Potemkin cycle

A simple solution (which should score about 60 points) is as follows: For each vertex v and for all neighbors x and y of v such that x and y are non-adjacent, we test whether there exists a cycle with the required properties containing xvy as a subpath. To do so, we first remove v and all its neighbors distinct from x and y, and if x and y are in the same component of the resulting graph G', then xvy together with a shortest path between x and y in G' forms the sought cycle. The time complexity of this solution is $O(NR^2)$.

To improve this solution, let us rephrase it a bit. Consider a vertex v, let S be the set of its neighbors, and let G_1, ..., G_k be the components of $G - v - S$. There exists a cycle with the required properties that contains v if and only if for some i, G_i has two non-adjacent neighbors in S; in that case, we proceed as before. Otherwise, the neighborhood of each of G_1, ..., G_k in S is a clique, and it is easy to see that the sought cycle cannot pass through a clique. Hence, the sought cycle is contained either in the subgraph of G induced by S, or in the subgraph of G induced by G_i and its neighbors in S for some i. Thus, we solve the problem by recursively applying the algorithm to these subgraphs. Since the subgraphs may overlap in some vertices, the analysis of the time complexity is not entirely straightforward, but (with a good implementation) it turns out to be $O(NR)$.

Calvinball championship

Firstly, observe that the valid descriptions of partitions of the players to teams are exactly the sequences such that for every k, the numbers $1, 2, \ldots, k-1$ appear before k in the sequence. Suppose that we are given an initial segment x_1, ..., x_m of a valid sequence, with $\max(x_1, \ldots, x_m) = t$, and we want to know in how many ways we can extend this sequence by adding n more elements. Clearly, this number depends only on n and t, and we will denote it by $s(n, t)$. Clearly, $s(0, t) = 1$ for every t. If $n > 0$, we can continue the sequence by adding one of the numbers 1, ..., t and extending it by $n - 1$ further elements in $s(n - 1, t)$ ways, or by adding $t + 1$ and extending it in $s(n - 1, t + 1)$ ways; hence, $s(n, t) = ts(n - 1, t) + s(n - 1, t + 1)$. Thus, we can compute these numbers by dynamic programming, starting with $s(0, \star)$, then $s(1, \star)$, ...

How many sequences are there in the lexicographic ordering before the sequence $Y = y_1, \ldots, y_m$? Each such sequence B matches Y at first, say in b elements, and then its $(b+1)$-th element is one of 1, ..., $y_{b+1} - 1$, and then it is arbitrary. Therefore, for a given b, we can choose the $(b + 1)$-th element in $y_{b+1} - 1$ ways, and then the rest of the sequence in $s(m - b - 1, \max(y_1, \ldots, y_b))$ ways. It follows that the total number of sequences before Y is

$$\sum_{b=0}^{m-1} (y_{b+1} - 1) s(m - b - 1, \max(y_1, \ldots, y_b)).$$

Given the values of $s(\star, \star)$, this can be easily computed.

Also by computing the sum for b from $m - 1$ downwards till 0, we can simultaneously compute the values of $s(0, \star)$, $s(1, \star)$, ... as needed, thus keeping the memory complexity linear (without this improvement, memory limit is exceeded for the largest inputs).

Pipes

The task is to find bridges in a graph with n vertices and $m \gg n$ edges, while only using $O(n)$ memory. There are several possible solutions, let us outline one of them.

We read the edges one by one, and by one instance of DFU data structure, we keep track of the connected components of the graph read so far. In the second DFU data structure, we will keep track about sets of vertices which are guaranteed to be 2-edge-connected to each other (but this is only a conservative information, vertices in different sets may be 2-edge-connected to each other as well). The process is as follows: Suppose we read an edge uv.

- If u and v are in different components, we join the corresponding sets in the first DFU structure.
- If u and v are in different sets in the second DFU, we join the corresponding sets.
- Otherwise, we throw uv away.

In this way, we have thrown away all but at most $2n - 2$ edges X (for that we joined sets in one of the DFU structures), and it is easy to see that the bridges of the original graph and of the graph with the edge set X are the same. The bridges in the latter graph can be found using the standard linear-time algorithm.