Курсовая работа по дисциплине СиАОД

Самарина А.В. группа БСТ1904

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:
Вы выбираете 3 стопки монет из оставшихся на столе.
Алиса забирает себе стопку с максимальным количеством монет.
Вы забираете одну из двух оставшихся стопок.
Боб забирает последнюю стопку.
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 у координаты не имеют значения в данной задаче. Координата xstart всегда меньше xend. Стрелу можно выстрелить строго вертикально (вдоль уоси) из разных точек x оси. Шарик с координатами xstart и xend уничтожается стрелой, если она была выпущена из такой позиции x, что xstart ⩽ x ⩽ 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

```
def ballons(arr):
    if not arr: return 0
    arr = sorted(arr)
    result = 1
    prev = arr[0]

    for balloon in arr:
        if prev[1] < balloon[0]:
            prev = balloon
            result += 1
        else:
            prev[1] = min(balloon[1], prev[1])
    return(result)

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

print("\n[[1,2],[3,4],[5,6],[7,8]] -->",ballons([[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.

```
# если встречается конец отрезка,
# который по значению меньше, чем начало следующего отрезка,
# то появляется новый, обособелнный от других отрезок и добавляем его в результат
# Иначе берем конец следующего отрезка и меняем текущий конец
def merge(intervals):
    result = []
    intervals.sort()
    for interval in intervals:
        if result == [] or result[-1][1] < interval[0]:
            result.append(interval)
        else:
            result[-1][1] = max(result[-1][1], interval[1])
    return result

print(merge([[1,3],[2,6],[7,8],[11,18]]))
```

```
[[1, 6], [7, 8], [11, 18]]
```

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):
        alist1=list(s)
        alist1.sort()
        alist2=list(t)
        alist2.sort()
        for i in range (0,len(s)):
            if alist2[i]>=alist1[i]:
                m=True
            else: m=False

        return m

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

    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("babad"))
bab
```

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
```