САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ
ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ
ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

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

## Вариант 15

Выполнил:
Левахин Лев Александрович
К3140

Проверил:
--

Санкт-Петербург
2024 г.

## Содержание отчета

**Задания по варианту: 4, 6**

Задача 4 – Наибольшая общая подпоследовательность двух последовательностей

Задача 6 – Наибольшая возрастающая подпоследовательность

**Задания по выбору: 1, 2**

Задача 1 – Обмен монет

Задача 2 – Примитивный калькулятор

## Задачи по выбору

### Задача №1. Обмен монет

Листинг кода.

```
from lab7 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def min_coins(money, coins):
    dp = [float('inf')] * (money + 1)  # Создаем массив для хранения
минимального количества монет для каждой суммы
    dp[0] = 0  # Нулевая сумма требует ноль монет

    # Заполняем массив dp
    for coin in coins:
        for x in range(coin, money + 1):
            dp[x] = min(dp[x], dp[x - coin] + 1)

    return dp[money] if dp[money] != float('inf') else -1  # Если сумма не
достижима


def input_data() -> (int, int, list):
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    line1 = data[0]
    line2 = data[1]
    money, k = map(int, line1.split())
    coins_list = utils.str_to_list(line2)
    return money, k, coins_list


def main():
    money, k, coins_list = input_data()
    result = min_coins(money, coins_list)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:

Листинг кода:

```
from lab7 import utils
from lab7.task1.src import *
import unittest
import tracemalloc
import datetime
```

```
from lab7.task1.src.task1 import min_coins


class TaskTest1(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        money, k, coins_list = 2, 3, [1, 3, 4]

        max_allowed_time = datetime.timedelta(seconds=1)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = min_coins(money, coins_list)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, 2)
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)


    def test_func_correctly(self):
        """Тест на корректность работы"""
        # given
        money, k, coins_list = 2, 3, [1, 3, 4]
        money2, k2, coins_list2 = 34, 3, [1, 3, 4]


        # when
        result1 = min_coins(money, coins_list)
        result2 = min_coins(money2, coins_list2)

        # then
        self.assertEqual(result1, 2)
        self.assertEqual(result2, 9)


if __name__ == "__main__":
    unittest.main()
```

Скрины работы тестов:

## Задача №2. Примитивный калькулятор

Листинг кода.

```
from lab7 import utils
import os
from collections import deque

CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))
```

```python
def find_min_operations(n: int) -> list:
    # Массив для хранения минимального количества операций до каждого числа
    operations = [float('inf')] * (n + 1)
    # Массив для хранения предшественников
    predecessor = [-1] * (n + 1)

    # Начальные условия
    operations[1] = 0
    queue = deque([1])

    while queue:
        current = queue.popleft()

        # Возможные следующие шаги
        next_steps = [current + 1, current * 2, current * 3]

        for next_num in next_steps:
            if next_num <= n and operations[next_num] == float('inf'):
                operations[next_num] = operations[current] + 1
                predecessor[next_num] = current
                queue.append(next_num)

    path = []  # Путь от n к 1
    step = n
    while step != -1:
        path.append(step)
        step = predecessor[step]

    path.reverse()  # Переворачиваем путь, чтобы получить от 1 до n

    return [operations[n], path]


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n = int(data[0])
    return n


def main():
    n = input_data()
    result = find_min_operations(n)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```python
from lab7.task2.src.task2 import *
from lab7 import utils
import unittest
import datetime
import tracemalloc


class TaskTest2(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        n = 96234

        max_allowed_time = datetime.timedelta(seconds=1)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = find_min_operations(n)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, [14, [1, 2, 6, 7, 21, 22, 66, 198, 594,
1782, 5346, 16038, 16039, 32078, 96234]])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)


    def test_func_correctly(self):
        """Тест на корректность работы"""
        # given
        n1 = 1
        n2 = 5
        n3 = 96234

        # when
        result1 = find_min_operations(n1)
        result2 = find_min_operations(n2)
        result3 = find_min_operations(n3)

        # then
        self.assertEqual(result1, [0, [1]])
        self.assertEqual(result2, [3, [1, 2, 4, 5]])
        self.assertEqual(result3, [14, [1, 2, 6, 7, 21, 22, 66, 198, 594,
1782, 5346, 16038, 16039, 32078, 96234]])


if __name__ == "__main__":
    unittest.main()
```

Скрины работы тестов:

## Задачи по варианту

### Задача №4. Наибольшая общая подпоследовательность двух последовательностей

Листинг кода.

```
from lab7 import utils
import os


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))


def longest_common_subsequence(n1:int, lst1:list, n2:int, lst2:list) ->
int:
    dp = [[0] * (n2 + 1) for i in range(n1 + 1)] # Таблица

    # Заполняем таблицу
    for i in range(1, n1 + 1):
        for j in range(1, n2 + 1):
            if lst1[i - 1] == lst2[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[n1][n2]


def input_data() -> (int, list, int, list):
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n1, lst1, n2, lst2 = int(data[0]), utils.str_to_list(data[1]),
int(data[2]), utils.str_to_list(data[3])
    return n1, lst1, n2, lst2


def main():
    n1, lst1, n2, lst2 = input_data()
    result = longest_common_subsequence(n1, lst1, n2, lst2)
    return result


if __name__ == "__main__":
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, [result])
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab7.task4.src.task4 import *
from lab7 import utils
import unittest
import tracemalloc
import datetime
```

```
class TaskTest4(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        n1, lst1, n2, lst2 = 3, [2, 7, 5], 2, [2, 5]

        max_allowed_time = datetime.timedelta(seconds=1)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = longest_common_subsequence(n1, lst1, n2, lst2)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, 2)
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 512)

    def test_func_correctly(self):
        """Тест на корректность работы"""
        # given
        len_a1, a1, len_b1, b1 = 3, [2, 7, 5], 2, [2, 5]
        len_a2, a2, len_b2, b2 = 1, [7], 4, [1, 2, 3, 4]
        len_a3, a3, len_b3, b3 = 4, [2, 7, 8, 3], 4, [5, 2, 8, 7]

        # when
        result1 = longest_common_subsequence(len_a1, a1, len_b1, b1)
        result2 = longest_common_subsequence(len_a2, a2, len_b2, b2)
        result3 = longest_common_subsequence(len_a3, a3, len_b3, b3)

        # then
        self.assertEqual(result1, 2)
        self.assertEqual(result2, 0)
        self.assertEqual(result3, 2)


if __name__ == "__main__":
    unittest.main()
```

## Задача №6. Наибольшая возрастающая подпоследовательность

Листинг кода.

```
from lab7 import utils
import os
from bisect import bisect_left


CURRENT_SCRIPT_DIR_PATH = os.path.dirname(os.path.abspath(__file__))
```

```python
def find_lis(n: int, sequence: list) -> list:
    """
    Принимает последовательность и находит длину и саму наибольшую
возрастающую подпоследовательность
    """
    if not sequence:
        return [0, []]

    tails = []  # Массив для хранения наименьших хвостов возрастающих
подпоследовательностей (значение, индекс, длина)
    predecessors = [-1] * len(sequence)
    lengths = [0] * len(sequence)

    for i, num in enumerate(sequence):
        pos = bisect_left(tails, num, key=lambda x: x[0])

        if pos == len(tails):
            if tails:
                predecessors[i] = tails[-1][1]
                lengths[i] = tails[-1][2] + 1
            else:
                lengths[i] = 1
            tails.append((num, i, lengths[i]))
        else:
            if pos > 0 and tails[pos-1][0] < num:
                  predecessors[i] = tails[pos-1][1]
                  lengths[i] = tails[pos-1][2] + 1
            else:
                  lengths[i] = 1
            tails[pos] = (num, i,lengths[i])

    # Восстанавливаем последовательность
    max_len = 0
    end_index = -1
    for i, l in enumerate(lengths):
        if l > max_len:
            max_len = l
            end_index = i

    path = []
    current_index = end_index
    while current_index != -1:
        path.append(sequence[current_index])
        current_index = predecessors[current_index]

    path.reverse()

    return [max_len, path]


def input_data():
    data = utils.read_file(CURRENT_SCRIPT_DIR_PATH)
    n, lst = int(data[0]), utils.str_to_list(data[1])
    return n, lst


def main():
    n, lst = input_data()
    result = find_lis(n, lst)
    return result


if __name__ == "__main__":
```

```
    result = main()
    utils.write_file(CURRENT_SCRIPT_DIR_PATH, result)
```

Текстовое объяснение решения.

Результат работы кода на примерах из текста задачи:
(скрины input output файлов)

Тесты к задаче:
Листинг кода:

```
from lab7.task6.src.task6 import *
from lab7 import utils
import unittest
import datetime
import tracemalloc


class TaskTest6(unittest.TestCase):

    def test_func_performance(self):
        """Тест функции на время и память"""
        # given
        n, lst = 6, utils.str_to_list("3 29 5 5 28 6")

        max_allowed_time = datetime.timedelta(seconds=2)  # Задаю
ограничение по времени

        # when
        tracemalloc.start()  # Запускаем счётчик памяти
        start_time = datetime.datetime.now()  # Запускаем счётчик времени

        result = find_lis(n, lst)

        finish_time = datetime.datetime.now()
        spent_time = finish_time - start_time  # Итоговое время

        current, peak = tracemalloc.get_traced_memory()
        memory_used = current / 10 ** 6

        # then
        self.assertEqual(result, [3, [3, 5, 28]])
        self.assertLessEqual(spent_time, max_allowed_time)
        self.assertLessEqual(memory_used, 256)


if __name__ == "__main__":
    unittest.main()
```

**Вывод**

Сложные задачи и реализация сложных структур данных требует огромной проработки, чтобы оптимизировать алгоритмы по времени и памяти. Однако, размер кода таких алгоритмов очень часто меньше, чем реализация самых простых алгоритмов. Так как в сложных структурах каждая строчка выполняет очень много функционала и максимально продумана.