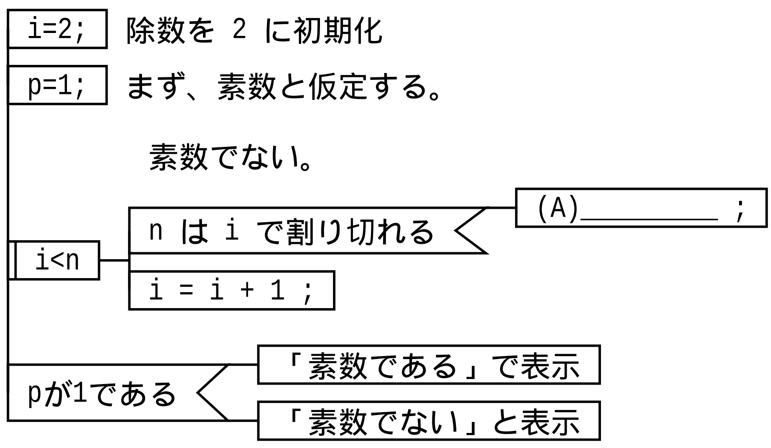

図 1: 素数の判断を定義通り行なうアルゴリズム

1　基本的考え方

素数 (prime number) とは、 1 とその数自身以外に約数をもたない正の整数のことである。したがって、ある正の整数 n が与えられた場合に、変数 i を 2 から $n-1$ までのすべての数で順に割っていき、一つでも割り切れる数（約数）があれば、 n は素数ではないといえる。それ以外の場合には、 n は素数であると判定できる。

2　そのままの方法

１の考え方をそのまま適用したアルゴリズムの PAD を 図 1に示す。ここで、素数であるか否かに応じて値 1,0 をとる変数 p を用いることにする。はじめに、 n が素数であると仮定し、変数 p を 1 に初期化しておく。次に、 $i = 2$ から始めて $i < n$ である間、 n が i で割り切れるか否かを判定し、割り切れた場合には p を 0 にする。繰り返し処理の終了した時点の p の値より、 n が素数であるか否かを知ることができる。

3　無駄な判定をしない方法

2の方法では、繰り返し処理がすべて終了していなくても、 n が i で割り切れた時点で n は素数ではないことが判明するにも関わらず、 $i = n - 1$ になるまで n が割り切れるかどうかを調べ続けることになり無駄である。これを回避するには、繰

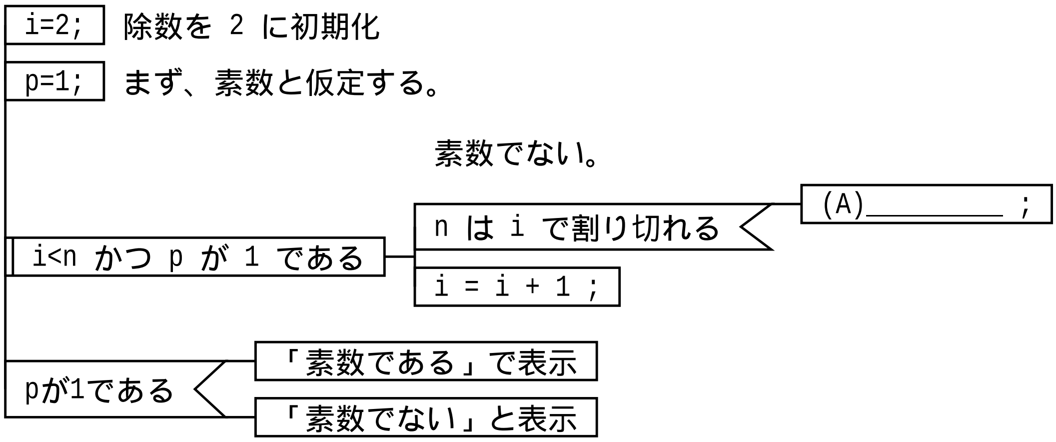

図 2: 無駄な判定をやめるアルゴリズム

り返し条件に「 p が 1 に等しい」をつけ加えればよい。このように修正したアルゴリズムの PAD を図 2に示す。

4　繰り返し処理からの抜け出しを用いる方法

3のアルゴリズムでは、 i が 1 増える毎に p の判定を行なっている。しかし、 p の値がいつ変化するかに着目すれば、 p に 0 を代入した時点で繰り返し処理を終了すればよいことがわかる。つまり、繰り返し条件は $i < n$ だけでよい。この場合、繰り返し処理が途中で終了した場合には、 n は素数でないことがわかる。すなわち、変数 p を用いなくても、繰り返し処理終了後に $i < n$ であるか否かで素数の判定ができることになる。このアルゴリズムの擬似コードを図 3に示す。

5　「番兵」を用いる方法

これまでのアルゴリズムの繰り返し処理には、実質的に二つの条件判断がある。すなわち、 n が i で割り切れるかどうかと、 $i = n - 1$ となるまで調べたかどうかである。これらの条件判断に関しては、より便利な方法がある。整数 n は n で常に割り切れることを利用して、繰り返し条件を「 n が i で割り切れない」にすると、 i を 1 ずつ増していけば i が n に等しくなった時に繰り返しは正しく終了する。したがって、とくに抜け出しを用いる必要はない。このアルゴリズムの PAD を図 4に示す。これは、「番兵」と同じ効果が自動的に得られた例となっている。

```
{
   i=2;
   while( i<n ){
      if( n は i で割り切れる ){
         (B)_______; /* while から抜け出す */
      }
      i = i + 1 ;
   }
   if( i<n ){
      「素数でない」と表示
   }else{
      「素数である」と表示
   }
}
```

図 3: 繰り返し処理からの抜け出しを用いるアルゴリズム

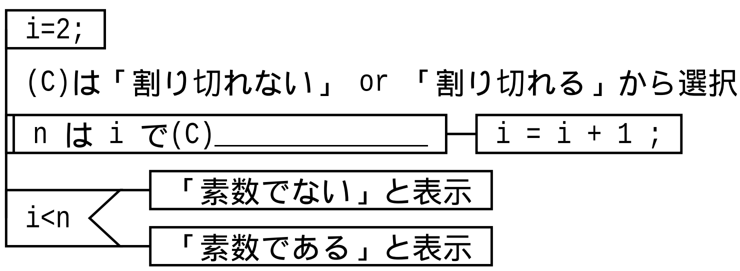

図 4: 「番兵」を用いるアルゴリズム

```
{
   i=2;
   while( i*i<=n ){
      if( n は i で割り切れる ){
         (B)_______; /* while から抜け出す */
      }
      i = i + 1 ;
   }
   if( i*i<=n ){
      「素数でない」と表示
   }else{
      「素数である」と表示
   }
}
```

図 5: より効率的なアルゴリズム

6　より効率のよい方法

これまでのアルゴリズムでは、 $2 < i < n-1$ の範囲で n が割り切れるかどうかを調べていた。これでは、 n の値が大きい時には、調べる回数も多くなることが予想される。そこで、 n を i で割った商を j とすると、 n が i で割り切れれば j でも割り切れることに着目する。すると、 i を順に増やしていき $i > j$ となった時点で、それ以上の i について調べる必要がなくなる。なぜなら、 j の定義より少なくとも n は j で割り切れるからである。したがって、 $2 < i \leqq \sqrt{n}$ の場合だけ調べればよい。つまり、 n が大きい場合に、調べる回数の大幅な減少が期待できる。このアルゴリズムの疑似コードを図 5に示す。この場合、 5で示した「番兵」を用いることはできない。