

Лекция 4
Рекурсия.
Рекурсивные алгоритмы.



Декомпозиция алгоритмов. Рекурсия.

Хорошо известно, что перед тем, чтобы начать решать задачу, надо разбить ее на подзадачи. Это действие получило название **декомпозиции**. Декомпозиция алгоритма на частные алгоритмы и модули проводится для сокращения сроков создания программ и упрощения проверки работоспособности.

Рекурсия – вызов функции из нее же самой. Пример: вычисление факториала:

Любой рекурсивный алгоритм может быть реализован через цикл for.

Но это не всегда удобно.

```
def fac(n):  
    if n == 1:  
        return 1  
    return fac(n - 1) * n  
  
print(fac(5))
```

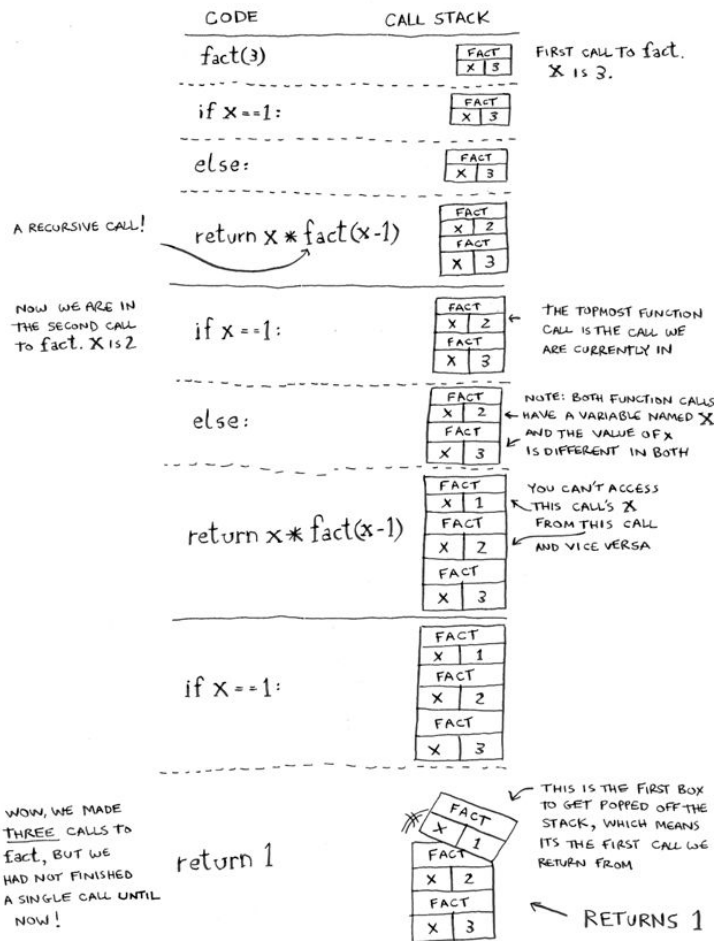
Декомпозиция + рекурсия приводят к принципу “разделяй и властвуй” (divide and conquer) – класс алгоритмов, в которых мы

- 1) Рекурсивно разбиваем решение исходной задачи на решение более простых подзадач, как правило пока они не станут элементарными
- 2) Собираем из решения подзадач решение исходной задачи

Стек вызовов. Прямой и обратный ход рекурсии

Когда функция производит вложенный (рекурсивный) вызов функции, происходит следующее:

- Выполнение текущей функции приостанавливается.
- Информация, связанная с вызовом предыдущей функции, запоминается в специальной структуре данных – стеке вызовов.
- Выполняются вложенные вызовы, для каждого из которых создается свой scope.
- Выполнение возвращается в scope исходной функции после выполнения всех вложенных вызовов



Простые рекурсивные алгоритмы

Алгоритм Евклида – поиск НОД двух чисел

1. Большее число делим на меньшее.
2. Если делится без остатка, то меньшее число и есть НОД (следует выйти из цикла).
3. Если есть остаток, то большее число заменяем на остаток от деления.
4. Переходим к пункту 1.

```
a = int(input())
b = int(input())

while a != 0 and b != 0:
    if a > b:
        a = a % b
    else:
        b = b % a

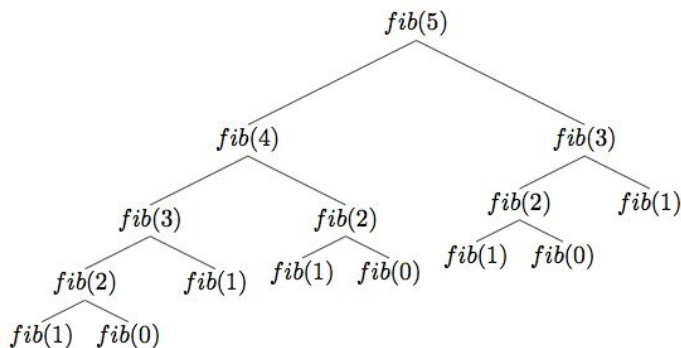
print(a + b)
```

Подсчет чисел Фибоначчи:

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

Еще немного про рекурсию

Когда мы считали числа Фибоначчи через рекурсию, на каждом шаге рекурсии мы делали два рекурсивных вызова



очевидно, это неэффективно – некоторые значения мы считываем по несколько раз

Чтобы оптимизировать алгоритм, давайте хранить список fibs уже посчитанных чисел, и передавать его во все функции, которые рекурсивно вызываем. Это называется рекурсия с кешированием (или мемоизацией)

```
def fib(N, fibs):  
    if N == 1:  
        fibs[0] = 0  
        return fibs[0]  
    if N == 2:  
        fibs[1] = 1  
        return fibs[1]  
  
    if fibs[N-1] != 0:  
        return fibs[N-1]  
  
    fibs[N-1] = fib(N - 1, fibs) + fib(N - 2, fibs)  
    print(fibs)  
    return fibs[N-1]
```

Динамическое программирование

Решаем задачу не “с конца”, а конструируем решение “с начала”

- Состояние динамики - подзадачи, к которым мы можем свести исходную задачу;
- Переход - правило пересчета, то есть способ вычислить ответ на задачу с помощью ответов на подзадачи;
- База динамики - набор тривиальных состояний и значений для них.

```
N = int(input())
fibs = [0 for i in range(N)]

fibs[0] = 0
fibs[1] = 1

for i in range(2, N):
    fibs[i] = fibs[i-1] + fibs[i-2]

print(fibs)
```

Задача о кузнечике

Пусть кузнечик прыгает на одну или две точки на координатной прямой вперед, а за прыжок в каждую точку необходимо заплатить определенную стоимость, различную для различных точек. Стоимость прыжка в точку i задается значением **price[i]** списка **price**. Необходимо найти минимальную стоимость маршрута кузнечика из точки **0** в точку **n**.

тогда итоговая минимальная стоимость прибытия в точку под номером i будет:

$$\mathbf{dp[i] = \min(dp[i-1], dp[i-2]) + price[i]}$$

Нам нужно заполнить этот список до позиции n , ответ будет лежать в **dp[n]**.

Задачу также можно расширить, введя отрицательные стоимости в **price**, тогда аналогичным способом можно решать задачу на максимум.

Если хранить в **dp[i]** не только стоимость, но и оптимальный путь, по которому мы приходим в точку i , то в **dp[n]** также получим оптимальный путь минимальной/максимальной стоимости. Это **задача с восстановлением пути**.

Алгоритм решения задач на ДП

- Сформулировать, что будет значить **состояние**. Пример **dp[i]** - максимальное число монет, которое можно собрать, дойдя до i
- Определить формулу (формулы) пересчета динамики;
- Определить **порядок**, в котором будут считаться состояния динамики. Например, в данном случае нам надо было перебирать от 0 до $n-1$ но в других задачах (например, в двумерной динамике) этот порядок может быть менее тривиальным.
- Задать значения для тривиальных состояний;
- Понять, какое состояние соответствует ответу на всю задачу.

Решето Эратосфена

Алгоритм поиска простых чисел.

1. Все четные числа, кроме двойки, - составные, т. е. не являются простыми, так как делятся не только на себя и единицу, а также еще на 2.
2. Все числа кратные трем, кроме самой тройки, - составные, так как делятся не только на самих себя и единицу, а также еще на 3.
3. Число 4 уже выбыло из игры, так как делится на 2.
4. Число 5 простое, так как его не делит ни один простой делитель, стоящий до него.
5. Если число не делится ни на одно простое число, стоящее до него, значит оно не будет делиться ни на одно сложное число, стоящее до него.

```
# Создается список из значений от 0 до N включительно
primes = [i for i in range(N + 1)]

# Вторым элементом списка является единица, которую
# не считают простым числом. Забиваем ее нулем
primes[1] = 0

# Начинаем с 3-го элемента
i = 2
while i <= N:
    # Если значение текущей ячейки до этого не было обнулено,
    # значит в этой ячейке содержится простое число
    if primes[i] != 0:
        # Первое кратное ему будет в два раза больше
        j = i + i
        while j <= N:
            # и это число составное,
            # поэтому заменяем его нулем
            primes[j] = 0
            # переходим к следующему числу,
            # которое кратно i (оно на i больше)
            j = j + i
        i += 1

# Избавляемся от всех нулей в списке
primes = [i for i in primes if i != 0]
print(primes)
```

Интересные ссылки

<https://habr.com/ru/articles/207988/>

<https://habr.com/ru/news/756266/>