ICH Christmas 2020 — Solutions

ICH Administration

December 25-28, 2020

Contents

§1 Shoe Selection

This is equivalent to computing

$$\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_i \cdot a_j = a_1 \cdot a_2 + a_1 \cdot a_3 + \ldots a_1 \cdot a_n + a_2 \cdot a_3 + a_2 \cdot a_4 + \ldots a_2 \cdot a_n + \ldots a_{n-1} \cdot a_n$$

However, doing this directly is too slow for full credit.

Notice that

$$(a_1 + a_2 + a_3 + \ldots a_n)^2 = 2\left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_i \cdot a_j\right) + \left(a_1^2 + a_2^2 + \ldots a_n^2\right)$$

$$\implies \left(\sum_{i=1}^{n}\sum_{j=i+1}^{n} a_i \cdot a_j\right) = \frac{(a_1 + a_2 + a_3 + \ldots a_n)^2 - \left(a_1^2 + a_2^2 + \ldots a_n^2\right)}{2}$$

We can find this in $O(n)$ time, which is fast enough.

```
n = int(input())
a = [int(x) for x in input().split()]
squares = 0
for x in a:
  squares += x**2
print((sum(a)**2-squares)//2)
```

§2 Rectangles

The key observation is that the number of distinct colors on the plane is equal to number the rectangles that are not completely engulfed. For each rectangle, see whether it is engulfed inside another rectangle. If not, increase the answer.

```
#include <bits/stdc++.h>
using namespace std;

struct rectangle { int x1, y1, x2, y2; };

int n; rectangle all[10000];

bool iINj (rectangle i, rectangle j) {
    return (i.x1>=j.x1 && i.y1>=j.y1 && i.x2<=j.x2 && i.y2<=j.y2);
}

int main() {
    cin >> n;
    for (int i = 0; i < n; i++)
        cin >> all[i].x1 >> all[i].y1 >> all[i].x2 >> all[i].y2;
    int ans = 0;
    for (int i = 0; i < n; i++) {
        bool distinct = true;
        for (int j = 0; j < n; j++)
            if (j != i && iINj (all[i], all[j])) distinct = false;
        ans += distinct;
    }
    cout << ans << endl;
    return 0;
}
```

Time complexity: $O(N^2)$

§3 Maximum Segment

This is a variant of the Maximum Sum Subarray problem, where the wrapping of the array needs to be handled. Ignoring wrapping, the maximum segment ending at position *pos* is

$$\sum_{i=0}^{pos} a_i - min(0, min\left(\sum_{j=0}^{x} a_j\right) \text{ for } 0 \leq x < pos)$$

To take into account wrapping, the new a can have size $2N$ and both halves of the new a can be the original a. Then we are looking for the maximum segment of length $\leq N$ in the new a. This can be done using the aforementioned insight as well as a `std::multiset` to keep track of the valid prefixes and their minimum.

```
#include <bits/stdc++.h>
using namespace std;

int a[200000], p[200000], n, ans = -INT_MAX;

int main() {
    cin >> n;
    for(int i = 0; i < 2 * n; i++) {
        if(i < n) cin >> a[i];
        else a[i] = a[i - n];
        p[i] = a[i] + p[max(0, i - 1)];
    }
    multiset<int> ms;
    ms.insert(0);
    for(int i = 0; i < 2 * n; i++) {
        if(i + 1 > n) ms.erase(ms.find(p[i-n]));
        ans = max(ans, p[i] - *ms.begin());
        ms.insert(p[i]);
        if(!i) ms.erase(ms.find(0));
    }
    cout << ans << endl;
    return 0;
}
```

Time complexity: $O(N \log N)$

§4 Farm Surveillance

Call a *zone* a collection of pastures which are monitored by any group of cows of size ≥ 1. The goal then is to make all zones contiguous while maximizing the number of zones. Call the *range* of a zone the segment between the first and last occurrence of a pasture monitored by that zone.

If within the range of a non-contiguous zone, there exists exactly one section of another zone, Farmer John must pair up exactly two cows to merge the two zones, making the resultant zone contiguous. In general, for every instance where the range of a zone overlaps with the range of another zone, Farmer John must pair two cows — merging the two zones into one.

What remains is to construct the ranges (segments) and count the number of required merges. This can be done by sweeping through the segments in ascending order, incrementing the answer whenever the current segment is overlapped with another.

```
#include <bits/stdc++.h>
using namespace std;

int n, m, p[100001], STARTS[100001], ENDS[100001];

int main() {
    cin >> n >> m;
    for(int i = 1; i <= n; i++) cin >> p[i];
    for(int i = 1; i <= n; i++) {
        if(STARTS[p[i]] == 0) STARTS[p[i]] = i;
        ENDS[p[i]] = i;
    }
    vector<pair<int, int>> s;
    for(int i = 1; i <= m; i++) {
        if(STARTS[i] != 0) {
            s.push_back(pair<int, int>(STARTS[i], -1));
            s.push_back(pair<int, int>(ENDS[i], 1));
        }
    }
    sort(s.begin(), s.end());
    int curcount = 0;
    int ans = 0;
    for(pair<int, int> i : s) {
        ans += (i.second == -1 && curcount > 0);
        curcount += -i.second;
    }
    cout << ans << endl;
    return 0;
}
```

Time complexity: $O(N \log N)$
Similar problem: https://codeforces.com/contest/1253/problem/D