A. Accommodation Plan (Roman Bilyi)

Let's count the number of arrangement such that vertex v is the highest among all vertices where all friends can meet. Let A_v be the number of vertices u such that $dist(v, u) \le L$. Let B_v be the number of vertices u such that $dist(v, u) \le L$ and $dist(parent(v), u) > L$. We should add $count(A_v, k) - count(A_v - B_v, k)$ to the answer, where $count(n, k) = n!/(n-k)!$.

Let $T(v, to)$ be the number of vertices u such that $dist(v, u) \le L$ and path from v to u starts with edge (v, to). Here (v, to) is oriented edge of the tree (there are $2N-2$ such edges). Values T can be computed using centroid decomposition, and values A_v, B_v can be expressed as a combination of T.

B. Card Game (Roman Bilyi)

Let's iterate over all divisors of N. For any given divisor K, check if it can be the number of suites (i. e. it's greater than or equal to the maximum given suite) and if the maximum number of occurrences of any given suite is not greater than N/K. If a single valid divisor is found, the answer is positive, otherwise it's negative.

C. The Most Expensive Gift (Roman Bilyi)

One of the three letters appears at least $n/3$ times in the given string. So we have substring with cost $n^2/9$. Now we need to iterate over all smaller values of cycle size. There are $3^8+3^7+...+1$ such substrings possible. We can check every cycle in $O(n)$.

D. Cut the Cake (Vitalii Herasymiv)

Let's solve X and Y axes separately. We can sort all x-coordinates and make cuts between coordinates with indices k and $k+1$, $2k$ and $2k+1$, and so on (1-based index). The same for y-coordinates, and after that check if the cuts are valid (i. e. each part has exactly one candle). You can see that if the answer exists, then this algorithm will find it. Otherwise, the validation will fail.

E. Message (Vitalii Herasymiv)

Slow solution to the problem would be a dynamic programming with $O(n^2)$ running time, which is a simple *dp[postFirst][posSecond]*, where we can skip a character of the first string only if *posSecond* is before the first position of that character in the second string, or after the last one.

Let's divide the second string in 52 (26x2) positions of the first and the last occurrence of each character. We can now actually process whole parts of second string at the same time instead of each character separately. When a part in the second string is fixed, we know that some of the 26 characters must be skipped in the first string, and some must be always taken. So if we skip those characters that

must be skipped, the check turns into string matching problem. So we can change our state to *dp[posFirst][partSecond]*, and check if a transition is valid using KMP algorithm or string hashing.

F. Bad Word (Roman Bilyi)

If the initial string isn't a palindrome, the answer is 1. If the string has one of the following structure, the answer is -1: *aaaaaaaa, abababababa, aaaabaaaa.*

Let's prove that in all other cases the answer is 2. Proof by contradiction. Let n be the length of the string. Let a be the first letter. Let p be position of the first letter other than a. Let's call such letter b (0-indexing). We know that position p exists and $p < n/2$ (in other case string matches the 1st or the 3rd pattern). We know that $s[n-1] = a$ and $s[n-1-p] = b$ as our string is palindrome. Also, $s[0..p]$ isn't a palindrome so $s[p+1..n-1]$ should be palindrome (otherwise we already found the answer 2). If $p > 1$ then $s[0..p+1]$ also isn't a palindrome so $s[p+2..n-1]$ should be palindrome. Both $s[p+1..n-1]$ and $s[p+2..n-1]$ should be palindromes and it's possible in only case that $s[p+1..n-1]$ consists of the same letter. But it contradicts with $s[n-1] = a$ and $s[n-1-p] = b$. So the only possible case is $p = 1$. That means that the string is *ab...ba*. Now $s[0..1]$ isn't palindrome so $s[2..n-1]$ should be palindrome. So the string should be *abab...ba*. We can continue this process and string will match the second pattern.

G. Zenyk, Marichka and Interesting Game (Roman Bilyi)

We can take size of each pile modulo $A+B$.
Then there are 2 solutions:

1. There are only 4 types of piles: $X < min(A,B)$; $min(A, B) \le X \le max(A, B)$; $\max(A,B) \le X < 2min(A,B)$; $2min(A,B) \le X$. And we can find the winner analytically for all cases.
2. Both players should always choose the biggest pile, so we can just iterate the process (no more than $2N$ moves). To prove this approach formally you have to examine all cases mentioned earlier, but the overall logic is the following. Consider the case $A < B$, and let's split all piles on each turn in two types based on the number of stones X: type 1 ($A \le X < B$) and type 2 ($X \ge B$). The player A can use piles of both types, but first off he has to choose piles of type 2 if possible (to try to minimize the number of moves the other player can make by spoiling type 2 piles, and possibly create new groups of type 1), and after that type 1 (which the other player can't use). Thus, choosing the maximum value on each turn for player A is an optimal strategy. The other player can only use piles of type 2, and on each turn he want's to choose the maximum value to make sure player A makes fewer moves to type 2 piles and makes fewer type 1 piles from piles of type 2.

The proof of "We can take size of each pile modulo A+B": It's obviously correct when second player wins smaller game (make turn to the same pile as first player if it's possible, or use his strategy in smaller game otherwise). First player makes first move as in smaller game and becomes second.

H. Frog Jumping (Vitalii Herasymiv)

If there is some set of frogs that can make it to the end without big jumps, then the other frogs can jump straight from the first to the last stones. So the subtask is to find the maximum number of frogs that can make it to the end simultaneously without big jumps. This can can be solved in the following way: consider 'window' of size *d*, and for each position of the window count the number of stones in it. Then the answer to the subtask is the minimum of those values. (Another way to find the answer is to use binary search and greedy). Those frogs that cost the most should go for it without big jumps, and the others – straight jump from the first to the last.

If no frog can make it without big jumps, it can be easily proven (by examining a contradictory situation) that the optimal way is to make the cheapest frog jump over all stones, while the rest of the frogs jump straight from the first to the last.

I. Slot Machine (Roman Bilyi)

Let *S(X)* be the number of distinct balls in box *X*. Let's look at the worst-case sequence of moves. Consider the case that 2 winning balls are from the same box. In such case answer is *min(S(X)+1)* over all boxes that contain at least one pair of same balls. Now we can leave only one ball of each color (apply unique operation to each box).

Let's transform worst-case sequence such that the answer wouldn't be worse. Let *LastB* be the box in such sequence.

1. Let *Q* be such moment of time when we firstly took some color that box *LastB* also has it. We can delete every choice of *LastB* before that moment.
2. We can choose only *LastB* after that moment.
3. Mathematical induction can be applied. So the sequence of moves are: make turns to some box, then switch the box only after the first moment when we get same color that has new box.
4. Such chain of boxes consists of 2 boxes.

So the solution is: iterate over the first box. Let C_i be the colors of balls in this box and D_i equals the minimal size of box that also contains C_i. Since we are anticipating the worst case scenario, sort all *D* in non-increasing order, i. e. $D_1 \geq D_2 \geq \ldots \geq D_k$. Then the answer (for the selected first box) can be found as the minimum of values $D_1 + 1$, $D_2 + 2$, ..., $D_k + k$.

J. Half is Good (Vitalii Herasymiv)

For each vertex, find the edge with the smallest weight that goes from that vertex. Any edge can occur at most twice, so it's enough for a half of spanning forest. It's correct because, since the spanning forest is unique, and starting a Prim's algorithm from any vertex would selected the smallest edge on the first iteration, each such edge is guaranteed to be in the minimum spanning forest.

K. Dance (Vitalii Herasymiv)

Let's say that each girl has already moved d to the left, and now each of them must stay on the same spot or move $2d$ to the right. Obviously, we have to group them in such a way that each group has only consecutive girls. Also note that the difference between the leftmost and the rightmost possible coordinates (after all girls move) cannot be greater than 200.

Now, let's say that the girls are not moving, and we have to cover some of the 200 coordinates in such a way that for each girl with position x, positions x or $x+2d$ are covered. Also, let's consider that consecutive covered positions are forming range, the size of which we will use for cost calculation (as $r-l$). Any consecutive pair of uncovered and covered points indicate a start of a range.

Now two cases are possible:

1. $2d \leq 20$: solve the problem using simple *dp[pos][mask]*, where *pos* is the current position, and *mask* is the state of last $2d$ cells (covered/uncovered). All necessary costs calculations and checks can be easily done with such state. The number of states is at most 200×2^{20}.
2. $2d > 20$: solve similar dp, but first off, cover positions 0, $2d$, $4d$, etc (not more that 10 of those), after that, cover 1, $2d+1$, $4d+1$, etc. So this is kind of a "broken profile" dp, and you also need to save the coverage of the first block (to match it with the last), so the total number of states is also 200×2^{20}.

L. Impress Her (Vitalii Herasymiv)

To avoid $O(n^4)$ running time, fix one bounding box first, and then iterate over all cells inside of it and check the corresponding pair of bounding boxes. Let's denote the sizes of connected components as s_1, s_2, ..., s_k, and $s_1 + s_2 + ... + s_k = n^2$. We can make a upper bound on the total area of bounding boxes: $ns_1 + ns_2 + ... + ns_k \leq n^3$, which proves that the running time of the algorithm is $O(n^3)$.