A BFS = DFS?

Time limit: 2s

Description

You have a graph G with $\boldsymbol{N}$ nodes and some bidirectional edges. The edge connecting node u and node v is denoted by edge (u, v). This graph is connected, which means each node can reach all other nodes using one or more edges. This graph is simple, which means all edges connect two distinct nodes and each pair of nodes is connected by at most one edge. This graph is bipartite, which means you can partition the nodes into two sets, such that each edge connects two nodes from different sets.

You want to get an ordering of nodes from your graph, so you give the graph G to an artificial intelligence (AI) agent. The AI agent uses two tools: the first tool produces an ordering of nodes using BFS algorithm, while the second tool produces an ordering of nodes using DFS algorithm.

BFS algorithm on a simple connected graph with $\boldsymbol{N}$ nodes produces an ordering of nodes as follows:

```
unexplored := set of integers containing 2 to N
queue := empty set
insert 1 to queue

while queue is not empty:
    u := element in queue inserted to queue the earliest
    remove u from queue
    print node u
    for each edge (u, v) in increasing order of v:
        // iterate all neighbours of node u in increasing order
        if v is in unexplored:
            insert v to queue
            remove v from unexplored
```

DFS algorithm on a simple connected graph with $\boldsymbol{N}$ nodes produces an ordering of nodes as follows:

```
unexplored := set of integers containing 1 to N

procedure DFS(u):
    if u is in unexplored:
        remove u from unexplored
        print node u
        for each edge (u, v) in increasing order of v:
            // iterate all neighbours of node u in increasing order
```


```
                DFS(v)

DFS(1)
```

Note that all neighbours of node u are iterated in increasing order.

The AI agent tells you that BFS and DFS algorithms actually produce the same ordering of nodes: $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_{\boldsymbol{N}}$. Now you are wondering, how many simple connected bipartite graphs with $\boldsymbol{N}$ nodes satisfy the mentioned condition. Two graphs P and Q are defined to be distinct if there is a pair of nodes such that:

- there is an edge connecting the pair of nodes in graph P, but
- there is no edge connecting the pair of nodes in graph Q.

Since the number of satisfying graphs can be large, print the remainder of the number of satisfying graphs divided by 998 244 353.

Input

The first line contains an integer $\boldsymbol{N}$ ($1 \leq \boldsymbol{N} \leq 200\,000$). The second line contains $\boldsymbol{N}$ integers $\boldsymbol{A}_1, \boldsymbol{A}_2, \ldots, \boldsymbol{A}_{\boldsymbol{N}}$ ($1 \leq \boldsymbol{A}_i \leq \boldsymbol{N}$; $\boldsymbol{A}_1 = 1$; $\boldsymbol{A}_i \neq \boldsymbol{A}_j$ for each $1 \leq i < j \leq \boldsymbol{N}$) separated by spaces.

Output

The first line contains the remainder of the number of satisfying graphs divided by 998 244 353.

Sample Input 1	Sample Output 1
4 1 3 2 4	2

Explanation of samples

In sample input 1, 2 satisfying graphs are illustrated by figure A.1.

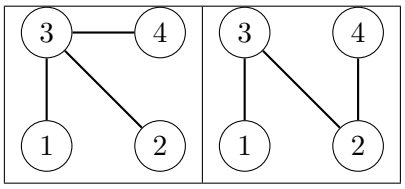

Figure A.1: 2 satisfying graphs in sample input 1.