Program Promenade

連載

無理数を近似する分数

田中 哲朗（東京大学情報基盤センター）
ktanaka@tanaka.ecc.u-tokyo.ac.jp

■問題の定義

今回取り上げる問題は，1999年に行われたアジア地区予選京都大会の問題A「Rational Irrationals」である．大会の問題は必ずしも難易度順に並んでいるわけではないが，問題Aは易しい問題が出題されることが多い．この問題も同じ大会の他の問題と比較すると易しめで，多くの参加チームが正解しているが，興味深い問題なので，ここで取り上げることにした．

整数 $n(n<10000)$ が与えられたときに，$x \le n, y \le n$ を満たす正の既約分数 $\frac{x}{y}$ 全体の集合を Q_n とする．このとき素数 $p(2 \le p < 10000)$ を与えたときに，$\frac{u}{v} < \sqrt{p}$ を満たす最大の $\frac{u}{v} \in Q_n$，および $\sqrt{p} < \frac{x}{y}$ を満たす最小の $\frac{x}{y} \in Q_n$ を求めるというのが問題の定義である．

問題を解く際には，例外的な場合がないかを検討する必要がある．たとえば，

1. $\frac{u}{v} = \sqrt{p}$ となる $\frac{u}{v}$ があった場合はどうするか？
2. 条件を満たす $\frac{u}{v}$ や $\frac{x}{y}$ が存在しない場合はどうするか？

などの点が気になるかもしれないが，1のケースは素数の平方根は必ず無理数であるため起こり得ない．これは大会参加者は常識として知っていることと思われるが，問題中でも親切に説明をしている．また，2のケースも，$2 \le p < 10000$ のとき，$1 < \sqrt{p} < p < n$ なので，$\frac{1}{1} < \sqrt{p}$ を満たす既約分数 $\frac{1}{1}$ が存在すること，$\sqrt{p} < \frac{n}{1}$ を満たす既約分数 $\frac{n}{1}$ が存在することから，起こり得ないことが分かる．

■素直な解法

今回もC++言語を使って解答を記述することにする．まずは，有理数を表現するクラスを作成する．すべての有理数を表すためには分子，分母を多倍長整数で表現する必要があるが，今回は分子も分母も9999以下であることが分かっているのでunsigned intで表現する．これらのフィールドは教科書では分子(numerator)，分母(denominator)，あるいは被除数(dividend)，除数(divisor)に因んだ名前を使って定義することが多いが，コンテストの際に使いなれない単語を使うとスペルミスによってコンパイル時間を無駄にすることがあるので，ここでは分子をx，分母をyと書くことにする．以下にクラスRationalの定義を示す．

```
/**
 * 正の有理数を表すクラス
 */
class Rational{
 private:
  unsigned int x,y;
 public:
  Rational(unsigned int x,unsigned int y)
    :x(x),y(y){}
  unsigned int getX() const{ return x;}
  unsigned int getY() const{ return y;}
  double getDouble() const{
    return (double)x/(double)y;
  }
};
```

Rational同士，Rationalとdoubleとの間の比較演算子をここでは次のように定義する．

```
bool operator<(Rational const& l,
               Rational const& r){
  return l.getDouble()<r.getDouble();
}
bool operator<(Rational const& l,
               double d){
  return l.getDouble()<d;
}
```

一番素直な解法は以下のようになるだろう．

(a) 分母 i を 1 から n まで動かして以下の (b)，(c) を繰り返し実行する．
(b) i*sqrt(p) を整数に変換して（切り捨てて）分子とした有理数を$\sqrt{p}$ 以下の最大値と比較し，必要に応じて更新する．
(c) i*sqrt(p)+1 を整数に変換して（切り捨てて）分子とした有理数を$\sqrt{p}$ 以上の最小値と比較し，必要に応じて更新する．
(d) $\sqrt{p}$ 以上の最小値，$\sqrt{p}$ 以下の最大値のペアを返す．

これをプログラミングしたものは以下のようになる．

```
pair<Rational,Rational>
solveOne(int p,int n){
  double sqrtP=sqrt((double)p);
  Rational leftMax((int)sqrtP,1);
  Rational rightMin((int)sqrtP+1,1);
  for(int i=2;i<=n;i++){
    int leftX=(int)(i*sqrtP);
    Rational left(leftX,i);
    int rightX=leftX+1;
    Rational right(rightX,i);
    if(leftX>n) break;
    if(leftMax<left)
      leftMax=left;
    if(rightX>n) break;
    if(right<rightMin)
      rightMin=right;
  }
  return make_pair(rightMin,leftMax);
}
```

問題の条件を満たす p, n の組合せは，全数チェックをすることも可能である．上のプログラムで，g++ 3.3, AthlonXP 3000+ で実行してみた結果，すべての組合せで正しい答えを返すことを確認できた．

■数値誤差に関する考察

前章のプログラムを見ると分子と分母が互いに素かどうかのチェックが抜けていることが分かる．

しかし，桁数の小さい整数（を変換した浮動小数点実数）同士の除算結果がある浮動小数点数のビットパターンになったときに，除数，被除数を整数倍した除算結果も同じビットパターンになれば，分母の小さい方からチェックしていくので，既約分数だけが残り問題がないことになる．

実際に AMD Opteron, Ultra SPARC III, PowerPC G4 等で試してみると 10000 以下の自然数同士の除算はすべてこの条件を満たしていた．

しかし，これが保証されていない以上，心配な場合は，Rational クラスの中に，インスタンスメソッド isIrreducible を

```
bool isIrreducible() const{
  return gcd(x,y)==1;
}
```

のように追加して，

```
if(leftMax<left)
```

の行を

```
if(left.isIrreducible() && leftMax<left)
```

のように変更し，rightMax の更新の際にも同様のチェックを加えればよい．

なお，gcd は最大公約数 (Greatest Common Divisor) を求める関数で，以下のように Euclid の互除法を使って定義することができる．

```
int gcd(int x,int y){
  int r=x%y;
  if(r==0) return y;
  return gcd(y,r);
}
```

別の解決策として，Rational 同士の比較演算子を

```
bool operator<(Rational const& l,
               Rational const& r){
return l.getX()*r.getY()<
       l.getY()*r.getX();
}
```

と定義する方法もある．約分して同じになる分数を比較した場合は必ず false になるので，誤差による影響は考えなくてよい．

比較演算子を double に変換してからの比較でなく，整数値のみで計算することのメリットとしては，大小の比較の際に誤差を考えなくてよいという点もある．10000 以下の場合，double を使えば問題はなかったが，仮に浮動小数点数の型として float を使うと，$\frac{4756}{3363} < \frac{8119}{5741}$ であるにもかかわらず，(float)4756/(float)3363≧(float)8119/(float)5741 となる処理系がある．

同様に，

```
int leftX=(int)(i*sqrtP);
```

の行でも誤差の影響を考える必要がある．i*sqrtP がある整数 j よりも少しだけ小さい場合に leftX が j になったり，i*sqrtP がある整数 j より少しだけ大きい場合に leftX が $j-1$ になったりする可能性がある☆1．

☆1 出てくる整数の範囲から考えると絶対値 1 以上の誤差は生じないと推察できる．

この誤差による影響も double では問題にならないが，float で計算をした時は問題となる場合がある．p=3, i=2911 のとき，$\sqrt{3}*2911$=5041.99990... だが，これを float で計算して int に変換すると 5042 になってしまう処理系があった．

この誤差による影響は，p の平方根が Rational の r より小さいかどうかを判定する以下の関数を定義すれば防ぐことができる．

```
bool lessThanSquare(int p,
                    Rational const& r){
  unsigned long long rxx=r.getX()*
                         r.getX();
  unsigned long long ryy=r.getY()*
                         r.getY();
  return p*ryy<rxx;
}
```

unsigned long long を用いているのは，unsigned int では は，$1<p<10000$, $1<r.getY()<10000$ な の で，$1<p*ryy<10^{12}$ となり，32bit では表現できない可能性があるが，64bit なら余裕で収まるためである．

unsined long long が使えない処理系では，代わりに double を使って，

```
bool lessThanSquare(int p,
                    Rational const& r){
  double rxx=(double)(r.getX()*r.getX());
  double ryy=(double)(r.getY()*r.getY());
  return (double)p*ryy<rxx;
}
```

とすればよい．double を使えば，仮数部が 53bit あり，この範囲の整数は問題なく表現できるし，乗算だけなので，誤差も生じない．

この lessThanSquare を使って，問題が起きた場合に補正を行うプログラムを後ろにつける．

```
pair<Rational,Rational>
solveOne(int p,int n){
  double sqrtP=sqrt((double)p);
  Rational leftMax((int)sqrtP,1);
  Rational rightMin((int)sqrtP+1,1);
  for(int i=2;i<=n;i++){
    int leftX=(int)(i*sqrtP);
    Rational left(leftX,i);
    int rightX=leftX+1;
    Rational right(rightX,i);
    while(lessThanSquare(p,left)){
      right=left;
      rightX--;
      leftX--;
      left=Rational(leftX,i);
    }
    while(!lessThanSquare(p,right)){
      left=right;
      rightX++;
      leftX++;
      right=Rational(rightX,i);
    }
    if(leftX>n) break;
    if(leftMax<left)
      leftMax=left;
    if(rightX>n) break;
    if(right<rightMin)
      rightMin=right;
  }
  return make_pair(rightMin,leftMax);
}
```

■ Stern-Brocot 木を用いた解法

この問題はまったく別のアプローチを使って解くこともできる．前準備のために少し説明を加える．

以下のようにすると，非負の既約分数（分子と分母の最大公約数が 1）すべてを系統的に作成することができる．まず，($\frac{0}{1}$, $\frac{1}{0}$) の 2 個の分数から始め（ここでは $\frac{1}{0}$ は正の無限大とする），次の操作を繰り返す．

隣接する分数 $\frac{x}{y}$ と $\frac{x'}{y'}$ の間に $\frac{x+x'}{y+y'}$ を挿入する．

たとえば，第 1 ステップでは，$\frac{0}{1}$ と $\frac{1}{0}$ の間に 1 個挿入する．

$$\frac{0}{1},\frac{1}{1},\frac{1}{0}$$

次のステップでは 2 個挿入する．

$$\frac{0}{1},\frac{1}{2},\frac{1}{1},\frac{2}{1},\frac{1}{0}$$

次のステップでは 4 個挿入する．

$$\frac{0}{1},\frac{1}{3},\frac{1}{2},\frac{2}{3},\frac{1}{1},\frac{3}{2},\frac{2}{1},\frac{3}{1},\frac{1}{0}$$

それ以降のステップでは，8 個，16 個と挿入していく．これらの全体は，無限の 2 分木と見なすことができ，Stern-Brocot 木（Stern-Brocot Tree）と呼ばれる．**図 -1** はこの木の一部を示す．

この構成法によって作られる分数列は以下の性質を持つ．

(a) 各ステップの後に得られる分数列は，小さい順に並んでいる．
(b) 各ステップの後に得られる分数列で隣接した数 $\frac{x}{y}$ < $\frac{x'}{y'}$ は，常に $x'y-xy'=1$ を満たす．
(c) 任意の既約分数 $\frac{p}{q}$ ($p>0$, $q>0$, p と q は互いに素) はこの方法を有限回繰り返して作成できる．

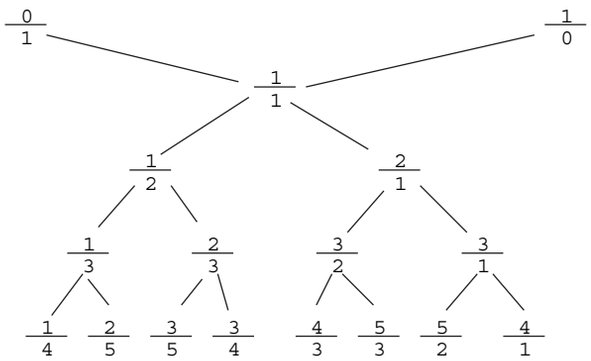

図-1　Stern-Brocot木

(a),(b) の証明は容易だが，(c) の証明は手間取るかもしれない．証明に興味のある方は文献 1）を読んでいただきたい．

あるステップの後に得られる分数列中で，ある無理数が区間 ($\frac{x}{y}$, $\frac{x'}{y'}$) に含まれているとする．次のステップで挿入される $\frac{x+x'}{y+y'}$ よりも，その無理数が大きいかどうかを判定するだけで，次のステップでどちらの区間に含まれるかを判定できる．

このステップを繰り返して，挿入しようとする数の分子か分母が n を越えたところで，得られる区間が，解答となる．プログラムは以下のようにシンプルに書ける上に一直線に正解にたどり着くので，効率的でもある．

```
Rational SBstep(Rational const& left,
                Rational const& right){
  return Rational(left.getX()+
                  right.getX(),
                  left.getY()+
                  right.getY());
}
pair<Rational,Rational>
solveSB(int p,int n){
  Rational left(0,1);
  Rational right(1,0);
  for(;;){
    Rational SB=SBstep(left,right);
    if(n<SB.getX() || n<SB.getY())
      return make_pair(right,left);
    if(lessThanSquare(p,SB)){
      right=SB;
    }
    else{
      left=SB;
    }
  }
}
```

ただし，最悪の場合の計算量という意味では，最初のプログラムと比べて勝っているわけではない．区間 ($\frac{sqrtP}{1}$,$\frac{sqrtP+1}{1}$) にたどり着くまでに sqrtP ステップ消費してしまうし，その後，最悪の場合 n ステップかかるからである．

なお，sqrtP を整数に変換した（切り捨てた）結果を SqrtP とするとき，初期値を

```
Rational left(SqrtP,1);
Rational right(SqrtP+1,1);
```

とすると，速くはなるが，sqrtP の値が正しくない場合のことを考慮すると，

```
while(lessThanSquare(p,left)){
  right=left;
  left=Rational(right.getX()-1,1);
}
while(!lessThanSquare(p,right)){
  left=right;
  right=Rational(left.getX()+1,1);
}
```

のような補正をする必要がある．しかし，こうしても改善は大したことはないし，せっかく浮動小数点計算を追放できたのに，わざわざ入れる必要はないだろう．

なお，ここでは詳しくは扱わないが連分数を使う方法もある．文献 1）にもあるように，連分数は Stern-Brocot 木と 1 対 1 に対応していて，かつ，連分数の 1 ステップが Stern-Brocot 木の数ステップにあたるので，速度の面では有利である．

なお，近くの大学院生にこの問題を解いてもらったところ，条件を満たす既約分数すべてを含む配列を作成して，ソートしてから $\sqrt{p}$ を含む範囲を探すという解法を披露してくれた．

Program Promenade の執筆者の一人でこの問題が出た京都大会で審判を務めた人の話によると，同様に，すべての既約分数を求める解法が続出したという．ソートをしなければ計算量は減って $O(n^2)$ となるが，それでも n が 10000 近いと時間制限にかかる可能性があるのは明らかなので，出題者としてはこのような解答は想定していなかったようだ．

もっとも，1999 年のコンテストの際に時間制限のため不合格になったプログラムの中には，最近の計算機で実行すれば制限時間内に終わってしまうものが含まれているかもしれない．問題を作る側もこのあたりの調整は難しいだろう．

参考文献

1）Graham, R.L., Knuth, D.E. and Patashnik, O 著，有澤，安村，萩野，石畑訳：コンピュータの数学 (CONCRETE MATHEMATICS : A Foundation for Computer Science)，共立出版 (1993).

（平成 16 年 2 月 6 日受付）