Заключительный (очный) этап научно-образовательного соревнования

Олимпиады школьников «Шаг в будущее» по профилю «Инженерное дело» специализации «Профессор Лебедев» (общеобразовательный предмет информатика), весна 2020 г.

11 класс

Вариант 1

Задача 1

Учитель информатики Игорь Петрович составил программу для проверки знаний пятиклассников по работе с натуральными дробями и смешанными числами. Она выдаёт примеры на сложение и вычитание двух дробей. Вам требуется написать другую программу, которая для заданных примеров будет определять правильный ответ.

Входные данные: арифметическое выражение с двумя смешанными числами вида A B/C + D E/F, где A - целая часть первого числа, B - числитель дробной части первого числа, C - знаменатель дробной части первого числа, D - целая часть второго числа, E - числитель дробной части второго числа, F - знаменатель дробной части второго числа. Между числами стоит знак операции “+” или “-” без кавычек. Целая, дробная части и знак операции разделены пробелами. Дробные части пробелов не содержат. Числа A, B, C, D, E, F не превышают 1000.

У числа могут присутствовать целая и дробная части вместе или только одна из них.

Выходные данные: результат вычисления выражения в виде смешанного числа с дробной частью в виде сокращённой натуральной дроби, разделёнными одним пробелом, или целого числа.

Пример

Исходные данные	Результат
2/5 + 1 5/6	2 7/30
3 + 10/5	5

Проверочные тесты

1 2/5 + 1 5/6	3 7/30
1 2/5 + 5/6	2 7/30
5 - 4	1
12 30/5 - 17 17/17	0
4 - 5	-1

Пример решения

```
def nod(a, b):
    while a != 0 and b != 0:
    if a > b:
    a -= b
    else:
    b -= a
    return a + b

def frac_to_ir(frac):
```

```
        if len(frac) == 1 and frac[0].isdigit():
        ir = [int(frac[0]), 1]
        elif len(frac) == 1 and not frac[0].isdigit():
        ir = [int(i) for i in frac[0].split('/')]
        else:
        frac_part = [int(i) for i in frac[1].split('/')]
        ir = [int(frac[0]) * frac_part[1] + frac_part[0], frac_part[1]]
        return ir

def parse_expr(expr):
        i = 0
        while expr[i] not in ['+', '-']:
        i += 1
        return [frac_to_ir(expr[:i]), frac_to_ir(expr[i + 1:]), expr[i]]

def calculate(parsed):
        en1 = parsed[0][0] * parsed[1][1]
        en2 = parsed[1][0] * parsed[0][1]
        den = parsed[0][1] * parsed[1][1]
        if parsed[2] == '+':
        return [en1 + en2, den]
        else:
        return [en1 - en2, den]

def ir_to_frac(ir):
        num = abs(ir[0]) // ir[1]
        en = abs(ir[0]) % ir[1]
        den = ir[1]
        n = nod(en, den)
        return [num, [en // n, den // n], ir[0] > 0]

expr = input().split()
parsed = parse_expr(expr)
ir = calculate(parsed)
frac = ir_to_frac(ir)
if frac[0] == 0 and frac[1][0] == 0:
        print(0)
elif frac[0] == 0:
        print('' if frac[2] else '-', frac[1][0], '/', frac[1][1], sep='')
elif frac[1][0] == 0:
        print('' if frac[2] else '-', frac[0], sep='')
else:
        print('' if frac[2] else '-', frac[0], ' ', frac[1][0], '/', frac[1][1], sep='')
```

Задача 2

Составить программу, которая выделит все элементы, образующие последовательность Фибоначчи и расположенные под побочной диагональю или лежащие на ней, в заданном порядке.

Числа Фибоначчи - элементы числовой последовательности, первые два числа которой равны 0 и 1, а каждое последующее - сумма двух предыдущих: 0, 1, 1, 2, 3, 5...

Побочной диагональю квадратной матрицы называется диагональ, идущая из правого верхнего угла в левый нижний.

Элементы требуется просматривать по спирали по часовой стрелке, начиная с правого верхнего угла.

Входные данные: в первой строке задано единственное натуральное число N <= 100 - порядок матрицы. Далее идёт N строк по N целых чисел, разделённых одним или несколькими пробелами и описывающими элементы матрицы построчно. Каждый из элементов матрицы по модулю не превышает 106.

Выходные данные: найденные элементы последовательности Фибоначчи, удовлетворяющие условию, записанные в строку через пробел.

Пример

Исходные данные	Результат
4 0 1 2 3 4 5 **3 0** 7 8 **5 1** **2 1** 2 3	0 1 1 2 3 5

Примечание: жирным шрифтом выделены элементы, включаемые в результирующую последовательность.

Проверочные тесты

4 0 1 2 3 4 5 3 0 7 8 5 1 2 1 2 3	0 1 1 2 3 5
1 0	0
2 1 0 1 1	0 1 1
5 9 9 9 9 9 9 9 9 9 9 9 9 5 0 9 9 2 1 1 9 9 9 9 9 9	0 1 1
5 9 9 9 9 9 9 9 9 2 9 9 9 1 0 9 9 2 1 3 9 1 9 9 9 0	0 1 1 2 3

Пример решения

```
n = int(input())
lst = []
for i in range(n):
```

```
        lst0 = list(map(int, input().split()))
        lst.append(lst0[n-i-1:])
ans = []
#print(lst)
for k in range((n+1)//2):
        for i in range(len(lst)):
        if lst[i] == []:
        continue
        ans.append(lst[i][-1])
        lst[i] = lst[i][:len(lst[i])-1]
        last = lst[-1]
        for i in last[::-1]:
        ans.append(i)
        lst = lst[:len(lst)-1]
        for i in range(len(lst)-1, -1, -1):
        if lst[i] == []:
        continue
        ans.append(lst[i][0])
        lst[i] = lst[i][1:]
res = []
a = ''
for i in ans:
        if i == 0 and res == []:
        res.append(i)
        a += '0'
        elif i == 1 and res == [0]:
        res.append(i)
        a += ' 1'
        elif len(res) >= 2 and i == res[-1] + res[-2]:
        res.append(i)
        a += ' ' + str(i)
print(a)
```

Задача 3

Даны два прямоугольника на плоскости. Для каждого из них известны координаты трёх из четырёх вершин. Требуется определить, пересекаются ли эти прямоугольники.

Считать, что прямоугольники пересекаются, если они имеют общие точки, расположенные внутри периметра фигур.

Входные данные: 2 строки по 6 вещественных чисел, записанных через пробел, в диапазоне от -1000 до 1000 и имеющих до 3-х знаков после точки. Каждая строка содержит координаты вершин x1, y1, x2, y2, x3, y3 соответствующего прямоугольника.

Выходные данные: YES, если прямоугольники пересекаются, и NO, если не пересекаются.

Пример

Исходные данные	Результат
0 0 0 10 10 10 -1 0 0 1 1 0	YES

0 0.1 2 0.1 0 1.1 10 11 15 11 10 15	NO

Проверочные тесты

1 1 1 10 10 10 -2 0 0 2 2 0	YES
0 0.1 2 0.1 0 1.1 10 11 15 11 10 15	NO
2 2 3 5 9 3 0 0 -2 3 3 4	YES
1 0 1 2 10 2 0.5 0 0.5 2 9.5 2	YES
0 0 -2 2 -4 0 0 0 -2 0 -2 2	YES

Пример решения

```
x11,y11,x12,y12,x13,y13 = map(float,input().split())
x21,y21,x22,y22,x23,y23 = map(float,input().split())

minx1 = min(x11,x12,x13)
maxx1 = max(x11,x12,x13)

miny1 = min(y11,y12,y13)
maxy1 = max(y11,y12,y13)

minx2 = min(x21,x22,x23)
maxx2 = max(x21,x22,x23)

miny2 = min(y21,y22,y23)
maxy2 = max(y21,y22,y23)

if minx1 > maxx2 or minx2 > maxx1 or miny1 > maxy2 or miny2 > maxy1:
        print("NO")
else:
        print("YES")
```

Задача 4

IP (Internet Protocol) - сетевой протокол, поддерживающий передачу пакетов между устройствами, расположенными в разных сетях (подсетях). В 4-й версии протокола (IPv4) адрес устройства состоит из четырёх октетов (8-разрядных неотрицательных чисел).

Подсетью называется (упрощённо) группа узлов, подключённых к одному сетевому коммутатору. Такие узлы могут направлять друг другу пакеты напрямую. В остальных случаях необходима маршрутизация пакета - передача его на маршрутизатор, который по специальной таблице определяет, в какую подсеть следует перенаправить пакет, чтобы он попал к адресату.

Подсети задаются с помощью битовых масок, которые совпадают с адресом по размеру и в двоичном представлении состоят из непрерывной последовательности единиц, дополненной

нулями до длины адреса. Два узла расположены в одной подсети, если их адреса, логически умноженные на значение маски, совпадают.

Задана таблица маршрутизации сетевого устройства, работающего по протоколу IPv4. Требуется определить, по какому из маршрутов отправится пакет, направленный узлу с заданным адресом.

Входные данные: в первой строке - IP-адрес - строка из 4-х чисел от 0 до 255, разделённых точками, и количество маршрутов N (N <= 100). Далее N строк описывают таблицу маршрутизации. В каждой строке таблицы через пробел записаны: адрес сети, маска, приоритет. Маска записана в виде 4-х чисел от 0 до 255, разделённых точками, причём числа всегда заданы такие, что соответствующая им всем битовая последовательность не может содержать нулей, идущих левее, чем единицы.

IP-адрес, для которого требуется определить маршрут, не может содержать нулей в первом и последнем октетах. Адрес сети не может начинаться с нулевого октета. Маска должна содержать не менее одного единичного разряда.

Приоритет указывает на то, какой маршрут должен быть применён, если подходят несколько. В случае, если применимы несколько маршрутов с одинаковым приоритетом, выбирается первый из них.

Выходные данные: номер маршрута либо -1, если маршрута нет.

Пример

Исходные данные	Результат
192.168.10.10 3 192.168.10.0 255.255.255.0 10 10.0.0.0 255.192.0.0 5 192.168.8.0 255.255.252.0 1	1
192.168.9.10 3 192.168.10.0 255.255.255.0 10 10.0.0.0 255.192.0.0 5 192.168.8.0 255.255.252.0 1	3

Проверочные тесты

192.168.10.11 3 192.168.10.0 255.255.255.0 10 10.0.0.0 255.192.0.0 5 192.168.8.0 255.255.252.0 1	1
192.168.9.10 3 192.168.10.0 255.255.255.0 10 10.0.0.0 255.192.0.0 5 192.168.8.0 255.255.248.0 1	3
192.168.9.10 1 192.168.10.0 255.255.255.0 10	-1
192.168.10.128 1 192.168.10.0 255.255.255.128 10	-1
192.168.10.11 3	3

192.168.10.0 255.255.255.0 10 10.0.0.0 255.192.0.0 5 192.168.8.0 255.255.252.0 11	

Пример решения

```
addr, n = input().split()
addr = list(map(int, addr.split(".")))
ans=-1
pr=-1
for i in range(int(n)):
        s = input().split()
        s[0] = list(map(int, s[0].split('.')))
        s[1] = list(map(int, s[1].split('.')))
        s[2] = int(s[2])
        network1 = []
        for j in range(4):
        network1.append(s[0][j]&s[1][j])
        network2 = []
        for j in range(4):
        network2.append(addr[j]&s[1][j])
        if network1==network2 and s[2]>pr:
        pr = s[2]
        ans=i+1
print(ans)
```

Задача 5

Двоичным деревом называется иерархическая структура данных, в которой каждый узел имеет не более двух потомков.

Двоичное дерево поиска - разновидность двоичного дерева, у которого оба поддерева (левое и правое) являются двоичными деревьями поиска; все узлы левого поддерева произвольного узла A меньше, чем значение самого узла A; все узлы правого поддерева произвольного узла A больше либо равны значению узла A.

При прямом обходе дерева сначала обрабатывается ключ (значение) корня, затем - ключи левого и правого поддеревьев. При обратном обходе сначала обрабатываются ключи левого поддерева, затем правого, затем - корня.

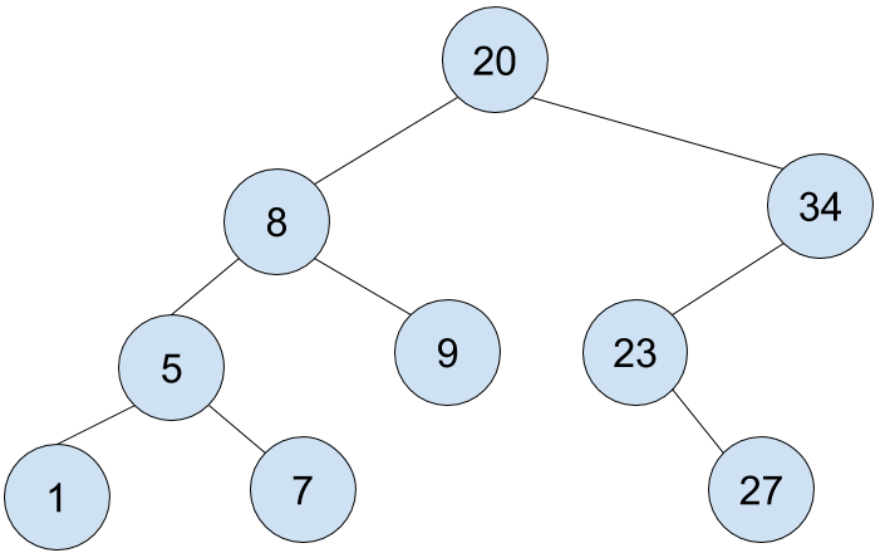
По заданным ключам вершин, полученным при прямом обходе, требуется вывести ключи того же дерева, получаемые при обратном обходе.

Входные данные: ключи всех вершин двоичного дерева поиска в порядке прямого обхода. Каждый ключ - натуральное число, не превышающее 105. Каждое значение задано в отдельной строке. Последняя строка состоит из одного символа - точки.

Выходные данные: ключи дерева, соответствующие обратному порядку обхода, по одному в строке.

Пример

Исходные данные и результат соответствуют дереву, изображённому на рисунке:



Исходные данные	Результат
20 8 5 1 7 9 34 23 27 .	1 7 5 9 8 27 23 34 20

Проверочные тесты

21 8 5 1 7 9 34 23 27 .	1 7 5 9 8 27 23 34 21
1	1

.	
2 1 3 .	1 3 2
4 3 2 1 5 6 7 .	1 2 3 7 6 5 4
4 3 1 2 6 5 7 .	1 3 2 5 7 6 4

Пример решения

```
#include <bits/stdc++.h>
using namespace std;

struct node{
	int v;
	node *l, *r;
};

void add(int k, node *&tree){
	if (tree == NULL){
	tree = new node;
	tree->v = k;
	tree->l = NULL;
	tree->r = NULL;
	}
	else if (tree->v <= k){
	if (tree->r != NULL){
	add(k, tree->r);
	}
	else{
	tree->r = new node;
	tree->r->v = k;
	tree->r->l = NULL;
	tree->r->r = NULL;
	}
	}
	else{
```

```
        if (tree->l != NULL){
        add(k, tree->l);
        }
        else{
        tree->l = new node;
        tree->l->v = k;
        tree->l->l = NULL;
        tree->l->r = NULL;
        }
        }
}

void ans(node *&tree){
        if (tree->l != NULL)
        ans(tree->l);
        if (tree->r != NULL)
        ans(tree->r);
        cout << tree->v << '\n';
        return;
}

signed main(){
        node *tree = NULL;
        int n;
        while(cin >> n)
        add(n, tree);
        ans(tree);
}
```

Задача 6

При планировании приёмной кампании в вузы города Энска в 3030 году городской департамент образования принял решение о переходе на дистанционную подачу документов через городскую компьютерную сеть. Ожидается, что абитуриентам больше не придётся обходить образовательные учреждения и проводить время в очередях: все смогут подать заявления на интересующие их специальности, не выходя из дома, а подведение итогов конкурса произойдёт автоматически, когда завершится подача заявлений.

При этом каждый вуз подводит итоги конкурса независимо друг от друга.

Для подведения итогов конкурса в вузе составляются **конкурсные списки** абитуриентов по каждой специальности, ранжированные по убыванию суммы их баллов по трём предметам. В случае равенства суммы баллов производится дополнительное ранжирование по убыванию баллов за 1-й предмет, затем за 2-й, затем за 3-й. Если выявляется несколько абитуриентов с абсолютно одинаковыми баллами, проходящих на специальность, где свободных мест осталось меньше их количества - принимаются все такие абитуриенты.

Кроме того, в каждой записи указывается, какой приоритет для абитуриента имеет эта специальность.

Согласно порядку приёма, абитуриент может подать заявления не более, чем на 3 специальности в один вуз.

Требуется написать программу, которая по заданным конкурсным спискам определит списки зачисленных на каждую специальность.

Входные данные: в первой строке - натуральное число N <= 50 - количество специальностей в вузе. Далее - N описаний конкурсных списков по каждой специальности.

Описание конкурсного списка начинается с заголовка - строки, где указан шифр или название специальности - строка из цифр, латинских букв, знаков подчёркивания, точек и запятых, длина которой не превышает 50 символов, и через пробел - количество мест для приёма. Количество мест - натуральное число, не превышающее 200. Далее идут строки, в каждой из которых указана информация об очередном абитуриенте. Строка с описанием абитуриента состоит из его имени - слова, записанного латинскими буквами, длина которого не превышает 20; затем через пробел 3 оценки - натуральных числа, не превышающих 100; затем натуральное число от 1 до 3 - приоритет данной специальности согласно предпочтению абитуриента.

Описание каждого конкурсного списка заканчивается пустой строкой.

Гарантируется, что у каждого абитуриента приоритеты не повторяются и номер приоритета не превышает количество выбранных им специальностей. Баллы каждого абитуриента одинаковы на всех специальностях.

Выходные данные: списки поступивших (зачисленных) на каждую специальность. Порядок специальностей должен соответствовать исходному.

В первой строке каждого списка указывается её название и через пробел - количество зачисленных; далее в каждой строке - имя абитуриента. Имена абитуриентов следует отсортировать по возрастанию в алфавитном порядке.

Пример

Исходные данные	Результат
2 01.03.02 2 Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1 09.03.01 3 Vasya 99 99 98 1 Ivan 100 90 80 2	01.03.02 2 Anya Petya 09.03.01 2 Ivan Vasya

Проверочные тесты

2 01.03.02 2 Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1 09.03.01 3 Vasya 99 99 98 1 Ivan 100 90 80 2	01.03.02 2 Anya Petya 09.03.01 2 Ivan Vasya
1 01.03.02 2	01.03.02 2 Anya

Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1	Petya
1 01.03.02 2 Petya 100 100 100 1 Anya 100 90 90 1 Ivan 100 90 80 1	01.03.02 2 Anya Petya
1 01.03.02 2 Petya 100 100 100 1 Anya 100 90 90 1 Ivan 100 90 90 1	01.03.02 3 Anya Ivan Petya
5 01.03.02 2 Petya 100 100 100 2 Anya 100 90 90 1 Ivan 100 90 90 1 Vasya 95 90 95 1 02.03.02 2 Petya 100 100 100 1 Anya 100 90 90 2 Ivan 100 90 90 2 Vasya 95 90 95 2 10.03.01 2 Petya 100 100 100 3 Anya 100 90 90 2 Ivan 100 90 90 1 Vasya 95 90 95 3	01.03.02 2 Anya Ivan 02.03.02 2 Petya Vasya 10.03.01 0

Пример решения

```
#include <iostream>
#include <string>
#include <map>
#include <set>
#include <algorithm>
#include <cmath>
#include <vector>
#include <fstream>

typedef long long ll;
typedef long double ld;

#define len(x) ((int)x.size())
```

```
#define f first
#define s second
#define all(x) x.begin(), x.end()

using namespace std;

struct abitur {
    int a, b, c, sum;
    int targ, prior;
    string name;
};

bool comp1(abitur &x, abitur &y) {
    if (x.sum != y.sum)
        return x.sum > y.sum;
    if (x.a != y.a)
        return x.a > y.a;
    if (x.b != y.b)
        return x.b > y.b;
    if (x.c != y.c)
        return x.c > y.c;
    if (x.name != y.name)
        return x.name < y.name;
    return x.prior < y.prior;
}

bool comp2(abitur &x, abitur &y) {
    return x.name < y.name;
}

bool operator ==(abitur &x, abitur &y) {
    return (
        (x.sum == y.sum) &&
        (x.a == y.a) &&
        (x.b == y.b) &&
        (x.c == y.c)
    );
}

int main()
{
    int n;
    cin >> n;

    vector <vector <abitur>> ans(n);

    vector <abitur> lst;
    vector <int> razm(n);
    vector <string> tp(n);
    string tmp;
    getline(cin, tmp);
```

```
for (int i = 0; i < n; i++) {
	cin >> tp[i] >> razm[i];
	getline(cin, tmp);
	bool flag = 1;
	while (flag) {
		getline(cin, tmp);
		if (tmp.empty()) {
			flag = 0;
		} else {
			int p = 0;
			string name = "";
			string a = "";
			string b = "";
			string c = "";
			string pr = "";
			while (tmp.back() != ' ') {
				pr += tmp.back();
				tmp.pop_back();
			}
			tmp.pop_back();
			while (tmp.back() != ' ') {
				c += tmp.back();
				tmp.pop_back();
			}
			tmp.pop_back();
			while (tmp.back() != ' ') {
				b += tmp.back();
				tmp.pop_back();
			}
			tmp.pop_back();
			while (tmp.back() != ' ') {
				a += tmp.back();
				tmp.pop_back();
			}
			tmp.pop_back();
			while (!tmp.empty()) {
				name += tmp.back();
				tmp.pop_back();
			}
			reverse(all(name));
			reverse(all(a));
			reverse(all(b));
			reverse(all(c));
			reverse(all(pr));
			abitur man;
			man.name = name;
			man.a = stoi(a);
			man.b = stoi(b);
			man.c = stoi(c);
			man.prior = stoi(pr);
			man.targ = i;
			man.sum = man.a + man.b + man.c;
```

```
                        lst.push_back(man);
                }
        }
    }

    sort(all(lst), comp1);
    set <string> was;

    for (int i = 0; i < len(lst); i++) {
        int want = lst[i].targ;
        string man = lst[i].name;
        if (was.find(man) != was.end())
                continue;
        if (len(ans[want]) < razm[want] || ans[want].back() == lst[i]) {
                was.insert(man);
                ans[want].push_back(lst[i]);
        }
    }

    for (int i = 0; i < n; i++) {
        cout << tp[i] << ' ' << len(ans[i]) << '\n';
        sort(all(ans[i]), comp2);
        for (auto &elem : ans[i])
                cout << elem.name << '\n';
    }

    //system("pause");
    return 0;
}
```

Задача 7

Требуется решить задачу №6 с тем отличием, что наборы экзаменов на разные программы различаются, а абитуриенты имеют более трёх оценок.

Для подведения итогов конкурса в вузе составляются **конкурсные списки** абитуриентов по каждой специальности, ранжированные по убыванию суммы их баллов по трём предметам. В случае равенства суммы баллов производится дополнительное ранжирование по убыванию баллов за 1-й предмет, затем за 2-й, затем за 3-й. Если выявляется несколько абитуриентов с абсолютно одинаковыми баллами, проходящих на специальность, где свободных мест осталось меньше их количества - принимаются все такие абитуриенты.

Кроме того, в каждой записи указывается, какой приоритет для абитуриента имеет эта специальность.

Согласно порядку приёма, абитуриент может подать заявления не более, чем на 3 специальности в один вуз.

Требуется написать программу, которая по заданным конкурсным спискам определит списки зачисленных на каждую специальность.

Входные данные: в первой строке - натуральное число N <= 50 - количество специальностей в вузе. Далее - N описаний конкурсных списков по каждой специальности.

Описание конкурсного списка начинается с заголовка - строки, где указан шифр или название специальности - строка из цифр, латинских букв, знаков подчёркивания, точек и запятых, длина

которой не превышает 50 символов, количество мест для приёма и шифры трёх предметов, закодированные заглавными латинскими буквами. Элементы строки разделены пробелами. Количество мест - натуральное число, не превышающее 200. Далее идут строки, в каждой из которых указана информация об очередном абитуриенте. Строка с описанием абитуриента состоит из его имени - слова, записанного латинскими буквами, длина которого не превышает 20; затем через пробел 3 оценки - натуральных числа, не превышающих 100; затем натуральное число от 1 до 3 - приоритет данной специальности согласно предпочтению абитуриента.

Описание каждого конкурсного списка заканчивается пустой строкой.

Гарантируется, что у каждого абитуриента приоритеты не повторяются и номер приоритета не превышает количество выбранных им специальностей. Баллы каждого абитуриента по одному и тому же предмету одинаковы на всех специальностях.

Выходные данные: списки поступивших (зачисленных) на каждую специальность. Порядок специальностей должен соответствовать исходному.

В первой строке каждого списка указывается её название и через пробел - количество зачисленных; далее в каждой строке - имя абитуриента. Имена абитуриентов следует отсортировать по возрастанию в алфавитном порядке.

Пример

Исходные данные	Результат
3 01.03.02 2 F M R Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1 09.03.01 3 I M R Anya 100 100 100 2 Vasya 99 99 98 1 Petya 95 90 90 2 Dima 95 90 90 1 Nikita 90 90 81 1 Ivan 90 90 80 2 09.03.02 2 I M R Vasya 99 99 98 2 Ivan 90 90 80 3 Nikita 90 90 81 2	01.03.02 2 Anya Petya 09.03.01 3 Dima Nikita Vasya 09.03.02 1 Ivan

Проверочные тесты

3 01.03.02 2 F M R Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1	01.03.02 2 Anya Petya 09.03.01 3 Dima

09.03.01 3 I M R Anya 100 100 100 2 Vasya 99 99 98 1 Petya 95 90 90 2 Dima 95 90 90 1 Nikita 90 90 81 1 Ivan 90 90 80 2 09.03.02 2 I M R Vasya 99 99 98 2 Ivan 90 90 80 3 Nikita 90 90 81 2	Nikita Vasya 09.03.02 1 Ivan
1 01.03.02 2 F M R Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 80 1	01.03.02 2 Anya Petya
1 01.03.02 2 F M R Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 91 89 1	01.03.02 2 Anya Ivan
1 01.03.02 2 F M R Anya 100 100 100 1 Petya 100 90 90 1 Ivan 100 90 90 1	01.03.02 3 Anya Ivan Petya
3 01.03.02 2 F M R Anya 100 100 100 3 Petya 100 90 90 2 Ivan 100 90 80 1 09.03.01 3 I M R Anya 100 100 100 2 Vasya 99 99 98 1 Petya 95 90 90 3 Dima 95 90 90 1 Nikita 90 90 81 1 Ivan 90 90 80 2 09.03.02 2 I M R Vasya 99 99 98 3 Kolya 90 90 80 1	01.03.02 1 Ivan 09.03.01 3 Dima Nikita Vasya 09.03.02 1 Kolya 09.03.04 2 Anya Petya

Nikita 90 90 81 3 09.03.04 3 I M R Anya 100 100 100 1 Vasya 99 99 98 2 Petya 95 90 90 1 Ivan 90 90 80 3 Nikita 90 90 81 2	

Пример решения

```
#include <iostream>
#include <string>
#include <map>
#include <set>
#include <algorithm>
#include <cmath>
#include <vector>
#include <fstream>

typedef long long ll;
typedef long double ld;

#define len(x) ((int)x.size())
#define f first
#define s second
#define all(x) x.begin(), x.end()

using namespace std;

struct abitur {
    map <char, int> mp;
    int targ, prior;
    string name;
};

vector <char> base;

bool comp1(abitur &x, abitur &y) {
    int sum1 = 0;
    int sum2 = 0;
    for (int i = 0; i < 3; i++) {
            sum1 += x.mp[base[i]];
            sum2 += y.mp[base[i]];
    }
    if (sum1 != sum2)
            return sum1 > sum2;
    if (x.mp[base[0]] != y.mp[base[0]])
            return x.mp[base[0]] > y.mp[base[0]];
    if (x.mp[base[1]] != y.mp[base[1]])
            return x.mp[base[1]] > y.mp[base[1]];
```

```
    if (x.mp[base[2]] != y.mp[base[2]])
        return x.mp[base[2]] > y.mp[base[2]];
    return x.name < y.name;
}

bool comp2(abitur &x, abitur &y) {
    return x.name < y.name;
}

bool operator ==(abitur &x, abitur &y) {
    int sum1 = 0;
    int sum2 = 0;
    for (int i = 0; i < 3; i++) {
        sum1 += x.mp[base[i]];
        sum2 += y.mp[base[i]];
    }
    return (
        (sum1 == sum2) &&
        (x.mp[base[0]] == y.mp[base[0]]) &&
        (x.mp[base[1]] == y.mp[base[1]]) &&
        (x.mp[base[2]] == y.mp[base[2]])
    );
}

int main()
{
    int n;
    cin >> n;

    vector <vector <abitur>> ans(n);

    vector <abitur> lst;
    vector <int> razm(n);
    vector <string> tp(n);
    vector <vector <char>> bs(n, vector <char> (3));
    string tmp;
    getline(cin, tmp);

    for (int i = 0; i < n; i++) {
        cin >> tp[i] >> razm[i] >> bs[i][0] >> bs[i][1] >> bs[i][2];
        getline(cin, tmp);
        bool flag = 1;
        while (flag) {
            getline(cin, tmp);
            if (tmp.empty()) {
                flag = 0;
            } else {
                int p = 0;
                string name = "";
                string a = "";
                string b = "";
                string c = "";
```

```
                        string pr = "";
                        while (tmp.back() != ' ') {
                                pr += tmp.back();
                                tmp.pop_back();
                        }
                        tmp.pop_back();
                        while (tmp.back() != ' ') {
                                c += tmp.back();
                                tmp.pop_back();
                        }
                        tmp.pop_back();
                        while (tmp.back() != ' ') {
                                b += tmp.back();
                                tmp.pop_back();
                        }
                        tmp.pop_back();
                        while (tmp.back() != ' ') {
                                a += tmp.back();
                                tmp.pop_back();
                        }
                        tmp.pop_back();
                        while (!tmp.empty()) {
                                name += tmp.back();
                                tmp.pop_back();
                        }
                        reverse(all(name));
                        reverse(all(a));
                        reverse(all(b));
                        reverse(all(c));
                        reverse(all(pr));
                        abitur man;
                        man.name = name;
                        man.mp[bs[i][0]] = stoi(a);
                        man.mp[bs[i][1]] = stoi(b);
                        man.mp[bs[i][2]] = stoi(c);
                        man.prior = stoi(pr);
                        man.targ = i;
                        lst.push_back(man);
                }
        }
}

set <string> was;

for (int pritt = 1; pritt < 4; pritt++) {
        for (int j = 0; j < n; j++) {
                base = bs[j];
                sort(all(lst), comp1);
                for (int i = 0; i < len(lst); i++) {
                        int want = lst[i].targ;
                        if (want != j)
                                continue;
```

```
                        if (lst[i].prior != pritt)
                                continue;
                        string man = lst[i].name;
                        if (was.find(man) != was.end())
                                continue;
                        if (len(ans[want]) < razm[want] || ans[want].back() == lst[i]) {
                                was.insert(man);
                                ans[want].push_back(lst[i]);
                        }
                }
        }
    }

    for (int i = 0; i < n; i++) {
        cout << tp[i] << ' ' << len(ans[i]) << '\n';
        sort(all(ans[i]), comp2);
        for (auto &elem : ans[i])
                cout << elem.name << '\n';
    }

    //system("pause");
    return 0;
}
```