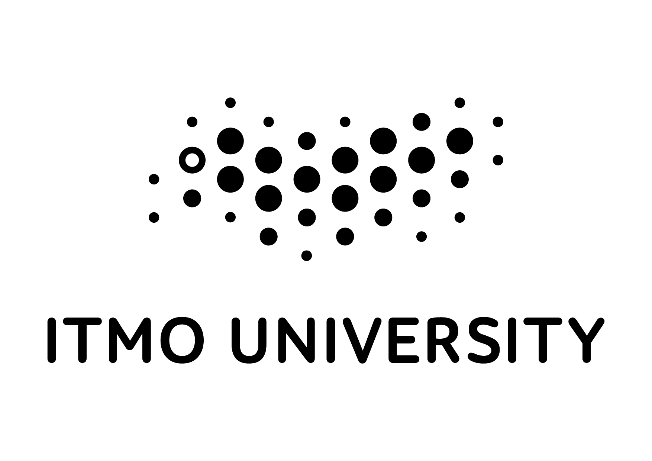
Санкт-Петербургский национальный исследовательский университет ИТМО

Факультет систем управления и робототехники



Алгоритмы и структуры данных.

Отчет №5.

Выполнил студент гр. R32362

Лаптев Максим Сергеевич

Санкт-Петербург 2023

**Содержание**

1. Цель
2. Задача №1067 – Структура папок
3. Задача №1650 – Миллиардеры
4. Задача №1080 – Раскраска карты
5. Вывод

**Цель**

Решить данные задачи, написав код, работа которого удовлетворяет условиям

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

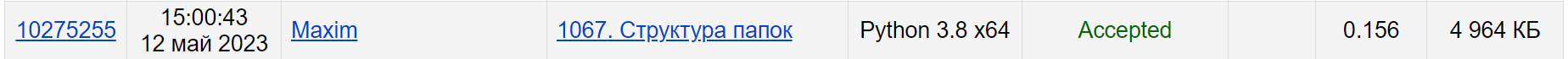
Исходный код:

n = int(input())  
cities = dict()  
people = []  
max\_cities = dict()  
for i in range(n):  
 name, city, money = input().split()  
 money = int(money)  
 people.append([name, city, money])  
 cities[city] = cities.get(city, 0) + money  
m, k = map(int, input().split())  
moves = []  
for i in range(k):  
 day, name, city\_move = input().split()  
 moves.append((int(day), name, city\_move))  
for i in range(1, m+1):  
 max\_sum = 0  
 flag = False  
 for city, value in cities.items():  
 if value == max\_sum:  
 flag = False  
 if value > max\_sum:  
 max\_sum = value  
 max\_city = city  
 flag = True  
 if flag: max\_cities[max\_city] = max\_cities.get(max\_city, 0) + 1  
 count = 0  
 for j in range(len(moves)):  
 if moves[j][0] == i:   
 count += 1  
 for human in people:  
 if human[0] == moves[j][1]:  
 cities[human[1]] -= human[2]  
 to\_city = moves[j][2]  
 human[1] = to\_city  
 cities[to\_city] = cities.get(to\_city, 0) + human[2]  
 break  
 else:  
 break  
 for \_ in range(count):  
 moves.pop(0)  
for city in sorted(max\_cities.keys()):  
 print(city, max\_cities[city])

Ограничения небольшие, времени много, так что задача скорее на реализацию – вопрос в том, где хранить данные. В python есть словари, многократной вложенностью получаем имитацию файловой системы – в каждой папке (словаре) свои подпапки (также словари) и тд. Если путь есть, доходим до его конца и создаем новую «ветку». В конце, когда вся файловая система заполнена, просто выводим результат.

Язык программирования: python

Результат



Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

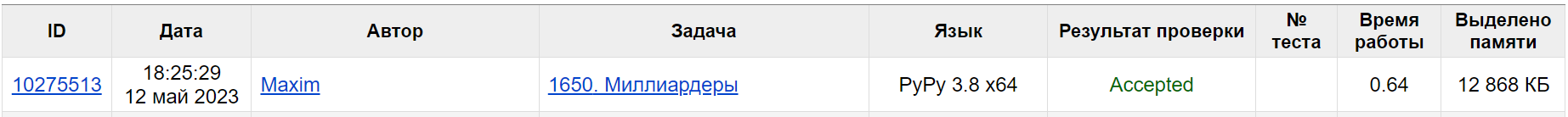
Исходный код:

n = int(input())  
cities = dict()  
people = dict()  
max\_cities = dict()  
for i in range(n):  
 name, city, money = input().split()  
 money = int(money)  
 people[name] = [city, money]  
 cities[city] = cities.get(city, 0) + money  
  
m, k = map(int, input().split())  
prev\_day = 0  
for i in range(k):  
 day, name, city\_move = input().split()  
 day = int(day)  
 human = people.get(name)  
 if day != prev\_day:  
 max\_sum = 0  
 for city, value in cities.items():  
 if value == max\_sum:  
 flag = False  
 elif value > max\_sum:  
 max\_sum = value  
 max\_city = city  
 flag = True  
 if flag: max\_cities[max\_city] = max\_cities.get(max\_city, 0) + (day - prev\_day)  
  
 cities[human[0]] -= human[1]  
 if cities[human[0]] == 0:  
 del cities[human[0]]  
 human[0] = city\_move  
 cities[city\_move] = cities.get(city\_move, 0) + human[1]  
  
 prev\_day = day  
  
max\_sum = 0  
for city, value in cities.items():  
 if value == max\_sum:  
 flag = False  
 elif value > max\_sum:  
 max\_sum = value  
 max\_city = city  
 flag = True  
if flag: max\_cities[max\_city] = max\_cities.get(max\_city, 0) + (m - prev\_day)  
  
for city in sorted(max\_cities.keys()):  
 print(city, max\_cities[city])

Хранить данные также будем в словарях – города, люди и ответ (города с максимальной посещаемостью). Заполняем входными данными и при вводе путешествий начинаем алгоритм. Каждый раз, когда меняется К, будем вести подсчёт какой город сейчас лидирует (для этого находим 2 максимальных). И на следующий итерации добавляем этому городу дни, когда он лидировал в словарь с ответом.

Язык программирования: python

Результат



Изображение выглядит как текст, снимок экрана, Шрифт, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, Прямоугольник

Автоматически созданное описание

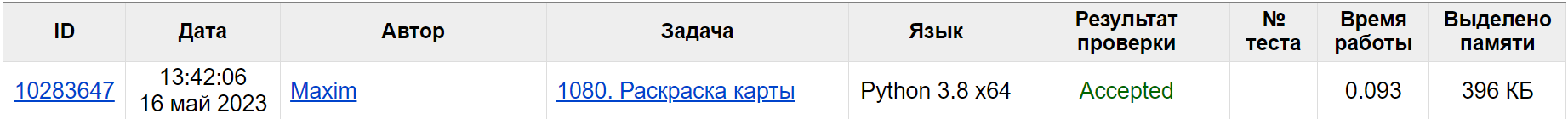
Исходный код:

def bfs(graph, node):  
 queue.append(node)  
  
 while queue:  
 m = queue.pop(0)  
  
 for neighbour in graph[m]:  
 if colors[neighbour] == -1:  
 colors[neighbour] = 1 if colors[m] == 0 else 0  
 queue.append(neighbour)  
 else:  
 if colors[neighbour] == colors[m]:  
 return False  
 return True  
  
  
n = int(input())  
neighbors = dict()  
colors = dict()  
for i in range(n):  
 x = input().split()[:-1]  
 for el in x:  
 el = int(el)  
 if neighbors.get(i + 1, False):  
 neighbors[i + 1].append(el)  
 else:  
 neighbors[i + 1] = [el]  
  
 if neighbors.get(el, False):  
 neighbors[el].append(i + 1)  
 else:  
 neighbors[el] = [i + 1]  
 if x == [] and not neighbors.get(i + 1, False):  
 neighbors[i + 1] = []  
 colors[i + 1] = -1  
  
queue = []  
colors[1] = 0  
if bfs(neighbors, 1):  
 for i in range(1, n + 1):  
 print(colors[i], end="")  
else:  
 print(-1)

Для решения использовался простейший поиск в ширину. С одним лишь отличием – окрашиваем вершину в противоположный цвет предыдущей вершины. Таким образом, соседи окрашиваются в разные цвета. Однако если мы встретим двух соседей с одинаковой окраской, значит покраска невозможна. Возвращаем -1, иначе выводим ответ из словаря colors.

Язык программирования: python

Результат:



**Вывод**

В результате проделанной работы были успешно решены 3 задачи, используя язык программирования python . Работа программ выдает правильный ответ, а также удовлетворяет условиям задачи (проходит по времени и памяти).