при поддержке
{а}вижн

Лекция 18

Перевод и адаптация материалов Джона ДеНиро (John DeNero). Используется с разрешения автора.

Измерение эффективности

Рекурсивное вычисление последовательности Фибоначчи

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

fib(5)

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

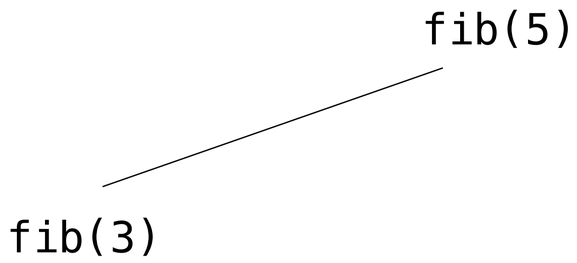
fib(5)

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

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

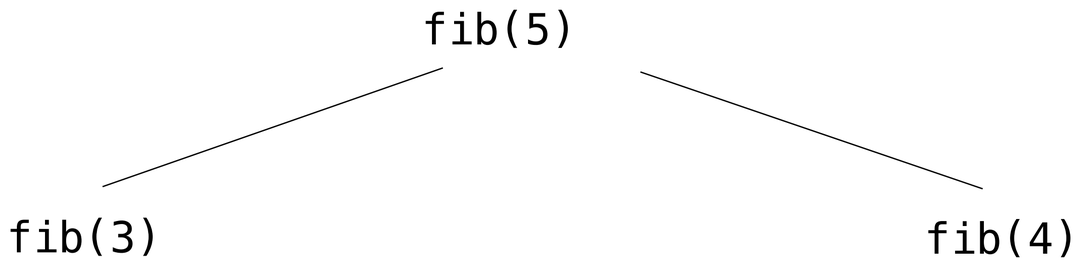



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

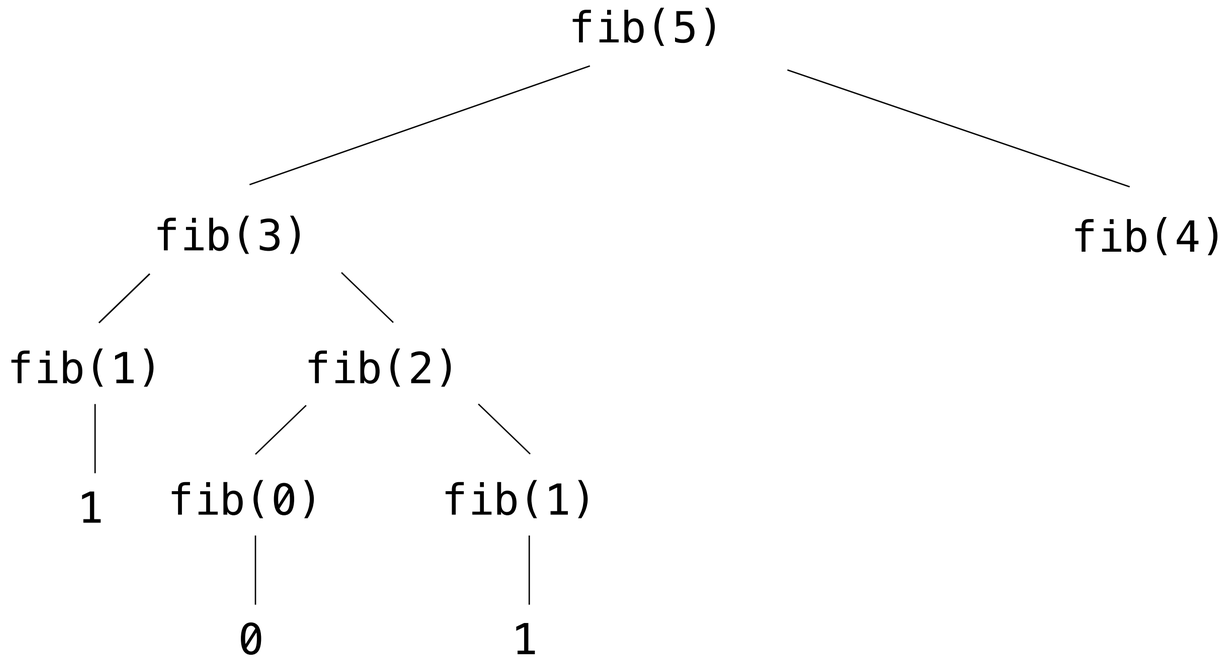



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:



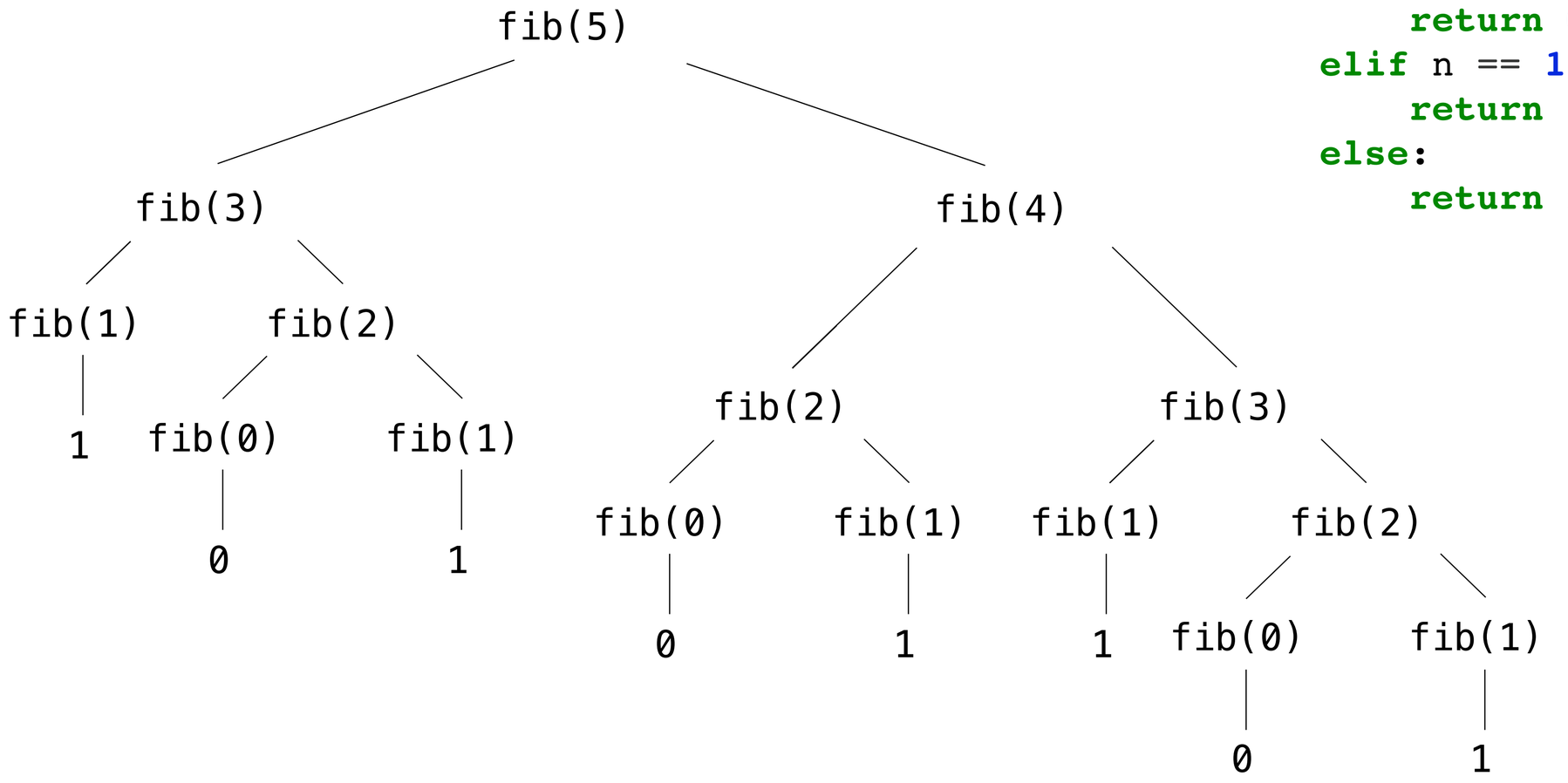

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

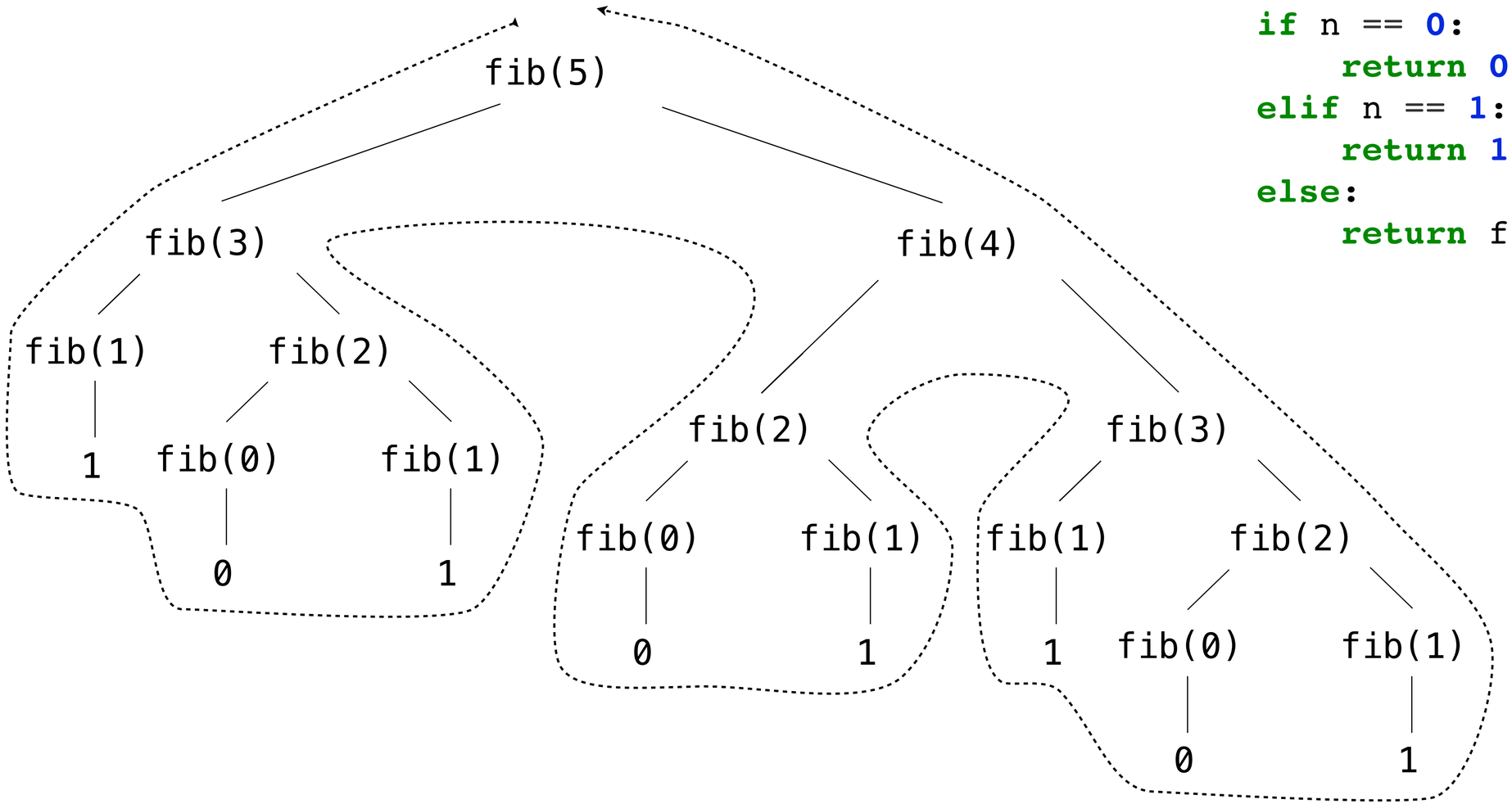
```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```



http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

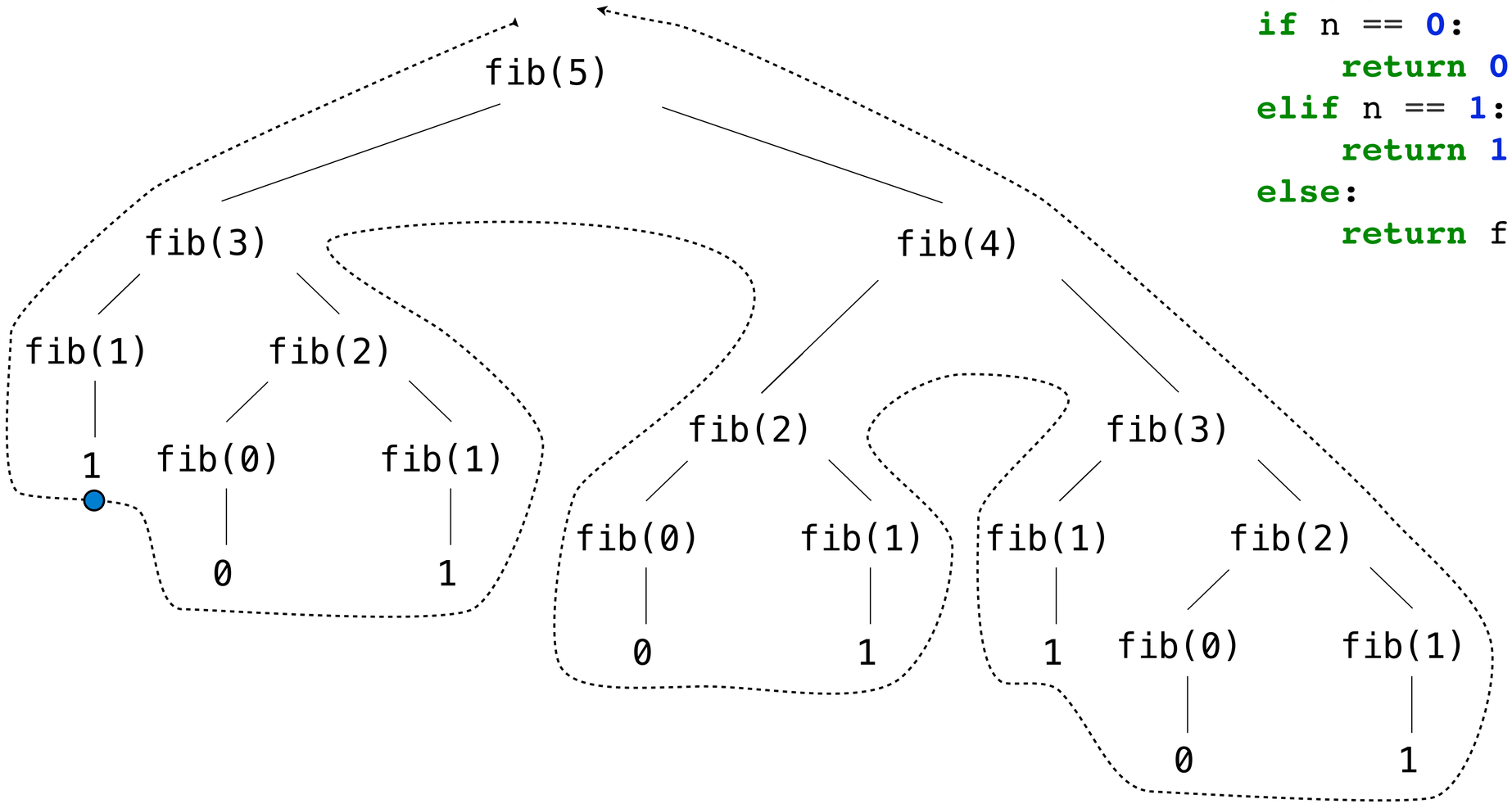



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

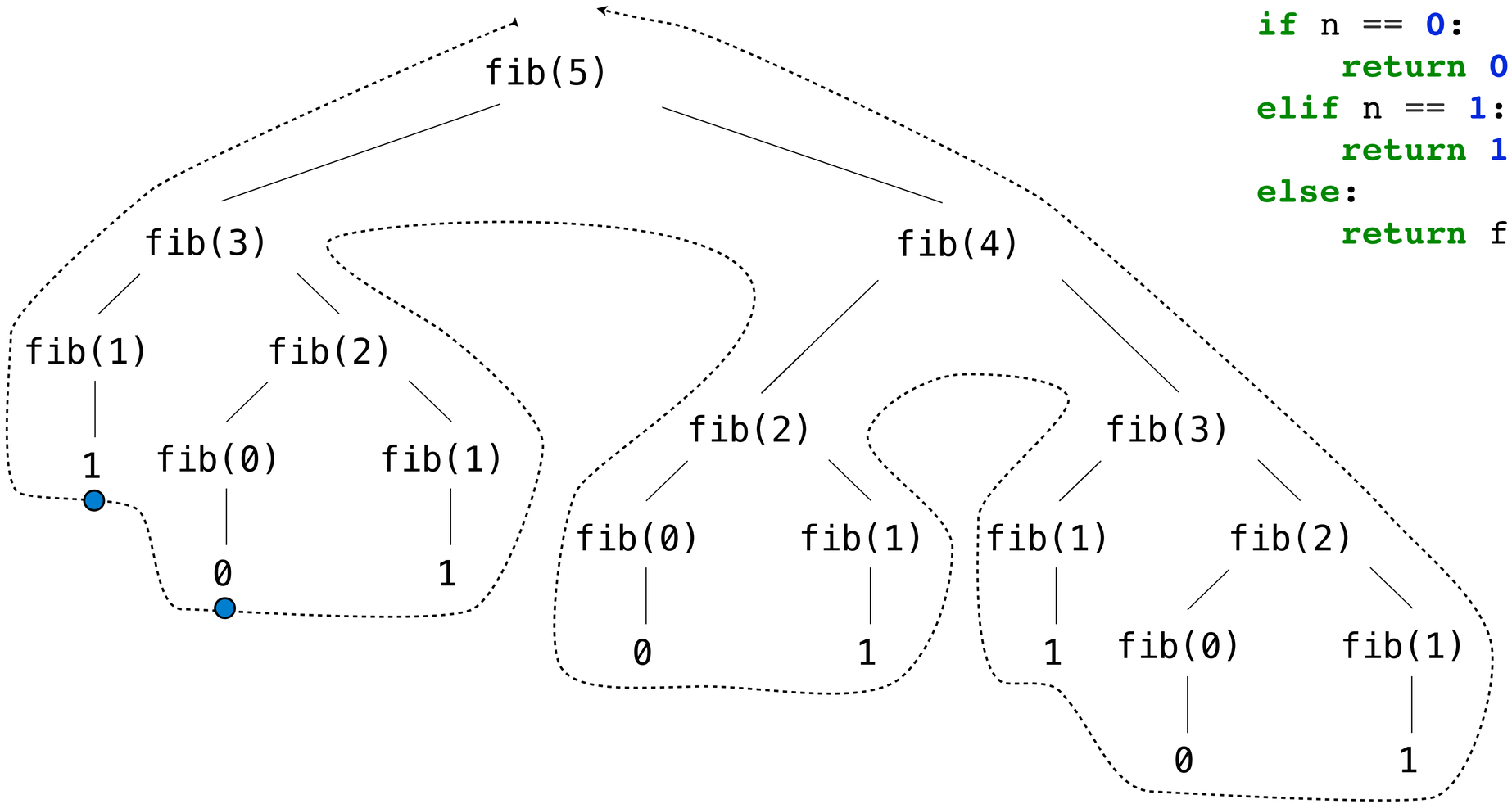



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

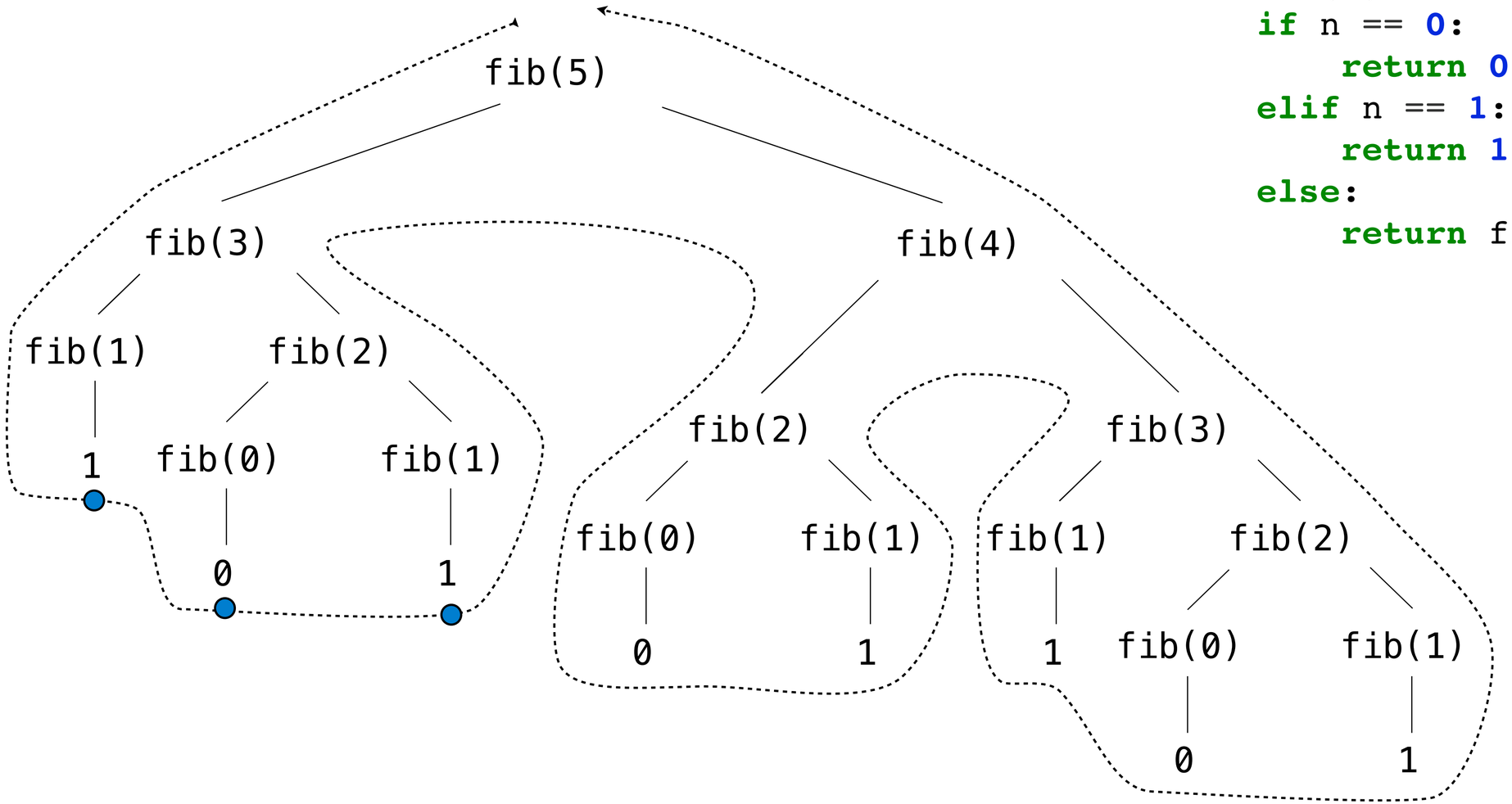



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

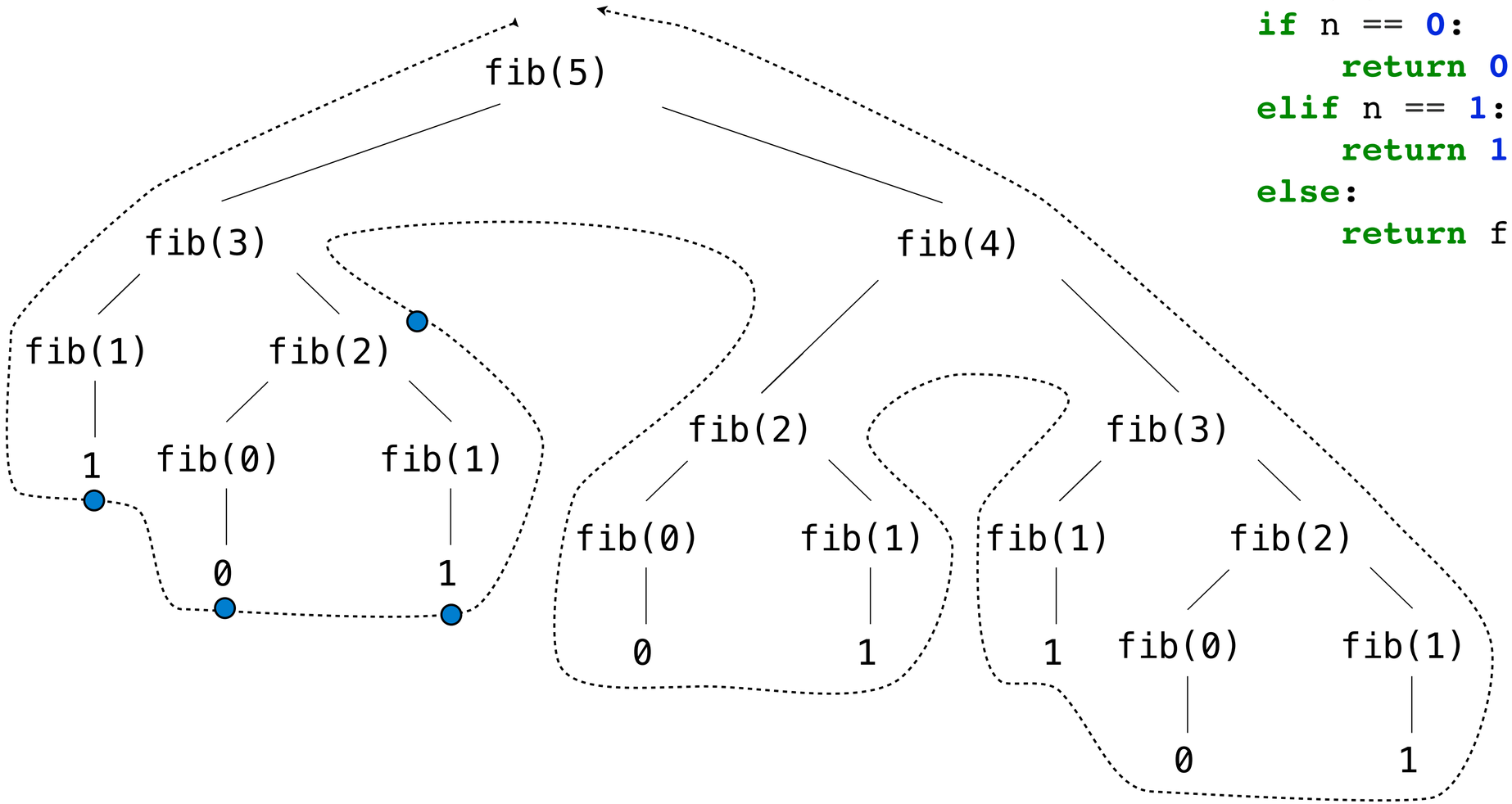



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

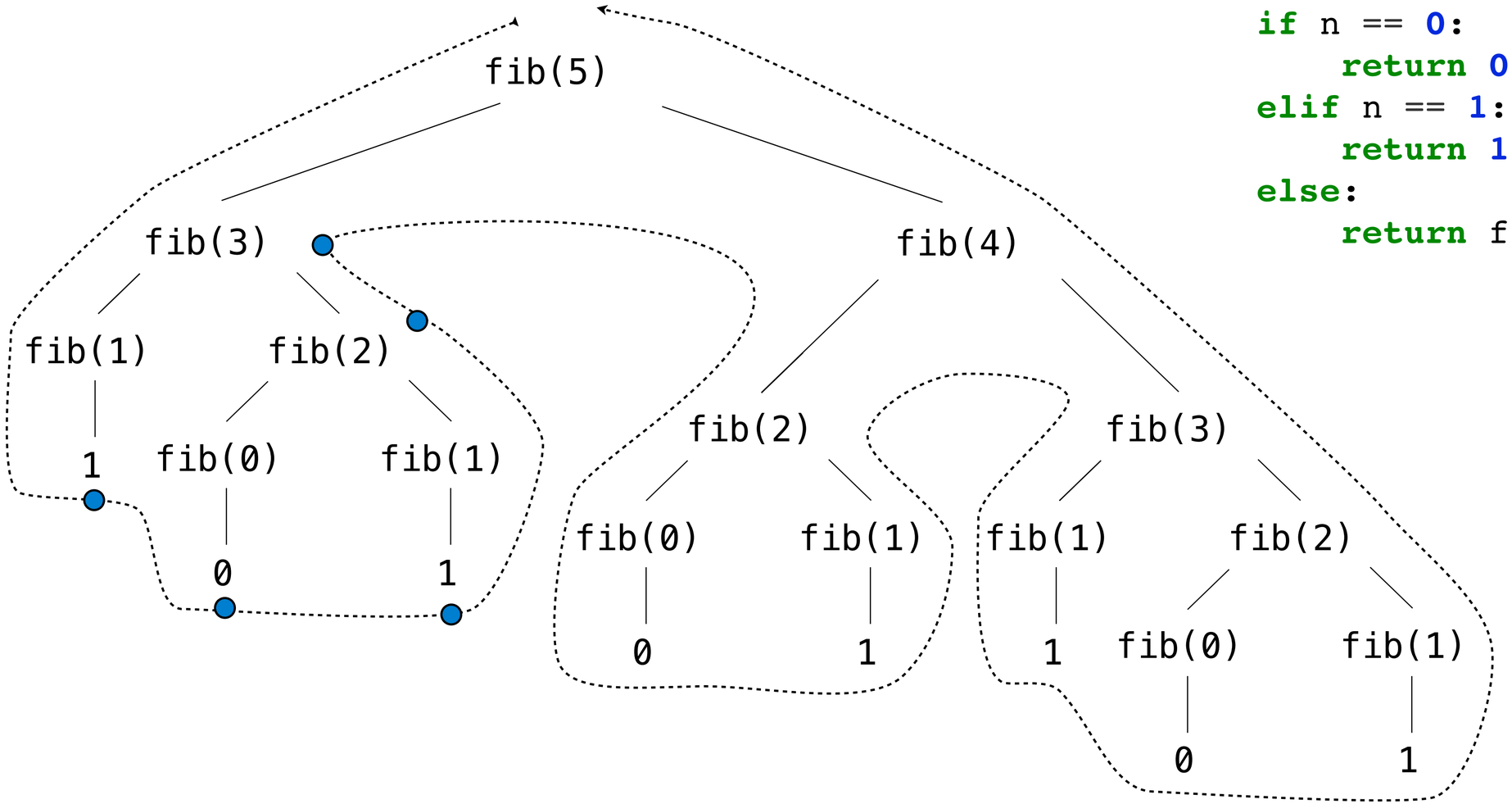



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

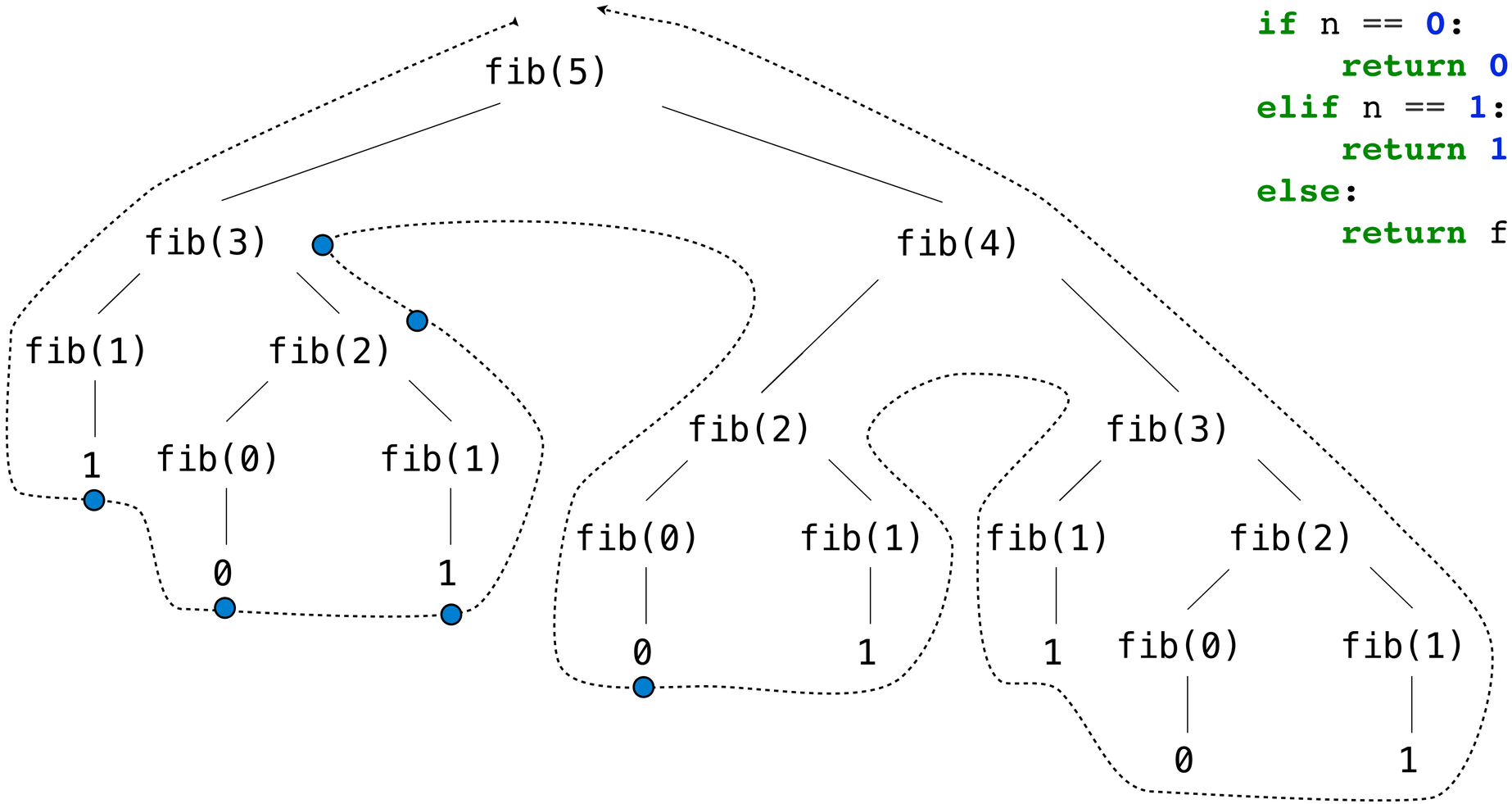



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

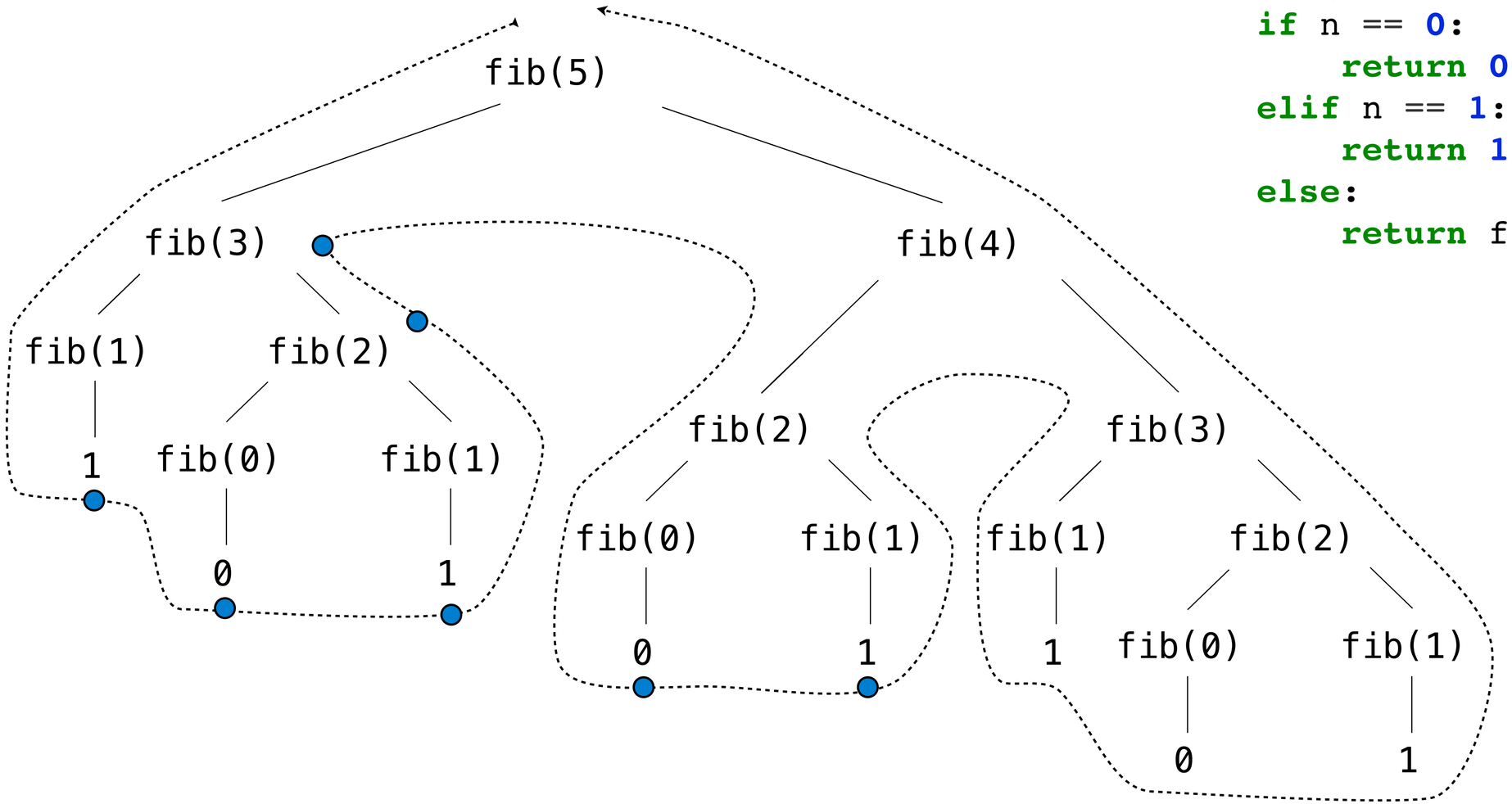



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

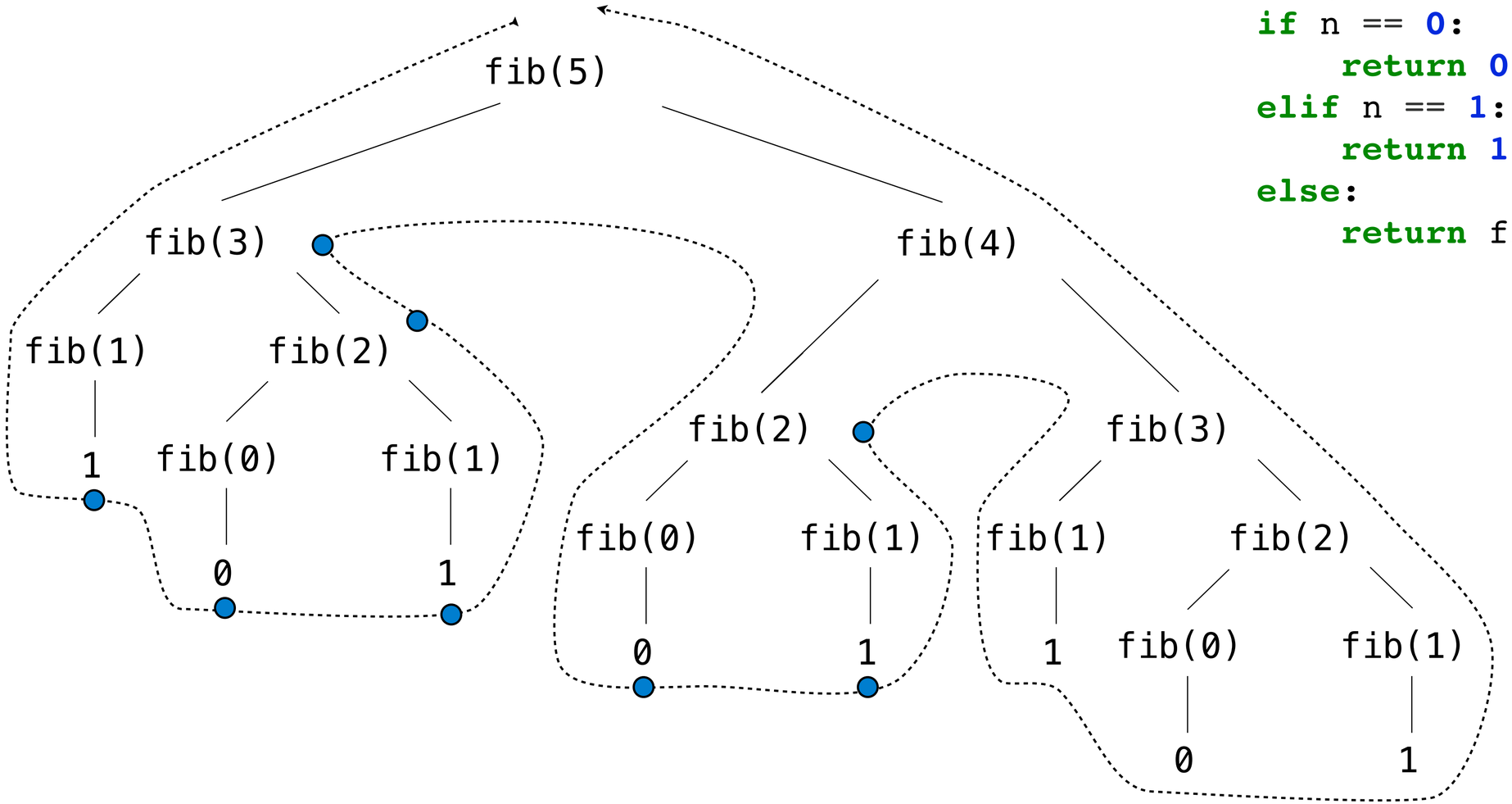



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

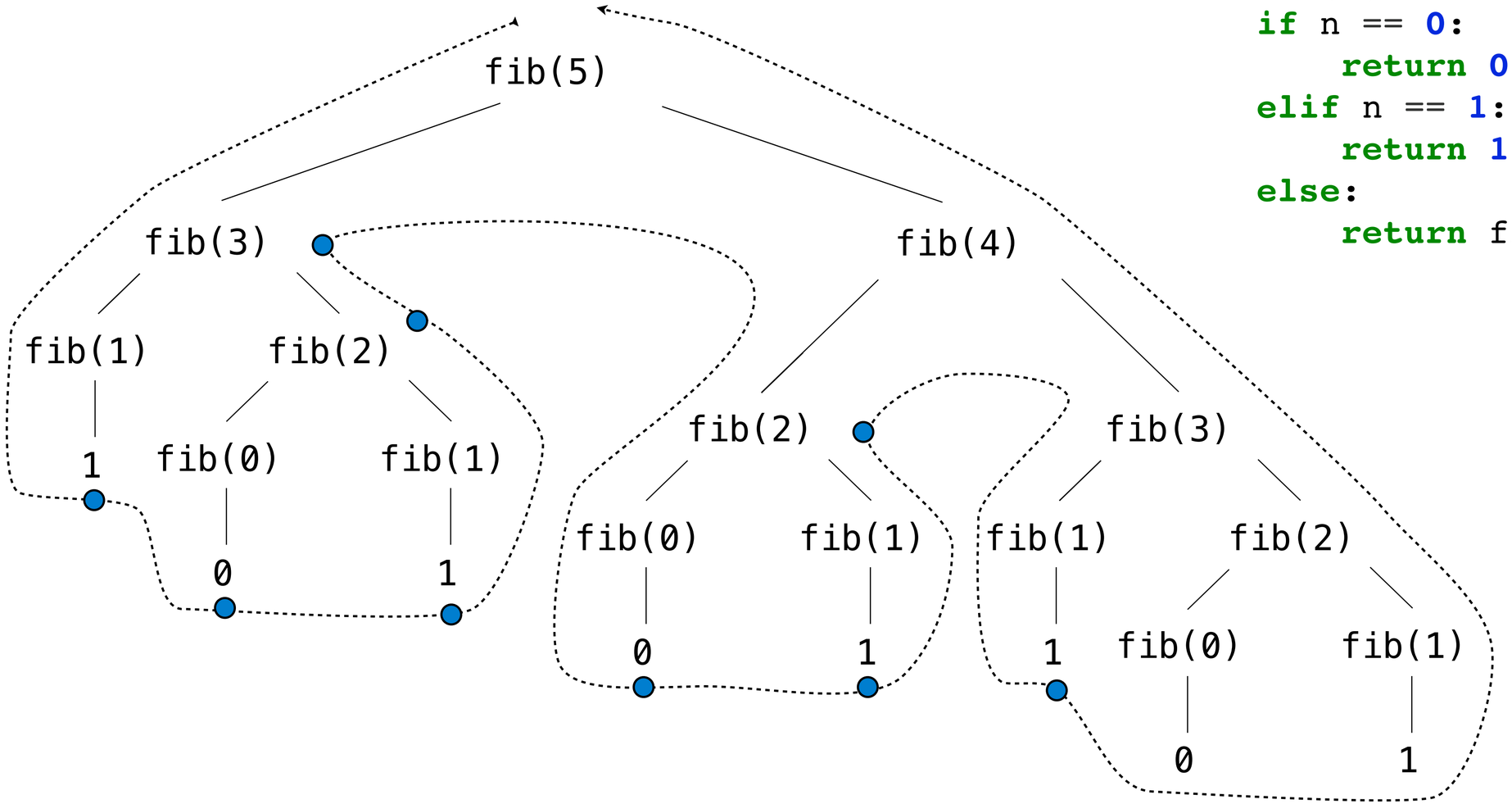



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

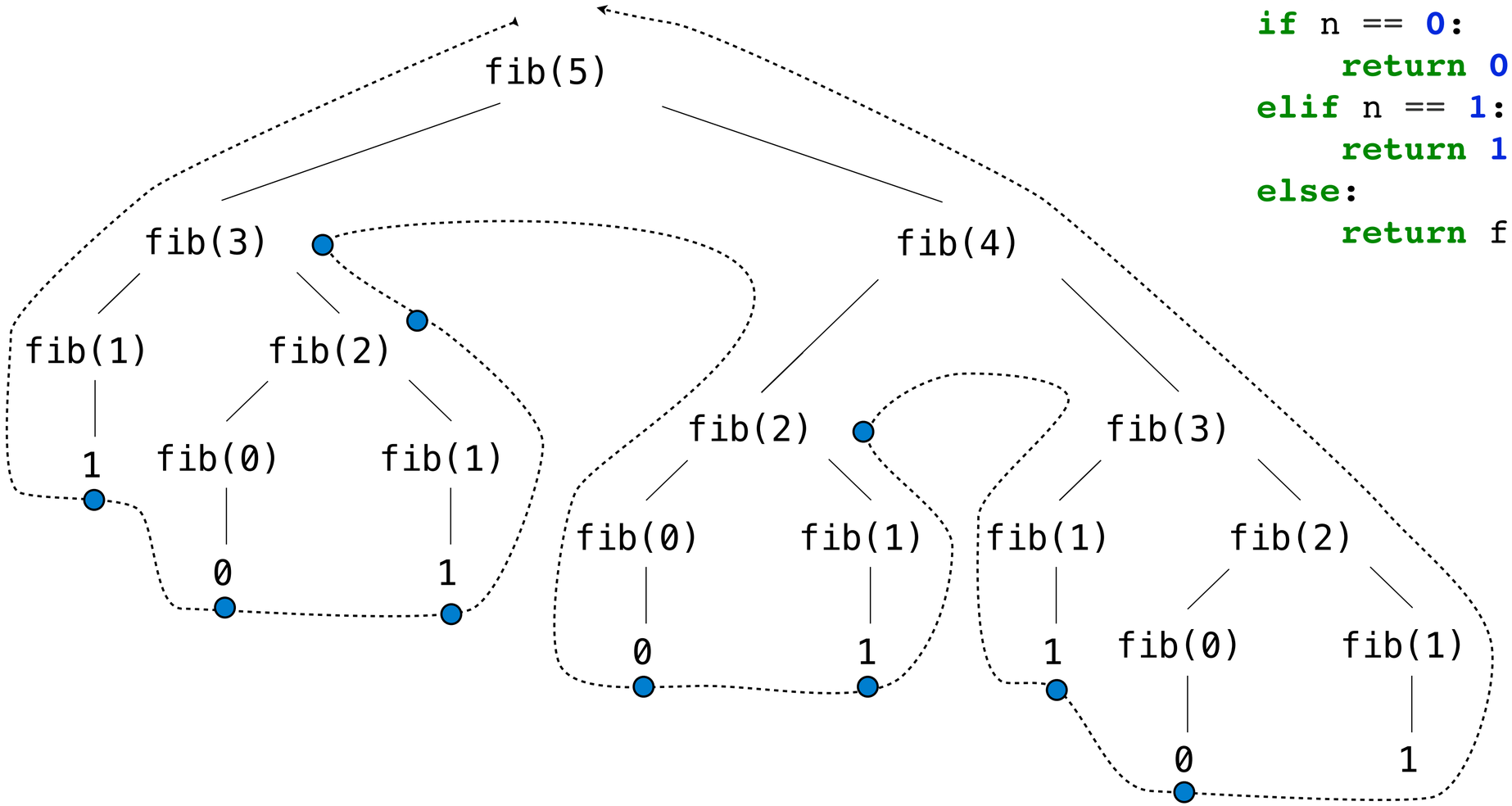



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

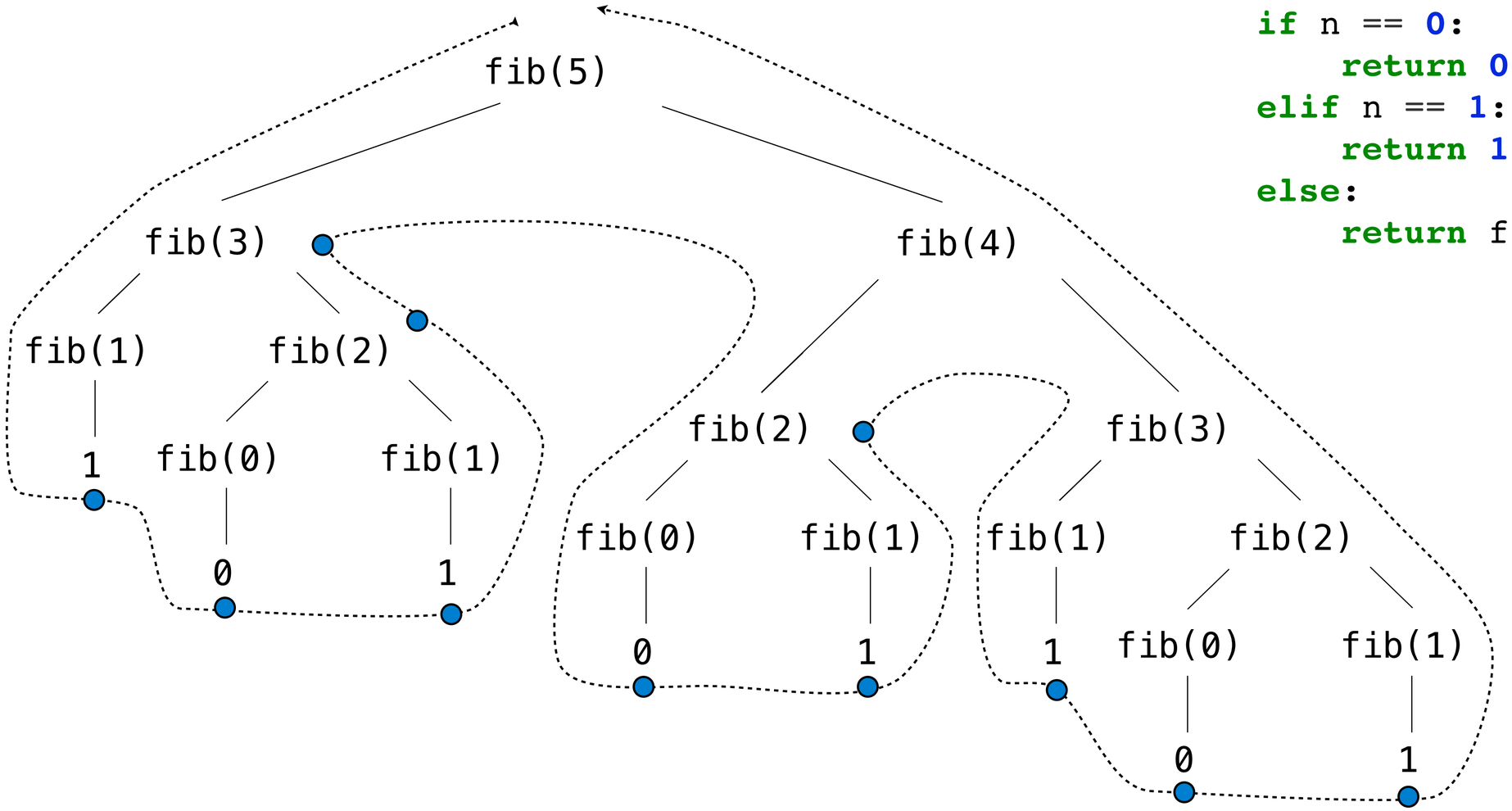



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

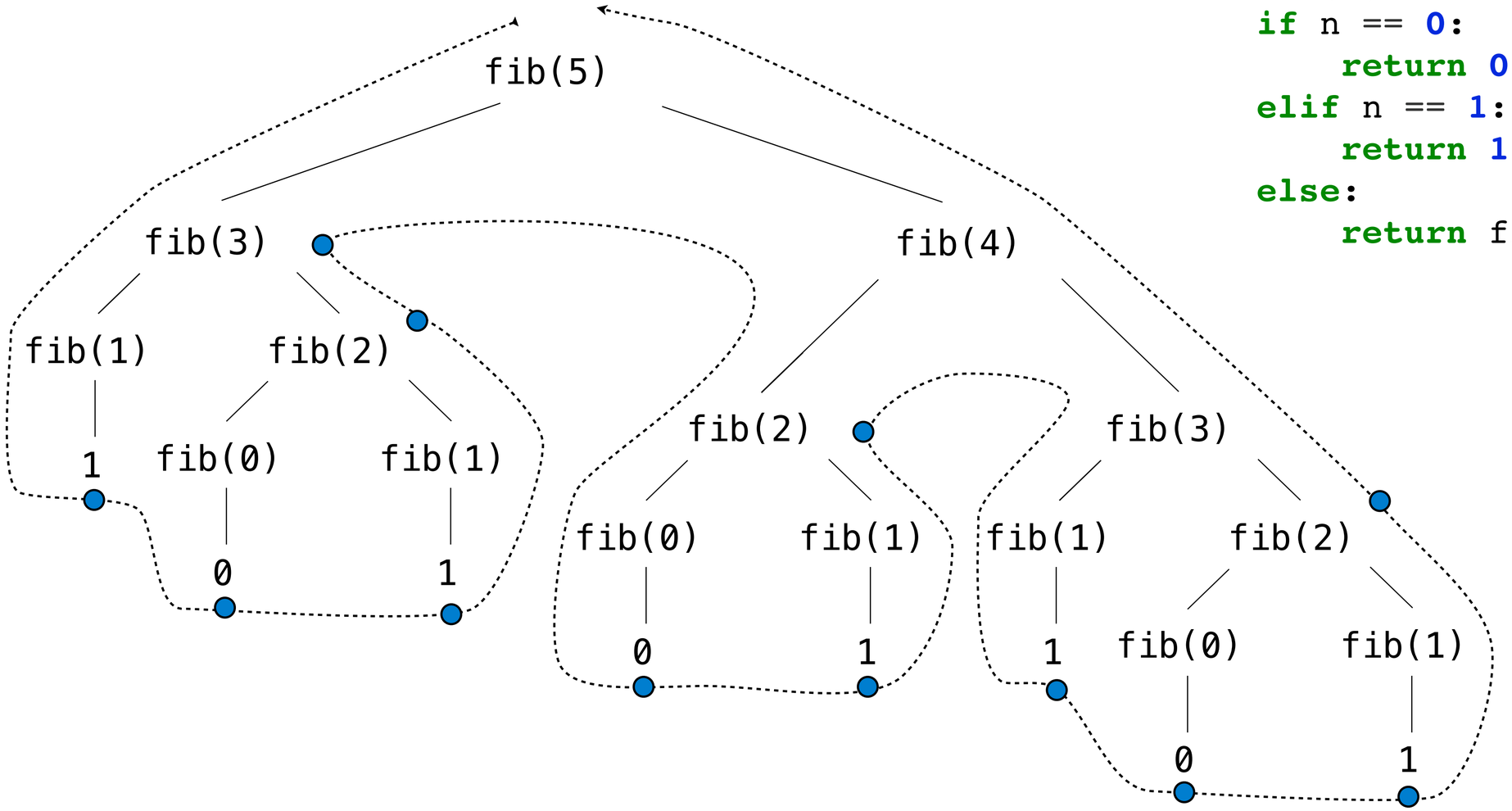



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

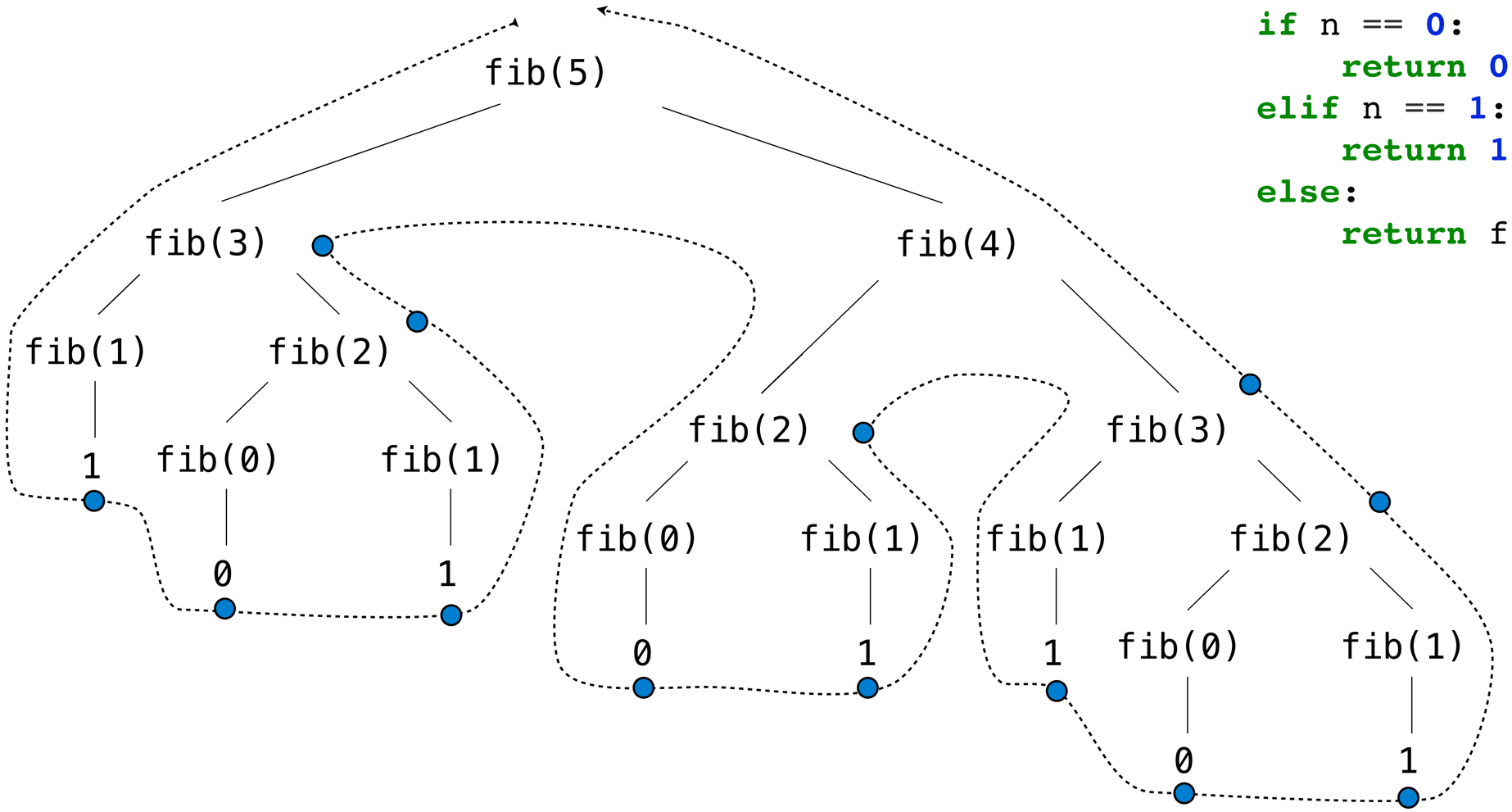



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

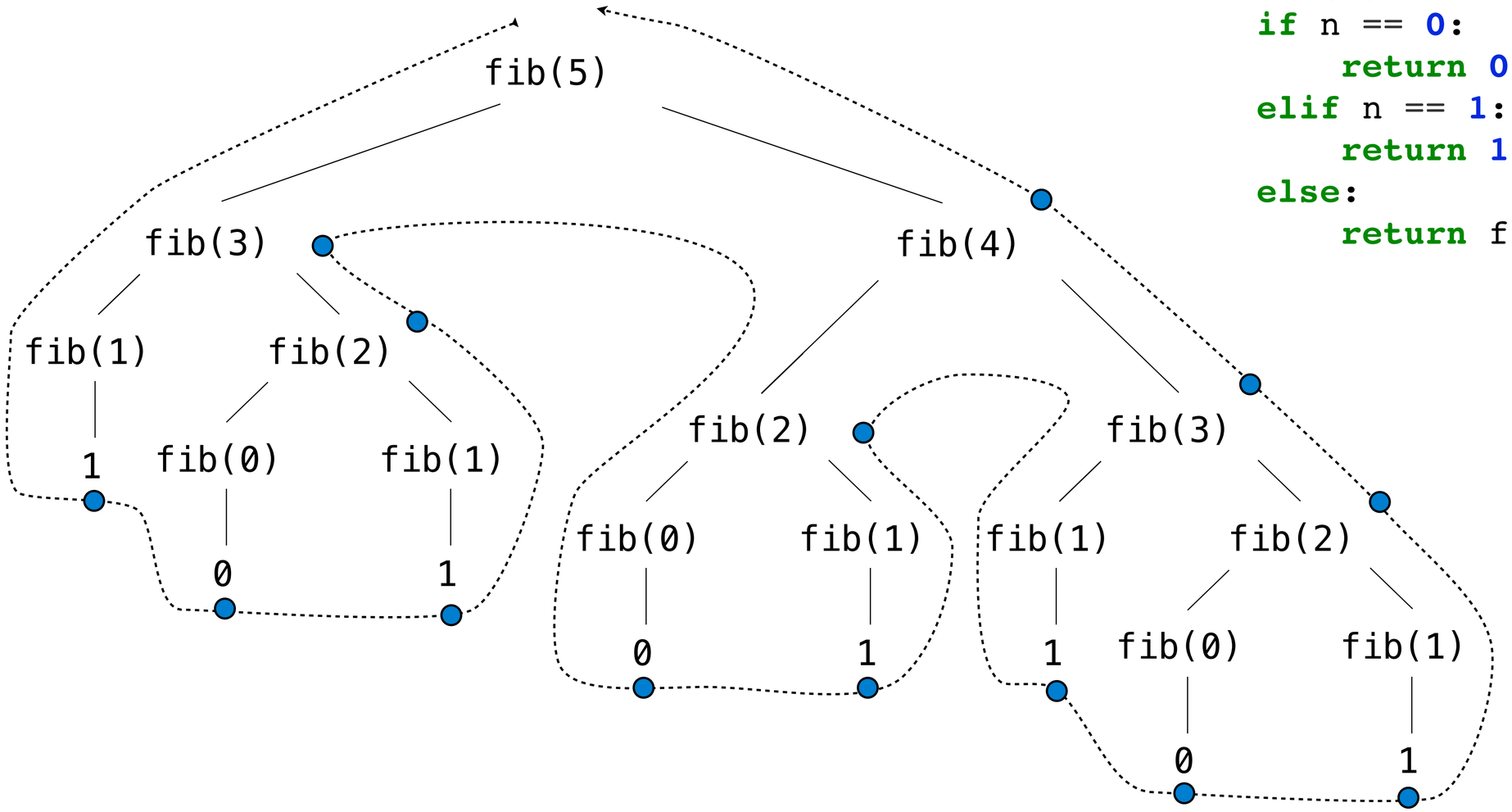



```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:



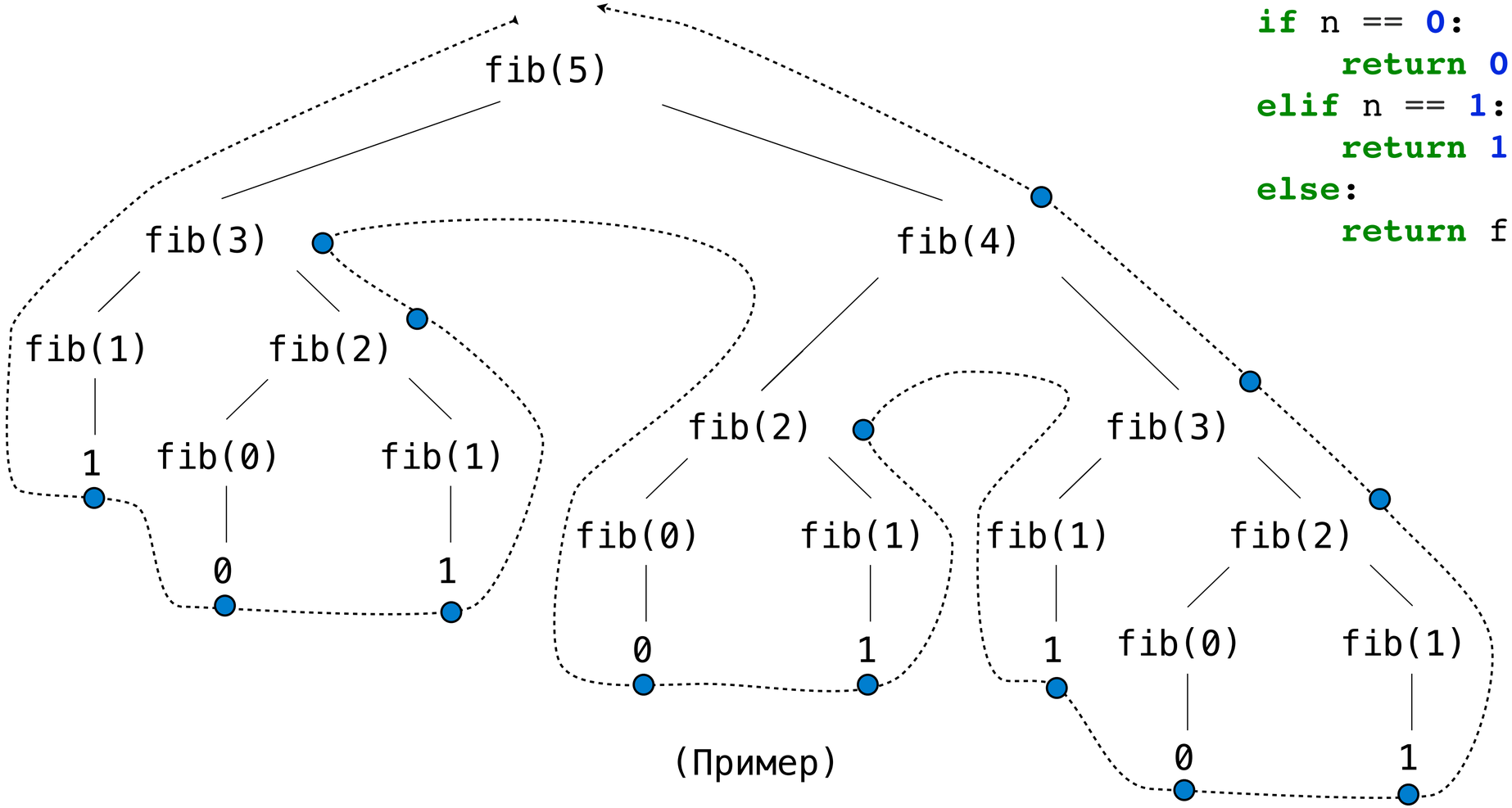

```
def fib(n):
    if n == 0:
        return 0
    elif n == 1:
        return 1
    else:
        return fib(n-2) + fib(n-1)
```

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Рекурсивное вычисление последовательности Фибоначчи

Первый пример о древовидной рекурсии:

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



(Пример)

http://en.wikipedia.org/wiki/File:Fibonacci.jpg

Запоминание

Запоминание

Запоминание

Идея: запоминать результаты, полученные ранее

Запоминание

Идея: запоминать результаты, полученные ранее

```
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

Запоминание

Идея: запоминать результаты, полученные ранее

```
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

Ключи — аргументы, значения — возвращаемые значения

Запоминание

Идея: запоминать результаты, полученные ранее

```
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

Ключи — аргументы, значения — возвращаемые значения

То же поведение, что и у f, если f — чистая функция

Запоминание

Идея: запоминать результаты, полученные ранее

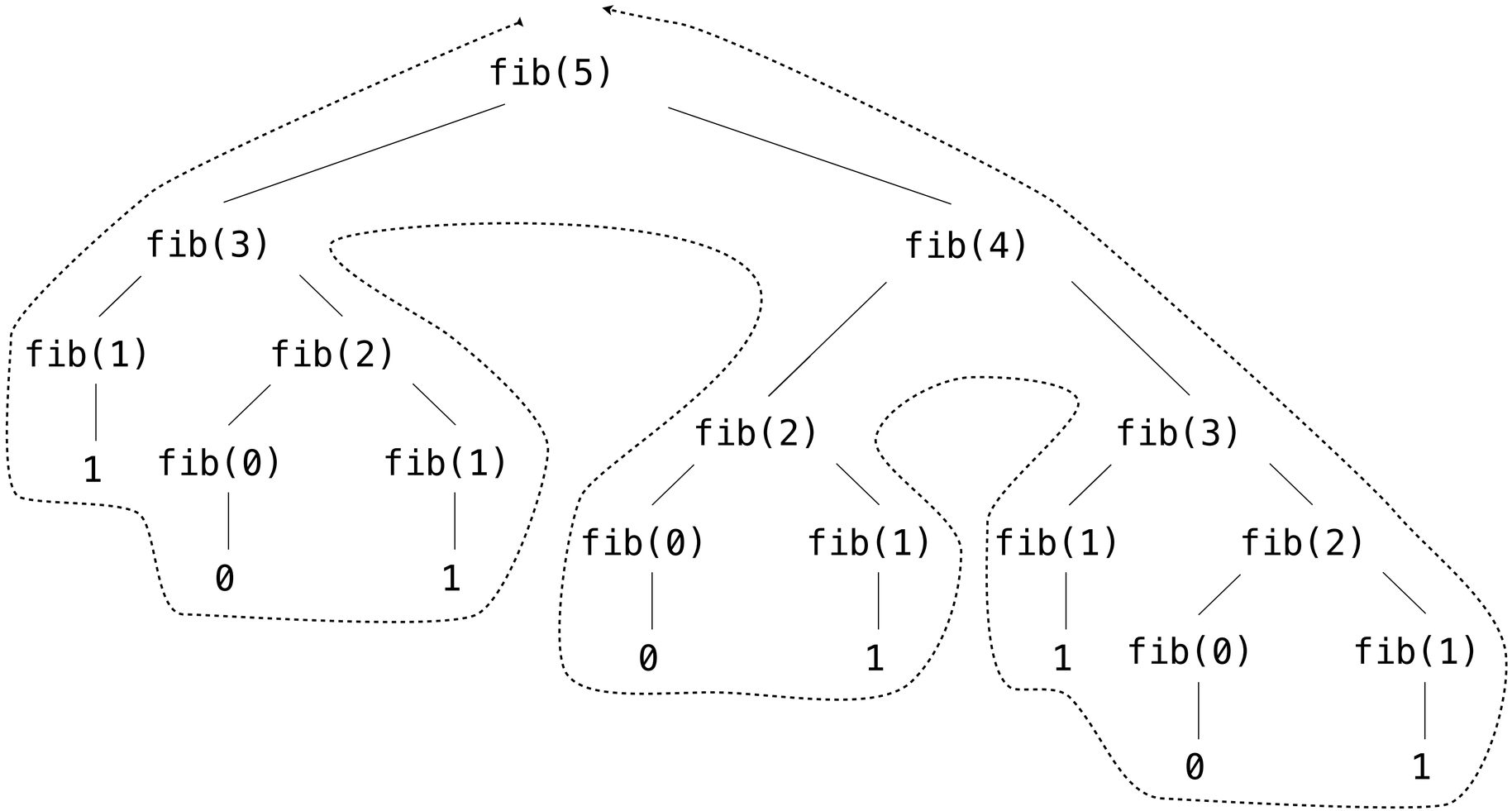
```
def memo(f):
    cache = {}
    def memoized(n):
        if n not in cache:
            cache[n] = f(n)
        return cache[n]
    return memoized
```

Ключи — аргументы, значения — возвращаемые значения

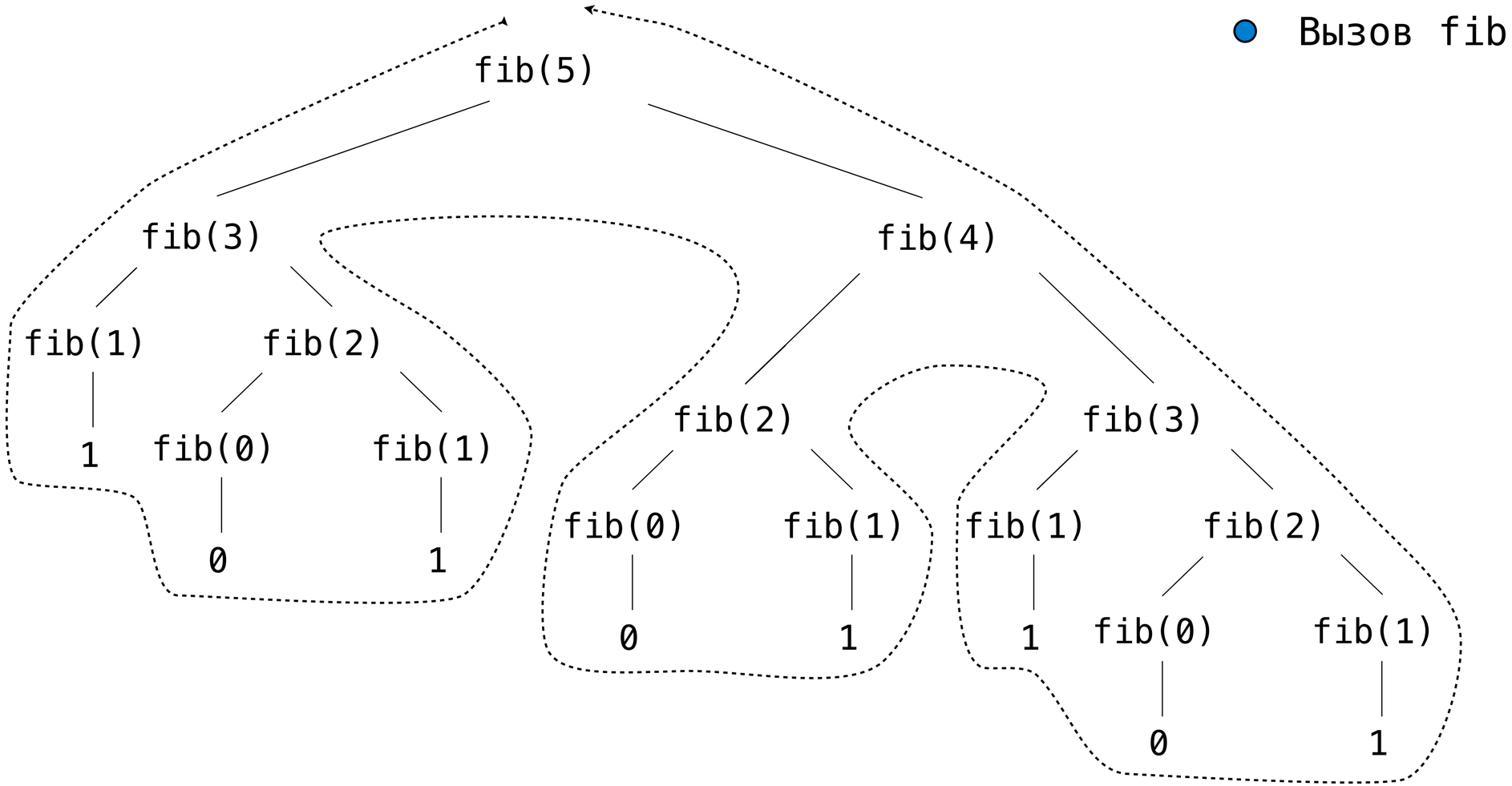
То же поведение, что и у f, если f — чистая функция

(Пример)

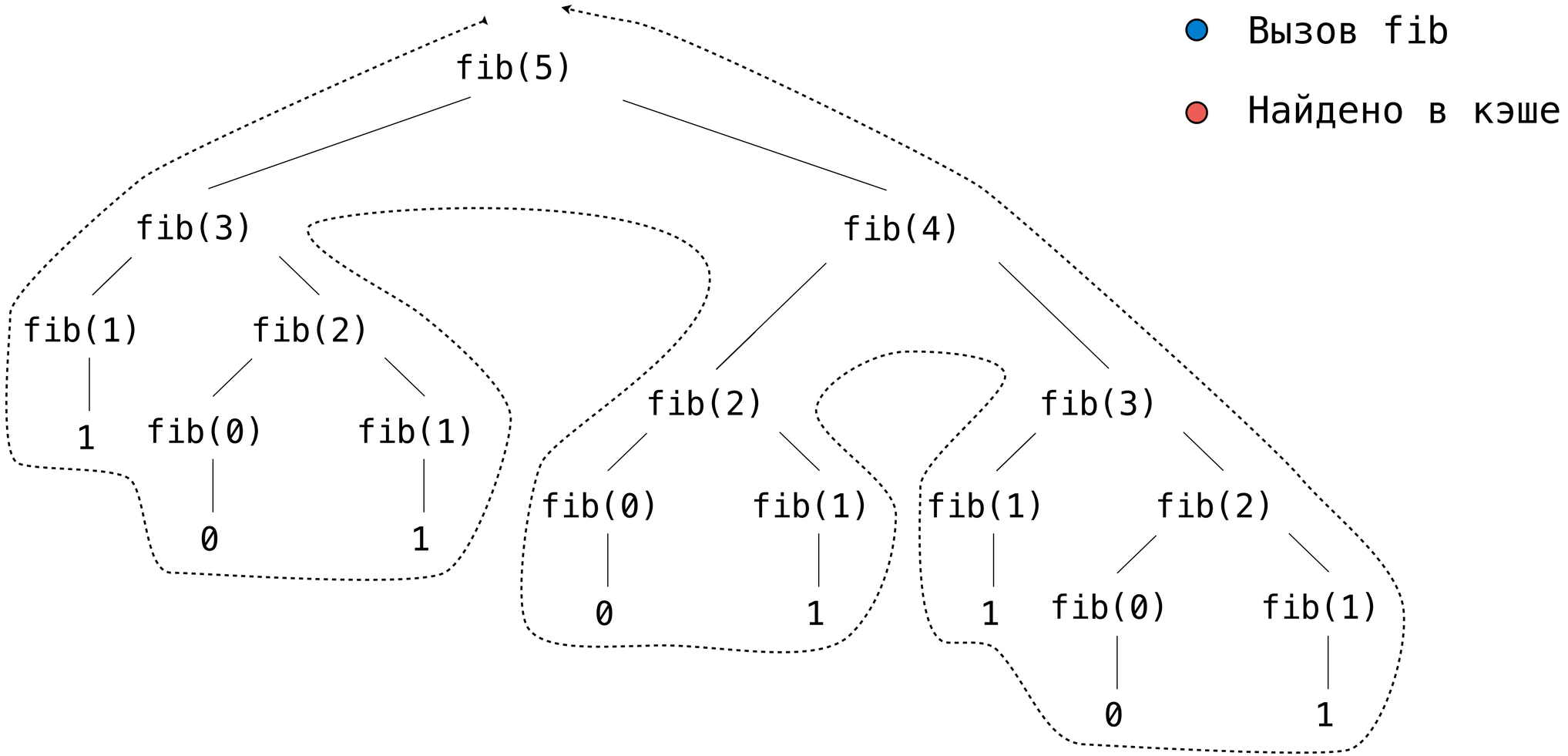
Запоминание в древовидной рекурсии



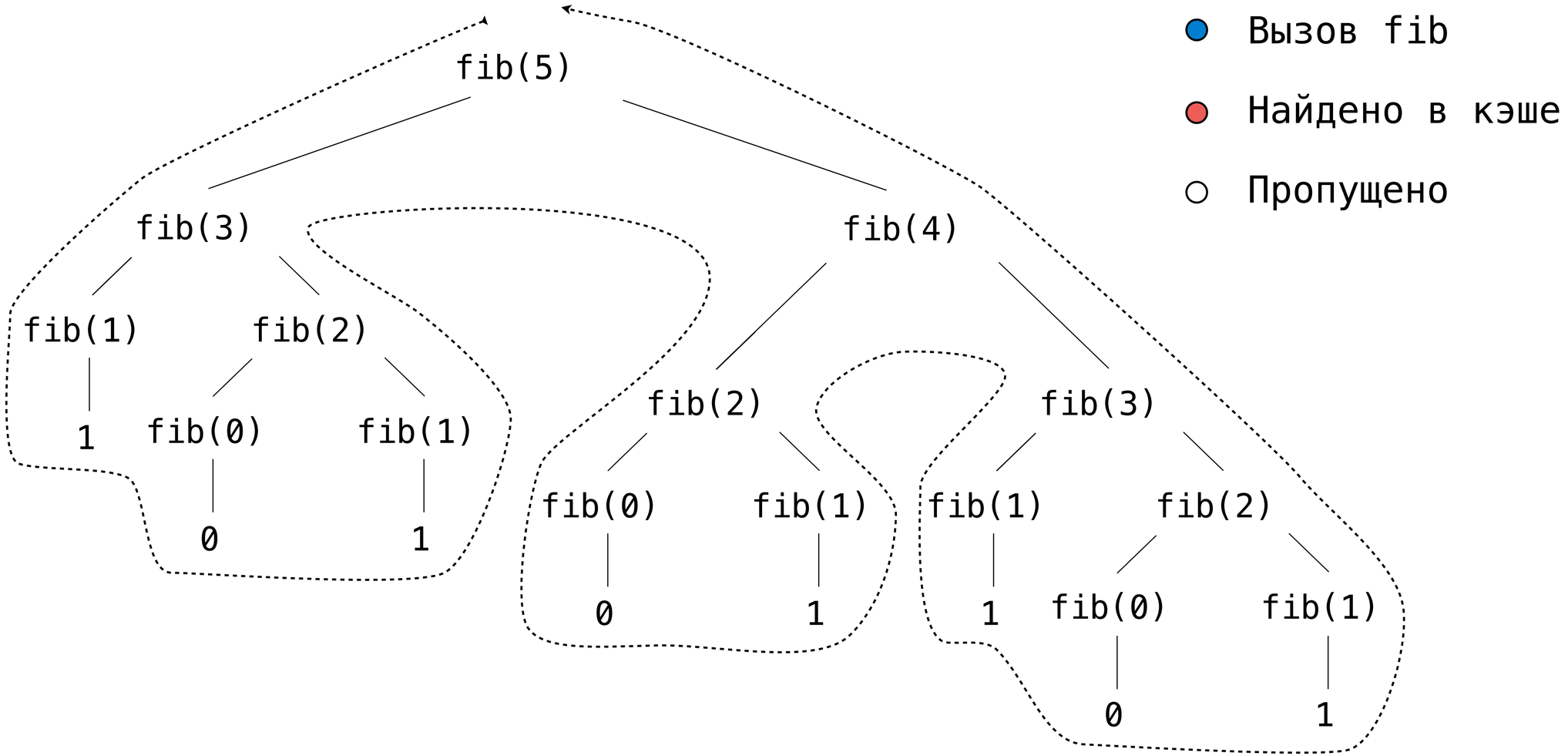
Запоминание в древовидной рекурсии



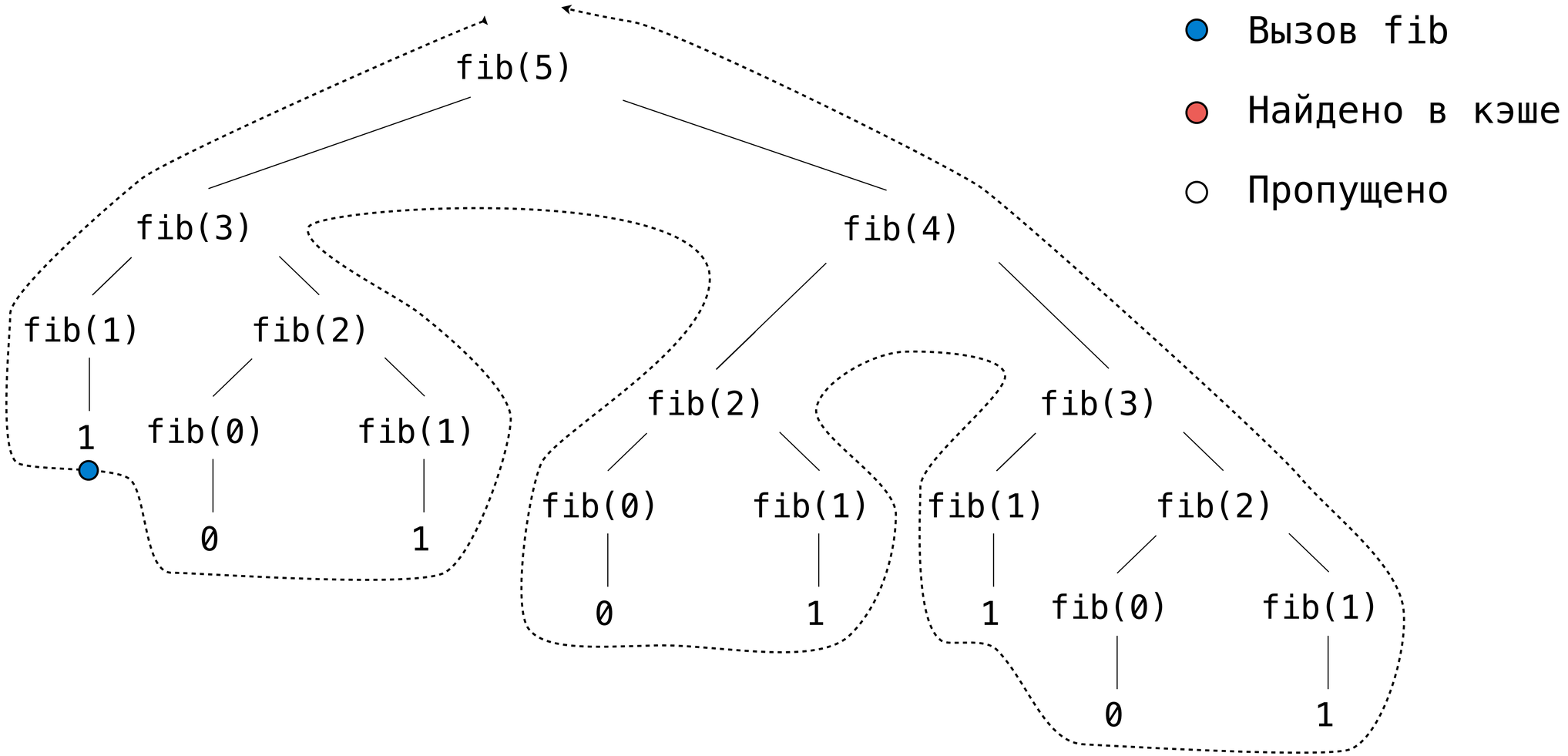
Запоминание в древовидной рекурсии



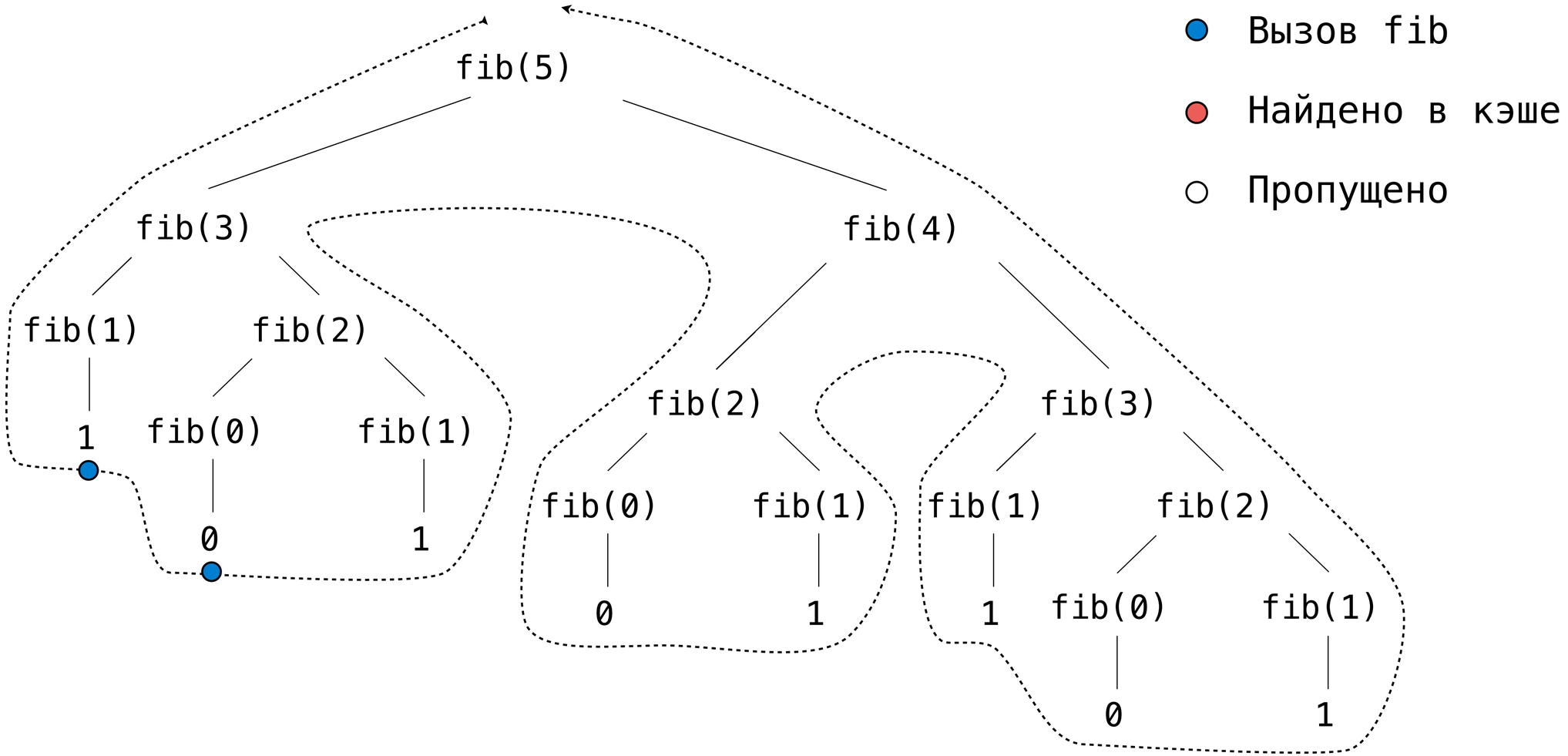
Запоминание в древовидной рекурсии



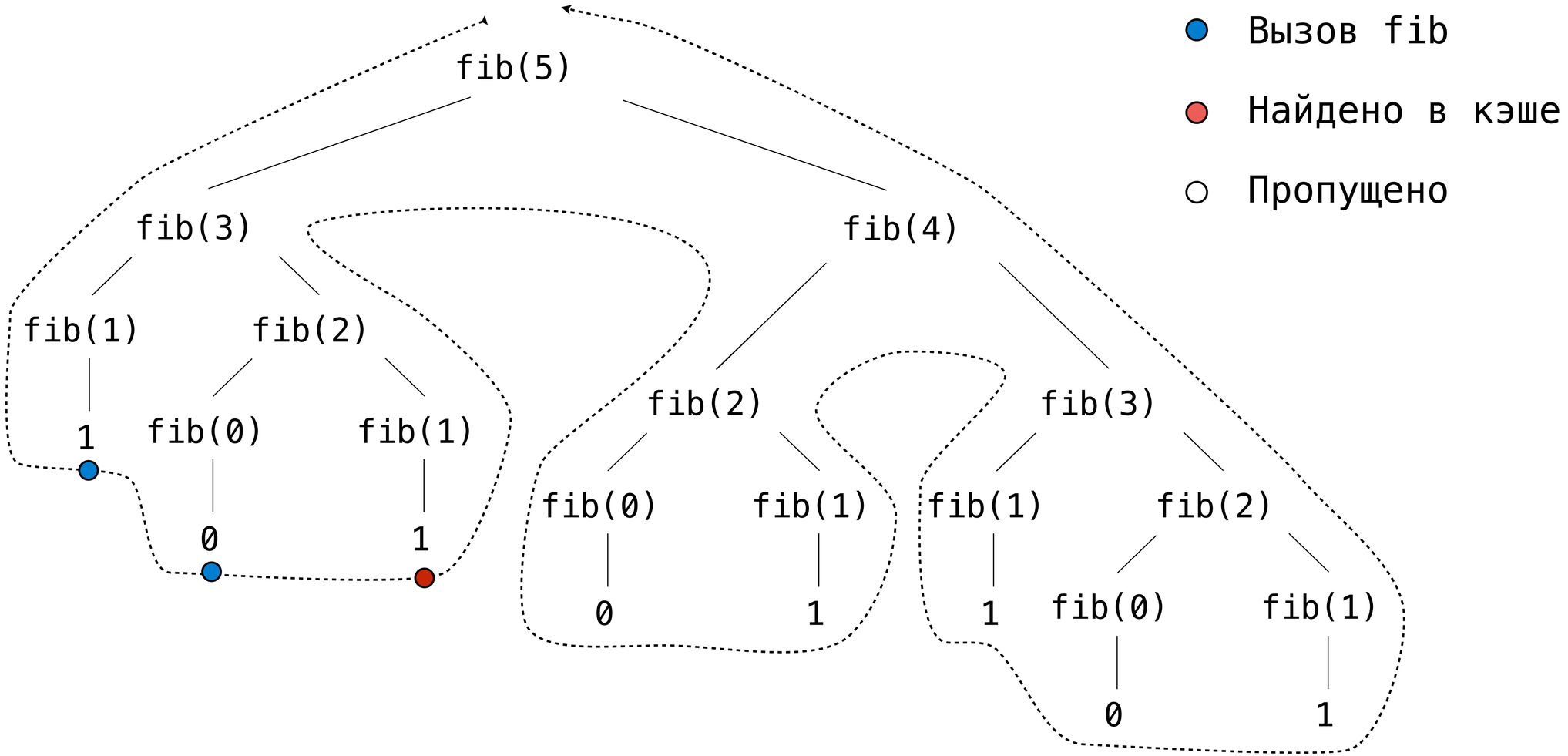
Запоминание в древовидной рекурсии



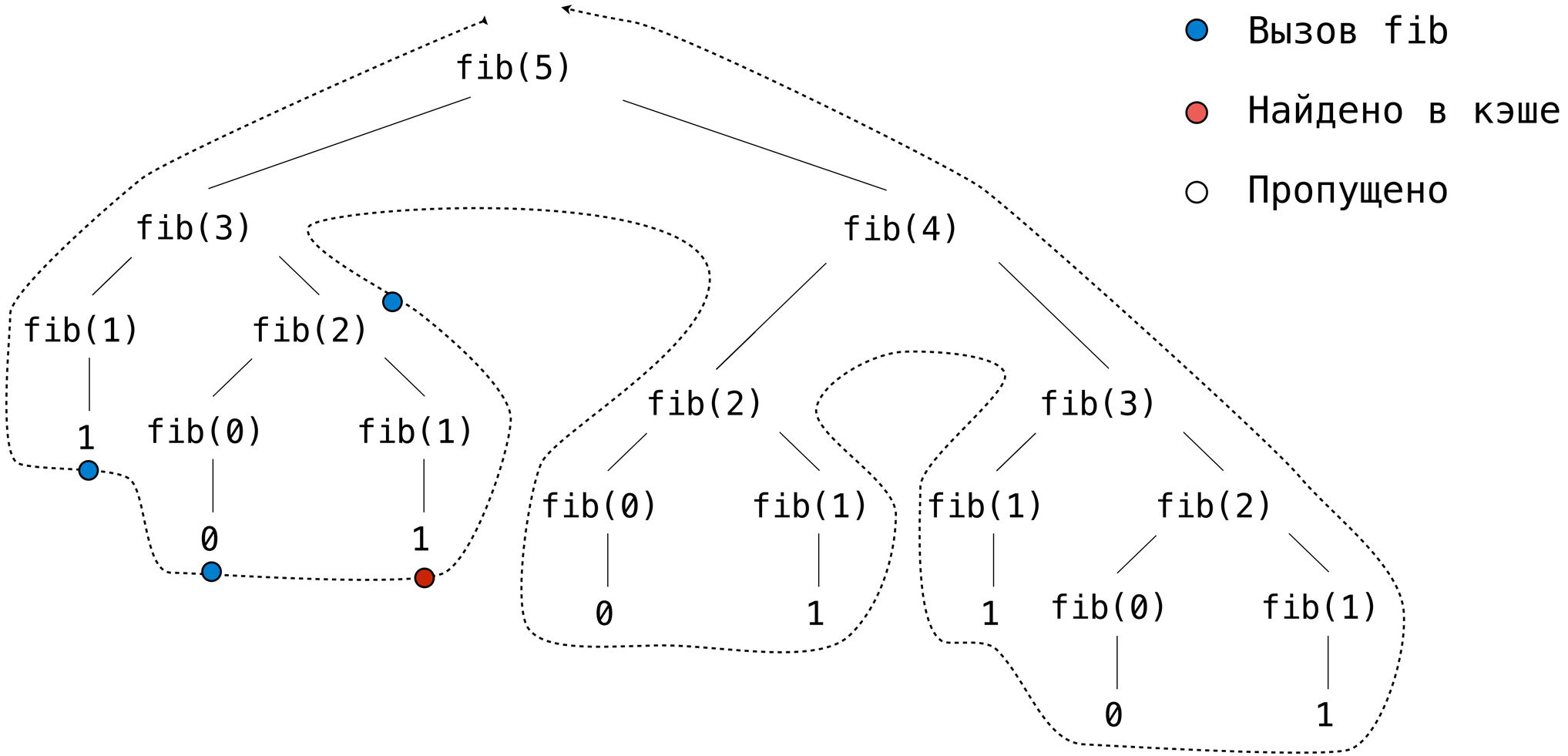
Запоминание в древовидной рекурсии



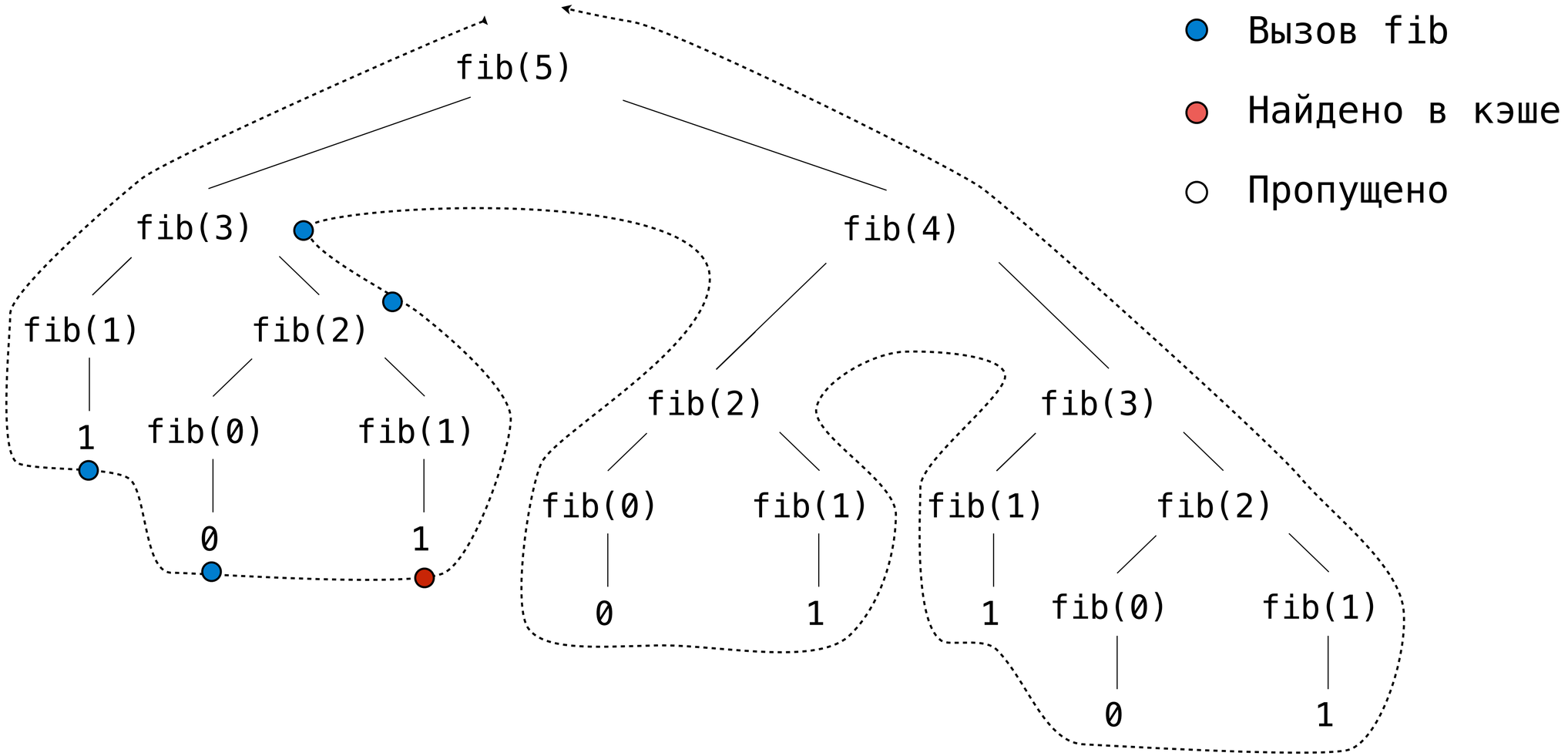
Запоминание в древовидной рекурсии



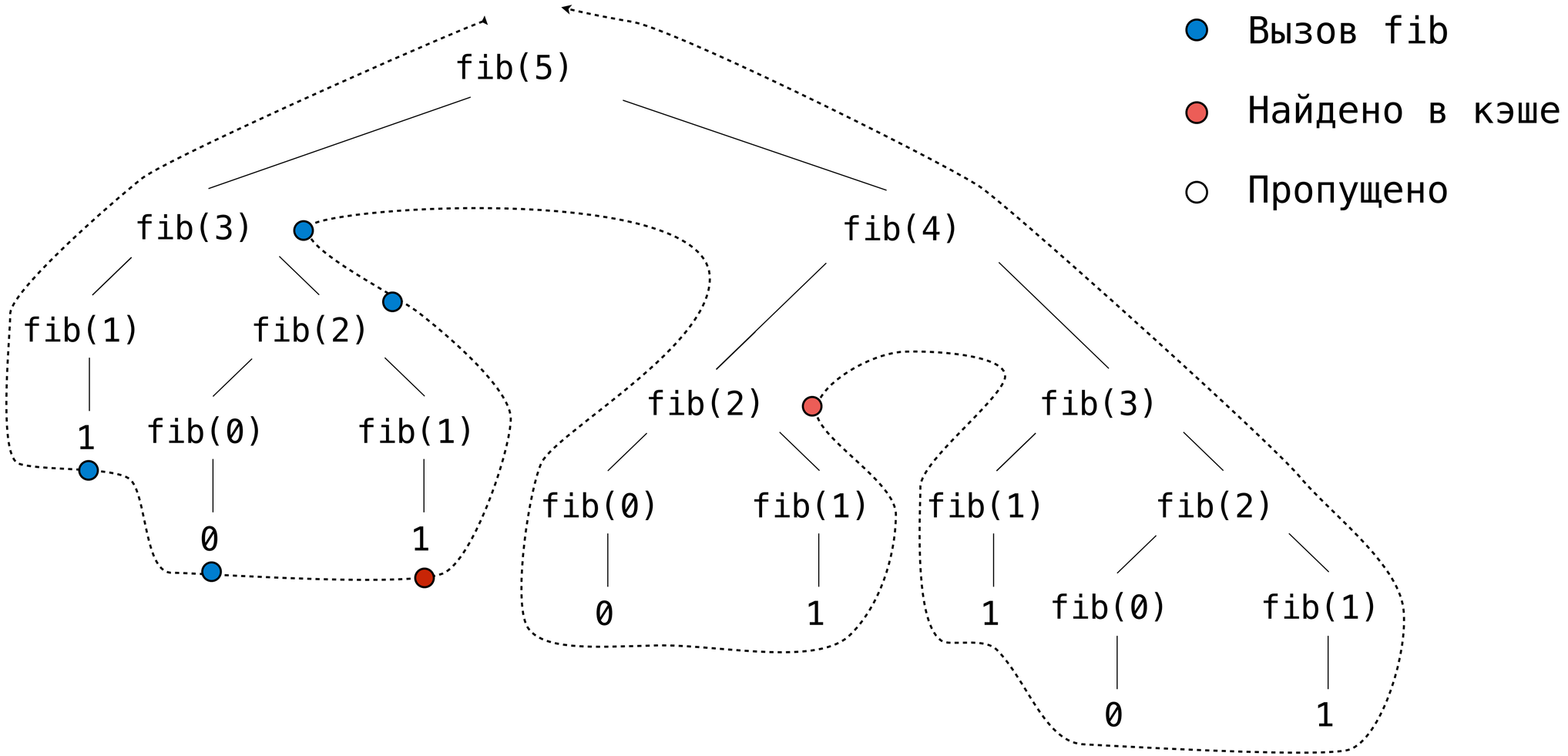
Запоминание в древовидной рекурсии



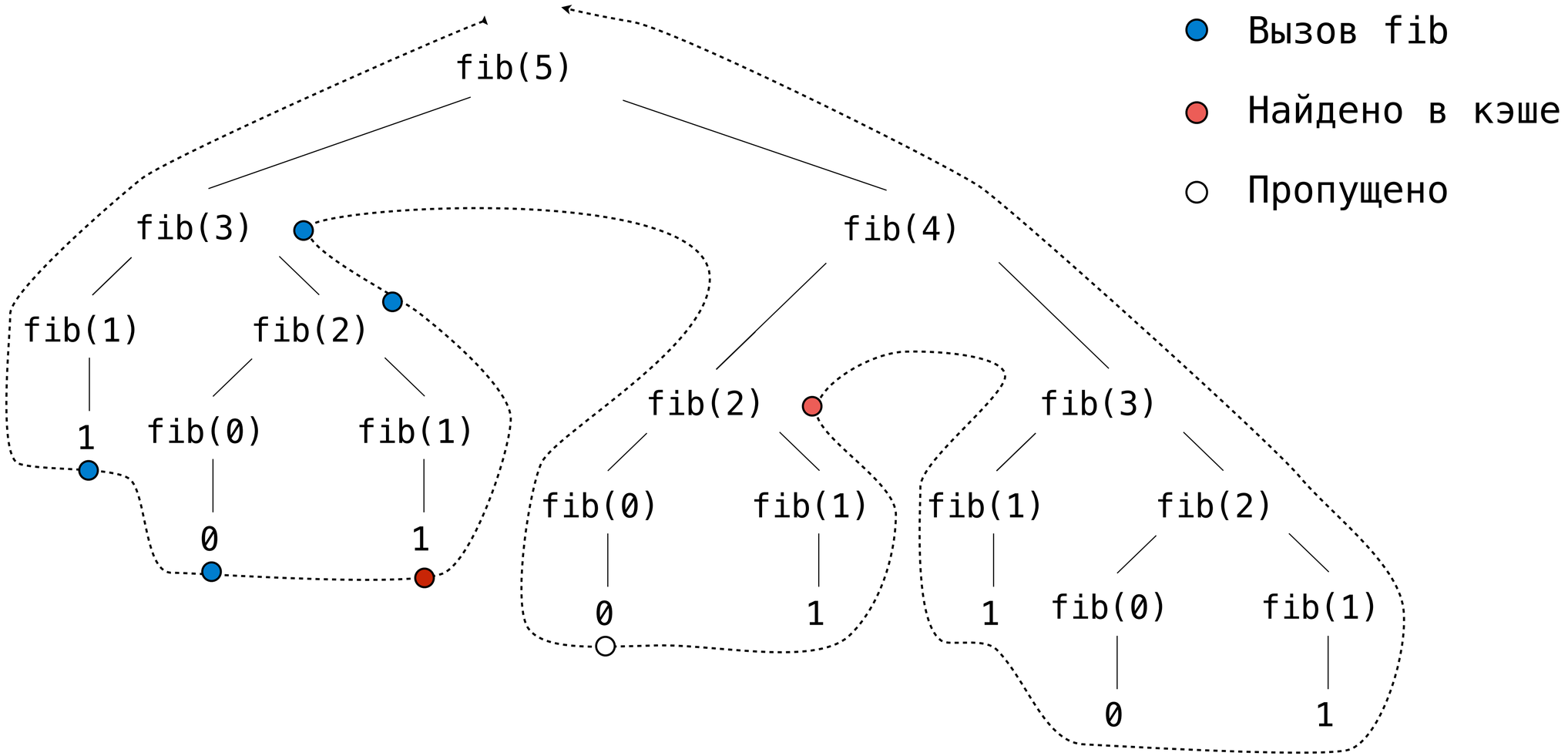
Запоминание в древовидной рекурсии



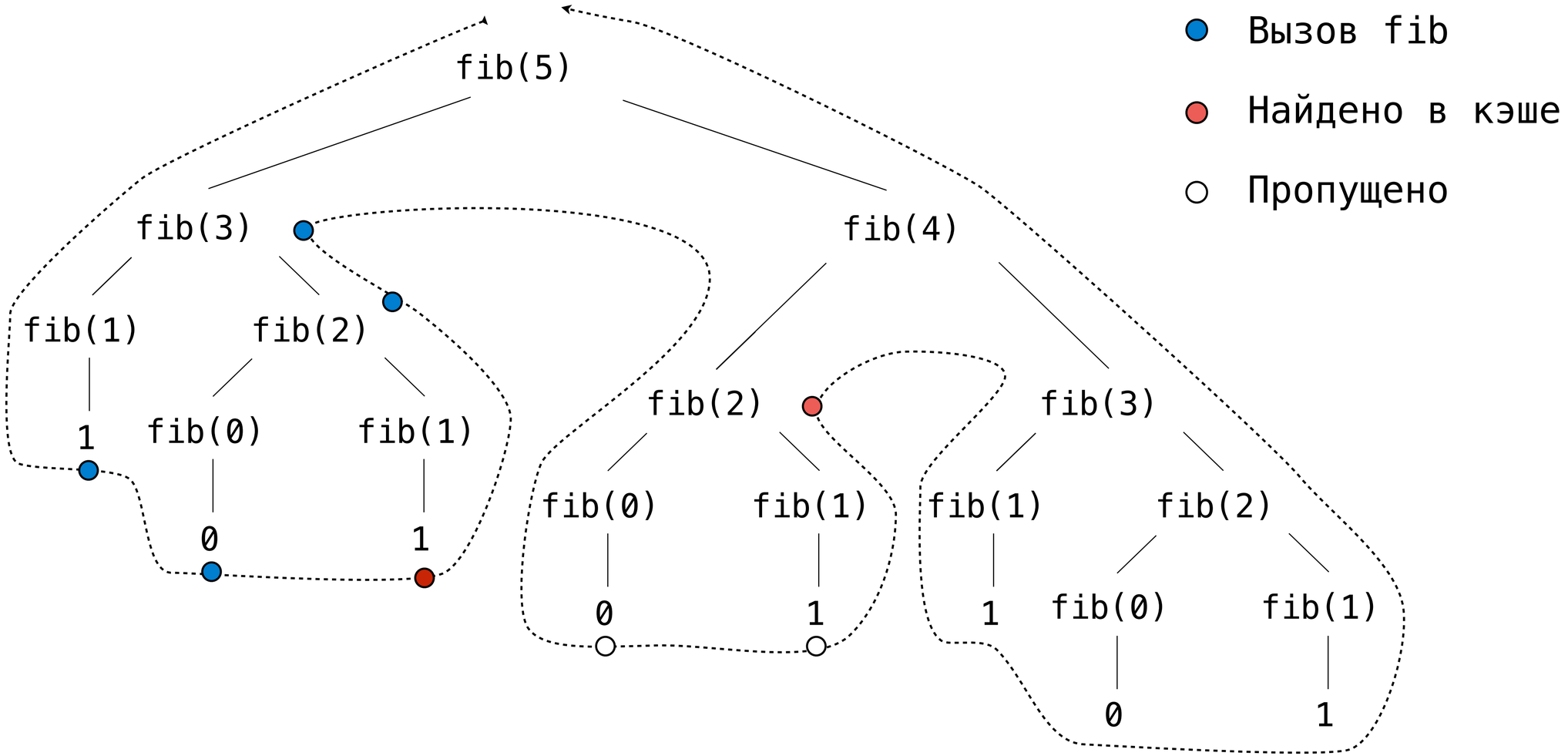
Запоминание в древовидной рекурсии



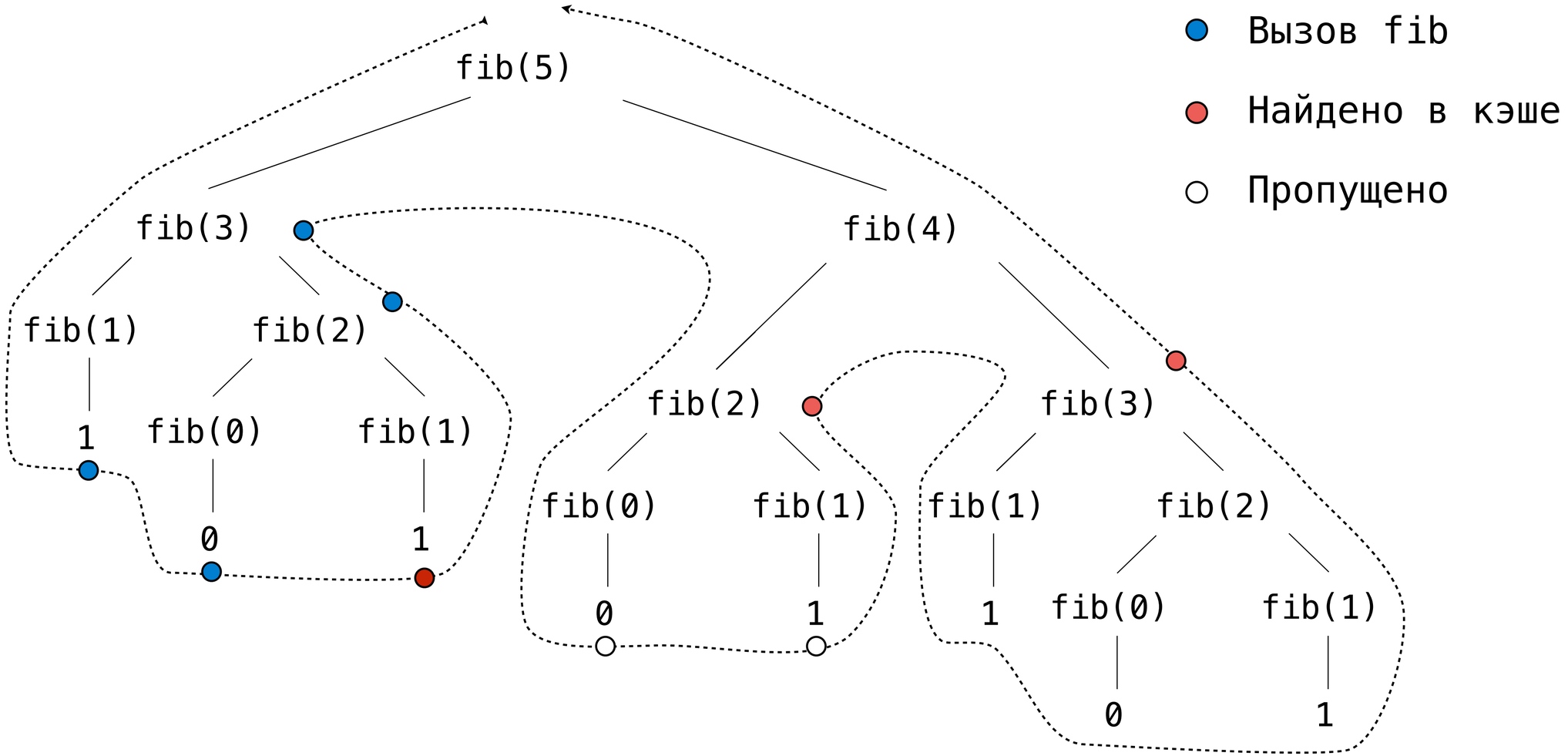
Запоминание в древовидной рекурсии



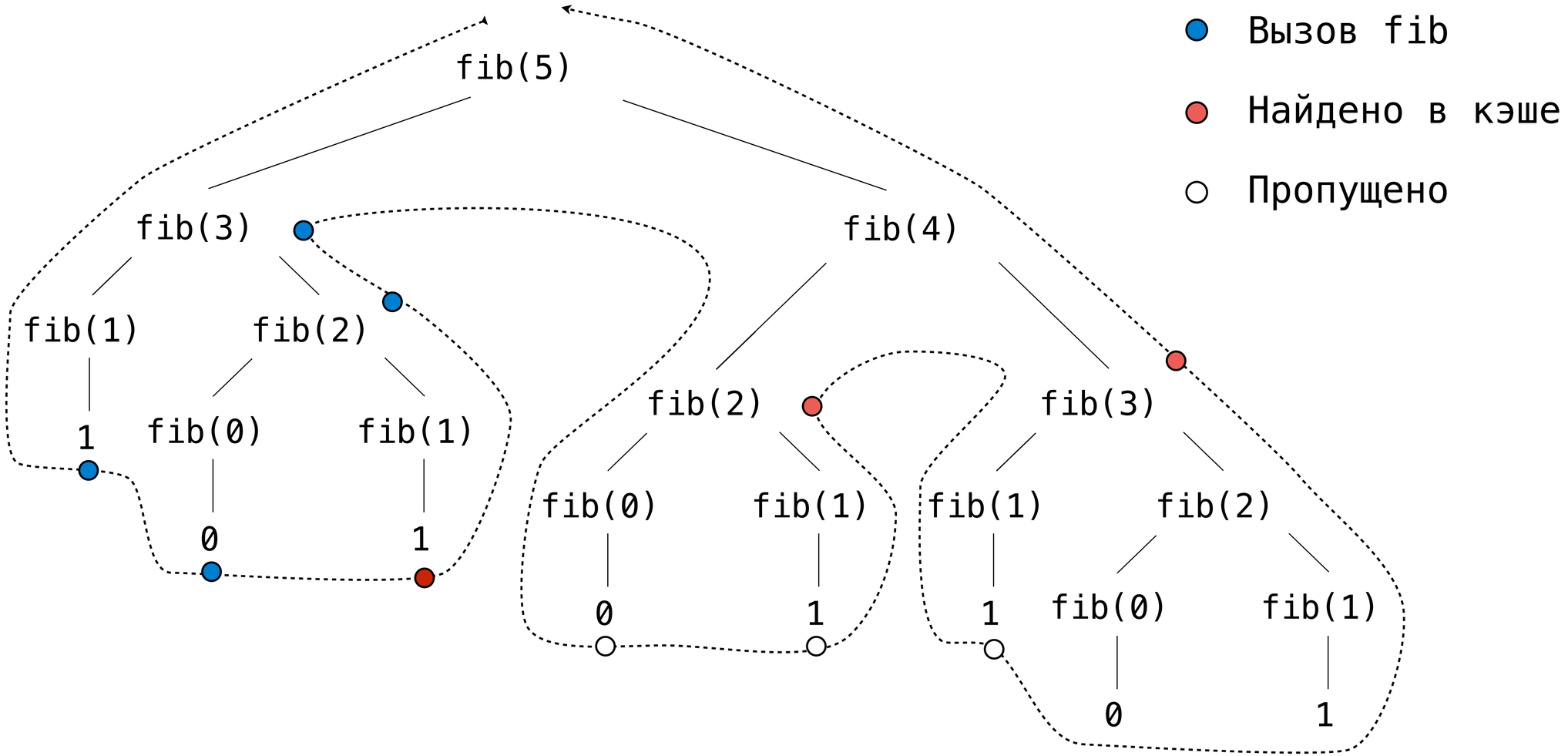
Запоминание в древовидной рекурсии



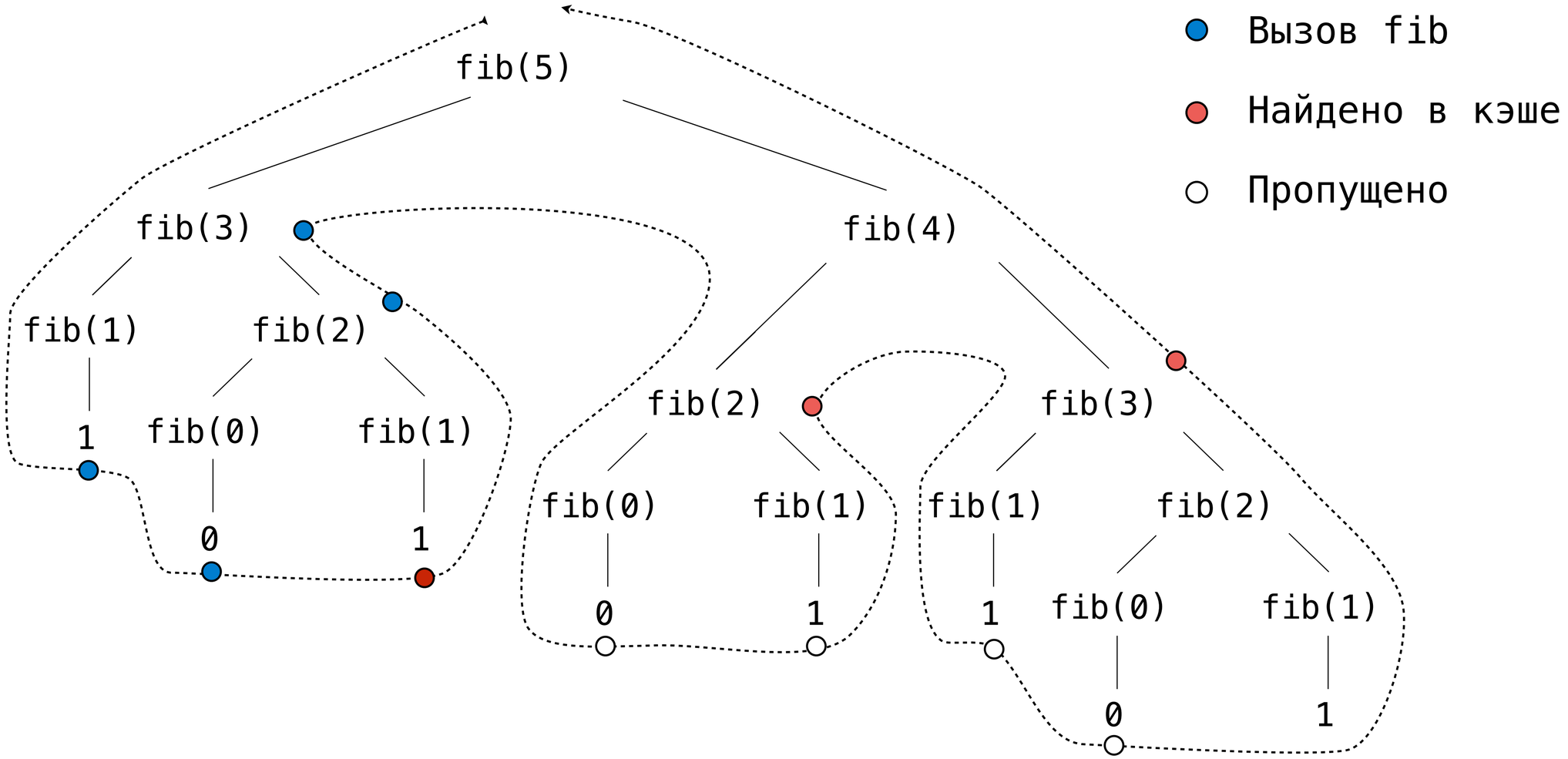
Запоминание в древовидной рекурсии



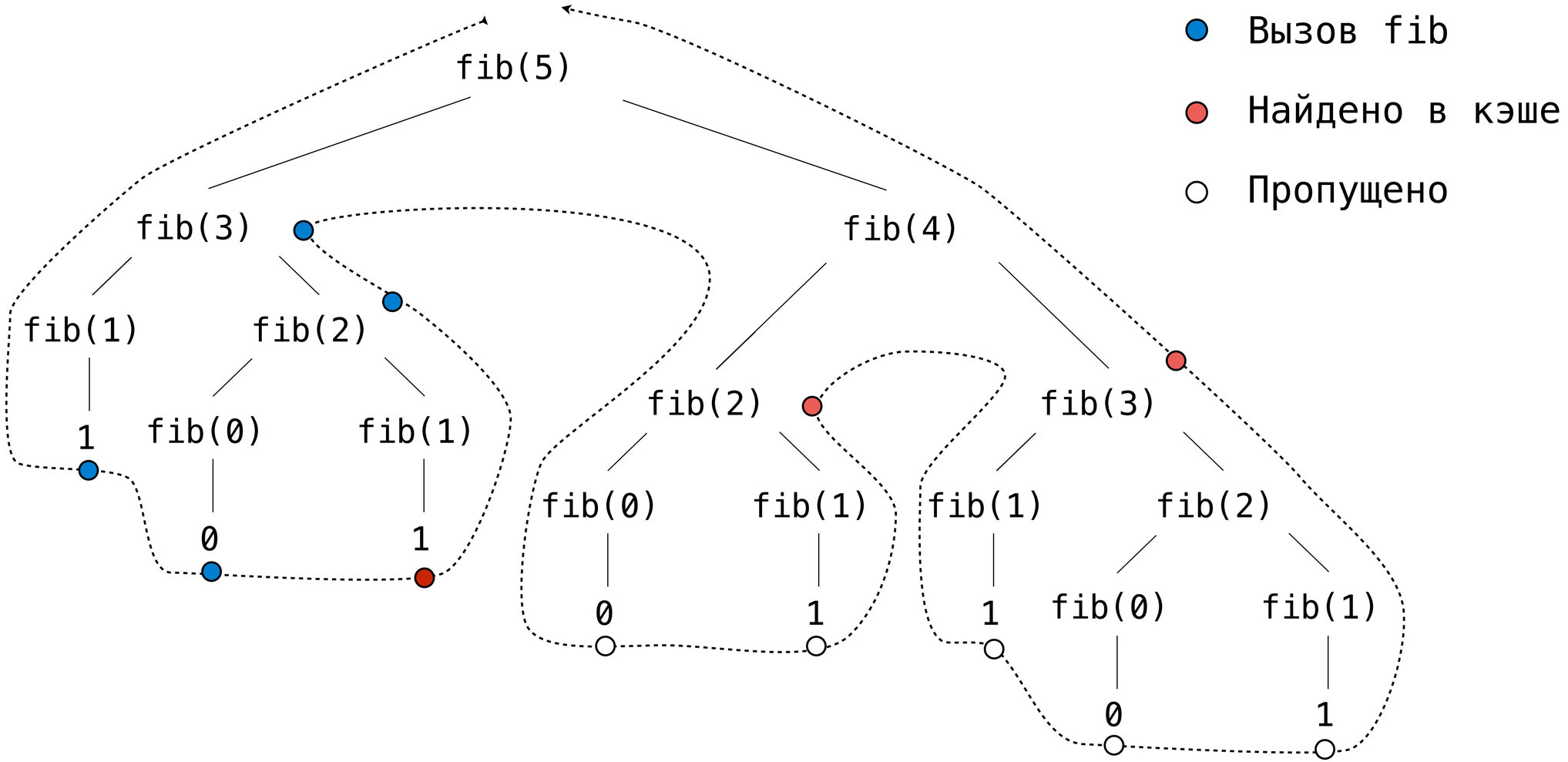
Запоминание в древовидной рекурсии



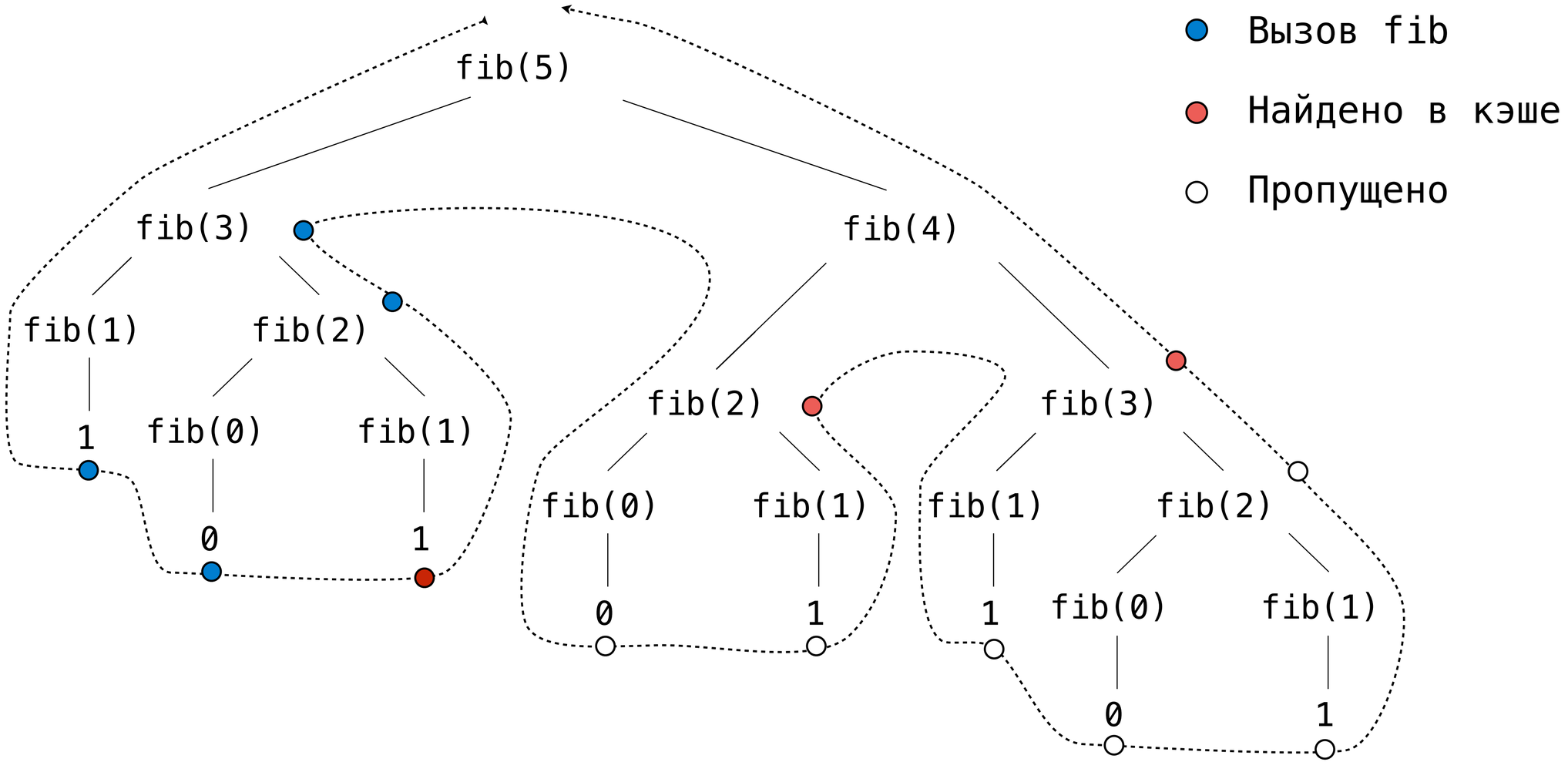
Запоминание в древовидной рекурсии



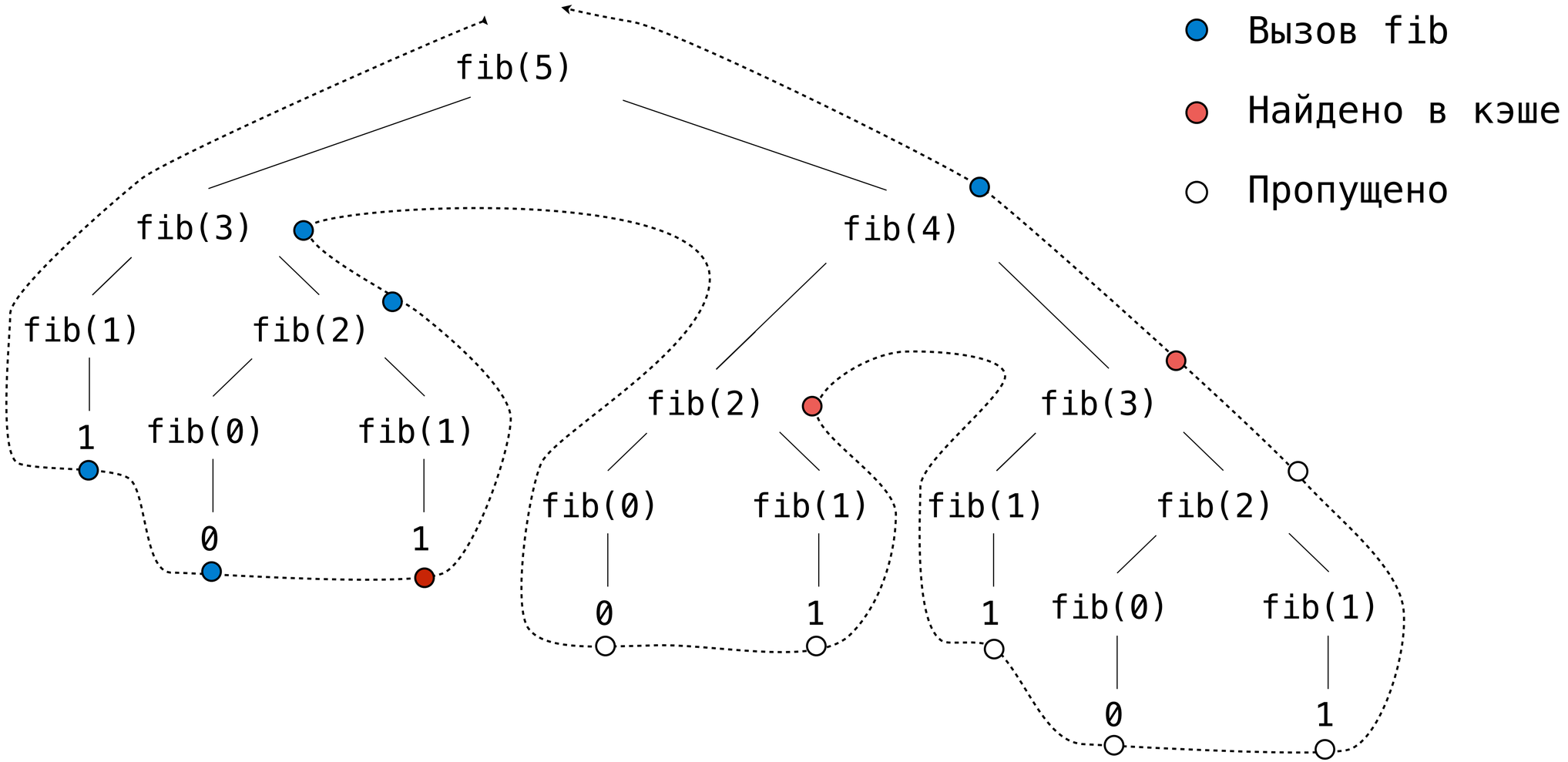
Запоминание в древовидной рекурсии



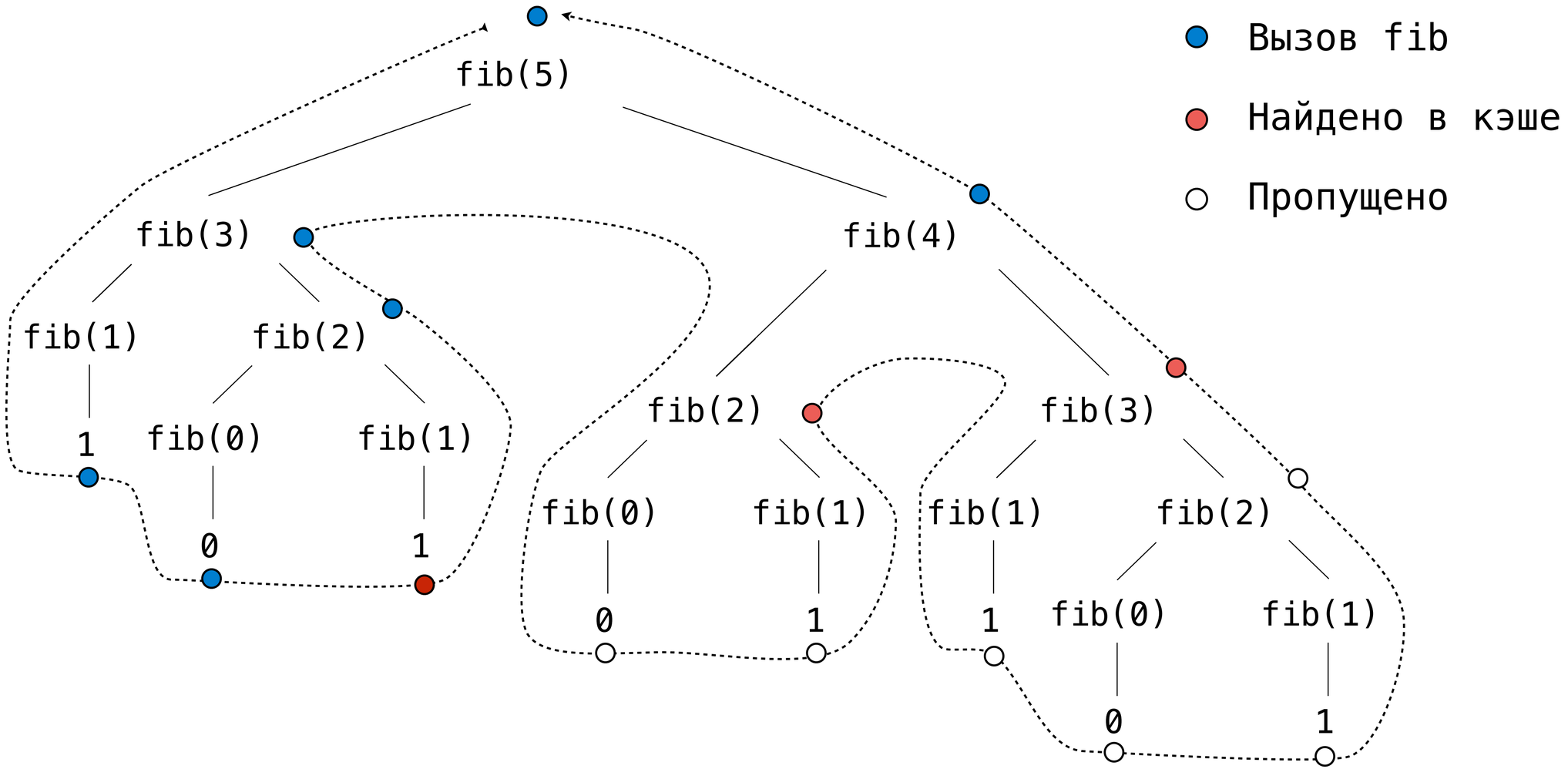
Запоминание в древовидной рекурсии



Запоминание в древовидной рекурсии



Запоминание в древовидной рекурсии



Пространство

Затраты пространства

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Память для других значений и фреймов может быть освобождена

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Память для других значений и фреймов может быть освобождена

Активные окружения:

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Память для других значений и фреймов может быть освобождена

Активные окружения:

- Окружения для любых вызовов функции в процессе текущего вычисления

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Память для других значений и фреймов может быть освобождена

Активные окружения:

- Окружения для любых вызовов функции в процессе текущего вычисления
- Родительские окружения для функций указаных в активных окружениях

Затраты пространства

Фреймы каких окружений нужно помнить во время вычисления?

В любой момент времени существует набор активных окружений

Значения и фреймы в активных окружениях потребляют память (т.е. место, т.е. пространство)

Память для других значений и фреймов может быть освобождена

Активные окружения:

- Окружения для любых вызовов функции в процессе текущего вычисления
- Родительские окружения для функций указаных в активных окружениях

(Пример)

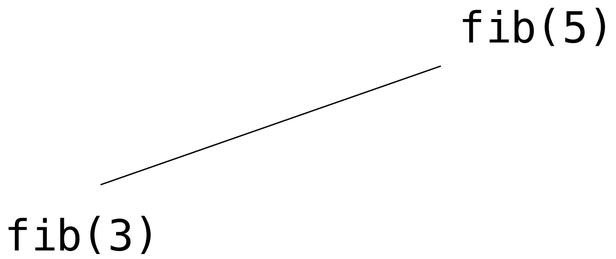
Интерактивная диаграмма

Пространственные затраты для последовательности Фибоначчи

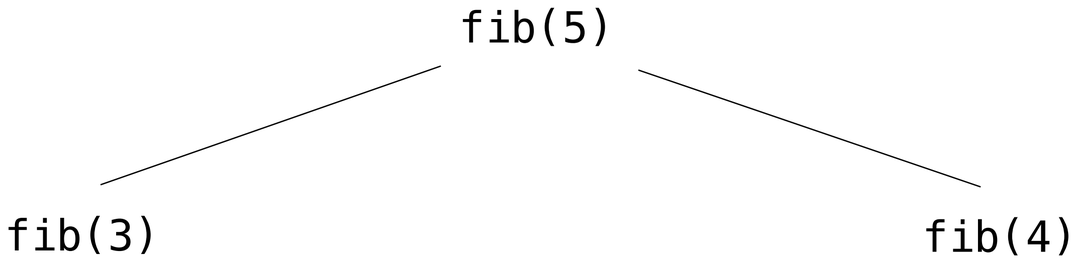
Пространственные затраты для последовательности Фибоначчи

```
fib(5)
```

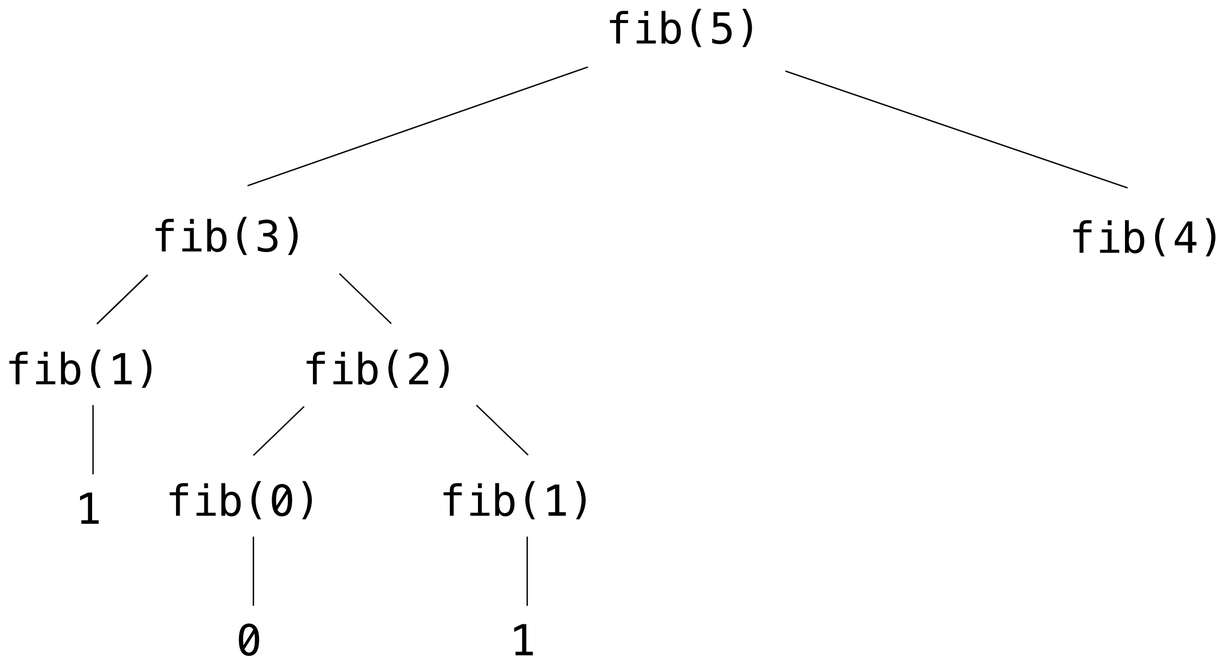
Пространственные затраты для последовательности Фибоначчи



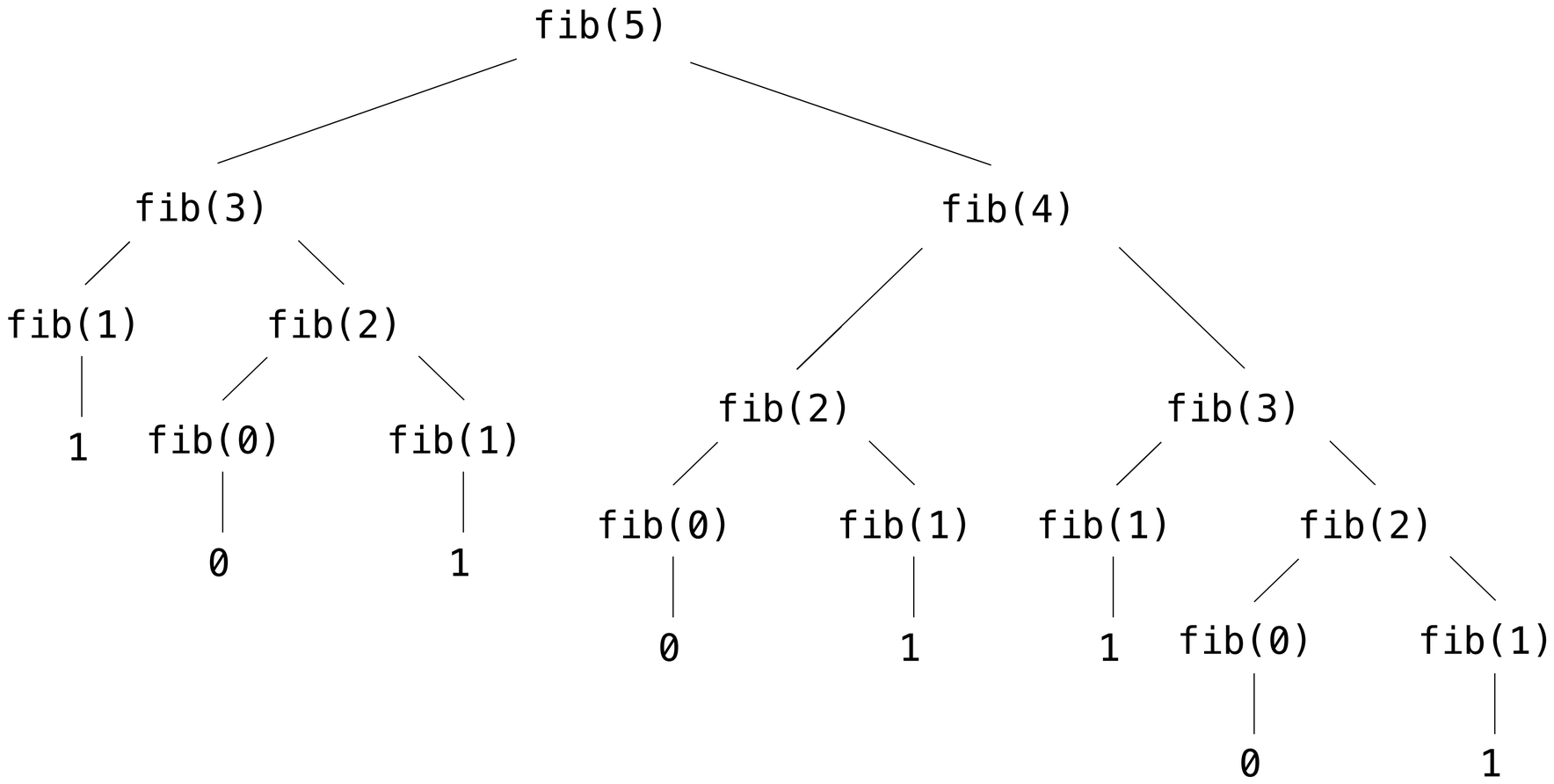
Пространственные затраты для последовательности Фибоначчи



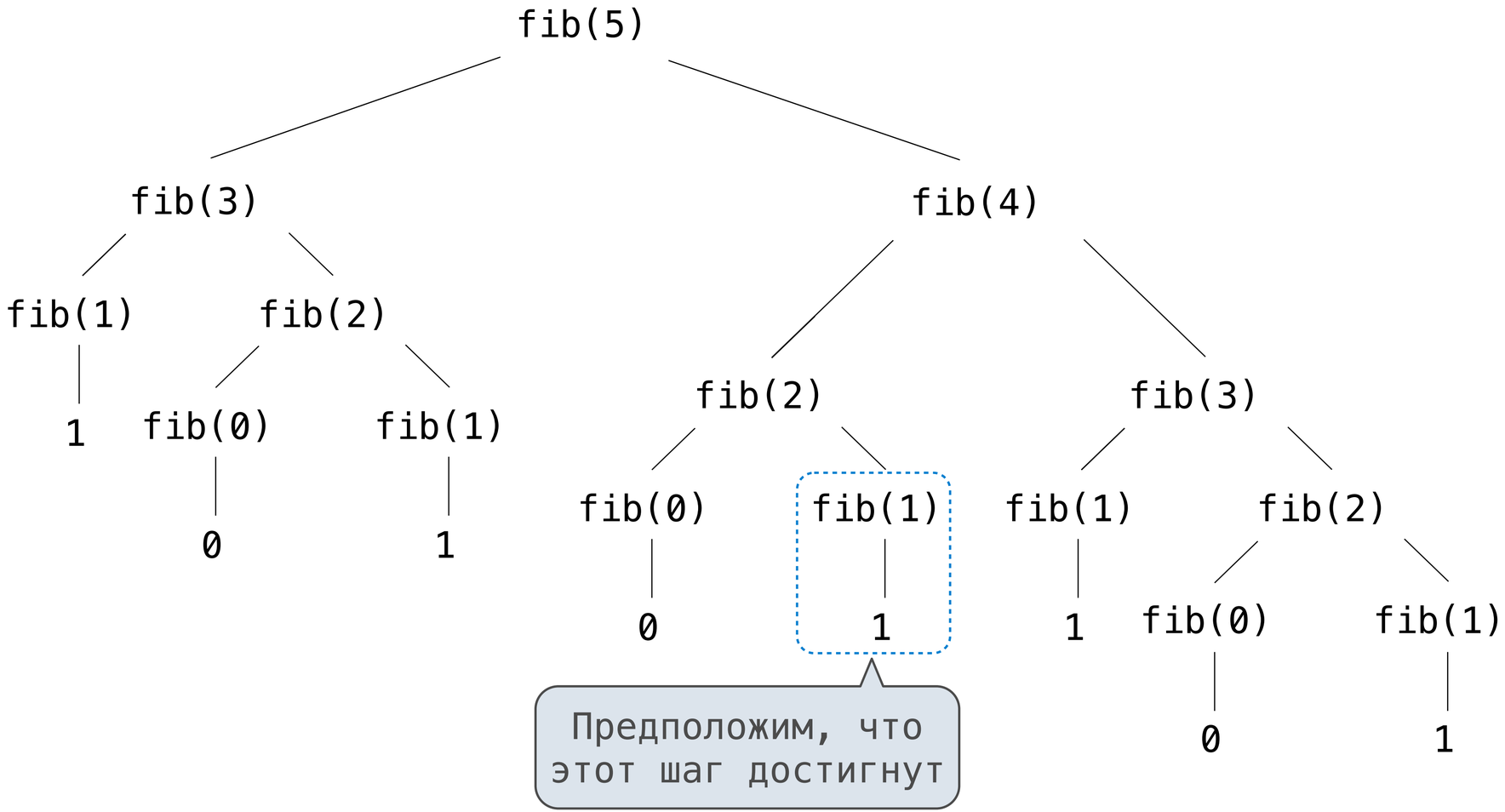
Пространственные затраты для последовательности Фибоначчи



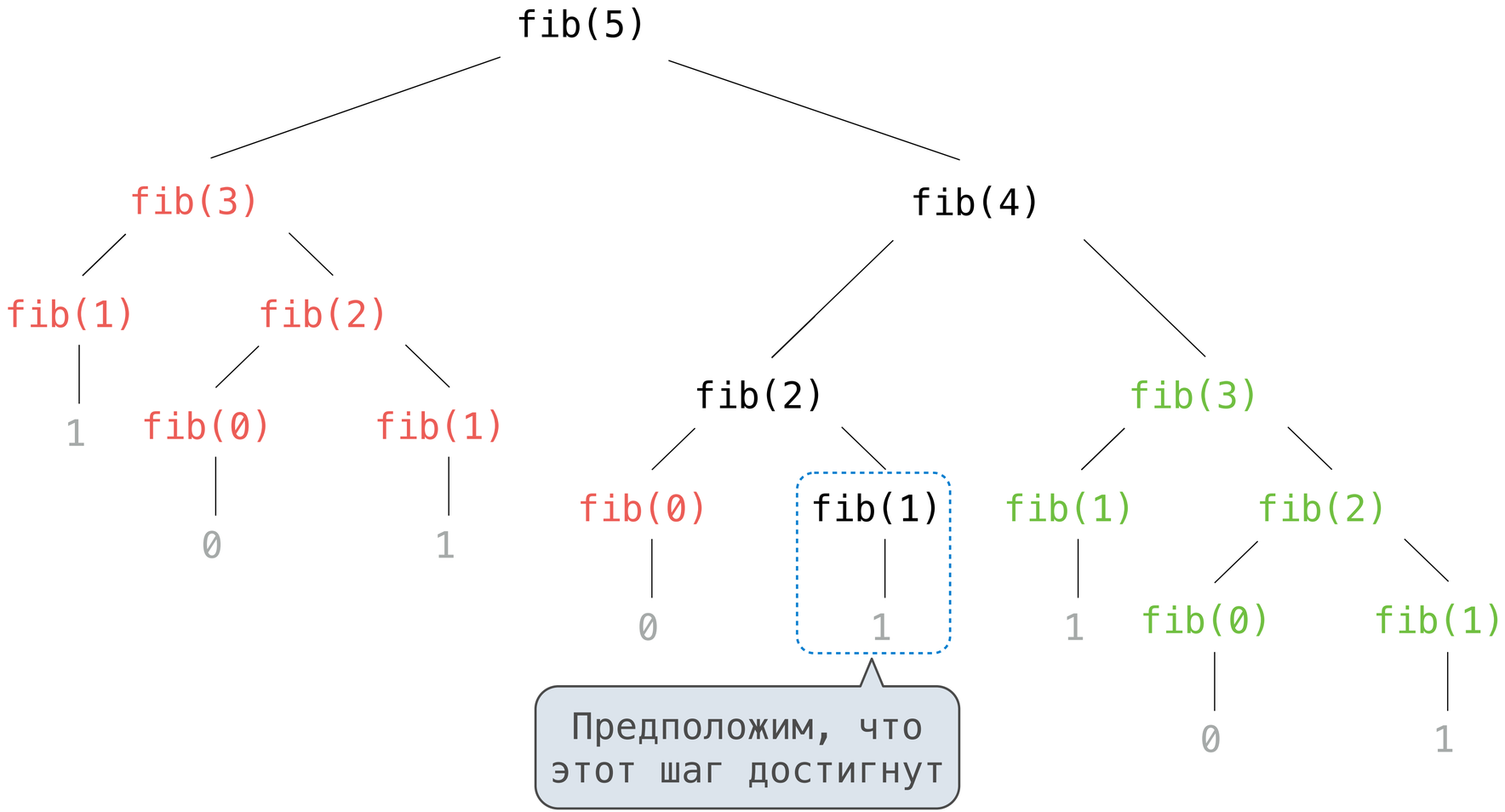
Пространственные затраты для последовательности Фибоначчи



Пространственные затраты для последовательности Фибоначчи

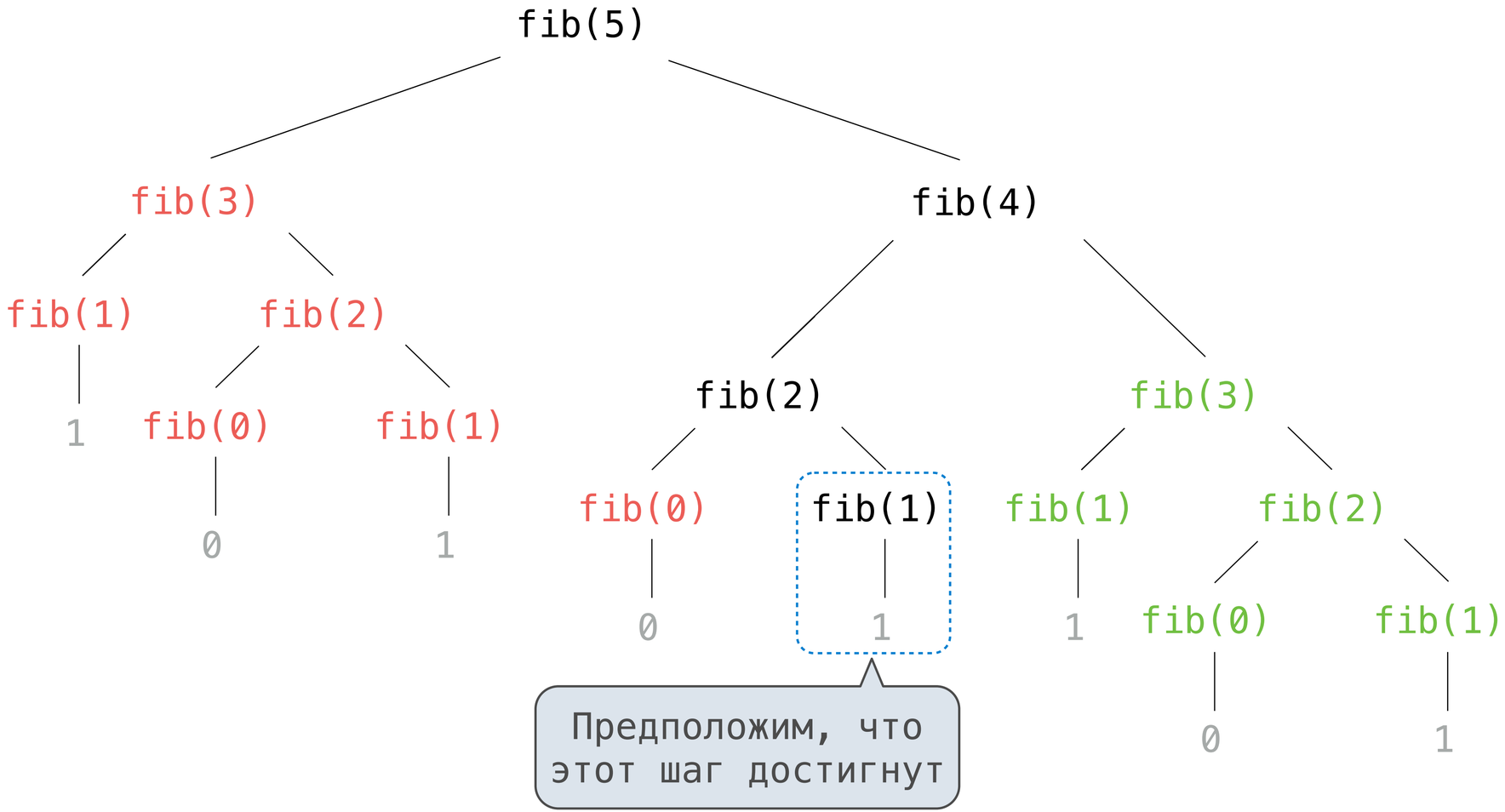


Пространственные затраты для последовательности Фибоначчи

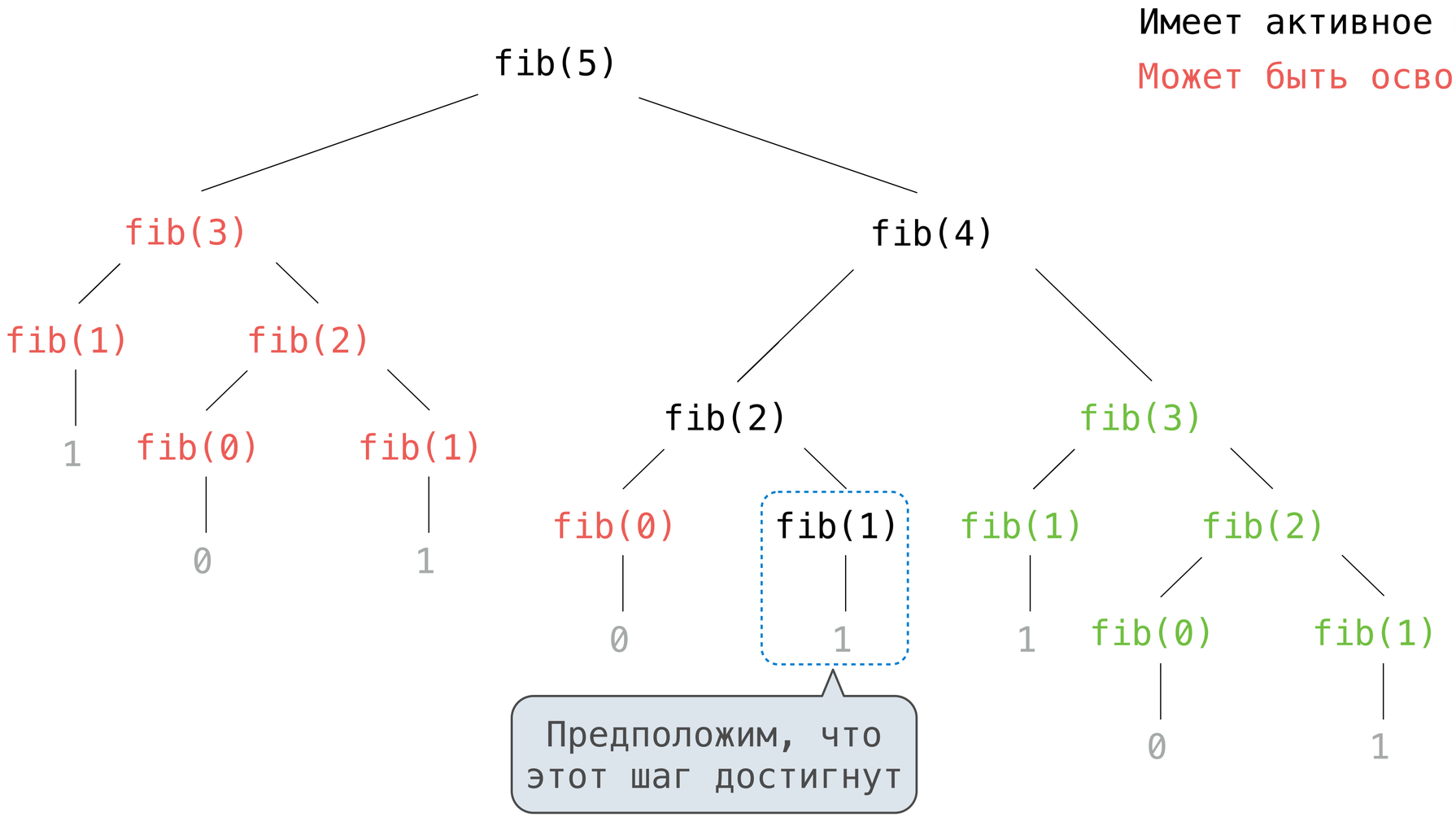


Пространственные затраты для последовательности Фибоначчи

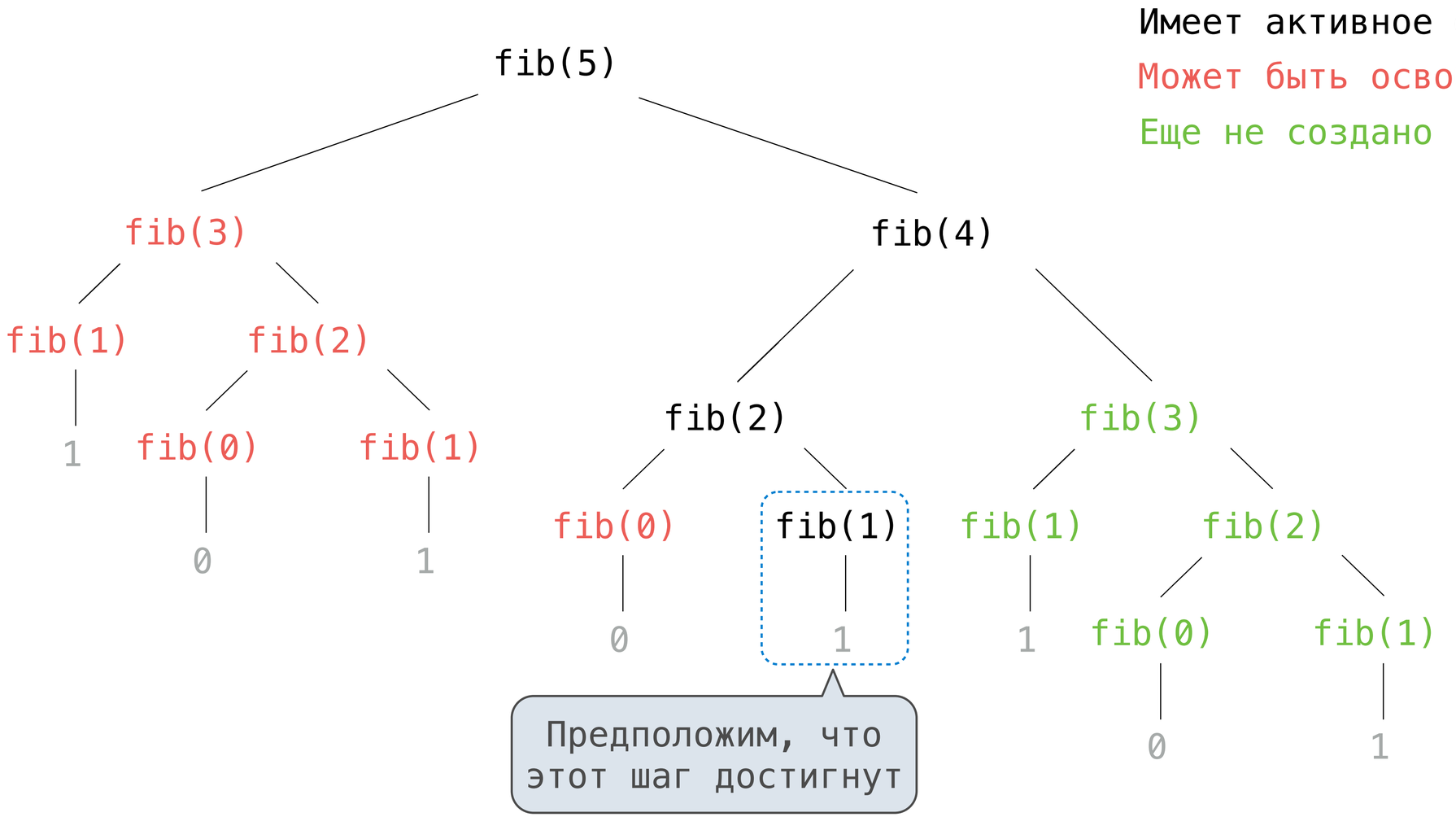
Имеет активное окружение



Пространственные затраты для последовательности Фибоначчи



Пространственные затраты для последовательности Фибоначчи



Время

Сравнение реализаций

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Время (количество делений)

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Время (количество делений)
n

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

	Время (количество делений)
Медленно: Проверяем каждый k от 1 до n	n
Быстро: Проверяем каждый k от 1 до корня из n Для любого k, n/k также множитель!	Наибольшее целое, меньшее $\sqrt{n}$

Сравнение реализаций

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

(Пример)

Время (количество делений)
n
Наибольшее целое, меньшее $\sqrt{n}$

Порядок роста

Порядки роста

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

означает, что существуют положительные константы k_1 и k_2 такие, что

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

означает, что существуют положительные константы $\mathbf{k_1}$ и $\mathbf{k_2}$ такие, что

$$k_1 \cdot f(n) \leq R(n) \leq k_2 \cdot f(n)$$

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

означает, что существуют положительные константы k_1 и k_2 такие, что

$$k_1 \cdot f(n) \leq R(n) \leq k_2 \cdot f(n)$$

для всех **n** больших некоторого минимума **m**

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

означает, что существуют положительные константы k_1 и k_2 такие, что

$$k_1 \cdot f(n) \leq R(n) \leq k_2 \cdot f(n)$$

для всех **n** больших некоторого минимума **m**

Порядки роста

Метод ограничения ресурсов, используемых функцией в терминах «размера» задачи

n: размер задачи

R(n): измерение некоторого использованного ресурса (время или пространство)

$$R(n) = \Theta(f(n))$$

означает, что существуют положительные константы k_1 и k_2 такие, что

$$k_1 \cdot f(n) \leq R(n) \leq k_2 \cdot f(n)$$

для всех **n** больших некоторого минимума **m**

Подсчет множителей

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

Для проверки *верхней границы:*

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

Для проверки *верхней границы:*

- Максимальное количество инструкций: $5 + 4\sqrt{n}$

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

Для проверки *верхней границы:*

- Максимальное количество инструкций: $5 + 4\sqrt{n}$
- Максимальное количество действий на инструкцию: некоторое **p**

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $\mathbf{k_1} = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

Для проверки *верхней границы:*

- Максимальное количество инструкций: $5 + 4\sqrt{n}$
- Максимальное количество действий на инструкцию: некоторое **p**

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Предположение: каждая инструкция (например инкремент, используя оператор +=), требует выполнения некоторого количества действий

Подсчет множителей

Количество действий, необходимых для подсчета множителей n, используя factors_fast: $\Theta(\sqrt{n})$

Для проверки *нижний границы*, выбираем $k_1 = 1$:

- Инструкции за пределами `while`: 4 или 5
- Инструкции внутри `while` (включая заголовок): 3 или 4
- Количество итераций `while`: между $\sqrt{n} - 1$ и $\sqrt{n}$
- Общее количество инструкций: по меньшей мере $4 + 3(\sqrt{n} - 1)$

Для проверки *верхней границы:*

- Максимальное количество инструкций: $5 + 4\sqrt{n}$
- Максимальное количество действий на инструкцию: некоторое p
- Выбираем $k_2 = 5p$ и $m = 25$

```
def factors_fast(n):
    sqrt_n = sqrt(n)
    k, total = 1, 0
    while k < sqrt_n:
        if divides(k, n):
            total += 2
        k += 1
    if k * k == n:
        total += 1
    return total
```

Предположение: каждая инструкция (например инкремент, используя оператор +=), требует выполнения некоторого количества действий

Порядок роста для подсчета множителей

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Время	Место

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

Медленно: Проверяем каждый k от 1 до n

Быстро: Проверяем каждый k от 1 до корня из n
Для любого k, n/k также множитель!

Время	**Место**
$\Theta(n)$	$\Theta(1)$

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

	Время	Место
Медленно: Проверяем каждый k от 1 до n	$\Theta(n)$	$\Theta(1)$
Быстро: Проверяем каждый k от 1 до корня из n Для любого k, n/k также множитель!	$\Theta(\sqrt{n})$	$\Theta(1)$

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

	Время	**Место**
Медленно: Проверяем каждый k от 1 до n	$\Theta(n)$	$\Theta(1)$
Быстро: Проверяем каждый k от 1 до корня из n Для любого k, n/k также множитель!	$\Theta(\sqrt{n})$	$\Theta(1)$

Предположение: целое занимает фиксированый размер

Порядок роста для подсчета множителей

Реализация одной и той же функциональной абстракции может требовать разного количества времени.

Задача: Сколько множителей у положительного n?

Множитель k числа n — это положительное целое, которое делит n нацело.

```
def factors(n):
```

	Время	**Место**
Медленно: Проверяем каждый k от 1 до n	$\Theta(n)$	$\Theta(1)$
Быстро: Проверяем каждый k от 1 до корня из n Для любого k, n/k также множитель!	$\Theta(\sqrt{n})$	$\Theta(1)$

Предположение: целое занимает фиксированый размер

(Пример)

Возведение в степень

Возведение в степень

Возведение в степень

Цель: возвести b в степень n

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
```

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ b \cdot b^{n-1} & \text{otherwise} \end{cases}$$

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ b \cdot b^{n-1} & \text{otherwise} \end{cases}$$

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ (b^{\frac{1}{2}n})^2 & \text{if } n \text{ is even} \\ b \cdot b^{n-1} & \text{if } n \text{ is odd} \end{cases}$$

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ b \cdot b^{n-1} & \text{otherwise} \end{cases}$$

```
def square(x):
    return x*x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n//2))
    else:
        return b * exp_fast(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ (b^{\frac{1}{2}n})^2 & \text{if } n \text{ is even} \\ b \cdot b^{n-1} & \text{if } n \text{ is odd} \end{cases}$$

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ b \cdot b^{n-1} & \text{otherwise} \end{cases}$$

```
def square(x):
    return x*x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n//2))
    else:
        return b * exp_fast(b, n-1)
```

$$b^n = \begin{cases} 1 & \text{if } n = 0 \\ (b^{\frac{1}{2}n})^2 & \text{if } n \text{ is even} \\ b \cdot b^{n-1} & \text{if } n \text{ is odd} \end{cases}$$

(Пример)

Возведение в степень

Цель: возвести b в степень n

Время	Место

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)


def square(x):
    return x*x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n//2))
    else:
        return b * exp_fast(b, n-1)
```

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)


def square(x):
    return x*x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n//2))
    else:
        return b * exp_fast(b, n-1)
```

Время	Место
$\Theta(n)$	$\Theta(n)$

Возведение в степень

Цель: возвести b в степень n

```
def exp(b, n):
    if n == 0:
        return 1
    else:
        return b * exp(b, n-1)


def square(x):
    return x*x

def exp_fast(b, n):
    if n == 0:
        return 1
    elif n % 2 == 0:
        return square(exp_fast(b, n//2))
    else:
        return b * exp_fast(b, n-1)
```

Время	Место
$\Theta(n)$	$\Theta(n)$
$\Theta(\log n)$	$\Theta(\log n)$

Сравнение порядков роста

Свойства порядков роста

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Если a и b оба имеют длину **n**, то overlap потребует $\Theta(n^2)$ шагов

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Если a и b оба имеют длину **n**, то overlap потребует $\Theta(n^2)$ шагов

Члены низкого порядка: Быстрорастущая часть вычисления в итоге доминирует

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Если a и b оба имеют длину **n**, то overlap потребует $\Theta(n^2)$ шагов

Члены низкого порядка: Быстрорастущая часть вычисления в итоге доминирует

$$\Theta(n^2)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Если a и b оба имеют длину **n**, то overlap потребует $\Theta(n^2)$ шагов

Члены низкого порядка: Быстрорастущая часть вычисления в итоге доминирует

$$\Theta(n^2) \qquad \Theta(n^2 + n)$$

Свойства порядков роста

Константы: множители не влияют на порядок роста процесса

$$\Theta(n) \qquad \Theta(500 \cdot n) \qquad \Theta(\frac{1}{500} \cdot n)$$

Логарифмы: основания логарифма не влияют на порядок роста процесса

$$\Theta(\log_2 n) \qquad \Theta(\log_{10} n) \qquad \Theta(\ln n)$$

Вложения: когда внутренний процесс повторяется на каждом шаге внешнего процесса, общее число шагов равно произведению количеств шагов внешнего и внутреннего процессов

```
def overlap(a, b):
    count = 0
    for item in a:
        if item in b:
            count += 1
    return count
```

Внешний: длина a

Внутренний: длина b

Если a и b оба имеют длину **n**, то overlap потребует $\Theta(n^2)$ шагов

Члены низкого порядка: Быстрорастущая часть вычисления в итоге доминирует

$$\Theta(n^2) \qquad \Theta(n^2 + n) \qquad \Theta(n^2 + 500 \cdot n + \log_2 n + 1000)$$

Сравнение порядков роста (n — размер задачи)

Сравнение порядков роста (n — размер задачи)

$$\Theta(b^n)$$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \dfrac{1+\sqrt{5}}{2} \approx 1.61828$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Удвоение размера задачи изменяет R(n) линейно.

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Удвоение размера задачи изменяет R(n) линейно.

$\Theta(1)$

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Удвоение размера задачи изменяет R(n) линейно.

$\Theta(1)$ Константный рост. Размер задачи не имеет значения.

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Удвоение размера задачи изменяет R(n) линейно.

$\Theta(1)$ Константный рост. Размер задачи не имеет значения.

Сравнение порядков роста (n — размер задачи)

$\Theta(b^n)$ Экспоненциальный рост. Рекурсивный `fib`

$\Theta(\phi^n)$ шагов, где $\phi = \frac{1+\sqrt{5}}{2} \approx 1.61828$

Увеличение размера задачи умножает R(n) на множитель

$\Theta(n^2)$ Квадратичный рост. Например, `overlap`

Увеличение n увеличивает R(n) на размер задачи n

$\Theta(n)$ Линейный рост. Например, медленный `factors` или `exp`

$\Theta(\sqrt{n})$ Рост корень из n. Например, `factors_fast`

$\Theta(\log n)$ Логарифмический рост. Например, `exp_fast`

Удвоение размера задачи изменяет R(n) линейно.

$\Theta(1)$ Константный рост. Размер задачи не имеет значения.