САНКТ-ПЕТЕРБУРГСКИЙ НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

ИНФОРМАЦИОННЫХ ТЕХНОЛОГИЙ, МЕХАНИКИ И ОПТИКИ

ФАКУЛЬТЕТ ИНФОКОММУНИКАЦИОННЫХ ТЕХНОЛОГИЙ

Отчет по лабораторной работе №1

по курсу «Алгоритмы и структуры данных»

Тема: Жадные алгоритмы. Динамическое программирование №2

Вариант 22

Выполнил:

Федюкин М. В.

К3244

Проверила:

Артамонова В.Е.

Санкт-Петербург

2023 г.

**Содержание отчета**

Содержание отчета 1

Задачи по варианту 2

Обязательные задачи

Задача №2 3

Задача №5 4

Задача №8 5

Задача №17 6

Задача №18 7

Дополнительные задачи

Задача №1 9

Задача №3 10

Задача №6 11

Задача №7 12

Задача №9 13

Задача №10 15

Задача №12 16

Задача №13 17

Задача №15 18

Задача №20 19

Задача №21 20

Вывод 22

**Задачи по варианту**

Задание к лабораторной работе № 2-1: https://drive.google.com/file/d/1nZaUIx4E\_ydeveK4B0Xs\_Jxxmd5C9FG\_/view?usp=sharing

Мой вариант – 22.  
  
Обязательные задачи: 2, (4), 8, 17, (19).  
Вместо 4-й и 19-й я решил 5-ю и 18-ю.  
  
Дополнительные задачи: 1, 3, 5, 6, 7, 9, 10, 12, 13, 15, 18, 20, 21.

**Обязательные задачи.**

**Задача 2**

Если до последней досягаемой заправочной станции не хватает милитража, тогда заправляемся на последней возможной. Если ни одной возможной станции на пути не найдется, а путь не преодолен – выводим -1.

d = int(input()) *# расстояние*m = int(input()) *# может проехать*n = int(input()) *# количество заправок*stop = list(map(int, input().split())) *# возможные остановки*  
  
stop = [m] + stop + [d]  
passed = 0 *# пройденный путь*s = 0 *# количество заправок*flag = *True  
for* i, station *in* enumerate(stop):  
 *if* station - stop[i - 1] > m:  
 print(-1)  
 flag = *False  
 break  
 if* station - passed > m:  
 passed = stop[i - 1]  
 s += 1  
*if* flag:  
 print(s)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 950  400  4  200 375 550 750 | 2 |
| 10  3  4  1 2 5 9 | -1 |
| 200  250  2  100 150 | 0 |
| 30  10  2  10 20 | 2 |

**Задача 5**

Создаю арифметическую прогрессию, сумма которой не превосходит к-во конфет. Если последний элемент выводит за границу, тогда вместо него предпоследний элемент увеличивается до достижения максимальной суммы прогрессии.

n = int(input())  
  
  
i = 0 *# количество детей*s = 0 *# счетчик конфет на человека*seq = [] *# последовательность распределенных конфет  
while* s < n:  
 i += 1  
 s += i  
 seq.append(i)  
*if* s == n:  
 print(i)  
 print(\*seq)  
*else*:  
 print(i - 1)  
 print(\*seq[:-2], i - 1 + n - sum(seq[:-1]))

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6 | 3  1 2 3 |
| 8 | 3  1 2 5 |
| 2 | 1  2 |
| 10 | 4  1 2 3 4 |

**Задача 8**

Без потенциальных комбинаций всегда возможно провести хотя бы одну лекцию.  
Сортирую временные промежутки по тому, как рано они заканчиваются. Последовательно ищу интервал, начало которго позже и ровно времени конца предыдущего. Как и в предыдущей задаче, смысл уместить как можно больше небольших интервало в доступный диапозон.

N = int(input()) *# количество заявок*slots = [] *# интервалы занятий  
for* \_ *in* range(N):  
 a, b = map(int, input().split())  
 slots.append((a, b))  
  
slots = sorted(slots, key=*lambda* v: (v[1], v[0]))  
cur\_slot = slots[0][1] *# конец предыдущего временного слота*sch = 1 *# количество предметов (расписание)  
for* slot *in* slots[1:]:  
 *if* slot[0] >= cur\_slot:  
 sch += 1  
 cur\_slot = slot[1]  
print(sch)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  5 10 | 1 |
| 4  1 8  1 7  1 3  1 2 | 1 |
| 4  1 8  2 4  3 5  6 7 | 2 |
| 6  1 2  2 3  3 4  4 5  5 7  6 7 | 5 |
| 5  1 7  7 9  2 3  3 4  4 5 | 4 |

**Задача 17**

Создаю словарь с возможными переходы от кажой цифры.

paths = {0: [4, 6],  
 1: [6, 8],  
 2: [7, 9],  
 3: [4, 8],  
 4: [0, 3, 9],  
 5: [],  
 6: [0, 1, 7],  
 7: [2, 6],  
 8: [1, 3],  
 9: [2, 4]}  
  
n = int(input())  
  
M = [[1 *for* \_ *in* range(10)] *for* \_ *in* range(n)] *for* i *in* range(1, n):  
 *for* j *in* range(10):  
 M[i][j] = sum([M[i - 1][k] *for* k *in* paths[j]])  
print((sum(M[n-1]) - M[n-1][0] - M[n-1][8]) % 10\*\*9)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1 | 8 |
| 2 | 16 |

**Задача 18**

n = int(input()) *# количество дней*price = [[]] *# стоимости обедов в течение n дней  
for* \_ *in* range(n):  
 price.append(int(input()))  
  
M = [[float("inf") *for* \_ *in* range(n+3)] *for* \_ *in* range(n+1)]  
M[0][1] = 0  
*for* i *in* range(1, n+1):  
 *for* j *in* range(1, n+2):  
 *if* price[i] <= 100:  
 M[i][j] = min(M[i-1][j] + price[i], M[i-1][j+1])  
 *else*:  
 M[i][j] = min(M[i-1][j-1] + price[i], M[i-1][j+1])  
minimum = float("inf") *# минмальная сумма за все походы на обед*left = 0  
*for* i, elem *in* enumerate(M[n]):  
 *if* elem <= minimum:  
 minimum = elem  
 left = i  
print(minimum)  
  
dinner\_days = []  
i, j = n, left  
*while* i != 0:  
 *if* price[i] > 100:  
 *if* M[i-1][j-1] + price[i] <= M[i-1][j+1]:  
 j -= 1  
 *else*:  
 j += 1  
 dinner\_days.append(i)  
 *else*:  
 *if* M[i-1][j] + price[i] > M[i-1][j+1]:  
 j += 1  
 dinner\_days.append(i)  
 i -= 1  
print(left - 1, len(dinner\_days))  
*for* days *in* reversed(dinner\_days):  
 print(days)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5  110  40  120  110  60 | 260  0 2  3  5 |
| 3  110  110  110 | 220  1 1  2 |
| 8  120  300  20  10  29  30  70  32 | 311  0 1  2 |

**Дополнительные задачи**

**Задача 1**

Сортирую предметы по удельной стоимости и максимально заполняю сумку. Если для последнего предмета по приоритету возможно добавить лишь его часть, заполняю оставшееся место в сумке удельной стоимостью последнего предмета по приоритету.

n, W = map(int, input().split()) *# n - количество предметов, W - вместимость сумки*items = [] *# предметы  
for* \_ *in* range(n):  
 items.append(tuple(map(int, input().split())))  
  
items.sort(key=*lambda* val: val[1]/val[0])  
v, i = 0.0, 0 *# v - итоговая стоимость добычи, i - счетчик  
while* W > 0:  
 *if* items[i][1] <= W:  
 W -= items[i][1]  
 v += items[i][0]  
 i += 1  
 *else*:  
 v += W \* (items[i][0]/items[i][1])  
 W = 0  
print(round(v, 4))

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 50  60 20  100 50  120 30 | 180.0 |
| 1 10  500 30 | 166.667 |
| 4 8  4 9  10 6  13 7  20 9 | 17.778 |
| 3 10  1 100  2 75  3 50 | 0.6 |

**Задача 3**

Чтобы произведение пары чисел было максимальным, их знаки должны совпадать, а модули – быть максимальными. Если же знаки не совпадают, то разница между модулями чисел должна быть минимальна. Отсортируем оба входных массива по возрастанию и попарно перемножим элементы.

n = int(input()) *# количество слотов*A = sorted(list(map(int, input().split()))) *# прибыль за клик i слота*B = sorted(list(map(int, input().split()))) *# количество кликов на i слот*  
  
s = 0 *# общий доход  
for* i *in* range(n):  
 s += A[i] \* B[i]  
print(s)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 1  23  39 | 897 |
| 3  1 3 -5  -2 4 1 | 23 |
| 4  -2 -9 7 6  1 1 1 1 | 2 |
| 1  99  0 | 0 |

**Задача 6**

Отсортируем числа по убыванию с условием: увеличим каждый элемент до длины максимального из них путем повторения последнего символа. Так числа будут сортроваться не по наибольшему из них, а по величине последовательных цифр в их записи. Полученый список выведем в виде одной слитой строки.

n = int(input()) *# количество чисел*A = list(input().split()) *# список чисел*  
  
L = len(max(A, key=len))print("".join(reversed(sorted(A, key=*lambda* i: i + i[-1]\*(L - len(i))))))

|  |  |
| --- | --- |
| **Input** | **Output** |
| 2  21 2 | 221 |
| 3  23 39 92 | 923923 |
| 5  12893 21 372 12 891 | 891372211289312 |
| 4  12 13 14 15 | 15141312 |

**Задача 7**

Задача состоит в том, чтобы из всех доступных вариантов выбрать как можно больше так, чтобы уложиться в заданную сумму. Поэтому просто всегда будем выбирать из доступных вариантов наименьший.

K, n = map(int, input().split()) *# K - длина раб.дня в минутах, n - количество сапог*T = sorted(list(map(int, input().split()))) *# список минут на сапог*  
  
i = 0  
*while* K - T[i] >= 0:  
 K -= T[i]  
 i += 1  
print(i)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 10 3  6 2 8 | 2 |
| 3 2  10 20 | 0 |
| 10 4  3 4 1 8 | 3 |
| 15 3  10 20 30 | 1 |

**Задача 9**

Проверяю насколько выгоднее оплатить определенный разряд последовательно уменьшающимися категориями.

N = int(input()) *# количество листов на печать*A = [] *# цены за категории*count = [] *# счетчик  
for* \_ *in* range(7):  
 A.append(int(input()))  
 count.append(0)  
  
*for* i *in* range(6, -1, -1):  
 count[i] = N // 10\*\*i  
 N %= 10\*\*i  
  
*for* i *in* range(7):  
 *if* sum([A[x] \* count[x] *for* x *in* range(i)]) >= A[i]:  
 count[i] += 1  
 *for* j *in* range(i):  
 count[j] = 0  
  
*for* i *in* range(6, -1, -1):  
 *for* j *in* range(6, i, -1):  
 *if* A[i] \* 10\*\*(j - i) < A[j]:  
 count[i] += count[j] \* int(10\*\*(j - i))  
 count[j] = 0  
  
s = sum([A[i] \* count[i] *for* i *in* range(7)]) *# итоговая сумма*print(s)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 980  1  9  90  900  1000  10000  10000 | 882 |
| 980  1  10  100  1000  900  10000  10000 | 900 |
| 980  1  10  101  1000  900  10000  10000 | 900 |
| 9801  1  10  101  1000  90001  10000  10000 | 9801 |

**Задача 10**

Сортируем яблоки по их итоговой способности увеличить или уменьшить + не измениться. Алиса должна есть увеличивающие яблоки с последовательно увеличивающемся качеством уменьшения, а затем так же последовательно есть уменьшающие яблоки. Если в один момент первичное уменьшение пересилит актуальный рост девочки, тогда невозможно съесть все яблоки – выводим -1.

num, s = map(int, input().split()) *# num - количество яблок, s - рост*Papples = [] *# увеличивают*Napples = [] *# уменьшают  
for* i *in* range(num):  
 n, p = map(int, input().split())  
 Papples.append([n, p-n, i + 1]) *if* p-n > 0 *else* Napples.append([n, p-n, i + 1])  
  
Papples.sort(key=*lambda* v: v[0])  
order = []  
flag = *True  
for* apple *in* Papples + Napples:  
 *if* s - apple[0] > 0:  
 s += apple[1]  
 order.append(apple[2])  
 *else*:  
 flag = *False* print(-1)  
 *break  
if* flag:  
 print(\*order)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3 5  2 3  10 5  5 10 | 1 3 2 |
| 3 5  2 3  10 5  5 6 | -1 |
| 3 5  4 10  3 2  3 1 | 1 2 3 |
| 2 5  5 10  3 4 | 2 1 |

**Задача 12**

Дана уже отсортированная по возрастанию последовательность.  
Нахожу ее сумму. Отнимем от нее последовательно уменьшающиеся числа списка, пока не достигаю эквилибриума. Так сделаем наименьшее количество шагов. Если сумма отнятых чисел перевесила половину суммы последовательности – вывожу -1.

n = int(input()) *# количество чисел*A = list(map(int, input().split())) *# список чисел*  
  
s1 = sum(A) *# первая часть*s2 = 0 *# вторая часть*i = n - 1  
*while* s2 < s1:  
 s1 -= A[i]  
 s2 += A[i]  
 i -= 1  
*if* s1 != s2:  
 print(-1)  
*else*:  
 print(n - 1 - i)  
 print(\*sorted(d))

|  |  |
| --- | --- |
| **Input** | **Output** |
| 3  1 2 3 | 1  3 |
| 5  1 2 3 3 3 | 2  4 5 |
| 6  1 2 3 2 2 6 | 2  5 6 |
| 3  1 1 1 | -1 |

**Задача 13**

Если общая сумма сувениров делится на три, то раскидываем их по трем массивам так, чтобы в массивах получились одинаковые суммы.

n = int(input()) *# количество сувениров*S = list(map(int, input().split())) *# стоимости сувенорив*  
  
*if* sum(S) % 3 != 0:  
 print(0)  
*else*:  
 share = sum(S) // 3  
 S.sort(reverse=*True*)  
 bundles = [] *# итоговые пачки сувениров троих  
 for* i *in* range(3):  
 bundle = [] *# итоговая пачка сувениров одного* j = 0 *#  
 while* sum(bundle) != share:  
 *if* sum(bundle) + S[j] <= share:  
 bundle.append(S[j])  
 *del* S[j]  
 *else*:  
 j += 1  
 *if* j + 1 > len(S):  
 *break* bundles.append(bundle)  
 print(1 *if* sum(bundles[0]) == sum(bundles[1]) == sum(bundles[2]) *and* len(S) == 0 *else* 0)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 4  3 3 3 3 | 0 |
| 1  40 | 0 |
| 11  17 59 34 57 17 23 67 1 18 2 59 | 1 |
| 13  1 2 3 4 5 5 7 7 8 10 12 19 25 | 1 |

**Задача 15**

*def* changes(seq, start, end):  
 *if* len(seq[start:end]) == 0:  
 *return* 0, []  
 *if* seq[end - 1] == '(' *or* seq[end - 1] == '[' *or* seq[end - 1] == '{':  
 variety = [(1 + changes(seq, start, end - 1)[0], changes(seq, start, end - 1)[1] + [end - 1])]  
 *else*:  
 variety = [(1 + changes(seq, start, end - 1)[0], changes(seq, start, end - 1)[1] + [end - 1])]  
 *for* i, bracket *in* enumerate(seq[start:end]):  
 *if* bracket == '(' *and* seq[end - 1] == ')' *or* bracket == '[' *and* seq[end - 1] == ']' \  
 *or* bracket == '{' *and* seq[end - 1] == '}':  
 one = changes(seq, start, start + i)  
 two = changes(seq, start + i + 1, end - 1)  
 variety.append((one[0] + two[0], one[1] + two[1]))  
 improved\_seq = min(variety, key=*lambda* para: para[0])  
 *return* improved\_seq  
  
given\_seq = input()  
  
to\_del = changes(given\_seq, 0, len(given\_seq))[1]  
new\_seq = ""  
*for* i *in* range(len(given\_seq)):  
 *if* i *not in* to\_del:  
 new\_seq += given\_seq[i]  
print(new\_seq)

|  |  |
| --- | --- |
| **Input** | **Output** |
| ([)] | () |
| ((([)))] | ((())) |
| {}[}](([)) | {}[](()) |
| (){}[]{()()} | (){}[]{()()} |

**Задача 20**

Чтобы любая буква являлась четной, добавлю символ «|» как разделитль между всеми буквами.  
Идем по строке и для каждого символа «расходимсяь» в обе стороны, пока не достигнем разницы, превышающей k, или не достигнем конца строки. Если при расхождении две крайних буквы не равны - увеличиваем счетчик разницы на 1.

n, k = map(int, input().split())  
s = input()  
  
wrd = "|"  
*for* i *in* s: *# слово с обозначеннымим промежутками между буквами* wrd += i + "|"  
count = 0 *# счетчик*i = 0  
nums = 0 *# расхождение  
for* center *in* range(len(wrd)):  
 *while* center - i >= 0 *and* center + i < len(wrd) *and* nums <= k:  
 *if* wrd[center - i] != wrd[center + i]:  
 nums += 1  
 *if* wrd[center - i] != "|" *and* nums <= k:  
 count += 1  
 i += 1  
 *else*:  
 i = 0  
 nums = 0  
print(count)

|  |  |
| --- | --- |
| **Input** | **Output** |
| 5 1  abdcd | 12 |
| 3 3  aaa | 6 |

**Задача 21**

Создаем два словаря-иерархии для мастей и рангов. Создаем двумерный массив, куда сохранем «свои» карты. При получении вражеской карты оцениваем, можно ли побить ее картой той же масти минимального старшего ранга. Если нельзя, ищем, есть ли у нас козырные карты, чтобы отбиться. Если нет – возвращаем отрицательный ответ.

Ranks = {'S': 0,  
 'C': 1,  
 'D': 2,  
 'H': 3}  
  
Suits = {'6': 0,  
 '7': 1,  
 '8': 2,  
 '9': 3,  
 'T': 4,  
 'J': 5,  
 'Q': 6,  
 'K': 7,  
 'A': 8}  
  
  
*def* durak(trump\_card, me, opponent):  
 m = [[0 *for* \_ *in* range(9)] *for* \_ *in* range(4)]  
 *for* item *in* me:  
 m[Ranks[item[1]]][Suits[item[0]]] = 1  
 *for* item *in* opponent:  
 row, col = Ranks[item[1]], Suits[item[0]]  
 flag = *True  
 for* j *in* range(col, 9):  
 *if* m[row][j] != 0:  
 m[row][j] = 0  
 flag = *False  
 break  
 if* flag *and* row != Ranks[trump\_card]:  
 flag = *True* row, col = Ranks[trump\_card], 0  
 *for* j *in* range(col, 9):  
 *if* m[row][j] != 0:  
 m[row][j] = 0  
 flag = *False  
 break  
 if* flag:  
 *return* "NO"  
 *return* "YES"  
  
  
n, m, trump = input().split()  
my\_deck = list(input().split())  
opp\_deck = list(input().split())  
  
print(durak(trump, my\_deck, opp\_deck))

|  |  |
| --- | --- |
| **Input** | **Output** |
| 6 2 С  KD KC AD 7C AH 9C  6D 6C | YES |
| 4 1 D  9S KC AH 7D  8D | NO |

**Вывод**

Мне понравилось решать данные задачи, во многом это была работа логики. И я люблю задачи-головоломки.

Наиболее трудно далась 15-я задача, однако спасибо прошлому мне, почти вся лабораторная мною была начата больше года назад, так что у меня были почти законченные задачи и некоторые наработки.

В предстоящих лабораторных требуется больше работы с пониманием фундаментальных базовых систем, что меня лично несколько пугает.