Лекция 4.
Поиск. Абстрактные типы данных

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.

Задача поиска элемента по ключу

- Имеется последовательность ключей

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

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

Пример

- **Дана** последовательность из 10 ключей
- **Требуется** найти элемент с ключом *key* = 148

Index	1	2	3	4	5	6	7	8	9	10
Key	187	192	144	126	148	133	208	196	110	108
Data										

- **Решение:** искомый элемент с индексом **5**

Линейный поиск (linear search)

```
function LinearSearch(A[1:n], n, key)
    for i = 1 to n do
        if A[i] = key then
            return i
        end if
    end for
    return -1
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):

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

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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	9	15	22	41	44	49	52	60	78	82	83	91	94	96	112	121	128	131	133	140	142

Поиск *key* = 94

Бинарный поиск (binary search)

```
function BinarySearch(A[1:n], n, key)
    low = 1          /* Левая граница массива */
    high = n         /* Правая граница массива */
    while low <= high do
        mid = (low + high) / 2
        /* Возможно переполнение mid */
        /* Решение: low + (high - low) / 2 */
        if A[mid] = key then
            return mid
        else if key > A[mid] then
            low = mid + 1
        else
            high = mid - 1
        end if
    end while
    return -1
end function
```

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	9	15	22	41	44	49	52	60	78	82	83	91	94	96	112	121	128	131	133	140	142

key = 144 (worst case)

Бинарный поиск (binary search)

```
function BinarySearch(A[1:n], n, key)
    low = 1          /* Левая граница массива */
    high = n         /* Правая граница массива */
    while low <= high do
        mid = (low + high) / 2      /* 2 оп. */
        /* Возможно переполнение mid */
        /* Решение: low + (high - low) / 2 */
        if A[mid] = key then        /* 2 оп. */
            return mid
        else if key > A[mid] then   /* 2 оп. */
            low = mid + 1           /* 1 оп. */
        else
            high = mid - 1
        end if
    end while
    return -1
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*
- $T_{while} = 2 + 2 + 2 + 1 = 7$

1	2																			21	22
4	9																			140	142

Бинарный поиск (binary search)

```
function BinaryS
    low = 1
    high = n
    while low <=
        mid = (l
        /* Возмо
        /* Решен
        if A[mid
            retu
```

Количество *k* разбиений массива:

- Массив длины n
- Массив длины $n / 2$
- Массив длины $n / 4$
- …
- Массив длины $n / 2^k = 1$

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

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

$$k = \log_2 n$$

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

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	9	15	22	41	44	49	52	60	78	82	83	91	94	96	112	121	128	131	133	140	142

key = 144 (worst case)

Бинарный поиск (binary search)

Бинарный поиск неэффективно использует кеш-память процессора: доступ к элементам массива непоследовательный (прыжки по массиву)

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	9	15	22	41	44	49	52	60	78	82	83	91	94	96	112	121	128	131	133	140	142

key = 144 (worst case)

«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

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

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

Поиск *key* = 82

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
4	9	15	22	41	44	49	52	60	78	82	83	91	94	96	112	121	128	131	133	140	142

- 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

- **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 $O(\log n)$

```
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
```

Понятие типа данных (data type)

- Язык программирования позволяет оперировать с величинами различного вида: строки, числа, логические значения
- **Тип данных** (data type) — это атрибут любого значения, которое встречается в программе
- Тип данных определяет две характеристики:
  - *множество допустимых значений*, которые могут принимать данные, принадлежащие к этому типу
  - *набор операций*, которые можно выполнять над данными этого типа

Типы данных (data type)

- **Базовые (примитивные) типы данных:**
  int, char, bool, string

- **Составные типы данных:**
  (агрегатные, структурные, композитные типы) — это типы данных, которые формируются на основе базовых (например, массивы, структуры и классы в языках C и C++)

- **Структура данных** (data structure) — программная единица, реализующая хранение и выполнение операций над совокупностью однотипных элементов
- Набор функций для выполнения операций над структурой данных называется её **интерфейсом** (interface)

Структура данных «связный список»

```c
/* Узел односвязного списка */
struct listnode {
    int value;                /* Значение узла */
    struct listnode *next;    /* Указатель на следующий узел */
};

/* list_addfront: добавление узла в начало списка */
struct listnode *list_addfront(struct listnode *list, int value);

/* list_lookup: поиск узла с заданным значением */
struct listnode *list_lookup(struct listnode *list, int value);

/* list_delete: удаление узла с заданным значением */
struct listnode *list_delete(struct listnode *list, int value);
```

Абстрактные типы данных

- **Абстрактный тип данных** (АТД, abstract data type) — это тип данных, который задан описанием своего интерфейса
- Способ хранения данных в памяти компьютера и алгоритмы работы с ними сокрыты внутри функций интерфейса (в этом суть абстракции)
- Если описать реализацию функций АТД, получим конкретную структуру данных

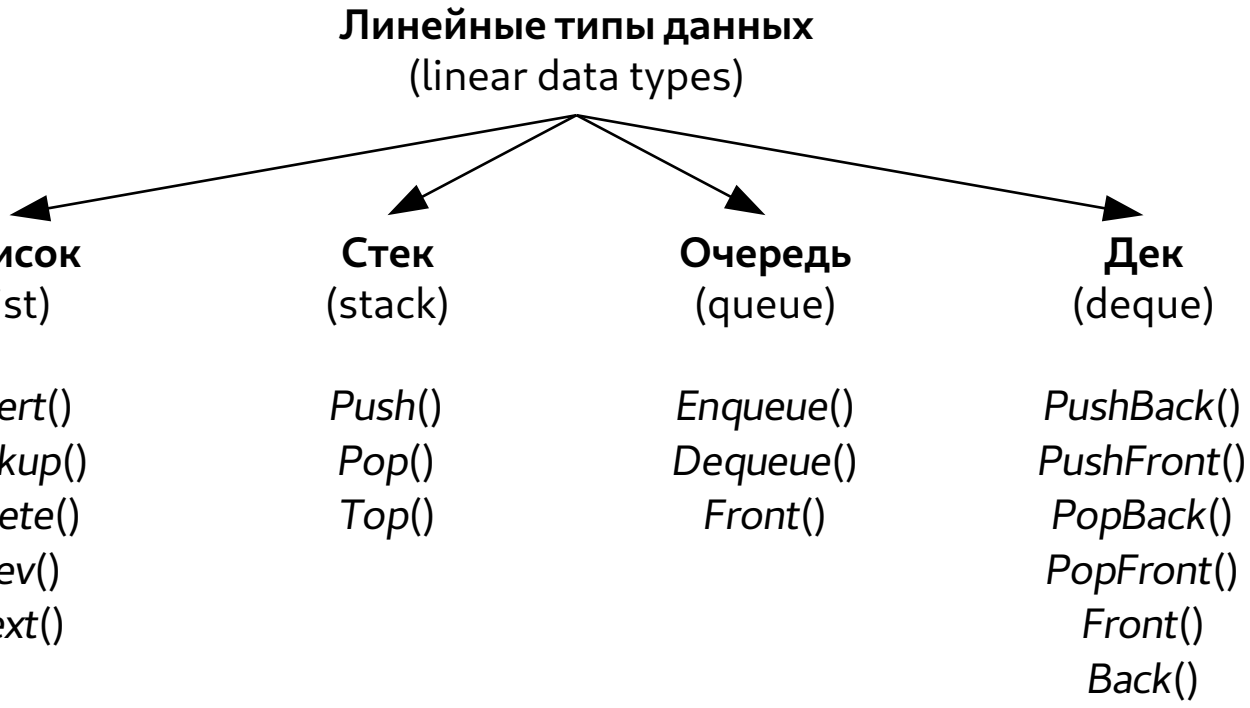
Абстрактный тип данных «список»

- **Интерфейс АТД «список» (list)**
  - Insert
  - Delete
  - Lookup
  - Next
  - Prev

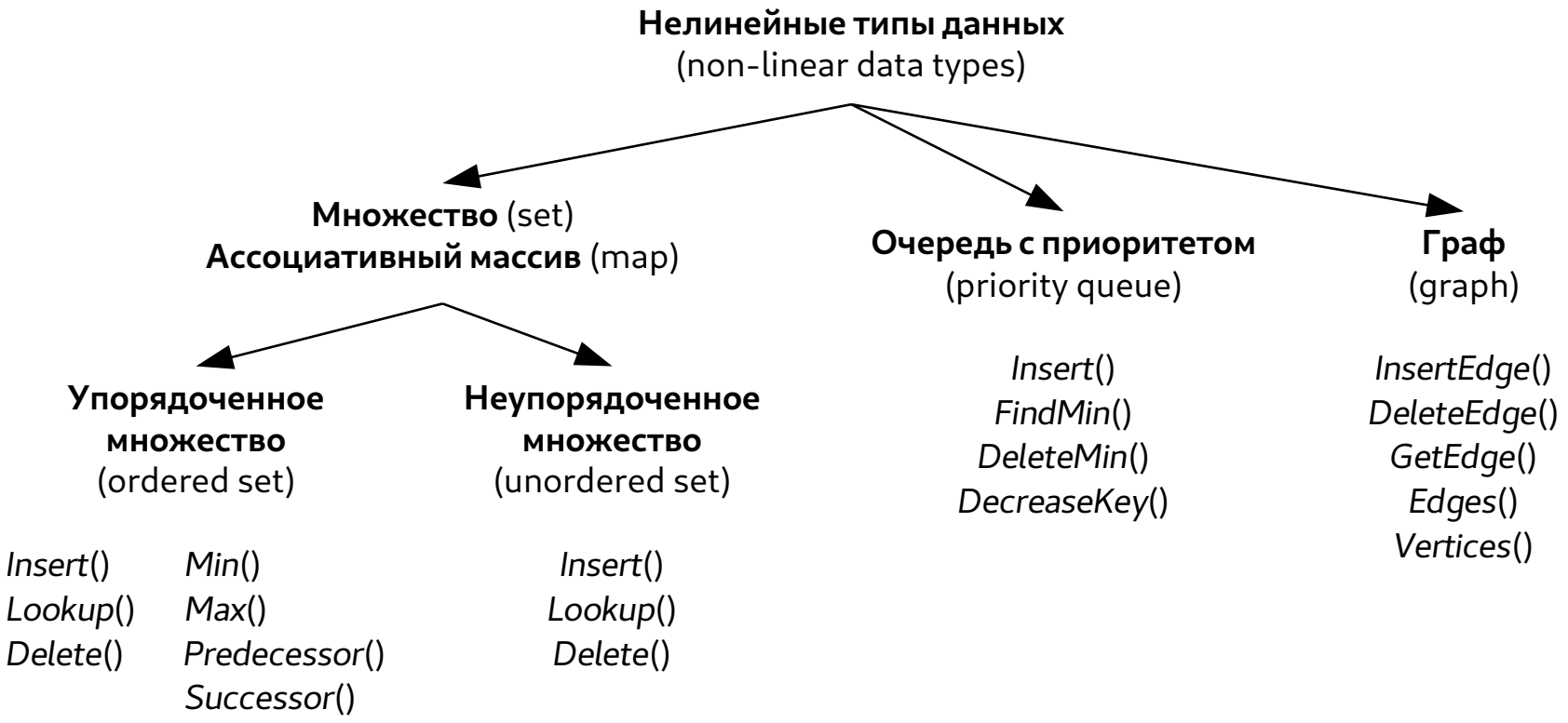
- **Возможные реализации АТД «список»**
  - Статический массив
  - Связный список

Вычислительная сложность и сложность по памяти функций разных реализаций могут быть различными

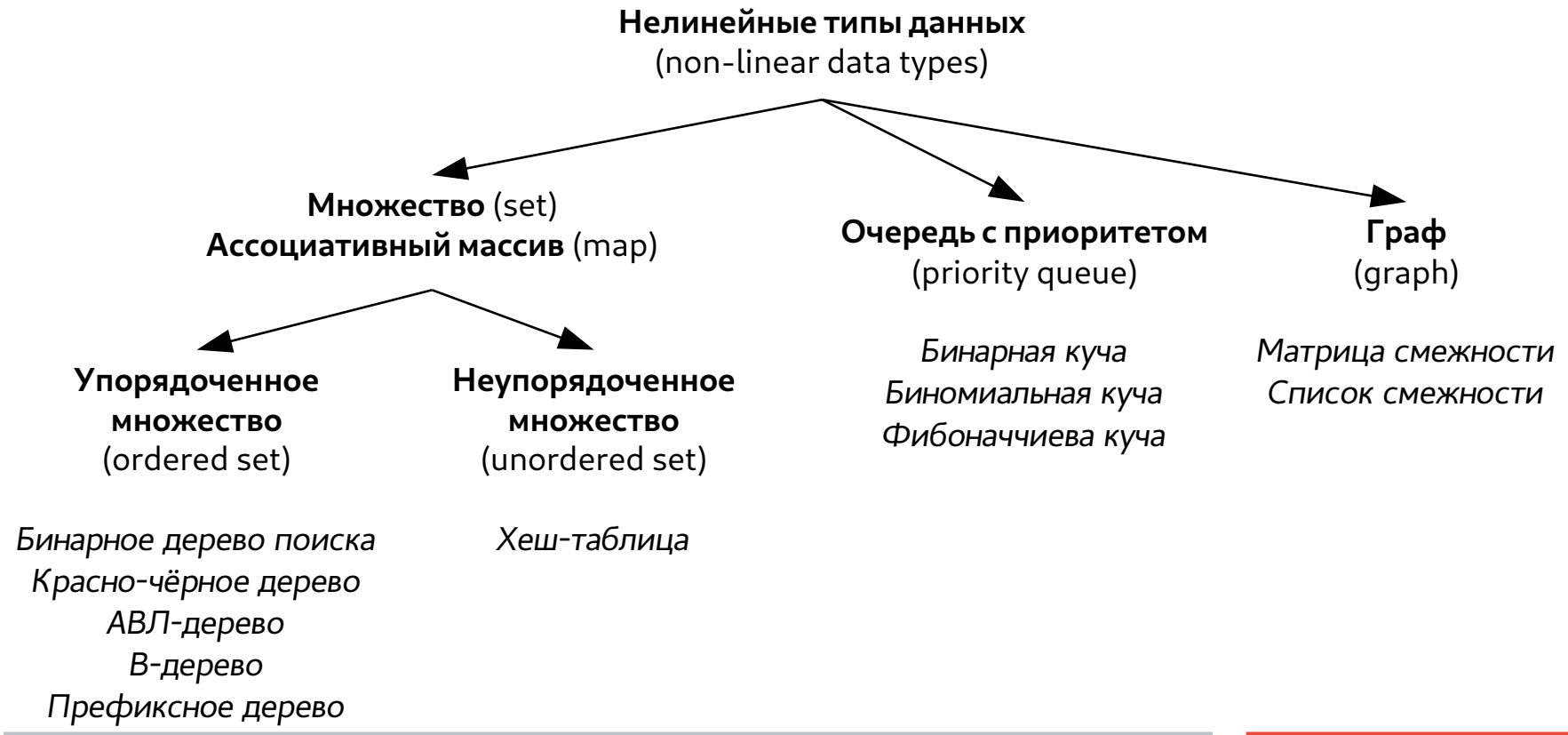
Линейные типы данных



Нелинейные типы данных

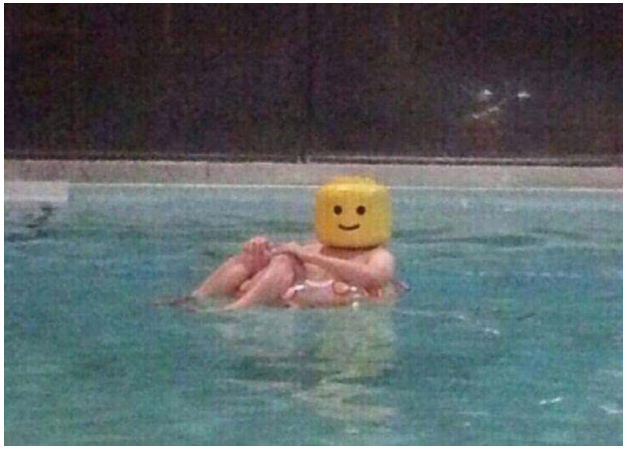


Нелинейные типы данных



Домашнее чтение

- [DSABook] Глава 4. «Поиск». Глава 5. «Абстрактные типы данных»



ご清聴ありがとうございました!

Даниил Михайлович Берлизов
Старший преподаватель Кафедры вычислительных систем СибГУТИ
E-mail: sillyhat34@gmail.com

Курс «Структуры и алгоритмы обработки данных»
Весенний семестр, 2021 г.