САНКТ-ПЕТЕРБУРГСКИЙ

НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ

УНИВЕРСИТЕТ ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №7 по курсу «Алгоритмы и структуры данных»

Вариант 23

Выполнил: Хабиби Я.

К3140

Проверил: Афанасьев А. В.

Санкт-Петербург 2024 г.

**Задача №2**

Необходимо реализовать алгоритм для нахождения минимального количества операций, необходимых для преобразования числа 1 в заданное число n. Программа должна выводить минимальное количество операций и саму последовательность чисел, которая приведет к заданному числу.

**Формат выходных данных:**

* Первая строка: минимальное количество операций.
* Вторая строка: последовательность чисел от 1 до n, полученная с помощью оптимальных операций.

Решение

from lab7.utils import measure\_performance,file\_write,file\_read  
import os, sys  
  
abspath = os.path.abspath(\_\_file\_\_)  
dname = os.path.dirname(abspath)  
sys.path.append(dname)  
os.chdir(dname)  
  
def optimal\_sequence(n):  
 *"""  
 Вычисляет минимальное количество операций для достижения числа n из 1  
 и возвращает последовательность промежуточных значений.  
 """* operations = [0] \* (n + 1)  
  
 for i in range(2, n + 1):  
 min\_ops = operations[i - 1] + 1  
 if i % 2 == 0:  
 min\_ops = min(min\_ops, operations[i // 2] + 1)  
 if i % 3 == 0:  
 min\_ops = min(min\_ops, operations[i // 3] + 1)  
 operations[i] = min\_ops  
  
 sequence = []  
 while n > 0:  
 sequence.append(n)  
 if n % 3 == 0 and operations[n] == operations[n // 3] + 1:  
 n //= 3  
 elif n % 2 == 0 and operations[n] == operations[n // 2] + 1:  
 n //= 2  
 else:  
 n -= 1  
  
 return operations[-1], sequence[::-1]  
  
  
def primitive\_calculator\_main():  
 n = file\_read()[0][0] # Чтение числа n  
 min\_ops, sequence = optimal\_sequence(n)  
 file\_write([min\_ops, sequence])  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 measure\_performance(primitive\_calculator\_main)

Тесты:

import unittest  
from lab7.task2.src.primitive\_calculator import optimal\_sequence  
from lab7.utils import measure\_performance  
  
class TestPrimitiveCalculator(unittest.TestCase):  
  
 def test\_example\_case\_1(self):  
 # given  
 n = 1  
  
 # when  
 result, elapsed\_time, peak\_memory = measure\_performance(optimal\_sequence, n)  
  
 # then  
 self.assertEqual(result, (0, [1]))  
 self.assertLessEqual(elapsed\_time, 1, "Execution time exceeded 1 second")  
 self.assertLessEqual(peak\_memory, 256, "Memory usage exceeded 256 MB")  
  
 def test\_example\_case\_2(self):  
 # given  
 n = 5  
  
 # when  
 result, elapsed\_time, peak\_memory = measure\_performance(optimal\_sequence, n)  
  
 # then  
 self.assertEqual(result, (3, [1, 2, 4, 5]))  
 self.assertLessEqual(elapsed\_time, 1, "Execution time exceeded 1 second")  
 self.assertLessEqual(peak\_memory, 256, "Memory usage exceeded 256 MB")  
  
 def test\_large\_input(self):  
 # given  
 n = 96234  
  
 # when  
 result, elapsed\_time, peak\_memory = measure\_performance(optimal\_sequence, n)  
  
 # then  
 self.assertGreater(len(result[1]), 0, "Sequence should not be empty")  
 self.assertLessEqual(elapsed\_time, 1, "Execution time exceeded 1 second")  
 self.assertLessEqual(peak\_memory, 256, "Memory usage exceeded 256 MB")  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

## Задание №4. Наибольшая общая подпоследовательность двух подпоследовательностей

**Описание задачи:**

Задача заключается в нахождении длины наибольшей общей подпоследовательности (НОП) для двух заданных последовательностей. Подпоследовательность — это упорядоченный набор элементов исходной последовательности, который не обязательно состоит из подряд идущих элементов. Например, для последовательностей:

* Первая последовательность: 2, 7, 5
* Вторая последовательность: 2, 5

Наибольшей общей подпоследовательностью будет 2, 5. Длина этой подпоследовательности равна 2.

Решение:

|  |
| --- |
| from lab7.utils import measure\_performance, file\_write, file\_read  import os, sys abspath = os.path.abspath(\_\_file\_\_) dname = os.path.dirname(abspath) sys.path.append(dname) os.chdir(dname)   def lcs\_length(seq1, seq2):  *"""  Вычисление длины наибольшей общей подпоследовательности.  """* n, m = len(seq1), len(seq2)  dp = [[0] \* (m + 1) for \_ in range(n + 1)]   for i in range(1, n + 1):  for j in range(1, m + 1):  if seq1[i - 1] == seq2[j - 1]:  dp[i][j] = dp[i - 1][j - 1] + 1  else:  dp[i][j] = max(dp[i - 1][j], dp[i][j - 1])   return dp[n][m]   def lcs\_main(seq1=None, seq2=None):  *"""  Главная функция. Если входные последовательности переданы, используется для тестов.  Иначе данные читаются из файлов.  """* if seq1 is None or seq2 is None: # Проверяем, переданы ли данные для тестирования  data = file\_read() # Читаем данные через file\_read  n, seq1 = data[0][0], data[1] # Первая строка: длина и последовательность 1  m, seq2 = data[2][0], data[3] # Вторая строка: длина и последовательность 2   result = lcs\_length(seq1, seq2) # Вычисляем LCS  file\_write([result]) # Записываем результат  else:  result = lcs\_length(seq1, seq2) # Возвращаем результат для тестов  return result   if \_\_name\_\_ == '\_\_main\_\_':  measure\_performance(lcs\_main) |

Тесты:

import unittest  
import random  
from lab7.task4.src.longest\_common\_subsequence import lcs\_main  
  
  
class TestLongestCommonSubsequence(unittest.TestCase):  
 def test\_basic\_case(self):  
 # given  
 input\_data = ([2, 7, 5], [2, 5])  
 expected\_output = 2  
  
 # when  
 result = lcs\_main(\*input\_data)  
  
 # then  
 self.assertEqual(result, expected\_output)  
  
 def test\_full\_match(self):  
 # given  
 input\_data = ([1, 2, 3, 4], [1, 2, 3, 4])  
 expected\_output = 4  
  
 # when  
 result = lcs\_main(\*input\_data)  
  
 # then  
 self.assertEqual(result, expected\_output)  
  
 def test\_large\_input(self):  
 # given  
 input\_data = (  
 [random.randint(1, 100) for \_ in range(100)],  
 [random.randint(1, 100) for \_ in range(100)],  
 )  
  
 # when  
 result = lcs\_main(\*input\_data)  
  
 # then  
 self.assertTrue(isinstance(result, int)) # Проверяем, что результат целое число  
  
 def test\_no\_common\_subsequence(self):  
 # given  
 input\_data = ([7], [1])  
 expected\_output = 0  
  
 # when  
 result = lcs\_main(\*input\_data)  
  
 # then  
 self.assertEqual(result, expected\_output)  
  
  
if \_\_name\_\_ == '\_\_main\_\_':  
 unittest.main()

|  |
| --- |
| ["put", "key2", "value2"], |
| ["delete", "key1"], |
| ["get", "key1"], |
| ["get", "key2"] |
| ] |
| expected\_results = ["<none>", "value2"] |
|  |
| # when |
| results = associative\_array(commands) |
|  |
| # then |
| self.assertEqual(results, expected\_results) |
|  |
| def test\_should\_check\_time\_data(self): |
| # given |
| expected\_time = 4 |
|  |
| # when |
| time = time\_data(main) |
|  |
| # then |
| self.assertLess(time, expected\_time) |
|  |
| def test\_should\_check\_memory\_data(self): |
| # given |
| expected\_memory = 256 |
|  |
| # when |
| current, peak = memory\_data(main) |
| # then |
| self.assertLess(current, expected\_memory) |
| self.assertLess(peak, expected\_memory) |
|  |
| if \_\_name\_\_ == "\_\_main\_\_": |
| unittest.main() |
|  |

# Вывод

Лабораторная работа №7 успешно выполнена, включая реализацию и тестирование всех задач. Решения организованы в соответствии с требованиями проекта: работа с файлами сосредоточена в модуле utils.py, алгоритмы реализованы в папках src, а тесты покрывают все необходимые сценарии. Все задачи продемонстрировали корректность решений и соответствие критериям производительности и памяти. Входные и выходные данные обработаны без ошибок, результаты подтверждены успешным прохождением тестов