- Что такое рекурсия?
- Правильная рекурсия
- Что такое стек вызовов?
- Что такое стек вызовов?
- Почему рекурсия это плохо
- Recursion depth
- Глубина рекурсии
- Почему рекурсия это хорошо
- Вариант задачи для рекурсии
- Хвостовая рекурсия
- Оптимизация хвостовой рекурсии и почему её нет в Python
- Пример когда рекурсия помогает
- Дополнительная литература
- Динамическое программирование
- Кэширование
- Поиск приблизительно совпадающих строк
- Рекурсивное решение
- Динамическое программирование в действии
- Вопросы-ответы

Что такое рекурсия?

Приём в программировании, когда задача может быть разделена на несколько таких же, но проще, задач.

```
def pow(x, n):
    # возведение числа в степень это
    # умножение числа на число
    # в степени n-1
    if n == 0:
        return 1
    return x * pow(x, n-1)
```

Правильная рекурсия

```
def pow(x, n):
    # хорошо бы проверить,
    # что база достижима
    assert n >= 0
    # base case / база рекурсии
    if n == 0:
        return 1
    # recursive case / шаг рекурсии
    return x * pow(x, n-1)
```

Что такое стек вызовов?

```
def foo(msg):
    print('{} foo'.format(msg))

def main():
    msg = 'hello'
    foo(msg)

if __name__ == '__main__':
    main()
```

Что такое стек вызовов?

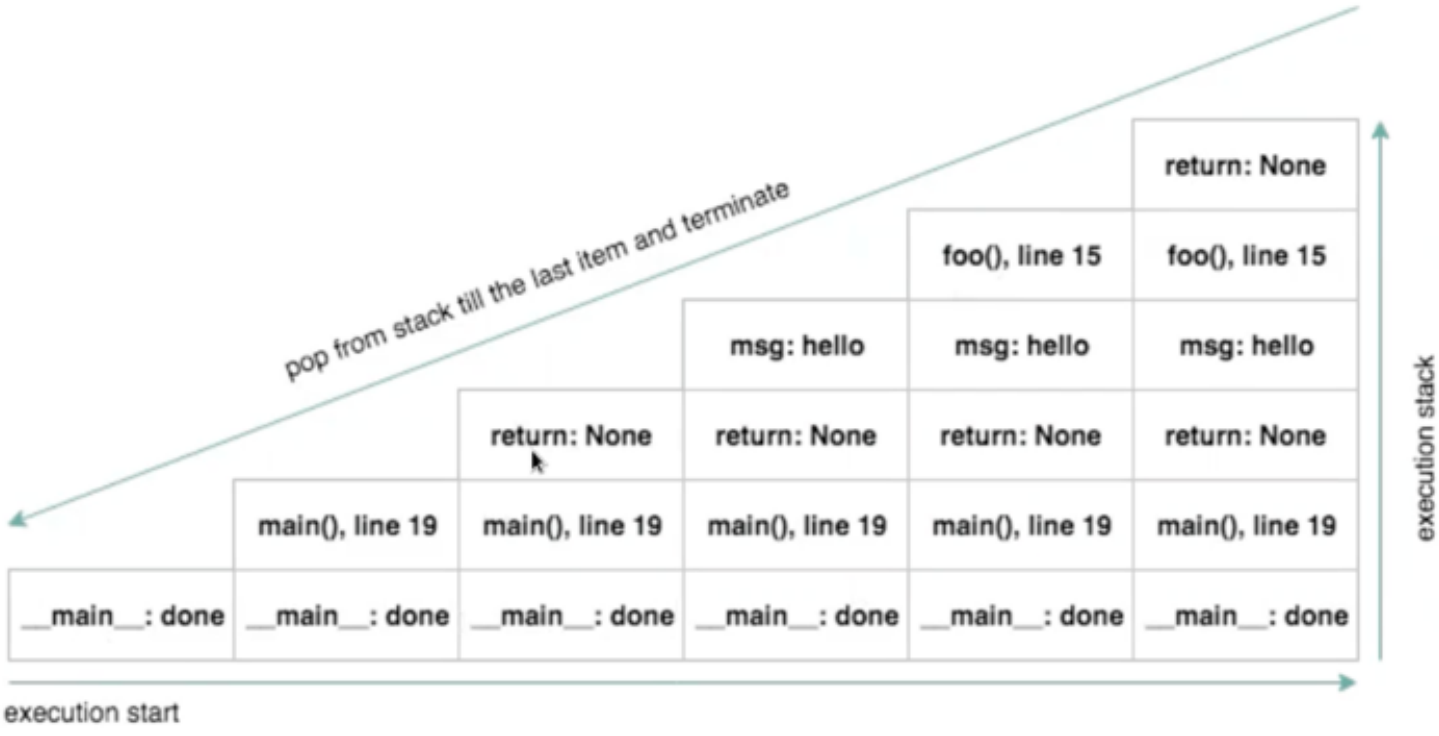


Figure 1: img

Почему рекурсия это плохо

- стек вызовов растёт вместе с ростом глубины рекурсии
- можно попасть в бесконечную рекурсию и истратить всю память на стек вызовов

Recursion depth

```
def inf_counter(x):
    print(x)
    return inf_counter(x+1)
inf_counter(0)
```

Глубина рекурсии

```
import sys

print(sys.getrecursionlimit())
sys.setrecursionlimit(
    sys.getrecursionlimit() + 234
)
print(sys.getrecursionlimit())
```

```
1000
1234
```

Почему рекурсия это хорошо

Помогает описать решение задачи понятным языком

```
# n! = n * (n-1)
def factorial(n):
    if n == 0:
        return 1
    return n * factorial(n-1)

print(factorial(5))
```

```
120
```

Хвостовая рекурсия

Рекурсия, не требующая действий с возвращённым результатом из шага рекурсии.

```
def factorial(n, collected=1):
    if n == 0:
        return collected
    return factorial(n-1, collected*n)

print(factorial(5))
```

```
120
```

Переписываем на цикл

```
def factorial(n):
    collected = 1
    while n > 0:
        collected *= n
        n -= 1
    return collected
```

Оптимизация хвостовой рекурсии и почему её нет в Python

- Интерпретаторы/компиляторы могут оптимизировать хвостовую рекурсию (Tail Call Optimization) и не делать записей в стек вызовов, а подменять переменные в стеке вызовов, таким образом код получится равнозначным обычному циклу
- Почему TCO нет и не будет в Python

Вариант задачи для рекурсии

Попробуйте реализовать решение этой задачи без использования рекурсии

Пример когда рекурсия помогает

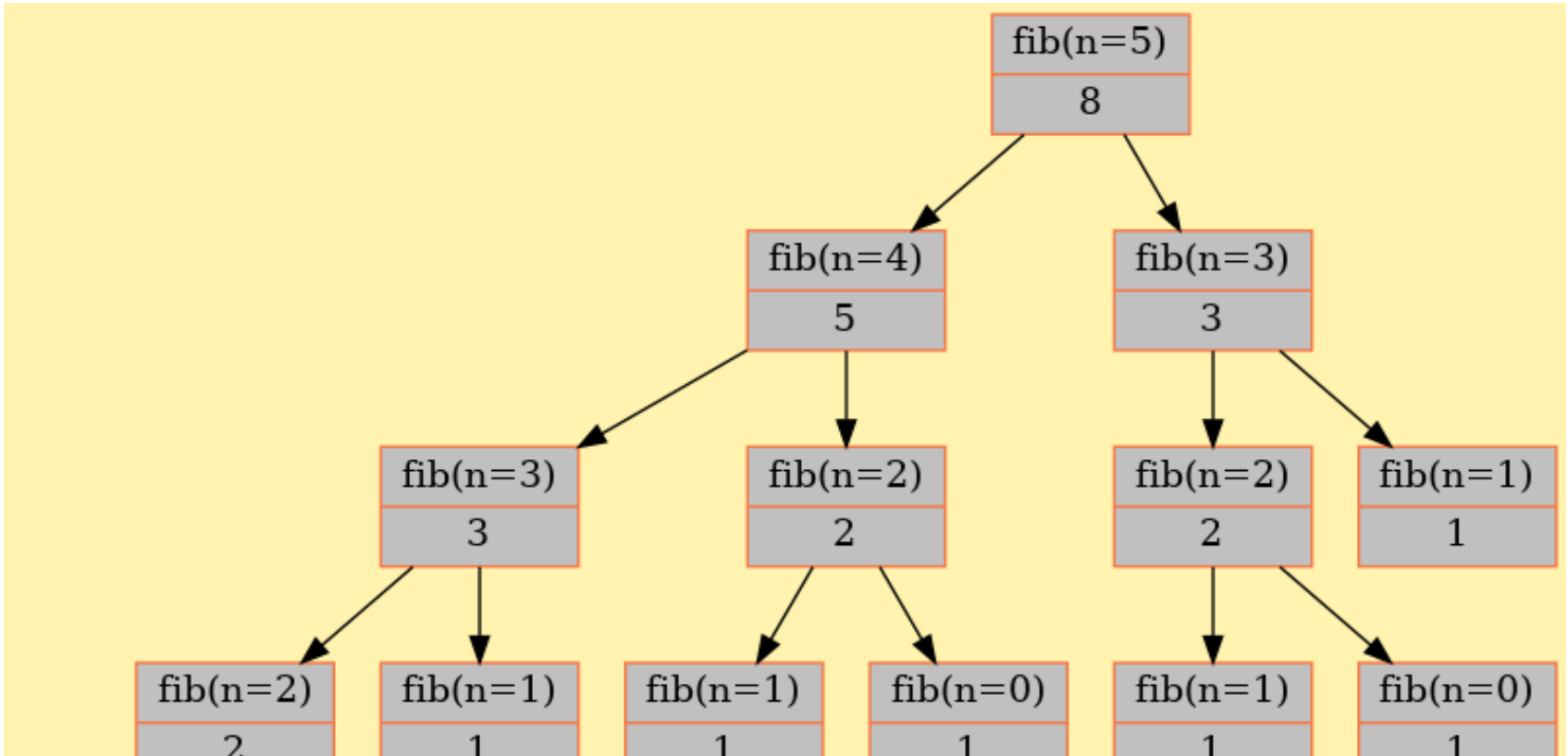
- **Задача:** У вас есть вложенная структура данных и вы хотите просуммировать значения поля X во всех объектах этой структуры.
- **Решение задачи:** https://github.com/Roxe322/recursion_webinar/blob/master/recursion_example.py

Дополнительная литература

- SICP

Динамическое программирование

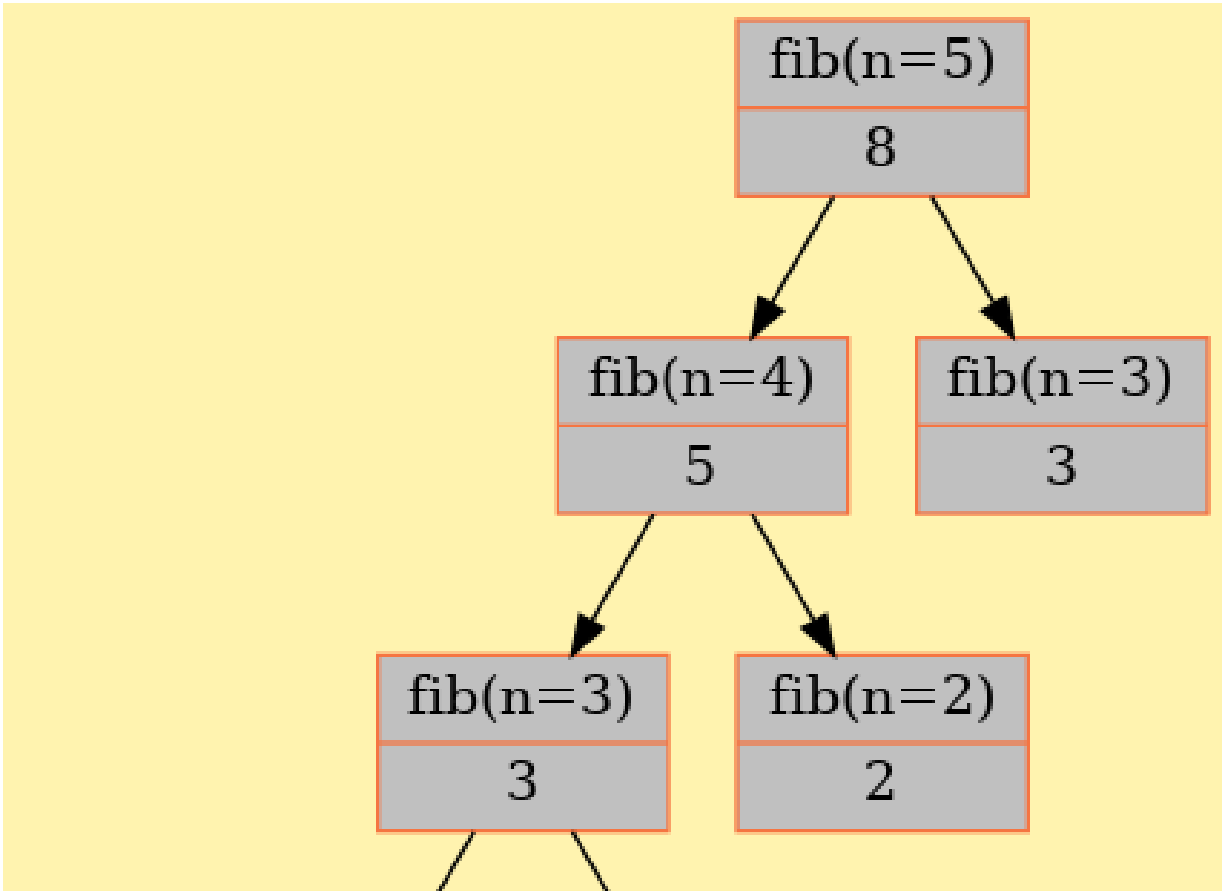
```
def fib(n):
    if n == 0:
        return 1
    if n == 1:
        return 1
    return fib(n=n-1) + fib(n=n-2)
```



Кэширование

```
cache = {0: 1, 1: 1}

def fib(n):
    if n not in cache:
        cache[n] = fib(n=n-1) + fib(n=n-2)
    return cache[n]
```



Поиск приблизительно совпадающих строк

Возможные действия над строками, каждое действие будет иметь стоимость (1)

- ***замена*:** заменить один символ в строку A1 на символ из строки A2. (“мама” → “рама”)
- ***вставка*:** вставить один символ в строку A1 так чтобы она совпала с подстрокой A2. (“роза” → “проза”)
- ***удаление*:** удалить один символ в строке A1 так чтобы она совпала с подстрокой A2. (“гроза” → “роза”)

Рекурсивное решение

```
def lev(a: str, b: str) -> int:
    if not a: return len(b)
    if not b: return len(a)
    return min([
        lev(a[1:],b[1:])+(a[0]!=b[0]),
        lev(a[1:],b)+1,
        lev(a,b[1:])+1
    ])

print(lev("salt", "foobar"))
print(lev("halt", "salt"))
```

- 6
- 1

Динамическое программирование в действии

```
def levenshtein(
    a: str, b: str, m: List[List[int]]
) -> int:
    for i in range(1, len(a)):
        for j in range(1, len(b)):
            m[i][j] = min(
                m[i-1][j-1] + (a[i] != b[j]),
                m[i][j-1] + 1,
                m[i-1][j] + 1
            )
    return m[len(a)-1][len(b)-1]
```

Вопросы-ответы