{a[p], a[p+1], ... , a[q]}
と定義する.

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

w=12:
[2,7]は実行可能
cur_min=16
cur_max=28

Algorithm using linked list

When an interval [p,q] satisfies the condition max(p,q)-min(p,q)≦w then this interval [p,q] is called *feasible*.
We can find a longest feasible interval by sweeping the array from a[0] to a[n-1].

At a[q], let [p,q] be the longest feasible interval with its right end at q, i.e.,
[p,q] is feasible and [p-1,q] is not
We define
cur_min=min{a[p], a[p+1], ... , a[q]}
cur_max=max{a[p], a[p+1], ... , a[q]}.

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

w=12:
[2,7] is feasible
cur_min=16
cur_max=28

現在の区間[p,q]において

min_after[i] = min{ a[j] | i≦j≦q}, i=p, ... , q

と定義する.（区間内でi番目以降の最小値を表す）

min_after[]は単調非増加である.

区間[p,q]内で, min_after[]の値で真に異なるものを順に連結したものを$\text{min_after}[i_1] > \text{min_after}[i_2] > \ldots > \text{min_after}[i_k]$とする.

同様に, max_after[i]=max{a[j]|i≦j≦q}, i=p,..., qも定義する.

これらにより, リストLIST_minとLIST_maxを定義する.

現在の区間[2,6], w=12

	0	1	2	3	4	5	6	7	8	9	
a	17	32	19	22	28	16	18	20	39	31	
min_after[]			16	16	16	16	18				リストLIST_min
max_after[]			28	28	28	18	18				リストLIST_max

At the current interval [p, q] we define

min_after[i] = min{ a[j] | i≦j≦q}, i=p, ... , q.

（to indicate the minimum in the interval after the i-th element)

min_after[] is monotonically nonincreasing.

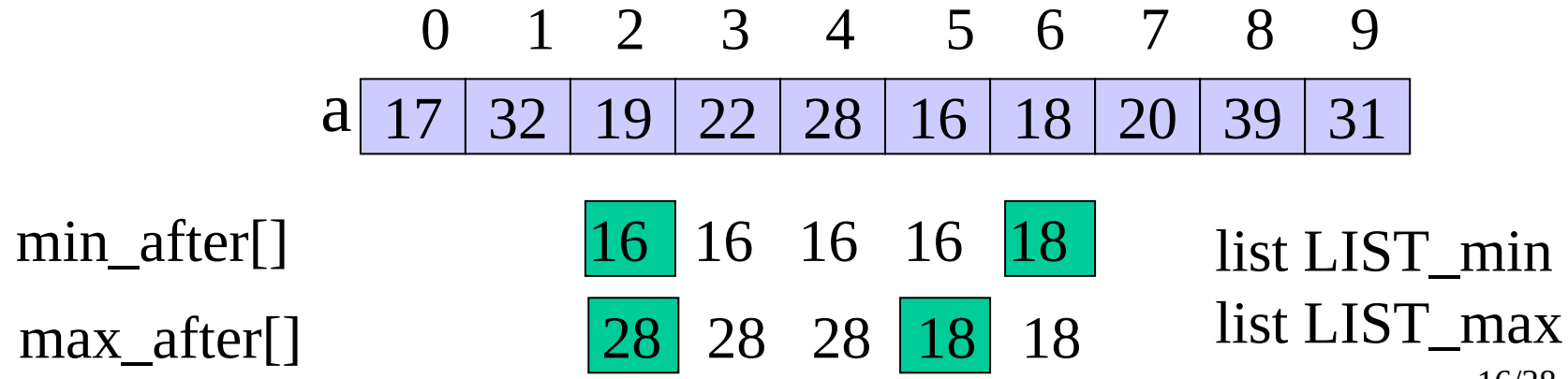
We connect values min_after[] by skipping the same values in the interval [p,q]. Let the resulting list be

min_after[i_1]>min_after[i_2]>...>min_after[i_k].

Similarly we define max_after[i]=max{a[j]|i≦j≦q}, i=p,..., q.

Using them we define a list LIST_minとLIST_max.

current interval [2,6], w=12



現在の区間[2,6], w=12

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

min_after[]	16	16	16	16	18		リストLIST_min
max_after[]	28	28	28	18	18		リストLIST_max

次にq=7を右端とする区間を考える.

LIST_minを右から左に辿り，a[q]より大きい要素を削除

LIST_maxを右から左に辿り，a[q]より小さい要素を削除

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

min_after[]	16				18		リストLIST_min
max_after[]	28					20	リストLIST_max

この場合は区間の右端を延長可能

current interval [2,6], w=12

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

min_after[]			16	16	16	16	18			
max_after[]			28	28	28	18	18			

list LIST_min

list LIST_max

Next, consider the interval with its right end at q=7.

Following LIST_min from right to left, delete elements > a[q].

Following LIST_max from right to left, delete elements < a[q].

	0	1	2	3	4	5	6	7	8	9
a	17	32	19	22	28	16	18	20	39	31

min_after[]			16				18			
max_after[]			28					20		

list LIST_min

list LIST_max

In this case we can extend the right end.

ケース1：a[q+1]を加えても実行可能であるとき
　このときは，[p,q]を[p,q+1]に拡張可能.
ケース2：a[q+1]-cur_min>wのとき，
　LIST_minを左端から右に辿り，$a[q+1]-a[i_k] \leqq w$が成り立つまで
　リストを右に辿り，途中の要素を削除する.
　さらに，区間の左端を$p=i_{k-1}+1$にセットする.
　同様にLIST_maxも更新する(i_{k-1}より左の部分を削除).
　現在の区間を[p,q+1]に更新.
ケース3：cur_max-a[q+1]>wのとき，
　LIST_maxを左端から右に辿り，$a[i_k]-a[q+1] \leqq w$が成り立つまで
　リストを右に辿り，途中の要素を削除する.
　さらに，区間の左端を$p=i_{k-1}+1$にセットする.
　同様にLIST_minも更新する(i_{k-1}より左の部分を削除).
　現在の区間を[p,q+1]に更新.

練習問題：上記のアルゴリズムを実装し，動作を確認せよ.

練習問題：上記のアルゴリズムの計算時間はO(n)であることを示せ

Case 1: interval remains feasible after adding a[q+1]

Then, we can extend [p,q] into [p,q+1].

Case 2: a[q+1]-cur_min>w,

Following LIST_min from its left end to the right, delete all the elements until $a[q+1]-a[i_k] \leqq w$ holds.

Furthermore, we set the left end of an interval at $p=i_{k-1}+1$.

Similarly, update LIST_max(delete the part to the left of i_{k-1}).

Update the current interval to [p,q+1].

Case 3: cur_max-a[q+1]>w,

Following LIST_max from its left end to the right, delete all the elements until $a[i_k]-a[q+1] \leqq w$ holds.

Furthermore, we set the left end of an interval at $p=i_{k-1}+1$.

Similarly, update LIST_min (delete the part to the left of i_{k-1}).

Update the current interval to [p,q+1].

Exercise: Implement the above algorithm to see its behavior.

Exercise: Show that the algorithm runs in O(n) time.

正整数の逆数の計算

問題P6：正整数 n に対して，その逆数 1/n を正確に計算せよ．
循環小数になる場合は，循環の開始場所も出力で指定すること．

例：循環部分の開始場所を#で示す

1/7 = 0.#142857
1/11 = 0.#09
1/13 = 0.#076923
1/17 = 0.#0588235294227647
1/28 = 0.03#571428
1/104 = 0.009#615384

余りを10倍して，n で割った余りを求める，
という操作を，余りが 0 になるか，以前に求めたのと同じ余りが出るまで続ける．

1/7の計算

10 / 7 = 1 ... 3
30 / 7 = 4 ... 2
20 / 7 = 2 ... 6
60 / 7 = 8 ... 4
40 / 7 = 5 ... 5
50 / 7 = 7 ... 1

以前に出た余り

Computation of reverse of a positive integer

Problem P6 : Given a positive integer n, calculate its reverse 1/n exactly. If it is a repeated decimal, specify where it starts.

Example : # indicates the start of a repeated decimal.

1/7 = 0.#142857
1/11 = 0.#09
1/13 = 0.#076923
1/17 = 0.#0588235294227647
1/28 = 0.03#571428
1/104 = 0.009#615384

Calculation of 1/7

10 / 7 = 1 ... 3
30 / 7 = 4 ... 2
20 / 7 = 2 ... 6
60 / 7 = 8 ... 4
40 / 7 = 5 ... 5
50 / 7 = 7 ... 1

the residue ever obtained

Repeat the operation to multiply residue by 10 and find residue after dividing it by n until the residue becomes 0 or we encounter the same residue we have ever had before.

方法1：余りを順に蓄える

毎回の操作は次の通り

- 余りを10倍したものをnで割り，商 q と余り r を求め，商 q を出力する．
- 余り r が0なら終了
- そうでなければ，余り r が以前に出現したかどうかを確かめる．
- 以前に出現していれば，その場所が循環の開始場所になる．
- 以前に出現していなければ，現在の余り r を配列に登録する．

アルゴリズムP6-A0:

```
printf(“0.”); r = 1; k=0;
do{
    q = r*10/n;  r = (r*10) % n;
    printf(“%1d”, q);
    t = 0;
    if(r > 0 && k>0)
        for(i=1; i<=k; i=i+1)
            if( a[i] == r) t = 1;
    if(r > 0 && t == 0)
        {k=k+1; a[k] = r;}
}while(r>0 && t==0);
```

Algorithm 1: Keeping the residues in order

Operations at each iteration:

- divide 10 times the residue by n to have quotient q and residue r, and output the quotient q.
- If r is 0 then stop.
- Otherwise, see whether the residue r has ever appeared.
- If it appeared before, it gives the starting position.
- Otherwise, keep the current residue r in an array.

```
Algorithm P6-A0:
printf("0."); r = 1; k=0;
do{
    q = r*10/n;  r = (r*10) % n;
    printf("%1d", q);
    t = 0;
    if(r > 0 && k>0)
        for(i=1; i<=k; i=i+1)
            if( a[i] == r) t = 1;
    if(r > 0 && t == 0)
        {k=k+1; a[k] = r;}
}while(r>0 && t==0);
```

計算時間の解析

- **繰り返し回数の上限は？**

  毎回nで割った余りを求めているが，余りの種類は高々n通りしかないので，
  繰り返し回数も高々n回

- **毎回の繰り返しでの計算時間**

  最も時間がかかるのは，今までに出現した余りと現在の余りとの比較．
  最悪の場合は，同じ余りが見つからない場合
  　このとき，k回目の繰り返しでkに比例する時間がかかる．

- **最悪の場合の計算時間**

  k 回目の繰り返しで最大 k に比例する時間がかかり，
  繰り返しの回数は最大 n だから，全体で必要な時間は，高々

  $$1 + 2 + ... + n = n(n+1)/2$$

  に比例する時間がかかる．

  $O(n^2)$

Analysis of Computation Time

- Upper bound on the number of iterations?

  The number if iterations is at most n since the loop is iterated if we have a new residue but there are at most n different residues.

- **Computation time at each iteration**

  It takes most time to compare the current residue with one obtained so far. The worst case is when the same residue is not found. In this case it takes time proportional to k at the k-th iteration.

- Computation time in the worst case

  The k-th iteration takes time O(k), and the number of iterations is at most n, Thus, the total time for computation is at most

  $1 + 2 + ... + n = n(n+1)/2.$

  $O(n^2)$

高速化

(1)繰り返しの回数を減らす
(2)毎回の繰り返しでの計算を簡単化する

(1)は難しそうなので，(2)について考える

観察

- 余りは 0 から n-1 までの n 通りだけ
- 同じ余りは2度出現しない
- 同じ余りが見つかったとき，何番目に出た余りかを知りたい．

Efficiency Improvement

(1)Decrease the number of iterations.
(2)Simplify computation at each iteration.

Since (1) looks hard, consider (2).

Observation

- There are n different residues, 0, ..., n-1.
- The same residue never happens twice.
- When the same residue is found, we want to know in what order it appeared.

新たなデータ構造

. k 回目に余り r が出たとき,

a[k] = r;

とするのではなく，逆に

a[r] = k;

として，余り r が k 回目に出たことを記録

・最初はすべての r について a[r] = 0として,
余り r はまだ出現していないことを示す.

・このようにすると，余り r がすでに出現したかどうかは,
１回の比較で十分

New Data Structure

. If the residue at k-th iteration is r, we do not set

a[k] = r;

but conversely we record that the residue r appeared at the k-th iteration by setting

a[r] = k;

- We initialize the array by setting

a[r] = 0;

for each r to indicate that r has not appeared as a residue.

- With this method, just one comparison suffices to know whether the residue r has already appeared.

```
for(i=1; i<n; i=i+1)
    a[i] = 0;
a[0] = -1;   /*余り0が出たときに終了できるよう */
a[1] = 1;    /* 余り1は1回目の余り */

r = 1;
for(k=1; k<n; k=k+1){
    b[k] = (r*10) / n;   /* 商を順に蓄える */
    r = (r * 10) % n;
    if(a[r] != 0) break;
    a[r] = k+1;
}
printf("0.");
if(r==0){
    for(i=1; i<=k; i=i+1)   printf("%1d", b[i]);
} else {
    for(i=1; i<a[r]; i=i+1)    printf("%1d", b[i]);
    printf("#");
    for(i=a[r]; i<=k; i=i+1)   printf("%1d", b[i]);
}
```

```
for(i=1; i<n; i=i+1)
    a[i] = 0;
a[0] = -1;   /*to stop when 0 appears as a residue */
a[1] = 1;    /* the residue is initialized as 1 */

r = 1;
for(k=1; k<n; k=k+1){
    b[k] = (r*10) / n;   /* keep quotients in order */
    r = (r * 10) % n;
    if(a[r] != 0) break;
    a[r] = k+1;
}
printf("0.");
if(r==0){
    for(i=1; i<=k; i=i+1)   printf("%1d", b[i]);
} else {
    for(i=1; i<a[r]; i=i+1)    printf("%1d", b[i]);
    printf("#");
    for(i=a[r]; i<=k; i=i+1)   printf("%1d", b[i]);
}
```

1/7の計算例

	0	1	2	3	4	5	6
a	-1	1	0	0	0	0	0

	0	1	2	3	4	5	6	7
b	/	.	.	.	.	.	.	.

k=1, q = 1, r = 3 (10 / 7 = 1... 3)

	0	1	2	3	4	5	6
a	-1	1	0	2	0	0	0

	0	1	2	3	4	5	6	7
b	/	1	.	.	.	.	.	.

k=2, q = 4, r = 2 (30 / 7 = 4 ... 2)

	0	1	2	3	4	5	6
a	-1	1	3	2	0	0	0

	0	1	2	3	4	5	6	7
b	/	1	4	.	.	.	.	.

k=3, q = 2, r = 6 (20 / 7 = 2 ... 6)

	0	1	2	3	4	5	6
a	-1	1	3	2	0	0	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	.	.	.	.

k=4, q = 8, r = 4 (60 / 7 = 8 ... 4)

	0	1	2	3	4	5	6
a	-1	1	3	2	5	0	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	8	.	.	.

k=5, q = 5, r = 5 (40 / 7 = 5 ... 5)

	0	1	2	3	4	5	6
a	-1	1	3	2	5	6	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	8	5	.	.

k=6, q=7, r = 1 (50 / 7 = 7 ... 1)

Example: calculation of 1/7

	0	1	2	3	4	5	6
a	-1	1	0	0	0	0	0

	0	1	2	3	4	5	6	7
b	/	.	.	.	.	.	.	.

k=1, q = 1, r = 3 (10 / 7 = 1... 3)

	0	1	2	3	4	5	6
a	-1	1	0	2	0	0	0

	0	1	2	3	4	5	6	7
b	/	1	.	.	.	.	.	.

k=2, q = 4, r = 2 (30 / 7 = 4 ... 2)

	0	1	2	3	4	5	6
a	-1	1	3	2	0	0	0

	0	1	2	3	4	5	6	7
b	/	1	4	.	.	.	.	.

k=3, q = 2, r = 6 (20 / 7 = 2 ... 6)

	0	1	2	3	4	5	6
a	-1	1	3	2	0	0	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	.	.	.	.

k=4, q = 8, r = 4 (60 / 7 = 8 ... 4)

	0	1	2	3	4	5	6
a	-1	1	3	2	5	0	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	8	.	.	.

k=5, q = 5, r = 5 (40 / 7 = 5 ... 5)

	0	1	2	3	4	5	6
a	-1	1	3	2	5	6	4

	0	1	2	3	4	5	6	7
b	/	1	4	2	8	5	.	.

k=6, q=7, r = 1 (50 / 7 = 7 ... 1)

実は配列を一切用いずに計算可能

観察１:

1/nが循環小数でない（有限小数）

$\Longleftrightarrow$ n が 2 と 5 だけで割り切れる

観察２:

整数 n を $2^i * 5^j * m$ の形に分解したとき,

m=1なら割り切れ,

m>1なら，非循環部分の長さはmax(i,j)で与えられる.

循環小数になる場合，非循環部分の最後での余りを記録しておいて，後はその余りが出現するまで繰り返せばよい.

In reality, it is computed without any array

Observation 1:

1/n is notrepeated decimal (finite decimal fraction)

$\Longleftrightarrow$ if n is divisible by 2 and 5.

Observation 2:

integer n is factored in the form $2^i * 5^j * m$:

if m=1 then it is divisible.

if m>1 then the length of the non-repeated part is given by max(i,j).

So, if it is a repeated decimal, it suffices to keep the last residue in the non-repeated part and repeat the loop until the residue appears.

例: 1/28 の場合

28 = 2^2 * 7 だから，非循環部分の長さは2.

10　/ 28 = 0 ... 10

100 / 28 = 3 ... 16　この余りを記録

160 / 28 = 5 ... 20

200 / 28 = 7 ... 4

40　/ 28 = 1 ... 12

120 / 28 = 4 ... 8

80　/ 28 = 2 ... 24

240 / 28 = 8 ... 16　：同じ余りを発見

答: 1/28 = 0.03#571428

Example: case of 1/28

28 = 2^2 * 7 therefore the length of non-repeated part is 2.
10 / 28 = 0 ... 10
100 / 28 = 3 ... 16 keep this residue
160 / 28 = 5 ... 20
200 / 28 = 7 ... 4
40 / 28 = 1 ... 12
120 / 28 = 4 ... 8
80 / 28 = 2 ... 24
240 / 28 = 8 ... 16 : the same residue has been found!

answer: 1/28 = 0.03#571428