Задание №5, Анализ и построение алгоритмов

Теория

Данное задание можно решить как аналитически (руками), так и программированием, написав алгоритм из условия.

> 💡 Часто встречающиеся в этом задании функции и методы Python:
>
> **int(n,m)** - перевод числа **n** (в строчном виде) из системы счисления с основанием **m** в десятичную.
>
> **bin(x)[2:]** - перевод числа **x** в двоичную систему счисления, через срез убираем префикс *0b*. Возвращает **строку.**
>
> **f'{x:b}'** - ещё один способ перевести x в двоичный вид, без префикса, тоже строка.
>
> **s[start:stop]** - берёт срез от элемента с индексом **start** (включительно) до элемента с индексом **stop** (не включительно).
>
> **sum(map(int,s))** - нахождение суммы цифр для строки **s**: sum возвращает сумму элементов списка (элементы должны являются числами), а map применяет функцию **int** к каждому символу строки **s.** Аналогично можно записать через генератор списка: **sum(int(c) for c in s)**
>
> **s.count(x)** - возвращает количество элементов **x** в строке/списке **s** (следите за типами)

> Полезное свойство: если какое-то число делится без остатка на **x**, то, если перевести это число в систему счисления с базисом **x**, то оно будет оканчивается на 0.

План решения

8BD535, аналог досрока 2025

На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится двоичная запись числа *N*.

2. Далее эта запись обрабатывается по следующему правилу:

а) если сумма цифр в двоичной записи числа чётная, то к этой записи справа дописывается 0, а затем два левых разряда заменяются на 10;

б) если сумма цифр в двоичной записи числа нечётная, то к этой записи справа дописывается 1, а затем два левых разряда заменяются на 11.

Полученная таким образом запись является двоичной записью искомого числа *R*.

3. Результат переводится в десятичную систему и выводится на экран.

Например, для исходного числа $6_{10} = 110_2$ результатом является число $1000_2 = 8_{10}$, а для исходного числа $4_{10} = 100_2$ это число $1101_2 = 13_{10}$.

Укажите **минимальное** число *N*, после обработки которого с помощью этого алгоритма получается число *R*, большее 19. В ответе запишите это число в десятичной системе счисления.

1. Объявим функцию R(n), которая будет применять описанный в условии алгоритм к n и возвращать получившееся число:

   ```
   def R(n):
   ```

2. Выполним пункт 1, переведём число в нужную систему счисления. Если требуется двоичный вид, то переводим число через встроенную в Python функцию **bin** (также есть ещё **oct** для 8-рич и **hex** для 16-рич), не забыв убрать префикс срезом [2:]. Для других систем счисления - вспоминаем 14 задание и пишем функцию перевода.

   ```
       n = bin(n)[2:]
   ```

3. Выполняем 2 пункт. Тут проверяется ваше умение писать код под нужные условия, поэтому важно нарешать много задач. Часто можно заметить некоторые полезные свойства, облегчающие написание кода: т.к. сумма цифр в двоичной записи равна количеству единиц в ней, то представим её как **n.count('1')**. Замену разрядов запишем через срезы: n[2:] - это n без двух левых разрядов. К этим срезам добавляем нужные разряды. Во втором условии обычно используем **elif** вместо **if**, т.к. число после операций первого **if** может стать подходящим и для второго, тогда число преобразуется дважды, чего быть не должно:

```
    if n.count('1') % 2 == 0:
        n += '0'
        n = '10' + n[2:]
    elif n.count('1') % 2 != 0:
        n += '1'
        n = '11' + n[2:]
```

4. Выполняем 3 пункт, возвращаем число в десятичном виде:

```
        return int(b,2)
```

5. Проверяем правильность алгоритма числами из примера. Для 6 результат должен быть 8, а для 4 - 13. Если все правильно, переходим к следующему шагу.

```
print(R(6), R(4))
```

6. Чтобы найти минимальное число N, для которого R(N) будет больше 19, просто перебираем значения N циклом и при первом же удовлетворительном R(N) останавливаем цикл через break, т.к. первое же подходящее N и будет минимальным, если писать range по возрастанию. Для нахождения максимального N можно просто сделать range убывающим: range(1000, 1, -1)

```
for n in range(1, 1000):
    if R(n) > 19:
        print(n)
        break
```

7. Получаем ответ **8**. Весь код:

```
def R(n):
    b = bin(n)[2:]
    if b.count('1') % 2 == 0:
        b += '0'
```

```
        b = '10' + b[2:]
    elif b.count('1') % 2 != 0:
        b += '1'
        b = '11' + b[2:]
    return int(b,2)
print(R(6), R(4))
for n in range(1, 1000):
    if R(n) > 19:
        print(n)
        break
```

❗ Не забывайте, что после приведения числа к не десятичной системе счисления оно будет иметь строчный вид, к которому не применимы арифметические операции (в т.ч. нахождение остатка от деления %), нужно явно привести тип к int. Это же касается сравнений: сравнение одного и то же числа в строчном и численном виде будет всегда возвращать False (‘0’ == 0 ⇒ False).

Алгоритм решения

0D73A6

На вход алгоритма подаётся натуральное число *N*. Алгоритм строит по нему новое число *R* следующим образом.

1. Строится троичная запись числа *N*.

2. Далее эта запись обрабатывается по следующему правилу:

а) если число *N* делится на 3, то к этой записи дописываются две последние троичные цифры;

б) если число *N* на 3 не делится, то остаток от деления умножается на 5, переводится в троичную запись и дописывается в конец числа.

Полученная таким образом запись является троичной записью искомого числа *R*.

3. Результат переводится в десятичную систему и выводится на экран.

Например, для исходного числа $11 = 102_3$ результатом является число $102101_3 = 307$, а для исходного числа $12 = 110_3$ это число $11010_3 = 111$.

Укажите максимальное число *N*, после обработки которого с помощью этого алгоритма получается число *R*, меньшее 159.

1. Объявляем функции **R(n)**, которая будет применять алгоритм из условия к числу, и **convert_to_trinary(n)**, которая будет переводить число в троичный вид. Объяснение алгоритма перевода в другую систему счисления смотрите в методичке по 14 заданию. Выполняем 1 пункт.

```
def convert_to_trinary(n):
    res = ''
    while n > 0:
        res += str(n%3)
        n //= 3
    return res[::-1]

def R(n):
    n = convert_to_trinary(n)
```

2. Троичный вид числа представлен строкой, поэтому напрямую проверить его делимость через **%** нельзя. Для второго пункта из условия можно использовать три подхода:

    a. Если троичный вид записать в другую переменную, например **t**, проверять на делимость можно исходное не преобразованное **n** (но все последующие операции проводить с **t**):

```
t = convert_to_trinary(n)
if n % 3 == 0:
```

b. Можно перевести троичное число в десятичный вид и уже его проверять на делимость:

```
n = convert_to_trinary(n)
if int(n,3) % 3  == 0:
```

c. Можно воспользоваться свойством систем счисления: число делится на базис только тогда, когда крайний правый разряд равен 0.

```
n = convert_to_trinary(n)
if n[-1]  == '0':
```

3. В этом решении будет использоваться последний вариант. Две последние троичные цифры берём через срез:

```
if n[-1]  == '0':
   n += n[-2:] # или n += n[-2] + n[-1]
```

4. Т.к. число не может одновременно делиться и не делиться на 3, то для второго условия просто используем **else.** Для операции перевожу число в десятичный вид, чтобы корректно выполнялось умножение и взятие остатка:

```
else:
   n += convert_to_trinary(int(n, 3) % 3 * 5)
```

5. Возвращаем результат в десятичном виде. Проверяем, что числа из примера возвращают нужный результат.

```
    return int(n,3)
print(R(11) ==  307, R(12) == 111)
```

6. Если все правильно, перебираем циклом различные n, чтобы найти подходящий. Т.к. n нужен максимальный, range пишем в обратном порядке:

```
for n in range(10000, 1, -1):
    if R(n) < 159:
        print(n)
        break
```

7. Получаем ответ **16.** Весь код:

```
def convert_to_trinary(n):
    res = ''
    while n > 0:
        res += str(n%3)
        n //= 3
    return res[::-1]

def R(n):
    n = convert_to_trinary(n)
    if n[-1]  == '0':
        n += n[-2:]
    else:
        n += convert_to_trinary(int(n, 3) % 3 * 5)
    return int(n,3)

print(R(11) ==  307, R(12) == 111)
for n in range(10000, 1, -1):
    if R(n) < 159:
        print(n)
        break
```