Основы программного конструирования

ЛЕКЦИЯ №4

6 МАРТА 2023

Сортировка за линейное время

- Для произвольных ключей доказали: количество сравнений не менее
- А что, если ключи не произвольные?
- Сколько операций нужно, чтобы отсортировать последовательность битов?

Сортировка подсчетом (Counting sort)

- Предположим, что все ключи – целые числа **0…K-1**.
- Заведем массив счетчиков **C[k]** размером **K**, изначально заполненный нулями.
- Подсчитаем количество вхождений каждого ключа.
- Добавим к каждому элементу **C[k]** все предыдущие, получая: **C[k]** = количество ключей не превышающих **k**.
- Идем с конца массива элементов, ставя элемент A[n] на позицию C[Key(A[n])]-1 и уменьшаем счетчик.

Свойства сортировки подсчетом

- Временная сложность **O(N+K)**
- Стабильность
- Дополнительная память:

Вспомогательный массив длины N (если стабильность не нужна, можно обойтись без него);

Массив счетчиков.

Поразрядная сортировка (Radix sort)

- Предположим, что ключи состоят из **d** «цифр»
- Алгоритм «от младшей к старшей»:
- Стабильная сортировка по 1-й цифре (младшей).
- … по 2-й цифре.
- …
- … по **d**-й цифре (старшей).

Анализ поразрядной сортировки (Radix sort)

- На практике 32-битное целое число можно разбить на 4 байта («цифры»).
- Сложность O(N).
- Идея годится для сортировки по произвольным составным ключам.

Сортировки на практике

- В прикладных программах достаточно стандартной библиотеки.
- Серьезные алгоритмы зачастую представляют смесь:
  - Быстрая + вставками (C, C++)
  - Быстрая + пирамидальная (Introsort)
  - Слиянием + вставками (Python, Java)
- Используйте алгоритм, подходящей к вашей задаче.

Задача о сортировках

Даны 7 массивов чисел:

1. C 1 4 F 2 B 8 D E 7 A 9 0 6 3 5
2. 1 4 2 B 8 C D 7 A 9 0 6 3 5 E F
3. D C B 5 A 9 8 3 1 7 2 4 0 6 E F
4. 0 1 2 3 4 B 8 D E 7 A 9 C 6 F 5
5. 1 2 4 8 B C F D E 7 A 9 0 6 3 5
6. 0 1 4 5 2 B 8 3 6 7 A 9 C E D F
7. 0 1 2 3 4 5 6 7 8 9 A B C D E F

Массив 1 — начальный.

Массив 7 — отсортированный.

Массивы 2-6 являются промежуточными состояниями при сортировке начального массива разными алгоритмами:

a. сортировка выбором,
b. пузырьковая сортировка (обычная, слева направо),
c. сортировка вставками,
d. быстрая сортировка (без перемешивания, первый элемент как медиана),
e. пирамидальная сортировка.

Определить, какой массив 2-6 соответствует какой сортировке a-e. Ответа недостаточно, нужно объяснение, почему именно этот массив может соответствовать именно этой сортировке.

Подсказка: нужно заметить характерные свойства каждой из сортировок.

Поиск подстроки в строке

- Задача: дана строка – «стог сена» длины **N**. Определить, встречается ли в ней строка – «иголка» длины **M** и если да, то на какой позиции.

Поиск подстроки в строке

- Задача: дана строка – «стог сена» длины **N**. Определить, встречается ли в ней строка – «иголка» длины **M** и если да, то на какой позиции.
- Метод грубой силы: подставляем «иголку» к каждой возможной позиции в «стоге сена», пока не найдем совпадение или «стог» не закончится.

Поиск подстроки в строке

- Задача: дана строка – «стог сена» длины **N**. Определить, встречается ли в ней строка – «иголка» длины **M** и если да, то на какой позиции.
- Метод грубой силы: подставляем «иголку» к каждой возможной позиции в «стоге сена», пока не найдем совпадение или «стог» не закончится.
- Количество сравнений: **O((N-M)*M)**

Поиск подстроки в строке

- Задача: дана строка – «стог сена» длины **N**. Определить, встречается ли в ней строка – «иголка» длины **M** и если да, то на какой позиции.
- Метод грубой силы: подставляем «иголку» к каждой возможной позиции в «стоге сена», пока не найдем совпадение или «стог» не закончится.
- Количество сравнений: **O((N-M)*M)**

Поиск подстроки в строке

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» b c **a b c b c** b a a …

«иголка» **a a a a a**

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» b c **a b c b c** b a a …

«иголка» **a a a a a**

- Явно нет смысла сдвигать «иголку» на одну позицию вправо.
- Хорошо бы как-то учесть частично совпадение строк и не проверять все символы заново каждый раз.

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» a b c b e c b a b c b c…

«иголка» **a b c b c**

- На сколько сдвигать??

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» a b c b e c b a b c b c…

«иголка» **a b c b c**

- На сколько сдвигать??

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» a b c b e c b a b c b c…

«иголка» **a b c b c**

- Сдвинули на 5 символов (символа e нет в искомой подстроке).
- На сколько двигать?

Алгоритм Боуэра-Мура (модифицированный)

«стог сена» a b c b e c b a b c b c…

«иголка» **a b c b c**

- Сдвинули на 2 символа (совмещаем символ c).

Суть алгоритма

- Подставляем «иголку» к началу «стога» (i **= M – 1**).
- Сравниваем символы с конца «иголки»: **h[k]** и **n[j]**, уменьшая **k** и **j** (изначально **k = i**, **j = M – 1**).
- Если **j** дошло до начала «иголки», подстройка найдена!
- Если на каком-то шаге **h[k] ≠ n[j]**, то сдвигаем «иголку» на **d[h[i]]** вправо (**i += d[h[i]]**).

Массив сдвигов

d – хитрый массив, индексируемый символами x из конечного алфавита. Для каждого символа:

- если **x** отсутствует в **n**, то **d[x]** равно **M**;
- если **x** – не последний в **n**, то **d[x]** равно расстоянию от последнего вхождения **x** в **n** до конца **n**;
- если **x** – последний в **n**, то **d[x]** равно расстоянию от предпоследнего вхождения x в **n** до конца **n**.

n = “abcbc”

d[‘b’] = 1

d[‘c’] = 2

d[‘a’] = 4

d[**…**] = 5

Анализ алгоритма Боуэра-Мура

- На практике работает очень хорошо, вплоть до: **O(N/M).**
- В худшем случае (поиск “abbbb” в “bbbbbbbbbb”): **O((N-M)*M).**

Алгоритм Рабина-Карпа

Вместо сравнения подстрок будем использовать сравнение их хешей.

- Хеш-функция быстро преобразует произвольную строку в некоторое численное значение, причем равные строки преобразуются в равные значения.

Основание системы счисления R

Некое большое простое число Q

- Например:

  hash(“abc”) =

Суть алгоритма

- - хеш «иголки».
- - хеш **M** символов «стога», начиная с позиции **k**.
- Перебираем **k = 0…(N-M)** и сравниваем
- Если совпали, то проверяем посимвольно, и в случае совпадения – успех.
- Иначе идем дальше.

Хитрость вычисления хешей в «стоге сена»

-
-
- Если мы знаем , то за константное время можно вычислить
-

Анализ алгоритма Рабина-Карпа

- В среднем алгоритм работает за **O(N-M)**.
- При достаточно большом простом **Q**, вероятность ложных совпадений (**1/Q**).
- В худшем случае **O((N-M)*M)**.
- Отлично подходит для поиска нескольких «иголок» в одном «стоге».

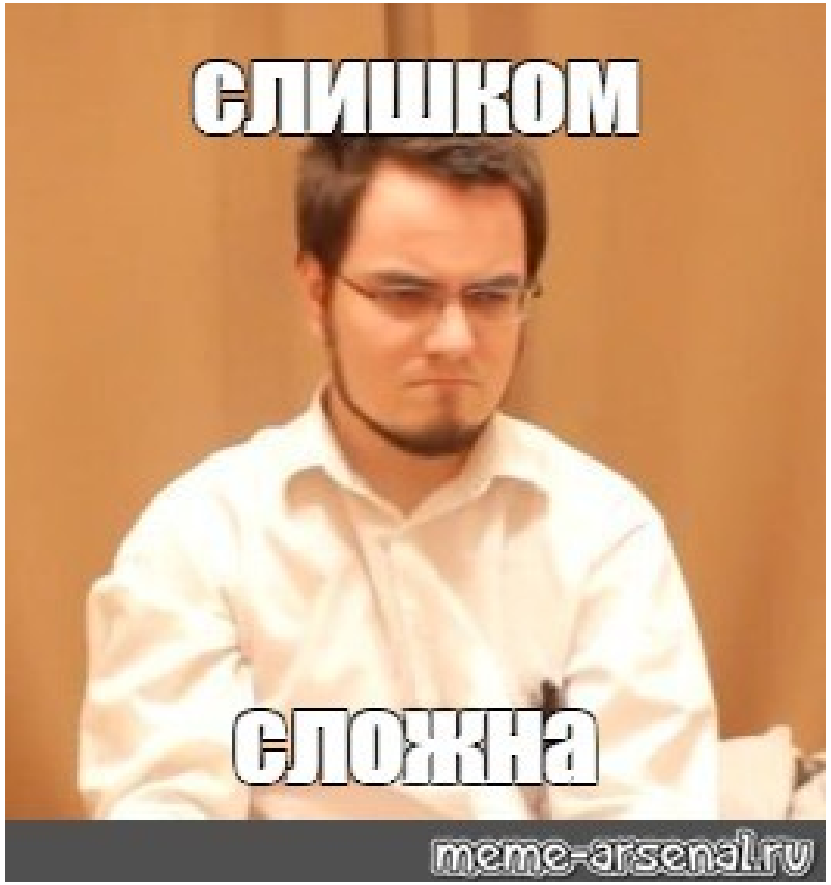
Алгоритм Кнута-Морриса-Пратта

- В худшем случае работает за **O(M+N)**.
- Предварительно строит по «иголке» двумерный массив сдвигов.
- Во время прохода по «стогу», никогда не идет назад.
- Оптимальный алгоритм, но непростой.

Профессия: программист

- Пожарные борются с огнем.
- Педагоги борются с детьми.
- Полиция борется с преступностью.
- **Программисты борются …**

… со сложностью!



Что сложнее?

Работа с черными ящиками

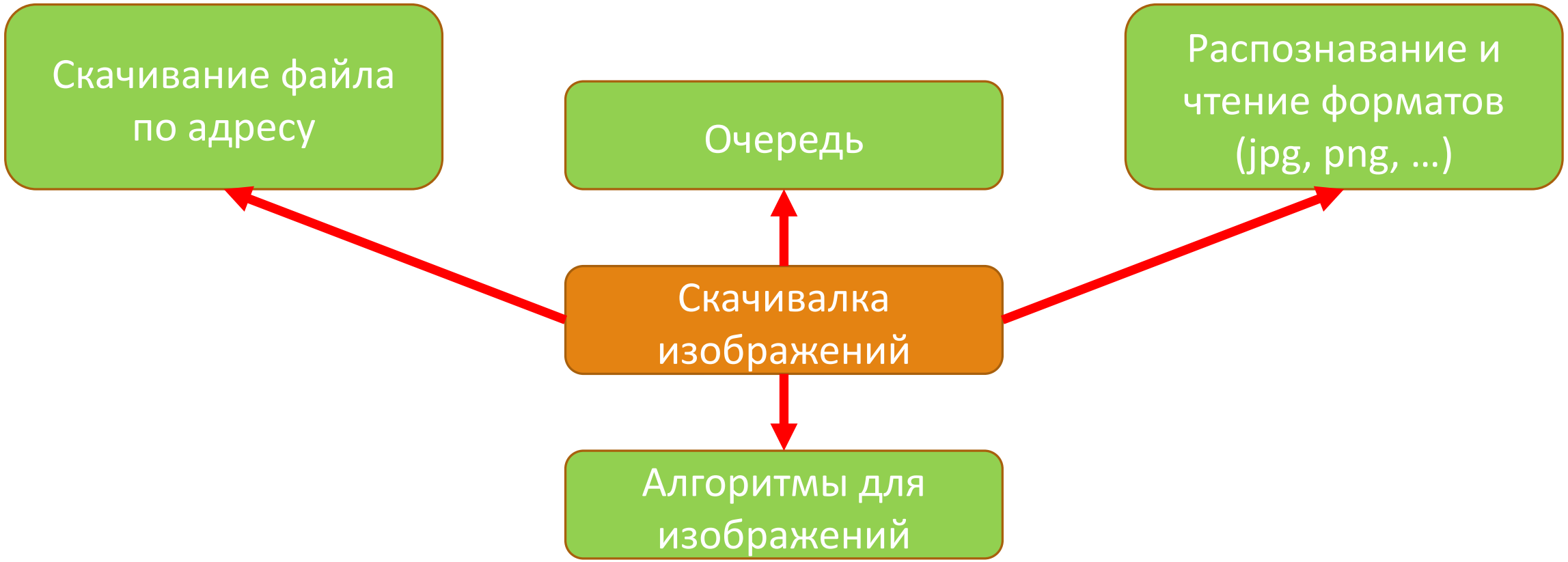
```
import blackbox


blackbox.prepare()
blackbox.work_hard()
```

Если нет черного ящика?

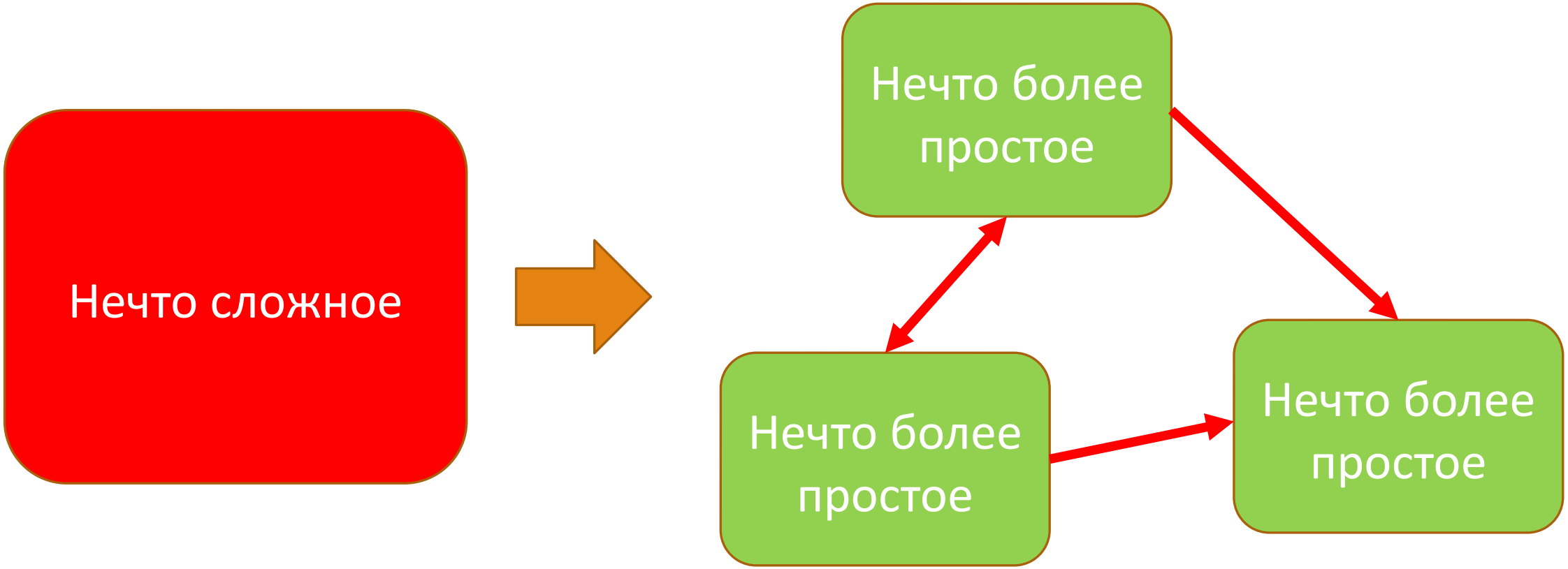
Сделать его с помощью других ящиков!

Скачивалка изображений

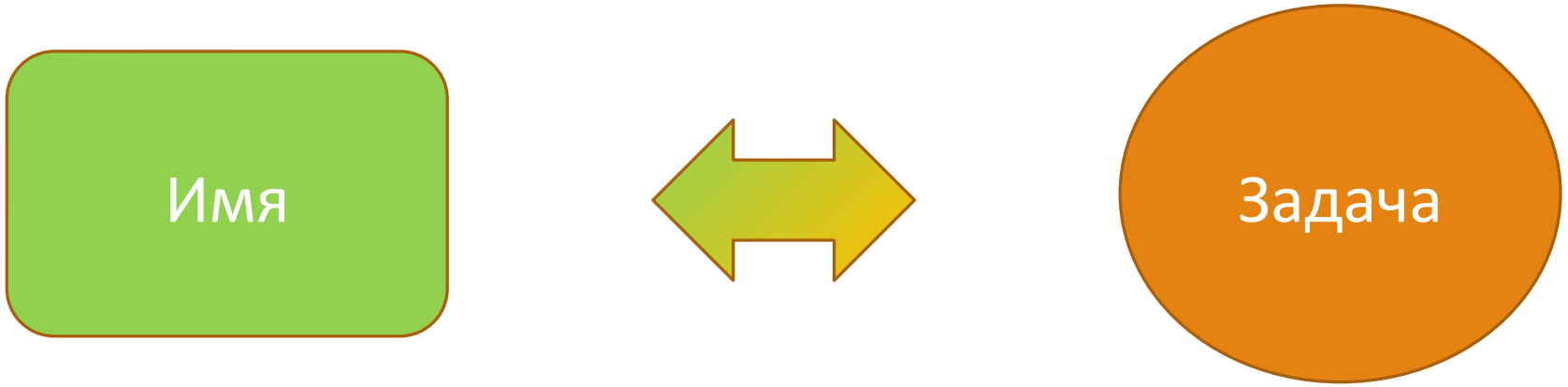

Скачивание файла по адресу
Очередь
Распознавание и чтение форматов (jpg, png, …)
Скачивалка изображений
Алгоритмы для изображений


Низкоуровневые функции работы с сетью
Динамическая память
Динамический массив
Скачивание файла по адресу
Очередь
Распознавание и чтение форматов (jpg, png, …)
Скачивалка изображений
Алгоритмы для изображений

Декомпозиция



Первое представление модуля



Простейшие способы декомпозиции

- Взаимодействие с пользователем (**U**ser **I**nterface) – отдельный модуль.
- Обработка разных структур данных – в разных модулях (**list**, **stack**, **queue**, …).
- Элементы функциональности, близкие по смыслу – в один модуль (**io**, **parse**, **errors**).
- Если процесс естественным образом разбивается на отдельные шаги, то каждый шаг – в свой модуль (**prepare**, **get_data**, **process**, **output**).

На сегодня все