Университет ИТМО

Инфокоммуникационные технологии и системы связи

Программирование в инфокоммуникационных сетях

Лабораторная работа #1

Коркина Анна Михайловна (K3120)

Преподаватель: Харьковская Татьяна Александровна

Г. Санкт-Петербург

10.04.2021

1 задача. Максимальная стоимость добычи

Вор находит гораздо больше добычи, чем может поместиться в его сумке. Помогите ему найти самую ценную комбинацию предметов, предполагая, что любая часть предмета добычи может быть помещена в его сумку.
Решение: сортируем массив цен и весов по цене за килограмм по убыванию и забираем всё пока не закончится место в рюкзаке.
Код:

```
in_file = open('input.txt')
n, max_weight = in_file.readline().split()
n, max_weight = int(n), int(max_weight)
things = []
for i in range(int(n)):
    line = in_file.readline().split()
    price = int(line[0])
    weight = int(line[1])
    things.append((price, weight, price / weight))
in_file.close()

things.sort(key=lambda x: x[2], reverse=True)
taked = 0

ans = ""
for i in things:
    new_weight = max_weight - i[1]
    if (new_weight >= 0):
        taked += i[0]
        max_weight -= i[1]
    else:
        taked += i[0] * (max_weight / i[1])
        max_weight = 0

    if (max_weight == 0):
        ans = f"{taked:.4f}"

out = open('output.txt', 'w')
out.write(ans)
print(ans)
out.close()
```

2 задача. Заправки

Вы собираетесь поехать в другой город, расположенный в d км от вашего родного города. Ваш автомобиль может проехать не более m км на полном баке, и
вы начинаете с полным баком. По пути есть заправочные станции на расстояниях
stop1, stop2, ..., stop(n) из вашего родного города. Какое минимальное количество
заправок необходимо?

Решение:

В данном решении осуществляется проход по заправкам и смотрится возможно ли доехать до следующей заправки без пополнения бака на текущей.

Код:

```
inp = open('input.txt')
need = int(inp.readline())
tank = int(inp.readline())
n = int(inp.readline())
stations = list(map(int, inp.readline().split()))
inp.close()

def getFillsCount(need, tank, stations):
    stations.append(need)
    curr = 0
    count = 0
    for i in range(len(stations) - 1):
        if (stations[i] - curr > tank):
            return -1
        if (stations[i + 1] - curr) > tank:
            curr = stations[i]
            count += 1
    if (stations[len(stations) - 1] - curr < tank):
        return count
    else:
        return -1


count = getFillsCount(need, tank, stations)
out = open('output.txt', 'w')
out.write(str(count))
print(str(count))
out.close()
```

3 задача. Максимальный доход от рекламы

У вас есть n объявлений для размещения на популярной интернет-странице. Для каждого объявления вы знаете, сколько рекламодатель готов платить за один клик по этому объявлению. Вы настроили n слотов на своей странице и оценили ожидаемое количество кликов в день для каждого слота. Теперь ваша цель - распределить рекламу по слотам, чтобы максимизировать общий доход.

Код:

```
inp = open("input.txt")
n = int(inp.readline())
a = list(map(int, inp.readline().split()))
b = list(map(int, inp.readline().split()))
inp.close()

a = list(sorted(a))
b = list(sorted(b))

summ = 0
for i in range(n):
    summ += a[i] * b[i]

out = open("output.txt", "w")
out.write(str(summ))
out.close()
```

Решение:
изначально сортируются два входных массива и произведения соответствующих элементов из двух массивов суммируются. Это позволяет нам получить максимальный доход даже при наличии отрицательных чисел в массивах.

5 задача.Максимальное количество призов

Вы организуете веселый конкурс для детей. В качестве призового фонда у вас есть n конфет. Вы хотели бы использовать эти конфеты для раздачи k лучшим местам в конкурсе с естественным ограничением, заключающимся в том, что чем выше место, тем больше конфет. Чтобы осчастливить как можно больше детей, вам нужно найти наибольшее значение k, для которого это возможно.

Решение:

```
fin = open('input.txt')
n = int(fin.readline())
sum = 0
list_num = []
for i in range(1, n+1):
    if sum + i <= n:
        sum += i
        list_num.append(i)
    else:
        list_num[-1] += (n-sum)
        break


fout = open('output.txt', 'r')
fout.write(len(list_num))
fout.write(*list_num)
```

в массив для хранения потенциально возможного разложения числа помещаются все целые числа, начиная с единицы до того момента, пока их сумма не станет больше, чем разность параметра и последнего элемента, добавленного в массив. После этого в массив добавляется последнее число, необходимое для того, чтобы сделать сумму равной параметру(завершить разложение)

6 задача.Максимальная зарплата

В качестве последнего вопроса успешного собеседования ваш начальник дает вам несколько листков бумаги с цифрами и просит составить из этих цифр наибольшее число.

Решение:

если числа одного порядка — то проблем нет, выбираем наибольшее

иначе станное условие: если наименьшее равно концу наибольшего и больше чем его начало то оно идет на первое место (например, 233 и 33 : 33 = 2 33

и 33 > 23 3 → 33 на первое место)

12 задача. Последовательность

Дана последовательность натуральных чисел a1, a2, ..., an, и известно, что ai ≤ i для любого 1 ≤ i ≤ n. Требуется определить, можно ли разбить элементы последовательности на две части таким образом, что сумма элементов в каждой из частей будет равна половине суммы всех элементов последовательности.

Решение:

```
inp = open("input.txt")
n = int(inp.readline())
seq = list(map(int, inp.readline().split()))
need = sum(seq)
if(need % 2 == 1):
    out = open("output.txt", "w")
    out.write("-1")
    out.close()
    exit()

need = need // 2
taked = 0
i = 0
while((i < len(seq)) and (taked < need)):
    taked += seq[i]
    i += 1

if(taked == need):
    res = ""
    takedSecondHalf = 0
    takedSecondCount = 0
    for index in range(i, len(seq)):
        takedSecondHalf += seq[index]
        takedSecondCount += 1
        res += f"{seq[index]} "
    if (takedSecondHalf == taked == need):
        out = open("output.txt", "w")
        out.write(f"{takedSecondCount} \n")
        out.write(res)
        out.close()
        exit()

out = open("output.txt", "w")
out.write("-1")
out.close()
```

Так как разбить последовательность надо на 2 части (то есть нельзя вырывать элементы из разных частей, а надо получить две непрерывные последовательности), то сначала просто в цикле while набираем необходимую нам сумму (сумму/2), если она набралась, то суммируем оставшиеся элементы и проверяем равны ли у нас первая и вторая часть.

13. Сувениры

Вы и двое ваших друзей только что вернулись домой после посещения разных стран. Теперь вы хотели бы поровну разделить все сувениры, которые все трое накупили.

```
fin = open('input.txt', 'r')
n = fin.readline()
x = list(map(int, fin.readline().split()))
fin.close()
fout = open('output.txt', 'w')
flag = 1

if sum(x) % 3 == 0:
  tmp = sum(x) / 3
  x.sort()
  x.reverse()
  for i in range(3):
    y = []
    j = 0
    while (sum(y) != tmp and j<len(x)):
      if ((sum(y) + x[j]) <= tmp):
        y.append(x[j])
        del(x[j])
      else:
        j += 1
    if sum(y) == tmp:
      pass
      #print(y)
    else:
      flag  = 0
      break
else:
  flag = 0


if flag:
  fout.write('1')
else:
```

```
  fout.write('0')
```

Решение: сначала находим сумму каждой группы,

дальше нужно равномерно распределить наибольшие числа по группам, поэтому сортируем массив по убыванию и жадным алгоритмом добавляем максимальные числа в группы

20 задача. Почти палиндром (3 балла)

Задача:
Слово называется палиндромом, если его первая буква
совпадает с последней, вторая – с предпоследней и т.д. Например: «abba», «madam», «x». Для заданного числа K слово называется почти палиндромом, если в нем
можно изменить не более K любых букв так, чтобы получился палиндром).
Решение:

Изначально перебираются все под слова которые можно составить из данного слова, и для каждого вызывается функция которая проверяет сколько “ошибок палиндрома” оно содержит, и если это количество не превосходит K, то мы засчитываем этот вариант. И в конце проверяется является ли само слово почти палиндромом.

Код:

```
inp = open('input.txt')
N, K = map(int, inp.readline().split())
word = inp.readline()
inp.close()

def check_palidrom(word, K):
    if len(word) == 1:
        return True
    first_half = word[:len(word) // 2]
    if len(word) % 2 == 0:
        second_half = word[len(word) // 2:]
    else:
        second_half = word[len(word) // 2 + 1:]
    errorsCount = 0
    for i in range(len(first_half)):
        if first_half[i] != second_half[len(second_half) - i - 1]:
            errorsCount += 1
        if errorsCount > K:
            return False
    return True

count = 0
for size in range(1, len(word)):
```

```
    for i in range(len(word) - size + 1):
        sub_word = word[i:i + size]
        if check_palidrom(sub_word, K):
            count += 1

if (check_palidrom(word, K)):
    count += 1

out = open('output.txt', 'w')
out.write(str(count))
out.close()
```

21 задача. Игра в дурака (3 балла)

Петя очень любит программировать. Недавно он решил реализовать популярную карточную игру «Дурак». Но у Пети пока маловато опыта, ему срочно нужна Ваша помощь. Как известно, в «Дурака» играют колодой из 36 карт. В Петиной программе каждая карта представляется в виде строки из двух символов, где первый символ означает ранг (‘6’, ‘7’, ‘8’, ‘9’, ‘T’, ‘J’, ‘Q’, ‘K’, ‘A’) карты, а второй символ означает масть (‘S’, ‘C’, ‘D’, ‘H’). Ранги перечислены в порядке возрастания старшинства.

Решение:

```
def getCardWeight(card):
    arr = ["6", "7", "8", "9", "T", "J", "Q", "K", "A"]
    return arr.index(card)


inp = open("input.txt")

myCount, enemyCount, trump = inp.readline().split()
myCount = int(myCount)
enemyCount = int(enemyCount)
myCards = inp.readline().split()
enemyCards = inp.readline().split()

myColorCards = {"S": [], "C": [], "D": [], "H": []}
enemyColorCards = {"S": [], "C": [], "D": [], "H": []}

for card in myCards:
    myColorCards[card[1]].append(card[0])
for card in enemyCards:
    enemyColorCards[card[1]].append(card[0])
```

```
cardColors = ["S", "C", "D", "H"]
for color in cardColors:
    myColorCards[color] = sorted(myColorCards[color], key=lambda x:
getCardWeight(x))
    enemyColorCards[color] = sorted(enemyColorCards[color], key=lambda x:
getCardWeight(x), reverse=True)
cardColors.remove(trump)


def CanWin():
    for enemyCard in enemyColorCards[trump]:
        isBreaked = False
        for myCard in myColorCards[trump]:
            if (getCardWeight(myCard) > getCardWeight(enemyCard)):
                myColorCards[trump].remove(myCard)
                isBreaked = True
                break
        if (isBreaked == False):
            return False

    for color in cardColors:
        for enemyCard in enemyColorCards[color]:
            isBreaked = False
            for myCard in myColorCards[color]:
                if (getCardWeight(myCard) > getCardWeight(enemyCard)):
                    myColorCards[color].remove(myCard)
                    isBreaked = True
                    break
            if (isBreaked == False):
                for myTrump in myColorCards[trump]:
                    isBreaked = True
                    myColorCards[trump].remove(myTrump)
                    break
                if (isBreaked == False):
                    return False
    return True


result = CanWin()

out = open("output.txt", "w")
if (result):
    out.write("YES")
else:
    out.write("NO")
out.close()
```

Данная задача решается при помощи жадных алгоритмов, изначально мы сортируем наши карты по возрастанию и карты противника по убыванию. Дальше мы проходимся по козырям противника и пытаемся их отбить (при поиске карты для биты мы берём минимальную карту, которая превосходит карту противника). Далее мы проходимся по всем не козырным картам противника, и бьём их картами той же масти, если же мы не можем побить мы обращаемся к нашим козырям, если же их нет, то мы не можем побить и возвращается False. Если же все карты побиты, то возвращаем True.

Вывод:

Для большинства задач наиболее оптимальными решениями будут — жадный алгоритм иои динамическое программирование. Так же в некоторых ситуациях можно обойтись и без прямого использования динамики или жадных алгоритмов, но скорость написания и размер такого кода будет уступать предыдущим вариантам.