Для олимпиады даются 4 задачи: 2 простых, средняя и сложная.
В следующей таблице дана сравнительная характеристика всех 4 задач

Задание	Тематика	Сложность алгоритма	Сложность реализации
1	циклы	Простая	Простая
2	строки	Простая	Простая
3	графы, массивы	Средняя	Простая
4	Длинная арифметика, реализация	Средняя	Сложная

Задача №1. Раздача токенов.

В задаче требуется найти максимальный делитель *D* числа *A*, меньший или равный *N* (количество друзей). Количество токенов у каждого друга тогда будет *A/D*. Для поиска перебираем все делители и выбираем максимальный из них не превышающий *N*. Если просто перебирать все делители от 1 до *N*, то из-за больших ограничений на *N* такое решение будет выполняться долго и не пройдёт тесты по времени.

Для быстрого решения перебор необходимо делать до квадратного корня из *A*, учитывая, что любое число, на которое делится *A*, даёт два делителя: *d* и *A/d*.

```
var a,n,i,j,k : integer;
begin
  read(a,n);
  k := round(sqrt(a));
  j := 1;
  for i := 1 to k do
  begin
    if a mod i = 0 then
    begin
      if (i > j) and (i <= n) then
        j := i;
      if (a div i > j) and (a div i <= n) then
        j := a div i;
    end;
  end;
  writeln(a div j, ' ', j);
end.
```

Задача №2. Восстановление адреса.

Заметим, что возможно всего 2 варианта строк, удовлетворяющих условию задачи: 01010101… или 10101010…

Таким образом задача сводится просто к проверке равенства входной строки с одним и вторым вариантом. Для этого вначале получаем эти 2 варианта строк одинаковой длины с входной строкой, и посимвольно сравниваем каждую строку с входной, игнорируя символы ? и сравнивая все остальные.

Если обе строки возможны, то ответ MANY. Если ни одна строка невозможна, то ответ NO. Если возможна только 1 из строк, то выводим эту строку.

```
var
  s, s0, s1 : string;
  good0, good1 : boolean;
  i : integer;
begin
  readln(s);
  s0 := '';
  s1 := '';
  good0 := true;
  good1 := true;
  for i := 1 to length(s) do
  begin
    s0 := s0 + chr(49 - i mod 2);
    s1 := s1 + chr(48 + i mod 2);
    if s[i] <> '?' then
    begin
      if s[i] <> s0[i] then
        good0 := false;
      if s[i] <> s1[i] then
        good1 := false;
    end;
  end;
  if good0 and good1 then
    writeln('MANY')
  else if good0 then
    writeln(s0)
  else if good1 then
    writeln(s1)
  else
    writeln('NO');
end.
```

Задача №3. Начало блокчейна.

Заведём 2 массива: текущий баланс каждого адреса и минимальный баланс каждого адреса. Вначале все балансы равны 0.

Симулируем выполнений всех транзакций блокчейна, уменьшая текущий баланс адреса с которого переводят и увеличивая текущий баланс адреса на который переводят на нужное количество токенов. Также после каждой транзакции обновляем минимальный баланс адреса если необходимо.

После выполнения всех транзакций, минимальные балансы некоторых адресов будут содержать отрицательные числа. Это значит, что во время выполнения исходное значение баланса уменьшалось до этого числа. Значит, исходный баланс должен быть равен минимальному балансу по модулю.

Складываем минимальные балансы всех адресов по модулю, это и будет ответом.

```
var
  min,a : array[0..1000005] of integer;
  addr : array[0..1000] of integer;
  sum,n,i,x,y,w : integer;
begin
  read(n);
  for i := 0 to 1000000 do
  begin
    a[i] := 0;
    min[i] := 0;
  end;
  for i := 1 to n do
  begin
    read(x,y,w);
    dec(a[x],w);
    if a[x] < min[x] then
      min[x] := a[x];
    inc(a[y],w);
  end;
  sum := 0;
  for i := 0 to 1000000 do
    sum := sum + (-min[i]);
  writeln(sum);
end.
```

Задача №4. Красивый баланс.

Простое переборное решение (увеличение числа на 1 до тех пор, пока не встретится палиндром) - при заданных ограничениях будет работать слишком долго. Одним из быстрых решений является следующее.

Будем восстанавливать первый палиндром, больший заданного числа, по 1 цифре с конца по алгоритму, схожему с алгоритмом увеличения числа на 1 "в столбик". Будем держать 1 "в уме" - если необходимо увеличить следующую цифру на 1. И 0, если увеличивать необязательно.

В самом начале алгоритма увеличить на 1 обязательно, т.к. число должно быть больше исходного. Далее идем поочередно с последней цифры до середины числа (i=1 для последней цифры, i=2 для предпоследней и т.п.). Возможны следующие варианты:

1) i-я цифра с конца равна i-й цифре с начала: ничего не меняем, переходим к следующей цифре, не меняя числа "в уме" (если увеличивать не надо было, то так всё и останется - в палиндроме i-я цифра с начала и с конца должны быть равны, а если увеличивать было необходимо - то i-я цифра все

равно должна быть одинаковой с конца и с начала, но чтобы этого добиться - необходимо увеличить на 1 следующую цифру)

2) i-я цифра с конца меньше i-й цифры с начала: увеличиваем i-ю цифру с конца, делая ее равной i-й цифре с начала. Признак "в уме" становится равным 0, т.к. мы увеличили i-ю цифру, значит следующую цифру менять необязательно

3) i-я цифра с конца больше i-й цифры с начала: чтобы уменьшить i-ю цифру с конца, необходимо увеличить на 1 следующую цифру, тогда i-я цифра может быть любой, а т.к. нам необходимо чтобы она была равна i-й с начала, то присваиваем i-й цифре с конца i-ю цифру с начала и устанавливаем признак "в уме" равным 1.

Дойдя таким образом до середины числа возможны варианты:

1) "В уме" равно 0. Выводим ответ, полученный в результате алгоритма. Т.к. на каждом шаге алгоритма мы присваивали i-й цифре с конца i-ю цифру с начала, то в результате у нас получился палиндром.

2) "В уме" равно 1. Прибавляем 1 к средней цифре (или к последней цифре "левой половины") с учетом переноса (если было 9, то эта цифра становится равной 0, и повторяем для следующей цифры). После этой операции в правой половине мы можем установить любое число, поэтому делаем получившееся число палиндромом, т.е. каждой i-й цифре с конца присваиваем i-ю цифру с начала. И выводим получившееся число.

В результате этого алгоритма всегда получится наименьшее число, большее заданного, и которое будет палиндромом.

```
var
  i,j,p : integer;
  s : string;
begin
  readln(s);
  p := 1;
  i := 1;
  j := length(s);
  while true do
  begin
    if i >= j then
    begin
      if p = 0 then break;
      if i > j then dec(i);
      while (i >= 1) and (s[i] = '9') do
      begin
        s[i] := '0';
        dec(i);
      end;
      if i = 0 then
      begin
        for i := 1 to length(s) do s[i] := '0';
```

```
        s := '1' + s; s[length(s)] := '1';
        break;
      end;
      inc(s[i]);
      for i := 1 to length(s) div 2 do
        s[length(s)-i+1] := s[i];
      break;
    end;

    if (s[i] = s[j]) then
    begin
      inc(i);
      dec(j);
    end else
    if s[j] < s[i] then
    begin
      s[j] := s[i];
      inc(i);
      dec(j);
      p := 0;
    end else
    begin
      s[j] := s[i];
      inc(i);
      dec(j);
      p := 1;
    end;
  end;
  writeln(s);
end.
```