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** $O(\log n)$
- **単調な関数の逆関数を求める**ことに二分探索が使える

1 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、

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

と定義すると、解きたい問題は $\mathrm{bsearch}(0, n)$ 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 $\mathrm{bsearch}(0, n) = n$ 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。

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

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**。

2 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 $\lceil n/2 \rceil$ もしくは $\lfloor n/2 \rfloor$ 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。

3 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。

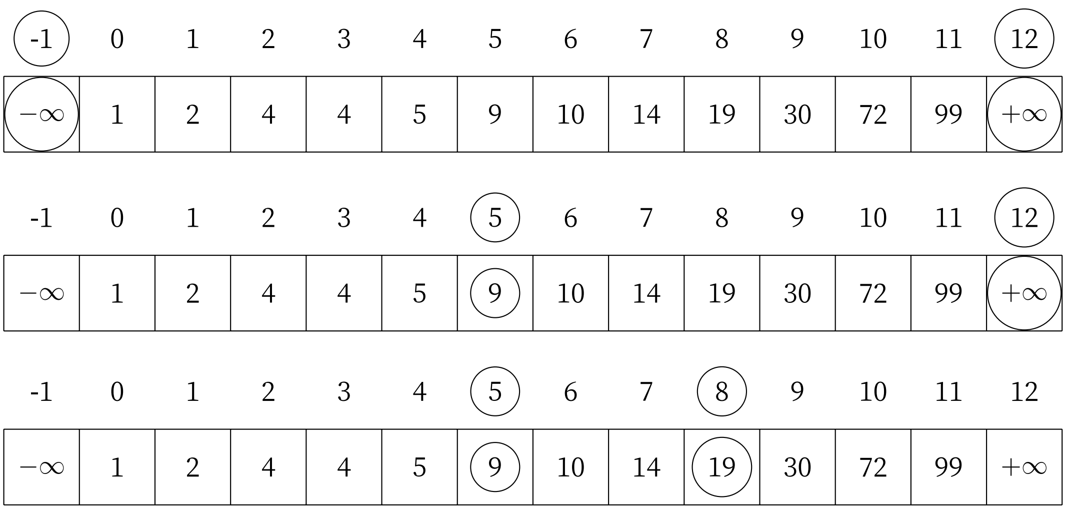
```
unsigned int binary_search(int A[], int x, int n){
  int lb = -1;
  int ub = n;
  while(ub - lb > 1){
    int m = (lb + ub) / 2;
    if(A[m] >= x) ub = m;
    else lb = m;
  }
  return ub;
}
```

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] = -∞ 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。



-1	0	1	2	3	4	5	(6)	7	(8)	9	10	11	12
$-\infty$	1	2	4	4	5	9	(10)	14	(19)	30	72	99	$+\infty$

-1	0	1	2	3	4	5	6	(7)	(8)	9	10	11	12
$-\infty$	1	2	4	4	5	9	10	(14)	(19)	30	72	99	$+\infty$

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。

4 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 $p\colon \{0,1,2,\ldots,n-1\} \to \{0,1\}$ 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 $x = 1,2,\ldots,n-1$ 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

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

という問題を二分探索で解くことができる。具体的には次のようなプログラムになる。

```
int lb, ub;
lb = 「絶対にp(lb) == 0となる値」;
ub = 「絶対にp(ub) == 1となる値」;
while(ub - lb > 1) {
  int m = (lb + ub) / 2;
  if(p(m)){
    ub = m;
  }
  else {
    lb = m;
  }
}
/* ここで ub が p(ub) == 1 となる最小の値
        lb が p(lb) == 0 となる最大の値となっている */
```

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、

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

とすると、$p(x) = 1$ となる最小の整数 x が $\lceil\sqrt{n}\rceil$ である。これを次のように書き換えることができる。

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

よって、

```
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**。今回は整数だけを考えているが、実数にも簡単に一般化できる。その場合 lb, ub, m は unsigned int ではなく double という型にする必要がある。

```
unsigned int p(double a, double x){
  return a * a >= x;
}

/* x は非負 */
double sqrt(double x){
  double lb, ub;
  if(x <= 1){
    lb = 0;
    ub = 1;
  }
  else {
    lb = 1;
    ub = x; // もっといい上界を使ってもいい
  }
  while(ub - lb > 0.000001){
    double m = (lb + ub) / 2;
    if(p(m, x)) ub = m;
    else lb = m;
  }
  return (lb + ub) / 2;
}
```

この関数 sqrt は引数の平方根を 0.000001 の半分以下の誤差で返す。ここで、ub - lb > 0.000001 をループの条件としているが、浮動小数演算の桁落ちのせいで、ub や lb がとても大きいときに、数値誤差の影響でループが停止しないことがあるため危険である。そこで、決まった回数だけループするという方法も一般的である。例えば 20 回反復すれば、最初の誤差 (x <= 1 なら 1、そうでなければ x-1) の 2^{21} 分の一の誤差になる。一方で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、単調性しか必要としない二分探索の方が適用範囲が広い。

5 二分法

二分探索と非常に良く似たアルゴリズムに「二分法」がある。これは**連続関数**の零点の一つを計算するためのアルゴリズムである。連続関数 $f\colon \mathbb{R} \to \mathbb{R}$ の零点の一つを計算することを考える。ある $y > x$ について、$f(x)$ と $f(y)$ の符号が違うとき、ある $z \in (x, y)$ が存在して $f(z) = 0$ となる (中間値の定理)。よって、まず最初に $f(x)$ と $f(y)$ の符号が異なるような x と y を知っていれば、$m = (x + y)/2$ について $f(m)$ を評価し、その符号によって (x, m) もしくは (m, y) のどちらかについて零点が必ず存在することが分かる。二分探索と同様に 1 ステップで区間が半分になるので n ステップ実行すれば、$(y - x)/2^{n+1}$ の誤差で零点が計算できる。

二分法を適用するために必要な f の性質は「連続性」である。一方、二分探索に必要な p の性質は「単調性」である。そのため、二分探索と区別して上記のアルゴリズムを二分法と呼ぶことにする。あまり区別しないで両方二分探索と呼んでいる人も少なくない。

6 全体の C 言語プログラム

例えば入力として整数 $n \geq 1$ をとり、$\lceil \sqrt{n} \rceil$ を出力するプログラムは次のようになる。

```
#include <stdio.h>

int n;

int p(int m){
  return (long long int) m * m >= n;
}

int main(){
  int lb, ub;
  scanf("%d", &n);
  lb = 0;
  ub = n;
  while(ub - lb > 1) {
    int m = (lb + ub) / 2;
    if(p(m)){
      ub = m;
    }
    else {
      lb = m;
    }
  }
  printf("%d\n", ub);
  return 0;
}
```

- `#include <stdio.h>` は毎回書くおまじない。入出力関係のライブラリを使うためのヘッダファイルの取り込み。これが無いと `scanf` や `printf` が使えない。
- `int n;` のような関数の外側にある変数は**グローバル変数**と呼ばれ、プログラムのどこからでもアクセスできる。
- `int main()` は **main 関数**という特別な関数で、プログラムを実行するとこの main 関数が実行される。
- `int lb, ub;` のような関数の内側にある変数は**ローカル変数**と呼ばれ、その関数の内側からだけアクセスできる。
- `scanf("%d", &n);` は**標準入力から整数を読み込み** `n` に代入する命令文である。標準入力から整数を 2 つ読み込んで `int` 型の変数 `a` と `b` に代入したい場合は `scanf("%d%d", &a, &b);` とする。
- `printf("%d\n", &n);` は標準出力へ整数 `n` と改行コード `\n` を出力する。
- `return 0;` は main 関数の最後に書いておこう (書かなくてもエラーが起きる訳ではないが)。これは「プログラムが正常に終了した」ということを表わしている。エラーで終了するときは `return 1;` など 0 以外の値を返すようにする。この授業ではエラーによる終了は扱わないので、main 関数の最後に `return 0;` を書くことを覚えておけばよい。

他によくある処理として、最初に数列の長さ n を、その後に数列 $A_0, \ldots, A_{n-1}$ を読み込みたい場合は

```
int i, n;
scanf(”%d”, &n);
for(i = 0; i < n; i++){
  scanf(”%d”, &A[i]);
}
```

とする。ここで

```
for(AAA; BBB; CCC){
  DDD
}
```

は

```
AAA;
while(BBB){
  DDD
  CCC;
}
```

と等価である。ここで、AAA, BBB, CCC はそれぞれ一つの文であり、DDD は文の列であるとする。

標準入力とは特定のファイルなどではなくプログラムが受け付ける入力であり、scanf を用いたプログラムを実行すると入力受け付け状態になる。入力と改行 (Enter キー) を入力すると、scanf により読み取られて残りの処理が実行される。毎回入力するのが面倒くさい場合はターミナルで `echo 100 | ./a.out` とすれば `./a.out` としてから 100 を入力して Enter を押すのと同じことになる。また、ファイル `file.txt` に入力を記入しておいてから、`./a.out < file.txt` としてもよい。

7 より高度な二分探索

二分探索とは、単調な関数 $p\colon \mathbb{Z} \to \{0,1\}$ について、$p(x) = 1$ となる最小の $x \in \mathbb{Z}$ を見つけるアルゴリズムであった。この二分探索は適用範囲が広く、様々な応用がある。例えば、今回の課題 (後述) の「りんご狩り」では

$$p(x) := \begin{cases} 1, & \text{if } x \text{ 個のりんごが入るりんごバッグを合計 } k \text{ 個配ると全員がりんごを持って帰れる} \\ 0, & \text{otherwise.} \end{cases}$$

と定義すれば p は単調性を満たし、$p(x) = 1$ となる最小の x が求める解となる。この $p(x)$ は効率的に計算することができるので、この問題を十分効率的に解くことができる。

8 課題

以下のすべての課題について、二分探索で用いる lb と ub の初期値はできるだけ良いものを選ぶこと。lb+1 や ub-1 が初期値として使用できないようなものを選ぶこと。最初の配列の問題以外は入力に依存しない初期値を考えればよい。入力制約に注意すること。

8.1　配列の二分探索 ★

問題文

単調非減少整数列 $a_0, a_1, \ldots, a_{n-1}$ と整数 k について、$a_x \geq k$ となる最小の整数 x を求めよ。ただし、そのような x が存在しないときは $x = n$ とする。

入力制約

$1 \leq n \leq 10^5$
$1 \leq k \leq 10^9$
$1 \leq a_i \leq 10^9, \quad i \in \{0, 1, \ldots, n-1\}$
$a_i \leq a_{i+1}, \quad i \in \{0, 1, \ldots, n-2\}$

入力

$n\ k$
$a_0\ a_1\ \cdots\ a_{n-1}$

出力

x

サンプル 1

入力:

```
12 17
1 2 4 4 5 9 10 14 19 30 72 99
```

出力:

```
8
```

サンプル 2

入力:

```
12 3
1 2 4 4 5 9 10 14 19 30 72 99
```

出力:

```
2
```

サンプル 3

入力:

```
1 10
9
```

出力:

```
1
```

8.2　りんご狩り ★★

問題文

りんご狩りに n 人集まった。各 $i = 1, 2, \ldots, n$ について i 番目の人は a_i 個のりんごを収穫した。合計 k 個のりんごバッグ (サイズは全て同じ) を配ると、全ての人はりんごを持ち帰ることができた。りんごバッグに入れることができるりんごの個数としてあり得るもののうち最小値 x をもとめよ。

入力制約

$1 \leq n \leq 10^5$
$1 \leq a_i \leq 10^9, \quad i \in \{1, 2, \ldots, n\}$
$n \leq k \leq 10^5$

入力

$n\ k$
$a_1\ a_2\ \cdots\ a_n$

出力

x

サンプル 1

入力:

```
5 7
1 2 3 4 5
```

出力:

```
3
```

サンプル 2

入力:

```
5 5
10 20 30 40 1000
```

出力:

```
1000
```

サンプル 3

入力:

```
7 1000
1 2 3 4 5 6 7
```

出力:

```
1
```

8.3 槍作り ★★

問題文

長さ m の木から長さ x の木製の槍が $\lfloor m/x \rfloor$ 本作れる。今 n 本の木があり、各木の長さは $a_1, a_2, \ldots, a_n$ である。これらの木から同じ長さの木製の槍を k 本作りたい。作ることができる槍の長さの最大値 x をもとめよ。

入力制約

$1 \leq n \leq 10^5$
$1 \leq a_i \leq 10^9, \quad i \in \{1, 2, \ldots, n\}$
$1 \leq k \leq 10^5$

入力

$n\ k$
$a_1\ a_2\ \cdots\ a_n$

出力

x

サンプル 1

入力:

```
5 7
1 2 3 4 5
```

出力:

```
1
```

サンプル 2

入力:

```
5 5
10 20 30 40 1000
```

出力:

```
200
```

サンプル 3

入力:

```
7 1000
1 2 3 4 5 6 7
```

出力:

```
0
```

8.4 仕事配分 ★★

問題文

n 時間の仕事を k 人の人に配分したい。それぞれの人は連続した何時間かを働き、また異る人が同時に働くことはないものとする。各時間の仕事量が整数列 $a_1, a_2, \ldots, a_n$ で与えられるとき、一番仕事量が多い人の仕事量を最小化するように仕事を配分するとする。そのような仕事配分をしたとき、一番仕事量が多い人の仕事量 x をもとめよ。

入力制約

$1 \leq n \leq 10^5$
$1 \leq a_i \leq 10^4, \quad i \in \{1, 2, \ldots, n\}$
$1 \leq k \leq n$

入力

$n\ k$
$a_1\ a_2\ \cdots\ a_n$

出力

x

サンプル 1

入力:

```
3 2
1 2 4
```

出力:

```
4
```

サンプル 2

入力:

```
3 2
1 4 2
```

出力:

```
5
```

サンプル 3

入力:

```
5 5
10 20 30 40 1000
```

出力:

```
1000
```