Задача A. Розы

При считывании значения числа лепестков l очередной розы проверяем условие $l > k$. При выполнении этого условия добавляем к текущему значению выручки слагаемое $2r$; в противном случае стоимость розы уменьшаем на величину $(k - l)$ и к значению выручки добавляем $r - (k - l)$.

Приведём код этой программы на языке C++.

```
int k, r, n, l;
cin >> k >> r >> n;

long long ans = 0;
for (int i = 0; i < n; ++i)  {
  cin >> l;
  if (l > k) ans += 2 * r;
  else ans += r - (k - l);
}
cout << ans;
```

Задача B. До последней стружки

Предположим, что имеется x заготовок; подсчитаем количество втулок, которое можно получить из этих заготовок. На первом этапе получится x втулок и x единиц стружки; из них в свою очередь можно получить x/m новых втулок (целочисленное деление). Значит, на втором этапе имеем $s = x + x / m$ втулок и $x / m + x \,\%\, m$ единиц стружки. (Число $x \,\%\, m$ равно остатку от деления x на m.) На следующем этапе рассуждения повторяются: из $x_1 = x / m$ единиц стружки снова получаем x_1 / m втулок и такое же количество единиц стружки, то есть число втулок увеличится на x_1 / m. Таким образом, для подсчёта числа бронзовых втулок из x заготовок можно использовать процедуру:

```
long long g(long long x) {
    long long s = x;
    while (x >= m)    {
        s += x / m;
        x = x / m + x % m;
    }
    return s;
}
```

Как найти необходимое количество заготовок? Например, можно воспользоваться бинарным поиском по ответу. Идея этого алгоритма состоит в следующем. Будем искать нужное количество заготовок в виде неизвестного числа x на отрезке $[1; n]$. Выберем в качестве начального значения середину этого отрезка, то есть число $x = n/2$. Если этого количества заготовок недостаточно, будем продолжать дальнейший поиск на отрезке $[n/2; n]$, иначе — на отрезке $[1; n/2]$. На каждом шаге выбираем середину текущего отрезка и сравниваем количество втулок, которое можно получить (процедура g), с требуемым количеством. Если полученных втулок меньше, передвинем *левую* границу искомого отрезка в эту середину, иначе — передвинем *правую* границу. Продолжаем этот процесс до тех пор, пока разность между левой и правой границей текущего отрезка больше 1.

Сложность этого алгоритма — $O(\log n \cdot \log m)$.

Задача C. Произведение цифр

(*Фольклор.*)

В задаче требуется найти наибольшее число, составленное из различных цифр, произведение цифр которого равно заданному значению n ($1 \leqslant n \leqslant 10^9$).

Для небольших значений n задачу можно решить, перебирая числа в порядке возрастания; для каждого из чисел нужно подсчитать произведение цифр в его десятичной записи, затем сравнить полученное произведение с заданным числом n.

Разложим число n в произведение простых множителей $n = p_1^{a_1} p_2^{a_2} \dots p_k^{a_k}$. (Как известно, для этого достаточно выполнить количество операций порядка $\sqrt{n}$.) Поскольку каждая цифра искомого числа меньше 10, это произведение должно содержать только простые множители $p_i < 10$. Поэтому если в разложение n входят простые числа p_i, большие 10, запишем ответ `-1`.

Итак, в разложение n могут входить только простые множители 2, 3, 5 и 7.

Теперь выясним, какие значения могут принимать показатели a_i этих простых множителей. В десятичной записи искомого числа x могут встретиться только четыре чётные цифры 2, 4, 6, 8, их произведение делится на 2^7, поэтому максимальный показатель простого числа $p = 2$ равен 7. Аналогично, есть только три цифры 3, 6, 9, которые делятся на 3; их произведение делится на 3^4, поэтому максимальный показатель простого числа $p = 3$ равен 4. Для простых чисел 5 и 7 максимальный показатель, очевидно, равен 1. Поэтому если в разложение n простые числа 2, 3, 5 и 7 входят с показателями больше 7, 4, 1 и 1 соответственно, снова запишем ответ `-1`.

Ясно, что цифры 5 и 7 входят в искомое число x только в случае, если показатели простых чисел 5 и 7 равны 1. Осталось разобрать случай $n = 2^{a_2} \cdot 3^{a_3}$, где $0 \leqslant a_2 \leqslant 7$, $0 \leqslant a_3 \leqslant 4$.

При $a_3 = 4$ запись числа x обязательно содержит цифры 3, 6 и 9, причём если $a_2 = 0$ — ответ `-1`. Если же $a_2 = 1$, других чётных цифр, кроме 6, у числа x нет. Для $a_2 = 2$ в записи x будут входить чётные цифры 2 и 6, и так далее. Аналогичным образом, разбираются остальные значения a_3.

Задача D. Ковбой Джонни

Пусть a_i — номер коровы, стоящей на месте i. Будем называть корову, стоящую на месте i, просто коровой i. В задаче требуется подсчитать количество последовательностей a_1, $a_2, \dots, a_n$, которые удовлетворяют следующим ограничениям:

- $1 \leqslant a_i \leqslant m$ для каждого i от 1 до n;
- $a_i < a_{i+2}$ для каждого i от 1 до $n-2$;
- $a_i < a_{i+3}$ для каждого i от 1 до $n-3$.

Подзадача 1. Переберём все возможные нумерации для каждой коровы i и отберём среди них те, которые удовлетворяют условию задачи.

Подзадача 2. Воспользуемся идей динамического программирования. Пусть $f_{i-1,j,k,l}$ — количество нумераций всех коров в предположении, что уже выбраны номера для коров от 1 до $i-1$, причём $a_{i-1} = j$, $a_{i-2} = k$ и $a_{i-3} = l$. Интересующие нас нумерации $f_{i,x,j,k}$, где $x = a_i$, можно получить из каждого варианта нумерации $f_{i-1,j,k,l}$ коров добавлением номера коровы i в виде числа x, большего k и l, и номера коровы $i+1$ как числа, большего j и k. Сложность такого решения — $O(nm^4)$.

Подзадача 3. Снова воспользуемся идей динамического программирования. Пусть $f_{i,j,k}$ — количество нумераций всех коров в предположении, что уже выбраны номера для коров от 1 до i, и пусть $a_i = j$ и $\max(a_{i-1}, a_{i-2}) = k$. По условию $a_{i+1} > k$, причём при переходе от i к $i+1$ имеем: $\max(a_i, a_{i-1}) \geqslant \max(j, k)$, то есть наименьшее возможное значение $\max(a_i, a_{i-1})$ равно $\max(j, k)$. Перебирая такие нумерации, как в подзадаче 2, получим решение, сложность которого $O(nm^3)$.

Подзадача 4. Пусть $m = \lceil n/2 \rceil$. Из условия задачи следует, что $a_i < a_{i+2} < a_{i+4} < \dots$ для любого i. В случае чётного n ($n = 2m$) количество чётных и нечётных мест совпадает с количеством номеров m, значит, и на чётных и на нечётных местах использованы все номера от 1 до m ровно по одному разу, причём их положение определяется однозначным образом. Другими словами, в случае чётного n существует только одна требуемая нумерация коров.

Если же n — нечётное ($n = 2m-1$), то количество нечётных мест совпадает с m, и значит, положение номеров на нечётных местах определяется однозначно. Количество же чётных мест равно $m-1$, поэтому ровно одно из чисел от 1 до m не использовано на этих местах. Значит, в случае нечётного n существует m требуемых нумераций.

Подзадача 5. Пусть $f_{i,j}$ — количество нумераций i коров вида a_1, $a_2, \dots, a_i$ с номерами в диапазоне $[1; j]$. Начальные значения этой величины очевидны: $f_{0,j} = 1$ и $f_{1,j} = j$. Тогда ответом в задаче будет число $f_{n,m}$.

Перепишем условие задачи в виде: при условии $1 < i + 1 < j \leqslant n$ должно выполняться $a_i < a_j$.

Для подсчёта вариантов нумераций i коров рассмотрим возможные положения коров с номерами j. Сразу заметим, что если $a_k = j$, то $k \in \{\, i-1, i \,\}$. Действительно, если $k < i-1$, то $a_i > a_k$, но a_k принимает самое большое возможное значение j в диапазоне $[1; j]$, противоречие.

Все возможные положения коров с номерами j можно описать с помощью четырёх множеств A_j (в соответствии с условием — $a_{i-1} = j$ или $a_i = j$):

1. $A_j = \varnothing$.

В последовательности нет номера j, количество таких нумераций равно $f_{i,j-1}$.

2. $A_j = \{i\}$.

Для коровы i есть только один возможный номер $a_i = j$. Значит, первые $(i-1)$ чисел просто составляют последовательность с весовым диапазоном $[1; j-1]$, поэтому для этой ситуации количество нумераций составляет $f_{i-1,j-1}$.

3. $A_j = \{i-1,\, i\}$.

Другими словами, $a_{i-1} = a_i = j$. Эта ситуация разбирается досточно просто. Поскольку первые $(i-2)$ чисел составляют последовательность с весовым диапазоном $[1; j-1]$, значит, общее число нумераций будет $f_{i-2,j-1}$.

4. $A_j = \{i-1\}$.

Другими словами, $a_{i-1} = j$. В этом случае подсчёт числа нумераций более сложный.

Поскольку $a_i \neq j$, должны выполняться неравенства $1 \leqslant a_i < j$. Рассматривая нумерации с числом $k = a_i \in [2; j-1]$, заметим, что первые $(i-2)$ числа образуют последовательность с рангом $[1; k-1]$. Всего получаем $\sum_{k=1}^{j-1} f_{i-2,k-1}$ способов нумерации, или, что то же самое, $\sum_{k=0}^{j-2} f_{i-2,k}$ способов.

Таким образом, уравнение перехода имеет вид

$$f_{i,j} = f_{i,j-1} + f_{i-1,j-1} + f_{i-2,j-1} + \sum_{k=0}^{j-2} f_{i-2,k}\,.$$

Используя префикс-суммы, можно оптимизировать подсчёт числа нумераций $f_{i,j}$ по этой формуле. Предварительный подсчёт суммы за $O(nm)$ операций, позволяет получить ответ за $O(1)$ операций. Окончательная сложность алгоритма — $O(n^2 + T)$.