UIUC ICPC Spring Coding Contest 2019
Editorial

A - AK the Problems

This problem is not very hard but requires a small leap of faith or good game-theory intuition.

We denote the state of a forest using the parity of vertices and the parity of edges, respectively. Therefore, there are four cases we need to discuss. When the game stops, it is on $(even, even)$; thus, if we can prove that $(even, even)$ can't remain in $(even, even)$ after one move and the other states can always become $(even, even)$ after one move, we can know that $(even, even)$ is a $P - position$ and the others are $N - positions$. Here is the proof:

1. $(even, even)$:
   There isn't any move that can delete even vertices and even edges at the same time. Thus, $(even, even)$ can't remain in $(even, even)$ after one move.

2. $(even, odd)$:
   The parity of the edges is odd. Therefore, there exists at least one edge. We can delete the edge and the state becomes $(even, even)$.

3. (odd, odd):
   After any moves, a forest is still a forest. Since the state is (odd, odd), there exists at least a leaf (the vertex whose degree is 1) in the forest. We can delete this vertex and the state becomes $(even, even)$.

4. $(odd, even)$:
   The degrees sum formula in graph theory implies that the number of vertices with odd degree is even. Therefore, if the forest is in $(odd, even)$, the number of vertices with even degree is odd. That is to say, there exists at least a vertex with even degree. We can delete this vertex and the state becomes $(even, even)$.

In conclusion, since only $(even, even)$ is the $P - position$, if the intial forest is $(even, even)$, you should output "Suzukaze loses all his ratings!". Otherwise, you should output "Suzukaze becomes a grandmaster!".

B - Bigram Language Model

For each query (s, t), the nominator is the number of co-occurrences of (s, t) in the corpus, and the denominator is the number of occurrences of s as non-terminal word in a sentence. These values can be pre-processed by a single pass over the corpus.

The tricky part is that the number of distinct words appearing in the corpus w may be up to 10^5, so it is impossible to explicitly construct the co-occurrence matrix of size $w \times w$. The observation is that the sum of all elements in this matrix is bounded by the size of corpus, so we can use some sparse-matrix representation techniques (e.g. only recording which entries are non-zero and the values they hold).

C - Corns

Formally, we need to find the maximum X such that $X \leq W$, and $X = \sum_i p_i$. This seems like a straight forward knapsack implementation. But simply knapsack would require $O(nW)$ time, thus exceeding time limit.
One can notice that there is another constraint, $\sum_i p_i \leq 2 * 10^5$. This indicates means that the number of different p_i is small. Formally, since $1 + ... + T = O(T^2)$, number of distinct p_i is $O(\sqrt{2 * 10^5})$. Therefore, one can group those same items together, and perform multi-knapsack. The total runtime is $O(\sqrt{2 * 10^5} W logN)$(with small constant) by binary lifting idea, or $O(\sqrt{2 * 10^5} W$ by monotonic queue.

D - Diameter

One can notice that the farthest pair of points would never involve any point that is in the middle of a segment. Therefore, one can sort all the points on each of the segments, take the two endpoints and brute force the answer. Total runtime is $O(nlogn + k^2)$.
Another approach is to ignore the segmetns, and directly compute the answer with standard diameter algorithm, namely, computing the convex hull and do rotate caterpillars. The runtime for this is $O(nlogn)$(In fact $O(nlogk)$ if using output sensitive convex hull algorithms)

F - Fruit on the Tree

The core of this problem is asking you how many three vertices sets satisfy "any of the vertex in the set is not on the simple path of the other two vertices". Therefore, the weights do not matter. However, this number is hard to compute directly. One way to compute it is to subtract the number of sets satisfy "one of the vertex in the set is on the simple path of the other two vertices" from

the total number of sets to get the answer. We just need to DFS one time on the tree and compute how many ways there are to form a set that the current vertex is on the simple path of the other two vertices for each node, which is equal to:

$$\sum_{1 \le i < j \le m} |V_i||V_j|$$

where m is the number of subtrees if we use the current vertex to be the root and $|V_i|$ is the size of the ith subtree. This can be done in $O(n)$.

G - Greenberg Mass Comparison

The core of this problem is asking you how many ways are there to partition a set of n elements into disjoint, non-empty subsets. If you know Bell number, that's exactly what we are asking for (but you still have to do the programming). If not, not a problem! Let's work out a recurrence formula for it.

Let B_n denote the answer for n. Consider the number of elements **NOT** in the subset containing element 1. If there are k such elements, then there are $\binom{n-1}{k}$ ways to choose these elements, and we can just ignore the subset containing 1 and do the partition on these k elements. Therefore,

$$B_n = \sum_{k=0}^{n-1} \binom{n-1}{k} B_k$$

with base case $B_0 = 1$.

Notice that Bell numbers can get very large quickly, so coding this problem in Python or Java will save you a lot of trouble.

H - Hamiltonian Farm

The graph in the problem is a tournament. And every tournament contains at least a Hamiltonian path. We can prove this by proving that it is possible to add a vertex into an existing simple path to form a new simple path, which is like the insertion sort and we will leave the proof as an exercise for you.

However, the complexity of the insertion sort is $O(n^2)$ and you will get Time Limit Exceeded. A quick observation is that we can find a pivot vertex and divide the graph into two parts, one part with vertices having directed edge to the pivot vertex and the other part with vertices having directed edge from the pivot vertex, and sort these two parts recursively. This results in an expected complexity of $O(nlogn)$. If you choose your pivot **randomly**, you can pass the problem.

A safer solution is using mergesort, in which you need to find a way to combine two simple paths into one. We will leave the method as an exercise for you again. Mergesort results in $O(nlogn)$, which can also pass the problem.

A notable thing is that to find the direction of an edge, you need around $log10^{15}$ computations. Therefore, the time limit for this problem is quite strict; you should implement a concise solution to pass the problem.

I - Innovate

This check-in problem is all about string manipulation. The easiest way to solve the problem is to to split each input sentence by whitespace, then iterate from $i = 1..maxLen$ - where $maxLen$ represents the maximum number of words in any sentence - and align words at every i'th index using whitespace for padding. As long as you're careful with how you insert spaces for padding, this problem should be straightforward.

J - Juicy

Lets first forget about the apple trees. The problem is asking us to find a spanning tree on the graph, with the constraint that the ratio of the sum of happiness to the sum of energy consumed is maximum. This is known as optimal ratio spanning tree. A solution to solve this is to binary search on the optimal ratio, replace the edge weights to $h - e * ratio$, then search for the minimum spanning tree. If this tree omits a non negative sum of weight, then there is a larger possible ratio, otherwise the ratio is too large and we need to lower it. Since the lowest possible difference of ratio is $\frac{1}{10^8-1} - \frac{1}{10^8}$, one has to be precise to at least $5 * 10^{-9}$. Adding the apple trees doesn't make a big difference. We are searching for a minimum steiner tree instead of a spanning tree. This can be done by brute force enumerating which apple trees to include.

K - Koolhash

Another check-in problem, Koolhash tests you're ability to bitshift. For the leftmost bit, the easiest way to determine the value is to 'and' each record with 1 shifted left by 31 bits. One can also note that with the input restriction of 2^{31}, this bit will always be zero. Can you see why?

For the rightmost bit, we can just 'and' each record with 1. Finally, for the number of 1-bits, we just need to check all the bits in the record, and record how many times we see one. This can be done by repeatedly shifting right and determining the leftmost bit through the procedure described above.