Поиск

Постановка задачи поиска

Пусть задано множество данных, которое описывается как ***массив***, состоящий из N элементов.

Требуется найти элемент обладающий заданным свойством.

Примеры

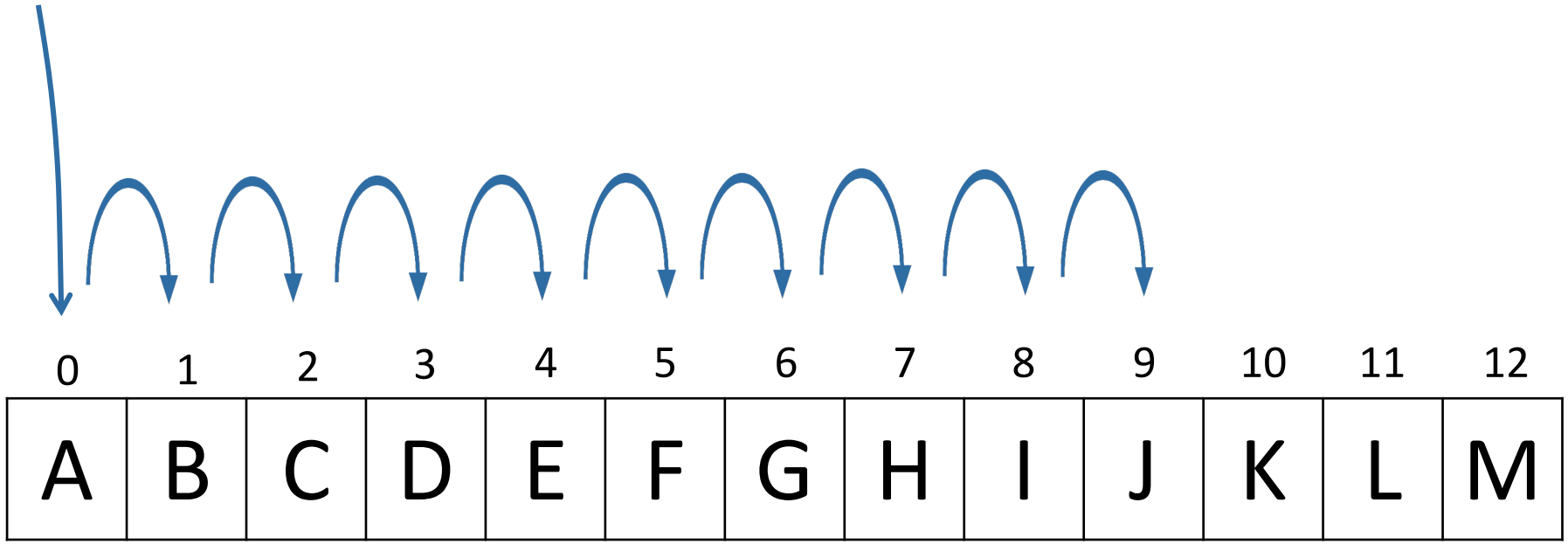
- Найти максимум/минимум/индекс максимума и минимума;
- Найти количество вхождений элемента;
- Количество элементов в интервале;
- И т.д.

Линейный поиск (linear search)

- Данный алгоритм является простейшим алгоритмом поиска
- Не накладывает никаких ограничений на функцию
- Имеет простейшую реализацию
- Поиск осуществляется простым сравнением очередного рассматриваемого значения множества данных с заданным значением, и если значения совпадают (с той или иной точностью), то поиск считается завершённым. **Внимание:** учесть ситуацию, когда заданного ключа нет в массиве

Пример

Найти "J"



Линейный поиск (linear search)

- Асимптотическая сложность - $O(n)$
- Худший случай – искомого элемента не существует
- Лучший случай – первый элемент является искомым

```
1.int function LinearSearch (Array A, int L, int R, int Key);
2.begin
3.  for X = L to R do
4.    if A[X] = Key then
5.      return X
6.  return -1; // элемент не найден
7.end;
```

Плюсы и минусы

- Плюсы
  1. Самая простая реализация поиска
- Минусы
  1. Алгоритм работает очень медленно

Бинарный поиск (binary search)

- Бинарный поиск производится в упорядоченном массиве
- Алгоритм может быть определен в рекурсивной и нерекурсивной формах
- При бинарном поиске искомый ключ сравнивается с ключом среднего элемента в массиве. Если они равны, то поиск успешен. В противном случае поиск осуществляется аналогично в левой или правой частях массива. В зависимости от постановки задачи, мы можем остановить процесс, когда мы получим первый или же последний индекс вхождения элемента. Последнее условие — это левосторонний/правосторонний двоичный поиск.

Бинарный поиск (binary search)

Пусть a – упорядоченный массив

Для простоты дальнейших определений будем считать, что $a[-1] = -\infty$ и что $a[n] = +\infty$ (массив нумеруется с 0).

Правосторонний бинарный поиск (англ. *rightside binary search*) – бинарный поиск, с помощью которого мы ищем $\max\limits_{i \in [-1, n-1]} \{i \mid a[i] \leq x\}$,

где a – массив, а x – искомый ключ

Левосторонний бинарный поиск (англ. *leftside binary search*) – бинарный поиск, с помощью которого мы ищем $\min\limits_{i \in [0, n]} \{i \mid a[i] \geq x\}$,

где a – массив, а x – искомый ключ

Использовав эти два вида двоичного поиска, мы можем найти отрезок позиций $[l, r]$ таких, что $\forall i \in [l, r]: a[i] = x$ и $\forall i \notin [l, r]: a[i] \neq x$

Бинарный поиск (binary search)

Задан отсортированный массив $[1, 2, 2, 2, 2, 3, 5, 8, 9, 11], x = 2$

Правосторонний поиск двойки выдаст в результате 4, в то время как левосторонний выдаст 1 (нумерация с нуля).

Отсюда следует, что количество подряд идущих двоек равно длине отрезка $[1;\ 4]$, то есть 4.

Если искомого элемента в массиве нет, то правосторонний поиск выдаст максимальный элемент, меньший искомого, а левосторонний наоборот, минимальный элемент, больший искомого.

Напоминание

В некоторых языках программирования присвоение $m = (l + r) / 2$ приводит к переполнению. Вместо этого рекомендуется использовать $m = l + (r - l) / 2$; или эквивалентные выражения

Бинарный поиск (binary search)

- Асимптотическая сложность - $O(\log n)$
- Худший случай – искомый элемент первый/последний
- Лучший случай – средний элемент является искомым

```
1. int binSearch(int[] a, int key):   // Запускаем бинарный поиск
2.     int l = -1                      // l, r – левая и правая границы
3.     int r = len(a)
4.     while l < r - 1                 // Запускаем цикл
5.         m = (l + r) / 2             // m – середина области поиска
6.         if a[m] < key
7.             l = m
8.         else
9.             r = m                   // Сужение границ
10.    return r
```

Плюсы и минусы

- Плюсы
  1. Реализация алгоритма достаточна легкая
  2. Быстрая работа алгоритма
- Минусы
  1. Для поиска массив должен быть упорядочен
  2. Двоичный поиск должен иметь возможность обратится к любому элементу данных (по индексу). А это значит, что все структуры данных, которые построены на связных списках использоваться не могут

Бинарный поиск. Пример

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Бинарный поиск. Пример

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Бинарный поиск. Пример

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Бинарный поиск. Пример

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16

Реализации бинарного поиска в стандартной библиотеке C++

// Параметр value должен иметь тип, на который указывают итераторы.

// Возвращает итератор на первый элемент, больший либо равный value
iterator std::lower_bound(iterator begin, iterator end, int value);

// Возвращает итератор на первый элемент, строго больший value
iterator std::upper_bound(iterator begin, iterator end, int value);

[lower_bound(x); upper_bound(x))

iterator lower_bound(iterator begin, iterator end, int value, comparator comp);
iterator upper_bound(iterator begin, iterator end, int value, comparator comp);

Задача

- К окончанию школы у Пети накопилось n дипломов, причём все они имели одинаковые размеры: w — в ширину и h — в высоту. Петя решил украсить свою комнату, повесив на одну из стен свои дипломы. Он решил купить специальную доску, чтобы прикрепить её к стене, а к ней — дипломы. Петя хочет, чтобы доска была квадратной. Каждый диплом должен быть размещён строго в прямоугольнике размером w на h. Дипломы запрещается поворачивать на 90 градусов. Прямоугольники, соответствующие различным дипломам, не должны иметь общих внутренних точек. Требуется написать программу, которая вычислит минимальный размер стороны доски, которая потребуется Пете для размещения всех своих дипломов.

Бинарный поиск по ответу

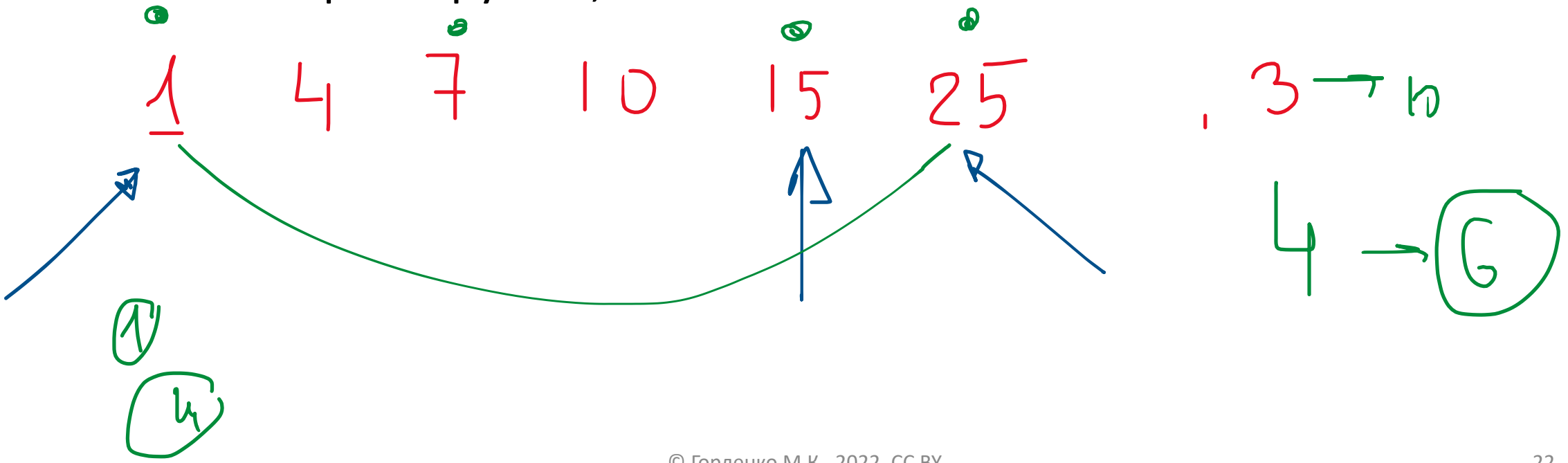
- Задачи, в которых требуется найти какое-либо значение, часто могут быть решены при помощи алгоритма бинарного поиска. В этом случае говорят о бинарном поиске по ответу. Для решения таких задач необходимо определить исходную область поиска: левую и правую границу поиска. Область поиска как раз и представляет собой упорядоченное множество потенциальных ответов, среди которых нужно выбрать тот, который удовлетворяет условию задачи. Важно предварительно сформулировать условие перехода в левую или правую половину области поиска на каждом шаге алгоритма. Для этого, как правило, происходит вычисление значения данной в задаче характеристики и переход в левую или правую область поиска в зависимости от того, больше или меньше эта характеристика, чем та, которая требуется для ответа.

Задача

```
1. l = 0; r = (w + h) * n + 1;
2. while (r - l > 1)
3. {
4.     int x = (r + l) / 2;
5.     if ((x / w) * (x / h) >= n)
6.             r = x;
7.     else
8.             l = x;
9.  }
10. cout << r;
```

Задача

- На прямой расположены n стойл (даны их координаты на прямой), в которые необходимо расставить k коров так, чтобы минимальное расстояние между коровами было как можно больше. Гарантируется, что 1<k<n.



Задача

```
1. bool check(int x) {
2.     int cows = 1;
3.     int last_cow = coords[0];
4.     for (int c : coords) {
5.         if (c - last_cow >= x) {
6.             cows++;
7.             lastcow = c;
8.             }
9.         }
10.    return cows >= k;
11.}
```

$n \log n + \log(r-l) \cdot n$

```
1. bool solve(int x) {
2.     sort(coords.begin(), coords.end());
3.      int l = 0; /* Так как коров меньше, чем
   стойл, x = 0 нам всегда хватит по условию
   есть хотя бы 2 коровы, которых мы в лучшем
   случае отправим в противоположные стойла: */
4.     int r = coords.back() - coords[0] + 1;
5.     while (r - l > 1) {
6.         int m = (l + r) / 2;
7.         if (check(m))
8.             l = m;
9.         else
10.            r = m;
11.    }
12.    return l;
13.}
```

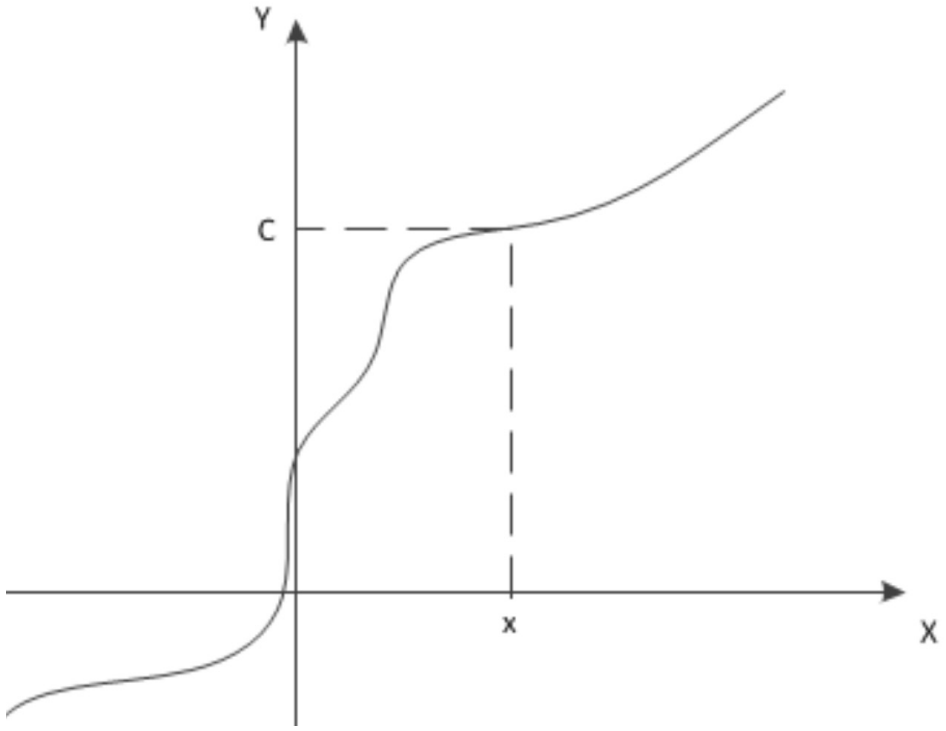
- 0 1 2. 3. 4. 5. 6
- 3 5 10 19 23 30 35, x = 4
- L = 1; R = 32; mid = 16 -> 3 коровы
- L = 1; R = 16; mid = 8 -> 3 коровы
- L = 1; R = 8; mid = 4 -> 6 коров
- L = 4; R = 8; mid = 6 -> 4 коровы
- L = 6; R = 8; mid = 7 -> 4 коровы
- L = 7; R = 8;

Вещественный бинарный поиск (Bisection method)

- Вещественный двоичный поиск (англ. Bisection method)— алгоритм поиска аргумента для заданного значения монотонной вещественной функции.

Постановка задачи поиска

Пусть нам задана монотонная функция f и какое-то значение C этой функции. Необходимо найти значение аргумента x этой функции, такое, что $f(x) = C$.



Вещественный бинарный поиск (Bisection method)

- Примерное количество итераций алгоритма $O\left(\log\frac{rigth-left}{eps}\right)$
- Критерии останова:
  - Заданная точность найденного значения;
  - Значение функции от найденного значения имеет заданную точность;
  - Максимально возможная точность найденного значения;
  - Заданное количество итераций.

Вещественный бинарный поиск (Bisection method)

- Примерное количество итераций алгоритма $O\left(\log\frac{rigth-left}{eps}\right)$

```
1. double findLeftBoard(C : double):
2.     x = -1
3.     while f(x) > C
4.         x = x * 2
5.     return x
```

```
1. double findRightBoard(C : double):
2.     x = 1
3.     while f(x) < C
4.         x = x * 2
5.     return x
```

```
1. double binSearch(C : double):
2.     left = findLeftBoard(C)
3.     right = findRightBoard(C)
4.     while right - left < eps
5.         mid = (left + right) / 2
6.         if f(mid) < C
7.             left = mid
8.         else
9.             right = mid
10.    return (left + right) / 2
```

Замечания

- Необходимо отметить, то функция должна быть:
  - строго монотонна, если мы ищем конкретный корень и он единственный;
  - нестрого монотонна, если нам необходимо найти самый левый (правый) аргумент. Если же функция не монотонна, то данный алгоритм не найдет искомый аргумент, либо найдет аргумент, но он не будет единственным.