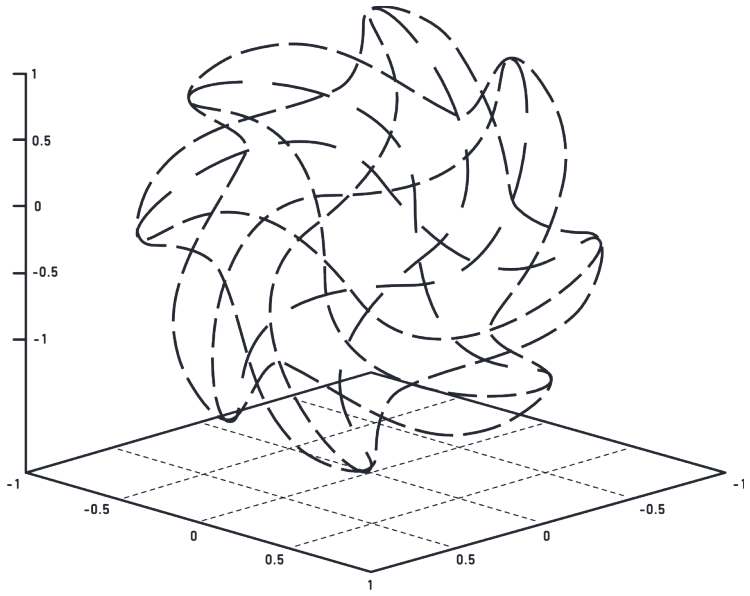


МАТЕРИАЛЫ ЗАДАНИЙ

командной инженерной олимпиады школьников

«Олимпиада Национальной технологической инициативы»

по профилю

БОЛЬШИЕ ДАННЫЕ И МАШИННОЕ ОБУЧЕНИЕ

2015/16 учебный год

http://nti-contest.ru

Оглавление

Профиль «Большие данные и машинное обучение» погружает участников в выполнение реальных задач, связанных с анализом больших объемов данных и разработкой реальных приложений для анализа данных.

Профиль включает в себя задачи по двум школьным предметам: **математика** и **информатика**.

§1 Первый отборочный этап

Первый отборочный тур проводится индивидуально в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. На решение задач первого отборочного этапа участникам давалось 3 недели. Решение каждой задачи дает определенное количество баллов. Для всех участников предлагается общий набор задач, но за решение задач участникам разных параллелей (9 класс или 10-11 класс) давались разные баллы. Решение задач по информатике подразумевало написание программ на языке Python. Баллы зачисляются в полном объеме за правильное решение задачи. Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 20 баллов.

1.1. Задачи по математике

Задача 1.1.1 (1 балл)

В поисках внеземной жизни ученые обнаружили интересный живой организм - Камкохоб. Эксперименты в земных условиях показали, что Камкохоб размножается делением. То есть родительский организм исчезает, и образуются новые особи. При этом каждая особь либо делится ровно на 5 потомков, либо не размножается, а остается одной особью . Экспериментальный образец, привезенный на Землю размножался, некоторые его потомки тоже размножались. **Выберите в таблице**, какое количество потомков могло получиться в итоге, а какое не могло.

Количество потомков	Могло получиться	Не могло получиться
9		
15		
1001		
2016		

РЕШЕНИЕ:

Правильный ответ на поставленную задачу представлен в таблице. Подробнее ход решения рассматривается в следующей задаче.

Количество потомков	Могло получиться	Не могло получиться
9	да	
15		да
1001	да	
2016		да

Задача 1.1.2 (1 балл)

В поисках внеземной жизни ученые обнаружили интересный живой организм - Камкохоб. Эксперименты в земных условиях показали, что Камкохоб размножается делением. То есть родительский организм исчезает, и образуются новые особи. При этом каждая особь либо делится ровно на 5 потомков, либо не размножается, а остается одной особью. Экспериментальный образец, привезенный на Землю, размножался, некоторые его потомки тоже размножались. Какое количество потомков могло получиться? **Опишите всю серию возможных ответов общей формулой**, используя переменную x (где x - натуральное число).

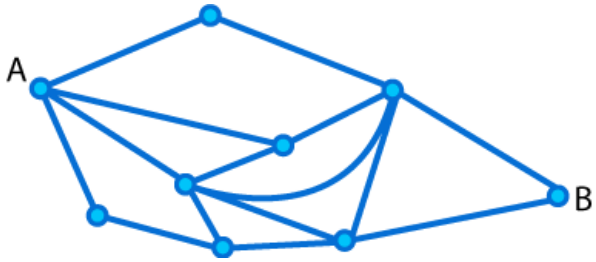
РЕШЕНИЕ:

Посмотрим, как меняется общее количество особей при одном размножении. Родительская особь исчезает, появляется 5 новых. То есть общее количество увеличивается на 4. Поскольку исходно у нас одна особь, то могло получиться только число, дающее остаток 1 от деления на 4.

Формула 4x+1, где x - натуральное число, описывает все возможные варианты. Действительно, чтобы получить 4x+1 особей, достаточно размножить любые x особей.

Задача 1.1.3 (1 балл)

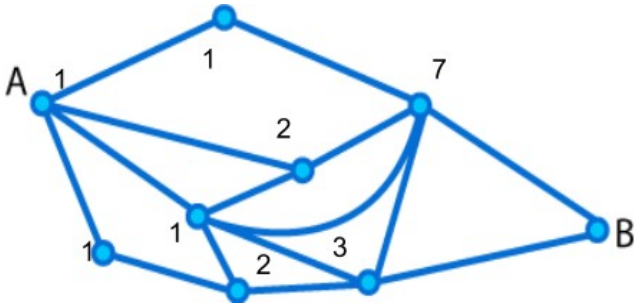
Автобус едет из пункта A в пункт B. При этом в любой момент времени он движется вправо, чтобы приближаться к пункту B иногда вправо вверх, иногда вправо вниз. Найдите количество способов доехать. На картинке приведена схема дорог между городами.



РЕШЕНИЕ:

Расставим количество способов доехать до каждой вершины, двигаясь слева

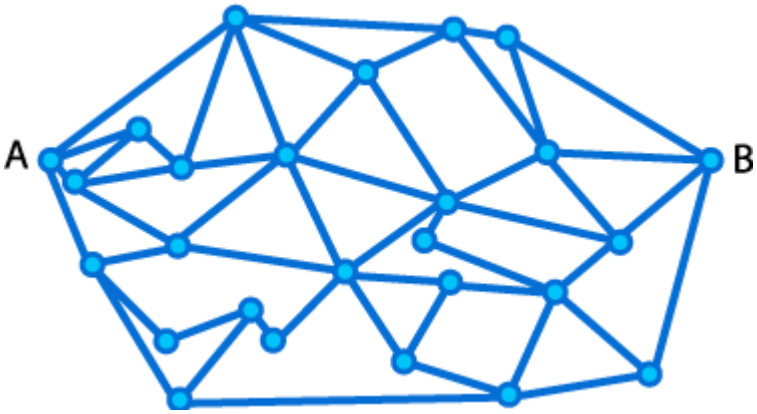
направо.



До вершины B 7+3=10 способов. Ответ: 10

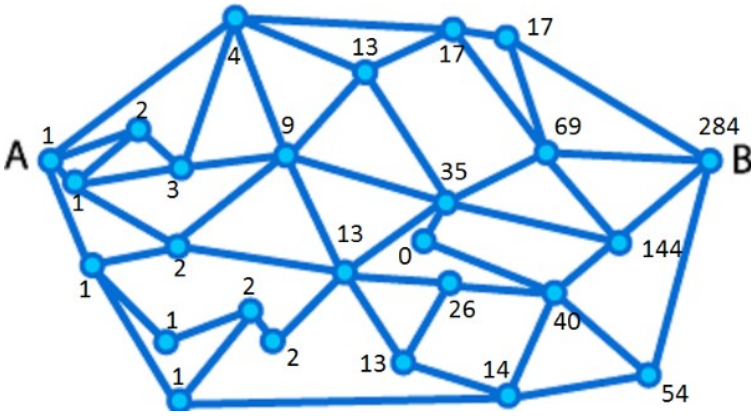
Задача 1.1.4 (1 балл)

Автобус едет из пункта A в пункт B. При этом в любой момент времени он движется вправо, чтобы приближаться к пункту B, иногда вправо вверх, иногда вправо вниз. Найдите количество способов доехать. На картинке приведена схема дорог между городами.



РЕШЕНИЕ:

Расставим количество способов доехать до каждой вершины, двигаясь слева направо.



Ответ: 284

Задача 1.1.5 (1 балл)

Незадачливый космонавт Иннокентий почувствовал себя плохо после центрифуги и не может определить направление, он находится в 5 метрах от комиссии и движется по прямой. Каждую секунду он с равной вероятностью либо приближается на метр к ней, либо отдаляется. Если он дошел до комиссии, то больше уже никуда не идет. Найдите вероятность попадания в руки комиссии не позже чем **на 6 секунде**.

Решение:

Пусть комиссия находится в пяти метрах справа от космонавта.

Любой путь космонавта за 6 секунд кодируется последовательностью из 6 символов П или Л (П, если он идет направо и Л - если налево). Всего таких последовательностей 64, из них нам годятся две: ПППППП и ПППППЛ.

Значит вероятность попасть к комиссии равна 2/64=1/32

Ответ: 1/32

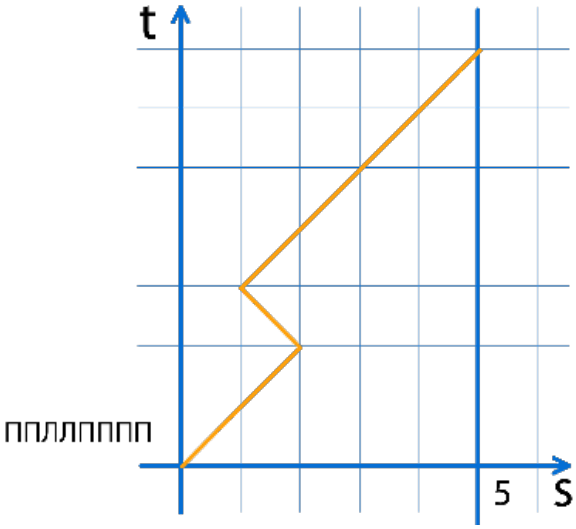
Задача 1.1.6 (1 балл)

Незадачливый космонавт Иннокентий почувствовал себя плохо после центрифуги и не может определить направление, он находится в 5 метрах от комиссии и движется по прямой. Каждую секунду он с равной вероятностью либо приближается на метр к ней, либо отдаляется. Если он дошел до комиссии, то больше уже никуда не идет. Найдите вероятность попадания в руки комиссии не позже чем **на 10-й секунде**.

РЕШЕНИЕ:

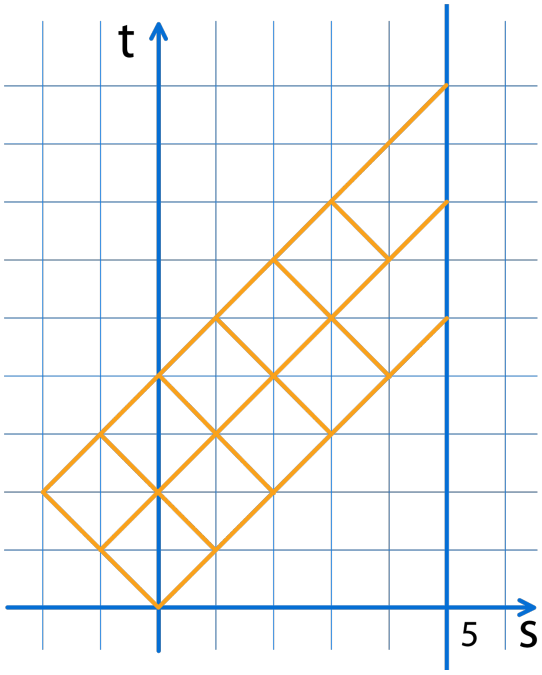
Изобразим путь космонавта на графике. По вертикальной оси отложим время, а по горизонтальной расстояние. Изначально космонавт находится в точке s=0, а комиссия в точке s=5.

На рисунке обозначен путь для последовательности ППЛПППП:

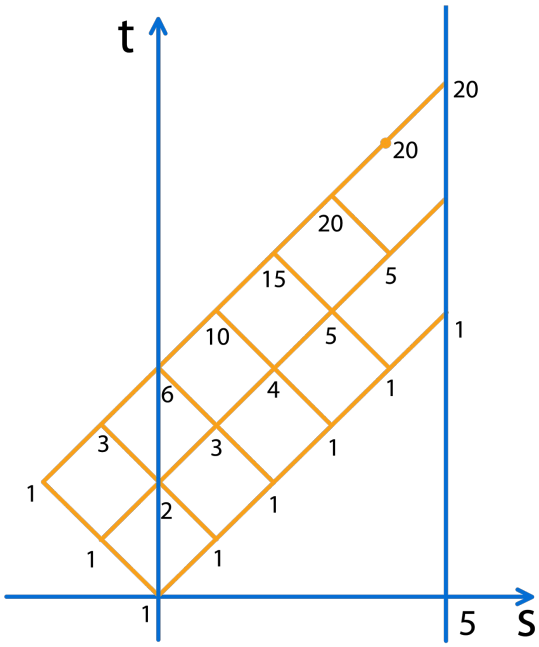


Заметим также, что космонавт может прийти к комиссии только на нечетных шагах. Таким образом, нас интересует, сколько существует путей, ведущих к комиссии за 5, 7 и 9 секунд.

Переформулируем задачу: сколько существует путей, по сетке на рисунке ниже, ведущих к прямой s=5. Ходить можно только двигаясь вверх.



Посчитаем количество таких путей для каждой точки, начиная от начальной.



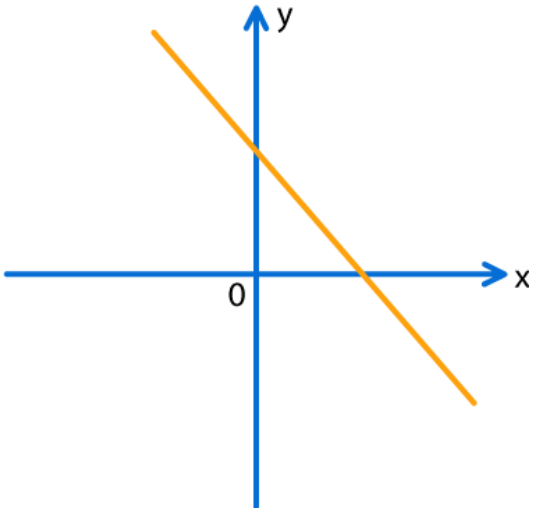
За 5 шагов приводит 1 путь, за 7 - 5 путей, за 9 - 20 путей.

Значит искомая вероятность равна 1/32+5/128+20/512 = 7/64

Ответ: 7/64

Задача 1.1.7 (1 балл)

На координатной плоскости задан график функции y=kx+b.



Найдите максимальное значение функции $y=kx^2+bx$.

РЕШЕНИЕ:

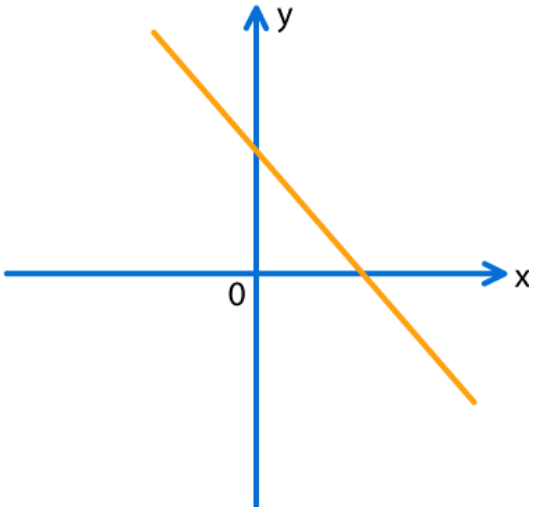
Поскольку y = kx+b убывает, k<0

$y = kx^2+bx = x(kx+b)$ парабола «рожками вниз», с нулями x = 0 и x = -b/k

Абсцисса оси параболы x= -b/2k подставим и найдем максимум $y=-b^2/4k$

Задача 1.1.8 (1 балл)

На координатной плоскости задан график функции $y=kx+b$.



Рассмотрите точки пересечения графика этой функции с графиком функции $y=kx^2+bx$. Выберите из этих точек точку с наименьшей абсциссой и выведите в ответе, чему равна эта абсцисса.

РЕШЕНИЕ:

Графиком квадратичной функции $y = x(kx + b)$ является парабола «рогами вниз», т.к. $k<0$

Один из нулей этой функции совпадает с нулем функции $y = kx + b$, а другой: $x = 0$. Между нулями расположена вторая точка пересечения графиков. Одним из корней уравнения $x(kx + b) = kx + b$ является $x = 1$, что даёт абсциссу второй точки пересечения данного и искомого графиков.

Ответ: 1

1.2. Задачи по информатике

Задача 1.2.1 «Кости» (1 балл)

В некоторых клетках квадратной доски 4×4 находятся четырёхгранные игральные кости. Кость снимается с поля, если количество соседних по стороне непустых клеток совпадает с числом, выпавшем на кости. Все кости, для которых выполняется это свойство, снимаются с поля одновременно. Если после снятия костей появляются новые кости, которые можно снять, то они снимаются по тем же правилам.

Нужно переставить одну кость так, чтобы снялось максимальное количество костей.

Формат входных данных:

На вход программе даётся четыре строки по четыре числа в каждом, разделённые одним пробелом. Всё числа — целые неотрицательные, не превышающие 4. Ноль означает отсутствие кости, любое положительное — наличие кости, на которой выпало указанное число.

Гарантируется, что на поле есть, по крайней мере, одна кость и одна свободная клетка

Пример ввода 1:

```
0 0 1 0
0 0 0 0
0 0 1 0
0 0 0 0
```

Пример ввода 2:

```
0 0 0 1
0 2 1 0
0 1 2 1
0 0 0 0
```

Формат выходных данных:

В виде ответа нужно вывести одно целое число — количество костей, которое можно снять с доски.

Пример вывода 1:

```
2
```

Пример вывода 2:

```
6
```

Пояснение. Во втором примере можно переставить единицу с первого ряда в четвёртый, под двойку. Тогда у левой верхней двойки будет два соседа, и она снимется. Так же с ней снимутся единица в четвёртом ряду и единица в четвёртом столбце, так как у них по одному соседу. После этого на доске останутся две единицы и одна двойка — их общий сосед. Все оставшиеся кости можно снять, так как правило снятия выполняется.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
      return [('0 0 1 0\n0 0 0 0\n0 0 1 0\n0 0 0 0\n', 2),
('0 0 0 1\n0 2 1 0\n0 1 2 1\n0 0 0 0\n', 6),
('0 3 3 2\n3 4 4 3\n3 4 4 2\n2 3 2 2\n', 15),
('1 1 1 2\n2 2 2 2\n2 2 2 1\n1 2 3 0\n', 15),
('1 0 0 0\n0 0 0 0\n0 0 0 0\n0 0 0 0\n', 0),
('4 4 4 4\n4 2 2 4\n4 2 1 4\n4 4 0 4\n', 0),
```

```
('0 3 0 2\n1 4 1 4\n4 0 4 0\n3 0 0 2\n', 3),
('2 4 1 1\n4 2 3 1\n4 3 1 1\n1 1 4 0\n', 4),
('2 3 1 0\n0 3 0 0\n1 1 1 4\n4 4 1 3\n', 5),
('0 3 3 3\n1 0 3 3\n2 2 0 0\n2 4 0 2\n', 6),
('2 0 1 2\n4 2 4 0\n0 3 0 4\n1 3 2 0\n', 7),
('3 1 4 1\n0 4 2 3\n4 2 0 1\n3 3 2 2\n', 8),
('2 4 0 4\n0 2 1 3\n3 2 1 0\n1 1 3 4\n', 9),
('4 3 3 3\n0 3 1 0\n2 2 4 1\n0 4 2 2\n', 10),
('4 3 3 1\n2 2 3 1\n1 1 4 3\n3 0 2 4\n', 11),
('0 3 2 1\n2 2 2 3\n1 0 2 1\n1 1 3 3\n', 12),
('1 3 1 2\n3 1 2 1\n2 2 0 3\n2 2 2 2\n', 13)]

def check(reply, clue):
    return int(reply) == int(clue)
```

РЕШЕНИЕ:

В этой задаче нужно аккуратно реализовать то, что указано в условии. Пример программы, реализующей данный алгоритм на языке Python:

```
1. import copy
2. import sys
3.
4. def neighbours(a, x, y):
5.     res = 0
6.     for dx, dy in [(1, 0), (0, 1), (-1, 0), (0, -1)]:
7.         xx = x + dx
8.         yy = y + dy
9.         res +=  int(-1 < xx < len(a) and -1 < yy < len(a[xx]) and a[xx][yy] !
= 0)
10.     return res
11.
12. def clear(a):
13.     b = []
14.     res = 0
15.     for x in range(len(a)):
16.         b.append([0] * len(a[x]))
17.         for y in range(len(a[x])):
18.             if a[x][y] == 0:
19.                 continue
20.             if a[x][y] != neighbours(a, x, y):
21.                 b[x][y] = a[x][y]
22.             else:
23.                 res += 1
24.     return (res, b)
25.
26.
27. def test(a, x1, y1, x2, y2):
28.     a = copy.deepcopy(a)
29.     a[x2][y2] = a[x1][y1]
30.     a[x1][y1] = 0
31.     res = 0
32.     while True:
33.         q, a = clear(a)
34.         if q == 0:
35.             return res
36.         res += q
37.
38. def solve(data):
39. a = list(map(lambda s: list(map(int, s.split())), filter(None, data.split('\
n'))))
40.     ans = 0
```

```
    for x in range(len(a)):
        for y in range(len(a[x])):
            if a[x][y] == 0:
                continue
            for xx in range(len(a)):
                for yy in range(len(a[x])):
                    if a[xx][yy] != 0:
                        continue
                    t = test(a, x, y, xx, yy)
                    if t > ans:
                        ans = t
    return str(ans)

solve(sys.stdin.read())
```

Задача 1.2.2 «Корни» (3 балла)

У мальчика Пети есть число *N*. Но оно ему не нужно, в отличие от числа *X*. Чтобы его получить Петя может брать целочисленный корень, умножать и складывать числа. Целочисленным корнем степени k из натурального числа n будем называть наибольшее натуральное число, для которого выполняется соотношение: $x^k \leq n$. Например, целочисленным корнем пятой степени из тысячи будет тройка, так как $3^5 = 243 \leq 1000$, а $4^5 = 1024 > 1000$. Обозначим это как $\sqrt[5]{1000} = 3$. Также будем считать, что степенью корня могут быть только натуральные числа.

Для Пети взятие целочисленного корня — тяжёлая задача, и он хочет минимизировать суммарную степень корней, которые встречаются в формуле получения *X*.

А ещё у Пети есть старший брат. Которого зовут Дима. Этот Дима решил добавить интереса задачке Пети, и дать такие ограничения:

1. Пете нельзя брать корни от чисел, отличных от *N*.
2. Можно перемножать только те числа, которые Петя получил с помощью операции взятия корня или умножения других чисел.
3. Складывать можно числа, которые получены как результат умножения, взятия корня или суммы других чисел.

Помогите Пете написать выражение, которое будет легко считаться и подходит под ограничения, наложенные Димой. Найдите минимальную сложность искомого выражения.

Формат входных данных:

В единственной строке даны два целых числа N и X ($1 \leq X, N \leq 1000$).

Пример ввода:

```
100 126
```

Формат выходных данных:

Выведите единственное натуральное число — ответ на задачу.

Пример вывода:

```
9
```

Пояснение к примеру: $126 = 100 + 10 + 4 \cdot 4 = \sqrt[1]{100} + \sqrt[2]{100} + \sqrt[3]{100} \cdot \sqrt[3]{100}$

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    return [('{} {}\n'.format(n, x), ans) for n, x, ans in [
(100, 126, 9),
(10, 10, 1),
(1000, 1000, 1),
(1, 1, 1),
(1, 1000, 1000),
(1000, 1, 10),
(1000, 999, 17),
(722, 966, 16),
(774, 717, 21),
(664, 177, 16),
(655, 657, 7),
(659, 65, 9),
(901, 559, 21),
(813, 314, 18),
(528, 131, 16),
(882, 258, 19),
(516, 583, 12),
(801, 767, 19),
(147, 222, 11),
(67, 743, 13),
(413, 335, 21),
(453, 467, 7),
(600, 104, 9),
(323, 209, 19),
(462, 822, 18),
(126, 743, 16),
(77, 917, 17),
(100, 999, 27),
(1, 999, 999),
(1000, 894, 29),
(999, 712, 28),
(123, 944, 24),
(432, 277, 24),
(945, 616, 28),
(100, 999, 27),
(1000, 894, 29),
(999, 712, 28),
(123, 944, 24),
(432, 277, 24),
(945, 616, 28)
]]

def check(reply, clue):
    return int(reply.strip()) == int(clue)
```

РЕШЕНИЕ:

Решение состоит из трёх этапов. На первом этапе нужно составить набор степеней и корней из N, которые может быть выгодно использовать. Если два корня с разной степенью равны, то нам выгодно использовать тот из них, степень которого меньше. Так как 210 > 1000, то для любого N будет не более 11 различных корней. На втором этапе методом динамического программирования получаем список чисел, которые можно получить перемножением корней с оптимальной суммарной степенью. На третьем этапе используя тот же метод получается список чисел, которые можно получить сложением чисел с предыдущего этапа с оптимальными суммами.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import sys
2.
3. INF = int(1e9)
4.
5. def getRoots(n, mx):
6.     ans = [INF, n]
7.     for x in range(n, 1, -1):
8.         while x ** len(ans) <= n:
9.             ans.append(x)
10.     ans.append(1)
11.     roots = dict()
12.     for i in range(len(ans)):
13.         if ans[i] != ans[i - 1] and ans[i] <= mx:
14.             roots[ans[i]] = i
15.     return roots
16.
17. def getProducts(roots, mx):
18.     ans = dict()
19.     ans[1] = 0
20.     for i in range(2, mx + 1):
21.         ans[i] = INF
22.         for k, v in roots.items():
23.             if i % k == 0:
24.                 d = i // k
25.                 if d in ans and ans[d] + v < ans[i]:
26.                     ans[i] = ans[d] + v
27.         if ans[i] == INF:
28.             ans.pop(i, None)
29.     ans[1] = roots[1]
30.     prods = [(k, v) for k, v in sorted(ans.items())]
31.     return prods
32.
33. def getSums(prods, mx):
34.     ans = [INF] * (mx + 1)
35.     ans[0] = 0
36.     for i in range(len(ans)):
37.         for k, v in prods:
38.             if k > i:
39.                 break
40.             if ans[i - k] + v < ans[i]:
41.                 ans[i] = ans[i - k] + v
42.     ans[0] = INF
43.     return ans
44.
45. def solve(dataset):
```

```
46.     n, x = list(map(int, dataset.strip().split()))
47.     roots = getRoots(n, x);
48.     prods = getProducts(roots, x)
49.     ans = getSums(prods, x)
50.     return str(ans[x])
51.
52. solve(sys.stdin.read())
```

Задача 1.2.3 «Прямая и точки» (3 балла)

На плоскости проведена прямая, проходящая через начало координат. Также на плоскости выбрано N точек, для каждой известно, с какой стороны от прямой она лежит. И M точек, для которых нужно ответить на аналогичный вопрос.

Формат входных данных:

В первой строке дано натуральное число N — число точек, для которых известно, с какой стороны прямой они лежат ($1 \le N \le 100000$). В следующих N строках идёт описание этих точек. Каждая строка состоит из трёх целых чисел, разделённых пробелом — координаты точки и указание, с какой стороны лежит точка. Все точки, для которых третье число равно нулю лежат по одну сторону от прямой, а единице — с другой. Других значений третьего параметра нет.

В следующей строке дано число M — число точек, для которых нужно указать сторону ($1 \le M \le 100000$). В следующих M строках пары целых чисел — координаты точек.

Координаты всех точек не превышают 1000000 по абсолютному значению. Гарантируется, что ни одна из точек не лежит на прямой.

Пример ввода:

```
4
1 0 1
0 1 1
-1 0 0
0 -1 0
2
1 1
-1 -1
```

Формат выходных данных:

Для каждой из M точек выведите 0 или 1 — сторону, с которой лежит точка, в том же смысле, как и во входных данных. Гарантируется, что ответ единственен.

Пример вывода:

```
1
0
```

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
      return [("4\n1 0 1\n0 1 1\n-1 0 0\n0 -1 0\n2\n1 1\n-1 -1\n", [1, 0])]

def check(reply, clue):
      if type(clue) is str:
            clue = ast.literal_eval(clue)
      ans = list(map(int, filter(None, reply.split())))
      if len(ans) != len(clue):
            return False
      for i in range(len(ans)):
            if ans[i] != clue[i]:
                  return False
      return True
```

РЕШЕНИЕ:

Если отразить все точки, у которых «сторона» равна 1, относительно начала координат, то все точки попадут в некоторый сектор плоскости. Остаётся найти прямую, которая не пересекает этот сектор. Её можно найти следующим образом: находим два луча, которые образуют границы сектора (например, используя псевдоскалярное произведение). Искомая прямая будет перпендикулярна биссектрисе этих лучей.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import sys
2. import math
3.
4. def length(p):
5.     return math.sqrt(p[0] * p[0] + p[1] * p[1])
6.
7. def solve(dataset):
8.     lines = dataset.strip().split('\n')
9.     points = []
10.     n = int(lines[0])
11.     lines = lines[1:]
12.     for i in range(n):
13.         x, y, s = list(map(int, lines[i].split()))
14.         s = 2 * s - 1
15.         x *= s
16.         y *= s
17.         points.append((x, y))
18.     lines = lines[n:]
19.
20.     mn = points[0]
21.     mx = points[0]
22.     for x, y in points:
23.         if x * mn[1] - y * mn[0] > 0:
24.             mn = (x, y)
25.         if x * mx[1] - y * mx[0] < 0:
26.             mx = (x, y)
27.
28.     mnLen = length(mn)
29.     mxLen = length(mx)
30.
```

```
31.     a = mn[0] / mnLen + mx[0] / mxLen
32.     b = mn[1] / mnLen + mx[1] / mxLen
33.
34.     m = int(lines[0])
35.     lines = lines[1:]
36.     res = []
37.     for i in range(m):
38.         x, y = list(map(int, lines[i].split()))
39.         res.append(str(int(x * a + y * b > 0)))
40.     return '\n'.join(res)
41.
42. solve(sys.stdin.read())
```

Задача 1.2.4 «Черный ящик» (5 баллов)

Есть чёрный ящик с двумя вещественными входами и одним бинарным выходом. Нужно по заданному набору входных и выходных данных максимально точно предсказать ответы чёрного ящика на данные из другой выборки входных данных.

Формат входных данных:

В первой строке дано число N — количество входных данных, для которых известны ответы. В следующих N строках дано по два вещественных и одному натуральному числу, разделённых пробелами — входные параметры и ответ чёрного ящика, соответственно.

В следующей строке дано число M — количество входных данных, для которых нужно узнать ответ. В следующих M строках даны по два вещественных числа, разделённых пробелом — сами входные данные.

Формат выходных данных:

В ответ нужно вернуть M чисел — предполагаемые ответы чёрного ящика.

Примечание 1: В этой задаче Вам нужно скачать файл с тестом, решить задачу на вашем компьютере и вернуть ответ за пять минут. В процессе решения вы можете пользоваться любыми материалами, языками и программами. Через пять минут после начала решения задачи тест помечается как устаревший и ответ к нему не принимается. После этого можно запросить новый тест, при этом таймер запускается заново. Количество попыток не ограничено. Гарантируется, что все тесты получены с использованием одного чёрного ящика с незначительной заменой внутренних констант.

Примечание 2: Эта задача подразумевает возможность частичного решения. В случае, если правильные ответы даны менее чем для половины входных, то за задачу начисляется 0 баллов. Если процент правильных ответов превышает 95%, то начисляется 5 баллов. Иначе начисляется $5* \frac{100}{81}(2p-1)^2$, где p — доля правильных ответов.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def sign(x):
      return int(x > 0)

def sqr(x):
       return x * x

eps = 0.1

def getPoint(a, b, rot):
            phi = random.uniform(0, 2 * math.pi)
            r = random.uniform(0, 10)
            x = r * math.cos(phi)
            y = r * math.sin(phi)
            f = x * x * a - y * y * b - 2
            xx = x * math.cos(rot) - y * math.sin(rot)
            yy = x * math.sin(rot) + y * math.cos(rot)
            if abs(f) < eps:
                  return None
            return (xx, yy, sign(f))

def generate():
      mode = random.choice([0, 1])
      mid = math.pi / 100 * 29
      phi = random.uniform(mid - 0.1, mid + 0.1)
      train = 1000
      test = 500
      rot = random.uniform(0, 2 * math.pi)
      a = 1/math.cos(phi) ** 2
      b = 1/math.sin(phi) ** 2

      data = '{}\n'.format(train)

      i = 0
      while i < train:
            p = getPoint(a, b, rot)
            if p is None:
                  continue
            data += '{0:.10f} {1:.10f} {2}\n'.format(p[0], p[1], p[2] ^ mode)
            i += 1

      ans = []
      data += '{}\n'.format(test)
      i = 0
      while i < test:
            p = getPoint(a, b, rot)
            if p is None:
                  continue
            data += '{0:.10f} {1:.10f}\n'.format(p[0], p[1])
            ans.append(p[2] ^ mode)
            i += 1

      return (data, ans)

MAXRES = 5
def check(reply, clue):
      ans = list(map(int, reply.split()))
      if len(ans) != len(clue):
            return (0, 'баллов: 0')
```

```
    correct = 0
    for i in range(len(clue)):
        correct += int(clue[i] == ans[i])
    if correct * 2 <= len(clue):
        return (0, 'баллов: 0')
    correct /= len(clue)
    res = min(1, 100 * (2 * correct - 1) ** 2 / 81)
    return (res, 'баллов: {0:.2f}'.format(MAXRES * res))
```

РЕШЕНИЕ:

Если нарисовать данный тест на плоскости, считая пары чисел как координаты, то можно заметить, что есть одна зона, в которую попадают все точки с одним ответом и две, куда попадают остальные. Причём границы этих областей имеют простую структуру. Дальше можно было либо понять, что граница представляет собой гиперболу и восстановить её параметры, либо воспользоваться специальными инструментами, либо реализовать подходящий алгоритм машинного обучения, например, «метод трёх соседей»: выбираем для изучаемой точки три ближайших. Ответ для точки будет совпадать с преобладающим ответом среди соседей.

Пример программы, реализующей данный алгоритм на языке Python:

```
1. import sys
2. import math
3. import statistics
4.
5. def sqr(x):
6.     return x * x
7.
8. def solve(data, k = 3):
9.     lines = data.split('\n')
10.     n = int(lines[0])
11.     lines = lines[1:]
12.     p = [0] * n
13.     for i in range(n):
14.         p[i] = list(map(float, lines[i].split()))
15.     res = ''
16.     lines = lines[n:]
17.     m = int(lines[0])
18.     lines = lines[1:]
19.     for i in range(m):
20.         x, y = list(map(float, lines[i].split()))
21.         pp = sorted((sqr(x - xx) + sqr(y - yy), t) for xx, yy, t in p)
22.         ts = [t for d, t in pp[:k]]
23.         res += '{0:.0f}\n'.format(round(statistics.mean(ts)))
24.     return res
25.
26. solve(sys.stdin.read())
```

1.3. Критерии определения призеров и победителей

Количество баллов, набранных при решении всех задач суммируется. Призерам первого отборочного этапа необходимо набрать 7 баллов (для 9 класса) и 9 баллов (для 10-11 класса). Победители первого отборочного этапа должны набрать 20 баллов.

§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет, работы оцениваются автоматически средствами системы онлайн-тестирования. Продолжительность второго отборочного этапа составляет 2 недели. Задачи носят междисциплинарный характер и в более простой форме воссоздают инженерную задачу заключительного этапа. Решение задач предполагало написание программ, допускалось использовать язык программирования Python. Решение каждой задачи дает определенное количество баллов. В данном этапе можно получить суммарно от 0 до 45 баллов.

2.1. Задачи по анализу данных

Задача 2.1.1 (10 баллов)

В летний лагерь приехало 1000 школьников. Когда школьник приезжал, его сразу переписывали (ставили порядковый номер начиная с нуля — каким школьник приехал в лагерь). Школьников сразу разбивали по отрядам с разным количеством человек в отряде: первые n1 школьников определили в первый отряд, следующие n2 школьников — во второй отряд, следующие n3 — в третий и так далее.

Однажды все четные отряды увезли на экскурсию. И в лагерь приехала комиссия и переписала всех школьников, которые остались (каждый школьник назвал номер, каким его записали, когда он приехал в лагерь). По ошибке некоторых школьников переписали несколько раз. Как теперь разобраться, какой школьник в каком отряде?

На вход подается массив из чисел, соответствующих порядковым номерам переписанных школьников, которые остались в лагере.

На выход подаются набор пар чисел: порядковый номер первого и конечного школьника в первом отряде, порядковый номер первого и конечного школьника во втором отряде и так далее**.**

За успешное решение задачи участники получают 10 баллов.

Пример ввода:

```
790 443 801 518 63 75 491 91 809 507 367 778 803 809 923 800 428 857 384
49 358 346 499 931 32 842 965 392 883 873 888 824 480 407 854 435 982 464 907
799 829 454 518 867 346 871 830 90 866 893 455 872 867 815 867 87 877 32 479 834
414 463 972 362 503 481 833 885 366 935 828 495 89 944 30 78 930 964 62 75 893
943 495 357 961 462 830 477 967 841 512 882 888 810 855 837 447 418 434 53 891
998 13 82 794 401 782 34 789 358 506 82 880 997 812 72 24 866 22 444 519 857 972
```

911 50 842 66 826 92 474 0 395 874 964 909 387 454 352 6 862 458 873 512 394 886
449 431 517 994 512 73 986 515 481 444 867 29 998 884 512 401 805 798 36 993 16
38 373 482 926 69 430 818 862 61 369 469 867 851 865 383 84 780 512 438 935 837
901 918 951 426 39 405 365 55 346 502 958 41 345 414 346 441 375 11 391 59 354
939 372 981 379 972 478 948 52 804 56 803 445 413 799 981 12 947 507 443 973 883
999 936 847 940 808 802 906 997 864 79 462 410 850 364 400 392 363 832 982 795
69 424 468 80 394 968 35 817 907 937 975 447 988 363 992 24 474 358 475 456 9
443 496 76 8 7 453 848 19 494 371 10 407 488 508 466 66 823 18 452 91 843 912
454 512 8 940 455 46 835 57 908 66 436 846 460 921 800 866 924 60 483 794 845 11
941 894 365 454 943 40 512 829 853 353 425 32 507 941 39 26 907 85 470 459 778
492 348 925 437 39 497 807 350 37 377 931 894 847 440 962 870 448 866 966 448
454 446 901 950 47 33 890 792 934 793 844 5 358 928 346 452 995 792 401 346 983
509 51 896 346 972 803 474 877 350 879 32 471 832 848 878 829 501 895 776 400
396 775 357 979 996 862 500 10 791 80 41 457 795 876 907 977 512 8 897 833 938
493 432 506 391 963 813 802 68 987 803 794 923 442 391 962 366 24 944 904 979
821 358 964 780 23 850 832 424 889 867 36 48 518 450 363 910 511 447 445 396 374
942 888 954 65 344 356 423 349 88 468 426 930 386 510 445 408 489 32 907 803 16
16 972 967 848 462 20 962 422 988 812 946 92 869 49 886 60 64 443 914 910 395
933 888 462 460 60 458 442 920 429 819 975 972 45 38 347 33 910 990 834 371 900
781 842 857 949 471 985 839 820 421 505 901 822 77 485 932 978 479 416 23 512 16
786 774 392 519 76 797 907 69 864 60 912 909 384 890 982 788 460 445 925 954 415
520 512 836 498 399 859 48 986 830 380 481 433 894 879 422 978 926 344 853 504
455 952 998 813 512 915 871 481 520 347 406 33 361 971 882 53 512 799 351 867 67
817 19 796 387 437 468 389 412 10 388 912 404 371 400 521 993 965 887 433 393
425 355 940 989 480 779 60 1 430 867 794 790 787 837 470 437 348 455 860 44 370
883 478 847 461 844 81 479 39 478 871 462 956 831 67 465 955 427 856 514 901 459
894 489 902 902 837 871 842 937 69 979 991 411 89 519 854 48 508 66 957 501 935
910 391 903 858 15 807 487 993 889 848 963 796 488 484 852 931 995 777 990 459
363 785 43 841 38 352 368 937 915 972 785 929 875 33 784 868 83 797 887 397 905
821 35 32 937 23 376 366 475 402 390 11 787 454 379 925 467 473 50 40 919 32 372
845 901 899 479 834 859 880 861 927 507 897 516 863 937 863 513 381 508 958 849
50 439 360 69 913 840 783 938 839 966 352 953 385 48 49 806 867 801 877 916 956
857 361 783 2 941 801 860 955 366 476 893 23 788 445 60 54 806 778 835 3 28 511
45 990 442 887 26 507 995 382 959 88 814 490 881 972 390 48 894 519 774 417 984
447 917 903 8 71 410 14 35 366 972 512 359 825 848 507 451 488 984 857 974 834
782 838 513 839 966 464 773 898 792 371 816 0 429 976 410 890 811 45 5 390 794
486 512 426 376 782 358 826 454 4 487 812 369 70 809 484 867 437 976 86 368 349
867 960 889 922 10 512 74 349 359 472 993 830 935 861 908 447 42 461 392 822 446
809 792 476 460 21 910 437 47 411 358 964 27 903 393 908 478 848 443 409 453 969
859 403 819 811 974 871 43 799 890 31 794 809 398 378 74 361 985 970 773 25 429
10 980 462 993 777 398 452 860 366 11 379 39 847 494 945 429 58 832 408 495 917

```
17 912 908 788 419 862 48 512 512 502 827 420 371 89 389 453 488 892 932
```

Пример вывода:

```
[(0, 92), (93, 343), (344, 521), (522, 772), (773, 999)]
```

Ограничение по времени исполнения программы: 15 с

Ограничение по использованию оперативной памяти: 256 Мб

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает возвращает набор тестов и правильных ответов:

```
def generate():
    num_tests = 30
    task_size=1000
    tests = list()
    for test in range(num_tests):
        a =[0]*task_size
        a= [i for i in range(0,task_size)]
        groupCount=4
        seqLen=task_size/groupCount

        line=int(seqLen)
        line+=1
        oldLen=len(a)
        for ch in range(int(int(groupCount/2))):
            c = random.randrange(oldLen)
            maxI = int(c+line)
            if (maxI>len(a)):
                maxI=len(a)
            for k in reversed(range(c,maxI)):
                if(len(a)>1):
                    del a[k]

        for ch in range(task_size-len(a)):
            c = random.randrange(len(a))
            a.append(a[c])
        random.shuffle(a)
        tc = ''.join(str(e)+' ' for e in a)
        tc+='\n'
        tests.append(tc)
    return tests

def check(reply, clue):
    return reply == clue
```

РЕШЕНИЕ:

Необходимо понять, что раз номера идут подряд, то после сортировки всех предложенных номеров, заполненные подряд идущие интервалы номеров будут нечетные отряды, а отсутствующие интервалы - четные. После этого существует несколько подходов к решению, но все они так или иначе связаны с сортировкой массива. Один из подходов подразумевает использование сортировки слиянием, добавляя новые элементы в уже отсортированный массив номеров школьников, заполняя при этом интервалы. Второй

состоит в заполнении массива и дальнейшей сортировкой стандартными встроенными алгоритмами и втором проходе по массиву выделяя заполненные интервалы.

Код программы на языке Python:

```
import random
import math
import sys
import array
import random
from collections import OrderedDict
import bisect

def solve(dataset):
    task_size=1000

    ranges=OrderedDict()

    a = dataset.split()

    intEnds=list()
    intBegins=list()

    for item in a:
        c=0
        while((int(c)<int(len(intEnds))) and(int(intEnds[c])<int(item)) ):
            c+=1
        lenEnds=len(intEnds)
        if (int(c) >= int(lenEnds)):
            if (lenEnds>0 and int(intEnds[lenEnds-1]) == int(item)-1):
                intEnds[lenEnds-1]=item
            else:
                intEnds.append(item)
                intBegins.append(item)
        else:
            if (int(intBegins[c])<=int(item)):
                continue
            keyE = intEnds[c]
            keyS = intBegins[c]
            if (int(keyS)==int(item)+1):
                intBegins[c]=item
            else:
                intEnds.insert(c,item)
                intBegins.insert(c,item)
            if (c>0):
                keyP = intEnds[c-1]
                if (int(keyP) == int(item)-1):
                   intBegins[c]=intBegins[c-1]
                   del intBegins[c-1]
                   del intEnds[c-1]
    res1 = list()
    if(int(intBegins[0])>0):
        res1.append((0,int(intBegins[0])))

    for k in range(0,len(intBegins)):
        res1.append((int(intBegins[k]),int(intEnds[k])))
        r1=int(intEnds[k])+1
        r2=task_size
        if(k<len(intBegins)-1):
            r2=int(intBegins[k+1])-1
        if(int(r1)<int(r2)):
```

```
57.             res1.append((int(r1),int(r2)))
58.     return str(res1)
```

Задача 2.1.2 (15 баллов)

Картограф составлял для каждого города список городов, с которыми он связан дорогами. Теперь его спрашивают о том, можно ли проехать между двумя далекими городами.

На вход подается два названия города, которые нужно проверить и дальше идет идет перечисление для городов, с какими другими городами он связан дорогой. Ответ необходимо давать в формате True и False.

За успешное решение задачи участники получают 15 баллов.

Пример ввода:

```
{'find': ('d', 'f'), 'd': ['a', 'b', 'c', 'e', 'f', 'g'], 'b': ['a', 'c', 'd', 'e', 'f', 'g'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u'], 'f': ['a', 'b', 'c', 'd', 'e', 'g'], 'a': ['b', 'c', 'd', 'e', 'f', 'g'], 's': ['n', 'o', 'p', 'q', 'r', 't', 'u', 'c'], 'e': ['a', 'b', 'c', 'd', 'f', 'g'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't'], 'r': ['n', 'o', 'p', 'q', 's', 't', 'u'], 'p': ['n', 'o', 'q', 'r', 's', 't', 'u'], 'c': ['a', 'b', 'd', 'e', 'f', 'g'], 'g': ['a', 'b', 'c', 'd', 'e', 'f'], 'o': ['n', 'p', 'q', 'r', 's', 't', 'u'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u'], 't': ['n', 'o', 'p', 'q', 'r', 's', 'u']}
```

Ограничение по времени исполнения программы: 3000 с

Ограничение по использованию оперативной памяти: 256 Мб

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы. Пример нескольких тестов с правильными ответами представлен ниже:

```
def generate():
    tests = list()
    tests.append("{'find': ('d', 'f'), 'd': ['a', 'b', 'c', 'e', 'f', 'g'], 'b': ['a', 'c', 'd', 'e', 'f', 'g'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u'], 'f': ['a', 'b', 'c', 'd', 'e', 'g'], 'a': ['b', 'c', 'd', 'e', 'f', 'g'], 's': ['n', 'o', 'p', 'q', 'r', 't', 'u', 'c'], 'e': ['a', 'b', 'c', 'd', 'f', 'g'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't'], 'r': ['n', 'o', 'p', 'q', 's', 't', 'u'], 'p': ['n', 'o', 'q', 'r', 's', 't', 'u'], 'c': ['a', 'b', 'd', 'e', 'f', 'g'], 'g': ['a', 'b', 'c', 'd', 'e', 'f'], 'o': ['n', 'p', 'q', 'r', 's', 't', 'u'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u'], 't': ['n', 'o', 'p', 'q', 'r', 's', 'u']}\n")
    tests.append("{'r': ['n', 'o', 'p', 'q', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'v': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'w', 'z', 'y', 'x'], 'e': ['a', 'b', 'c', 'd', 'f'], 'y': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'x'], 'z': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'y', 'x'], 'q': ['n', 'o', 'p', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'f': ['a', 'b', 'c', 'd', 'e'], 'd': ['a', 'b', 'c', 'e', 'f'], 's': ['n', 'o', 'p',
```

```
'q', 'r', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 't': ['n', 'o', 'p', 'q', 'r',
's', 'u', 'v', 'w', 'z', 'y', 'x'], 'n': ['o', 'p', 'q', 'r', 's', 't', 'u',
'v', 'w', 'z', 'y', 'x'], 'w': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v',
'z', 'y', 'x'], 'c': ['a', 'b', 'd', 'e', 'f'], 'a': ['b', 'c', 'd', 'e', 'f'],
'o': ['n', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'find': ('r',
'v'), 'x': ['n', 'o', 'p', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y'], 'p':
['n', 'o', 'q', 'r', 's', 't', 'u', 'v', 'w', 'z', 'y', 'x'], 'b': ['a', 'c',
'd', 'e', 'f'], 'u': ['n', 'o', 'p', 'q', 'r', 's', 't', 'v', 'w', 'z', 'y',
'x', 'c']}\n")
    tests.append("{'s': ['n', 'o', 'p', 'q', 'r'], 'r': ['n', 'o', 'p', 'q',
's'], 'n': ['o', 'p', 'q', 'r', 's'], 'c': ['a', 'b', 'd', 'e', 'f', 'g', 'h',
'i'], 'e': ['a', 'b', 'c', 'd', 'f', 'g', 'h', 'i'], 'i': ['a', 'b', 'c', 'd',
'e', 'f', 'g', 'h'], 'g': ['a', 'b', 'c', 'd', 'e', 'f', 'h', 'i'], 'a': ['b',
'c', 'd', 'e', 'f', 'g', 'h', 'i'], 'o': ['n', 'p', 'q', 'r', 's'], 'find':
('h', 's'), 'h': ['a', 'b', 'c', 'd', 'e', 'f', 'g', 'i'], 'p': ['n', 'o', 'q',
'r', 's'], 'f': ['a', 'b', 'c', 'd', 'e', 'g', 'h', 'i'], 'b': ['a', 'c', 'd',
'e', 'f', 'g', 'h', 'i'], 'd': ['a', 'b', 'c', 'e', 'f', 'g', 'h', 'i'], 'q':
['n', 'o', 'p', 'r', 's']}\n")
#...
    return tests;

def check(reply, clue):
    return str(reply) == str(clue)
```

РЕШЕНИЕ:

Необходимо распознать в задаче проблему поиска пути между двумя вершинами в графе. Школьникам необходимо выбрать способ представления графа в памяти и осуществить поиск в ширину (Bread-first search) от одной вершины к другой. Если алгоритм закончил работу не встретив второй вершины, то пути нет, иначе — есть.

Код решения на языке Python:

```
 1. import ast
 2.
 3. class Graph(object):
 4.
 5.     def __init__(self, graph_dict={}):
 6.         """ initializes a graph object """
 7.         self.__graph_dict = graph_dict
 8.
 9.     def vertices(self):
10.         """ returns the vertices of a graph """
11.         return list(self.__graph_dict.keys())
12.
13.     def edges(self):
14.         """ returns the edges of a graph """
15.         return self.__generate_edges()
16.
17.     def add_vertex(self, vertex):
18.         """ If the vertex "vertex" is not in
19.             self.__graph_dict, a key "vertex" with an empty
20.             list as a value is added to the dictionary.
21.             Otherwise nothing has to be done.
22.         """
23.         if vertex not in self.__graph_dict:
24.             self.__graph_dict[vertex] = []
25.
26.     def add_edge(self, edge):
27.         """ assumes that edge is of type set, tuple or list;
```

```
            between two vertices can be multiple edges!
        """
        edge = set(edge)
        vertex1 = edge.pop()
        if edge:
            # not a loop
            vertex2 = edge.pop()
        else:
            # a loop
            vertex2 = vertex1
        if vertex1 in self.__graph_dict:
            self.__graph_dict[vertex1].append(vertex2)
        else:
            self.__graph_dict[vertex1] = [vertex2]

    def __generate_edges(self):
        """ A static method generating the edges of the
            graph "graph". Edges are represented as sets
            with one (a loop back to the vertex) or two
            vertices
        """
        edges = []
        for vertex in self.__graph_dict:
            for neighbour in self.__graph_dict[vertex]:
                if {neighbour, vertex} not in edges:
                    edges.append({vertex, neighbour})
        return edges

    def __str__(self):
        res = "vertices: "
        for k in self.__graph_dict:
            res += str(k) + " "
        res += "\nedges: "
        for edge in self.__generate_edges():
            res += str(edge) + " "
        return res

    def find_isolated_vertices(self):
        """ returns a list of isolated vertices. """
        graph = self.__graph_dict
        isolated = []
        for vertex in graph:
            print(isolated, vertex)
            if not graph[vertex]:
                isolated += [vertex]
        return isolated

    def find_path(self, start_vertex, end_vertex, path=[]):
        """ find a path from start_vertex to end_vertex
            in graph """
        graph = self.__graph_dict


        if (start_vertex==end_vertex):
            return True
        checked=[]
        added=[]
        checked.append(start_vertex)
        added.append(start_vertex)
        res=False
        while (len(added)>0):
```

```
            newAdded=[]
            for vert in added:
               for newVert in graph[vert]:
                  if newVert not in checked:
                      if newVert == end_vertex:
                          res=True
                          break
                      checked.append(newVert)
                      newAdded.append(newVert)
            added=newAdded
        return res


        path = path + [start_vertex]
        if start_vertex == end_vertex:
            return path
        if start_vertex not in graph:
            return None
        for vertex in graph[start_vertex]:
            if vertex not in path:
                extended_path = self.find_path(vertex,
                                               end_vertex,
                                               path)
                if extended_path:
                    return extended_path
        return None


    def find_all_paths(self, start_vertex, end_vertex, path=[]):
        """ find all paths from start_vertex to
            end_vertex in graph """
        graph = self.__graph_dict
        path.append(start_vertex)
        if start_vertex == end_vertex:
            return [path]
        if start_vertex not in graph:
            return []
        paths = []
        nextPath=path
        for vertex in graph[start_vertex]:
            nextPath.append(vertex)

        for vertex in graph[start_vertex]:
            if vertex not in path :
                extended_paths = self.find_all_paths(vertex,
                                                     end_vertex,
                                                     nextPath)
                for p in extended_paths:
                    paths.append(p)
        return paths

    def is_connected(self,
                     vertices_encountered = None,
                     start_vertex=None):
        """ determines if the graph is connected """
        if vertices_encountered is None:
            vertices_encountered = set()
        gdict = self.__graph_dict
        vertices = list(gdict.keys()) # "list" necessary in Python 3
        if not start_vertex:
            # chosse a vertex from graph as a starting point
```

```
            start_vertex = vertices[0]
        vertices_encountered.add(start_vertex)
        if len(vertices_encountered) != len(vertices):
            for vertex in gdict[start_vertex]:
                if vertex not in vertices_encountered:
                    if self.is_connected(vertices_encountered, vertex):
                        return True
        else:
            return True
        return False

    def vertex_degree(self, vertex):
        """ The degree of a vertex is the number of edges connecting
            it, i.e. the number of adjacent vertices. Loops are counted
            double, i.e. every occurence of vertex in the list
            of adjacent vertices. """
        adj_vertices =  self.__graph_dict[vertex]
        degree = len(adj_vertices) + adj_vertices.count(vertex)
        return degree

    def degree_sequence(self):
        """ calculates the degree sequence """
        seq = []
        for vertex in self.__graph_dict:
            seq.append(self.vertex_degree(vertex))
        seq.sort(reverse=True)
        return tuple(seq)

    @staticmethod
    def is_degree_sequence(sequence):
        """ Method returns True, if the sequence "sequence" is a
            degree sequence, i.e. a non-increasing sequence.
            Otherwise False is returned.
        """
        # check if the sequence sequence is non-increasing:
        return all( x>=y for x, y in zip(sequence, sequence[1:]))


    def delta(self):
        """ the minimum degree of the vertices """
        min = 100000000
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree < min:
                min = vertex_degree
        return min

    def Delta(self):
        """ the maximum degree of the vertices """
        max = 0
        for vertex in self.__graph_dict:
            vertex_degree = self.vertex_degree(vertex)
            if vertex_degree > max:
                max = vertex_degree
        return max

    def density(self):
        """ method to calculate the density of a graph """
        g = self.__graph_dict
        V = len(g.keys())
        E = len(self.edges())
```

```
211.         return 2.0 * E / (V *(V - 1))
212.
213.     def diameter(self):
214.         """ calculates the diameter of the graph """
215.
216.         v = self.vertices()
217.         pairs = [ (v[i],v[j]) for i in range(len(v)) for j in range(i+1,
len(v)-1)]
218.         smallest_paths = []
219.         for (s,e) in pairs:
220.             paths = self.find_all_paths(s,e)
221.             smallest = sorted(paths, key=len)[0]
222.             smallest_paths.append(smallest)
223.
224.         smallest_paths.sort(key=len)
225.
226.         # longest path is at the end of list,
227.         # i.e. diameter corresponds to the length of this path
228.         print(smallest_paths[-1])
229.         diameter = len(smallest_paths[-1])
230.         return diameter
231.
232.     @staticmethod
233.     def erdoes_gallai(dsequence):
234.         """ Checks if the condition of the Erdoes-Gallai inequality
235.             is fullfilled
236.         """
237.         if sum(dsequence) % 2:
238.             # sum of sequence is odd
239.             return False
240.         if Graph.is_degree_sequence(dsequence):
241.             for k in range(1,len(dsequence) + 1):
242.                 left = sum(dsequence[:k])
243.                 right =  k * (k-1) + sum([min(x,k) for x in dsequence[k:]])
244.                 if left > right:
245.                     return False
246.         else:
247.             # sequence is increasing
248.             return False
249.         return True
250.
251. def solve(dataset):
252.     g = ast.literal_eval(dataset);
253.     a, b = g['find']
254.     g.pop("find", None)
255.     graph = Graph(g)
256.     return str(graph.find_path(a, b));
```

Задача 2.1.3 (0-20 баллов)

В задаче фигурирует придуманный нами граф социальной сети (для каждого пользователя указано, какие пользователя добавили его в друзья). Для каждого пользователя вычисляется его популярность, она основана на том, сколько людей дружит с теми пользователями, с которым он дружит.

Популярность вычисляется как X - суммарное количество людей, которые дружат с его друзьями, считая его самого.

Необходимо рассчитать два перцентиль 50 и 90, т.е. два наименьших значения популярности такие, что с вероятностью 50% для первого и 90% для второго популярность случайного пользователя будет меньше данного значения.

Задачу необходимо решить у себя на компьютере и в систему загрузить решение в виде двух чисел для каждой задачи.

Если для всех тестов вы посчитали хотя бы перцентиль 50 (с вероятностью 50 процентов популярность окажется меньше), то вы получаете половину баллов от задачи.

Пример ввода:

```
[{0: [5, 8], 1: [7, 2], 2: [8], 3: [10, 0], 4: [0, 10, 2, 1], 5: [1, 5, 3, 7], 6: [7, 3, 0], 7: [12, 13, 0, 8], 8: [8, 11], 9: [6, 2, 13], 10: [13], 11: [2, 0, 8], 12: [0, 13], 13: [4, 11, 8], 14: [0, 4, 12, 2]}, {0: [14, 11], 1: [7, 14, 1], 2: [0], 3: [10], 4: [13], 5: [8, 0], 6: [5, 3], 7: [2, 11, 8, 10], 8: [3, 10, 14, 7], 9: [0, 11, 7, 4], 10: [1, 7], 11: [10, 5, 12, 4], 12: [14, 5], 13: [7, 6, 3, 1], 14: [10, 6, 0, 8]}]
```

Пример вывода:

```
[(4, 6), (7, 9)]
```

Ограничение по использованию оперативной памяти: 256 Мб

Время одной попытки: 5 мин

СПОСОБ И КРИТЕРИЙ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    numOfTests=3
    maxFrCount=200
    numOfV = 10000
    avgFr = 7
    dispersion = 4

    testsList=list()
    for test in range(numOfTests):
        graph={}
        for v in range(numOfV):
            dispVal = int(random.randrange(dispersion) - (int(dispersion)//2))
            sign = random.randrange(2)
            frCount = 0
            if int(sign) == 0:
                dispVal = int(random.randrange(avgFr))
                frCount = int(avgFr )- int(dispVal)
            else:
                dispVal = int(random.randrange(int(maxFrCount) - int(avgFr )))
                frCount=  int(avgFr) +int(dispVal)
            frList=list()
            for frN in range(frCount):

                frN=random.randrange(numOfV)
                while frN in frList:
```

```
                    frN=random.randrange(numOfV)
                frList.append(frN)
            graph.update({v:frList})
        testsList.append(graph)
        return str(testsList)

def check(reply, clue):
    repA= ast.literal_eval(reply)
    clA = ast.literal_eval(clue)
    if len(repA)!=len(clA):
        return False

    resHalf=True
    resFull=True
    for ind in range(len(repA)):
        rep=repA[ind]
        cl=clA[ind]
```

Функция проверки `check` возвращает показатель точности решения (от 0 до 1). Это значение умножается на 20, что дает число баллов, полученных командой.

РЕШЕНИЕ:

Для данное задачи необходимо сначала принять решение о внутреннем представлении данных графа социальной сети. После этого для всех пользователей рассчитать значение популярности. Затем необходимо отсортировать пользователей по популярности (основываясь на знаниях из первой задачи) и взять соотвествующий элемент массива (средний для перцентиля 50 и находящийся на позиции 0.9*N где N-число элементов массива, для перцентиля 90).

Код программы на языке Python:

```
 1. import random
 2. import ast
 3. import math
 4.
 5. def solve(dataset):
 6.     checkTests = ast.literal_eval(dataset)
 7.     answ=list()
 8.     for test in checkTests:
 9.         testLen = len(test)
10.         valList=list()
11.         for v in range(testLen):
12.
13.
14.             frVList=test[v]
15.             frfrList=list()
16.             for frV in frVList:
17.                 frfrList.extend(test[frV])
18.             frfrList=set(frfrList)
19.             val = int(len(frfrList))
20.
21.             uppId=int(len(valList))
22.             lowId=0
23.             found=False
24.             while int(math.fabs(int(uppId)-int(lowId)))>1:
```

```
                med=int((int(uppId) + int(lowId))//2)
                if int(valList[med])<int(val):
                    lowId=med
                else:
                    uppId=med
                    if int(valList[med])== int(val):
                        break
            if (len(valList)>0 and int(val) > int(valList[lowId])):
                valList.insert(uppId,val)
            else:
                valList.insert(lowId,val)

        res1,mod=divmod(testLen,2)
        if mod>0:
            res1+=0
        res1-=1
        res2,mod=divmod(testLen*9,10)

        if (mod>0):
            res2+=0
        res2-=1

        answ.append((valList[res1],valList[res2]))
    return(str(answ))
```

2.2. Критерии определения призеров и победителей

Количество баллов, набранных при решении всех задач, суммируется. Призерам второго отборочного этапа было необходимо набрать 10 баллов (для учащихся 9 класса) и 15 баллов (для учащихся 10-11 класса). Победители второго отборочного этапа должны были набрать 45 баллов.

§3 Заключительный этап: индивидуальная часть

Заключительный этап олимпиады состоит из двух частей: индивидуальное решение задач по предметам (математика, информатика) и командное решение инженерное задачи. На индивидуальное решение задач дается по 2 часа на один предмет. Задачи по математике и информатике общие на параллели 9 и 10-11 класс. Решение каждой задачи дает определенное количество баллов (см. критерии оценки далее). По математике за каждую задачу можно получить от 0 до указанного количества баллов в соответствии с описанными критериями. Баллы по информатике зачисляются в полном объеме за правильное решение задачи. Решение задач по информатике подразумевало написание задач на языке Python. Участники получают оценку за решение задач в совокупности по всем предметам данного профиля (математика и информатика) — суммарно от 0 до 24 баллов.

3.1. Задачи по математике

Задача 3.1.1 (макс. 6 баллов)

Группа психологов разработала тест, пройдя который, каждый человек получает оценку – число Q – показатель его умственных способностей (чем больше Q, тем больше способности). За рейтинг страны принимается среднее арифметическое значений Q всех жителей этой страны.

Задание 3.1.1а (1 балл). Группа граждан страны А эмигрировала в страну Б. Покажите, что при этом у обеих стран мог вырасти рейтинг.

Задание 3.1.1б (3 балла). После этого группа граждан страны Б (в числе которых могут быть и бывшие эмигранты из А) эмигрировала в страну А. Возможно ли, что рейтинги обеих стран опять выросли?

Задание 3.1.1в (2 балла). Группа граждан страны А эмигрировала в страну Б, а группа граждан Б – в страну В. В результате этого рейтинги каждой страны оказались выше первоначальных. После этого направление миграционных потоков изменилось на противоположное — часть жителей В переехала в Б, а часть жителей Б – в А. Оказалось, что в результате рейтинги всех трех стран опять выросли (по сравнению с теми, которые были после первого переезда, но до начала второго). (Так, во всяком случае, утверждают информационные агентства этих стран.) Может ли такое быть (если да, то как, если нет, то почему)?

(Предполагается, что за рассматриваемое время Q граждан не изменилось, никто не умер и не родился.)

РЕШЕНИЕ:

а) Пусть, например, в А жили всего два человека с показателями 3 и 5, а в Б – один человек с показателем 1. После переезда человека с показателем 3 из А в Б в обеих странах рейтинг повысится.

б) Сначала заметим, что если все население страны разбито на две группы X и Y с рейтингами соответственно Q_X и Q_Y, то рейтинг Q всей страны находится между Q_X и Q_Y (причем равенство будет только в случае $Q_X = Q_Y$).

Обозначим через a и b рейтинги стран А и Б до эмиграции из А в Б, через a_1 и b_1 – рейтинги этих стран после этой эмиграции, а через c – рейтинг группы эмигрантов. По условию $a < a_1$. Отсюда, как показано выше, следует, что $c < a < a_1$ (до эмиграции А разбита на группу эмигрантов с рейтингом с и группу остающихся с рейтингом a_1).

Аналогично $b < b_1 < c$. Итак, $b < a$ и $b_1 < a_1$. Первое неравенство показывает, что увеличение рейтингов обеих стран возможно только при эмиграции из страны с большим рейтингом в страну с меньшим рейтингом. Второе неравенство показывает, что рейтинг страны А остался выше рейтинга страны Б. Таким образом, одновременное увеличение рейтингов при эмиграции из Б в А невозможно.

в) Пусть в стране А всего два жителя с показателями $Q = 1$ и 2, в стране Б – четыре жителя ($Q = 2, 2, 4, 10$), в стране В – один житель ($Q = 1$). При первой волне эмиграции из А в Б эмигрировал один человек с $Q = 1$, а из Б в В – двое с $Q = 2$. При второй волне из В в Б переехал один человек с $Q = 1$, а из Б в А – двое с $Q = 1$ и 4. Рейтинги стран при этом менялись так: А – $1,5 \to 2 \to 2^1/_3$; Б – $4,5 \to 5 \to 5,5$; В – $1 \to 1^2/_3 \to 2$.

Критерии оценки:

- (а) приведен любой верный пример - 1 балл
- (б) доказательство основывается на неверном утверждении, но может быть доведено до правильного - 1 балл
  - в доказательстве используются верные утверждения, которые не доказаны - 2 балла
  - полное доказательство - 3 балла
- (в) приведен пример, как такое возможно - 2 балла

Задача 3.1.2 (6 баллов)

Администрация социальной сети ВКонтакте решила создать сообщество «Всех тех, у кого меньше половины друзей состоит в этом сообществе». Для этого им нужно включить в

сообщество пользователей так, чтобы в итоге:

- у всех, кто в этом сообществе, меньше половины друзей были в нем же;
- у всех, кто не в этом сообществе, не меньше половины друзей были в нем.

Всегда ли им удастся создать такое сообщество?

(Предполагается, что пользователи не сами вступают в сообщество, а распределяются администрацией социальной сети)

РЕШЕНИЕ:

Представим социальную сеть в виде графа. Вершины – пользователи, ребра – отношения дружбы. Тогда нам надо доказать, что любой граф можно покрасить в два цвета: синий и красный, так чтобы

- у синих вершин меньше половины соседей были синими
- у красных вершин не меньше половин соседей были синими

Покрасим граф как-нибудь. Теперь будем перекрашивать вершины, для которых не выполнено условие.

Если перекрашиваем красную вершину (у нее меньше половины синих соседей), количество ребер между синими и красными увеличивается.

Если перекрашиваем синюю (у менее не меньше половины синих соседей), количество ребер между синими и красными не уменьшается, а количество красных увеличивается. Поэтому перекраска не может идти бесконечно. Значит в какой-то момент для всех вершин будут выполнены необходимые условия.

Ответ: да, всегда

Критерии оценки:

- доказано, что в любой сети такое возможно — 6 баллов
- есть верные идеи инварианта, но в доказательстве есть пробелы — 3 балла

3.2. Задачи по информатике

Задача 3.2.1 «Всюду реклама» (1 балл)

Когда-то давно Паша создал свой интернет-портал. Теперь его сайт стал достаточно популярным, и молодой человек решил, что пришло время его монетизировать. Если конкретнее, он решил вводить рекламу. Стандартным решениям Паша не доверяет, поэтому решил написать свою небольшую рекламную платформу.

На сайте есть N рекламных блоков. Про каждый блок известна его выгодность v_i, а

именно, сколько раз на него кликнули за последний месяц(количество кликов не зависит от содержания объявления). Через форму заявок Паше поступило M рекламных объявлений от рекламодателей. Каждый рекламодатель указывает для своего объявления цену c_i в рублях, которую он готов заплатить за 100 кликов.

Паша считает, что чем больше рекламодатель готов заплатить, тем более выгодное место его объявление должно занимать. Если объявлений слишком много, то объявления с очень маленькой ценой показаны не будут, если объявлений слишком мало, то самые невыгодные блоки останутся пустыми.

От вас требуется сопоставить рекламные объявления и рекламные блоки на сайте в соответствии с Пашиными принципами.

Формат входных данных:

В первой строке даны два числа $0 \le N,M \le 10^5$. N — количество рекламных блоков на сайте и M — количество рекламных объявлений. Во второй строке через пробел даны N чисел vivi. Число $0 \le v_i \le 5000$ характеризует выгодность i-ого рекламного объявления. В третьей строке через пробел даны M чисел $0 \le c_j \le 10000$. Число c_j показывает цену, которую готов заплатить рекламодатель за 100 кликов на j-ое объявление.

Формат выходных данных:

Выведите N строк, в каждой строке выведите пару чисел: выгодность рекламного блока и цену за 100 кликов соответствующего ему рекламного объявления, которое будет располагаться в этом блоке. Если на данное место не хватило рекламного объявления, вместо номера объявления укажите -1.

Если решений больше одного, выведите любое.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    num_tests = 10
    tests = []
    n = random.randint(4, 15)
    m = random.randint(1, n-1)
    list_v = []
    list_c = []
    for i in range(n):
        list_v.append(random.randint(0, 5000))
    for i in range(m):
        list_c.append(random.randint(0, 10000))
    strc = ' '.join(str(c) for c in list_c)
    strv = ' '.join(str(v) for v in list_v)
    test_case = "{} {}\n{}\n{}\n".format(n, m, strv, strc)
    tests.append(test_case)
    for test in range(num_tests):
```

```
        n = random.randint(1, 100000)
        m = random.randint(1, 100000)
        list_v = []
        list_c = []
        for i in range(n):
            list_v.append(random.randint(0, 5000))
        for i in range(m):
            list_c.append(random.randint(0, 10000))
        strc = ' '.join(str(c) for c in list_c)
        strv = ' '.join(str(v) for v in list_v)
        test_case = "{} {}\n{}\n{}\n".format(n, m, strv, strc)
        tests.append(test_case)
    return tests



def check(reply, clue):
    replines = reply.splitlines()
    clulines = clue.splitlines()
    replines.sort()
    clulines.sort()
    if replines != clulines:
        return False
    return True
```

РЕШЕНИЕ:

В данной задаче работает идея жадных алгоритмов. По условиям мы сопоставляем максимальный элемент из набора объявлений с максимальным элементом из набора рекламных блоков. Второе по цене объявление располагаем на второе по стоимости место. И так далее. Следовательно, достаточно отсортировать оба массива по возрастанию и взять первые N пар элементов. Необходимо учитывать, что если M < N, То в последних M-N записях необходимо вместо стоимости с выводить -1.

Реализация этого алгоритма на языке программирования Python:

```
1. n, m = map(int, input().split())
2. v = sorted(map(int, input().split()), reverse=True)
3. c = sorted(map(int, input().split()), reverse=True)
4. ans = ["{} {}".format(v[i], c[i] if i < m else -1) for i in range(n)]
5. print("\n".join(ans))
```

Задача 3.2.2 «Кто на свете всех умнее?» (1 балл)

Исследование данных требует внимания к деталям и усидчивости, пусть и не очень большой. Сегодня к вам в руки попал файл с данными, которые управление районом собирает обо всех учениках. Многие из этих данных могут приоткрыть специалистам информацию о том, какие факторы и как влияют на хорошее обучение учеников.

На данный момент управление районом интересует два вопроса:

во-первых, ему важно знать средний возраст школьников, которые ходят на дополнительные занятия в школах. А во-вторых, фамилию и имя школьника, с максимальным баллом по химии, среди тех, кто учится в 8 или 9 классе.

Формат входных данных:

В первой строке задано число школьников 1≤N≤10^5, чьи данные собрала школа. Следующие N строк содержат данные об учениках. Каждая строка имеет вид ['Имя', 'Фамилия', 'Отчество', число полных лет, ['школа (номер или название)', класс, тип каникул ('t'/'q')], [средний балл по математике(число от 1 до 10), средний балл по русскому языку (число от 1 до 10), средний балл по физике (число от 1 до 10), средний балл по химии (число от 1 до 10), средний балл по биологии (число от 1 до 10)], количество дополнительных занятий в школе, количество детей в семье ребенка]**.**

Формат выходных данных:

В первой строке выведите средний возраст школьников, которые ходят на дополнительные занятия в школе, округленное вниз до ближайшего целого числа. Во второй строке через пробел выведите фамилию и имя школьника, с максимальным баллом по химии, среди тех, кто учится в 8 или 9 классе. Если детей с одинаковым баллом несколько, выведите того, кто в файле записан раньше. Если ответ на какой-то вопрос не может быть корректно получен, замените отсутствующий ответ на строку "No answer".

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    num_tests = 10
    tests = ["4\n['Aleksandr', 'Ivanov', 'Sergeevich', 12, ['1383', 6, 't'], [9,
7, 5, 9, 6], 1, 2]\n['Petr', 'Ivanov', 'Sergeevich', 14, ['1383', 8, 'q'], [3,
4, 5, 5, 6], 3, 2]\n['Ivan', 'Smirnov', 'Petrovich', 9, ['1383', 4, 't'], [9, 9,
10, 9, 10], 0, 1]\n['Anna', 'Eremina', 'Vladimirovna', 13, ['1383', 7, 't'],
[10, 7, 9, 4, 4], 1, 3]\n", "1\n['Aleksandr', 'Ivanov', 'Sergeevich', 12,
['1383', 6, 't'], [9, 7, 5, 9, 6], 0, 2]\n", "1\n['Aleksandr', 'Ivanov',
'Sergeevich', 12, ['1383', 6, 't'], [9, 7, 5, 9, 6], 1, 2]\n",
             "1\n['Aleksandr', 'Ivanov', 'Sergeevich', 12, ['1383', 8, 't'], [9,
7, 5, 9, 6], 0, 2]\n"]

    for test in range(num_tests):
        strin = []
        n = random.randint(3, 100000)
        for kind in range(n):
            name_len = random.randint(4, 10)
            name = ''.join([random.choice('qwertyuiopasdfghjklzxcvbnm') for let
in range(name_len)])
            fname_len = random.randint(8, 15)
            fname = ''.join([random.choice('qwertyuiopasdfghjklzxcvbnm') for let
in range(fname_len)])
            pname_len = random.randint(8, 15)
            pname = ''.join([random.choice('qwertyuiopasdfghjklzxcvbnm') for let
in range(pname_len)])
            age = random.randint(5, 21)
            school = str(random.randint(1, 9999))
            clas = random.randint(1, 11)
```

```
            vac = random.choice(['t', 'q'])
            math = random.randint(1, 10)
            rus = random.randint(1, 10)
            phis = random.randint(1, 10)
            him = random.randint(1, 10)
            bio = random.randint(1, 10)
            dops = random.randint(0, 5)
            ch = random.randint(1, 6)
            strin.append("['{}', '{}', '{}', {}, [{}, {}, '{}'], [{}, {}, {},
{}, {}], {}, {}]".format(name, fname, pname, age, school, clas, vac,  math, rus,
phis, him, bio, dops, ch))
        tests.append("{}\n{}\n".format(n, '\n'.join(strin)))
    return tests

def check(reply, clue):
    return reply == clue
```

РЕШЕНИЕ:

Так как задача поиска максимального, минимального и среднего значения на неотсортированном потоке данных может быть решена за время сравнимое со временем считывания, хорошее решение построчно считывает данные о школьниках. Перед считывание k-ой строки хорошее решение знает ответ для файла состоящего из первых k-1 строк исходного файла. На k-ом шаге необходимо выделить данные из строки и обновить следующие переменные в соответствии с условиями:

- количество школьников, которые ходят на дополнительные занятия;
- суммарный возраст школьников, которые ходят на дополнительные занятия;
- фамилия и имя восьми-/девятиклассника с максимальным баллом по химии;
- максимальный балл по химии среди восьми-/девятиклассников.

Когда обработаны все записи, необходимо проверить, что существует хотя бы один 8/9классник и, что существует хотя бы один школьник, который ходит на допзанятия. Если таких школьников не существует, то необходимо вывести “No answer” на соответствующий вопрос.

Реализация этого алгоритма на языке программирования Python:

```
01. n = int(input())
02. num_task_1 = 0
03. age_task_1 = 0
04. max_task_2 = -1
05. name_task_2 = ''
06.
07. for i in range(n):
08.     tmp_str = input().replace('[', '').replace(']', '').replace(',', '')
09.     arr = list(tmp_str.split())
10.     if int(arr[12]) > 0:
11.         num_task_1 += 1
12.         age_task_1 += int(arr[3])
13.     if arr[5] in ('8', '9'):
14.         if int(arr[-4]) > max_task_2:
15.             max_task_2 = int(arr[-4])
```

```
16.             name_task_2 = "{} {}".format(arr[0].replace("'", ''),
arr[1].replace("'", ''))
17. ans = []
18. ans.append("No answer" if num_task_1 == 0 else str(age_task_1 //
num_task_1))
19. ans.append("No answer" if max_task_2 == -1 else name_task_2)
20. print('\n'.join(ans))
```

Задача 3.2.3 «Субботник» (3 балла)

На следующих выходных ученики всех классов отправятся на субботник в парк. Ребята хотят сами разделиться на команды. Было решено, что для того чтобы попасть в определенную группу, у школьника обязательно должен быть в этой группе хотя бы один друг. Считается, что если первый школьник дружит со вторым, то и второй дружит с первым. Каждая команда поедет на субботник на отдельном автобусе, поэтому количество команд должно быть минимальным.

Осталось только заказать автобусы и составить списки для каждого автобуса, кто из учеников в нем едет. Составьте программу, которая решит эту задачу.

Формат входных данных:

В первой строке задано два числа: N — число школьников, M — число строк, описывающих отношения дружбы между школьниками $0 \le N,M \le 1000$. В следующих M строках даны пары натуральных чисел $1 \le a,b \le N$, означающие что школьник номер a дружит со школьником под номером b.

Формат выходных данных:

В первой строке выведите T — число автобусов. В следующих T строках выведите информацию о школьниках, которые едут в автобусах, в порядке возрастания номеров школьников. Во второй строке укажите в квадратных скобках число школьников, далее через пробел в порядке возрастания номера школьников, которые едут в первом автобусе, если необходимо, в третьей строке — в квадратных скобках число школьников, далее номера школьников, которые едут во втором автобусе, и так далее. Каждый школьник должен присутствовать в одном и только одном автобусе. Если возможно несколько ответов, выведите любой.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    num_tests = 10
    tests = ["6 4\n4 2\n1 4\n6 4\n3 6\n", "6 4\n3 1\n1 2\n5 4\n2 3\n", "6 4\n4
1\n1 3\n6 2\n2 3\n"]
    for i in range(num_tests):
```

```
        n = random.randint(110, 1000)
        m = random.randint(0, 1000)
        tmp = []
        for j in range(m):
            a = random.randint(1, n - 100)
            b = random.randint(a + 1, n)
            tmp.append("{} {}".format(a, b))
        tests.append("{} {}\n{}\n".format(n, m, '\n'.join(tmp)))
    return tests

def check(reply, clue):
    stripper = lambda s: re.sub(r'^\s*|\s*$', '', s)
    squeezer = lambda s: re.sub(r'\s+', ' ', s)
    lines_reply = list(map(lambda s: squeezer(stripper(s)), reply.splitlines()))
    lines_clue = list(map(lambda s: squeezer(stripper(s)), clue.splitlines()))
    if int(lines_reply[0]) != int(lines_clue[0]):
        return False
    lines_rep = sorted(lines_reply[1:])
    lines_clu = sorted(lines_clue[1:])
    return lines_rep == lines_clu
```

РЕШЕНИЕ:

У автобусов нет максимального ограничения по количеству учеников, которые в нем едут. Из этой посылки и условия, что в решении должно быть минимальное число автобусов, легко вывести идею, что оптимальное решение не будет разделять ни одну пару школьников, которые между собой дружат.

Таким образом суть задачи в выделении компонент связности для неориентированного графа. Эту задачу можно решать, например, используя поиск в ширину или поиск в глубину.

Пример программы, реализующей этот алгоритм, на языке Python:

```
01. def dfs(x):
02.     used[x] = 1
03.     c.append(x+1)
04.     for i in gr[x]:
05.         if used[i] == 0:
06.             dfs(i)
07.
08. n, m = map(int, input().split())
09. gr = [[] for i in range(n)]
10. for i in range(m):
11.     a, b = map(int, input().split())
12.     gr[a - 1].append(b - 1)
13.     gr[b - 1].append(a - 1)
14. used = [0] * n
15. ans = 0
16. answer_list = []
17. for i in range(n):
18.     if used[i] == 0:
19.         ans += 1
20.         c = []
21.         dfs(i)
22.         answer_list.append(c)
23. res = []
24. res.append(str(ans))
```

```
25. for al in answer_list:
26.     al.sort()
27.     res.append("[{}] {}".format(len(al), ' '.join([str(tmp) for tmp in
al])))
28. print("\n".join(res))
```

Задача 3.2.4 «Итоговая аттестация» (3 балла)

Конец года — беспокойное время не только для школьников, которые готовятся к экзаменам, но и для составителей экзаменационных заданий. Составляя любой тест, необходимо учитывать, насколько сложной будет задача для школьников, и определить, сколько учащихся сдадут тест успешно.

В этом году было решено провести тестовый экзамен, пригласив 100 учеников разных школ решить 5 задач. Каждая задача оценивается в ai баллов. Задача либо решена на полный балл, либо не решена совсем, а значит за нее не начисляются баллы. Частичные решения проверяющие не учитывают. После экзамена составители получили результаты школьников. Для каждого школьника известны результаты проверки всех задач.

Необходимо посчитать, сколько школьников получит не менее K баллов, если экзамен будут сдавать 1000000 школьников.

Обратите внимание, что невозможно достаточно надежно найти вероятность решить определенный набор задач, но будем считать, что возможно достаточно надежно оценить вероятность решить одну задачу.

Формат входных данных:

В первой строке дано число K — число баллов, необходимое для успешной сдачи теста. Во второй строке 5 натуральных чисел — баллы за задачи. Первое число соответствует баллам за первую задачу, второе — за вторую и так далее.

Далее следует 100 строк. В каждой строке 5 чисел, обозначающих, решена ли соответствующая по номеру задача или нет. На первом месте в строке указано решена ли первая задача, на втором решена ли вторая, и так далее. Если задача решена, то в строке будет указана 1, если нет — 0.

Формат выходных данных:

В единственной строке выведите ожидаемое число людей, которые успешно сдадут тот же тест если решать его будет 1000000 школьников.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы:

```
def generate():
    num_tests = 10
```

```
    tests = []
    for test in range(num_tests):
        num_tasks = 5
        points = []
        for i in range(num_tasks):
            points.append(random.randint(1, 100))
        sum_points = sum(points)
        results = []
        for i in range(100):
            results.append(' '.join([random.choice(['0', '1']) for i in
range(num_tasks)]))
        good_points = random.randint(0, sum_points)
        test_case = "{}\n{}\n{}\n".format(good_points, ' '.join([str(point) for
point in points]), '\n'.join(results))
        tests.append(test_case)
    return tests

def check(reply, clue):
    return int(reply) - int(clue) < int(2)
```

РЕШЕНИЕ:

Считаем, сколько участников решили каждую задачу. Из этого определяем вероятность решить определенную задачу.

Исходя из этой информации вычисляем вероятность решить определенный набор задач. Если этот набор задач соответствует успешному прохождению тестирования, увеличиваем суммарную вероятность успешно пройти тест.

Пример правильной программы на языке Python:

```
01. good_point = int(input())
02. points = list(map(int, input().split()))
03. tasks_res = [0] * 5
04. p = [0] * 5
05. for i in range(100):
06.     sh_res = list(map(int, input().split()))
07.     for j in range(5):
08.         tasks_res[j] += sh_res[j]
09. for i in range(5):
10.     p[i] = tasks_res[i] / 100
11. sum_p = 0
12. for a0 in (0, 1):
13.     for a1 in (0, 1):
14.         for a2 in (0, 1):
15.             for a3 in (0, 1):
16.                 for a4 in (0, 1):
17.                     if a0 * points[0] + a1 * points[1] + a2 * points[2] + a3
* points[3] + a4 * points[4] >= good_point:
18.                         tmp = 1
19.                         tmp =(tmp*(1-p[0])) if a0==0 else (tmp*p[0])
20.                         tmp =(tmp*(1-p[1])) if a1==0 else (tmp*p[1])
21.                         tmp =(tmp*(1-p[2])) if a2==0 else (tmp*p[2])
22.                         tmp =(tmp*(1-p[3])) if a3==0 else (tmp*p[3])
23.                         tmp =(tmp*(1-p[4])) if a4==0 else (tmp*p[4])
24.                         sum_p += tmp
25. print(int(1000000*sum_p))
```

Задача 3.2.5 «Минимальное остовное дерево» (4 балла)

Когда мы работаем с неориентированными графами, часто возникает следующая задача: сделать граф как можно меньше, но так, чтобы все вершины остались соединены друг с другом. Например, представьте, что вы хотите построить железную дорогу между несколькими городами и вам принесли огромную карту, на которой проложены все возможные варианты железнодорожных путей и цены их постройки. Вы хотите из всех путей взять минимально необходимый набор, чтобы все города были связаны друг с другом, но сама постройка железнодорожной сети обошлась как можно дешевле.

Города и дороги образуют взвешенный неориентированный граф. Города - вершины графа, дороги - ребра, цены дорог - веса ребер. Теперь наша задача свелась к чистой математике: нам нужно выбрать в графе некоторый набор ребер, такой, что граф остается связным, а суммарный вес ребер наименьший возможный.

Нетрудно заметить, что для того, чтобы соединить N городов, нам необходима N−1 дорога. Также понятно, что если какие-то из выбранных дорог образуют цикл, значит можно было какую-то из дорог в этом цикле не строить, но транспортная сеть осталась бы связной. А раз ее можно было не строить, значит результат можно было бы получить дешевле. Связные графы, в которых число ребер на единицу меньше числа вершин и не имеют циклов, называются деревьями. Значит нам нужно получить как раз дерево, но не любое, а самое дешевое. Самое дешевое дерево, соединяющее все вершины графа, называется минимальным остовным деревом.

Для того, чтобы построить минимальное остовное дерево, можно воспользоваться алгоритмом Крускала. Его идея очень простая: мы будем добавлять дороги по одной, начиная с самой дешевой. Но если дорога соединяет какие-то города, между которыми уже есть проезд, то эта дорога бесполезна (она создаст цикл) и брать ее мы не будем, а возьмем следующую по цене дорогу. Через N−1 шаг мы объединим все N городов, если это было возможно. Теперь разберем алгоритм чуть детальнее. Вашей задачей будет реализовать его в виде программы на Python.

На каждой итерации нашего алгоритма города и построенные дороги образуют несколько несвязных. Изначально эти деревья очень маленькие, каждое дерево состоит из одной единственной вершины-города, а ребер-дорог никаких нет. Пронумеруем города. Также мы будем нумеровать деревья и про каждую вершину будем хранить информацию о том, к какому дереву она относится (нам это нужно, чтобы точно понимать, какие вершины соединять нет необходимости). Изначально каждая вершина принадлежит номеру с таким же номером, как у самой вершины, эту информацию мы сохраним в специальный массив, в

котором будут храниться номера деревьев каждой вершины. На каждой итерации мы будем объединять ребром два дерева в одно; при этом нам придется менять этот массив, чтобы показать, что вершины из объединенных деревьев теперь принадлежат одному дереву. Например, мы можем взять все вершины из второго дерева (мы ведь знаем номер этого дерева) и указать, что теперь эти вершины принадлежат первому дереву (для этого достаточно в массиве в соответствующие элементы записать номер первого дерева).

Теперь нам осталось только выбрать, какие ребра выбирать. Для начала создадим массив, в котором будут все возможные ребра, и отсортируем его по весу от наименьшего к наибольшему. Теперь мы будем перебирать все ребра по-порядку и выбирать, какие из них добавить, а какие пропустить. Если ребро соединяет две вершины разных деревьев, мы его берем в наш минимальный остов, попутно объединяя эти два дерева. Если ребро соединяет две вершины, которые уже принадлежат одному дереву, мы его пропускаем и переходим к следующему.

Формат входных данных:

Первая строка входного файла содержит два натуральных числа N и M - количество вершин и ребер графа соответственно $1 \le N \le 1000, 0 \le M \le 2000$. Следующие M строк содержат описание ребер по одному на строке. Ребро номер i описывается тремя натуральными числами bi, ei, wi — номера концов ребра и его вес соответственно $1 \le bi, ei \le N, 0 \le wi \le 100000$.

Гарантируется, что граф состоит из одной компоненты связности.

Формат выходных данных:

Выведите единственное целое число — вес минимального остовного дерева.

СПОСОБ ОЦЕНКИ РАБОТЫ:

Для генерации уникального условия и проверки результата используется следующий код на языке Python. Функция `generate` возвращает набор тестов и правильные ответы. Пример нескольких тестов представлен ниже:

```
def generate():
    num_tests = 10
    tests = [("4 4\n1 2 1\n2 3 2\n3 4 5\n4 1 4\n", "7\n"),
("5 10\n4 3 3046\n4 5 90110\n5 1 57786\n3 2 28280\n4 3 18010\n4 5 61367\n4 1
18811\n4 2 69898\n3 5 72518\n3 1 85838\n", "107923\n"),
("2 1\n1 2 10986\n", "10986\n"),
("3 2\n1 3 15891\n3 2 90498\n", "106389\n"),
("10 17\n8 7 83353\n7 10 74636\n7 4 47938\n10 3 4456\n8 1 90055\n3 6 22856\n10 5
84755\n3 9 77963\n5 2 58908\n8 4 44704\n8 3 36890\n8 5 28033\n8 2 30743\n7 10
83866\n7 4 95412\n7 3 48170\n7 6 38877\n","374577\n"),
#...
 ("6 9\n1 5 1\n5 6 3\n6 2 4\n6 3 5\n5 4 2\n1 4 3\n2 3 6\n1 2 5\n3 4 7\n",
"15\n")]
    return tests

def check(reply, clue):
```

```
    return int(reply) == int(clue)
```

РЕШЕНИЕ:

Алгоритм решения подробно описан в условии задачи. Задача требует внимательной реализации.

```
01. n, m = map(int, input().split())
02. gr = []
03. tree_num = [i for i in range(n)]
04. for i in range(m):
05.     b, e, w = map(int, input().split())
06.    gr.append((w, b-1, e-1))
07.
08. gr.sort()
09. ans = 0
10. for reb in gr:
11.     if tree_num[reb[1]] == tree_num[reb[2]]:
12.        continue
13.     ans += reb[0]
14.     min_num = min(tree_num[reb[1]], tree_num[reb[2]])
15.     max_num = max(tree_num[reb[1]], tree_num[reb[2]])
16.     for i in range(n):
17.         if tree_num[i] == max_num:
18.             tree_num[i] = min_num
19. print(str(ans))
```

`tree_num` — массив, в котором на i-ом месте номер дерева к которому сейчас относится i-ая вершина

§4 Заключительный этап: командная часть

Постановка задачи. Участникам командной части заключительного этапа было необходимо решить серию задач по анализу графа пользователей социальных сетей: предсказать возраст пользователей, не указавшего его в своем профиле; предсказать регион проживания пользователя; предположить, кто из других пользователей социальной сети является знакомым пользователя.

Участники должны были писать программы на языке Python. Продолжительность командной части заключительного этапа — 3 дня (всего 18 астрономических часов). Участники имели доступ к сети Интернет и могли пользоваться своими телефонами и ноутбуками.

Всего командам предлагалось 3 задачи — по одной на каждый день. Условие задачи становилось известно участникам утром соответствующего дня. Каждая задача оценивалась в баллах (подробнее см. далее).

Для каждой задачи было подготовлено два подграфа реальной социальной сети «Одноклассники»:

1. участникам представлялся специально подготовленный, очищенный и анонимизированный подграф;
2. проверка качества решения осуществлялась автоматически на полном графе, в котором присутствовали данные, вычищенные из первого графа.

Для каждой задачи участникам предоставлялось работающее базовое решение с низкой эффективностью, и участники стояли перед выбором: программировать с нуля свое собственное решение, которое сможет решить поставленную задачу качественнее, или дорабатывать предложенное решение. При этом можно было использовать базовое решение частично, например только модель данных или только распознаватель входных данных.

4.1. Описание исходных данных

Во всех задачах участникам предоставлялся граф пользователей (связи между пользователями) и файл с демографией (анонимизированные данные по каждому пользователю).

4.1.1. Граф пользователей

Граф сохранен в формате разреженной матрицы, где по каждой связи есть информация о её типе (родственник, друг и т.д.) в виде битовой маски. Каждая строка

матрицы соответствует друзьям одного пользователя и имеет формат:

```
ID_пользователя1 {(ID_друга1,маска1), (ID_друга2,маска2),…}
```

Матрица партиционирована по ID пользователя на 16 файлов, каждый из которых сжат стандартным протоколом сжатия GZip.

Пары в списке связей отсортированы по ID друга (по возрастанию). Пример записей из графа:

```
102416
{(5362439,0),(7321627,0),(7345280,0),(9939258,0),(9976393,0),(11260492,0),
(11924364,0),(16498676,0),(16513827,0),(21716731,0),(21826340,0),(23746537,0),
(23751503,0),(24412936,0),(24423533,0),(30287856,0),(32321147,0),(34243036,0),
(37592142,0),(39485706,0),(41505243,0),(42791620,0),(52012206,0),(52671472,0),
(54652307,0),(57293803,0),(59242794,0),(59252048,0),(62535397,0),(62563866,0),
(62567154,0),(64588902,0)}

102608
{(4167808,32784),(6019974,32),(6152844,16),(9570536,64),(10699806,33),
(13290514,0),(15064491,128),(16432948,512),(24473204,0),(24655822,0),
(25833075,256),(28000951,64),(30834507,2048),(34567533,16),(35766667,0),
(37385121,0),(40123805,512),(43134386,1024),(45439608,0),(45484652,0),
(47562525,0),(52378153,256),(52403136,512),(52493894,1024),(53483990,0),
(54048767,0),(54286279,2048),(57401158,0),(57956631,0),(58183281,0),
(61117236,32),(61898065,0),(61936634,0),(64512205,512),(65014849,0),
(65112662,0),(65259449,0)}
```

В маске связи могут быть установлены следующие биты:

1. Любовь
2. Супруг или Супруга
3. Родитель
4. Ребенок
5. Брат или сестра
6. Дядя или тетя
7. Родственники
8. Близкие друзья
9. Коллеги
10. Одноклассники
11. Племянник
12. Дед или бабушка
13. Внук или внучка
14. Однокурсник
15. Дружба в армии
16. Приемный родитель

17. Приемный ребенок
18. Крестный отец
19. Крестный сын
20. Совместная игра в спортивные игры

Помимо перечисленных битов в маске отношений может быть установлен, а может и не быть установлен нулевой бит. Этот бит играет чисто техническую роль и не имеет физического смысла. В итоге, например, отношение типа Ребенок может кодироваться числами 16 или 17.

Данные были подготовлены с использованием инструмента для хранения больших данных Apache Pig и содержат два соответствующих файла с заголовками, позволяющие участникам использовать этот инструмент и для предварительной обработки/фильтрации данных.

4.1.2. Демография пользователей

Данные о демографии предоставлены для того же миллиона пользователей, что и информация о социальных связях в формате списка атрибутов:

`userId create_date birth_date gender ID_country ID_Location loginRegion`

где:

- `userId` – идентификатор пользователя
- `create_date` – дата создания пользовательского аккаунта (количество миллисекунд от 01.01.1970)
- `birth_date` – дата рождения пользователя (количество дней от 01.01.1970, может быть отрицательным!)
- `gender` – пол пользователя (1 – мужчины, 2 – женщины)
- `ID_country` – идентификатор страны, указанной в профиле
- `ID_Location` – идентификатор региона/города, указанный в профиле.
- `loginRegion` – идентификатор региона, откуда чаще всего авторизуются в данной социальную сети пользователь (может отсутствовать!)

Пример данных:

```
44053078 1166032023073 3067 1 10414533690 2423601 99
12495764 1177932393270 1138
2 10405172143 188081
25646929 1165304175170 3756 2 10414533690 3953941 22
25646999 1160728984480 3884 2 10414533690 241372 120
12495833 1176909723643 3363 2 10414533690 2724941 11
```

Демография партиционирована по той же схеме, что и граф, но не сжата (передается в виде открытых текстов). Так же может быть обработана с помощью стандартного инструмента хранения больших данных Apache Pig или любого другого инструмента, поддерживающего CSV.

4.2 Формулировки задач

Задача 4.2.1 «Дата рождения»

Представленный для анализа фрагмент социального графа включает информацию о связях **100 тысяч пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа социальной сети со всеми связями и файл демографии, в котором указан данные по пользователям, включая возраст, однако возраст указан не для всех пользователей.

По пользователям которые присутствуют в графе, но не присутствуют в демографии необходимо установить значение их атрибута `birth_date` (дату рождения).

Данные записываются в файл в формате:

(<id_ползователя>\t(знак табуляции)<birth_date>)

Посчитанные результаты участников принимаются в файле формата txt и сравнивается с полными данными специально написанной программой, которая считает расхождение между данными участников и настоящими данными.

Чем меньше расхождение, тем выше оценивается результат команды согласно схеме, представленной далее в разделе «Методика оценки».

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. import random
 2. import math
 3.
 4. from GraphParser import graphParser
 5. def printall(res):
 6.     for i in range(0,len(res)):
 7.         print(res[i])
 8.         print("---------")
 9. cols = list()
10. cols.append(2)
11. (demog,fd) = graphParser.parseFolder("Task1\\Task1\\trainDemography",0,"",0,
cols)
12. c=1
13. for key, value in demog.items():
14.     print(str(key) + " val "+ str(value))
15.     c+=1
16.     if c==3:
17.         break
18.
```

```
#print(demog)
graph =  graphParser.parseFolderBySchema("Task1\\Task1\\graph",30,"")
#print(graph[0])
minBD = 9999999999999999
maxBD = 0
keyVal = 0 #'birth_date'
for key, value in demog.items():
    #print(key)
    bd = int(value[keyVal])
   # print( people)
    if bd>maxBD:
        maxBD=bd
    if bd<minBD:
        minBD=bd
print(minBD)
print(maxBD)
randCount=15
diffSum1 = 0
diffSum = [0]*randCount
for people in graph[0]:
    pId =  people['from']
    if pId not in demog.keys():
        print("err for "+str(pId))
        continue
    dateSum = 0
    totalLen=0
    maxBDp = 0
    minBDp = 9999999999999999
    #print(people['links'])
    for links in people['links']:

        pIdr = links['to']
        print(links['to'])


        if pIdr not in demog.keys():
            print("err for "+str(pIdr))
            continue
        totalLen+=1
        bd = int(demog[pIdr][keyVal])
        if bd>maxBDp:
            maxBDp=bd
        if bd<minBDp:
            minBDp=bd
        dateSum+=int(bd)

    if  (totalLen == 0):
        continue
    #    avg=propBirthDate
    #else:
    avg=(dateSum)/(totalLen)

    if (totalLen>=4):
        print("TOTAL Len big!"+str(totalLen))
        avg=(dateSum-maxBDp-minBDp)/(totalLen-2)
    else:
        print("total len small!"+str(totalLen))
        avg=(dateSum)/(totalLen)

    #avg=propBirthDate

```

```
80.     trueVal = int(demog[pId][keyVal])
81.     diffSum1+= abs(trueVal-avg)
82.     for ind in range(0,len(diffSum)):
83.         avg =  random.randrange(minBD,maxBD)
84.         diffSum[ind]+= abs(trueVal-avg)
85.
86. print(diffSum1)
87. for ind in range(0,randCount):
88.     print(diffSum[ind])
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
from random import randint
import math
dir_name = 'testDemography'
files = ['part-v004-o000-r-00000', 'part-v004-o000-r-00001', 'part-v004-o000-r-
00002', 'part-v004-o000-r-00003', 'part-v004-o000-r-00004', 'part-v004-o000-r-
00005', 'part-v004-o000-r-00006', 'part-v004-o000-r-00007', 'part-v004-o000-r-
00008', 'part-v004-o000-r-00009', 'part-v004-o000-r-00010', 'part-v004-o000-r-
00011', 'part-v004-o000-r-00012', 'part-v004-o000-r-00013', 'part-v004-o000-r-
00014', 'part-v004-o000-r-00015']
files = [dir_name + '/' + i for i in files]
df = pd.DataFrame()
frames = []
for file_name in files:
    d = pd.read_csv(file_name, sep='\t', names=['id', 'date', 'num', 'bla1',
'bla2', 'bla3', 'bla4', 'bla5'])
    del d['date']
    del d['bla1']
    del d['bla2']
    del d['bla3']
    del d['bla4']
    del d['bla5']
    frames.append(d)
test_data = pd.concat(frames, ignore_index=True)
answers = pd.read_csv('results.txt', sep='\t', names=['id', 'num'])

def compare(res, test):
    to_sum = []
    for i, row in res.iterrows():
        vals = test[test['id'] == row['id']]['num'].values
        if(len(vals)):
            stds = []
            for v in vals:
                if ((not math.isnan(v)) and not math.isnan(row['num'])):
                    stds.append(math.pow(v - row['num'], 2))
            if(len(stds)):
                print('stds', stds)
                to_sum.append(min(stds))
    return sum(to_sum)

s = compare(answers, test_data)
print(s)
```

РЕШЕНИЕ:

В первую очередь надо определить что поставленная задача является регрессионой задачей, после чего можно провести исследование особенностей данной задачи, найти корреляции между свойствами пользователей и провести исследование на поиск лучшей модели для работа с этими данными.

Также важные результаты могут показать гипотезы о возрасте друзей пользователя.

Задача 4.2.2 «Регион»

Представленный для анализа фрагмент социального графа включает информацию о связях **100 тысяч пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа социальной сети со всеми связями и файл демографии, в котором указан данные по пользователям, включая регион, однако регион указан не для всех пользователей.

По пользователям которые присутствуют в графе, но не присутствуют в демографии необходимо установить их аттрибут `ID_Location` (регион).

Ответ записывается в текстовый файл в формате:

`(<id_ползователя>\t(знак табуляции)<ID_Location>)`

Посчитанные результаты участников принимаются в файле формата txt и сравнивается с полными данными специально написанной программой, которая считает расхождение между данными участников и настоящими данными.

Чем меньше расхождение, тем выше оценивается результат команды согласно схеме, представленной в разделе «Методика оценки».

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. import math
 2. import sys
 3.
 4. def bl(graph, locs, fd=False):
 5.     res = list()
 6.     count = int(0)
 7.
 8.     for pId,conns  in graph.items():
 9.         count+=1
10.         if count%1000 == 0:
11.             print(count)
12.         dateSum = 0
13.         totalLen=0
14.         locIds=dict();
15.         print(pId)
16.         try:
17.             if locs[pId] != None:
18.                 continue
19.         except:
20.             pass
21.         if type(conns) == int:
22.             conns=[conns]
```

```
23.         for links in conns:
24.             totalLen+=1
25.             try:
26.                 frLoc=locs[links]
27.             except:
28.                 continue
29.             try:
30.                 locIds[frLoc]+=1 #int(demog[links])
31.             except:
32.                 locIds[frLoc]=1
33.
34.         resId=0
35.         popId=0
36.         for locId, total in locIds.items():
37.             if total>popId:
38.                 popId=total
39.                 resId=locId
40.
41.         res.append([pId,resId])
42.         if (fd):
43.             fd.write(str(pId)+'\t'+str(resId)+'\n')
44.     return res
45.
46. from GraphParser import graphParser
47.
48. pass
49. cols = list()
50. cols.append("userId")
51. cols.append("ID_Location")
52. (locs,fd) =
graphParser.parseFolderBySchema("Task2\\Task2\\trainDemography",0,"","userId",
cols, True)
53. cols = list()
54. cols.append("from")
55. cols.append("to")
56. cols.append("links")
57. (graph, fd) =
graphParser.parseFolderBySchema("Task2\\Task2\\graph",0,"","from",cols,True)
58. print("data loaded")
59. fdres=open("results.txt",'w')
60. bl(graph,locs, fdres)
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
import math
import ast

test_df = pd.read_csv('task2/test.tsv', sep='\t', names=['id', 'groups'])
results_df = pd.read_csv('task2/results.tsv', sep='\t', names=['id', 'groups'])

def compare(results, test):
    #Iterate over all submitters results
    score = 0
    not_found_penalty = -5
    false_found_penalty = -5
```

```
    found_reward = 10
    for i, row in test.iterrows():
        test_groups = ast.literal_eval(row['groups'])
        #No such user
        if(not (any(results.id == row['id']))):
            score = score + (len(test_groups) * not_found_penalty)
            continue
        #Get fit
        result_groups = ast.literal_eval(results[results['id'] == row['id']]
['groups'].values[0])
        for tg in test_groups:
            if (tg in result_groups):
                score = score + found_reward
                result_groups.remove(tg)
                test_groups.remove(tg)
        #Get penalty
        score = score + (len(result_groups) * not_found_penalty)#not found
        score = score + (len(test_groups) * false_found_penalty)#false found
    return score
compare(results_df, test_df)
```

РЕШЕНИЕ:

В первую очередь надо определить, что поставленная задача является задачей на классификацию, после чего можно провести исследование особенностей задачи, найти корреляции между свойствами пользователей и провести исследование на поиск лучшей модели для работа с этими данными.

Так же важные результаты могут показать гипотезы о месте атрибута `location_id` друзей пользователя, особенно тех, которые учились с ним в одной школе.

Вторая задача похожа на первую, хоть и принадлежит к другому классу задач машинного обучения, таким образом участники могли использовать свои наработки первой задачи для решения второй.

Задача 4.2.3 «Поиск связей»

Представленный для анализа фрагмент социального графа включает информацию о связях **1 миллиона пользователей**, попавших в двухшаговую окрестность сотни случайно выбранных пользователей. Участникам предоставляются файлы графа и демографии по пользователям. Часть связей в предоставленном социальном графе скрыта и задачей участников является максимально полно и точно раскрыть их.

Сокрытие связей коснулось только пользователей из исходного миллиона, остаток от деления аттрибут ID которых на 11 равен 7 (id % 11 == 7), сокрытию подверглось порядка 10% связей для каждого из этих пользователей. Были скрыты только ведущие в исходный миллион связи.

В прогнозе достаточно восстановить наличие связи, ее тип не важен. Результаты прогноза нужно представить в формате CSV файла вида:

```
ID_пользователя1 ID_кандидата1.1 ID_кандидата1.2 ID_кандидата1.3
ID_пользователя2 ID_кандидата2.1 ID_кандидата2.2
```

Записи в файле отсортированы по ID пользователя (по возрастанию), а затем по предсказанной релевантности кандидатов (по убыванию, саму релевантность при этом в файл писать не надо). Пример результатов:

```
5111 178542 78754
18807 982346 1346 57243
```

Результаты участников оцениваются с помощью метрики Нормализованной скидочной совокупной выгоды (Normalized Discounted Cumulative Gain, NDCG), используемой в индустрии для оценки точности работы алгоритма для этой и аналогичных ей задач. Метрика рассчитывается отдельно по каждому из пользователей, для которых есть скрытые связи, а затем усредняться. Записи в файле результата, не имеющие отношения к пользователям со скрытыми связями, при оценке результата учитываться не будут. Если по какому-то пользователю не будет предложено ни одного кандидата, то значение метрики для него будет считаться за 0.

Предоставляемое участникам базовое решение (дает 1 балл):

```
 1. # Для чтения/записи csv файлов
 2. import csv
 3. # Для работы с архивами
 4. import gzip
 5. # Для работы с файловой системой
 6. import os
 7. # Эффективные массивы простых типов
 8. import numpy
 9. # Работа с матрицами (подсчет общих друзей реализован как умножение матрицы
графа самое на себя)
10. import scipy
11. from scipy.sparse import coo_matrix, csr_matrix
12. # Пути к данным
13. dataPath = "./"
14. graphPath = os.path.join(dataPath, "trainGraph")
15. predictionPath = os.path.join(dataPath,"prediction.gz")
16.
17. # Основные параметры графа
18. numUsers = 107474
19. numLinks = int(72384968 / 2)
20. maxUserId = 9418031
21. # В этих массивах мы будем собирать данные. Инициализируем их заранее нужным
размером чтобы
22. # небыло лишнего копирования
23. form = numpy.zeros( (numLinks), dtype=numpy.int32 )
24. to = numpy.zeros( (numLinks), dtype=numpy.int32 )
25. data = numpy.ones( (numLinks), dtype=numpy.int32 )
26.
27. # Здесь храним позицию, на которую надо записать новую связь
28. current = 0
29.
30. # Итерируемся по файлам в папке
31. for file in [f for f in os.listdir(graphPath) if f.startswith("part")]:
```

```
32.     csvinput = gzip.open(os.path.join(graphPath, file), mode='rt')
33.     csv_reader = csv.reader(csvinput, delimiter='\t')
34.     # А теперь по строкам в файле
35.     for line in csv_reader:
36.         user = int(line[0])
37.         # Разбираем идшки и маски друзей
38.         for friendship in line[1].replace("{(", "").replace(")}",
"").split("),("):
39.             parts=friendship.split(",")
40.             # Записываем связь в массивы и двигаем указатель
41.             form[current] = user
42.             to[current] = int(parts[0])
43.             current += 1
44.
45.     # Не забываем закрыть файл
46.     csvinput.close()
47. # Создаем из массивов матрицу. Изначальна матрица хранится в виде списка
[i,j,v], но для эффективной
48. # дальнейшей работы нам надо преобразовать в вид [i->[j,v]]
49. fullMatrix = coo_matrix(
50.     (data, (form, to)),
51.     shape=(numLinks + 1, numLinks + 1)).tocsr()
52.
53. # Массивы больше не нужны, удаляем их из памяти
54. del form
55. del to
56. del data
57. # Считаем транспонированную матрицу (колонки и ряды поменяны местами) и её
тоже приводим в вид [i->[j,v]]
58. reversedMatrix = scipy.transpose(fullMatrix).tocsr()
59. # Поскольку прогноз нам нужно строить только для части пользователей,
остальных из
60. # исходной матрицы уберем (забьем нулями)
61. for i in range(maxUserId + 1):
62.     if i % 11 != 7:
63.         ptr = fullMatrix.indptr[i]
64.         ptr_next = fullMatrix.indptr[i+1]
65.         if ptr != ptr_next:
66.             fullMatrix.data[ptr:ptr_next].fill(0)
67.
68. # Чтобы нули не мешались при умножении, вычистим их и подуменьшим матрицу
69. fullMatrix.eliminate_zeros()
70. # Здесь и происходит основная магия - через умножение матриц получаем
счетчики общих друзей,
71. # По которым сделаем прогноз
72. commonFriends = fullMatrix.dot(reversedMatrix)
73. # Теперь осталось его записать в файл. Открываем врайтеры
74. f = open('prediction.csv', 'w')
75. writer = csv.writer(f, delimiter='\t')
76.
77. for i in range(maxUserId + 1):
78.     # Два указателя дают нам границы в которых лежат данные для этого i в
матрице
79.     ptr = commonFriends.indptr[i]
80.     ptr_next = commonFriends.indptr[i+1]
81.     # Если они не равны, значит данные есть и можно экспортировать
82.     if ptr != ptr_next:
83.         # Достаем счетчики общих друзей и создаем порядок на них от большего
к меньшему
84.         counts = commonFriends.data[ptr:ptr_next]
85.         order = numpy.argsort(-counts)
```

```
86.
87.         # Не забываем что из прогноза надо убрать себя и своих известных
друзей
88.         mineFriends =
set(fullMatrix.indices[fullMatrix.indptr[i]:fullMatrix.indptr[i+1]])
89.         mineFriends.add(i)
90.
91.         # Достаем идшки друзей, сортируем, фильтруем, обрезаем и пишем
92.         ids = commonFriends.indices[ptr:ptr_next]
93.         writer.writerow([i] + list(filter(lambda x: x not in mineFriends,
ids[order]))[:42])
94.
95. # Не забываем закрыть файл
96. f.close()
```

СПОСОБ ОЦЕНКИ РЕЗУЛЬТАТА:

Для получения количественной оценки правильности результата использовалась следующая программа-компаратор на языке Python:

```
import pandas as pd
import numpy as np
dir_name = 'testDemography'
files = ['part-v004-o000-r-00000', 'part-v004-o000-r-00001', 'part-v004-o000-r-
00002', 'part-v004-o000-r-00003', 'part-v004-o000-r-00004', 'part-v004-o000-r-
00005', 'part-v004-o000-r-00006', 'part-v004-o000-r-00007', 'part-v004-o000-r-
00008', 'part-v004-o000-r-00009', 'part-v004-o000-r-00010', 'part-v004-o000-r-
00011', 'part-v004-o000-r-00012', 'part-v004-o000-r-00013', 'part-v004-o000-r-
00014', 'part-v004-o000-r-00015']
files = [dir_name + '/' + i for i in files]
df = pd.DataFrame()
frames = []
for file_name in files:
    print(file_name)
    d = pd.read_csv(file_name, sep='\t', names=['id', 'date', 'num', 'bla1',
'bla2', 'bla3', 'bla4', 'bla5'])
    del d['date']
    del d['bla1']
    del d['bla2']
    del d['bla3']
    del d['bla4']
    del d['bla5']
    frames.append(d)
result = pd.concat(frames, ignore_index=True)
result
def dcg_at_k(r, k, method=0):
    """Score is discounted cumulative gain (dcg)
    Relevance is positive real values.  Can use binary
    as the previous methods.
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]
    Returns:
        Discounted cumulative gain
    """
    r = np.asfarray(r)[:k]
    if r.size:
        if method == 0:
            return r[0] + np.sum(r[1:] / np.log2(np.arange(2, r.size + 1)))
```

```
        elif method == 1:
            return np.sum(r / np.log2(np.arange(2, r.size + 2)))
        else:
            raise ValueError('method must be 0 or 1.')
    return 0.

def ndcg_at_k(r, k, method=0):
    """Score is normalized discounted cumulative gain (ndcg)
    Relevance is positive real values.  Can use binary
    as the previous methods.
    Args:
        r: Relevance scores (list or numpy) in rank order
            (first element is the first item)
        k: Number of results to consider
        method: If 0 then weights are [1.0, 1.0, 0.6309, 0.5, 0.4307, ...]
                If 1 then weights are [1.0, 0.6309, 0.5, 0.4307, ...]
    Returns:
        Normalized discounted cumulative gain
    """
    dcg_max = dcg_at_k(sorted(r, reverse=True), k, method)
    if not dcg_max:
        return 0.
    return dcg_at_k(r, k, method) / dcg_max

r = [3, 2, 3, 0, 0, 1, 2, 2, 3, 0]
ndcg_at_k(r, 7)
len(test[test['id'] == 15102006]['num'].values)

pd.Series([123, 0], index=['id', 'num'])
any(result.id == 115368359)
t.to_csv('results.csv',  sep='\t', index = False, header = False)
def compare(res, test):
    #Iterate over all submitters results
    to_sum = []
    for i, row in test.iterrows():
#       If there is no such id crete with zero
        if(not (any(res.id == row['id']))):
            res.append(pd.Series([row['id'], 0], index=['id', 'num']))

    for i, row in res.iterrows():
        vals = test[test['id'] == row['id']]['num'].values
        if(len(vals)):
            stds = []
            for v in vals:
                if (not math.isnan(v)):
                    stds.append(math.pow(v - row['num'], 2))
            if(len(stds)):
                to_sum.append(min(stds))

    return sum(to_sum)
t1 = result.copy()
# t1[t1['id'] == 11536835]['num'] = t1[t1['id'] == 11536835]['num'] + 1
t1.loc[1,'num'] = t1.loc[1,'num'] + 1
t1.loc[2,'num'] = t1.loc[2,'num'] + 2
t1.loc[3,'num'] = t1.loc[3,'num'] - 5
# t1[t1['id'] == 11536835]['num']
s = compare(t1, result)
s
```

РЕШЕНИЕ:

В качестве примера решения задачи, точность прогноза которого надо превзойти,

используется логистическая регрессия, натренированная на трех признаках:

1. количестве общих друзей двух пользователей,
2. разнице в возрасте и
3. факте совпадения или различия полов.

Помимо улучшения алгоритма, необходимо так же учитывать вычислительную сложность, которая ставит требование не только качественно повысить эффективность базового решения, но и успеть рассчитать результаты для всех пользователей.

Поэтому для эффективного решения необходимо выделить только те особенности для включения в модель, которые имеют достаточно высокую корреляцию с дружбой пользователей.

Это ставит перед участниками потребность в постановке гипотез о том, какие факторы имеют высокую корреляцию, а какие — низкую, и проверку гипотез на предоставленных данных.

4.3. Методика расчета баллов

По каждой из задач была написана программа-компаратор (указанная выше), которая сравнивает решение участников с полными данными, которые были не раскрыты и имеются только у жюри Олимпиады.

Чем сильнее расхождение между результатами участников и полными данными, тем ниже результат участников в баллах. Таким образом, проверка работ участников и размер начисляемых баллов были полностью автоматизированы.

Максимальный балл заключительного этапа по командной части мог составить 56 баллов. Максимальный балл по задачам распределялся следующим образом:

- Задача 1 — 10 баллов;
- Задача 2 — 16 баллов;
- Задача 3 — 30 баллов.

Для расчета количества баллов, начисляемых участникам, использовалась логистическая регрессия с округлением до целочисленного количества баллов, в которой максимальному баллу соответствовал уровень результата, в 4 раза превосходящий базовое решение, а минимальный балл (1 балл) начислялся за уровень результата, равного предложенному базовому решению.

0 баллов начислялось в случае, если результат не был показан (команда не сдала валидное решение в срок) или если точность результата была ниже базового решения.

§5 Критерий определения победителей и призеров заключительного этапа

В заключительном этапе олимпиады баллы участника складываются из двух частей: он получает баллы за индивидуальное решение задач по предметам (математика, информатика) и за командное решение практической задачи. Итоговая оценка участника олимпиады получается по следующей формуле:

$S = S_1 + S_2$,

где S_1 — количество баллов, набранное в рамках индивидуальной части заключительного этапа (максимум — 24 балла); S_2 — количество баллов, набранное в рамках командной части заключительного этапа (максимум — 56 баллов).

Критерий определения победителей и призеров:

	Призеры	**Победители**
Ученики 9 класса	От 40 до 43 баллов	44 баллов и выше
Ученики 10 и 11 класса	От 31 до 43 баллов	44 баллов и выше