Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский университет ИТМО»

Факультет инфокоммуникационных технологий

ОТЧЕТ

по лабораторной работе №1. Динамическое программирование №2
по дисциплине «Алгоритмы и структуры данных»

Выполнил студент 1 курса, группы К3140

Байков Иван

Преподаватель: Харьковская Татьяна Александровна

03 Июня, 2022 года, г. Санкт-Петербург

Задание 1

Описание

Найти наиболее ценную комбинацию предметов, учитывая, что может быть взята любая часть предмета. Задача о дробном рюкзаке.

Исходный код

```python
def max_worth(volume, array):
    worth = {}
    weight = {}
    for i in range(len(array)):
        worth[i] = array[i][0] / array[i][1]
        weight[i] = array[i][1]
    worth = dict(sorted(worth.items(), reverse=True, key=lambda x: x[1]))
    list_order = list(worth.keys())
    capacity = 0
    i = 0
    result = 0
    while capacity != volume:
        if capacity + weight[list_order[i]] <= volume:
            capacity += weight[list_order[i]]
            result += weight[list_order[i]] * worth[list_order[i]]
            i += 1
        else:
            m = volume - capacity
            capacity += m
            result += m * worth[list_order[i]]
    print(result)
```

Решение

Сначала для каждой пары (Стоимость/Вес) вычисляем ценность за 1 единицу веса, после чего заносим полученные значения в словарь (worth) и сортируем его в порядке убывания. Так же создадим словарь weight, в котором будем сохранять значения веса с такими же ключами, как в weight. Дальше запускаем цикл до тех пор, пока не наберем необходимый объем. Ценность предметов упорядочена по убыванию. Если предмет вмещается целиком – то начинаем брать следующие предметы до тех пор, пока не заполним всю сумку. Если предмет не помещается целиком – берем только его часть.

Тесты

```
###TESTS
n1 = 50
n2 = 10
n3 = 70
test1 = [[60, 20], [100, 50], [120, 30]]
test2 = [[500, 30]]
test3 = [[500, 30], [600, 30], [400, 25], [10, 1]]
max_worth(n1, test1)
max_worth(n2, test2)
max_worth(n3, test3)
```

```
180.0
166.66666666666669
1260.0
```

Взял два теста из условия и еще один рандомный.

Вывод

Реализован алгоритм решения популярной задачи на дробный рюкзак. Время работы алгоритма O(n), где n – количество предметов.

Задание 2

Описание

Нужно проехать расстояние d с запасом бензина на расстояние m за минимальное количество дозаправок, если заправочные станции расположены на расстояниях stations[i] от начала пути.

Исходный код

```python
def refueling(d, m, n, stations):
    # Count of refills
    count = 0
    # Last refill
    last = 0
    # Tank volume
    volume = m

    if volume >= d:
        return 0

    while volume < d:
        if last >= n or stations[last] > volume:
            return -1
        while last < n - 1 and stations[last + 1] <= volume:
            last += 1

        count += 1
        volume = m + stations[last]
        last += 1

    return count
```

Решение

В первую очередь проверяем, что мы не сможем проехать расстояние без заправок. Дальше запускаем цикл, который прервется, если расстояние от последней станции до конца пути больше m. Если проехать маршрут можно, то после каждых i*m «метров» мы проверяем, приехали ли мы. Если нет, то заправляемся на «прошедшей заправке» и плюсуем в count.

Тесты

```
print(refueling(950, 400, 4, [200, 375, 550, 750]))  # 2
print(refueling(10, 3, 4, [1, 2, 5, 9]))  # -1
```

```
2
-1
```

Тесты взяты из условия.

Вывод

Время работы алгоритма равно O(n*m), где n – количество станций, а m – это количество промежутков без дозаправки.

Задание 3

Описание

Дано n объявлений, каждое из которых приносит доход за клик равный a, и количество кликов по каждому слоту для объявлений b. Цель – максимизировать доход от рекламы.

Исходный код

```python
def max_income(n, a, b):
    result = 0
    a.sort()
    b.sort()

    for i in range(n):
        result += a[i] * b[i]

    return result
```

Решение

Достаточно простая задача. Нужно просто отсортировать массивы по возрастанию и перемножить каждые i-ы элементы. Все решение сделает за нас математика, ведь таким образом доход получится максимальным.

Тесты

```
print(max_income(1, [23], [39]))
print(max_income(3, [1, 3, -5], [-2, 4, 1]))
```

```
897
23
```

Тесты взяты из условия. Алгоритм стабильно работает даже с учетом отрицательного количества кликов (???).

Вывод

Время работы алгоритма 2 * O(nlog(n)) + O(n) –> TimSort, встроенный в питон + перемножение. Выполнена, скорее всего, наиболее простая реализация алгоритма за минимальное возможное время.

Задание 8

Описание

Дано N заявок на проведение лекций с промежутками a и b, a<b. Нужно узнать максимальное количество лекций, не пересекающихся по времени, которое получится провести в один день в одной аудитории.

Исходный код

```
def lectures(array):
    sorted_array = sorted(array, key=lambda x: x[1])
    end_hour = 0
    count = 1
    for i in range(1, len(sorted_array)):
        if sorted_array[i][0] >= sorted_array[end_hour][1]:
            count += 1
            end_hour = i
    return count
```

Решение

Начальный массив мы сортируем по возрастанию часов окончания – так будет проще, к тому же можно спокойно пропустить первый элемент, так как он всегда будет входить в итоговый результат, ведь его часы окончания и начала идут раньше остальных. Осталось только пробежать по всем элементам и проверить, для каких элементов часы начала[i+1] > часы начала[i].

Тесты

```
test = [[5, 6], [1, 5], [2, 3], [5, 8], [12, 20], [3, 4], [1, 2]]
```

```
5
```

Взял один тест из условия + немного усложнил. Так как часы могут быть только положительными и варьируются от 0 до 24 (преподаватели не спят) + array[1]>array[0], то вариантов тестов не так и много.

Вывод

Время работы O(nlog(n)) + O(n) -> TimSort() + перебор всех N элементов.

Задание 11

Описание

Дается N золотых слитков, нужно забрать как можно больше так, чтобы их суммарный вес был <= W.

Исходный код

```python
def max_gold(volume, base_ingots):
    ingots = [0] + base_ingots
    dict_ingots = {}

    for i in range(0, volume + 1):
        dict_ingots[(i, 0)] = 0
    for i in range(len(ingots)):
        dict_ingots[(0, i)] = 0

    for i in range(1, len(ingots)):
        for weight in range(1, volume + 1):
            dict_ingots[(weight, i)] = dict_ingots[(weight, i - 1)]

            if ingots[i] <= weight:
                val = dict_ingots[(weight - ingots[i], i - 1)] + ingots[i]

                if dict_ingots[(weight, i)] < val:
                    dict_ingots[(weight, i)] = val
    return max(dict_ingots.values())
```

Решение

Весь алгоритм строится на том, что мы просто перебираем почти все возможные варианты (тут используется мемоизация). Далее отбирается те, которые меньше необходимого веса, а в конце среди всех находится наибольшее значение.

Тесты

```
result = max_gold(10, [1, 4, 8])
```

```
9
```

Тесты из условия. Так как числа только положительные и в процессе перебираются все их варианты.

Вывод

Время работы 2 * O(n) + O(n^2). Не самый эффективный, но не самый сложный алгоритм. Кроме того, используется немало памяти, так как сохраняет в словаре все варианты.

Задание 12

Описание

Нужно проверить, можно ли разбить данный массив на 2 части, суммы значений которых будут равны.

Исходный код

```python
def splitter(array, n):
    left_sum = 0
    for i in range(0, n):
        left_sum += array[i]
    right_sum = 0
    for i in range(n - 1, -1, -1):
        right_sum += array[i]
        left_sum -= array[i]
        if right_sum == left_sum:
            return i

    return -1


def printer(arr, n):
    split_point = splitter(arr, n)
    if split_point == -1 or split_point == n:
        print(-1)
        return

    for i in range(0, n):
        if split_point == i:
            print("")
        print(arr[i], end=" ")

    print()
```

Решение

Достаточно простой алгоритм. Так как ai < i, то мы сначала находим сумму всего массива, после чего отнимаем от нее значения по порядку, начиная с конца, и добавляем их в right_sum. Если left_sum будет равно right_sum, то массив можно разбить на 2 части, если же правая сумма больше, то нельзя. Ну и функция printer для вывода.

Тесты

```
arr = [1, 2, 3, 3, 3, 2, 2, 1, 1, 6]
arr2 = [1, 2, 3]
printer(arr, len(arr))
printer(arr2, len(arr2))
```

```
1 2 3 3 3
2 2 1 1 6
1 2
3
```

Тест из условия + со множеством одинаковых элементов, идущих не по порядку, но с соблюдением условий ввода.

Вывод

Довольно простой, но эффективный алгоритм, работающий за O(2n).

Задание 17

Описание

Необходимо реализовать алгоритм, который находит количество возможных телефонных номеров длины N, которые не могут начинаться с 0 или 8.

Исходный код

```python
def f(n):
    array = [[0 for j in range(n + 1)] for i in range(10)]
    mod = 10 ** 9
    for i in range(10):
        array[i][1] = 1

    for num in range(2, n + 1):
        for k in range(10):
            match k:
                case 0:
                    array[0][num] = (array[4][num - 1] + array[6][num - 1]) % mod
                case 1:
                    array[1][num] = (array[6][num - 1] + array[8][num - 1]) % mod
                case 2:
                    array[2][num] = (array[9][num - 1] + array[7][num - 1]) % mod
                case 3:
                    array[3][num] = (array[8][num - 1] + array[4][num - 1]) % mod
                case 4:
                    array[4][num] = (array[0][num - 1] + array[3][num - 1] + array[9][num - 1]) % mod
                case 6:
                    array[6][num] = (array[0][num - 1] + array[1][num - 1] + array[7][num - 1]) % mod
                case 7:
                    array[7][num] = (array[6][num - 1] + array[2][num - 1]) % mod
                case 8:
                    array[8][num] = (array[1][num - 1] + array[3][num - 1]) % mod
                case 9:
                    array[9][num] = (array[2][num - 1] + array[4][num - 1]) % mod
```

```python
    total_sum = 0
    for i in range(1, 10):
        if i != 8:
            print("array: ", array[i][n])
            total_sum = (total_sum + array[i][n]) % mod

    return total_sum


print(f(5))
```

Решение

Пусть array[i][num] – количество номеров, набираемых ходом коня, которые начинаются с цифры i и состоят из num цифр. Тогда array[i][1]=1 для всех i, а array[i][num] для любого d вычисляется через сумму array[i] [num-1] для num>1. Так, например, array[4][num] = array[0][num-1]+array[3] [num-1]+array[9][num-1]. Увеличивая num от 2 до n мы получим значения array[i][n], сумма которых (за вычетом array[0][n] и b[8][n]) и даст ответ на поставленную задачу

Тесты

```
print(f(1))
print(f(2))
print(f(1000000))
```

```
8
16
622882816
```

Два теста из условия + тест с максимальным n, который смог пройти.

Вывод

Непростой алгоритм, завязанный на переборе всех вариантов для каждой цифры.

Задание 18

Описание

Открылось кафе, в котором работает система скидок – за каждый обед на сумму дороже 100р вам дается купон на 1 бесплатный обед. Вам попался прейскурант на n дней. Вы хотите ходить в кафе каждый день. Необходимо написать алгоритм, который рассчитает минимальную сумму, необходимую на все обеды, дни, в которые лучше потратить купоны, а также количество использованных купонов

Исходный код

```
def cafe(array):
    price = 0
    dinner_price = []
    coupon = []
    i = 0
    indexes = []
    while len(array) > 0:
        i += 1
        cur_price = array.pop()
        if len(dinner_price) < len(coupon):
            dinner_price.append(cur_price)
            indexes.append(i)
        else:
            if len(coupon) == 0:
                if cur_price > 100:
                    coupon.append(cur_price)
                else:
                    price += cur_price
                continue

            if cur_price < dinner_price[-1]:
                if cur_price > 100:
                    coupon.append(cur_price)
                else:
                    price += cur_price
```

```
            else:
                last_bought = dinner_price.pop()
                indexes.pop()
                dinner_price.append(cur_price)
                indexes.append(i)
                if last_bought > 100:
                    coupon.append(last_bought)
                else:
                    price += last_bought

    print(price + sum(coupon))
    print(len(coupon) - len(dinner_price), len(dinner_price))
    for i in indexes:
        print(i)


data = [110, 110, 110]
data2 = [110, 40, 110, 120, 60, 110, 120]
cafe(data[::-1])
cafe(data2[::-1])
```

Решение

Все решение строится на проверке текущей цены и сравнением ее с число 100 (чтобы определить, можно ли выдать купон) и переборе лучших вариантов для его реализации. В лист dinner_price отдельно сохраняются цены за отданный обед, а в indexes – номер дня, в который был (или мог быть) реализован купон. При более подходящем варианте старые значения стираются.

Тесты

```
data = [110, 110, 110]
data2 = [110, 40, 110, 120, 60, 110, 120]
```

```
220
1 1
3
430
1 2
4
7
```

Тесты из условия + немного усложненный вариант.

Вывод

Время выполнения O(n). Считаю эту задачу крайне показательной для представления динамического программирования. Она включает в себя все основные аспекты, по типу разбиения на подзадачи и т.п.

Выводы по проделанной работе

Было полезно детальнее разобраться в динамическом программировании, учитывая, что многие из этих задач являются классическими на собеседованиях (в целом как и большинство задач по данной дисциплине). В теории, данные знания уже могут быть применены на практике, учитывая, что многие задачи являются перебором всех вариантов с целью поиска наилучшего.