Лекция 3. Поиск

E-mail: lazycoyote11@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2026 г.

Задача поиска элемента по ключу

- Дана последовательность из *n* ключей

$$a_1, a_2, \ldots, a_n$$

- Требуется найти номер (индекс) элемента, совпадающего с заданным ключом *key*

Пример

- Дана последовательность из 9 ключей
- Требуется найти элемент с ключом *key* = 55

index	1	2	3	4	5	6	7	8	9
key	63	19	55	65	22	38	71	36	32

index = 3

Линейный поиск (linear search)

```
1 function LinearSearch(A[1:n], n, key)
2     for i = 1 to n do
3         if A[i] = key then
4             return i
5         end if
6     end for
7     return -1
8 end function
```

$T_{LinearSearch} = O(n)$

- Просматриваем элементы, начиная с первого, и сравниваем ключи
- В худшем случае искомый элемент находится в конце массива или отсутствует
- Количество операций в худшем случае (*worst case*):

$$T(n) = O(n)$$

LinearSearch(A, 9, 55)

index	1	2	3	4	5	6	7	8	9
key	63	19	55	65	22	38	71	36	32

i

Линейный поиск (linear search)

```
1 function LinearSearch(A[1:n], n, key)
2     for i = 1 to n do
3         if A[i] = key then
4             return i
5         end if
6     end for
7     return -1
8 end function
```

$T_{LinearSearch} = O(n)$

- Просматриваем элементы, начиная с первого, и сравниваем ключи
- В худшем случае искомый элемент находится в конце массива или отсутствует
- Количество операций в худшем случае (*worst case*):

$$T(n) = O(n)$$

LinearSearch(A, 9, 55)

index	1	2	3	4	5	6	7	8	9
key	63	19	55	65	22	38	71	36	32

i

Линейный поиск (linear search)

```
1 function LinearSearch(A[1:n], n, key)
2     for i = 1 to n do
3         if A[i] = key then
4             return i
5         end if
6     end for
7     return -1
8 end function
```

$T_{LinearSearch} = O(n)$

- Просматриваем элементы, начиная с первого, и сравниваем ключи
- В худшем случае искомый элемент находится в конце массива или отсутствует
- Количество операций в худшем случае (*worst case*):

$$T(n) = O(n)$$

LinearSearch(A, 9, 55)

index	1	2	3	4	5	6	7	8	9
key	63	19	55	65	22	38	71	36	32

i

Линейный поиск (linear search)

```
1 function LinearSearch(A[1:n], n, key)
2     for i = 1 to n do
3         if A[i] = key then
4             return i
5         end if
6     end for
7     return -1
8 end function
```

$T_{LinearSearch} = O(n)$

- Просматриваем элементы, начиная с первого, и сравниваем ключи
- В худшем случае искомый элемент находится в конце массива или отсутствует
- Количество операций в худшем случае (*worst case*):

$$T(n) = O(n)$$

LinearSearch(A, 9, 32)

index	1	2	3	4	5	6	7	8	9
key	63	19	55	65	22	38	71	36	32

i

Бинарный поиск (binary search)

- Имеется **упорядоченная** последовательность ключей

$$a_1 \le a_2 \le \ldots \le a_i \le \ldots \le a_n$$

- Требуется найти позицию элемента, ключ которого совпадает с заданным ключом *key*
- Бинарный поиск *(binary search)*
  1. Если центральный элемент **равен** искомому, **конец** алгоритма
  2. Если центральный элемент **меньше**, делаем текущей **правую** половину массива
  3. Если центральный элемент **больше**, делаем текущей **левую** половину массива

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2
6          if A[mid] = key then
7              return mid
8          else if key > A[mid] then
9              low = mid + 1
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

- `mid = (low + high) / 2`: возможно переполнение *mid*
- Решение: `low + (high - low) / 2`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2
6          if A[mid] = key then
7              return mid
8          else if key > A[mid] then
9              low = mid + 1
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

BinarySearch(A[1:19], 19, 41)

- mid = (low + high) / 2:
- mid = (1 + 19) / 2 = 10

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2
6          if A[mid] = key then
7              return mid
8          else if key > A[mid] then
9              low = mid + 1
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

BinarySearch(A[1:19], 19, 41)

- mid = (low + high) / 2:
- mid = (11 + 19) / 2 = 15

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2
6          if A[mid] = key then
7              return mid
8          else if key > A[mid] then
9              low = mid + 1
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

BinarySearch(A[1:19], 19, 41)

- mid = (low + high) / 2:
- mid = (11 + 14) / 2 = 12

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2
6          if A[mid] = key then
7              return mid
8          else if key > A[mid] then
9              low = mid + 1
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

Худший случай:

`BinarySearch(A[1:19], 19, 72)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

```
1  function BinarySearch(A[1:n], n, key)
2      low = 1
3      high = n
4      while low <= high do
5          mid = (low + high) / 2     // 2 операции
6          if A[mid] = key then       // 2 операции
7              return mid
8          else if key > A[mid] then  // 2 операции
9              low = mid + 1          // 1 операция
10         else
11             high = mid - 1
12         end if
13     end while
14     return -1
15 end function
```

Количество операций в худшем случае:

$$T(n) = 2 + kT_{while} + 1 = 7k + 3$$

k – количество итераций цикла *while*

T_{while} – количество операций в теле цикла *while*

$T_{while} = 2 + 2 + 2 + 1 = 7$

Бинарный поиск (binary search)

Количество *k* разбиений массива:

- Массив длины *n*
- Массив длины *n* / 2
- Массив длины *n* / 4
- ...
- Массив длины *n* / 2^k = 1

$$\frac{n}{2^k}=1; \qquad n=2^k$$

$$\log_2 n=\log_2 2^k$$

$$k=\log_2 n$$

$$T(n)=c_1\log_2 n+c_2=O(\log n)$$

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

Бинарный поиск (binary search)

Бинарный поиск неэффективно использует кеш-память процессора:

доступ к элементам массива непоследовательный (прыжки по массиву)

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`GallopingSearch(A[1:19], 19, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`GallopingSearch(A[1:19], 19, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`GallopingSearch(A[1:19], 19, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`GallopingSearch(A[1:19], 19, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`BinarySearch(A[8:14], 7, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

«Поиск от края» (galloping search)

- Задан отсортированный массив A[*n*]
- Алгоритм поиска от края проверяет ключи с индексами

$$1, 3, 7, 15, \ldots, 2^i - 1, \ldots$$

- Проверка идёт до тех пор, пока не будет найден элемент

$$A[2^i - 1] > key$$

- Далее выполняется бинарный поиск в интервале

$$2^{i-1} + 1, \ldots, 2^i - 1$$

- $T_{Galloping}(n) = O(\log n)$

`BinarySearch(A[8:14], 7, 28)`

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
2	6	9	12	16	19	22	26	28	34	39	41	46	48	52	59	61	64	72

- Galloping search = one-sided binary search, exponential search, doubling search
- J. L. Bentley, A. C.-C. Yao. An almost optimal algorithm for unbounded searching // Information processing letters, 5(3):82—87, 1976

Поиск в массиве

```
function Search(A[1:n], n, key)
    if issorted = false then
        Sort(A, n)
        issorted = true
    end if
    return GallopSearch(A, n, key)
end function
```

- Задан неупорядоченный массив ключей, новые элементы добавляются крайне редко
- Требуется периодически осуществлять поиск в массиве
- Решение 1, «в лоб»
  - Каждый раз при поиске использовать линейный поиск за O(n)
- Решение 2, в среднем за O(log$n$)
  - Один раз отсортировать массив за O($n$log$n$) или за O(n + k)
  - Использовать экспоненциальный поиск (*galloping search*) за O(log$n$)

Дальнейшее чтение

- **[DSABook]** Глава 4. «Поиск».