Задача 1. «Ролевая игра»

Данная задача является наиболее простой в комплекте задач для первого тура, и ее решение основано на вычислении для каждого игрока в отдельности необходимого ему числа значков. Если $m < k$, то ответ — m белых значков. Иначе игроку могут понадобиться $(k - 1)$ белых значков, чтобы набрать уровень $(k - 1)$, а чтобы набрать уровень m, игроку понадобятся $\left\lfloor \frac{m}{k} \right\rfloor$ красных значков. Если полученное суммарное количество значков умножить на число игроков n, то это произведение и будет ответом на исходную задачу.

Ниже приведен фрагмент программы, реализующий описанный подход.

```
read(n, m, k);
if (m < k) then
  writeln(n * m)
else
  writeln(n * (k - 1 + m / k));
```

Задача 2. «Колесо Фортуны»

Решение данной задачи следует начать с ответа на вопрос: какое количество переходов между секторами может совершить стрелка при заданных значениях величин a, b и k. Легко видеть, что при фиксированной начальной угловой скорости x стрелка совершит $\left\lfloor \frac{x}{k} \right\rfloor$ переходов между секторами. Таким образом, стрелка может совершить от $l = \left\lfloor \frac{a}{k} \right\rfloor$ до $r = \left\lfloor \frac{b}{k} \right\rfloor$ переходов между секторами.

Теперь рассмотрим случай движения по часовой стрелке. Если придать стрелке минимально допустимую начальную скорость, то она остановится в секторе с номером $start = (l \bmod n) + 1$. Также стрелка может остановиться и в последующих $(r - l)$ секторах, тогда ответом на задачу будет максимальное число, записанное в секторе с номером $start$ и последующих $(r - l)$ секторах. Если $r - l \geq n - 1$, то стрелка может остановиться в любом из секторов, в этом случае поиск ответа будет связан с нахождением максимального из всех чисел на барабане.

Для поиска максимума необходимо просмотреть все числа в упомянутых секторах. Это можно осуществить за время O(n), просматривая каждый из секторов не более одного раза.

Аналогичным образом рассматривается случай движения стрелки в противоположном

направлении, то есть, против часовой стрелки. Получаемая в итоге временная сложность решения задачи будет составлять O(n).

Задача 3. «Форматирование текста»

Решение данной задачи основано на внимательном прочтении условия задачи и корректном преобразовании исходного текста в соответствии с описанными правилами. Удобнее всего это сделать, считав весь текст и произведя разбиение по пустым строкам на абзацы. Если абзац не содержит символов, то его не стоит далее рассматривать в решении. Далее каждый из абзацев следует разбить на последовательности символов, не содержащие пробелы, используя в качестве разделителей символы пробела и перевода строк. При этом следует отделять слова — неразрывные последовательности букв и цифр— от прочих последовательностей. Как и в случае с абзацами, получившиеся пустые слова следует убрать из дальнейшего рассмотрения.

После выполнения названных действий следует соединить слова с последующими последовательностями из знаков препинания, образуя тем самым «расширенные» слова, а затем необходимо последовательно вывести абзацы. Для вывода абзаца рекомендуется в каждый момент поддерживать номер символа начальной позиции для вывода следующего «расширенного» слова. Если «расширенное» слово не умещается на текущей строке, то следует начать его вывод с новой строки.

Задача 4. «Космические исследования»

Решение данной задачи следует начать с рассмотрения возможности перемещения какой-либо фиксированной позиции зоны съемки. Понятно, что для перемещения этой зоны съемки вверх необходимо, чтобы были сфотографированы все объекты на нижней линии зоны съемки. Если объектов на этой строке нет, то можно считать, что все объекты сфотографированы. В противном случае, если крайний правый и крайний левый объекты строки уже были сфотографированы, то и все объекты между ними тоже можно было сфотографировать. В результате такого анализа можно сделать следующий вывод: для перемещения зоны съемки вверх необходимо знать, были ли сфотографированы крайние объекты в рассматриваемой горизонтали.

С учетом сказанного один из подходов решения данной задачи основан на использовании идей динамического программирования. В частности, для каждой возможной позиции зоны съемки и подмножества уже сфотографированных крайних объектов, находящихся на горизонталях, занимаемых зоной съемки, рассчитывается минимальное количество сдвигов, которое необходимо для достижения описанного выше состояния. При этом считается, что все объекты, расположенные на горизонталях ниже горизонталей,

занимаемых зоной съемки, уже сфотографированы.

Для перемещения зоны съемки можно использовать следующие варианты:

1) можно сдвинуть зону съемки на одну позицию вправо или влево;
2) если оба крайних объекта нижней горизонтали, занимаемой зоной съемки, уже были сфотографированы, то можно переместить зону съемки на одну позицию вверх (на север).

В обоих случаях необходимо проверять, покрывает ли зона съемки какие-то крайние объекты. Если это так, то их можно считать сфотографированными. Если на какой-то горизонтали объектов нет, то также можно считать, что крайние объекты сфотографированы.

Начальное состояние динамики — зона съемки перемещена в левый нижний угол и сфотографированы все крайние объекты, которые она покрывает. Обход состояний можно совершать с помощью обхода в глубину. Все состояния, содержащие горизонталь, на которой находятся объекты с самой большой *y*-координатой и для которых все крайние объекты сфотографированы, являются возможными конечными позициями при фотографировании всех объектов. В итоге получаем, что ответом на задачу является минимальное число сдвигов, которое необходимо для достижения хотя бы одного из выше указанных состояний.

Сложность данного алгоритма оценивается исходя из количества состояний динамики. Так как динамика подсчитывает для всех возможных разумных (в пределах области) позиций зоны съемки и для всех возможных состояний *2k* крайних объектов, то число состояний динамики составляет $O\left(S \cdot 2^{2k}\right)$, где S — площадь области. Переход из одного состояния в другое происходит за время $O(k)$, так как необходимо для каждой из горизонталей, занимаемых зоной съемки, проверить крайние объекты на попадание в новую позицию зоны. С учетом сказанного общая временная сложность представленного алгоритма будет определяться $O\left(S \cdot k \cdot 2^{2k}\right)$.

Задача 5. «Шахматная доска»

Для решения данной задачи прежде всего следует проанализировать зависимость искомого ответа от размера шахматной доски. Если общее число клеток четно, то на поле будет равное количество белых и черных клеток. Если же число клеток нечетно, то большее количество клеток будут иметь такой же цвет, что и у угловых клеток доски.

Несложно заметить, что все клетки, имеющие четную сумму номеров строки и столбца, окрашены в тот же цвет, что и угловая клетка поля. Таким образом, зная номера строки и столбца и цвет одной из клеток, легко определить цвет угловой клетки и каких клеток больше.

Фрагмент программы, реализующий описанный подход, представлен ниже.

```
read(m, n, i, j, c);
if ((m mod 2) * (n mod 2) = 0) begin
  writeln('equal');
end else
  if ((((i + j) mod 2) xor c) = 0) then
    writeln('black')
  else
    writeln('white');
```

Задача 6. «Чемпионат по стрельбе»

На первом этапе решения данной задачи необходимо узнать, сколько очков имеют победители соревнований. Для этого необходимо найти максимум среди всех очков. Далее можно определить, какое максимальное число очков мог иметь папа школьника. Для этого можно использовать следующий алгоритм: перебирая позиции слева направо, для каждой проверить, что очки участника оканчиваются на пять, а у следующего участника очков меньше. Здесь также необходимо проверить, был ли до этого участник, являвшийся победителем, то есть, имевший максимум очков.

При реализации описанного выше алгоритма достаточно хранить флаг, который обозначал бы, что победитель уже встречался ранее, и обновлять его после рассмотрения каждого игрока. Фрагмент программы, реализующий эти идеи, представлен ниже.

```
read(n);
for i := 1 to n do
  read(a[i]);
max := 0;
for i := 1 to n do
  if (max < a[i]) then
    max := a[i];
winner := false;
fp := 0;
for i := 1 to n - 1 do begin
  if (winner) and (a[i] mod 10 = 5)
      and (a[i] > a[i + 1]) and (fp < a[i]) then
     fp := a[i];
  if (a[i] = max) then
    winner := true;
end;
if (fp = 0) then
  writeln(0)
else begin
  more := 0;
  for i := 1 to n do
    if (a[i] > fp) then
```

```
        inc(more);
    writeln(more + 1);
end;
```

Задача 7. «Делители»

Для решения данной задачи необходимо, прежде всего, получить все возможные делители числа *n*. Это можно осуществить, перебирая все числа до $\sqrt{n}$ и проверяя, являются ли они делителями *n.* Оставшиеся делители получаются путем деления *n* на уже полученные делители. При указанных в условии задачи ограничениях число различных делителей может достигать 768.

Далее, как не так сложно догадаться, получить искомый ответ можно с использованием перебора всех возможных наборов из *k* различных делителей числа *n*. При этом для получения эффективного решения необходимо использовать ряд отсечений. Во-первых, для каждого из делителей *d* можно создать список делителей числа *n*, которые больше *d* и взаимно просты с *d*. Это позволяет, перебирая делители в возрастающем порядке, просматривать в качестве следующего возможного числа в наборе не все делители.

Во-вторых, если в набор уже включены *i* чисел и *i*-м числом является делитель *d* и произведение чисел в наборе *m,* то при $m \cdot d^{k-i} \geq n$ дополнить набор до *k* элементов уже не получится.

Задача 8. «Дом у дороги»

Существует несколько способов решения данной задачи. Первый из них базируется на идеях двоичного поиска и пересечения полуплоскостей. Рассмотрим его более подробно.

Будем условно называть трассы прямыми. Допустим известно, что максимальное расстояние от искомой точки до самой дальней от нее прямой составляет не более *l*. Тогда вокруг каждой из прямых образуется полоса — множество точек, отдаленных от прямой не более чем на *l* и определяющих все возможные точки постройки офиса. Если расстояние до дальней из прямых не более *l*, то пересечение полос будет содержать хотя бы одну точку. При этом любая точка из пересечения всех полос обладает следующим свойством: расстояние от нее до самой дальней от нее прямой не превышает *l*. Если же построить полосы для значения *l,* меньшего чем для искомой точки, то пересечение полос будет пустым.

Основываясь на этой идее можно получить решение, которое с помощью двоичного поиска будет находить минимальное максимальное расстояние до прямых. Для этого при фиксированном значении расстояния необходимо построить пересечение всех полос. Каждая полоса, как не сложно заметить, представляет собой пересечение двух полуплоскостей, то есть, пересечение полос является пересечением 2*n* полуплоскостей.

Для решения этой подзадачи можно использовать один из алгоритмов, определяющих пересечение O(n) полуплоскостей за O($n \log n$) действий. Данный алгоритм совершает $O(steps \cdot n \cdot \log n)$ действий, где *steps* — необходимое количество итераций двоичного поиска, которое для данных в условии задачи ограничений составляет ~50.

Стоит заметить, что в алгоритме пересечения полуплоскостей логарифмическая часть возникает из-за сортировки полуплоскостей по соответствующим им векторам нормали. Однако в данном случае на новой итерации двоичного поиска вектора нормалей не изменяются (изменяются другие параметры полуплоскостей), что позволяет совершать сортировку всего один раз. Таким образом временные затраты сокращаются до $O(n \cdot (steps + \log n))$.

Второй подход решения исходной задачи базируется на следующей идее. Сформулируем данную задачу в терминах задачи линейного программирования. Переменными здесь будут являться искомые координаты x и y, а также минимизируемое расстояние z. При этом должны выполняться следующие ограничения:

1) $z \geq 0$ — расстояние должно быть неотрицательным;
2) если i-я прямая имеет нормализованные коэффициенты a_i, b_i, c_i и $a_i^2 + b_i^2 = 1$, тогда должны выполняться неравенства $a_i \cdot x + b_i \cdot y + c_i + z \geq 0$ и $-a_i \cdot x - b_i \cdot y - c_i + z \geq 0$. Несложно заметить, что из этих неравенств следует неравенство $-z \leq a_i \cdot x + b_i \cdot y + c_i \leq z$, то есть, $|a_i \cdot x + b_i \cdot y + c_i| \leq z$, что гарантирует ограничение на расстояние до любой прямой.

В общей сложности в этом случае получается $O(n)$ неравенств. Полученную задачу можно решить за линейное относительно количества неравенств время. Применив такой метод, получаем алгоритм решения исходной задачи, работающий за линейное время.