Краткие методические рекомендации по решению задач

9–11 классы

При разработке задач для основных туров муниципального этапа олимпиады по информатике Региональная предметно-методическая комиссия исходила из того, что все задачи должны быть оригинальными, разнообразными по тематике и не требовать для своего решения специальных знаний.

При определении уровня сложности авторы задач исходили из того, что комплект задач должен содержать как задачи, доступные многим участникам муниципального этапа, так и задачи, позволяющие проявить себя наиболее сильным участником. В этой связи количество задач на основном туре равно пяти, и как минимум одна задача из них такой сложности, что все участники муниципального этапа ВсОШ по информатике должны её решить полностью, включая и школьников более младших классов. Более того, задачи являются многоуровневыми и предполагают наличие как полных, так и частичных решений, что также будет способствовать тому, что ни одна задача из предложенного комплекта не останется без внимания участников, а сильным участникам позволит продемонстрировать все свои лучшие качества.

Задачи пробного тура предназначены для того, чтобы участники смогли проверить все особенности компьютерной техники и программного обеспечения на своем рабочем месте. Из четырех задач пробного тура задачи W и X являются совсем простыми. Задачи Y и Z предназначены для того, чтобы участники могли лучше познакомиться с системой получения информации о результатах окончательной проверки на примере достаточно сложных задач.

Из пяти задач основного тура **задача 1. «Радиопеленгация»** является самой простой и ориентирована на широкий круг участников.

Задача 1. «Радиопеленгация»

В этой задаче требовалось перебрать все возможные случаи расположения участника соревнования по радиопеленгации относительно базы. Отметим на рисунке, что должна вывести программа для всех областей:

NW	N	NE
W	**БАЗА**	E

SW	S	SE

Например, программа должна вывести ***NE*** при условиях $x > x_2$ ***и*** $y > y_2$, вывести ***N*** при условиях $y > y_2$ и $x_1 < x < x_2$ и т. д. Необходимо аккуратно разобрать все случаи.

Пример решения на языке Python

```
x1 = int(input())
y1 = int(input())
x2 = int(input())
y2 = int(input())
x = int(input())
y = int(input())
if x > x2 and y > y2:
        print("NE")
elif x > x2 and y < y1:
        print("SE")
elif x < x1 and y < y1:
        print("SW")
elif x < x1 and y > y2:
        print("NW")
elif y > y2:
        print("N")
elif y < y1:
        print("S")
elif x > x2:
        print("E")
else:
        print("W")
```

Но у задачи есть и более простое решение. Заметим, что программа должна вывести букву ***N*** во всех случаях, когда спортсмен находится севернее базы, в том числе и в областях ***NE*** и ***NW***, то есть при условии $y > y_2$. Аналогично нужно вывести букву ***S*** всегда при выполнении условия $y < y_1$, букву ***W*** при условии $x < x_1$, букву ***E*** при условии $x > x_2$. При этом все «угловые» направления ***NW, NE, SW, SE*** получатся автоматически – сначала будет выведена одна из букв ***N*** или ***S***, а затем одна из букв ***W*** или ***E***. Такое решение содержит всего четыре условные инструкции if.

Пример решения на языке Python

```
x1 = int(input())
y1 = int(input())
x2 = int(input())
y2 = int(input())
x = int(input())
y = int(input())
ans = ""
if y > y2:
    ans += "N"
if y < y1:
    ans += "S"
if x < x1:
    ans += "W"
if x > x2:
    ans += "E"
print(ans)
```

Задача 2. «Л.О.Р.Д. Вычитание»

Частные решения. Первая подзадача.

Если число ***N*** – однозначное, то ответ равен самому числу. Если число ***N*** – двузначное четное, то потребуется целая часть от деления ***(N – 8) / 2*** операций, чтобы число стало равно 8. Итого *ans = (n - 8) //2 + 8*. Если число ***N*** – двузначное нечетное, то потребуется целая часть от деления ***(N – 9) / 2*** операций, чтобы число стало равно 9. Итого *ans = (n – 9) // 2 + 9*.

```
n = int (input ())
if n % 2:
    ans = (n — 9) //2+9
else :
    ans = (n — 8) //2+8
if n < 10:
    ans = n
print (ans)
```

Вторая подзадача. Моделирование процесса вычитания «в чистом виде»: пока число не обратится в 0, вычитать из него его длину, подсчитывая количество операций.

```
n = int (input ())
ans = 0
while n:
    ans += 1
    n – = len ( str (n))
print (ans)
```

Полное решение. Разберем конкретный пример. Пусть ***N*** = 123456.

Найдем первое пятизначное число, встретившееся нам в процессе уменьшения ***N***. Очевидно, оно имеет такой же остаток от деления на 6, что и ***N***. Возьмем число 99999 и будем уменьшать его на 1, пока остатки не равны. Получим новое ***N*** = 99996. При этом мы использовали целую часть от деления (123456 — 99996) /6 = 3910 операций.

Далее аналогично. Найдем первое четырехзначное число, встретившееся нам в процессе уменьшения ***N***. Очевидно, оно имеет такой же остаток от деления на 5, что и ***N***. Возьмем число 9999 и будем уменьшать его на 1, пока остатки не равны. Получим новое ***N*** = 9996. При этом мы использовали целую часть от деления (99996 – 9996) / 5 = 18000 операций (всего 21910).

Найдем первое трехзначное число, встретившееся нам в процессе уменьшения ***N***. Очевидно, оно имеет такой же остаток от деления на 4, что и ***N***. Возьмем число 999 и будем уменьшать его на 1, пока остатки не равны. Получим новое ***N*** = 996. При этом мы использовали целую часть от деления (9996 — 996) / 4 = 2250 операций (всего 24160).

Найдем первое двузначное число, встретившееся нам в процессе уменьшения ***N***. Очевидно, оно имеет такой же остаток от деления на 3, что и ***N***. Возьмем число 99 и будем уменьшать его на 1, пока остатки не равны. Получим новое ***N*** = 99. При этом мы использовали целую часть от деления (996 – 99) / 3 = 299 операций (всего 24459).

Наконец, найдем первое однозначное число, встретившееся нам в процессе уменьшения ***N***. Очевидно, оно имеет такой же остаток от деления на 2, что и ***N***. Возьмем число 9 и будем уменьшать его на 1, пока остатки не равны. Получим новое ***N*** = 9. При этом мы использовали целую часть от деления (99 – 9) / 2 = 45 операций (всего 24504).

Не забудем, что ***N*** нужно уменьшить до 0, поэтому добавим к ответу еще 9. Итого 24513.

Запрограммируем этот процесс: будем последовательно находить наибольшие числа длины меньше, чем исходное на 1, которое встретится нам в процессе вычитания (и будет последним, полученным при вычитании длины исходного числа), пока исходное число не уменьшится до однозначного

```
n = int (input ())
```

```
ans = 0
while n > 9:
    new_n = int ( ' 9 ' * (len ( str (n)) – 1))
    while new_n % len ( str (n)) != n % len ( str (n)):
        new_n – = 1
    ans + = (n – new_n) // len (str (n))
    n = new_n
ans + = n
print(ans)
```

Задача 3. «Робот»

Для решения подзадачи 1 можно использовать полный перебор.

Для решения подзадачи 2 можно использовать динамическое программирование. Обозначим как *dp[i]* минимальное число команд, необходимое, чтобы мощность батареи робота составляла *i*.

Тогда *dp[i] = min (dp[i – 1]; dp[i – 2]) + 1*, если *i* не кратно *c* и *dp[i] = +1*, если *i* кратно *c*.

Для подзадачи 3, где *c = 2* оптимальна следующая последовательность команд: необходимо *(b – a) / 2* раз использовать команду *Y*, поскольку *b* и *a* оба нечетны, все промежуточные значения будут нечетными.

Полное решение.

Рассмотрим два соседних значения, кратных *c* и числа между ними: *ck; ck+1; ck+2; ... ; c(k+1).* Заметим, что поскольку значение мощности, равное *ck,* запрещено, первое из этих значений, которое примет мощность батареи, равно *ck+1*. Будем далее использовать команду *Y*, пока мощность батареи не примет одно из двух значений: *ck+k–2* или *ck+k–1*. В первом случае далее следует применять команду *X*, чтобы мощность равнялась *ck+k–1*. Теперь, применив команду *Y*, мы переводим мощность в следующий отрезок чисел между кратными *c*.

Таким образом, для преодоления такого отрезка требуется $\lfloor(c + 1)/2\rfloor$ команд (здесь $\lfloor x\rfloor$ означает x, округленное вниз).

Осталось разобраться с начальным отрезком от *a* до первого числа, кратного *c,* и *c* заключительным отрезком от последнего числа, кратного *c,* до *b*. Это можно сделать аналогично. Наконец, следует не забыть о случае, когда между *a* и *b* совсем нет чисел, кратных c, в этом случае ответ равен $\lfloor(b – a) / 2\rfloor$.

Задача 4. «Автогонки»

Решение подзадачи 1.

Отметим очевидный факт: за финишировавшим последним автомобилем, приехавшим последним, никого больше нет. Поэтому для каждого i-го автомобиля (первый элемент в паре i-ой строке) пытаемся найти пару (среди вторых элементов), то есть автомобиль, который приехал вторым. Как мы знаем, для автомобиля, приехавшего последним, такого нет. Действуя таким образом, за $O(n^2)$ операций мы найдем номер последнего финишировавшего автомобиля. По этому номеру восстанавливаем всю очередь.

Аналогично рассуждая, за основу можно взять автомобиль, финишировавший первым.

Решение подзадачи 2.

Очередь автомобилей образует перестановку чисел от 1 до n, поэтому можно подсчитать количество вхождений каждого числа и определить два номера – первого и последнего автомобиля, которые входят по одному разу. Одновременно в специальном массиве запоминаем номера автомобилей, стоящих перед каждым из них. Для автомобиля, финишировавшего последним, этот номер (по умолчанию) равен 0. Определив номер последнего автомобиля, восстанавливаем всю очередь, используя специальный массив финишировавших впереди автомобилей. Итоговое время работы – O(n).

Задача 5. «TOP»

Решение подзадачи 1 и 2.

Выберем одно из чисел и будем просматривать все числа справа от него, подсчитывая на каждом шаге количество совпадающих с ним чисел. Как только обнаружим число, которое встречается более чем $n / 2$ раз, задача решена.

Решение подзадачи 3.

Отсортируем исходный массив чисел и будем сравнивать соседние числа массива. В случае их совпадения будем увеличивать значение счётчика на единицу, подсчитывая таким образом количество вхождений в массив данного числа. Если значение счётчика оказывается больше $n/2$, оптимальный рейтинг найден. В противном случае переходим к просмотру следующего числа и т.д.

Полное решение — алгоритм Боера-Мура.

Рассмотрим первое число массива, будем считать его «претендентом» на звание оптимального числа.

1. Присвоим значение 1 счётчику числа вхождений этого числа в массив.

2. Увеличим значение счетчика на 1, если очередное считываемое число совпадает с «претендентом» и уменьшаем на 1, если не совпадает. Так делаем, пока счётчик не станет равным 0, или не закончится поток.

3. Если после какого-то шага счетчик стал = 0, следующим шагом выполняем шаг 1.

4. То число, которое является «претендентом» в момент окончания потока, и есть искомый популярный элемент.

Приведём основной фрагмент кода на языке Pascal:

```
for i := 1 to n do begin
        read(r);
        if counter = 0 then begin
                majority := r;
                counter := 1;
        end
        else
                if r = majority then inc(counter)
                else dec(counter);
end;
write(majority);
```