Всероссийская олимпиада школьников по информатике 2012

Региональный этап

Разбор задач

Автор разбора — Владимир Михайлович Гуровиц

Задача 1. «Цапли»

Обозначим через k возможное количество цапель. Если Петя увидел a ног, то цапель не больше, чем a (поскольку каждая стоит хотя бы на одной ноге). С другой стороны, их не меньше, чем $a / 2$ (иначе ног у них было бы меньше, чем $(a / 2) * 2 = a$). Таким образом,

$$a / 2 \le k \le a.$$

Аналогично,

$$b / 2 \le k \le b.$$

Поскольку по условию задачи хотя бы одно решение данной системы неравенств существует, получаем:

$$\max(a, b) / 2 \le k \le \min(a, b).$$

Если max(a, b) / 2 – нецелое число, его нужно округлить вверх. Это можно сделать например так: (max(a, b) + 1) // 2 (здесь // обозначает целочисленное деление).

Приведем фрагмент программы на языке C++:

```
cin >> a >> b;
cout << (max(a, b) + 1) / 2 << " " << min(a, b);
```

Задача 2. «Круглый стол»

Попробуем сначала рассадить вокруг стола какое-нибудь количество мальчиков и девочек, так чтобы все принимали активное участие в обсуждении.

Посадим за стол одного мальчика. Слева и справа от него нужно посадить девочек:

…GBG…

Теперь рядом с каждой девочкой нужно посадить еще по девочке:

…GGBGG…,

а рядом с новыми девочками – еще по одному мальчику:

…BGGBGGB…

Можно заметить закономерность: если рассаживать детей по схеме

B – GG – B – GG – B – GG - B - … - GG,

то в итоге *все* мальчики и девочки будут принимать участие в обсуждении. В этом случае количество девочек окажется вдвое больше количества мальчиков.

Покажем теперь, как будет выглядеть одна из возможных оптимальных рассадок в остальных случаях.

1. Количество девочек больше, чем удвоенное количество мальчиков ("лишние" девочки подчеркнуты):

   BGGBGGBGG … BGGBGGGGG … GG

2. Количество девочек четно и меньше, чем удвоенное количество мальчиков:

BGGBGGBGG … BGGBGGBBB … BB

3. Количество девочек нечетно и меньше, чем удвоенное количество мальчиков:

BGGBGGBGG … BGGBGGBBB … BG

Докажем, что в этих примерах количество участвующих в обсуждении детей максимально возможное.

1. $n > 2m$. Поскольку каждый мальчик может стимулировать к активному участию в обсуждении не больше двух девочек, всего удастся привлечь к активному обсуждению не более $2n$ девочек. В приведенном для этого случая примере в обсуждении будут участвовать все n мальчиков и ровно $2n$ девочек.
2. $n = 2k < 2m$. В приведенном примере в обсуждении участвуют *все* $2k$ девочек и $k – 1$ мальчиков. Пусть мы можем пересадить школьников так, чтобы $k + x$

   Рассмотрим пары рядом сидящих детей разного пола. Каждая активная девочка входит ровно в одну такую пару, каждая неактивная девочка может входить в две такие пары. Пусть $k + x$ мальчиков активны ($x > 0$), тогда пар мальчик-девочка (или девочка-мальчик) должно быть не менее $2(k + x) = 2k + 2x$, откуда неактивных девочек должно быть хотя бы $2x$. В этом случае общее количество активных детей равно $(k + x) + (2k – 2x) = 3k – x$. Это число больше $3k – 1$ лишь при $x = 0$. Таким образом, если и можно рассадить детей лучше, то в этом случае активными будут ровно k мальчиков и все $2k$ девочек. Докажем, что и этого сделать нельзя.

   Действительно, чтобы все девочки были активны, они должны сидеть парами (рядом с каждой девочкой должна быть девочка), с другой стороны, чтобы k мальчиков были активны, они должны сидеть между двумя девочками каждый, то есть рядом с девочками должны сидеть только активные мальчики. Таким образом, каждый активный школьник должен сидеть рядом с двумя другими активными школьниками, то есть никакой активный школьник не сидит рядом с неактивным, что невозможно, если все школьники сидят за круглым столом.

   3. $n = 2k + 1 < 2m$. Доказательство этого случая аналогично предыдущему. В приведенном примере активны k мальчиков и 2k девочек: всего 3k школьников. Пусть при некоторой рассадка $k + x$ мальчиков окажутся активными ($x > 0$), тогда пар мальчик-девочка (или девочка-мальчик) должно быть не менее $2(k + x) = 2k + 2x$, откуда неактивных девочек должно быть хотя бы $2x$, но поскольку общее количество девочек нечетно, а количество активных девочек должно быть четно, то количество неактивных девочек нечетно, а потому не меньше $2x + 1$. В этом случае общее количество активных детей равно $(k + x) + (2k + 1 – (2x + 1)) = 3k – x$. Это число не больше $3k$, следовательно, приведенный пример является оптимальным.

   Приведем фрагмент программы на языке python3:

```
if 2 * m == n:
    print('BGG' * m)
elif 2 * m < n:
    print('BGG' * m  + 'G' * (n - 2 * m))
else:
    if n % 2 == 1:
         print('BGG' * (n//2) + 'B' * (m – n // 2) + 'G')
    else:
         print('BGG' * (n // 2) + 'B' * (m - n // 2))
```

Задача 3. «Поврежденный XML»

Поскольку ограничения на размер входной строки в этой задаче небольшие, ее можно решать полным перебором. Будем пытаться исправлять каждый символ по порядку на все допустимые (латинские буквы, символы '<', '>', '/'), до тех пор пока не получится корректная xml-строка.

Приведем пример кода на языке python3:

```
s = input()
alph = 'qwertyuiopasdfghjklzxcvbnm<>/'
for i in range(n):
    for c in alph:
        test = s[:i] + c + s[i + 1:]
        if correct_xml(test):
            print(test)
            exit()
```

Осталась чисто техническая часть решения: написать функцию correct_xml, которая проверяет, является ли данная строка корректным xml-фрагментом. Разобьем эту задачу на несколько шагов.

1) Убедимся, что строка начинается с символа '<', а заканчивается символом '>' и удалим начальный и конечный символы.
2) Разрежем строку по парам символов '><' (удаляя при этом эти символы). Получим список строк-тегов (открывающих или закрывающих).
3) Проверим, что в этих тегах нет символов '<' или '>', а также нет символов '/' не на первой позиции.
4) Проверим, что эти теги образуют "правильную скобочную последовательность".

Опишем подробнее выполнение последнего шага. Заведем стек тегов (строк), изначально пустой. Будем идти по тегам слева направо. Рассмотрим два случая:

I. *Очередной тег – открывающий*. В этом случае добавим новый тег в конец стека.

II. *Очередной тег* – закрывающий.

а) Стек пуст. В этом случае мы нашли закрывающий тег, которому не соответствует открывающий тег, следовательно, строка не является корректной xml-строкой.

б) Стек не пуст. Извлечем из стека последний элемент. Если этот тег не соответствует нашему закрывающему тегу, то xml-строка некорректная.

Если мы прошли весь список тегов, и ни на каком шаге не выяснили, что строка некорректна, нам осталось лишь проверить, что в конце стек оказался пуст (то есть что у нас нет "лишних" открывающих тегов).

Приведем пример реализации такой функции на языке python3:

```
def check(test):
    if test[0] != '<' or test[-1] != '>':
        return False
    testlist = test[1:-1].split('><')
    for s in testlist:
        if '<' in s or '>' in s or '/' in s[1:]:
            return False
```

```
    q = deque()
    for s in testlist:
        if s[0] != '/':
            q.append(s)
        elif not q or q.pop() != s[1:]:
            return False
    if q:
        return False
    return True
```

Задача 4. «Игра с числами»

Обозначим через *first* число, выбранное на первом ходу. Тогда каждое очередное число нужно выбирать так, чтобы НОД разностей всех выбранных чисел с первым числом был строго больше 1. Заметим, что если d – это НОД разностей выбранных чисел с первым числом, то оставшиеся числа делятся на две группы: числа, при выборе которых на последующих ходах НОД не изменится, и числа, которые при их выборе уменьшат НОД. При этом с одной стороны *для дальнейшей игры нам не важно, какие именно числа находятся в первой группе* (а важно лишь их количество), а с другой стороны, определить, принадлежит ли некоторое число a второй группе, можно, проверив, делится ли $|a - first|$ на d. Таким образом, позицию в игре (начавшейся выбором числа first) можно задавать двумя параметрами: количеством сделанных ходов *num* и НОД d разностей всех выбранных чисел с первым числом.

Для каждого возможного первого хода будем решать задачу отдельно. Поскольку количество теоретически возможных значений НОД может быть довольно велико, будем решать задачу рекурсией с запоминанием ("ленивым" динамическим программированием).

Рассмотрим позицию (num, d).

1) Если $num = n$, то позиция проигрышная (все числа уже выбраны; ход сделать нельзя).
2) Подсчитаем, количество чисел *count* в изначальном наборе чисел, разности которых с числом *first* делятся на d:

   а) Рассмотрим сначала числа, выбор которых уменьшает d, то есть такие числа x, для которого $|x - first|$ не делится на d, и при этом НОД(d, $|x - first|$) должен быть больше 1 (в противном случае выбрать число x уже нельзя). Позиция (num, d) выигрышная, если ($num + 1$, НОД(d, $|x - first|$) – проигрышная.

   б) если $count > num$, то мы можем выбрать число, делящееся на d, получив при этом позицию ($num + 1$, d). Если она проигрышная, то (num, d) – выигрышная.
3) В противном случае позиция (num, d) – проигрышная.

Осталось отметить один технический момент: если $num = 1$, то выбрано пока только число *first*. В этом случае все рассуждения останутся верными, если положить $d = 0$.

Приведем программу на языке python3:

```
#python3
from fractions import gcd
fin = open("game.in")
fout = open("game.out", "w")
n = int(fin.readline()
a = list(map(int, fin.readline().split()))
def solve(num, d):
    if (num, d) in res:
```

```
            return res[(num, d)]
        if num == n:
            res[(num, d)] = False
            return False
        count = 0
        chance = False
        for el in b:
            if el != 0:
                if d > 0 and el % d == 0:
                    count += 1
                else:
                    newgcd = gcd(d, el)
                    if gcd(d,el) > 1 and not solve(num + 1, newgcd):
                        res[(num, d)] = True
                        return True
        else:
            if count >= num:
                if not solve(num + 1, d):
                    res[(num, d)] = True
                    return True
        res[(num, d)] = False
        return False

answer = []
for first in a:
    res = dict()
    b = [abs(a[i] - first) for i in range(n)]
    if not solve(1, 0):
        answer.append(first)
print(len(answer), file = fout)
print(" ".join(map(str, answer)), file = fout)
```

Задача 5. «Кондиционер»

Приведём два возможных подхода к решению задачи.

Первое решение. Заметим, что результат зависит только от режима работы кондиционера и взаимного расположения чисел t_{room} и t_{cond}. Рассмотрим все комбинации и запишем результат в виде таблицы:

	freeze	**heat**	**auto**	**fan**
$t_{room} < t_{cond}$	t_{room}	t_{cond}	t_{cond}	t_{room}
$t_{room} = t_{cond}$	t_{room}	t_{cond}	t_{cond}	t_{room}
$t_{room} > t_{cond}$	t_{cond}	t_{room}	t_{cond}	t_{room}

Приведем фрагмент решения задачи на языке C++

```
cin >> troom >> tcond;
```

```
cin >> s;
if (s=="freeze") || (s=="heat" && troom>tcond) || (s=="freeze")
{
    cout << troom;
} else {
    cout <<tcond;
}
```

Второе решение.

Заметим, что при каждом режиме работы кондиционер реализует некоторую функцию, которая вычисляет результат по двум аргументам troom и tcond.

В режиме "freeze" кондиционер реализует функцию min(x, y), в режим "heat" – функцию max(x, y), в режиме "auto" – функцию $f(x, y) = y$ (возвращает второй аргумент), а в режиме "fan" – функцию $g(x, y) = x$ (возвращает первый аргумент).

Приведем фрагмент программы на языке C++, реализующий данную идею:

```
if (s == "freeze") cout << ((troom > tcond) ? tcond : troom);
if (s == "heat") cout << ((troom < tcond) ? tcond : troom);
if (s == "fan") cout << troom;
if (s == "auto") cout << tcond;
```

Задача 6. «Праздничный ужин»

Пусть некоторому программисту предожили p вариантов ужина, а следующему – q.

$$p = a_1 \times a_2 \times \ldots \times a_i \times \ldots \times a_k, \quad q = a_1 \times a_2 \times \ldots \times (a_i - 1) \times \ldots \times a_k, \quad p / q = a_i / (a_i - 1),$$

откуда

$$a_i = p / (p - q).$$

Таким образом, когда программисту предложили p вариантов ужина, одно из блюд предлагалось в $p / (p - q)$ вариантах.

Постепенно анализируя входные данные, мы будем получать новую информацию о количестве блюд того или иного вида.

Заведем два массива: в одном (start) будем хранить изначальное количество каждого из видов блюд (сначала массив будет пустой, а по мере получения новой информации мы будем его расширять). Во втором массиве (current) в элементе с тем же номером мы будем хранить текущее количество вариантов того же вида блюда (на самом деле решений может быть несколько: мы будем хранить количества, соответствующие одному из них).

Пусть в некоторый момент мы обнаружили, что было a_i вариантов некоторого блюда. Возможны две ситуации:

1) в массиве current уже есть число a_i. Тогда будем считать, что это то же самое блюдо, и уменьшим этот элемент массива current на 1.
2) в массиве current нет числа a_i. Это означает, что мы получили информацию о блюде, про которое ранее не было известно ничего. Добавим в массив start число a_i, а в массив current – число $a_i - 1$.

(Заметим, что если в пункте 1 мы бы создали новое блюдо, то в итоге могли бы получить количество блюд, большее, чем указано в условии.)

После обработки всех входных данных мы получим массив start, длина которого меньше или равна количеству блюд k. Если она равна k, то задача решена. Если длина меньше k, посмотрим на количество вариантов r, которое было предложено на выбор последнему программисту. В массиве current уже есть информация о количестве блюд

некоторых видов к этому моменту: разделим r на произведение элементов массива current и будем считать, что у нас есть еще один вид блюд с таким изначальным количеством.

Если же и теперь количество видов блюд меньше, чем требуется, добавим оставшиеся виды блюд с изначальным количеством, равным 1.

Например, если последнему программисту остался выбор из 4 комбинаций блюд, всего блюд было 5, и в массиве start лежат числа 2 и 3, будем считать, что изначально был выбор из 2-х первых блюд, 3-х вторых блюд, 4-х третьих блюд, 1-го четвертого и 1-го пятого.

Приведем пример программы на языке python3:

```
types = 0
start = []
current = []
for i in range(1, n):
    y = a[i - 1] // (a[i - 1] - a[i])
    for j in range(types):
        if current[j] == y:
            current[j] = y - 1
            break
    else:
        types += 1
        start.append(y)
        current.append(y - 1)
last = a[n - 1]
for value in cur:
    last //= value
print(" ".join(map(str, start)), end = " ", file = fout)
if len(start) < k:
    print(last, '1 ' * (k - 1 - types), file = fout)
```

Задача 7. «Космический кегельбан »

Переформулируем задачу на геометрическом языке. Дано несколько кругов радиуса r (кегли) и полоса ширины $2q$ (траектория шара). Требуется подсчитать, сколько окружностей имеют хотя бы одну общую точку с полосой.

Заменим окружности на точки, а полосу ширины $2q$ на полосу ширины $2(q + r)$. Тогда задача сведется к эквивалентной: сколько точек (центров кеглей) лежат в расширенной полосе, то есть находятся на расстоянии не более $(q + r)$ от прямой, по которой движется центр шара. Заметим, что поскольку по условию задачи каждая точка шара лежит изначально ниже любой точки каждой кегли, то можно рассматривать не часть полосы выше шара, а всю полосу: на ответ это не повлияет.

Запишем неравенством условие принадлежности точки полосе. Пусть A – начальное положение центра шара, вектор AB имеет координаты (v_x, v_y), P – центр некоторой кегли.

Тогда расстояние от точки P до прямой AB равно $|[AP, AB]| / |AB|$ (квадратными скобками обозначено псевдоскалярное произведение векторов), а условие принадлежности точки полосе запишется так:

$$|[AP, AB]| / |AB| \le q + r.$$

Приведем теперь три решения исходной задачи, отличающиеся по времени работы.

Первое решение (асимптотическая сложность порядка n^2).

Проверим отдельно принадлежность центра $P(x_p, y_p)$ каждой кегли нашей полосе. Для этого запишем полученное выше неравенство в координатах. Пусть точки A и B имеют координаты (x, y) и $(x + v_x, y + v_y)$ соответственно (B – точка на прямой, по которой движется центр шара). Тогда неравенство примет вид:

$$|(x_P - x)\, v_y - (y_P - y)\, v_x)| \,/\, \sqrt{v_x^2 + v_y^2} \le q + r. \quad (1)$$

Домножим обе части части на знаменатель и возведем в квадрат:

$$((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2.$$

Подставляя последовательно в неравенство центры всех кеглей, находим ответ.

Последнее условие требует вычислений только в целых числах, что с одной стороны избавляет нас от всех возможных проблем, связанных с точностью вычислений, но с другой может создать проблемы с переполнением: правая часть неравенства в ограничениях задачи может достигать $((10^{12})^2 + (10^{12})^2) * (10^9)^2 = 10^{32}$, что превосходит границу чисел, которые можно хранить даже в 64-битных целых типах.

Приведем фрагмент программы на языке python3 (где длина целого числа ограничена лишь объемом доступной памяти; на других языках можно воспользоваться длинной арифметикой, либо написать решение с вещественными числами):

```
result = 0
for yp in range(n): # yp = 0, 1, … , n - 1
    res_line = 0
    for xp in range(-i, i + 1, 2): # xp = -i, -i + 2, …, i – 2, i
        if abs(a * (i - y) - b * (j - x)) ** 2 <=
                                  (q + r) ** 2 * (a ** 2 + b ** 2):
            res_line += 1
        result += res_line
print(result, file = outfile)
```

Заметим, что общее количество кеглей равно $1 + 2 + \ldots + n = n(n + 1)/2 \sim n^2$, поэтому асимтотическая сложность приведенного алгоритма – порядка n^2. Такое решение набирает 40 баллов.

Второе решение (асимптотическая сложность порядка n).

Заметим, что для каждого горизонтального ряда кеглей достаточно найти самую левую, и самую правую кеглю, которые собьет шар. Для этого решим неравенство

$$((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)\, (q + r)^2$$

относительно x_p для каждого значения y_p. Это квадратичное неравенство, решением которого является отрезок с концами

$$left = k(y_p - y) - c + x, \quad right = k(y_p - y) + c + x,$$

где $k = v_x/v_y$, $c = (q + r)\, \sqrt{v_x^2 + v_y^2}\, / v_y$.

Теперь нам необходимо найти самый левый и самый правый центр кегли на этом отрезке, то есть самое левое/правое четное или нечетное целое число (четность зависит от четности y_p), а также ограничить координату по модулю числом y_p. Запишем соответствующий фрагмент программы на языке python3:

```
left = math.ceil(left) #округляем "вверх"
right = math.floor(right) #округляем "вниз"
if left % 2 != yp % 2:
    left += 1
if right % 2 != yp % 2:
    right -= 1
left = max(left, -yp)
```

```
right = min(right, yp)
```

Теперь количество сбитых кеглей в горизонтальном ряду легко вычислить по формуле max((*right* – *left*) // 2 + 1, 0) (в процессе вычисления left и right может оказаться, что right < left: в этом случае шар не сбивает ни одной кегли в данном ряду).

Третье решение (целочисленное; ассимптотическая сложность n log n**).**

Преимущество этого решения в том, что оно, оставаясь достаточно быстрым, оперирует лишь целыми числами, а значит, лишено возможных проблем, возникающих из-за ошибок, связанных с округлением вещественных чисел при вычислениях.

Перепишем неравенство (1), избавившись от модуля:

$$-(q + r)\sqrt{v_x^2 + v_y^2} \le ((x_P - x)\, v_y - (y_P - y)\, v_x) \le (q + r)\sqrt{v_x^2 + v_y^2}.$$

Заметим, что при движении по горизонтальному ряду кеглей слева направо средняя часть неравенства возрастает (x_p возрастает, v_y = const > 0). Следовательно, искать целочисленные решения этого неравенства можно бинарным поиском: найдем самое левое и самое правое целое число, принадлежащее отрезку, и прибавим/вычтем 1, если число окажется ненадлежащей четности.

Мы пока еше не достигли цели, поскольку в неравенстве присутствует нецелое число – квадратный корень, а при возведении в квадрат средняя часть неравенства перестает быть возрастающей функцией. Для решения этой проблемы воспользуемся таким трюком. Перепишем условие

$$(x_P - x)\, v_y - (y_P - y)\, v_x) \le (q + r)\sqrt{v_x^2 + v_y^2}$$

эквивалентным образом (пользуясь тем, что $q + r > 0$):

$$(x_P - x)\, v_y - (y_P - y)\, v_x) < 0 \quad \texttt{or} \quad ((x_P - x)\, v_y - (y_P - y)\, v_x))^2 \le (v_x^2 + v_y^2)(q + r)^2.$$

Для проверки этого условия требуются уже только вычисления в целых числах, а значит, цель достигнута. Аналогично переписывается и левая часть двойного неравенства.

Опять же заметим, что вычисления нельзя провести в 64-битном целом типе из-за больших ограничений в условии задачи.

Задача 8. «Abracadabra»

Преобразуем данный нам словарь в такой вид, чтобы в нем можно было быстро искать слово с нужным супрефиксом. Для этого каждую строку $s[0..n-1]$ из словаря преобразуем в такую:

$$s[0]\ s[n-1]\ s[1]\ s[n-2]\ s[2]\ s[n-3]\ \ldots\ s[n-1]\ s[0].$$

Например, слово 'table' превратится в такое: 'tealbblaet'.

Отсортируем обновленный словарь в лексикографическом порядке (по алфавиту). Заметим, что теперь все слова с одинаковым супрефиксом идут подряд.

С каждым из образцов проделаем такое же преобразование. Теперь для каждого обновленного образца *s* найдем бинарным поиском самую первую позицию (left) в словаре, куда его можно вставить, не нарушая лексико-графический порядок. Возьмем строку, полученную из *s* заменой последнего символа на следующий в таблице ASCII:

```
s[n] = chr(ord(s[n] + 1)),
```

и найдем самую левую позицию в словаре, куда его можно вставить (right). Заметим, что все слова, имеющие супрефикс *s*, лежат в промежутке [left, right), поэтому искомое количество слов равно right – left.

Продемонстрируем сказанное на примере. Пусть словарь после обработки имеет вид:

s[0] = 'aa';

s[1] = 'aaaa';

s[2] = 'aaaabbaaaa';

s[2] = 'abba';

s[3] = 'acbbca';

а образец имеет после обработки вид 'aaaa'.

Тогда left будет равно самой левой позиции, куда можно вставить 'aaaa', то есть 1, а right будет равно самой левой позиции, куда можно вставить 'aaab', то есть 3. Ответ в задаче будет равен right – left = 3 – 1 = 2.

Приведем пример кода программы на языке python3:

```
def fkey(word):
   s = [word[i] + word[-i - 1] for i in range(len(word))]
   return "".join(s)

from bisect import bisect_left, bisect_right
fin = open("sufpref.in")
fout = open("sufpref.out", "w")s = fin.read().split('\n')

n = int(s[0])
words = [fkey(word) for word in s[1:n+1]]
words.sort()
k = int(s[n+1])
suf = [fkey(word) for word in s[n+2:n+k+2]]

res = [str(bisect_left(words, s[:-1] + chr(ord(s[-1])+1)) -
bisect_left(words, s)) for s in suf]
print("\n".join(res), file = fout)
```