Министерство цифрового развития, связи и массовых коммуникаций
Ордена трудового Красного Знамени федеральное государственное бюджетное
образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Отчет по курсовой работе

по дисциплине «Структуры и алгоритмы обработки данных»

Выполнил: студент группы

Бст1904

Самарина А.В.

Москва, 2021

Задача 1. «Треугольник с максимальным периметром» Массив A состоит из целых положительных чисел - длин отрезков. Составьте из трех отрезков такой треугольник, чтобы его периметр был максимально возможным. Если невозможно составить треугольник с положительной площадью - функция возвращает 0.

```
def maxPerimeter(arr):
   maxi = 0
   n = len(arr)
   arr.sort(reverse = True)
   for i in range(0, n - 2):
      if arr[i] < (arr[i + 1] + arr[i + 2]):
         maxi = max(maxi, arr[i] +
                  arr[i + 1] + arr[i + 2])
         break

   if(maxi == 0):
      return "0"
   else:
      return  str(maxi)

print(maxPerimeter([3,5,7,2]))
```

15

Задача 2. «Максимальное число» Дан массив неотрицательных целых чисел nums. Расположите их в таком порядке, чтобы вместе они образовали максимально возможное число.

```
class MaxNumKey(str):
   def __lt__(x, y):
      return x+y > y+x

class Solution:
   def maxNumber( nums):
      max_num = ''.join(sorted(map(str, nums), key=MaxNumKey))
      return '0' if max_num[0] == '0' else max_num

print(Solution.maxNumber([10,11,0,7]))
```

711100

Задача 3. «Сортировка диагоналей в матрице» Дана матрица mat размером m * n, значения целочисленные. Напишите функцию, сортирующую каждую диагональ матрицы по возрастанию и возвращающую получившуюся матрицу.

```
def sortir(mat):
   m, n = len(mat), len(mat[0])
   t = [[] for i in range(m+n)]
```

```
    # Добавляем каждую диагональ в массив t
    for i in range(m):
        for j in range(n):
            t[i - j].append(mat[i][j])
    # Сортируем каждую диагональ массива
    for line in t:
        line.sort(reverse = True)
    # "Линии" в диагонали матрицы
    for i in range(m):
        for j in range(n):
            # pop() удаляет последний элемент массива и возвращает его
            mat[i][j] = t[i-j].pop()
    return mat


arr = [[2,3,1,1], [2,3,1,2], [1,3,1,2]]
print("Matrix:")
for l in arr:
    print(l)
print("\nSorted: ")
arr = sortir(arr)
for l in arr:
    print(l)
```

```
Matrix:
[2, 3, 1, 1]
[2, 3, 1, 2]
[1, 3, 1, 2]

Sorted:
[1, 1, 1, 1]
[2, 2, 2, 2]
[1, 3, 3, 3]
```

Задача 4 «Стопки монет»

На столе стоят 3n стопок монет. Вы и ваши друзья Алиса и Боб забираете стопки монет по следующему алгоритму:

1. Вы выбираете 3 стопки монет из оставшихся на столе.
2. Алиса забирает себе стопку с максимальным количеством монет.
3. Вы забираете одну из двух оставшихся стопок.
4. Боб забирает последнюю стопку.
5. Если еще остались стопки, то действия повторяются с первого шага.

Напишите функцию, возвращающую максимальное число монет, которое вы можете получить.

```
def max_coins(arr):
```

```
    arr.sort()
    n=len(arr)//3
    res = 0
    for i in range (n,len(arr),2):
        res+=arr[i]
    return res

print(max_coins([2,4,7,6,8,3]))
print(max_coins([2,7,3]))
```

11

3

Задача 5 «Шарики и стрелы»

Некоторые сферические шарики распределены по двухмерному пространству. Для каждого шарика даны x координаты начала и конца его горизонтального диаметра. Так как пространство двумерно, то y координаты не имеют значения в данной задаче. Координата xstart всегда меньше xend.

Стрелу можно выстрелить строго вертикально (вдоль yоси) из разных точек x оси. Шарик с координатами xstart и xend уничтожается стрелой, если она была выпущена из такой позиции x, что xstart $\leqslant$ x $\leqslant$ xend. Когда стрела выпущена, она летит в пространстве бесконечное время (уничтожая все шарики на пути).

Напишите функцию, возвращающую минимальное количество стрел, которые нужно выпустить, чтобы уничтожить все шарики

```
class Solution:
    def arrowShots(points) -> int:
        if not points: return 0
        points.sort()
        prev=points[0]
        total=1
        for s,e in points[1:]:
            if s>prev[1]:
                total+=1
                prev=[s,e]
            else:
                prev[1]=min(prev[1],e)

        return total

print("[[10, 16],[2,8],[1,6],[7,12]] -->", Solution.arrowShots([[10, 16],[2,8],[1,6],[7,12]]))

print("\n[[1,2],[3,4],[5,6],[7,8]] -->", Solution.arrowShots([[1,2],[3,4],[5,6],[7,8]]))
```

```
[[10, 16],[2,8],[1,6],[7,12]] --> 2

[[1,2],[3,4],[5,6],[7,8]] --> 4
```

Задача 6 «Объединение отрезков»
Дан массив отрезков intervals, в котором некоторые отрезки могут пересекаться.
Напишите функцию, которая объединяет все пересекающиеся отрезки в один и возвращает новый массив непересекающихся отрезков.

```
import collections
class Solution:
  def overlap(a, b):
    return a[0] <= b[1] and b[0] <= a[1]

  def buildGraph(intervals):
    graph = collections.defaultdict(list)

    for i, interval_i in enumerate(intervals):
      for j in range(i+1, len(intervals)):
        if Solution.overlap(interval_i, intervals[j]):
          graph[tuple(interval_i)].append(intervals[j])
          graph[tuple(intervals[j])].append(interval_i)

    return graph

  def mergeNodes(nodes):
    min_start = min(node[0] for node in nodes)
    max_end = max(node[1] for node in nodes)
    return [min_start, max_end]

  def getComponents(graph, intervals):
    visited = set()
    comp_number = 0
    nodes_in_comp = collections.defaultdict(list)

    def markComponentDFS(start):
      stack = [start]
      while stack:
        node = tuple(stack.pop())
        if node not in visited:
          visited.add(node)
          nodes_in_comp[comp_number].append(node)
          stack.extend(graph[node])


    for interval in intervals:
      if tuple(interval) not in visited:
        markComponentDFS(interval)
        comp_number += 1
```

```
        return nodes_in_comp, comp_number


    def merge(intervals):
        graph = Solution.buildGraph(intervals)
        nodes_in_comp, number_of_comps = Solution.getComponents(graph, intervals)

        return [Solution.mergeNodes(nodes_in_comp[comp]) for comp in range(number_of_comps)]
print("[[1,3],[2,6],[8,10],[15,18]] -->", Solution.merge([[1,3],[2,6],[8,10],[15,18]]))
print("[[2,5],[4,8]] -->", Solution.merge([[2,5],[4,8]]))
```

```
[[1,3],[2,6],[8,10],[15,18]] --> [[1, 6], [8, 10], [15, 18]]
[[2,5],[4,8]] --> [[2, 8]]
```

Задача 7 Победа строки Даны две строки: s1 и s2 с одинаковым размером, проверьте, может ли некоторая перестановка строки s1 “победить” некоторую перестановку строки s2 или наоборот. Строка x может “победить” строку y (обе имеют размер n), если x[i]> = y [i] (в алфавитном порядке) для всех i от 0 до n-1.

```
class Won:
    def wonStr(s,t):
        if (len(s) != len(t)): return False
        alist1=list(s)
        alist1.sort()
        alist2=list(t)
        alist2.sort()
        for i in range (0,len(s)):
            if alist2[i]>=alist1[i]:
                matches=True
            else: matches=False

        return matches

print(Call.wonStr("abc","xya"))
print(Call.wonStr("abe","acd"))
```

```
          True
False
```

Задача 8 Полиндромная подстрока Дана строка s, вернуть самую длинную полиндромную подстроку в s

```
class Solution:
    def longestPalindrome(s):
        longest = ""
        for i, _ in enumerate(s):
            candidate = Solution.get_palindrome(s, start = i, end = i)
            if len(candidate) > len(longest):
```

```
            longest = candidate
        return longest
    @staticmethod
    def get_palindrome(s, start, end):
        while end + 1 < len(s) and s[end+1] == s[start]:
            end += 1
        while start > 0 and end + 1 < len(s) and s[start - 1] == s[end + 1]:
            start -= 1
            end += 1
        return s[start:end + 1]
print(Solution.longestPalindrome("ccbd"))
```
```
cc
```

Задача 9 Конкатеция строки Вернуть количество отдельных непустых подстрок текста, которые могут быть записаны как конкатенация некоторой строки с самой собой (т.е. она может быть записана, как a + a, где a - некоторая строка).

```
class Solution:
    def dist(s):
        result = set()
        for l in range(1, len(s)//2+1):
            count = sum(s[i] == s[i+l] for i in range(l))
            for i in range(len(s)-2*l):
                if count == l:
                    result.add(s[i:i+l])
                count += (s[i+l] == s[i+l+l]) - (s[i] == s[i+l])
            if count == l:
                result.add(s[len(s)-2*l:len(s)-2*l+l])
        return len(result)

print("abcabcabc -->", Solution.dist("abcabcabc"))
print("ctaacatacat -->", Solution.dist("ctaacatacat"))
```
```
abcabcabc --> 3
ctaacatacat --> 2
```