Лекция 4
Поиск
Абстрактные типы данных

Курносов Михаил Георгиевич

E-mail: mkurnosov@gmail.com
WWW: www.mkurnosov.net

Курс «Структуры и алгоритмы обработки данных»
Сибирский государственный университет телекоммуникаций и информатики (Новосибирск)
Весенний семестр, 2016

Задача поиска элемента по ключу

- **Имеется** последовательность ключей

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

- **Требуется** найти номер элемента, который совпадает с заданным ключом *key*

Пример

Дана последовательность из 10 ключей
Требуется найти элемент с ключом *key* = 181

Index	1	2	3	4	5	6	7	8	9	10
Key	178	150	190	177	155	181	179	167	204	175
Data										

Решение – искомый элемент с индексом **6**

Линейный поиск (linear search)

```
function LinearSearch(v[1..n], n, value)
    for i = 1 to n do
        if v[i] = value then
            return i
        end if
    end for
    return -1
end function
```

$$T_{LinearSearch} = O(n)$$

- Просматриваем элементы с 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** <u>упорядоченная</u> последовательность ключей

$$a_1 \le a_2 \le \cdots \le a_i \le \cdots \le a_n$$

- **Требуется** найти позицию элемента, ключ которого совпадает с заданным ключом *key*

- **Бинарный поиск (Binary search)**

1. Если центральный элемент равен искомому, конец
2. Если центральный меньше, делаем текущей правую половину массива
3. Если центральный 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	23
2	**3**	**3**	**5**	**7**	**9**	**13**	**23**	**25**	**29**	**31**	**33**	**37**	**41**	**42**	**46**	**49**	**50**	**52**	**67**	**73**	**81**	**94**

Поиск *key* = 42

Бинарный поиск (Binary Search)

```
function BinarySearch(v[1..n], n, key)
    l = 1       // Левая граница массива (low)
    h = n       // Правая граница массива (high)
    while l <= h do
        mid = (l + h) / 2    // Возможно переполнение mid,
                             // Решение: mid = l + ((h - l) / 2)
        if v[mid] = key then
            return mid
        else if key > v[mid] then
            l = mid + 1
        else
            h = 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	23
2	**3**	**3**	**5**	**7**	**9**	**13**	**23**	**25**	**29**	**31**	**33**	**37**	**41**	**42**	**46**	**49**	**50**	**52**	**67**	**73**	**81**	**94**

***key* = 100 (worst case)**

Эффективность бинарного поиска

```
function BinarySearch(v[1..n], n, key)
    l = 1       // Левая граница массива (low)
    h = n       // Правая граница массива (high)
    while l <= h do
        mid = (l + h) / 2      // 2 операции
        if v[mid] = key then   // 2 оп.
            return mid
        else if key > v[mid] then  // 2 оп.
            l = mid + 1            // 1 оп.
        else
            h = 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
- $T_{while} = 2 + 2 + 2 + 1 = 7$

Эффективность бинарного поиска

```
function BinarySearch(v[1..n],
    l = 1       // Левая грани
    h = n       // Правая грані
    while l <= h do
        mid = (l + h) / 2
        if v[mid] = key then
            return mid
        else if key > v[mid] -
            l = mid + 1
        else
            h = mid - 1
        end if
    end while
    return -1
end function
```

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	23
2	**3**	**3**	**5**	**7**	**9**	**13**	**23**	**25**	**29**	**31**	**33**	**37**	**41**	**42**	**46**	**49**	**50**	**52**	**67**	**73**	**81**	**94**

***key* = 100**
(worst case)

Эффективность бинарного поиска

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	23
2	**3**	**3**	**5**	**7**	**9**	**13**	**23**	**25**	**29**	**31**	**33**	**37**	**41**	**42**	**46**	**49**	**50**	**52**	**67**	**73**	**81**	**94**

***key* = 100**
(worst case)

Galloping search – «поиск от края»

- **Задан** отсортированный массив $A[n]$
- Алгоритм Galloping проверяет ключи с индексами
$$1, 3, 7, 15, \dots, 2^i - 1, \dots$$
- Проверка идет то тех пор, пока не будет найден элемент
$$A\left[2^i - 1\right] > key$$
- Далее выполняется бинарный поиск в интервале
$$2^{i-1} - 1, \dots, 2^i - 1$$
- $T_{Galloping}(n) = O(\log n)$

Поиск *key* = 31

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
2	**3**	**3**	**5**	**7**	**9**	**13**	**23**	**25**	**29**	**31**	**33**	**37**	**41**	**42**	**46**	**49**	**50**	**52**	**67**	**73**	**81**	**94**

- Galloping search = One sided binary search, exponential search, doubling search
- J. L. Bentley and 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 добавляться крайне редко)
- **Требуется** периодически осуществлять поиск в массиве
- **Решение 1 – “в лоб” за $O(n)$**
  Каждый раз при поиске использовать линейный поиск за $O(n)$
- **Решение 2 – в среднем за $O(\log n)$**

1. Один раз отсортировать массив за $O(n \log n)$ или за $O(n + k)$
2. Использовать экспоненциальный поиск (Galloping) – $O(\log n)$

```
function Search(v[1:n], n, key)
    if issorted = false then
        Sort(v, n)  // Устойчивая сортировка, вызывается редко
        issorted = true
    end if
    return GallopSearch(v, n, key)  // Эксп. поиск O(logn)
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»

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


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

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

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

Абстрактные типы данных

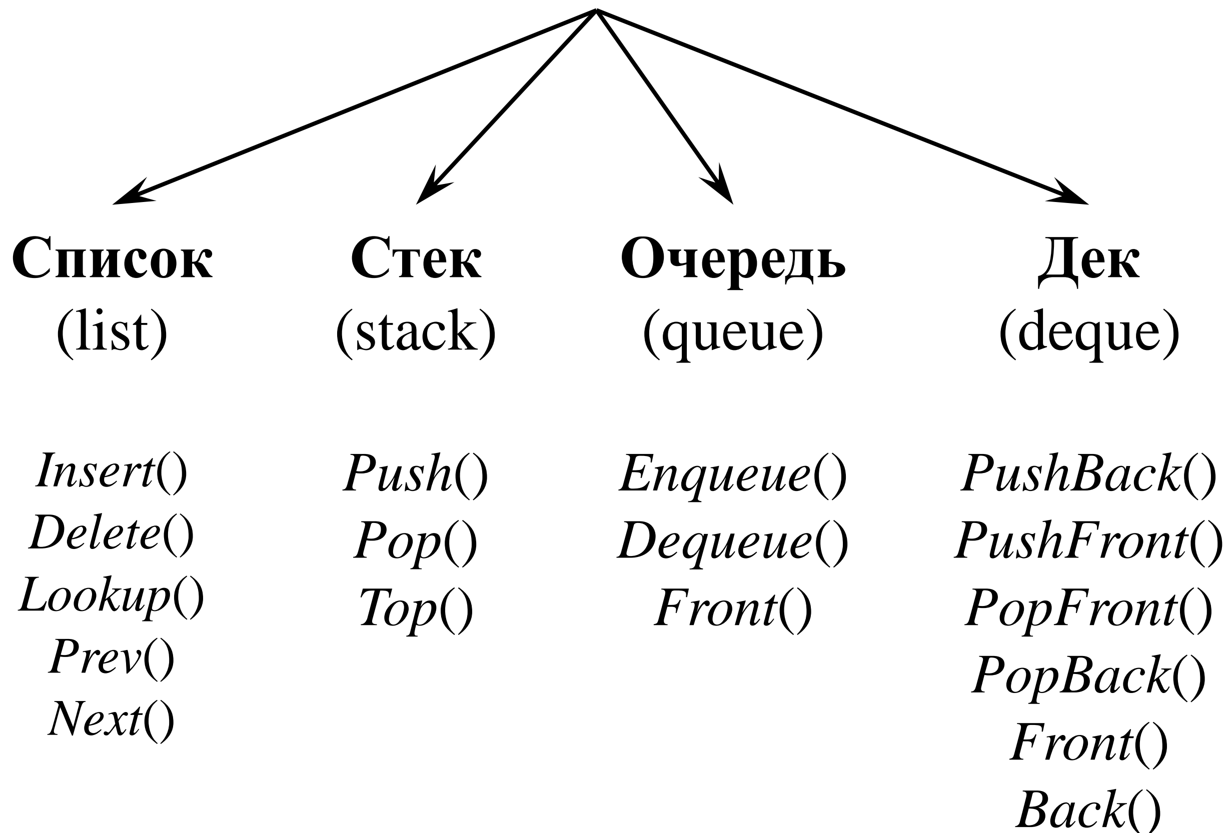
- **Абстрактный тип данных** (АТД, abstract data type) – это тип данных, который задан описанием своего интерфейса
- Способ хранения данных в памяти компьютера и алгоритмы работы с ними сокрыты внутри функций интерфейса (в этом суть абстракции)
- Если описать реализацию функций АТД, получим конкретную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

Абстрактный тип данных «список»

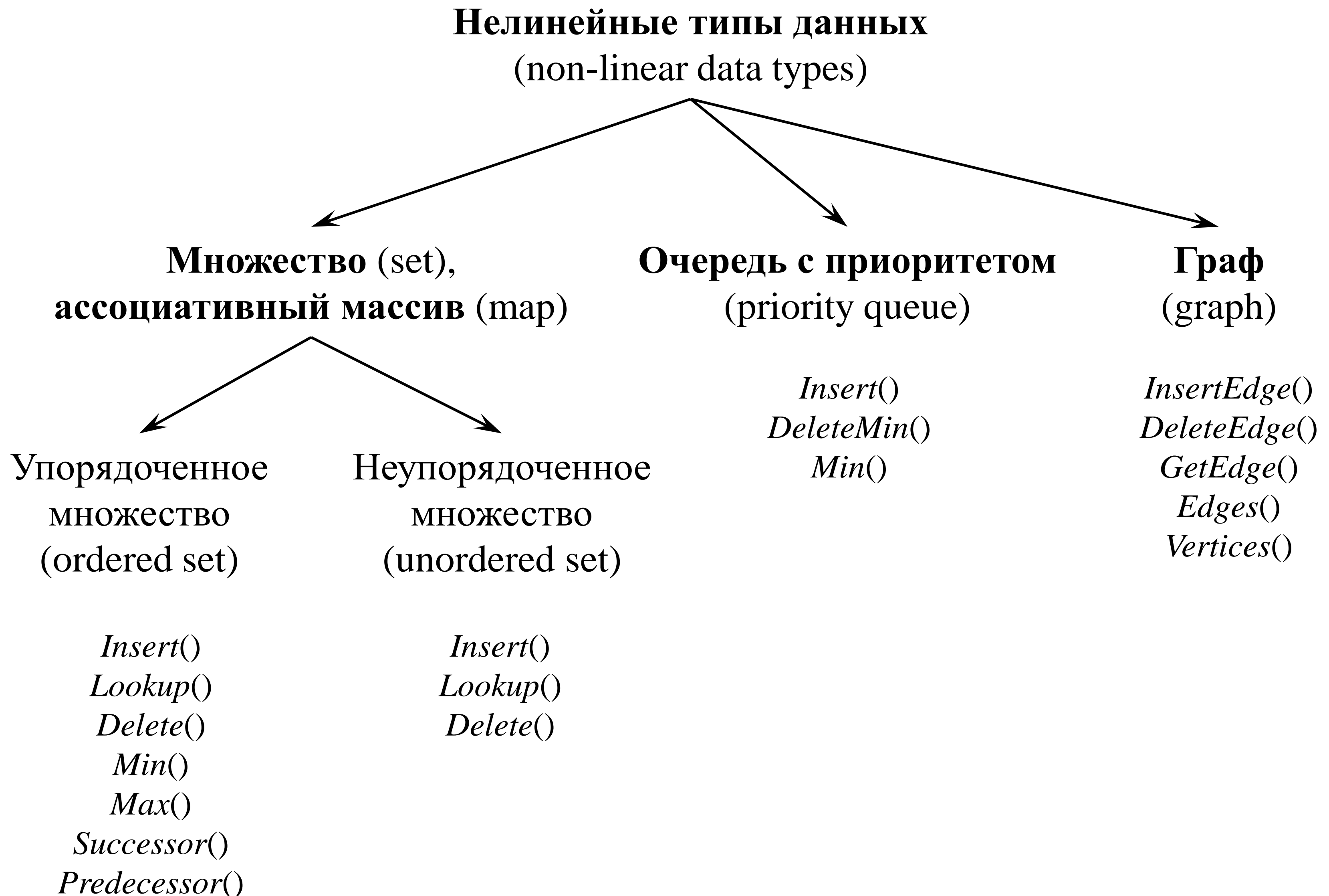
- **Интерфейс АТД список (list)**
  - Insert
  - Delete
  - Lookup
  - Next
  - Prev
- **Возможные реализации АТД список**
  - Статический массив
  - Связный список

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

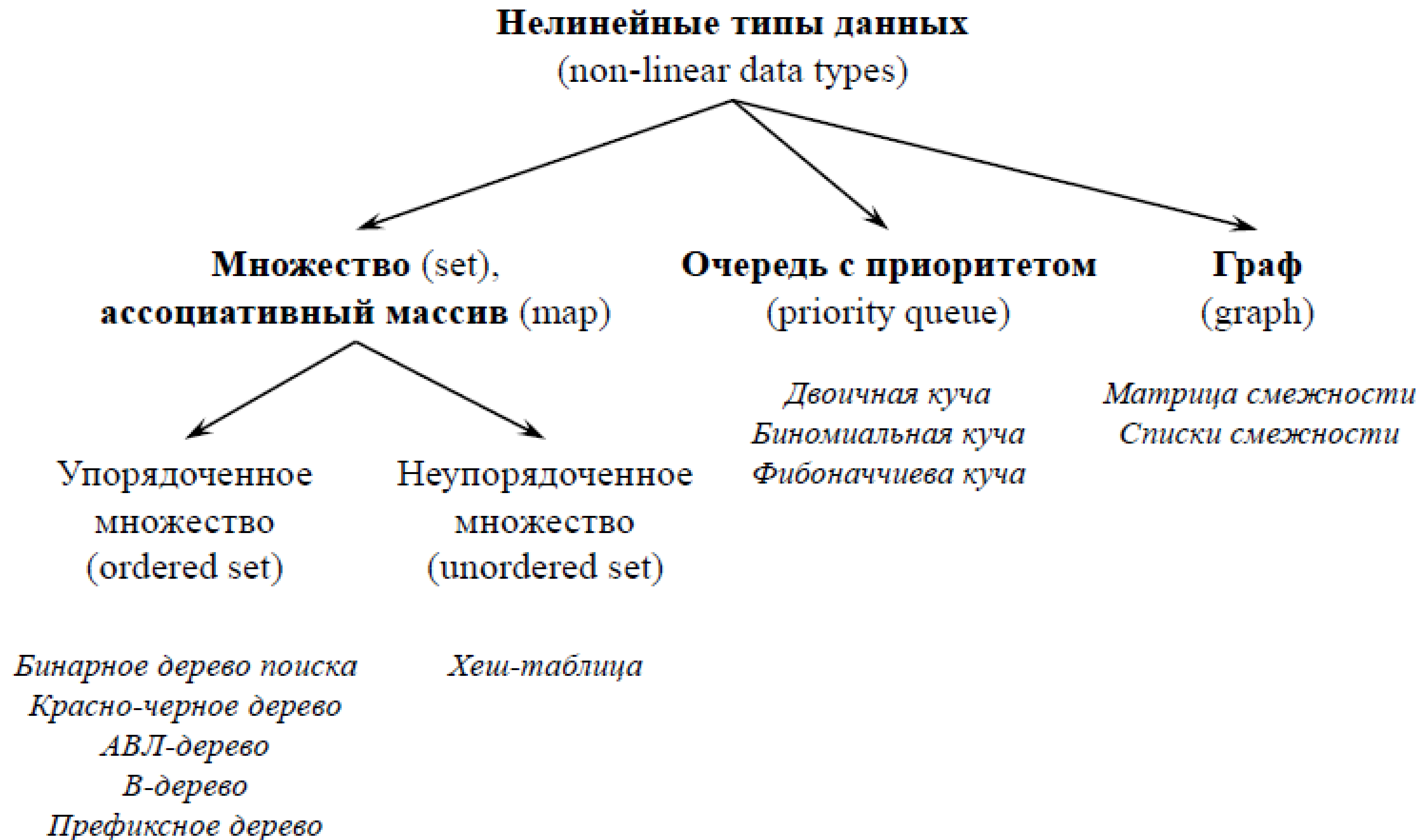
Линейные типы данных



Нелинейные типы данных



Нелинейные типы данных



Домашнее чтение

- [DSABook, Главы 4-5]