アルゴリズムとデータ構造

二分探索

森 立平
mori@c.titech.ac.jp

2018 年 6 月 15 日

今日のメッセージ

- **ソートしてある配列から値を探すときは二分探索**
- **二分探索の時間計算量は** $O(\log n)$

1 時間計算量のオーダー

アルゴリズムの計算量は通常オーダー記法 $O(\cdot)$ を用いて表わす。$f(n) = O(g(n)) \overset{\text{def}}{\iff} \exists c > 0, \forall n \geq 1, f(n) \leq cg(n)$ という意味である。なので n も $100000000000n$ も $O(n)$ であるし、$\log n$ も $O(n)$ である (もちろん $\log n$ は $O(\log n)$ でもある)。最大公約数の計算をするユークリッドの互除法と binary GCD は両方とも $O(n^2)$ である (n は入力長であり、最大公約数を計算したい整数の桁数)。ただし、定数倍の差で binary GCD の方が通常のコンピュータで高速となる。

2 二分探索

ソートしてある整数の列 $A_0 \leq A_1 \leq A_2 \leq \cdots \leq A_{n-1}$ の中から、x 以上である最小のものを見つける問題を考えよう。便宜上 $A_n := +\infty$ と定義し、

$$\text{bsearch}(\ell, r) := \min\{i \in \{\ell, \ell+1, \ldots, r-1, r\} \mid A_i \geq x\}$$

と定義すると、解きたい問題は $\text{bsearch}(0, n)$ である (もしも A_0 から A_{n-1} に x 以上のものが無ければ $\text{bsearch}(0, n) = n$ となる)。「アルゴリズム ≈ 漸化式」であるから、漸化式を立てることを考えよう。

$$\text{bsearch}(\ell, r) = \begin{cases} \ell, & \text{if } \ell = r \\ \text{bsearch}(\lfloor \frac{\ell+r}{2} \rfloor + 1, r), & \text{if } A_{\lfloor \frac{\ell+r}{2} \rfloor} < x \\ \text{bsearch}(\ell, \lfloor \frac{\ell+r}{2} \rfloor), & \text{otherwise.} \end{cases}$$

この漸化式に基づいてアルゴリズムを設計できる。

```
int binary_search(int A[], int x, unsigned int l, unsigned int r){
  unsigned int m;
  if(l >= r) return r;
  m = (l + r) / 2; // 桁溢れを考慮すると m = l + (r - l) / 2
  if(A[m] >= x) return binary_search(A, x, l, m);
  else return binary_search(A, x, m + 1, r);
}
```

便宜上 $A_n = +\infty$ として bsearch を定義したが、このアルゴリズムの中で**実際に** A[n] **がアクセスされることはない**。

3　二分探索の時間計算量

一回の漸化式の適用で探索するべき候補は約半分になるので $O(\log n)$ 回 (より具体的には $\lceil \log(n+1) \rceil$ 回) の漸化式の適用で停止する。

4　反復による二分探索

再帰によるプログラムを素直に反復に書き直すと次のようになる。

```
unsigned int binary_search(int A[], int x, int n){
  unsigned int l = 0;
  unsigned int r = n;
  while(r > l){
    unsigned int m = (l + r) / 2;
    if(A[m] >= x) r = m;
    else l = m + 1;
  }
  return r;
}
```

これはこれで構わないが、少し覚えにくい。二分探索はとても簡単なアルゴリズムだが**正確に書くのは難しい** (誰でも一度はバグらせたことがある)。次のようなプログラムを覚えるとよい。

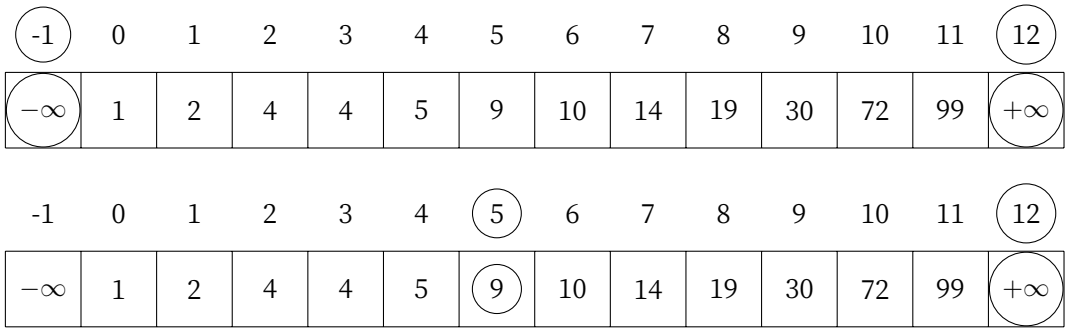
```
unsigned int binary_search(int A[], int x, int n){
  int lb = -1;
  int ub = n;
  while(ub - lb > 1){
    int m = (lb + ub) / 2;
    if(A[m] >= x) ub = m;
    else lb = m;
  }
  return ub;
}
```

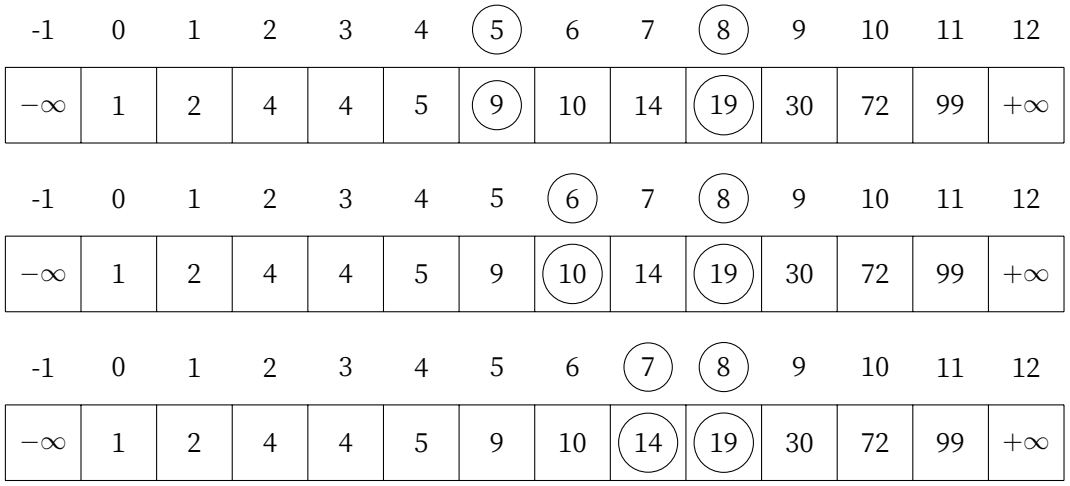
これらの変数には次のような意味がある。

- lb は「A[lb] < x であることが分かっている最大の数」
- ub は「A[ub] >= x であることが分かっている最小の数」

まず A[-1] = $-\infty$ と解釈して、lb = -1; ub = n と初期化する。もし ub == lb + 1 ならば、ub が解となる。ub > lb + 1 ならば、m = (lb + ub) / 2 とおく。ここで、m > lb && ub > m が成り立つことに注意する。よって、A[-1] や A[n] はアクセスされないし、ub - lb は 1 ステップで必ず減少する。

最後に $n = 12, x = 17$ とした場合の例を紹介する。





最終的に `ub == 8` となる。`A[lb] < x` と `A[ub] >= x` が常に満たされていることに注意しよう。**このように覚えておけば二分探索のプログラムを忘れることはない**。二分探索はとても応用が広く、様々な問題を解くのに使うことができる。その場合に特にこの覚えやすい二分探索の書き方が役に立つ。

5 二分探索の応用

二分探索はソートされた配列から値を探すという目的以外にも使用することができる。ソートされた配列の代わりに単調な二値関数 $p\colon \{0,1,2,\dots,n-1\} \to \{0,1\}$ について考えよう。ここで p が単調というのは $p(x) \geq p(x-1)$ が $x = 1,2,\dots,n-1$ について満たされるという意味である。このような p について

$$p(x) = 1 \text{ を満たす最小の } x \in \{0,1,2,\dots,n-1\} \text{ をもとめよ}$$

という問題を二分探索で解くことができる。具体的には前節のプログラムのうち 4 行目からの 5 行を次のように変更すればよい。

```
while(ub - lb > 1) {
  int m = (lb + ub) / 2;
  if(p(m)) ub = m;
  else lb = m;
}
```

実際に変更したのはこの中の 3 行目の `if` 文の条件だけである。この考え方を用いると、上記の問題について非常に大きな n に対しても現実的な時間で解くことができる。

例えば簡単な例として整数 n の平方根の切り上げ $\lceil\sqrt{n}\rceil$ を求める問題を考えよう。この場合、

$$p(x) := \begin{cases} 0, & \text{if } x < \lceil\sqrt{n}\rceil \\ 1, & \text{if } x \geq \lceil\sqrt{n}\rceil \end{cases}$$

となる。これは x が整数だとすると、次のように書き換えることができる。

$$p(x) = \begin{cases} 0, & \text{if } x^2 < n \\ 1, & \text{if } x^2 \geq n \end{cases}$$

この場合、

```
int p(unsigned int x){
  return x * x >= n;
}
```

と定義してあげれば、二分探索で $\lceil \sqrt{n} \rceil$ が計算できる。変数 `lb` と `ub` の初期値は `lb = 0, ub = n` とすればよい (もちろん、もっと良い $\lceil \sqrt{n} \rceil$ の上下界を使ってもよい)。そうすると反復回数は $\lceil \log n \rceil$ である。このように、**簡単に計算ができる単調な関数の逆関数は二分探索を用いて計算ができる**。ただし平方根の計算にはニュートン法というもっと効率的な方法が存在する。しかしニュートン法は関数 (この場合 x^2 にあたるもの) の凸性も用いるので、二分探索の方が適用範囲が広い。