Министерство науки и высшего образования Российской Федерации
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«Национальный исследовательский университет ИТМО»
(Университет ИТМО)

Факультет Инфокоммуникационных технологий (ИКТ)

Образовательная программа Мобильные и сетевые технологии

О Т Ч Е Т

по Лабораторной работе 5

Дисциплина: Алгоритмы и структуры данных.

Тема: Жадные алгоритмы. Динамическое программирование. Метод «разделяй и властвуй».

Специальность: 09.03.03 Прикладная информатика.

Проверил:	Выполнил:
Мусаев А. А. _____	Балцат К. И.,
Дата: «__»_____2023 г.	студент группы К33401
Оценка: _____	

Санкт-Петербург
2023

ВЫПОЛНЕНИЕ

1 Задача 1

Вор пробрался в музей и хочет украсть N экспонатов. У каждого экспоната есть свой вес и цена. Вор может сделать M заходов, каждый раз унося K кг веса. Определить, что должен унести вор, чтобы сумма украденного была максимальной.

Решение:

Для решения этой задачи с помощью жадного алгоритма, мы будем выбирать на каждом шаге экспонат с наибольшим соотношением цена/вес, чтобы максимизировать ценность украденных экспонатов за каждый заход.

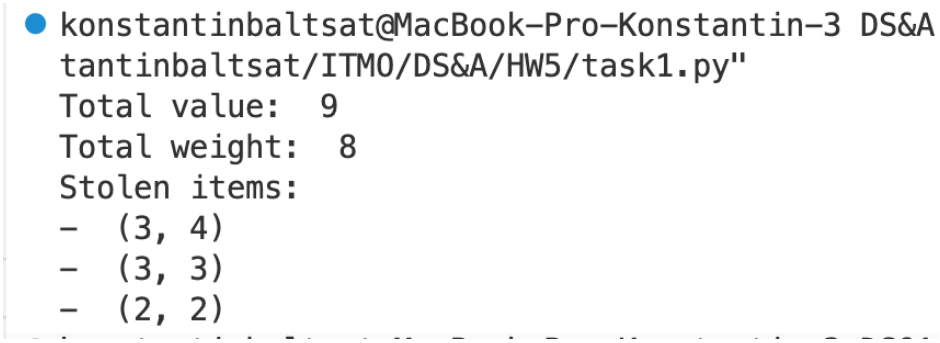
Так, код реализации с примером будет выглядеть следующим образом:

```python
def knapsack_greedy(n, m, k, items):
    items = sorted(items, key=lambda x: x[0], reverse=True)
    items = sorted(items, key=lambda x: x[1] / x[0], reverse=True)
    total_value = 0
    total_weight = 0
    num_items = n
    stolen_items = []
    for i in range(m):
        weight_left = k
        for item in items:
            if weight_left == 0 or num_items == 0:
                break
            if item[0] <= weight_left and item not in stolen_items:
                weight_left -= item[0]
                total_value += item[1]
                total_weight += item[0]
                stolen_items.append(item)
                num_items -= 1
            elif item[0] > weight_left:
                continue
    return total_value, total_weight, stolen_items


if __name__=='__main__':
    items = [(1, 1), (2, 2), (3, 3), (3, 4), (4, 100)] # items = [(weight, value)]
    K = max_weight_per_visit = 3
    M = max_visits = 3
    N = 3

    total_value, total_weight, stolen_items = knapsack_greedy(N, M, K, items)
    print("Total value: ", total_value)
    print("Total weight: ", total_weight)
    print("Stolen items: ")
    for item in stolen_items:
        print("- ", item)
```

Результат выполнения:



2 Задача 2

Дана последовательность матриц A, B, C, … , Z таким образом, что с ними можно выполнить ассоциативные операции. Используя динамическое программирование, минимизируйте количество скалярных операций для нахождения их произведения.

Решение:

Это классическая задача о переменожении матриц. Для решения данной задачи с помощью динамического программирования мы можем использовать следующий алгоритм:

Создаем матрицу dp размером n x n, где n - количество матриц в последовательности. dp[i][j] будет содержать минимальное количество скалярных операций, необходимых для перемножения матриц от i до j включительно.

Заполняем диагональные элементы матрицы dp нулями, так как перемножение одной матрицы не требует скалярных операций.

Для каждого возможного отрезка [i, j], где i < j, мы находим минимальное количество скалярных операций, необходимых для перемножения матриц от i до j, используя уже вычисленные значения в dp. Для этого мы перебираем все возможные точки k на отрезке [i, j-1] и вычисляем количество операций, необходимых для перемножения матриц от i до k и от k+1 до j. Затем мы суммируем это количество скалярных операций

с количеством скалярных операций для перемножения матриц от i до k и от k+1 до j, которые мы уже вычислили ранее. Минимальное значение из всех возможных точек k будет записано в ячейку dp[i][j].

В конечном итоге, минимальное количество скалярных операций, необходимых для перемножения всех матриц, будет храниться в ячейке dp[0][n-1].

Вот реализация данного алгоритма:

```python
def matrix_multiply_cost(matrix_list):
    n = len(matrix_list)
    dp = [[0] * n for _ in range(n)]

    # заполняем диагональные элементы нулями
    for i in range(n):
        dp[i][i] = 0

    # перебираем все возможные отрезки и находим минимальное количество скалярных операций
    for length in range(2, n+1):
        for i in range(n-length+1):
            j = i + length - 1
            dp[i][j] = float('inf')
            for k in range(i, j):
                cost = dp[i][k] + dp[k+1][j] + matrix_list[i][0] * matrix_list[k][1] * matrix_list[j][1]
                dp[i][j] = min(dp[i][j], cost)

    # возвращаем минимальное количество скалярных операций
    return dp[0][n-1]

if __name__=='__main__':
    matrix_list = [(5, 10), (10, 20), (20, 35)]
    print(matrix_multiply_cost(matrix_list))  # Output: 4500
```

Результатом работы программы будет: 4500.

Действительно, перемножать несколько матриц можно несколькими способами. Например, если у нас имеются матрицы X, Y и Z, то вычислить XYZ можно либо как (XY)Z, либо как X(YZ). Пусть X имеет размер 5×10, Y имеет размер 10×20, Z имеет размер 20×35. Подсчитаем количество умножений, необходимых для перемножения трех матриц в каждом из этих двух случаях:

(XY)Z

$5 \times 10 \times 20 = 1000$ умножений для определения матрицы (XY), имеющей размер 5×20.

Потом $5 \times 20 \times 35 = 3500$ умножений для нахождения конечного результата.

Общее количество умножений: 4500.

X(YZ)

10 × 20 × 35 = 7000 умножений для определения матрицы (YZ), имеющей размер 10 × 35.

Потом 5 × 10 × 35 = 1750 умножений для нахождения конечного результата.

Общее количество умножений: 8750.

3 Задача 3

Дан массив N, состоящий из n случайных целых чисел, находящихся в диапазоне от -100 до 100. Найти наибольшую непрерывную возрастающую последовательность из чисел внутри массива (длину серии, для которой верно $N[i]<N[i+1]<N[i+2]<\ldots<N[i+m]$, где $i\geq 0$, а $i+m\leq n-1$).

Решение:

Опишем решение задачи за O(N^2). Построим массив d, где d[i] — это длина наибольшей возрастающей подпоследовательности (НВП), оканчивающейся в элементе, с индексом i. Массив будем заполнять постепенно — сначала d[0], потом d[1] и т.д. Ответом на нашу задачу будет максимум из всех элементов массива d.

Заполнение массива будет следующим: если d[i]=1, то искомая последовательность состоит только из числа a[i]. Если d[i]>1, то перед числом a[i] в подпоследовательности стоит какое-то другое число. Переберем его: это может быть любой элемент a[j] (j=0...i−1), но такой, что a[j]<a[i].

Пусть на каком-то шаге нам надо посчитать очередное d[i]. Все элементы массива d до него уже посчитаны. Значит наше d[i] мы можем посчитать следующим образом: d[i]=1+max{j=0...i−1}(d[j]) при условии, что a[j]<a[i].

Пока что мы нашли лишь максимальную длину наибольшей возрастающей подпоследовательности, но саму ее мы вывести не можем. Для восстановления ответа заведем массив prev[0...n−1], где prev[i] будет означать индекс в массиве a, при котором достигалось наибольшее значение d[i].

Для вывода ответа будем идти от элемента с максимальным значениям d[i] по его предкам.

Код решения выглядит следующим образом:

```
def findLIS(a):
    n = len(a)           # размер исходного массива
    prev = [-1] * n      # массив, хранящий индексы предыдущих элементов в НВП
    d = [1] * n          # массив, хранящий длины НВП для каждого элемента массива

    for i in range(n):
        for j in range(i):
            if a[j] < a[i] and d[j] + 1 > d[i]:
                d[i] = d[j] + 1
                prev[i] = j   # запоминаем индекс предыдущего элемента в НВП

    pos = 0               # индекс последнего элемента в НВП
    length = d[0]         # длина НВП
    for i in range(n):
        if d[i] > length:
            pos = i       # обновляем индекс последнего элемента в НВП
            length = d[i] # обновляем длину НВП

    answer = []
    while pos != -1:
        answer.append(a[pos])
        pos = prev[pos]   # восстанавливаем НВП, переходя к предыдущему элементу

    answer.reverse()     # переворачиваем список, чтобы он был в порядке возрастания

    return answer
```

Результат решения:

```python
if __name__ == '__main__':
    import random
    N = 20
    a = [random.randint(-100, 100) for _ in range(N)]
    print('a', a)
    print('LIS', findLIS(a))
```

PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
konstantinbaltsat@MacBook-Pro-Konstantin-3 DS&A % /Users/konstantinbaltsat/.pyenv/versio
baltsat/ITMO/DS&A/HW5/task3.py"
a [-12, -72, 8, 8, -3, -53, -8, -25, -10, -63, 18, 44, 9, 31, -76, 40, -94, 7, -8, 14]
LIS [-72, -53, -25, -10, 18, 31, 40]
```

ВЫВОД

Я на практике изучил алгоритмы динамического программирования. Теперь я способен эффективно применять каждый изученный алгоритм в зависимости от решаемой задачи.