Всероссийская олимпиада школьников по информатике
Вологодская область, 2021-2022 учебный год
II (муниципальный) этап
9 - 11 классы
Методические рекомендации по разбору задач

Задача 1. Последние цифры (100 баллов)

Первую подзадачу можно решить «в лоб» – просто складывать в цикле последние цифры – остатки от деления на 10.

Для решения второй подзадачи вначале заметим следующее. Чтобы найти ответ для чисел от M до N, можно взять ответ для чисел от 0 до N и вычесть из него ответ для чисел от 0 до M-1. Задача свелась к нахождению ответа на интервале от 0 до X.

Заметим, что сумма цифр в каждом десятке одинакова и равна 0+1+2+…9 = 45. Количество полных десятков равно X div 10. Плюс ещё надо добавить сумму цифр в последнем неполном десятке, которая равна 0+1+…+(X mod 10). По формуле суммы арифметической прогрессии это (X mod 10) * (X mod 10 + 1) / 2. Итого ответ для X равен 45 * (X div 10) + (X mod 10) * (X mod 10 + 1) / 2.

Подставим вместо X числа N и M-1 и вычтем результаты – получаем ответ на задачу. Пример решения на языке Python:

```
def last_digit_sum(x):
    return 45 * (x // 10) + (x % 10) * (x % 10 + 1) // 2

m = int(input())
n = int(input())
print(last_digit_sum(n) - last_digit_sum(m - 1))
```

Задача 2. Парные носки (100 баллов)

Отсортируем носки по цвету. Затем разобьём их на группы следующим образом. Идём слева направо. Пока выполняется $a[i+1] - a[i] \le k$, это всё ещё продолжается одна группа. Если же стало $a[i+1] - a[i] > k$, то с $(i+1)$-го носка начинается следующая группа. Понятно, что группы никак не связаны – никакие два носка из разных групп парными стать не могут.

Теперь посмотрим на каждую группу. Если количество носков в группе чётно, то мы можем использовать их все (первый со вторым, третьим с четвертым, и т.д.). Если количество носков нечётно, то один носок придётся не брать. В этом случае выгодно не брать первый

носок в группе, так как у него наименьший цвет. Пример решения на языке Python:

```
n = int(input())
k = int(input())
a = [0] * n
for i in range(n):
    a[i] = int(input())
a.sort()
a.append(1000000) # barrier element
group_start = 0
ansPairs = ansSum = 0
for i in range(1, len(a)):
    if a[i] - a[i - 1] > k:
        ansPairs += (i - group_start) // 2
        if (i - group_start) % 2 != 0:
            ansSum += a[group_start]
        group_start = i
print(ansPairs, ansSum, sep='\n')
```

Первая подзадача рассчитана на участников, которые не умеют выполнять быструю сортировку (как используя стандартные средства языка, так и писать самостоятельно).

Рассмотрим ещё одно решение на полный балл, в котором нет сортировки в явном виде. Поскольку вариантов цветов всего 1000, то мы можем создать массив, в котором для каждого цвета сосчитать, сколько таких носков имеется.

Пойдём по этому массиву справа налево. Возьмём очередной цвет i. Пусть у нас есть $c[i]$ носков цвета i. Если $c[i] = 0$, то движемся дальше. Иначе посмотрим, не осталось ли у нас не распределённого ранее носка с цветом больше i. Если нераспределённый носок есть, но разница цветов с ним больше k, то этот носок отправляется в непарные, а иначе к $c[i]$ прибавляем 1.

Далее, если $c[i]$ чётное, то мы используем все носки, а если нечётное, то один носок цвета i становится нераспределённым. Пример такого решения на Python:

```
n = int(input())
k = int(input())
c = [0] * 1001
for _ in range(n):
    c[int(input())] += 1
ans_pairs = ans_sum = 0
rest = -1 # unpaired sock color, -1 if absent
for i in range(1000, 0, -1):
    if c[i] == 0:
        continue
```

```
    if rest >= 0:
        if rest - i <= k:
            c[i] += 1
        else:
            ans_sum += rest
        rest = -1
    ans_pairs += c[i] // 2
    if c[i] % 2 != 0:
        rest = i
if rest >= 0:
    ans_sum += rest
print(ans_pairs, ans_sum, sep='\n')
```

Задача 3. Параллельные вычисления (100 баллов)

Для решения первой подзадачи можно вручную найти ответы ко всем возможным входным данным. Поскольку ответ не зависит ни от того, какими буквам обозначены переменные, ни от того, какие знаки операций используются, то количество вариантов совсем небольшое (здесь символом тильды обозначена любая операция):

a
a b ~
a b c ~ ~
a b ~ c ~
a b c d ~ ~ ~
a b ~ c d ~ ~
a b c ~ d ~ ~
a b ~ c ~ d ~

Во второй подзадаче вариантов больше, и получить их все вручную трудоёмко. Здесь возможны какие-либо переборные или эвристические решения.

Для полного решения задачи воспользуемся алгоритмом вычисления выражения в постфиксной записи, который описан прямо в условии задачи. Заменим все переменные в выражении на нули, а любую операцию (неважно, какую) будем всегда вычислять так: берётся максимум из операндов, и к нему прибавляется 1. Тогда результатом вычисления выражения и будет ответ задачи.

По сути, значение каждого операнда теперь – это количество секунд, нужное, чтобы его вычислить. Если у какой-то операции один аргумент станет готов в момент времени t_1, а второй аргумент – в момент t_2, то результат данной операции станет готов в момент $max(t_1, t_2)+1$.

При реализации удобно использовать стек – тогда сложность решения будет линейной. Впрочем, входные данные здесь небольшие, поэтому могут получить максимальный балл и решения, работающие с квадратичной и даже более высокой сложностью. Пример решения на языке Python:

```
p = input().split()
s = []
for e in p:
    if e >= 'a' and e <= 'z':
        s.append(0)
    else:
        result = max(s[-1], s[-2]) + 1
        s.pop()
        s.pop()
        s.append(result)
print(s[0])
```

Задача 4. Количество троек (100 баллов)

Чтобы решить первую подзадачу, достаточно просто перебрать все тройки чисел.

Для решения второй подзадачи вначале заполним следующий массив: $c[i]$ – это количество таких индексов j, что $i < j$ и $a[i]$ делится на $a[j]$. Затем снова переберём все пары индексов $i < j$. Если $a[i]$ делится на $a[j]$, то добавляем к ответу $c[j]$. Сложность составляет $O(n^2)$. Пример решения второй подзадачи на Python:

```
n = int(input())
a = [0] * n
for i in range(n):
    a[i] = int(input())
d = [0] * n
for i in range(n):
    for j in range(i + 1, n):
        if a[i] % a[j] == 0:
            d[i] += 1
ans = 0
for i in range(n):
    for j in range(i + 1, n):
        if a[i] % a[j] == 0:
            ans += d[j]
print(ans)
```

Для полного решения задачи можно воспользоваться методом динамического программирования. Пусть $c[d]$ – текущее количество элементов, равных d. Пусть $f[d]$ – текущее количество пар, в которых первый элемент равен d, и первый элемент пары делится на второй.

Пойдём по входной последовательности справа налево. Встав на очередное число $a[i]$, получим все его делители – пусть это числа d_1, d_2, ..., d_k. К ответу нужно добавить $f[d_1] + f[d_2] + ... + f[d_k]$. Также нам нужно обновить значение $f[a[i]]$ – оно увеличится на $c[d_1] + c[d_2] + ... + c[d_k]$. И также нужно ещё увеличить $c[a[i]]$ на 1.

Сложность такого алгоритма составит $O(n \cdot \sqrt{MaxVal})$, если искать делители циклом до корня из числа. Здесь $MaxVal=10^5$ – максимально возможное входное число. При n и $MaxVal$, равных 10^5, получается около 30 миллионов действий. На компилируемых языках это с запасом укладывается в лимит времени, а при использовании Питона нужно сдавать решение на PyPy. Пример решения на языке Python (с отправкой в систему на PyPy):

```
MaxVal = 1000000

n = int(input())
a = [0] * n
for i in range(n):
    a[i] = int(input())
ans = 0

# c[d] - number of elements = d
c = [0] * (MaxVal + 1)

# f[d] - number of pairs with first element = d
f = [0] * (MaxVal + 1)

for i in range(n - 1, -1, -1):
    divisors = []
    r = 1
    while r * r <= a[i]:
        if a[i] % r == 0:
            divisors.append(r)
            div = a[i] // r
            if div != r:
                divisors.append(div)
        r += 1
    f_ai = f[a[i]]
    for d in divisors:
        ans += f[d]
        f_ai += c[d]
    f[a[i]] = f_ai
    c[a[i]] += 1

print(ans)
```

Задача 5. Полигон для роботов (100 баллов)

В первой подзадаче можно либо получить вручную все (или часть) ответов и внести их в программу, либо написать какой-нибудь перебор вариантов. Сам перебор может выполняться на машине участника, а в систему отправляться только найденные ответы.

Полное решение выглядит так. Во-первых, заметим, что при чётном K решения нет. Это следует из того, что сумма координат клеток в начале и конце пути чётна, а на каждом шаге чётность меняется. При нечётном же K, удовлетворяющем ограничениям в условии, решение всегда существует. Более того, верхняя граница K ($2K \le N^2+1$) специально выбрана достаточно мягко, чтобы можно было построить решение относительно несложно.

Итак, нам нужно, чтобы кратчайший путь содержал ровно K клеток. Очевидно, что если путь будет единственным, то он же будет и кратчайшим. Будем ставить препятствия так, чтобы на полигоне образовался единственный путь, и чтобы он имел длину K.

Напрашивается идея сделать путь в виде «змейки». И, действительно, для многих входных данных этого будет достаточно. В таблице приведены примеры такого пути для N=6 и разных K:

N=6, K=13	N=6, K=15	N=6, K=17
..####	...###	....##
#.####	##.###	###.##
..####	...###	....##
.#####	.#####	.#####
.....	.....	.....
#####.	#####.	#####.

Как видим, изгибы змейки могут не доходить до последнего столбика на разные расстояния, чтобы обеспечить нужную длину пути.

К сожалению, описанное решение не работает для случая, когда N делится на 4, а число K близко к $N^2/2$. В этом случае длины змейки может не хватать. Ниже показан пример решения для N=8, K=29, когда змейки хватило впритык:

```
........
#######.
........
.#######
........
#######.
#######.
#######.
```

Если увеличить *K* до 31, то такое решение уже не работает. Возможный вариант: вместо того, чтобы в конце пути идти вправо и затем вниз, мы добавим ещё вертикальную змейку в последних четырёх строках. Пример для *N*=8, *K*=31:

```
......##
#####.##
......##
.#######
.#...###
.#.#.###
.#.#.###
...#....
```

Как видим в этом примере, вначале идёт горизонтальная змейка (вправо-влево), а в последних четырёх строках – вертикальная (вниз-вверх). Пример решения на языке Python:

```
n = int(input())
k = int(input())
if k % 2 == 0:
    print("No solution")
else:
    blocks = (n - 1) // 4
    length = (n*2 + 2) * blocks + n + (n - 4*blocks - 1)
    end_by_var2 = False
    if length < k:
        length = (n*2+2)*blocks + (n-2)//2*4 + (n-(n-2)//2)
        end_by_var2 = True
    i = 0
    while i + 4 < n:
       len = max(0, n - 1 - (length - k) // 2)
       length -= (n - 1 - len) * 2
       print('.' * (len + 1) + '#' * (n - len - 1))
       print('#' * len + '.' + '#' * (n - len - 1))
```

```
    print('.' * (len + 1) + '#' * (n - len - 1))
    print('.' + '#' * (n - 1))
    i += 4
if not end_by_var2:
    print('.' * n)
    i += 1
    while i < n:
        print('#' * (n - 1) + '.')
        i += 1
else:
    print('.#..' * ((n - 4) // 4) + '.###')
    print('.#.#' * ((n - 4) // 4) + '.###')
    print('.#.#' * ((n - 4) // 4) + '.###')
    print('...#' * ((n - 4) // 4) + '....')
```