Санкт-Петербургский Национальный Исследовательский Университет
Информационных Технологий, Механики и Оптики

Факультет инфокоммуникационных технологий

Лабораторная работа №3

Выполнили:

Петрова Н. Г

Садовая А. Р

Оншин Д. Н

Проверил:

Мусаев А.А.

Санкт-Петербург,

2023

СОДЕРЖАНИЕ

Стр.

ВВЕДЕНИЕ

Данная работа представляет собой отчет о выполненных заданиях:

1. Задача 1 с использованием жадного алгоритма.
2. Задача 2 с использованием жадного алгоритма.
3. Алгоритм Дейкстры для графа городов России.

Жадные алгоритмы являются одним из наиболее распространенных подходов к решению различных задач. Они основываются на принципе максимизации выгоды на каждом шаге решения задачи, не обращая внимание на будущие последствия. Этот подход может показаться неразумным, но на практике жадные алгоритмы зачастую демонстрируют высокую эффективность и скорость работы, что делает их одними из самых популярных методов решения задач.В этом контексте важно понимать, какие задачи можно решить с помощью жадных алгоритмов, и какие нюансы необходимо учитывать при их применении. В данном отчёте будут продемонстрированы примеры решения подобных задач.

1 Задача 1

```
def task_1():
    ms1_str = input('Input m1 and s1: ')
    ms1 = str_to_list(ms1_str)
    ms2_str = input('Input m2 and s2: ')
    ms2 = str_to_list(ms2_str)
    ms3_str = input('Input m3 and s3: ')
    ms3 = str_to_list(ms3_str)
    ms4_str = input('Input m4 and s4: ')
    ms4 = str_to_list(ms4_str)
    n = int(input('Input n: '))
    n_old = n
    n_dict = {}
    coins = {}
    values = [ms1[1], ms2[1], ms3[1], ms4[1]]
    values.sort()
    values.reverse()
    for i in range(len(values)):
        if values[i] == ms1[1]:
            coins[ms1[1]] = ms1[0]
        elif values[i] == ms2[1]:
            coins[ms2[1]] = ms2[0]
        elif values[i] == ms3[1]:
            coins[ms3[1]] = ms3[0]
        elif values[i] == ms4[1]:
            coins[ms4[1]] = ms4[0]
```

Рисунок 1.1 — Код функции для задачи 1

Сначала выбирается монета наибольшего номинала, которую можно использовать для получения сдачи. Затем выбирается монета наибольшего номинала из оставшихся, которую можно использовать, чтобы получить оставшуюся сдачу, и так далее, пока не будет получена вся сдача.

```
for i in coins:
    if n == 0:
        break
    k = n // i
    if k <= coins[i]:
        n_dict[i] = k
        n -= i * k
    else:
        n_dict[i] = coins[i]
        n -= i * coins[i]
s = 0
for i in n_dict:
    s += i * n_dict[i]
if s == n_old:
    print(n_dict)
else:
    print('There is less money, than it needs!')
```

Рисунок 1.2 — Продолжение кода функции для задачи 1

```
Input task number: 1
Input m1 and s1: 10 10
Input m2 and s2: 10 5
Input m3 and s3: 10 2
Input m4 and s4: 10 1
Input n: 34
{10: 3, 5: 0, 2: 2}
```

Рисунок 1.3 — Пример вывода функции для задачи 1

2 Задача 2

```
def task_2():
    exhibits = {'Mona Lisa': '1 10', 'David': '20 15', 'Madonna': '1 7', 'Peacock': '10 20',
                '3': '12 50', '4': '3 1', '5': '16 23', '6': '2 5'} #exhibits: weight price
    while True:
        n = int(input(f'Input N (not more than {len(exhibits)}): '))
        if n <= len(exhibits):
            break
        else:
            print('Wrong input!')
    k = int(input('Input K: '))
    m = int(input('Input M: '))
    loot = {}
    #exhibits = {1: 10, 20: 15, 1: 7, 10: 20, 3: 5, 10: 10, 3: 30}
    whole_price = 0
    for i in range(m):
        if n == 0:
            break
        k_new = k
        while True:
            max_price = 0
            exh = ''
            weight = 0
```

Рисунок 2.1 — Код функции для задачи 2

```
            for j in exhibits:
                price = int(exhibits[j][-2] + exhibits[j][-1])
                if max_price < price and int(exhibits[j][0] + exhibits[j][1]) <= k_new:
                    max_price = price
                    exh = j
                    weight = int(exhibits[j][0] + exhibits[j][1])
            if exh == '':
                break
            if weight < k_new:
                k_new -= weight
                whole_price += max_price
                n -= 1
                exhibits.pop(exh)
                loot[exh] = str(weight) + ' ' + str(max_price)
            else:
                whole_price += max_price
                n -= 1
                exhibits.pop(exh)
                loot[exh] = str(weight) + ' ' + str(max_price)
                break
    print(whole_price)
    print(loot)
```

Рисунок 2.2 — Продолжение кода функции для задачи 2

```
Input task number: 2
Input N (not more than 13): 11
Input K: 4
Input M: 7
68
{'Eggs': '3 30', 'Mona Lisa': '1 10', '2': '3 10', 'Madonna': '1 7', 'Dance': '3 5', '6': '2 5', '4': '3 1'}
```

Рисунок 2.3 — Пример вывода функции для задачи 2

3 Вывод по результатам решения двух задач

В обоих заданиях можно применить жадный алгоритм для получения приближенного решения. Однако, не всегда наилучшие решения на каждом шаге приводят к наилучшему конечному результату.

В задаче о даче сдачи, жадный алгоритм приносит наибольшую пользу, так как одним из уловий является наименьшее число монет. Однако, если монеты имеют дробные номиналы, то жадный алгоритм может дать неверный результат, что только усложнит решение задачи. В данной задаче логично использовать жадный алгоритм, так как он позволяет решать задачу быстро и эффективно.

В задаче о воре в музее жадный алгоритм может быть применим в некоторых случаях, но не всегда он дает наилучший результат. Он может быть применен, если мы рассматриваем экспонаты только по одному критерию, например, только по их стоимости или только по их весу. Тогда мы можем отсортировать экспонаты по этому критерию и начинать выбирать наиболее дорогие или наиболее легкие экспонаты, пока не достигнем максимального веса, который может унести вор.Однако, если мы рассматриваем экспонаты по двум и более критериям, то применение жадного алгоритма может привести к неверному решению. Однако, если мы рассматриваем экспонаты по двум и более критериям, то применение жадного алгоритма может привести к неверному решению. Например, если музей содержит экспонаты с разной ценностью и разным весом, то при выборе экспонатов для кражи мы должны учитывать оба этих критерия. В таком случае жадный алгоритм может привести к выбору набора экспонатов с максимальной стоимостью, но слишком большим весом, который вор не сможет унести. Или наоборот, к выбору набора экспонатов с меньшей стоимостью, но достаточно легких для транспортировки. Самым оптимальным алгоритмом для решения данной задачи было бы рассчитать отношение цены к весу для каждого экспоната и отсортировать экспонаты в порядке убывания отношения цены к весу, таким образом вор будт извлекать наибольшую выгоду от каждого захода.

Таким образом, необходимо учитывать возможные исключения и недостатки жадного алгоритма при его применении, особенно в задачах с нестандартными условиями.

4 Задача 4

```
def task_4():
    def arg_min(t, s):
        amin = -1
        m = max(t)
        for i in range(len(t)):
            if t[i] < m and i not in s:
                m = t[i]
                amin = i
        return amin

    print('0 - Moscow, 1 - St. Petersburg, 2 - Kazan, 3 - Yekaterinburg, 4 - Novos
    cities = [[  0,  635,  815,    0,    0,    0,    0,    0,    0, 410,   0],
              [635,    0, 1208,    0,    0,    0,    0,    0,    0,   0,   0],
              [815, 1208,    0,  892,    0,    0,    0,    0,    0, 419,   0],
              [  0,    0,  892,    0, 1498,    0,    0,    0, 1417,   0,   0],
              [  0,    0,    0, 1498,    0, 6147,    0, 1569,  641,   0,   0],
              [  0,    0,    0,    0, 6147,    0, 4789,    0,    0,   0,   0],
              [  0,    0,    0,    0,    0, 4789,    0, 1060,    0,   0,   0],
              [  0,    0,    0,    0, 1569,    0, 1060,    0, 1422,   0,   0],
              [  0,    0,    0, 1417,  641,    0,    0, 1422,    0,   0,   0],
              [410,    0,  419,    0,    0,    0,    0,    0,    0,   0, 453],
              [  0,    0,    0,    0,    0,    0,    0,    0,    0, 453,   0]]
    n = len(cities)
    t = [math.inf] * n #список с кратчайшими путями
    v = int(input('Input start: '))
    s = [v] #список с просмотренными вершинами
    t[v] = 0
```

Рисунок 4.1 — Код функции для задания 4

```
    n = len(cities)
    t = [math.inf] * n #список с кратчайшими путями
    v = int(input('Input start: '))
    s = [v] #список с просмотренными вершинами
    t[v] = 0
    while v != -1:
        for i in range(len(cities[v])):
            if cities[v][i] > 0:
                if i not in s:
                    t[i] = min(t[i], t[v] + cities[v][i])
        v = arg_min(t, s)
        if v >= 0:
            s.append(v)
    print(t)
```

Рисунок 4.2 — Продолжение кода функции для задания 4

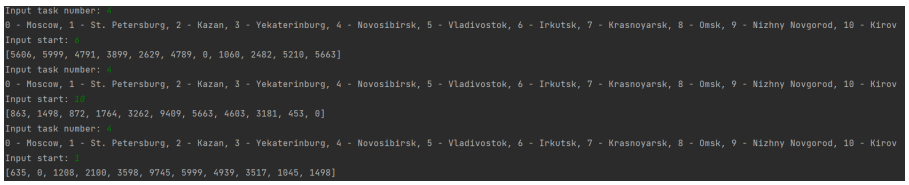

Рисунок 4.3 — Пример вывода функции для задания 4

Вывод Данный код реализует алгоритм Дейкстры для поиска кратчайшего пути в графе (граф содержит в себе расстояние между городами России). Функция ищет в списке t индекс элемента с наименьшим значением, который еще не был просмотрен и не находится в списке s. Список "cities"представляет собой матрицу смежности, где элемент с индексом [i][j] содержит расстояние между городами i и j. Если расстояние между городами неизвестно, то соответствующий элемент матрицы равен 0. Переменная n содержит количество городов в графе. Список t содержит кратчайшие расстояния от начальной вершины до каждой вершины графа. Изначально все элементы списка равны бесконечности, кроме начальной вершины, расстояние до которой равно 0. Переменная v содержит индекс начальной вершины, вводится пользователем. В начале алгоритма начальная вершина добавляется в список s, а расстояние до нее становится равным 0. Затем в цикле происходит обновление значений списка t. Для каждой вершины, смежной с текущей, если ее индекс не находится в списке s, то ее расстояние до начальной вершины обновляется, если это возможно, с помощью формулы: t[i] = min(t[i], t[v] + cities[v][i]). Таким образом, значение t[i] становится равным либо предыдущему значению t[i], либо значению t[v] + cities[v][i].

Затем снова выбирвется новая вершина и добавляется в список s. Если все вершины уже были просмотрены, то значение переменной v становится равным -1, что приводит к завершению цикла.

В конце алгоритма на экран выводится список t, в котором содержится кратчайшее расстояние от начальной вершины до каждой вершины графа. Каждый элемент списка соответствует индексу вершины.

ЗАКЛЮЧЕНИЕ

Таким образом, для каждой задачи были написаны программы на языке программирования Python. Были изучены жадные алгоритмы и рассмотрены их достоинства и недостатки. Для каждого алгоритма приведены программы и их вывод.

Все программы можно найти на репозитории в GitHub [1].

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. GitHub [Электронный ресурс]: `https://github.com/NatalyaPetrova/Algoritms` (дата обращения 20.03.2023).