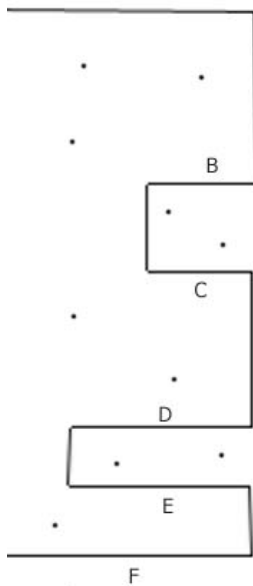


## Zadatak arhitekt

Da bi pojednostavili rješenje zadatka, prvo ćemo translirati poligon (i točke) tako da nema negativnih koordinata. Budući da znamo da je orijentacija poligona u smjeru kazaljke na satu, možemo «kretanje poligonom» definirati ovako:

- kada idem desno (horizontalna stranica za koju je  $A[i].x < A[i+1].x$ ), oduzimam broj točaka koji se nalazi u pravokutniku određenog x-koordinatama stranice i y-koordinatom stranice te apscisom (podsjetimo se da sada ne postoje negativne koordinate)
- kada idem lijevo (horizontalna stranica za koju je  $A[i].x > A[i+1].x$ ), zbrajam broj točaka koji se nalazi u pravokutniku «ispod» te stranice



Primjer:

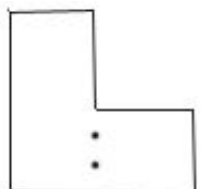
Neka je  $f(x)$  broj točaka ispod stranice  $x$ .

Ako promatramo ovaj dio poligona prema gornjem algoritmu broj točaka bit će jednak

$$f(A) - f(B) + f(C) - f(D) + f(E) - f(F)$$

To možemo zamisliti i ovako: krećemo se po  $y$  osi odozgo: sada, kada promatramo pojedinu stranicu ili točke ispod nje isključujemo (ako će ta stranica parni puta prekriti neki interval na  $x$ -osi) ili uključujemo (neparni), slično kao kod provjere da li se neka točka nalazi u poligonu, što je poznat algoritam.

Kao poseban slučaj još ostaju poligoni u kojima se točke nalaze na istom pravcu kao i neki vertikalni segment -> tada se može dogoditi da neke točke brojimo ili oduzimamo prevelik odnosno premalen broj puta.



Takav se slučaj može riješiti tako što ćemo poligon malo proširiti, odnosno svaku njegovu stranicu produljiti za neki realan broj  $0 < p \leq 0.5$ ; na taj način ćemo osigurati da se niti jedno drvo ne nalazi na istom pravcu kao neka stranica poligona, a budući da su sve koordinate iz zadatka cijeli brojevi, niti jedna nova točka neće «ući» u poligon, odnosno nijedna neće «izaći» nakon te transformacije. To možemo postići jednostavnom strategijom; ponovno se krećemo po poligonu i definiramo ove radnje:

- za desno orijentirane stranice: umanji  $y$  koordinatu za  $p$
- za lijevo orijentirane stranice: uvećaj  $y$  koordinatu za  $p$
- za stranice orijentirane prema dolje: umanji  $x$  koordinatu za  $p$
- za stranice orijentirane prema gore: uvećaj  $x$  koordinatu za  $p$

Jedini nam još problem predstavlja kako brzo prebrojati broj točaka u pravokutniku. Postoji nekoliko struktura koje to mogu efikasno napraviti, jedna od njih je modifikacija intervalnog stabla -> pomoću nje na takvo pitanje se uz prekalkuliranje u  $O(n \lg n)$  vremenu i jednako toliko memorije, može odgovoriti u  $O(\lg^2 n)$ , a uz malo optimizacije i u  $O(\lg n)$ . Ovdje ćemo ukratko opisati prvi pristup, koji je implementiran u službenom rješenju.

Napomena: U daljnjem tekstu pretpostavlja se da je čitatelj upoznat s interval stablom.

Da bismo mogli odgovarati na tražena pitanja efikasno potrebno je modificirati interval stablo na sljedeći način: svaki čvor je «zadužen» za neki interval x-eve, i također on sadrži **sortirani** popis y koordinata točaka koje imaju x-eve u intervalu pojedinog čvora.

Primijetimo da popis y koordinata za roditelja možemo stvoriti iz njegove djece meranjem (slično kao kod merge sorta). Tako popunjavamo dubinu po dubinu stabla (počevši od listova) sve dok ne dođemo do roota stabla. Vremenska složenost tog postupka je  $O(n \lg n)$  jer je dubina stabla  $O(\lg n)$ , a zbroj broja operacija na svakoj dubini  $O(n)$ . Memorijska složenost je također  $O(n \lg n)$  jer je svaka točka pohranjena u listama najviše  $O(\lg n)$  svojih predaka.

Ovakvom implementacijom dolazimo do konačnog rješenja cijelog zadatka u složenosti:

- a) vrijeme:  $O(n \lg n + q * k * \lg^2 n)$ ;
- b) memorija:  $O(n \lg n)$

## Zadatak binarni

Zadatak ćemo riješiti principom matematičke indukcije. Pretpostavimo da smo do sada riješili zadatak za neki  $N$  i da smo izgradili pravilan niz  $A_1, A_2, \dots, A_i$ . Ukoliko taj niz sada pretvorimo u:

$0A_1, 0A_2, 0A_3, \dots, 0A_{i-1}, 0A_i, 1A_i, 1A_{i-1}, \dots, 1A_3, 1A_2, 1A_1$

Možemo vidjeti da smo na ovaj način napravili pravilan niz za  $N+1$ . Ovim principom je moguće sagraditi pravilan niz za bilo koji prirodni broj  $N$  tako da krećemo od trivijalnog slučaja  $N = 1$  gdje su rješenja  $\{0, 1\}$  i primijenjujući gore opisani algoritam  $N-1$  puta.

## Zadatak čuvar

Zbog malih ograničenja u zadatku, moguće je provjeriti na sve načine gdje sve možemo zavezati uzicu psa. Tih mjesta ima  $\leq 41 * 41 = 1681$ . Kada provjeravamo neku poziciju, možemo „pohlepno“ produžiti uzicu do najbližeg zida, te potom provjeriti je li uzica dovoljno dugačka da dođe do svih otvora.

## Zadatak kalendar

Zbog malih ograničenja možemo provjeriti svaku godinu iz intervala  $[1900, 2100]$  zadovoljava li uvjete zadatka. Pažljivom implementacijom generiramo cjelokupni kalendar za svaku od traženih godina, te tada ga pretražimo sadrži li traženi pravokutnik. To radimo tako da na sve načine probamo odrediti gornji lijevi vrh malog pravokutnika na kalendaru, te tada samo provjerimo jesmo li naišli na ono što tražimo.

## Zadatak kemija

Prvo što treba primijetiti kod zadatka je da mi tražimo  $n$  koeficijenata  $A_1, A_2, \dots, A_n$  takve da je broj nekog elementa na lijevoj strani jednak broju elemenata na desnoj strani, odnosno da je njihova razlika jednaka nuli. Pogledajmo jednadžbu  $Al + O_2 = Al_2O_3$ ...

Neka koeficijent ispred  $Al$  bude  $a$ , ispred  $O_2$  bude  $b$ , dok je  $c$  ispred  $Al_2O_3$ ... Sada primjećujemo da je broj elemenata  $Al$  na lijevoj strani jednak  $a$ , dok je na desnoj jednak  $2c$ . Iz čega možemo zaključiti da  $a - 2c = 0$ , slično tome možemo za  $O$  zaključiti:  $2b - 3c = 0$ . Sada smo dobili sustav jednadžbi:

$$a - 2c = 0$$

$$2b - 3c = 0$$

No ovaj sustav još nije jedinstveno određen, tj. za sada ima beskonačno rješenja. Kako bi ga pretvorili u jedinstveni sustav, dodati ćemo potreban broj puta jednadžbe tipa  $A_i = 1$ . Dakle, ukoliko sustav nema jedinstveno rješenje, dodamo takvu jednadžbu za varijablu koja do sada nije jedinstveno određena i tako sve dok ne dobijemo rješenja. Da bismo provjerili ima li sustav jedinstveno rješenje, možemo koristiti kramerovo pravilo, ili možemo upotrijebiti široko poznatu Gaussovu eliminaciju za rješavanje sustava linearnih jednadžbi.

Nakon što smo dobili rješenje, ono može na primjer biti oblika  $A_1 = 1/4$ ,  $A_2 = 2/3$ ,  $A_3 = 2$ . Tada možemo sve varijable izmnožiti s najmanjim zajedničkim višekratnikom njihovih nazivnika (u ovom slučaju je to 12) kako bismo dobili  $A_1 = 3$ ,  $A_2 = 8$ ,  $A_3 = 24$ . Zbog ovoga je važno primijetiti kako je Gaussovu eliminaciju potrebno raditi nad razlomcima, a ne na realnom tipu podataka, kako bismo mogli lagano izračunati nazivnike, te pritom izbjegli pogreške u nepreciznosti.

## Zadatak pekar

Ovaj zadatak je najlakše riješiti metodom takozvanog binarnog pretraživanja. Mi možemo pitati je li moguće ispeći sve kolače u nekom zadanom vremenu  $t$ . Ukoliko nije moguće, znamo da moramo povećati vrijeme  $t$ , inače znamo da ga ne moramo povećati te ga probamo sniziti. Više o binarnom pretraživanju možete pronaći na adresi:

<http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=binarySearch>

Provjeriti je li moguće sve ispeći u nekom vremenu  $t$  je vrlo jednostavan podzadatak koji se može riješiti u  $O(n)$  vremenu.

## Zadatak spoj

Da biste lakše shvatili rješenje ovog zadatka potrebno je definirati neke osnovne matematičke pojmove:

- $A * A * A * \dots A = A^B$  (gdje se A pojavljuje B puta)
- (A povrh B) označava binomni koeficijent, na koliko se načina može od A stvari odabrati njih B ukoliko nam poredak nije važan. Tako na primjer  $(4 \text{ povrh } 2) = 6$ . Više o tome možete pročitati na: <http://en.wikipedia.org/wiki/Combination>
- Ostatak broj A pri dijeljenju sa M označimo sa  $A \bmod M$
- $A * B \bmod M = (A \bmod M) * (B \bmod M) \bmod M$
- $A / B \bmod P = A * I_B \bmod P$  gdje je  $I_B$  modularni inverz od B s obzirom na P, da biste saznali više o modularnom inverzu i njegovom efikasnom pronalaženju pogledajte na [http://en.wikipedia.org/wiki/Extended\\_Euclidean\\_algorithm](http://en.wikipedia.org/wiki/Extended_Euclidean_algorithm)

Podijeliti ćemo problem na nekoliko slučajeva:

$A < B$ ) U ovom slučaju rješenje je očito 0.

$A = B$ ) Rješenje je očito  $1 * 2 * 3 * \dots * A = A!$

Ostatak teksta će se baviti slučajem  $A > B$ . Zadatak ćemo riješiti formulom uključivanja i isključivanja: Koliko postoji spajanja prekidača ako nemamo uvjet da svaka žarulja mora biti spojena na barem jedan. Odgovor je  $B^A$ . Koliko je od tih spajanja krivih tako da smo nismo spojili ništa na točno jednu žarulju? Točno  $B * (B-1)^A$ . Nakon toga smo previše puta izbacili one koje nisu spojene na točno 2 žarulje, pa dodamo  $(B \text{ povrh } 2) * (B-2)^A$ , pa oduzimamo  $(B \text{ povrh } 3) * (B-3)^A$  i tako dalje.

Sada nam još jedino problem predstavlja što ovaj rezultat moramo izračunati modulo 1000000007 (veliki prosti broj). Dok je lako efikasno izračunati  $X^Y \bmod P$  pomoću brzog potenciranja, izračunavanje  $(X \text{ povrh } Y)$  nam predstavlja veći problem. Tu će nam pomoći modularni inverz. Prisjetimo se da  $(X \text{ povrh } Y) = X! / (Y! * (X-Y)!)$ . Možemo brzo konstruirati niz F takav da  $F_x = x! \bmod P$ , te D takav da  $D_x = I_1 * I_2 * \dots * I_x$ . Nakon toga izračunavanje  $(X \text{ povrh } Y)$  postaje  $(F_x * D_y * D_{x-y}) \bmod P$ .

## Zadatak suma

Prva stvar koju je potrebno primijetiti je da su ograničenja prevelika da bi za svaki broj  $x$  odredili sve njegove djelitelje te ih dodali konačnoj sumi. No, zadatak je jednostavniji nego što nam se čini na prvi pogled. Pokušajmo riješiti ovaj podzadatak:

Koliko ima brojeva djeljivih sa 7 među  $\{ 1, 2, 3, \dots, 100 \} \rightarrow [100/7] = 14$

Koliko ima brojeva djeljivih sa 7 među  $\{ 7, \dots, 77 \} \rightarrow [77/7] - [(7-1)-7] = 11 - 0 = 11$

Koliko ima brojeva djeljivih sa  $X$  među  $\{ L, L+1, \dots, R-1, R \} \rightarrow [R/X] - [(L-1)/X]$

Nakon toga možemo lagano odrediti koliko je brojeva iz intervala djeljivo sa nekim brojem, te tada samo pozbrajamo rezultate za svaki mogući djelitelj koji se može pojaviti djelitelj.

**Napomena:** postoji algoritam unutar jedne sekunde izračunava vrijednost  $f(x)$  za svaki broj iz skupa  $\{ 1, 2, \dots, 1\,000\,000 \}$ . Pokušajte ga pronaći.

## Zadatak torta

Ovaj zadatak zahtjeva malo bolje poznavanje teorije grafova. Prvo možemo primijetiti da ukoliko postoji dva grada  $A$  i  $B$ , takvi da postoji (direktni ili indirektni) put od  $A$  do  $B$ , ta također postoji put od  $B$  do  $A$ , da ukoliko dođemo do jednog grada, da možemo automatski doći i do drugog grada te se vratiti natrag bez posljedica. Sada podijelimo graf u minimalni broj komponenta takvih da za neka 2 grada u istoj komponenti vrijedi da postoji put od  $A$  do  $B$ , te od  $B$  do  $A$ . Te komponente se zovu dvostruko povezane komponente i o njima možete više saznati na adresi:

[http://en.wikipedia.org/wiki/Strongly\\_connected\\_component](http://en.wikipedia.org/wiki/Strongly_connected_component)

Ukoliko promatramo cijelu jednu komponentu kao zasebni grad, primijećujemo da nam se originalni problem nije uopće promijenio, samo što sada u grafu ne postoji niti jedan ciklus, tj. put koji počinje i završava u istom gradu. Ovo nam je bilo potrebno olakšanje te ga sada možemo riješiti.

Na početku se jedan trgovac nalazi u komponenti 1, obiđe sve gradove u njoj te onda gleda gdje ide dalje. Neka se komponenta u koju će on ići dalje zove sparena komponenta sa 1 (nazovimo je  $A$ ). Nakon toga on obiđe sve u  $A$  te ide u komponentu sparenu sa  $A$  (nazovimo je  $B$ ) i tako dalje sve dok jednom neka komponenta ne bude sparena. Nakon toga uzimamo drugog trgovca i ulazimo u neki drugi „lanac sparivanja“. Na kraju možemo primijetiti da je broj potrebnih trgovaca ustvari jednak broju nesparenih komponenta, stoga je potrebno maksimizirati broj sparenih komponenta. Primijetimo da komponenta  $A$  može biti sparena sa  $B$  ukoliko postoji put iz bilo kojeg grada unutar  $A$  do bilo kojeg grada unutar  $B$ . Problem traženja maksimalnog sparivanja je dobro poznati problem i o njegovom rješavanju više možete naučiti na:

<http://www.topcoder.com/tc?module=Static&d1=tutorials&d2=maxFlow>

(druga sekcija, podnaslov „Maximum Bipartite Matching“)