Министерство образования и науки Российской Федерации
Федеральное государственное автономное образовательное учреждение высшего образования «Санкт-Петербургский национальный исследовательский университет информационных технологий, механики и оптики»

Мегафакультет трансляционных информационных технологий
Факультет инфокоммуникационных технологий

Дисциплина: Алгоритмы и структуры данных

Отчет по Лабораторной работе №1

Выполнила: Микулина Алиса Романовна

Группа: К3143, 1 курс

Преподаватель: Харьковская Татьяна Александровна

Санкт-Петербург

24.05.2022

Описание задания

Задание 4.

Пишем волшебный код для сбора подписей. Есть несколько отрезков, нужно покрыть их наименьшим возможным количеством точек. Должно решаться сортировкой точек начала и конца.

Задание 7.

Необходимо найти максимальное количество сапог, которое сапожник сможет починить за фиксированный рабочий день.

Задание 11.

Упрощенный рюкзак. Нужно собрать как можно больше золота в сумку. Есть емкость сумки и вес слитков. Здравствуй, динамическое программирование!

Задание 12.

При заданной последовательности чисел нужно определить, возможно ли разбить ее на две последовательности так, чтобы их суммы были равны. Решается простым жадником.

Задание 13.

На первый взгляд задача похожа на предыдущую, однако нееет :)))) Тут нужно уже решать динамикой, а мозг уже плавится)) Нужно узнать, возможно ли разделить последовательности на три равные части.

Задание 16.

Задача на Эйлеров обход. Нужно обойти несколько городов, при этом пройдя минимальное расстояние. Не так проста как кажется, динамика сложная, пришлось использовать бит маски.

Задание 17.

Нужно написать код, который будет считать сколькими вариантами возможно набрать телефонный номер заданной длины ходом коня при условии, что номер не может начинаться с 0 или 8.

Задание 20.

Реализовать код, который будет считать, сколько в строке почти палиндромов. Дается слово и количество букв, которые могут отличаться. Надо как-то написать все так, чтобы работало и мозг не взорвался :)))

Задание 21.

Очень простая задача на три балла, в которой нужно понять, сможет ли игрок, имея данный набор карт, отбиться от списка карт, которые ему подкинули. (У меня есть аж три разных версии кода, и все работают идеально).

Описание решения и исходный код

Задача 4.

Эту задачу я решила далеко не первой, долго над ней мучалась, потому что концепт сортировки начала и конца всех отрезков очень неочевидный. Это вроде как и не жадник, а вроде как и жадник, но решается максимально наивно.

По сути просто берем все отрезки, присваиваем точкам начала и конца, собственно, циферки 1 и 0, закрытие и открытие соответственно. В таком порядке, потому что тогда sorted сможет их правильно отсортировать.

Логика говорит, что если после точки открытия стоит точка закрытия, то в точке закрытия надо ставить точку посещения, иначе отрезок, который только что открылся, не пороется.

Вроде логично, вроде работает, как и все жадники, но звучит подозрительно, конечно))))

```
def min_dots(lines):
    open_close = []
    # 0 = open
    # 1 = close
    for i in range(len(lines)):
        open_close.append((lines[i][0], 0))
        open_close.append((lines[i][1], 1))
    open_close = sorted(open_close)
    dots = []
    for i in range(len(open_close) - 1):
        action = open_close[i][1]
        dot_next = open_close[i + 1][0]
        action_next = open_close[i + 1][1]
        if action == 0 and action_next == 1:
            dots.append(dot_next)
    return dots




with open('input_4.txt') as f:
    num_lines = int(f.readline())
    lines = []
    for i in range(num_lines):
        new_line = list(map(int, f.readline().split()))
        lines.append((new_line[0], new_line[1]))

ans = min_dots(lines)
print(len(ans))
print(' '.join(str(x) for x in ans))
```

Задача 7.

Думаю, нет задачи в этой лабораторной, проще чем эта. Прочитав ее, я подумала, что тут буквально все решается сортировкой. Не поверила. Написала. Работает. Подозрительно)))

Но на самом деле это самое логичное решение, берем ботинки, которые нужно чинить меньше всего по времени – и Ура! – оно работает!

Даже не знаю что еще сказать)

```
def problem_solver(day_ln, num_boots, boots):
    ordered_boots = sorted(boots)
    how_many = 0
    time_taken = 0
    for i in range(num_boots):
        if time_taken + ordered_boots[i] >= day_ln:
            return how_many
        time_taken += ordered_boots[i]
        how_many += 1
        if how_many == num_boots:
            return how_many


with open ("input.txt") as f:
    day_ln, num_boots = f.readline().split()
    day_ln, num_boots = int(day_ln), int(num_boots)
    boots = f.readline().split()
    for i in range(num_boots):
        boots[i] = int(boots[i])

solution = problem_solver(day_ln, num_boots, boots)
print(solution)
```

Задача 11.

Обычный, стандартный, элементарный, так еще и упрощенный рюкзак. Кто с ним возился 4 часа? Правильно, я. Почему? Потому что динамика.

Это буквально первая задача на динамику, при решении которой я на полном серьезе поверила (пусть и временно) что я начинаю понимать, как работает динамика.

Концепт того, что суть динамики в том, что нужно решать микрозадачи, каждый раз обращаться к уже решенным и хранить информацию в матрице ну ооооочень сложен для моего мозга. На момент решения рюкзака я буквально не представляла от слова совсем как оно должно дышать и жить.

Спасибо человеку, сидевшему со мной все те 4 несчастных часа, объяснявшему мне прикол таблички и зачем она нужна.

На самом деле все просто. Мы делаем матрицу, наполненную ноликами, и начинаем ее потихоньку заполнять единицами. Если в ячейке единица – значит мы может собрать данную массу собрав данные элементы, если ноль – то нет. В результате, самая большая собранная сумма будет являться ответом.

На момент написания отчета я уже даже умею восстанавливать путь по элементам рюкзака, считаю огромным достижением, между прочим, что сама въехала как это делается.

```
def max_possible_value(max_value, num_ingots, ingots):
    ingots.append(0)
    ingots = sorted(ingots)
    weights = [[False for i in range(max_value + 1)] for j in range(num_ingots + 1)]
    weights[0][0] = True
    for i in range(1, num_ingots + 1):
        for j in range(max_value + 1):
            difference = j - ingots[i]
            if difference >= 0:
                if weights[i - 1][difference]:
                    weights[i][j] = True
            if weights[i - 1][j]:
                weights[i][j] = True
    ans = 0
    i = len(weights[0]) - 1
    while ans == 0 and i >= 0:
        if weights[-1][i] == True:
            ans = i
        i -= 1
    return ans




# with open('input_11.txt') as f:
#     max_value, num_ingots = f.readline().split()
#     max_value, num_ingots = int(max_value), int(num_ingots)
#     ingots = f.readline().split()
#     for i in range(num_ingots):
#         ingots[i] = int(ingots[i])


max_value, num_ingots = list(map(int, input().split(' ')))
ingots = list(map(int, input().split(' ')))
ans = max_possible_value(max_value, num_ingots, ingots)
print(ans)
```

Задача 12.

Казалось бы, жадники так просто решаются. Но в этом их проблема. Сидишь и смотришь, а точно так можно? Это вообще работает? Правда? Серьезно? То есть так правда можно и по-другому никак? Ну ладно…

Тут то же самое, метод-то элементарный, сортируем все имеющиеся элементы и добавляем поочередно их к массиву с наименьшей суммой. В теории если на два вообще можно разделить, оно разделится. Там есть какой-то коэффициент вероятности, но оно в большинстве случаев работает, а лучше еще не придумали, поэтому вот такое решение.

Работает, загадочно, ну да и ладно.

```
def separate_into_two(num_elms, elms):
    if sum(elms) % 2 != 0:
        return -1
    expected_sum = sum(elms) // 2
    first = []
    second = []
    elms = sorted(elms)
    for i in range(len(elms) - 1, -1, -1):
        if sum(first) < sum(second):
            first.append(elms[i])
        else:
            second.append(elms[i])
    if sum(first) == expected_sum and sum(second) == expected_sum:
        return f'-1 \n{first}'
    else:
        return -1



with open('input_12.txt') as f:
    num_elms = int(f.readline())
    elms  = list(map(int, f.readline().split()))

ans = separate_into_two(num_elms, elms)
print(ans)
```

Задача 13.

По описанию задача похожа на предыдущую, однако это не совсем так :) Это уже не жадник, а динамика, причем динамика с кубом.

Сколько времени мне на это понадобилось? Около пяти часов. Зато! оно живет и работает.

Принцип такой: i обозначает сколько сувениров мы уже рассмотрели, j – сумма, которую наберет первый чел, если возьмет сувенир, k – сумма, которую наберет второй чел, если возьмет этот сувенир. В принципе можно еще и третьего добавить, но какой смысл, если мы знаем, что он соберет треть, если и первые два соберут треть.

Идем, получается, вперед по массивчику, добавляем новые сувениры. В результате ответ будет лежать в ячейке, где i это число сувениров, а j и k – треть всей суммы сувениров. Если значение равно единице – можем собрать. Если нулю – то нет.

На словах все просто, но как же тяжело придумать этот подход. Наверное, это была самая сложная для понимания задача, на которой я и начала нормально понимать суть динамики.

```
def separate_into_three(num_suvs, suvs):
    suvs = sorted(suvs)
    if sum(suvs) % 3 != 0 or num_suvs < 3:
        return 0
    equal_cost = sum(suvs) // 3
    counts = [[[0 for k in range(equal_cost + 1)] for j in range(equal_cost + 1)] for
i in range(num_suvs + 1)]
    to_add = suvs[0]
    counts[0][to_add][0] = 1
    counts[0][0][to_add] = 1
    counts[0][0][0] = 1
    for i in range(0, num_suvs):
        for j in range(equal_cost + 1):
            for k in range(equal_cost + 1):
                to_add = suvs[i]
                if counts[i][j][k] == 1:
                    if j + to_add <= equal_cost:
                        counts[i + 1][j + to_add][k] = 1
                    if k + to_add <= equal_cost:
                        counts[i + 1][j][k + to_add] = 1
                    if i + 1 < num_suvs:
                        counts[i + 1][j][k] = 1

    return counts[num_suvs][equal_cost][equal_cost]




with open('input_13.txt') as f:
    num_suvs = int(f.readline())
    suvs  = list(map(int, f.readline().split()))

ans = separate_into_three(num_suvs, suvs)
print(ans)
```

Задача 16.

Вы хотели моей смерти? Наверное, да.

От этой задачи у меня еще осталась куча эмоций, ибо дорешала я ее около 12 часов назад, чему несказанно рада, ведь это значит, что ее мне больше НЕ НАДО РЕШАТЬ.

У меня есть три абсолютно разные версии этой задачи, одна не работает вообще, вторая работает криво, а третью я считаю просто гениальнейшим решением на планете, и аргументы не принимаются.

Почему оно такое гениальное? (просто я стирала его с лаской))))

На самом деле уставший от жизни мозг очень устал от огромных затрат памяти из кода первой версии, решил не дорешивать и пошел писать что-то кардинально новое.

Что же такое это новое? Бит маски. И если честно я еще не видела в питоне такой красивой и одновременно ужасной для понимания функции как абсолютно все функции хоть как либо связанные с двоичным счислением и бит масками. Да, красиво. Да, удобно. Но ЗА ЧТО такой синтаксис!?!?!?!

Ладно. Принцип решения заключается в том, что у нас есть табличка, где i это количество городов, а j это максимальное десятичное значение бит маски для данного количества городов.

Например, для пяти городов это 31, что записывается как 11111 в двоичном коде.

Каждая единичка значит, что челик уже побывал в городе. Таблица маленькая, так как на каждом этапе выбирается наикратчайший путь до данной точки. Так как в 10110, например, можно прийти множеством способов, то мы выбираем самый короткий из доступных путей, и записываем последнюю посещенную вершинку в значение вместе с пройденным кратчайшим путем. Это, как раз, нужно чтобы восстановить маршрут)

Таким образом перебираем все доступные пути, и в самом углу таблицы у нас будет лежать кратчайший путь. Теперь просто восстанавливаем маршрут с помощью инлексов, которые мы сохраняли, и все!!!!

Звучит просто, но это НЕ ПРОСТО. Это БОЛЬНО. Зато работает)))

Где-то на этом этапе я уже нормально так въехала в динамику и поняла, как решать палиндром. То есть to be continued))

```
from numpy import Inf
```

```python
def find_route(distances, cities, maximum):
    length_way = [Inf, []]
    max_possible = 2 ** cities - 1
    ways = [[[Inf, Inf] for j in range(max_possible + 1)] for i in range(cities)]
    for i in range(cities):
        path = 2 ** i
        ways[0][path] = [0, i]

    for i in range(cities):
        for k in range(max_possible + 1):
            if ways[i][k][0] != Inf:
                for j in range(cities):
                    if (k & (2 ** j)) == 0 and (k + 2 ** j) <= max_possible:
                        current = ways[i + 1][k + 2 ** j][0]
                        new = ways[i][k][0] + distances[ways[i][k][1]][j]
                        if current > new:
                            ways[i + 1][k + 2 ** j][0] = new
                            ways[i + 1][k + 2 ** j][1] = j

    length_way[0] = ways[cities - 1][max_possible][0]
    way = []
    way.append(ways[cities - 1][max_possible][1])
    j = max_possible
    for i in range(cities - 1, 0, -1):
        j = j - 2 ** way[-1]
        to_add = ways[i - 1][j][1]
        way.append(to_add)

    for i in range(len(way)):
        way[i] = way[i] + 1

    length_way[1] = way

    return length_way




with open ("input_16.txt") as f:
    cities = int(f.readline())
    distances  = []
    maximum = 0
    for i in range(cities):
        line = f.readline()
        line_split = list(map(int, line.split()))
        distances.append(line_split)
        if max(line_split) > maximum:
            maximum = max(line_split)

ans = find_route(distances, cities, maximum)
print(ans)
```

Задача 17.

Вроде и задача на динамику, но даже какая-то смешная. Почти как задачка с монетками, только чуть-чуть пошире. Самое сложное, без шуток, было прописать возможные шаги во все стороны для каждой цифры)

Нет, я конечно, не выделываюсь, я еще честно пыталась решить эту задачу рекурсией, еще до того, как вспомнила, что вся лаба на динамику))))

Короче суть простая, просто идем вперед по табличке и делаем шаги вперед, сохраняем значения, обращаемся к предыдущим и т. д.…. В общем, все весьма просто) Ничего нового тот не расскажу, все буквально как в самой банальной динамике.

```
def phone_numbers(count):
    steps = {1: [6, 8], 2: [7, 9], 3: [8, 4], 4: [9, 3, 0], 5: [], 6: [1, 7, 0], 7:
[2, 6], 8: [1, 3], 9: [4, 2], 0: [4, 6]}
    numbers = [[0 for i in range(10)] for j in range(count + 1)]
    for i in range(10):
        if i == 0 or i == 8:
            numbers[0][i] = 0
        else:
            numbers[0][i] = 1
    for i in range(count):
        for j in range(10):
            for k in steps[j]:
                numbers[i + 1][k] += numbers[i][j]
    return sum(numbers[-2])



n = int(input())
ans = phone_numbers(n)
print(ans)
```

Задача 20.

У меня, опять таки, две версии решения этой задачи, одно нединамическое и неотдебажено, поэтому такое себе, а второе настолько прекрасно, что смотреть приятно.

Суть в том, что прорисовав раза 4 динамику на листочке, я заметила, что начиная с третьей строки, мы начинаем обращаться за информацией ячейке, которая хранит в себе, сколько ошибок в строке короче на два символа, при этом сравниваем крайние новые два символа.

Таким образом, если новые символы одинаковые, то значение ошибок остается. Если нет – то прибавляем единичку.

Эта задача даже прошла до 11 теста спокойно на acmp, что уже меня шокировало. Та сайте очень маленькое ограничение по памяти, не рассчитанное на динамику, поэтому не влезло решение, а так оно ну прям прекрасно.

Горжусь тем, что к этому моменту научилась видеть динамику в задачах вообще без подсказок. Ну и не удивительно, это моя последняя решенная задача.

```
def count_palindromes(length, max_diff, line):
    letters = []
    counter = 0
    for letter in line:
        letters.append(letter)
    samples = [[0 for i in range(length)] for j in range(length)]
    for i in range(length):
        for j in range(length):
            if length - j <= i:
                samples[i][j] = None
            elif i == 0:
                samples[i][j] = 0
                counter += 1
            elif i == 1:
                if letters[j] == letters[j + 1]:
                    samples[i][j] = 0
                    counter += 1
                else:
                    samples[i][j] = 1
                    if 1 <= max_diff:
                        counter += 1
            else:
                middle = samples[i - 2][j + 1]
                if letters[j] == letters[j + i]:
                    samples[i][j] = middle
                    if middle <= max_diff:
                        counter += 1
                else:
                    samples[i][j] = middle + 1
                    if middle + 1 <= max_diff:
                        counter += 1

    return counter



with open('input.txt') as f:
    length, max_diff = list(map(int, list(f.readline().split())))
    line = f.readline()
```

```
ans = count_palindromes(length, max_diff, line)
print(ans)

with open('output.txt', 'w') as d:
    d.write(str(ans))
```

Задача 21.

Во-первых, задача на удивление элементарная для трех баллов.

Во-вторых, я думала тут есть подвох, а его не оказалось.

В-третьих, у меня есть три абсолютно разные версии этого кода, все из которых теперь проходят тесты, с которыми я сидела около 6 часов.

Все задачи падали на одном тесте. На девятом. В течение пяти часов, так как первая версия кода была готова уже через час.

Но все падало на девятом тесте.

А проблема была в том, что я значения чисел считывала как строки и сравнивала их, и заметила это абсолютно случайно. Как только я это заметила, сразу все заработало, все три версии прошли тесты на acmp…

На этом этапе я морально умерла)))

```
def win(num_my, num_beat, my_cards, to_beat, kos):
    if num_my < num_beat:
        return 'NO'
    elif num_my == 0:
        return 'NO'
    elif len(my_cards[kos]) < len(to_beat[kos]):
        return 'NO'
    elif (len(my_cards[kos]) == 0) and (len(to_beat[kos]) != 0):
        return 'NO'
    while len(to_beat[kos]) != 0:
        cur_beat = to_beat[kos][0]
        beat = 0
        for i in range(len(my_cards[kos])):
            if my_cards[kos][i] > cur_beat:
                to_beat[kos].pop(0)
                my_cards[kos].pop(i)
                beat = 1
                break
        if beat == 0:
            return 'NO'
    types = ['S', 'C', 'D', 'H']
```

```
    for type in types:
        while len(to_beat[type]) != 0:
            cur_beat = to_beat[type][0]
            beat = 0
            if len(my_cards[type]) > 0:
                for i in range(len(my_cards[type])):
                    if my_cards[type][i] > cur_beat:
                        to_beat[type].pop(0)
                        my_cards[type].pop(i)
                        beat = 1
                        break
            if beat == 0:
                if len(my_cards[kos]) > 0:
                    for i in range(len(my_cards[kos])):
                        to_beat[type].pop(0)
                        my_cards[kos].pop(i)
                        beat = 1
                        break
            if beat == 0:
                return 'NO'
    beat_remain = 0
    for type in types:
        beat_remain += len(to_beat[type])
    if beat_remain == 0:
        return 'YES'
    else:
        return 'NO'



def reform(cards):
    equals = {'6': 6, '7': 7, '8': 8, '9': 9, 'T': 10, 'J': 11, 'Q': 12, 'K': 13,
'A': 14}
    reformed_cards = {'S': [], 'C': [], 'D': [], 'H': []}
    for i in range(len(cards)):
        reformed_cards[cards[i][1]].append(equals[cards[i][0]])
    for name, values in reformed_cards.items():
        sorted_values = sorted(values)
        reformed_cards[name] = sorted_values
    print(reformed_cards)
    return reformed_cards



with open('input.txt') as f:
    num_my, num_beat, kos =  list(f.readline().split())
    my_cards  = list(f.readline().split())
    to_beat  = list(f.readline().split())
    my_cards = reform(my_cards)
    to_beat = reform(to_beat)
```

```
ans = win(num_my, num_beat, my_cards, to_beat, kos)

with open('output.txt', 'w') as d:
    d.write(ans)
```

Описание проведенных тестов.

4

Input	Output
3 1 3 2 5 3 6	1 3
4 4 7 1 3 2 5 5 6	2 3 5

7

Input	Output
10 3 6 2 8	2
3 2 10 20	0

11

Input	Output
10 3 1 4 8	9
209 38 16 21 21 96 129 144 159 253 254 259 259 267 285 290 304 351 351 383 411 429 493 494 527 530 534 596 619 625 692 717 727 727 745 772 833 853 856 946	202

12

Input	Output
3 1 2 3	3 [2, 1]

13

Input	Output
4 3 3 3 3	0
1 40	0
11 17 59 34 57 17 23 67 1 18 2 59	1
13 1 2 3 4 5 5 7 7 8 10 12 19 25	1

16

Input	Output
5 0 183 163 173 181 183 0 165 172 171 163 165 0 189 302 173 172 189 0 167 181 171 302 167 0	666 [4, 5, 2, 3, 1]
5 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0	0 [5, 4, 3, 2, 1]
5 0 200 200 200 200 200 0 200 200 200 200 200 0 200 200 200 200 200 0 200 200 200 200 200 0	800 [5, 4, 3, 2, 1]
5 0 200 200 200 200 200 0 200 200 200 200 200 0 200 200 200 **100** 200 0 200 200 200 200 200 0	700 [5, 3, 2, 4, 1]

17

Input	Output
1	8
2	16
20	46157824

20

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
17196078	24.05.2022 6:46:56	Микулина Алиса Романовна	0268	Python	Memory limit exceeded	11	0,171	32 Мб
17195802	24.05.2022 0:51:58	Микулина Алиса Романовна	0268	Python	Memory limit exceeded	11	0,203	31 Мб

21

ID	Дата	Автор	Задача	Язык	Результат	Тест	Время	Память
17196084	24.05.2022 6:48:47	Микулина Алиса Романовна	0698	Python	Accepted		0,046	362 Кб
17196082	24.05.2022 6:48:32	Микулина Алиса Романовна	0698	Python	Wrong answer	9	0,046	354 Кб
17165726	17.05.2022 23:31:10	Микулина Алиса Романовна	0698	Python	Accepted		0,062	358 Кб
17165724	17.05.2022 23:30:15	Микулина Алиса Романовна	0698	Python	Accepted		0,046	350 Кб
17165721	17.05.2022 23:26:34	Микулина Алиса Романовна	0698	Python	Wrong answer	9	0,031	362 Кб
17165716	17.05.2022 23:23:36	Микулина Алиса Романовна	0698	Python	Wrong answer	9	0,046	350 Кб
17165715	17.05.2022 23:22:33	Микулина Алиса Романовна	0698	Python	Wrong answer	9	0,031	354 Кб
17165710	17.05.2022 23:18:59	Микулина Алиса Романовна	0698	Python	Wrong answer	9	0,031	362 Кб

Выводы по проделанной работе.

Динамическое программирование всегда казалось очень страшным, неприступным и невозможным. Вообще, это очень интересный и красивый подход, но только после того как решить несколько задач, пропсиховаться, покричать в бездну, поплакать, поныть кому-нибудь в плечо, решить еще несколько задач, нарисовать кучу картинок и ОСОЗНАТЬ)))

Мне кажется, я если не поняла, то очень близка к пониманию этого подхода. По крайней мере, я начала видеть способы решения задач динамикой, а раньше вообще не могла.

Жадники это вообще отдельная тема, они работают настолько прямолинейно, что даже не верится, и поэтому страшно что неправильно, хотя правильно)