Online Contests Solutions

Saman Saadi

Contents

Chapter 1

HackerRank

1.1 New Year Chaos

You can find the question in this link.

We define $index_i$ as the current index for person i. For example if we have $1, 2, 3, 4$ and 4 bribes 3, the queue looks like $1, 2, 4, 3$. So $index_4 = 3$. Since no body can bribe more than 2 times, $index_i \geq i - 2$ for $1 \leq i \leq n$. Consider person n. No body can bribe that person. So $n - 2 \leq index_n \leq n$. After we retruned that person to his actual place we can consider $n - 1$. So we have $n - 3 \leq index_{n-1} \leq n - 1$ (note that at this moment $index_n = n$).

```
void minimumBribes(vector<int> q) {

    const auto& n = q.size();
    int res = 0;
    for (int num = n; num > 0; --num)
    {
        for (int i = max(0, num - 3); i < num - 1; ++i)
        {
            if (q[i] == num)
            {
                ++res;
                swap(q[i], q[i + 1]);
            }
        }
        if (q[num - 1] != num)
        {
            cout << "Too chaotic" << endl;
            return;
        }
    }
    cout << res << endl;
```

```
}
```

1.2 Minimum Swaps 2

See the problem statement in this link.

Note that this solution is based on Selection Sort in which the number of swaps are minimum. According to Wikipedia: "One thing which distinguishes selection sort from other sorting algorithms is that it makes the minimum possible number of swaps, n − 1 in the worst case.". Altourh Selection sort has minimum number of swaps among all sorts agorithms, it has $O(n^2)$ comparisons. Since the final result is $\{1, 2, \ldots, n\}$, it's like we have the set in sorted order so we can bypass comparisons and use Selection Sort advantage which is the minimum number of swaps.

We define $index_i$ as the current index of number i. Suppose we have n numbers, so $1 \le index_i \le n$. The goal is to have $index_i = i$. Without losing generality suppose $i < j \land index_i = j$. There are two cases to consider:

1. If $index_j = i$, then by swapping arr_i and arr_j, we put both i and j in their corresponding positions.

2. If $index_j = k \land k \neq i \land k \neq j$. In this case by swapping arr_i and arr_j we only put i in its corresponding position. So we need to do an extra swap to put j in its correct position.

We can start from $i = 1$ to $i = n$ and make sure i is in correct position; otherwise we perform a swap. In each iteration we fix the position of one or two numbers. A good example is {4, 3, 2, 1}.

```
int minimumSwaps(vector<int> arr) {

const auto& n = arr.size();
vector<int> index(n + 1);

for (int i = 0; i < n; ++i)
    index[arr[i]] = i;
int cnt = 0;
for (int num = 1; num <= n; ++num)
{
    if (index[num] != num - 1)
    {
        ++cnt;
        index[arr[num - 1]] = index[num];
        swap(arr[index[num]], arr[num - 1]);
        index[num] = num - 1;
    }
}
```

```
    return cnt;
}
```

1.3 Count Triplets

Problem statement.

We use dynamic programming to solve it. For mathematical induciton we define $cnt[num][n]$ like this:

$$cnt[a_{i_1}][0] = |\{a_{i_0} \in arr \mid a_{i_1} = a_{i_0} \times r \ \wedge \ i_1 < i_2\}|$$
$$cnt[a_{i_2}][1] = |\{(a_{i_0}, a_{i_1}) \in arr \times arr \mid a_{i_k} = a_{i_{k-1}} \times r \ \wedge \ i_{k-1} < i_k \text{ for } 1 \leq k \leq 2\}|$$

So the final answer is:

$$\sum_{n \in arr} cnt[n][1]$$

Then for each number n we have

$$cnt[n \times r][0] = cnt[n \times r][0] + 1$$
$$cnt[n \times r][1] = cnt[n \times r][1] + cnt[n][0]$$

Since $r = 1$, the order of assignments are very important.

```
long countTriplets(vector<long> arr, long r) {
    const auto n = arr.size();
    unordered_map<long, array<long, 2>> cnt;
    //cnt[a[j]][0] = |{a[i]}| in which i < j and a[j] = a[i] * r
    //cnt[a[k]][1] = |{a[i], a[j]}| in which i < j < k and
    //a[k] = a[j] * r and a[j] = a[i] * r

    long res = 0;
    for (const auto& num : arr)
    {
        res += cnt[num][1];
        const auto next = num * r;
        cnt[next][1] += cnt[num][0];
        ++cnt[next][0];
    }
    return res;
}
```