Конспект 3

Strim_23_3

Задача № 1 (2761)

Исполнитель Калькулятор преобразует число на экране. У исполнителя есть три команды, которым присвоены номера:

1. Прибавить 1
2. Прибавить 2
3. Умножить на 2

Программа для исполнителя Калькулятор – это последовательность команд. Сколько существует программ, состоящих из 6 команд, для которых при исходном числе 1 результатом является число 20?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=0h2m20s

Решение

Для решения задачи о количестве программ, состоящих из шести команд, которые преобразуют число 1 в число 20, используя три операции (прибавление 1, прибавление 2 и умножение на 2), важно учитывать ограничение на количество шагов. Поэтому необходимо отслеживать, сколько шагов уже совершено исполнителем.
Мы можем воспользоваться рекурсией, чтобы перебрать возможные последовательности операций. Однако стандартная рекурсия не учитывает количество выполненных шагов. Чтобы решить эту проблему, добавим в нашу рекурсивную функцию дополнительный параметр step, который будет хранить количество сделанных шагов.

Функция, реализующая такое решение:

```
def f(curr, end, step):
    # Если текущее число превышает целевое, возвращаем 0
    if curr > end:
        return 0

    # Если текущее число достигло целевого значения,
    # проверяем, совпадает ли количество шагов с требуемым (6)
    if curr == end:
        return step == 6

    # Если текущее число меньше целевого, продолжаем рекурсию
    if curr < end:
        # Для каждой возможной операции увеличиваем счетчик шагов на 1
        return (
            f(curr + 1, end, step + 1) +
            f(curr + 2, end, step + 1) +
            f(curr * 2, end, step + 1)
```

```
        )
```

Параметр step играет ключевую роль в решении данной задачи, так как именно он позволяет отслеживать количество шагов, которое совершает исполнитель. В начале работы программы значение этого параметра равно нулю (step=0), поскольку изначально никакие шаги ещё не сделаны. При каждом рекурсивном вызове функции f происходит увеличение значения step на 1. Это отражает тот факт, что каждая операция (будь то прибавление 1, прибавление 2 или умножение на 2) считается одним шагом. Например, если на текущем этапе значение curr равно 4, и мы решаем применить операцию «прибавить 1», то следующее состояние функции будет выглядеть так:

```
f(curr + 1, end, step + 1)
```

Здесь step + 1 указывает, что текущий шаг выполнен, и теперь общее количество шагов увеличилось на 1.
Когда текущее число достигает целевого значения, curr == end, функция проверяет, соответствует ли количество сделанных шагов заданному ограничению, в данном случае 6. Если условие выполняется step == 6, возвращается 1, что говорит о том, что найден правильный путь преобразования числа. Если же количество шагов отличается от 6, то возвращается 0, так как такая последовательность шагов не удовлетворяет условиям задачи.
Таким образом, параметр step служит своеобразным счётчиком шагов, позволяя программе отслеживать выполнение всех возможных комбинаций операций и отбрасывать те, которые не укладываются в лимит шагов.

Вызываем функцию с начальными значениями:

```
def f(curr,end,step):
    if curr>end: return 0
    if curr==end: return step==6
    if curr<end: return f(curr+1,end,step+1)+f(curr+2,end,step+1)+f(curr*2,end,step+1)

print(f(1,20,0))
```

Запустив этот код, мы получим ответ 36, что означает, что существует 36 различных последовательностей из шести команд, которые приводят к числу 20, начиная с числа 1.

Ответ: 36

Задача №2 (5786)

Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:
1. Прибавь 1
2. Умножь на 2
3. Вычти 3

Выполняя первую из них, исполнитель увеличивает число на экране на 1, выполняя вторую – умножает на 2, выполняя третью – уменьшает на 3. Программой для исполнителя называется последовательность команд. Сколько существует программ длиной не более 7 команд, которые преобразуют число 1 в число 10?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=0h13m20s

Решение

Особенность этой задачи состоит в том, что команды могут выполняться не только в положительном направлении, увеличивая число, но и в обратном, уменьшая его. Это означает, что можно сначала увеличить число выше требуемого значения, а затем уменьшить его обратно до нужного результата. Например, можно получить 16, а затем дважды применить команду «вычесть 3», чтобы достичь 10. Из-за возможности возврата назад простое условие "если текущее число превышает нужное" перестает работать, поскольку всегда остается шанс вернуться обратно. Вместо этого будем использовать глубину поиска (количество выполненных команд) как критерий завершения. Если количество команд превышает 7, прекращаем дальнейший поиск, так как считаем его неэффективным. Таким образом, условие остановки теперь звучит так: если количество шагов больше 7, то прерываем выполнение и возвращаем 0. Если же мы достигли целевого значения, то возвратим 1, так как условие глубины гарантирует, что шаги не превысят 7.
Если текущее число меньше целевого, мы продолжаем вычисления, пока количество шагов не достигнет 7. Важно отметить, что здесь мы не ограничиваемся только натуральными числами; можно получать и отрицательные значения, что вполне допустимо согласно условиям задачи.
Для реализации этого подхода создаётся рекурсивная функция f, которая принимает начальное число c, целевое число e и текущий шаг step. Функция возвращает количество возможных программ для достижения цели.

```
def f(c, e, step):
    # Если количество шагов превышает 7, возвращаем 0
    if step > 7:
        return 0

    # Если достигли целевой цифры, возвращаем 1 + результаты дальнейших действий
    if c == e:
        return 1 + f(c + 1, e, step + 1) + f(c * 2, e, step + 1) + f(c - 3, e, step +
1)

    # Продолжаем поиск, если количество шагов не превышает 7
    if step <= 7:
        return f(c + 1, e, step + 1) + f(c * 2, e, step + 1) + f(c - 3, e, step + 1)

# Запуск функции с начальными значениями
print(f(1, 10, 0))
```

Этот код учитывает возможность выполнения команд, превышающих целевую цифру, и завершает работу, когда достигнуто максимальное количество шагов.

Ответ: 38

Задача № 3 (3449)

У исполнителя Калькулятор есть три команды, которым присвоены номера:
1. Прибавить 1
2. Прибавить 5

3. Умножить на 3
   Сколько разных чисел на отрезке [1000, 1024] может быть получено из числа 1 с помощью программ, состоящих из 8 команд?

   Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=0h25m0s

   Решение

   Для начала создадим функцию, которая будет вычислять количество возможных путей из числа c в число e, используя ровно восемь команд. Функция будет рекурсивной и учитывать ограничения по количеству шагов.
   Затем используем эту функцию для подсчета количества чисел на заданном отрезке, которые могут быть получены из числа 1 с помощью программы длиной в восемь команд.

```
def f(c, e, s):
    # Если текущее значение превышает целевое или превышено максимальное количество
шагов,
    # возвращаем 0, так как дальнейшие пути невозможны
    if c > e or s > 8:
        return 0

    # Если достигли целевой точки точно за 8 шагов, возвращаем 1 (успешная траектория)
    if c == e:
        return s == 8

    # Иначе продолжаем рекурсию, прибавляя 1, 5 или умножая на 3
    if c < e:
        return (
            f(c + 1, e, s + 1) +
            f(c + 5, e, s + 1) +
            f(c * 3, e, s + 1)
        )
# Счетчик для хранения количества подходящих чисел
k = 0

# Перебираем каждое число на отрезке [1000, 1024]
for x in range(1000, 1025):
    # Проверяем, существует ли хотя бы одна программа из 8 команд, приводящая к числу
x
    if f(1, x, 0) > 0:
        k += 1

# Выводим итоговый результат
print(k)
```

   Теперь используем эту функцию для подсчета количества чисел на заданном отрезке, которые могут быть получены из числа 1 с помощью программы длиной в восемь команд.

   Ответ: 1

Задача № 4 (3446)

У исполнителя Калькулятор есть три команды, которым присвоены номера:

1. Прибавить 1
2. Прибавить 5
3. Умножить на 3

Сколько разных чисел может быть получено из числа 1 с помощью программ, состоящих из 4 команд?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=0h34m10s

Решение

Рассмотрим два способа решения этой задачи. Первый способ заключается в использовании функции, которая подсчитывает количество возможных путей длиной ровно 4. После этого программа перебирает все возможные значения от 2 до 999 и считает, сколько раз результат функции равен 1. Это даёт общее количество уникальных чисел, которые могут быть получены за 4 команды.

```
# Функция f выполняет проверку возможности получения числа e из числа c с помощью программы длиной s
def f(c, e, s):
    # Если превышено максимальное количество команд (s > 4) или текущее число больше целевого (c > e), возвращаем 0
    if c > e or s > 4:
        return 0
    # Если текущее число совпадает с целевым и использовано ровно 4 команды, возвращаем 1
    if c == e and s == 4:
        return 1
    # Если текущее число меньше целевого и количество команд еще не достигло максимума, продолжаем рекурсию
    if c < e and s < 4:
        # Рассматриваем три варианта: прибавляем 1, прибавляем 5, умножаем на 3
        return (
            f(c + 1, e, s + 1) +
            f(c + 5, e, s + 1) +
            f(c * 3, e, s + 1)
        )
# Счетчик уникальных чисел
k = 0
# Перебираем все возможные целевые числа от 2 до 999
for x in range(2, 1000):
    # Проверяем, возможно ли получить число x из 1 с помощью программы длиной 4
    if f(1, x, 0) > 0:
        # Если возможно, увеличиваем счетчик
        k += 1
# Выводим итоговый результат
print(k)
```

Недостатками этого подхода являются:

· Перебор большого количества значений, что приводит к высокой вычислительной сложности.

· Повторный расчёт одних и тех же подзадач.

Второй способ. Здесь создается функция, которая вместо проверки конкретного целевого числа, добавляет каждое полученное число в множество d, т.е. когда количество выполненных команд достигает 4, текущее число добавляется в множество. После завершения работы функции выводится длина множества d, которая равна количеству уникальных чисел, полученных за 4 команды.

Преимущества этого способа:

1. Использование множества позволяет автоматически учитывать уникальность чисел. Множество хранит только уникальные элементы, поэтому нет необходимости проверять каждую комбинацию отдельно.
2. Эффективность. Поскольку множество исключает повторяющиеся числа, вычисления происходят быстрее, чем при полном переборе всех возможных значений.
3. Отсутствие избыточных проверок. Рекурсивная функция просто добавляет новые числа в множество, а затем завершает работу после выполнения четырех команд. Это делает решение проще и понятнее.

```
# Создаем множество для хранения уникальных чисел
d = set()
# Функция f генерирует все возможные числа, которые можно получить из числа c с помощью программы длиной s
def f(c, s):
    # Если превышено максимальное количество команд (s > 4), возвращаем 0
    if s > 4:
        return 0
    # Если использовалось ровно 4 команды, добавляем текущее число в множество и возвращаем 1
    if s == 4:
        d.add(c)
        return 1
    # Если количество команд еще не достигло максимума, продолжаем рекурсию
    if s < 4:
        # Рассматриваем три варианта: прибавляем 1, прибавляем 5, умножаем на 3
        return (
            f(c + 1, s + 1) +
            f(c + 5, s + 1) +
            f(c * 3, s + 1)
        )
# Запускаем функцию с начальным числом 1 и нулевой программой
f(1, 0)
# Выводим количество уникальных чисел, которые можно получить
print(len(d))
```

Ответ: 43

Примечание Джобса: можно описать функцию, которая будет возвращать целевое множество.

```
def f(c, s):
```

```
  if s == 4:
    return {c}
  return f(c+1, s+1) | f(c+5, s+1) | f(c*3, s+1)

print(len(f(1,0)))
```

Задача № 5 (3443)

У исполнителя Калькулятор есть три команды, которым присвоены номера:

1. Прибавить 1
2. Прибавить 5
3. Умножить на 3

Найдите длину самой короткой программы, в результате выполнения которой при исходном числе 1 результатом является число 227.

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h3m5s

Решение

В этой задаче нужно определить наименьшее количество шагов, необходимых для достижения результата.
В отличие от задачи поиска всех возможных путей преобразования, наша задача состоит именно в том, чтобы найти самый короткий путь. То есть, если существует несколько программ, ведущих к числу 227, нам нужна та, которая содержит наименьшее количество команд.
Мы можем решить эту задачу методом полного перебора. Для этого создадим рекурсивную функцию f, которая будет проверять, возможно ли достичь числа 227 за определенное количество шагов.

```
def f(c, e, s):
    # Если текущее число превышает целевое или количество шагов превысило допустимое, возвращаем 0
    if c > e or s > ml:
        return 0

    # Если достигли цели, проверяем, соответствует ли количество шагов текущему значению ml
    if c == e:
        return s == ml

    # Продолжаем перебор, прибавляя 1, 5 или умножая на 3
    if c < e:
        return f(c + 1, e, s + 1) + f(c + 5, e, s + 1) + f(c * 3, e, s + 1)
```

Теперь, используя цикл, будем увеличивать максимальное количество шагов (ml) и проверять, достигнуто ли число 227 за данное количество шагов. Как только найдем первую траекторию, выведем результат.
Полный код программы:

```
def f(c,e,s):
    if c>e or s>ml: return 0
    if c==e: return s==ml
```

```
    if c<e: return f(c+1,e,s+1)+f(c+5,e,s+1)+f(c*3,e,s+1)
# Перебираем возможные значения максимальной длины программы
for ml in range(2, 20):
    if f(1, 227, 0) > 0:
        print(ml)
```

Результат работы программы:

```
7
8
9
10
11
12
13
14
15
16
17
18
19
```

Ответ: 7

Примечание Джобса: можно написать функцию, которая будет возвращать нужное значение, схожим образом выставив ограничение на количество шагов.

```
def f(a, b, s):
  if a == b:
    return s
  if a > b or s > 20:
    return float('inf')
  return min(f(a+1, b, s+1),
             f(a+5, b, s+1),
             f(a*3, b, s+1))

print(f(1, 227, 0))
```

Или с помощью цикла «ограничителя».

```
def f(a, b, s):
  if a == b:
    return s
  if a > b or s < 1:
    return 1
  return min(f(a+1, b, s-1),
             f(a+5, b, s-1),
             f(a*3, b, s-1))

for steps in range(1, 20):
  if f(1, 227, steps) == 0:
```

```
    print(steps)
```

Задача №6 (3444)

У исполнителя Калькулятор есть три команды, которым присвоены номера:

1. Прибавить 1
2. Прибавить 2
3. Умножить на 2

Сколько существует программ минимальной длины, в результате выполнения которых при исходном числе 1 результатом является число 28?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h10m40s

Решение

Для начала нужно понять, что нам требуется не просто общее количество возможных программ, ведущих от 1 к 28, но именно минимальное количество шагов, необходимых для достижения этого результата. В этом контексте под шагом понимается применение одной из трех команд.
Чтобы решить задачу, воспользуемся методом динамического программирования. Будем рекурсивно вычислять количество способов достижения числа 28 через различные комбинации команд, отслеживая текущую длину программы.
Обозначим функцию f(c, e, s) следующим образом:

- c – текущее число,
- e – целевое число (в нашем случае 28),
- s – текущая длина программы.

Наша функция будет работать так:

1. Если текущее число c превышает целевое число e, или если текущая длина программы s превысила максимальную допустимую длину (ms), то возвращаем 0 (таких путей нет).
2. Если текущее число c достигло целевого числа e, проверяем, совпадает ли текущая длина программы с максимальной длиной (ms). Если совпала, значит, найден путь нужной длины, иначе – нет.
3. Если текущее число еще меньше целевого, продолжаем исследовать возможные пути, применяя каждую из трех команд и увеличивая счетчик шагов на единицу.

Таким образом, рекурсия продолжается до тех пор, пока не будут найдены все пути минимальной длины.

```
def f(c, e, s):
    # Если текущее число больше целевого или превышена максимальная длина программы
    if c > e or s > ms:
        return 0
    # Если достигли цели и текущая длина равна максимальной длине программы
    if c == e:
        return s == ms
    # Продолжаем исследовать пути
    if c < e:
        return
            f(c + 1, e, s + 1) + f(c + 2, e, s + 1) + f(c * 2, e, s + 1)
# Перебор различных значений максимальной длины программы
```

```
for ms in range(2, 10):
    if f(1, 28, 0) > 0:
        print (ms, f(1, 28, 0))
```

Результат работы программы:

```
5 3
6 27
7 67
8 198
9 505
```

Из списка полученных ответов выбираем значение равное 3 программам минимальной длины, состоящим из 5 шагов.

Ответ: 3

Задание №7 (4948)

Исполнитель Калькулятор преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:

1. Прибавь 1
2. Прибавь 2
3. Умножь на 2

Первая команда увеличивает число на экране на 1, вторая увеличивает его на 2, третья – умножает на 2. Программа для исполнителя – это последовательность команд. Сколько существует программ, которые преобразуют исходное число 1 в число 15 и при этом не содержат двух команд умножения подряд?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h17m10s

Решение

Основная идея для решения задачи заключается в том, что каждая программа представляет собой последовательность команд, где каждая следующая команда может либо увеличить текущее значение на 1, либо увеличить его на 2, либо удвоить его. Однако важно учитывать ограничение, что две команды умножения не могут идти друг за другом. Чтобы учесть предыдущее состояние, необходимо ввести дополнительный параметр в рекурсию, который будет хранить информацию о предыдущей выполненной команде. Этот параметр позволяет избежать последовательного применения двух команд умножения.

Рассмотрим решение пошагово:

1. Рекурсивная функция принимает три аргумента: текущее число c, целевое число e и информация о последней выполненной операции p.

o Если текущее число больше целевого (c > e), то возвращаем 0, поскольку такой путь невозможен.

o Если текущее число равно целевому (c == e), то возвращаем 1, так как найден допустимый путь.

o Если текущее число меньше целевого (c < e) и последняя операция не была умножением (p != '*2'), тогда добавляем к результату все возможные пути: увеличение числа на 1, увеличение на 2 и удвоение.

o Если текущая операция была умножением (p == '*2'), то исключаем возможность следующего умножения, оставляя только варианты увеличения на 1 и на 2.
2. Инициализация: Начинаем с пустого значения предыдущего хода, так как в начале программы еще нет выполненных операций.
Таким образом, код выглядит следующим образом:

```
def f(c, e, p):
    # Если текущее число превышает целевое, то таких путей нет
    if c > e:
        return 0
    # Если достигли цели, значит нашли один возможный путь
    if c == e:
        return 1
    # Если текущее число меньше целевого и последняя операция не была умножением
    if c < e and p != '*2':
        # Рассматриваем все возможные следующие шаги
        return
            f(c + 1, e, '+1') + f(c + 2, e, '+2') + f(c * 2, e, '*2')

    # Если последнее действие было умножением, то удваивать снова нельзя
    if c < e and p == '*2':
        # Рассматриваем только увеличение на 1 и на 2
        return f(c + 1, e, '+1') + f(c + 2, e, '+2')
# Вызываем функцию с начальными значениями: текущее число = 1, целевое число = 15,
предыдущее действие отсутствует
print(f(1, 15, ''))
```

Этот алгоритм корректно подсчитывает количество всех возможных программ, удовлетворяющих условиям задачи, и выводит итоговый результат — 1545
Ответ: 1545

Задание № 8 (5220)

Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:
1. Прибавь 1
2. Умножь на 2
3. Умножь на 3

Первая команда увеличивает число на экране на 1, вторая умножает его на 2, третья – умножает на 3. Сколько существует различных программ, которые преобразуют исходное число 1 в число 157 и содержат больше команд умножения, чем сложения?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h25m40s

Решение

Для решения задачи необходимо учитывать, что программа должна содержать больше команд умножения, чем сложения. Воспользуемся рекурсией. Для этого создадим функцию f, которая принимает четыре аргумента:
· c — текущее число,

· e — целевое число (в нашем случае 157),
· k0 — количество выполненных команд сложения,
· k1 — количество выполненных команд умножения.

Функция f начинает работу с начальных значений: c = 1, e = 157, k0 = 0, k1 = 0. Эти значения соответствуют началу процесса преобразования числа 1 в 157 с нулевыми счётчиками сложений и умножений.

На каждом шаге функция анализирует текущее состояние и вызывает себя рекурсивно с обновленными значениями параметров после применения одной из трёх возможных команд.

Если текущая ветка решений приводит к превышению целевого числа, она отбрасывается (возвращается 0).

Когда достигается целевая сумма 157, функция проверяет, выполнено ли условие о том, что команд умножения больше, чем команд сложения. Если условие выполняется, возвращается 1, означающее успешную траекторию.

```
def f(c,e,k0,k1):
    if c>e: return 0
    if c==e: return k1>k0
    if c<e: return f(c+1,e,k0+1,k1)+f(c*2,e,k0,k1+1)+f(c*3,e,k0,k1+1)

print(f(1,157,0,0))
```

Таким образом, итоговый результат — это общее количество успешных траекторий, удовлетворяющих условиям задачи.

Ответ: 113

Задание №9 (5273)

Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:

1. Умножь на 5
2. Умножь на 3
3. Прибавь 45

Первая команда умножает число на экране на 5, вторая – умножает на 3, третья – увеличивает на 45. Сколько существует различных программ, которые преобразуют исходное число 1 в число 2970, и при этом траектория вычислений не более 4 команд «умножь на 5», не менее 2 команд «умножь на 3», и ровно 5 команд «прибавь 45»?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h34m0s

Решение

Наша цель состоит в том, чтобы найти количество различных программ, которые могут преобразовать исходное число 1 в число 2970, соблюдая следующие ограничения:
· Программа должна содержать не более 4 команд «умножь на 5».
· В программе должно быть не менее 2 команд «умножь на 3».
· Программа обязательно должна включать ровно 5 команд «прибавь 45».

Для того чтобы решить эту задачу, мы будем отслеживать текущее состояние программы через несколько параметров:
· c — текущее значение числа на экране,
· k1 — количество выполненных команд «умножь на 5»,

· k2 — количество выполненных команд «умножь на 3»,
· k3 — количество выполненных команд «прибавь 45».
Наша функция будет проверять различные варианты выполнения команд и возвращать количество допустимых программ, удовлетворяющих всем условиям.

```
def f(c, e, k1, k2, k3):
    # Если текущее число превышает целевое, значит программа ошибочна
    if c > e:
        return 0

    # Если текущее число достигло целевого значения, проверяем, выполнены ли все
условия
    if c == e:
        return k1 <= 4 and k2 >= 2 and k3 == 5

    # Если текущее число меньше целевого, продолжаем выполнение команд
    if c < e:
        return
            f(c * 5, e, k1 + 1, k2, k3) +  f(c * 3, e, k1, k2 + 1, k3) + f(c + 45, e,
k1, k2, k3 + 1)

print(f(1,2970,0,0,0))
```

Ответ: 74

Задание №10 (6008)

Исполнитель Калькулятор преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены коды:
A. Прибавь 1
B. Умножь на 2
C. Умножь на 3
Первая команда увеличивает число на 1, вторая – умножает его на 2, третья – умножает на 3. Программа для исполнителя – это последовательность команд. Сколько существует программ, для которых при исходном числе 2 результатом будет являться число 27, при этом программа содержит подпоследовательность команд BCA.

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h38m55s

Решение
Основная идея решения этой задачи заключается в том, чтобы отслеживать текущее состояние программы (текущий результат), а также фиксировать выполненные команды в виде строки. Каждая операция изменяет текущее значение следующим образом:
· Операция A увеличивает текущее число на 1.
· Операция B удваивает текущее число.
· Операция C утраивает текущее число.
Нам важно учитывать не только текущее значение, но и всю последовательность команд, которая привела к этому значению. Это позволит проверить наличие нужной подпоследовательности BCA.
Рассмотрим функцию f, которая принимает три аргумента:
· c — текущее значение числа,
· e — целевое значение (в нашем случае 27),
· tr — строка, содержащая последовательность выполненных команд.

Функция работает следующим образом:
1. Если текущее значение больше целевого (c > e), возвращаем 0, потому что невозможно достичь цели через допустимые операции.
2. Если текущее значение равно целевому (c == e), проверяем, содержится ли в строке tr подпоследовательность BCA. Если да, возвращаем 1, иначе 0.
3. В противном случае продолжаем выполнение функций для каждого возможного следующего шага:
o Применяем операцию A: увеличиваем текущее значение на 1 и добавляем символ 'A' в строку истории.
o Применяем операцию B: удваиваем текущее значение и добавляем символ 'B' в строку истории.
o Применяем операцию C: утраиваем текущее значение и добавляем символ 'C' в строку истории.

```
def f(c, e, tr):
    # Если текущее значение превышает целевое, прекращаем поиск
    if c > e:
        return 0

    # Если достигли целевого значения, проверяем наличие BCA в истории
    if c == e:
        return 'BCA' in tr

    # Продолжаем рекурсию для каждой возможной операции
    if c < e:
        return f(c + 1, e, tr + 'A') + f(c * 2, e, tr + 'B') + f(c * 3, e, tr + 'C')

# Вызываем функцию с начальными параметрами
print(f(2, 27, ''))
```

Ответ: 5

Задание №11 (7209)

У исполнителя Калькулятор имеются три команды, которые обозначены латинскими буквами:
A. Вычесть 1
B. Прибавить 2
C. Умножить на 2
Найдите количество существующих программ, для которых при исходном числе 3 результатом является число 40, и при этом траектория вычислений содержит число 30 и не содержит числа 20, а программа не содержит двух команд A подряд.

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=1h47m0s

Решение

Для решения этой задачи нам нужно разработать рекурсивную функцию, которая будет учитывать ограничения и требования к программе. Рассмотрим ключевые моменты:
1. Разрыв последовательности команд: При достижении числа 30 необходимо следить за тем, чтобы предыдущие команды были учтены правильно. В частности, важно помнить, что выполнение двух команд A подряд недопустимо.

2. Контроль числа 30: Программа должна проходить через число 30 хотя бы один раз. Для этого удобно использовать дополнительный параметр, который будет подсчитывать количество появлений числа 30.
3. Ограничение на число 20: Программа не должна включать число 20 в своей траектории вычислений.
4. Запрет на два подряд идущих A: Это важное ограничение, которое нужно проверять каждый раз перед выполнением команды A.
1. Функция f принимает четыре параметра:
o c: Текущее значение
o e: Целевой результат (в данном случае 40)
o p: Предыдущая команда (для контроля на запрет двух последовательных A)
o k30: Счётчик количества прохождений через число 30
2. Проверки условий:
o Если текущее значение превышает целевое значение плюс 1 ($c > e + 1$) или равно 20, возвращается 0, так как такая программа не подходит под условия задачи.
o Если текущее значение достигает целевой точки ($c == e$), проверяется, было ли пройдено число 30 ($k30 > 0$).
3. Рекурсия:
o Если предыдущая команда не была A, выполняются все три возможные операции (A, B, C) и суммируются результаты их вызовов.
o Если же предыдущая команда была A, то исключается повторный вызов A, остаются только варианты B и C.
Важные моменты:
· Отслеживание числа 30: Параметр k30 используется для учёта прохождения через число 30, что критично для удовлетворения условий задачи.
· Запрет на двойное использование A: Контролируется путём проверки предыдущей команды перед вызовом новой команды A.

```
def f(c, e, p, k30):
    # Если текущее значение равно 30, увеличиваем счетчик k30 на 1,
    # чтобы зафиксировать прохождение через число 30
    if c == 30:
        k30 += 1

    # Если текущее значение больше целевого значения плюс 1 или равно 20,
    # значит данная ветвь поиска некорректна, возвращаем 0
    if c > e + 1 or c == 20:
        return 0

    # Если текущее значение совпадает с целевым, проверяем, был ли достигнут 30
    if c == e:
        return k30 > 0

    # Если предыдущая команда не была "A", то можно выполнить любую из трех команд
    if p != 'a':
        # Возвращаем сумму результатов выполнения всех трех команд
        Return (f(c - 1, e, 'a', k30) +
            f(c + 2, e, 'b', k30) +
            f(c * 2, e, 'c', k30))

    # Если предыдущая команда была "A", то нельзя выполнить эту команду еще раз
```

```
    if p == 'a':
        # Выполняем только команды "B" и "C"
        return (f(c + 2, e, 'b', k30) +
            f(c * 2, e, 'c', k30))

# Запускаем функцию с начальными параметрами: начальное значение 3, цель 40, нет предыдущей команды, счетчик 30 равен 0
print(f(3, 40, '', 0))
```

Ответ : 26906168

Задание №12 (5543)

Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:

1. Прибавь 2
2. Умножь на 3
3. Умножь на 4

Выполняя первую из них, исполнитель увеличивает число на экране на 2, выполняя вторую – умножает на 3, выполняя третью – умножает на 4. Программой для исполнителя называется последовательность команд. Сколько существует различных программ, которые преобразуют исходное число 1 в число 600, и при этом траектория вычислений не содержит двух идущих подряд нечётных чисел.

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=2h1m45s

Решение

Для начала заметим, что каждое действие приводит либо к увеличению текущего числа, либо к его умножению. Таким образом, после каждого шага значение числа будет увеличиваться. Однако, нам необходимо следить за тем, чтобы не возникало ситуации, когда два последовательных числа были бы нечётными.
Если текущее число чётное, то любое из трёх действий допустимо, поскольку результат любого действия также будет чётным числом. В таком случае программа может продолжить выполнение любой из трёх команд без нарушения условия.
Однако, если текущее число нечётное, ситуация усложняется. Добавление 2 к нечётному числу даст снова нечётный результат, а умножение на 3 также приведёт к нечётному числу. Оба эти действия недопустимы, так как они нарушат правило отсутствия двух последовательных нечётных чисел. Единственное возможное действие в такой ситуации — умножение на 4, которое всегда даёт чётное число.
Таким образом, для каждой текущей позиции числа (чётной или нечётной), мы должны выбрать одно или несколько допустимых действий, продолжая до тех пор, пока не достигнем числа 600.
Рекурсивная функция работает следующим образом:
1. Если текущее число больше целевого (c > e), то возврат 0, так как невозможно получить число 600, начав с большего значения.
2. Если текущее число равно целевому (c == e), то возвращаем 1, так как найдено решение.

3. Если текущее число меньше целевого и является чётным (c % 2 == 0), то возможны любые действия, поэтому возвращаем сумму результатов всех трёх возможных действий.
4. Если текущее число меньше целевого и является нечётным (c % 2 != 0), то возможно только умножение на 4, поэтому возвращаем результат только одного действия.

```
def f(c, e):
    # Если текущее число больше целевого, нет решений
    if c > e:
        return 0

    # Если текущее число равно целевому, найдено решение
    if c == e:
        return 1

    # Если текущее число чётное, можно применить любую команду
    if c < e and c % 2 == 0:
        return f(c + 2, e) + f(c * 3, e) + f(c * 4, e)

    # Если текущее число нечётное, возможна только команда умножения на 4
    if c < e and c % 2 != 0:
        return f(c * 4, e)

# Находим количество программ, переводящих 1 в 600
print(f(1, 600))
```

Ответ : 20375

Задание №13 (7177)

У исполнителя Калькулятор имеются три команды, которые обозначены латинскими буквами:
A. Прибавить 2
B. Прибавить 3
C. Умножить на 4
Найдите количество существующих программ, для которых при исходном числе 1 результатом является число 100, и при этом траектория вычислений содержит строго больше чётных чисел, чем нечётных. В ответе запишите сумму цифр этого числа.

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=2h6m55s

Решение

Для решения задачи требуется найти количество программ, которые преобразуют число 1 в число 100, используя команды A, B и C, где каждая команда соответственно прибавляет к числу 2, 3 или умножает его на 4. При этом необходимо учитывать, что среди всех промежуточных значений должно быть больше четных чисел, чем нечетных.
В процессе решения важно отслеживать четность каждого числа, которое появляется в ходе выполнения программы. Для этого удобно использовать два счетчика: один для четных чисел (k0),

другой для нечетных (k1). Эти счетчики должны обновляться перед выполнением каждой операции, чтобы правильно учесть каждое значение.
При выполнении команд важно проверять, находится ли текущее число с больше, меньше или равно целевому значению e. Если c > e, значит программа зашла в тупик и должна вернуть 0. Если c == e, проверяется условие, что четных чисел оказалось больше, чем нечетных. Если это условие выполняется, функция возвращает 1, иначе – 0. Если же c < e, продолжается рекурсивный поиск решений путем применения каждой из трех команд.
Чтобы избежать повторных вычислений, используется декоратор lru_cache из модуля functools, который позволяет кэшировать результаты предыдущих вызовов функции.

```
from functools import lru_cache  # Импортируем декоратор для кеширования результатов функций

@lru_cache(None)  # Используем кеширование для ускорения работы функции за счёт сохранения ранее полученных результатов
def f(c, e, k0, k1):

    # Если текущее число чётное, увеличиваем счётчик чётных чисел
    if c % 2 == 0:
        k0 += 1
    # Если текущее число нечётное, увеличиваем счётчик нечётных чисел
    if c % 2 != 0:
        k1 += 1

    # Если текущее число больше целевого, программа завершается неудачно
    if c > e:
        return 0
    # Если текущее число равно целевому, проверяем, больше ли чётных чисел, чем нечётных
    if c == e:
        return k0 > k1
    # Если текущее число меньше целевого, продолжаем рекурсивный поиск решений
    if c < e:
        # Рассчитываем новые варианты, применяя каждую команду (прибавить 2, прибавить 3, умножить на 4)
        return f(c + 2, e, k0, k1) + f(c + 3, e, k0, k1) + f(c * 4, e, k0, k1)

# Вызываем функцию с начальными параметрами: начинаем с числа 1, целью является 100, изначально нет ни чётных, ни нечётных чисел
result = f(1, 100, 0, 0)

# Преобразуем результат в строку и суммируем все цифры результата
print(sum([int(digit) for digit in str(result)]))
```

Ответ : 58

Задание №14 (5257)

Исполнитель преобразует число, записанное на экране. У исполнителя есть три команды, которым присвоены номера:

1. Прибавь 2

2. Умножь на 2
3. Умножь на 3

Первая команда увеличивает число на экране на 2, вторая умножает его на 2, третья – умножает на 3. Сколько существует различных программ, которые преобразуют исходное число 1 в число 300, и при этом траектория вычислений содержит не более 3 чисел кратных 6 (не считая первое и последнее числа)?

Ссылка на видео-разбор с таймингом: https://vk.com/video-205546952_456241280?t=2h14m15s

Решение

Для решения этой задачи удобно использовать рекурсивный подход с мемоизацией. Мы будем строить дерево возможных состояний, где каждый узел представляет собой текущее состояние (число на экране), а переходы между узлами соответствуют применению одной из трёх команд. Чтобы ограничить количество чисел, кратных 6, введём дополнительный параметр k6, который будет отслеживать количество таких чисел на пути от начального состояния до текущего.
В этой функции:

- c — текущее число,
- e — целевое число (в нашем случае 300),
- k6 — количество чисел, кратных 6, встреченных на пути.

Функция возвращает 1, если текущая последовательность команд приводит к числу 300 и удовлетворяет ограничению на кратность 6, иначе возвращается 0.

```
def f(c, e, k6):
    # Если текущее число кратно 6, увеличиваем счётчик
    if c % 6 == 0:
        k6 += 1

    # Если текущее число превышает целевое, возвращаем 0
    if c > e:
        return 0

    # Если достигли целевого числа, проверяем, удовлетворяет ли оно условию
    if c == e:
        return k6 <= 4  # Учитываем, что 300 также кратно 6

    # Если ещё не достигли цели, продолжаем рекурсию
    if c < e:
        return (
            f(c + 2, e, k6) +
            f(c * 2, e, k6) +
            f(c * 3, e, k6)
        )

print(f(1,300,0))
```

Этот код вернёт количество различных программ, которые преобразуют число 1 в число 300, учитывая ограничения на кратность 6.

Ответ : 4912