Курс “Алгоритмы на python”

Занятие #5
Рекурсия

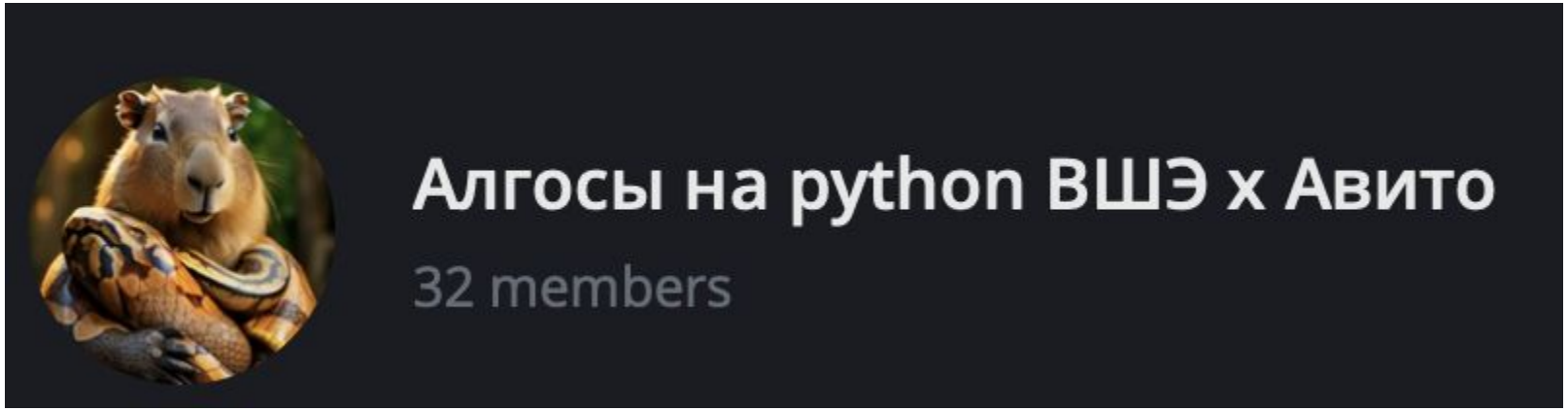
Сентябрь 2025

Avito ВШЭ ФКН

Единая точка входа/выхода – степик

- https://stepik.org/course/251189/

Вопросы и обсуждения – чат



Посещаемость

Орг моменты

Дедлайны

1 дз
жесткий дедлайн сегодня 8 октября

2 дз
* без штрафов — 3 октября включительно
* минус балл — 17 октября включительно

3 дз:
* без штрафов 12 октября включительно
* минус балл — 26 октября включительно

4 дз
* без штрафов 20 октября включительно
* минус балл — 3 ноября включительно

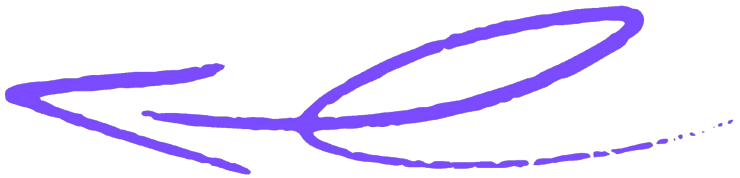
Структура курса «Алгоритмы на питоне»

1 модуль

- Введение в алгоритмы
- Базовые структуры данных
- Хеш-таблицы
- Бинарные деревья поиска
- Рекурсия
- Кучи
- Сортировки

2 модуль

- Графы
- Динамическое программирование
- Алгоритмы в строках
- Алгоритмы в ML и LLM
- Итоговый контест



План занятия

- Часть I. Фреймы и python
- Часть II. Рекурсия и простые случаи
- Часть III. Рекурсия и деревья
- Часть IV. Обратный путь рекурсии

Рекурсия

Текущий контекст выполнения, фреймы и python

3.2.13.2. Frame objects

Frame objects represent execution frames. They may occur in traceback objects, and are also passed to registered trace functions.

3.2.13.2.1. Special read-only attributes

`frame.f_back`	Points to the previous stack frame (towards the caller), or `None` if this is the bottom stack frame
`frame.f_code`	The code object being executed in this frame. Accessing this attribute raises an auditing event `object.__getattr__` with arguments `obj` and `"f_code"`.
`frame.f_locals`	The mapping used by the frame to look up local variables. If the frame refers to an optimized scope, this may return a write-through proxy object. *Changed in version 3.13:* Return a proxy for optimized scopes.
`frame.f_globals`	The dictionary used by the frame to look up global variables
`frame.f_builtins`	The dictionary used by the frame to look up built-in (intrinsic) names
`frame.f_lasti`	The "precise instruction" of the frame object (this is an index into the bytecode string of the code object)
`frame.f_generator`	The generator or coroutine object that owns this frame, or `None` if the frame is a normal function. *Added in version 3.14.*

Текущий контекст выполнения, фреймы и python

`inspect` — Inspect live objects

Source code: Lib/inspect.py

The `inspect` module provides several useful functions to help get information about live objects such as modules, classes, methods, functions, tracebacks, frame objects, and code objects. For example, it can help you examine the contents of a class, retrieve the source code of a method, extract and format the argument list for a function, or get all the information you need to display a detailed traceback.

Текущий контекст выполнения, фреймы и python

```
>>> import inspect
>>>
>>> def a():
...     b()
...
>>> def b():
...     c()
...
>>> def c():
...     for frame in inspect.stack():
...         print(frame.function)
...
>>> a()
c
b
a
<module>
```

Текущий контекст выполнения, фреймы и python

```
>>> import inspect
>>>
>>> def f():
...     frame = inspect.currentframe()
...     print("Текущая функция:", frame.f_code.co_name)
...     print("Вызывающая функция:", frame.f_back.f_code.co_name)
...
>>> def g():
...     f()
...
>>> g()
Текущая функция: f
Вызывающая функция: g
```

Текущий контекст выполнения, фреймы и python

```
>>> def f(a):
...     frame = inspect.currentframe()
...     print("Параметры текущей функции:", frame.f_locals)
...     print("Параметры вызывающей функции:", frame.f_back.f_locals)
...
>>> def g(b):
...     f(b)
...
>>> g(10)
Параметры текущей функции: {'a': 10, 'frame': <frame at 0x10216b580, file '<stdin>', line 3, code f>}
Параметры вызывающей функции: {'b': 10}
```

Текущий контекст выполнения, фреймы и python

```
>>> def c(z):
...     stack = inspect.stack()
...     for frame_info in stack:
...         print(f"Функция: {frame_info.function}")
...         print(f"  Аргументы и значения: {frame_info.frame.f_locals}")
...         print("-" * 40)
...
>>> def b(y):
...     c(y + 1)
...
>>> def a(x):
...     b(x * 2)
...
>>> a(5)
Функция: c
  Аргументы и значения: {'z': 11, 'stack': [FrameInfo(frame=<frame at 0x10212f780, file '<stdin>', line 5, code c>, filename='<stdin>', lineno=
2, function='c', code_context=None, index=None, positions=Positions(lineno=2, end_lineno=2, col_offset=12, end_col_offset=27)), FrameInfo(frame
=<frame at 0x1023e96f0, file '<stdin>', line 2, code b>, filename='<stdin>', lineno=2, function='b', code_context=None, index=None, positions=P
ositions(lineno=2, end_lineno=2, col_offset=4, end_col_offset=12)), FrameInfo(frame=<frame at 0x1023e97a0, file '<stdin>', line 2, code a>, fil
ename='<stdin>', lineno=2, function='a', code_context=None, index=None, positions=Positions(lineno=2, end_lineno=2, col_offset=4, end_col_offse
t=12)), FrameInfo(frame=<frame at 0x102499bc0, file '<stdin>', line 1, code <module>>, filename='<stdin>', lineno=1, function='<module>', code_
context=None, index=None, positions=Positions(lineno=1, end_lineno=1, col_offset=0, end_col_offset=4))], 'frame_info': FrameInfo(frame=<frame a
t 0x10212f780, file '<stdin>', line 5, code c>, filename='<stdin>', lineno=2, function='c', code_context=None, index=None, positions=Positions(
lineno=2, end_lineno=2, col_offset=12, end_col_offset=27))}
----------------------------------------
Функция: b
  Аргументы и значения: {'y': 10}
----------------------------------------
Функция: a
  Аргументы и значения: {'x': 5}
----------------------------------------
Функция: <module>
  Аргументы и значения: {'__name__': '__main__', '__doc__': None, '__package__': None, '__loader__': <class '_frozen_importlib.BuiltinImporter'
>, '__spec__': None, '__annotations__': {}, '__builtins__': <module 'builtins' (built-in)>, 'inspect': <module 'inspect' from '/Users/vsmunkina
/.pyenv/versions/3.12.8/lib/python3.12/inspect.py'>, 'f1': <function f1 at 0x102498860>, 'f2': <function f2 at 0x1024989a0>, 'f3': <function f3
 at 0x1024987c0>, 'f': <function f at 0x102499580>, 'g': <function g at 0x102499760>, 'c': <function c at 0x102499940>, 'b': <function b at 0x1
02499800>, 'a': <function a at 0x102499a80>}
----------------------------------------
```

Глубина рекурсии

```
>>> print(sys.getrecursionlimit())
1000
```

Факториал. Рисуем на доске.

Фибоначчи. Рисуем на доске.

Backtracking. Рисуем на доске.

BST и рекурсия. Продолжаем рисовать

BST и рекурсия. Продолжаем рисовать. Поиск, вставка.

Обратный путь рекурсии. Доска

Балансировка и обратный путь рекурсии. Пример AVL

Красно-черное дерево

Properties [edit]

In addition to the requirements imposed on a binary search tree the following must be satisfied by a red–black tree:[17]

1. Every node is either red or black.
2. All null nodes are considered black.
3. A red node does not have a red child.
4. Every path from a given node to any of its leaf nodes (that is, to any descendant null node) goes through the same number of black nodes.
5. (Conclusion) If a node **N** has exactly one child, the child must be red. If the child were black, its leaves would sit at a different black depth than **N**'s null node (which is considered black by rule 2), violating requirement 4.

Красно-черное по Кормену

В книге Кормена "Алгоритмы: построение и анализ" дается немного иное определение красно-черного дерева, а именно:

Двоичное дерево поиска является красно-чёрным, если обладает следующими свойствами:

1. Каждая вершина — либо красная, либо черная
2. Каждый лист — черный
3. Если вершина красная, оба ее ребенка черные
4. Все пути, идущие от корня к листьям, содержат одинаковое количество черных вершин

То, что только черная вершина может иметь красных детей, совместно с 4-ым свойством говорит о том, что корень дерева должен быть черным, а значит определения можно считать эквивалентными.