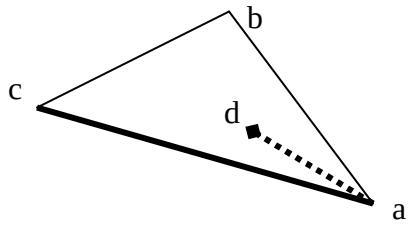
IMPA Omgång 5 – 2010-10-06

361 – Cops and Robbers

Geometri. Testa alla kombinationer av hörnpunkter som bildar trianglar.

Hur ser man att en punkt är inuti en triangel? Jo, vandra runt längs kanten och punkten ska antingen alltid vara till höger eller alltid till vänster. Det kontrolleras med kryssprodukt. Kontrollera om kryssprodukten, $(c_x - a_x)(d_y - a_y) - (c_y - a_y)(d_x - a_x)$, är positiv eller negativ. Detta kan göras med heltal, 1001*1001 < 2^{31}, vilket innebär att man inte har problem med jämförelser vid gränsfall. Glöm inte att göra specialjämförelser om triangelns punkter ligger på en linje eller om flera av dem sammanfaller.



10594 – Data Flow

Minimumkostnadsflöde där tid är pengar.

Kostnaden för att skicka paket är tiden det tar. Implementera en vanlig maximalflödesalgoritm som Edmonds-Karp eller Ford–Fulkerson. Modifiera sedan DFS- alternativt BFS-sökningen till att vara billigaste kvarvarande vägen med avseende på tiden att nå målet. Klart!
http://www.csc.kth.se/contest/ioi/ioitraning/flow.htm

10290 – {sum+=i++} to Reach N

Talteori, svårt på grund av hård tidsgräns.
$N = \sum_{n=a}^{b} n = \frac{(b-a+1)(a+b)}{2} = \frac{k(2a+k-1)}{2}$ där k är antalet tal att summera. Notera att om k är udda så är $2a+k-1$ jämn och tvärtom. Kan vi bara dela *N* med ett udda tal så kan vi skapa en summa. Alltså, primtalsfaktorisera.

Faktorisering görs enklast genom att skapa en tabell över primtal, som genereras i början, och gå igenom den och dividera *N* allteftersom man hittar en faktor. Räkna antalet udda faktorer. Man vill alltså välja ett godtyckligt antal av varje faktor och bilda produkten, det udda talet. Antalet kombinationer blir $\prod(1+n_i)$ där n_i är antalet faktorer av ett specifikt primtal. Ett lite striktare matematisk motivering finns på http://www.qbyte.org/puzzles/p092s.html.

En fullständig faktorisering blir för långsam. Därför måste man vara smart i slutet. När man inser att det som mest finns två faktorer kvar kan man urskilja tre fall: Faktorerna är lika, olika eller att det är ett primtal. Det första fallet är enkelt att kontrollera med kvadratrot. Det tredje kan kontrolleras med en snabb metod för primtalstest. Ett exempel är http://en.wikipedia.org/wiki/Miller–Rabin_primality_test. Notera att i Java finns `BigInteger.isProbablePrime(probability)` färdigimplementerad. Eftersom vi kan skicka in lösningar ett godtyckligt antal gånger kan sannolikheten modifieras efter vad som krävs mot testdata för rätt svar inom rätt körtid.

Den sista svårigheten lär aldrig dyka upp under en riktig programmeringstävling eftersom koden inte är deterministisk. Undvik även rand() i C/C++ eftersom användning av denna kan vara förbjudet och ge runtime error. Använd en egen deterministisk generering av ”slumptal”.