Evidence Board

General note: Let's rephrase the problem. Instead of "building" the tree from scratch, we will do the opposite and remove edges from the tree. After that, we will reverse our obtained answer. In the tutorial we will solve the problem in the format of "removing the tree". The list of stickers in the tree vertex will be called a stack, and the top of the stack will always be the sticker that should be removed first.

- **Subtask №1**

  This subtask is solved by full bruteforce. There are $(n-1)!$ possible orders of edge removal. Each order is trivially checked in $\mathcal{O}(n)$.

  Asymptotics: $\mathcal{O}(n!)$

- **Subtask №2**

  Bamboo.

  Dynamic programming on the prefix. $dp[v][i]$ a boolean value: whether it is possible to remove the bamboo from the vertices from 1 to v, but if the stack for vertex v contains a single element $c_{v,i}$ ($i \in \{0,1\}$).

  Dynamic programming base: $dp[1][0] = dp[1][1] = true$

  Recalculation of dynamic programming: we need to consider all four variants with which elements of the stacks the edge between v and $v-1$ will be removed, if this edge can be removed, we recalculate through $dp[v-1][i]$.

  The answer exists if $dp[n][0] = true$, otherwise no.

  To restore the answer, for each edge, we need to remember with which i_1, i_2 it was removed. First remove all edges of the form $i_1 = 0, i_2 = 0$, then $i_1 = 1, i_2 = 0$, $i_1 = 0, i_2 = 1$ and $i_1 = 1, i_2 = 1$.

  Asymptotics: $\mathcal{O}(n)$

- **Subtask №3**

  Star.

  Go through the leaves from 2 to n. Let's say we are currently considering leaf v. In the array c_1, find the minimum number $\geq w_v - c_{v,1}$. It is easy to understand that it is optimal to remove the edge leading to v with this number. After that, remove the found number from c_1. In the implementation, we maintain c_1 in std::multiset and use the lower_bound method.

  The required order of edge removal corresponds to the original order of numbers in c_1.

  Asymptotics: $\mathcal{O}(n \log n)$

- **Subtask №4**

  Two stars, bases connected by an edge.

  Solve two stars similarly to group 4. After that, there will be one number left in both bases. If they allow removing the edge between the bases, the answer is Yes, otherwise No. The suitable order of edge removal: all edges of the first star before the base vertex, all edges of the second star before the base vertex, the edge between the bases, all edges of the first star after the base vertex, all edges of the second star after the base vertex.

  Asymptotics: $\mathcal{O}(n \log n)$

- **Subtask №5**

  The values of the numbers in the vertices are increasing.

  If at some point an edge can be removed, then it can always be removed later. Therefore, at any time, it is permissible to remove any possible edge. The solution will be a greedy algorithm: we look for any edge that can be removed and remove it from the tree, repeating $n-1$ times. If at some point there are no edges ready for removal, the answer is No.

Asymptotics: $\mathcal{O}(n^2)$

- **Subtask №6**

  Note that for any test, the following fact is true: if we reverse all the stacks of the original test and call it a *new* test, then if there is no answer for the original test, there is no answer for the new one. And if the answer exists, then the answer to the *new* test will be the reversed answer for the original test. From this observation, the solution of group 6 directly follows from the solution of group 5: we need to reverse all the arrays, as well as the final answer.

  Asymptotics: $\mathcal{O}(n^2)$

- **Full solution**

  The full solution is based on one key idea: "For any way to assign a pair of stack elements to each edge of adjacent vertices (without repetitions), there exists an edge removal order such that each edge will be removed together with its assigned elements".

  Proof of the key idea: Consider any assignment. Each vertex points to one adjacent edge to which the top element of the stack is assigned, as "half ready for removal". Since there are n vertices and $n-1$ edges, there will be an edge to which both adjacent vertices point. This edge can be removed. After removing it, the tree splits into two other trees, and the same reasoning applies to them.

  After the key idea, the problem split into two parts.

  1. Build any correct assignment of edges to stack elements
  2. Restore the order of edge removal

  To build the assignment, we use a greedy algorithm similar to the solution of group 3. We write down the order of vertices of any tree traversal. We iterate through the vertices in the reverse order of traversal (from the leaves). Let the currently iterated vertex be v. Then we assign the minimum suitable number ($\geq w_v - c_v$) from the parent vertex to the only remaining number in the stack of vertex v.

  For the implementation, we store a `std::multiset` of numbers in each vertex. We use `lower_bound` and `erase` in it.

  Restoration: for each edge, we maintain how ready it is for removal (0/1/2). The edges ready for removal are stored in the stack. Iteratively, we take any edge from the stack, remove it, and increase the "readiness for removal" of no more than 2 other edges. If any of them becomes ready for removal, we add it to the stack.

  Final asymptotics: $\mathcal{O}(n \log n)$