Бинарный поиск

Denis Bakin

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$
- $x > 44$? NO $\Rightarrow x \in [39; 44]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$
- $x > 44$? NO $\Rightarrow x \in [39; 44]$
- $x > 41$? YES $\Rightarrow x \in [42; 44]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$
- $x > 44$? NO $\Rightarrow x \in [39; 44]$
- $x > 41$? YES $\Rightarrow x \in [42; 44]$
- $x > 43$? NO $\Rightarrow x \in [42; 43]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$
- $x > 44$? NO $\Rightarrow x \in [39; 44]$
- $x > 41$? YES $\Rightarrow x \in [42; 44]$
- $x > 43$? NO $\Rightarrow x \in [42; 43]$
- $x > 42$? NO $\Rightarrow x \in [42; 42]$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Интуиция: игра в «данетку»

- Загадано число от 1 до 100
- Разрешено спрашивать: «$x > m$?»
- Каждый вопрос сокращает диапазон поиска в 2 раза
- Пусть загадано число 42

Пример:

- $x > 50$? NO $\Rightarrow x \in [1; 49]$
- $x > 25$? YES $\Rightarrow x \in [26; 49]$
- $x > 38$? YES $\Rightarrow x \in [39; 49]$
- $x > 44$? NO $\Rightarrow x \in [39; 44]$
- $x > 41$? YES $\Rightarrow x \in [42; 44]$
- $x > 43$? NO $\Rightarrow x \in [42; 43]$
- $x > 42$? NO $\Rightarrow x \in [42; 42]$
- $x = 42$

- 7 шагов = примерно $\log_2(100)$
- В общем случае — $O(\log n)$ шагов

Бинарный поиск: идея

- Метод “половинного деления” — каждый раз делим область поиска пополам
- Применяется, когда мы можем придумать следствие из ответа на вопрос. Например:
  - область поиска **упорядочена** (массив, диапазон)
  - или **функция монотонна**
- На каждом шаге:
  - вычисляем середину `m = (l + r) / 2`
  - решаем, в какой половине искать дальше
- Останавливаемся, когда область становится достаточно маленькой

Поиск в отсортированном массиве

Пусть массив a отсортирован по возрастанию.

Задача: найти индекс элемента `x` или вернуть `-1`, если его нет.

1. Задаём `l = 0`, `r = n - 1`
2. Находим середину `m = (l + r) / 2`
3. Если `a[m] == x` — нашли
4. Если `a[m] < x` — сдвигаем `l = m + 1`
5. Если `a[m] > x` — сдвигаем `r = m - 1`
6. Повторяем, пока `l <= r`

Оценка длины области после k итераций

- Начальная длина: $r - l$
- После 1-й итерации: $\frac{r-l}{2}$
- После 2-й итерации: $\frac{r-l}{2^2}$
- После k итераций: $\frac{r-l}{2^k}$

$$\frac{r-l}{2^k} \geq 1 \implies k \leq \log_2(r-l) \implies k = O(\log n)$$

Реализация в C++

```cpp
int find(const std::vector<int>& data, int elem_to_find) {
    int l = 0, r = (int)data.size() - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (data[m] == elem_to_find)
            return m;
        else if (data[m] < elem_to_find)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
```

Асимптотика:

Реализация в C++

```cpp
int find(const std::vector<int>& data, int elem_to_find) {
    int l = 0, r = (int)data.size() - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (data[m] == elem_to_find)
            return m;
        else if (data[m] < elem_to_find)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
```

Асимптотика:

- Время: $O(\log n)$

Реализация в C++

```cpp
int find(const std::vector<int>& data, int elem_to_find) {
    int l = 0, r = (int)data.size() - 1;
    while (l <= r) {
        int m = (l + r) / 2;
        if (data[m] == elem_to_find)
            return m;
        else if (data[m] < elem_to_find)
            l = m + 1;
        else
            r = m - 1;
    }
    return -1;
}
```

Асимптотика:

- Время: $O(\log n)$
- Память: $O(1)$

Готовые функции C++ STL

- `std::lower_bound(a.begin(), a.end(), x)` — первый элемент $\geq x$
- `std::upper_bound(a.begin(), a.end(), x)` — первый элемент $> x$
- `std::binary_search(a.begin(), a.end(), x)` — проверяет, есть ли элемент x

Пример:

```
auto it = std::lower_bound(a.begin(), a.end(), x);
if (it != a.end() && *it == x)
    std::cout << "Found\n";
```

- При каких условиях будет корректно работать пример выше?

Бинарный поиск по монотонному свойству

- Интуиция: обобщить операция сравнения в упорядоченном массиве на произвольное монотонное свойство

Бинарный поиск по монотонному свойству

- Интуиция: обобщить операция сравнения в упорядоченном массиве на произвольное монотонное свойство
- Идея: если свойство меняется **ровно один раз**, можно найти точку перехода

Бинарный поиск по монотонному свойству

- Интуиция: обобщить операция сравнения в упорядоченном массиве на произвольное монотонное свойство
- Идея: если свойство меняется **ровно один раз**, можно найти точку перехода
- Реализация: найти монотонную по i функцию $f(i)$ с множеством значение $\Big\{0, 1\Big\}$

Бинарный поиск по ответу

- Формулируем задачу как:

  «Найти максимальное x, для которого выполняется свойство $P(x)$»

- Свойство $P(x)$ монотонно:
  - если выполняется для x, то выполняется и для всех меньших (или наоборот)

Пример: «Коровы в стойла»

*Есть n стойл с координатами и k коров. Нужно разместить коров так, чтобы **минимальное расстояние** между ними было **максимальным***

Как прийти к решению

- Найти функцию от ответа x с булевым значением $P(x)$

Как прийти к решению

- Найти функцию от ответа x с булевым значением $P(x)$
- Убедиться, что она монотонна

Как прийти к решению

- Найти функцию от ответа x с булевым значением $P(x)$
- Убедиться, что она монотонна
- Применить бинарный поиск по x

Решение через бинарный поиск по ответу

1. Сортируем координаты стойл
2. Проверяем, можно ли расставить коров на расстоянии x
3. Ищем **наибольшее возможное** x

```
bool check(int x) {
    int cows = 1, last = coords[0];
    for (int c : coords) {
        if (c - last >= x) {
            cows++;
            last = c;
        }
    }
    return cows >= k;
}
```

Бинарный поиск для задачи про коров

```
int solve() {
    sort(coords.begin(), coords.end());
    int l = 0, r = coords.back() - coords[0] + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (check(m))
            l = m;   // можно - увеличиваем расстояние
        else
            r = m;   // нельзя - уменьшаем
    }
    return l;
}
```

Асимптотика:

Бинарный поиск для задачи про коров

```
int solve() {
    sort(coords.begin(), coords.end());
    int l = 0, r = coords.back() - coords[0] + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (check(m))
            l = m;   // можно - увеличиваем расстояние
        else
            r = m;   // нельзя - уменьшаем
    }
    return l;
}
```

Асимптотика:

- Внутренняя проверка $\mathcal{O}(n)$

Бинарный поиск для задачи про коров

```
int solve() {
    sort(coords.begin(), coords.end());
    int l = 0, r = coords.back() - coords[0] + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (check(m))
            l = m;   // можно - увеличиваем расстояние
        else
            r = m;   // нельзя - уменьшаем
    }
    return l;
}
```

Асимптотика:

- Внутренняя проверка $\mathcal{O}(n)$
- Внешний бинпоиск $\mathcal{O}(\log X)$

Бинарный поиск для задачи про коров

```
int solve() {
    sort(coords.begin(), coords.end());
    int l = 0, r = coords.back() - coords[0] + 1;
    while (r - l > 1) {
        int m = (l + r) / 2;
        if (check(m))
            l = m;   // можно - увеличиваем расстояние
        else
            r = m;   // нельзя - уменьшаем
    }
    return l;
}
```

Асимптотика:

- Внутренняя проверка $\mathcal{O}(n)$
- Внешний бинпоиск $\mathcal{O}(\log X)$
- Общая сложность: $\mathcal{O}(n \log X)$

Пример: «Принтеры»

Первый принтер печатает 1 лист за x минут, второй — за y минут. Сколько минут потребуется, чтобы напечатать n листов?

Решение

Функция-предикат — «успеют ли за t минут»:

```
bool check(long long t) {
    return (t / x) + (t / y) >= n;
}
```

Бинарный поиск с вещественными аргументами

Теперь аргументы — **действительные числа**, а не целые. Пример: найти $\sqrt{2}$ из уравнения $x^2 = 2$.

Свойство:

$$f(x) = \begin{cases} 0, & x^2 < 2 \\ 1, & x^2 \geq 2 \end{cases}$$

Монотонное — можно применить бинарный поиск.

Дихотомия для унимодальной функции

- Функция унимодальна — сначала возрастает, потом убывает
- Пример: парабола, функция прибыли по времени

Цель: найти $x_{\max}$, где функция достигает максимума

Алгоритм дихотомии

1. Задаём начальный отрезок $[l, r]$
2. Находим две точки:
$$m_1 = l + \frac{r-l}{3}, \quad m_2 = r - \frac{r-l}{3}$$
3. Если $f(m_1) < f(m_2)$ — максимум в $[m_1, r]$
4. Иначе — в $[l, m_2]$
5. Повторяем до сходимости ($r - l < \varepsilon$)

Пример кода дихотомии

```
double ternary_search(double l, double r) {
    for (int i = 0; i < 100; i++) {
        double m1 = l + (r - l) / 3;
        double m2 = r - (r - l) / 3;
        if (f(m1) < f(m2))
            l = m1;
        else
            r = m2;
    }
    return (l + r) / 2;
}
```

- Работает для **унимодальных функций**
- Находит **точку экстремума** (обычно максимум)
- Время: $O(\log \varepsilon)$ по требуемой точности

Сравнение подходов

Тип задачи	Что ищем	Аргументы	Пример
Бинарный поиск	индекс/значение	целые	поиск в массиве
Бинпоиск по ответу	максимальный/минимальный параметр	целые	«коровы», «принтеры»
Дихотомия	максимум/минимум функции	вещественные	унимодальные функции