Бинарный поиск

Обычный бинарный поиск

Дан посорченный массив a. Найти в нем наибольший/наименьший индекс k, такой, что a[k] <= / < / > / >= заданного числа x.

Вариант 1.

```
// находит наименьший индекс i, такой, что a[i] >= x
// возвращает -1 если нет такого индекса
int lower_bound(int[] a, int x) {
      // границы left, right включительные
      int left = 0;
      int right = a.length - 1;
      int ans = -1; // что мы вернем, если ответ не найден
      while (left <= right) {
            int mid = (left + right) / 2;
            if (a[mid] >= x) {
                  // это нас устраивает
                  // обновляем ответ
                  // ответ всегда улучшается
                  ans = mid;
                  right = mid - 1;
            } else {
                  left = mid + 1;
            }
      }
}
```

В начале ans инициализируется значением, которое надо вернуть, если ответ не найден.
Всегда рассматривается включительный интервал [left, right]
На каждом шаге интервал обновляется либо как right=mid-1, либо как left=mid+1
Если условие нас устраивает, обновляется ответ как ans=mid

Вариант 2.

Пусть функция ok() сначала принимает true, а потом false.

```
int step = 1 << 29; // должна быть степень двойки
int ans = 0;
while (step > 0) {
      if (ok(ans + step)) { // для предыдущей задачи: ok(x) = (a[i] < x)
            ans += step;
      }
      step /= 2;
}
// для предыдущей задачи:
// мы бы нашли последний элемент для которого a[i] < x, и вернули бы ans + 1
```

в конце будет ans - наибольшее число, для которого выполнено ok(ans)

Бинпоиск в даблах

Задача: Пусть f(x) - монотонно возрастающая функция. Пусть f(L) < 0, f(R) > 0. Найти корень уравнения f(x) = 0.

```
// EPS = маленькое число, например 1e-12
```

Вариант 1 (неверный)

```
while (R - L > EPS) {
      double mid = (L + R) / 2;
      if (f(mid) > 0) {
            R = mid;
      } else {
            L = mid;
      }
}
```

y = 1e18 * x - график представляет собой почти вертикальную прямую.
Когда R - L <= EPS, разница между f(L) и f(R) все еще около 1e6.
И фиг угадаешь, где на [L,R] будет f = 0.

Вариант 2 (неверный)

```
while (f(R) - f(L) > EPS) {
      double mid = (L + R) / 2;
      if (f(mid) > 0) {
            R = mid;
      } else {
            L = mid;
      }
}
```

y = x / 1e18 - график представляет собой почти горизонтальную прямую.
Когда f(R) - f(L) <= EPS, разница между L и R все еще около 1e6.
И фиг угадаешь, где на [L,R] будет f = 0.

Вариант 3 (**верный**)

```
for (int i = 0; i < 100; i++) {
      double mid = (L + R) / 2;
      if (f(mid) > 0) {
            R = mid;
      } else {
            L = mid;
      }
}
```

Бинарный поиск по ответу

Задача. Есть точки x[0], ... , x[n-1], упорядоченные по возрастанию. Надо добраться из первой в последнюю. За один ход можно прыгать на любое число метров, которое <= S (можно разные для разных прыжков). После прыжка можно приземляться только в точках x[0], …, x[n-1]. Найти минимальное S, что можно допрыгать за K ходов *(даны x[0], …, x[n-1] и K, найти минимальное S)*.

Фраза “бинпоиск по ответу” намекает, что надо искать ответ бинпоиском.
Внутри цикла бинпоиска (который жрет логарифм) остается время на линейную проверку

```
int left = 1, right = x[n-1]-x[0];
int ans = -1;
while (left <= right) {
      int mid = (left + right) / 2;
      if (ok(mid)) {
            // с длиной прыжка mid получилось
            // надо попробовать меньшие длины
            ans = mid;
            right = mid - 1;
      } else {
            left = mid + 1;
      }
}
функция ok() делает проверку за O(N)
```

Предпосылки использовать бинпоиск:

1. В задаче есть какая то величина, при возрастании/убывании которой ответ сначала существует, а потом не существует.
2. Когда требуют найти минимальное/максимальное значение, при котором что-то там выполняется.

Они не всегда срабатывают, но если в задаче есть одно из этих свойств, имеет смысл подумать, а не решается ли это бинпоиском.

Задачи

http://acm.timus.ru/problem.aspx?space=1&num=1133
http://acm.timus.ru/problem.aspx?space=1&num=1184
http://codeforces.com/problemset/problem/51/C
http://codeforces.com/problemset/problem/65/C
http://codeforces.com/problemset/problem/68/B
http://codeforces.com/problemset/problem/83/B
http://codeforces.com/problemset/problem/180/E
http://codeforces.com/problemset/problem/474/B
http://codeforces.com/problemset/problem/590/B

Время осталось, так что: битовые операции

& - побитовый AND
| - побитовый OR
^ - побитовый XOR
~ - побитовое отрицание
<< - сдвиг влево (справа остаются нули)
>>> - сдвиг вправо (слева остаются нули, java only)
>> - сдвиг вправо (слева остается тот бит, что был на самой левой позиции изначально)

1) дано i, получить 2^i
2) проверить, является ли x степенью двойки (известно, что x > 0)
3) поставить (сделать равным 1) i-ый бит в числе x
4) убрать (сделать равным 0) i-ый бит в числе x
5) флипнуть i-ый бит в числе x (т.е. 1 поменять на 0, а 0 на 1)
6) проверить, есть ли i-ый бит в числе x
7) посчитать массив bc, где bc[x] - сколько единичек в числе x. за линейное время

Ответы:

1) 1 << i
2) (x & (x - 1)) == 0 // x имеет вид 000.0001000.000, x-1 будет иметь вид 000...0000111..111
3) x |= (1 << i)
4) x &= ~(1 << i)
5) x ^= (1 << i)
6) (x & (1 << i)) != 0, (x | (1 << i)) == x, ((x >>> i) & 1) != 0
7) `for (int i = 1; i < N; i++) bc[i] = bc[i >>> 1] + (i & 1);`