CRACK A HACK

Sherlock's Array Merging Algorithm

SARVESHKUMAR

01FE15BCS173

Problem statement:-

Watson gave Sherlock a collection of arrays V. Here each V_i is an array of variable length. It is guaranteed that if you merge the arrays into one single array, you'll get an array, M, of n distinct integers in the range $[1, n]$.

Watson asks Sherlock to merge V into a sorted array. Sherlock is new to coding, but he accepts the challenge and writes the following algorithm:

- $M \leftarrow [\,]$ (an empty array).
- $k \leftarrow$ number of arrays in the collection V.
- While there is at least one non-empty array in V:
  - $T \leftarrow [\,]$ (an empty array) and $i \leftarrow 1$.
  - While $i \leq k$:
    - If V_i is not empty:
      - Remove the first element of V_i and push it to T.
    - $i \leftarrow i + 1$.
  - While T is not empty:
    - Remove the minimum element of T and push it to M.
- Return M as the *output*.

Explanation:-

We need to use Dynamic Programming. First, we define $\text{cnt}(i)$ as count of rows with $\text{length} \geq i$. For example, rows can be like this:

```
    Col 1 2 3
Row     _ _ _
        _ _
        _ _ _
        _
```

So, $\text{cnt}(3) = 2$ and $\text{cnt}(2) = 3$ and $\text{cnt}(1) = 4$. Note, that $\text{cnt}(i) \geq \text{cnt}(i+1)\ \forall i$.

Now, we will instead of building solution column wise, we store current $\text{cnt}(i)$ and also how many elements in array A left can be used.
$f(i, j)$ is number of distinct matrices possible using elements $A_1, A_2, \ldots, A_i$ if $\text{cnt}(\text{last column}) = j$. Our answer will be $\sum_{i=1}^{n} f(N-1, i)$.

Let's see how we define recurrences. We want to build a column using j values $A_{i-j+1}, A_{i-j}, \ldots, A_i$. If this sequence is increasing(because matrices is column wise sorted) then we can choose these j elements to form a column, else there are 0 number of ways.

Now, if we have used j elements, state i, gets reduced to $i - j$. Also, for the next colmumn c we are going to build $\text{cnt}(c) \geq j$.

```
//returns f(i, j)
def rec(i, j):
    ret=0;

    for k = j; k <= i - j + 1; k++
        if(A[i - j + 1, i] is increasing){
            //now k is cnt(c), where c is the previous column
            //now, out of these k positions we can choose any j positions
            //and put any of the value from A[i - j + 1, i]
            //so we multiply number of ways by
            //choose(k, j) * factorial(j)
            int nextways = rec(i-j, k) * choose(k, j) * factorial(j);
            ret += nextways;
        }
    }

    return ret;
```

This is current $O(N^3)$ solution. I couldn't think of anything better than this. But we should note the small constant factor helps us set constraints $N = 10^3$. If you calculate it would turn out to be $O(\frac{N^3}{12})$.

CODE:-

```
import java.io.OutputStream;
import java.io.IOException;
import java.io.InputStream;
import java.io.OutputStream;
import java.io.PrintWriter;
import java.util.Arrays;
import java.io.BufferedWriter;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.StringTokenizer;
import java.io.Writer;
import java.io.OutputStreamWriter;
import java.io.BufferedReader;
import java.io.InputStream;


/**
 * Built using CHelper plug-in
 * Actual solution is at the top
 */
public class Solution {
    public static void main(String[] args) {
        InputStream inputStream = System.in;
        OutputStream outputStream = System.out;
        InputReader in = new InputReader(inputStream);
        OutputWriter out = new OutputWriter(outputStream);
        SherlocksArrayMergingAlgorithm solver = new SherlocksArrayMergingAlgorithm();
```

```
    solver.solve(1, in, out);
    out.close();
}

static class SherlocksArrayMergingAlgorithm {
    public int mod = 1000000007;
    public boolean[][] inc;
    public long[] fact;
    public long[] ifact;
    public int n;
    public long[][] dp;

    public void solve(int testNumber, InputReader in, OutputWriter out) {
        n = in.nextInt();
        long[][] w = Factorials.getFIF(2 * n, mod);
        fact = w[0];
        ifact = w[1];
        int[] arr = in.readIntArray(n);
        inc = new boolean[n][n];
        for (int i = 0; i < n; i++) {
            inc[i][i] = true;
            for (int j = i + 1; j < n; j++) {
                inc[i][j] = inc[i][j - 1] && (arr[j] > arr[j - 1]);
            }
        }
        dp = new long[n][n];
        for (long[] x : dp) Arrays.fill(x, -1);
```

```
        long ans = 0;
        for (int i = 0; i < n && inc[0][i]; i++) {
            ans = (ans + dfs(i + 1, i + 1)) % mod;
        }
        out.println(ans);
    }

    public long dfs(int curidx, int lastblock) {
        if (curidx == n) return 1;
        if (dp[curidx][lastblock] != -1) return dp[curidx][lastblock];
        long ret = 0;
        for (int next = curidx; next < n && next < curidx + lastblock && inc[curidx][next]; next++) {
            int cblock = (next - curidx + 1);
            ret = (ret + fact[lastblock] * ifact[lastblock - cblock] % mod * dfs(next + 1, cblock)) % mod;
        }
        return dp[curidx][lastblock] = ret;
    }

}

static class InputReader {
    public BufferedReader reader;
    public StringTokenizer tokenizer;

    public InputReader(InputStream stream) {
        reader = new BufferedReader(new InputStreamReader(stream), 32768);
        tokenizer = null;
```

```
    }

    public String next() {
        while (tokenizer == null || !tokenizer.hasMoreTokens()) {
            try {
                tokenizer = new StringTokenizer(reader.readLine());
            } catch (IOException e) {
                throw new RuntimeException(e);
            }
        }
        return tokenizer.nextToken();
    }

    public int[] readIntArray(int tokens) {
        int[] ret = new int[tokens];
        for (int i = 0; i < tokens; i++) {
            ret[i] = nextInt();
        }
        return ret;
    }

    public int nextInt() {
        return Integer.parseInt(next());
    }

}
```

```
static class OutputWriter {
    private final PrintWriter writer;

    public OutputWriter(OutputStream outputStream) {
        writer = new PrintWriter(new BufferedWriter(new OutputStreamWriter(outputStream)));
    }

    public OutputWriter(Writer writer) {
        this.writer = new PrintWriter(writer);
    }

    public void close() {
        writer.dose();
    }

    public void println(long i) {
        writer.println(i);
    }

}

static class Factorials {
    public static long[][] getFIF(int max, int mod) {
        long[] fact = new long[max];
        long[] ifact = new long[max];
        long[] inv = new long[max];
        inv[1] = 1;
```

```
        for (int i = 2; i < max; i++) {
            inv[i] = (mod - mod / i) * inv[mod % i] % mod;
        }
        fact[0] = 1;
        ifact[0] = 1;
        for (int i = 1; i < max; i++) {
            fact[i] = fact[i - 1] * i % mod;
            ifact[i] = ifact[i - 1] * inv[i] % mod;
        }
        return new long[][]{fact, ifact, inv};
    }


  }
}
```

REFENCES:

Question :- https://www.hackerrank.com/contests/university-codesprint-2/challenges/sherlocks-array-merging-algorithm

Editorial link :- https://www.hackerrank.com/contests/university-codesprint-2/challenges/sherlocks-array-merging-algorithm/editorial