Belgian Olympiad In Informatics

Training weekend 6/2/2015-8/2/2015

Jorik Jooken

Introduction

- Let the students introduce themselves (name, known programming language(s), experience with programming contests/olympiads in general…).

- Explain what competitive programming is like. Examples of difference between competitive programming and industry programming: use of global variables, pick the simplest (not necessarily fastest) solution, limited set of known algorithms, no research problems (all the problems given must have been solved by at least the author of the problem) …

- Important tool to measure running time of an algorithm: time complexity. Explain + examples + table of common running times

- What algorithms are they expected to know? The most recent version (2013) of the IOI syllabus: http://www.ioinformatics.org/a_d_m/isc/iscdocuments/ioi-syllabus.pdf

- Theory is important, but in a programming contest, there will only be exercises. Mastering the theory is important to be able to get a good score in a programming contest, but one has to do many exercises in order to get really good at programming contests.

- Golden rule: there is no easy way to achieve this. The only rule is: practice, practice and practice.

- Sites to practice on: 1) www.codeforces.com 2) www.topcoder.com 3) www.usaco.org 4) www.uva.onlinejudge.org 5) www.hackerrank.com 6) www.codechef.com (7) www.projecteuler.net)

General tips and tricks

- Collection of random advice for the contestants.

- Always beware of corner cases. You wouldn't believe how many solutions fail because of this.

- Practice regularly! I know this seems very obvious, but just do it. If you only read about how to solve problems and even if you understand a solution, that doesn't mean that you are able to implement a solution. Try implementing your ideas.

- In a contest, it is very important to stay calm at all times. You can think much better when you're only worrying about solving a problem rather than worrying about your score.

- In the first few contests (if you're new to competitive programming) you might get unexpected bad scores. Don't be demotivated by this, it's normal. Competitive programming is one of those things that you get really good at by doing it a lot. It's useless to compare yourself with people who have much more experience than you do.

- Think about time management during a contest. If you're stuck on a problem, and you really can't find the solution within a reasonable amount of time, then set your pride aside and just skip this particular problem.

- Especially useful for BeOI/IOI: the subtasks can really differ a lot from the actual problem! Read the subtasks carefully and maybe you will score some points for solving this particular subtask even though you don't know the correct algorithm for the general problem.

- People often worry a lot about coding speed during a competition. Coding speed is an important thing, but it shouldn't be your main focus! It's better to work a bit slower, but careful. Programming is one of those things were you cannot afford a single mistake. If you solve 9/10 cases, in most programming competitions, you will get a score of 0 points.

- Consider buying the book Competitive Programming 3 or try reading the first version on the internet (completely free)

- Most important remark of all: enjoy problem solving! You advance most if you enjoy what you're doing.

Advanced exercises

NOTE: these exercises might involve techniques that are not being described in the above text. They are just some challenging exercises in case someone might be done early or find the other exercises too easy. Feel free to ask questions!

There will be hints provided in the section that follows.

1) http://codeforces.com/contest/504/problem/B
2) http://codeforces.com/problemset/problem/493/D
3) http://codeforces.com/problemset/problem/492/D
4) http://codeforces.com/problemset/problem/474/E
5) http://codeforces.com/problemset/problem/459/D
6) http://codeforces.com/problemset/problem/431/D
7) https://www.hackerrank.com/challenges/sherlock-and-queries
8) https://www.hackerrank.com/challenges/pairs
9) http://www.spoj.com/problems/INVCNT/
10) Given a convex polygon with n sides. (4 <= n <= 1000). Now you want to form a quadrilateral such that every vertex of the quadrilateral is also a vertex from the convex polygon that is given. Find such quadrilateral that has the maximum area. The area of a quadrilateral with points (a1, b1), (a2, b2), (a3, b3), (a4, b4) is given by: 0.5*abs((a1*b2)+(a2*b3)+(a3*b4)+(a4*b1)-(a1*b4)-(a2*b1)-(a3*b2)-(a4*b3))

Hints for the advanced exercises

1) At first sight, it doesn't look like it, but you need to know segment tree/binary indexed tree to be able to solve this problem efficiently.
2) Try to find an invariant. For example, what happens if you always move right? The solution can be proved by induction once you find it.
3) Binary search the answer
4) You will need to use segment tree+rmq
5) You will need to compress the data to a smaller range and then use a segment tree/fenwick tree to solve the problem.
6) Binary search the answer
7) The complexity of the solution is O(N*(1/N+2/N+3/N+…+N/N)). This is roughly equal to O(N*log(N)). Does this bring you to any ideas?
8) Sort+binary search
9) This problem can be solved in a lot of ways. One way is to think of merge sort: how can we adapt this algorithm in order to get the correct answer? Another way is to use segment tree/binary indexed tree
10) Trying all the possible combinations of the 4 points will give a Time Limit Exceeded (TLE). However, you can iterate through all combinations of 2 points (these points are the opposite points of the quadrilateral). Now that these 2 points are fixed: you can binary search for the other 2 points! Solution complexity: O(N^2*log(N))