Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Nikola Dmitrović, Karlo Franić, Gabrijel Jambrošić, Marin Kišić, Josip Klepec, Vedran Kurdija, Daniel Paleka, Ivan Paljak, Stjepan Požgaj i Paula Vidas. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima.

Zadatak: Osijek

Pripremio: Nikola Dmitrović
Potrebno znanje: naredba učitavanja i ispisivanja, operacija dijeljenja

Rješenje ovisi o ostatku pri dijeljenju broja N s brojem K. Ako su oni djeljivi bez ostatka, očito je da će Lega točno „N podijeljeno s K" puta izvući K komada čipsa iz kutije s N komada. Ako nisu djeljivi bez ostatka tada će u kutiji, nakon što Lega iz nje „N podijeljeno s K" puta izvuče K komada, ostati još čipsa koji će se moći izvaditi u još jednom vađenju.

Programski kod (pisan u `Python 3`):

```
N = int(input())
K = int(input())

if N % K == 0:
  print(N // K)
else:
  print(N // K + 1)
```

ili

```
N = int(input())
K = int(input())
print(N // K + (N % K != 0))
```

Zadatak: Radio

Pripremili: Karlo Franić i Marin Kišić
Potrebno znanje: naredba ponavljanja, osnovne aritmetičke operacije, poznavanje tipova podataka za spremanje velikih brojeva

Promotrimo najprije parcijalu koja nosi 15 bodova. Unosimo tri broja X, A i B. Stjepan će u tom slušaju pojačati radio za B jedinica na $X + B$, a potom ga smanjiti za A jedinica na $X + B - A$. Dakle, za osvajanje prve parcijale bilo je dovoljno ispisati $X + B - A$.

Za osvajanje preostalih bodova potrebno je primijetiti da će se za svaki Anin upit početna jačina promijeniti za $B_i - A_i$ jedinica. Petljom prolazimo kroz sve Anine upite i trenutnoj jačini dodajemo $Bi - Ai$ za i-ti upit.

Promatrajmo slučaj u kojemu je N maksimalan, svaki B maksimalan, a svaki A minimalan. Tada će konačna jačina glazbe biti $N \cdot MAXB + X = 100000 * 100000 + X$. Primijetimo da taj broj više ne stane u klasičan 32-bitni cjelobrojni tip podatka (npr. `int` u jezicima `C/C++`). Da biste uspjeli spremiti rješenje potrebno je koristiti barem 64-bitni cjelobrojni tip podatka (npr. `long long` u jezicima `C/C++`).

Između ostalog, ovim su zadatkom autori odlučili provjeriti koliko je natjecatelja, obzirom na rezultate drugog zadatka prethodnog kola, naučilo ispravno koristiti brojčane tipove podataka u svojim omiljenim programskim jezicima.

Zadatak: ACM

Pripremili: Vedran Kurdija i Marin Kišić
Potrebno znanje: naredba učitavanja, naredba ponavljanja, rad sa stringovima, sortiranje

Za 20 bodova bilo je dovoljno zaključiti da će lista nakon odmrzavanja izgledati isto kao i zamrznuta lista. Dakle, trebalo je pronaći u kojem se retku zamrznute liste nalazi naš tim i ispisati indeks tog retka.

Za preostalih 30 bodova, trebalo je zaključiti da su, u najgorem slučaju za naš tim, svi ostali timovi točno riješili sve zadatke sa `'?'`, tj. zadatke poslane dok je lista bila zamrznuta. Intuicija je jasna, što više točnih rješenja imaju drugi timovi, to gore za naš tim. Zato, u zamrznutoj listi možemo za sve timove osim našega pretvoriti sve `'?'` u `'+'`. Ovime smo dobili odmrznutu listu u najgorem slučaju za naš tim. Sada nas zanima koliko će se timova u tako odmrznutoj listi naći iznad našeg tima.

Najprije ćemo iz odmrznutog retka našeg tima koji smo dobili na ulazu prebrojiti broj točno riješenih zadataka te izračunati *penalty* vrijednost našeg tima. Uvest ćemo pomoćnu varijablu pozicija sa početnom vrijednošću 1, na koju ćemo dodati koliko je boljih timova od našeg.

Petljom ćemo obići sve timove u odmrznutoj listi (osim našeg) te za svaki prebrojiti koliko točno riješenih zadataka imaju, kao i kolika je njihova *penalty* vrijednost. Ako neki tim ima više riješenih zadataka od našega, povećat ćemo varijablu `pozicija` za 1. Ako neki tim ima jednak broj riješenih zadataka kao naš, usporedit ćemo *penalty* vrijednosti tog i našeg tima te ako taj tim ima manju *penalty* vrijednost, uvećati varijablu `pozicija` za 1. Ako neki tim pak ima i jednak broj riješenih zadataka kao naš i jednaku *penalty* vrijednost kao naš, usporedit ćemo imena timova. Ako je ime tog tima prije po abecedi, uvećat ćemo varijablu `pozicija` za 1. Na kraju ćemo ispisati varijablu `pozicija`.

Zadatak: Slagalica

Pripremio: Gabrijel Jambrošić
Potrebno znanje: analiza slučajeva, pohlepni algoritmi

Prvu parcijalu moguće je riješiti jednostavnom analizom slučajeva koristeći par `if` naredbi.

Drugu parcijalu moguće je riješiti isprobavanjem svih permutacija u vremenskoj složenosti $\mathcal{O}(n!)$. Implementaciju ovog rješenja možete vidjeti u izvornom kodu `slagalica_n_fact.cpp`

Primijetimo da ćemo u trećoj parcijali naizmjence stavljati komadiće oblika 1 i 4 pa je optimalno uvijek uzeti onaj s najmanjim brojem kojeg možemo staviti. Ako nam na kraju ostane višak, rješenje ne postoji.

Za četvrtu parcijalu pretpostavimo da smo složili neke komadiće te zadnji kojeg smo stavili na desnom kraju ima udubinu. Tada sljedeći komadić kojeg ćemo staviti na lijevom kraju mora imati izbočinu, a taj uvjet zadovoljavaju komadići oblika 1 i 2. Ako stavimo komadić oblika 2, na desnoj će strani opet biti udubina te ćemo opet trebati komadić oblika 1 ili 2. S druge strane, ako stavimo komadić oblika 1, onda ćemo u sljedećem koraku trebati komadiće oblika 3 ili 4. Drugim riječima, komadić oblika 1 mijenja traženi oblik sljedećeg komadića, a isto vrijedi i za komadić oblika 4. Stoga ćemo u ovom podzadatku najprije postaviti sve komadiće oblika 2 (ili 3, ovisno o obliku početnog komadića), zatim ćemo staviti komadić oblika 1 ili 4, a onda ćemo postaviti sve oblika 3 (ili 2). Uvijek postavljamo komadić traženog oblika s najmanjim brojem jer će tada i niz biti najmanji. Ako na kraju ne možemo spojiti zadnji komadić, rješenje ne postoji. Dodatno, moramo paziti na slučaj kada ne postoje komadići oblika 1 i 4.

Za potpuno rješenje zadatka, uzmimo na početku jedan naivan pohlepni algoritam kojim uvijek uzimamo komadić traženog oblika s najmanjim brojem. Problem je taj da na kraju mogu ostati komadići oblika 2 ili 3 (ako ostanu komadići oblika 1 ili 4 rješenje ne postoji) koje ne možemo više staviti, a prije smo ih mogli. To možemo riješiti na način da te komadiće stavimo na zadnje mjesto na koje smo ih još uvijek mogli staviti jer će tamo najmanje povećati konačan niz. Možemo primijetiti da će to mjesto biti neposredno ispred zadnjeg komadića oblika 1 ili 4 jer smo tamo posljednji put promijenili traženi oblik komadića.

Vremenska složenost: $\mathcal{O}(n)$.

Zadatak: Checker

Pripremili: Paula Vidas i Daniel Paleka
Potrebno znanje: matematička indukcija, vezana lista

Tvrdnja 1: U svakoj triangulaciji za $N \geq 4$ postoje barem dva "uha", tj. trokut koji dijeli dvije stranice s mnogokutom.

Skica dokaza: Indukcija. Vrijedi za kvadrat. Neka dijagonala dijeli mnogokut na dva manja mnogokuta. Ako je neki od njih trokut, to je traženo uho; inače se možemo induktivno pozvati dalje.

Prvo ćemo opisati postupak provjere triangulacije. Naivni algoritam miče uha te uvijek održava trenutnu triangulaciju. To je moguće implementirati u mnogim složenostima, pa je $\mathcal{O}(n^2)$ implementacija bila dovoljna za prva dva podzadatka. Ovdje dajemo jedan mogući $\mathcal{O}(n \log n)$ algoritam.

Za svaku dijagonalu ab, napravimo usmjerene bridove (a, b) i (b, a). Sada svih $2(N-2)$ bridova sortirajmo po duljini iz početnog do krajnjeg vrha u smjeru kazaljke na satu. Jasno je da prvi element sortiranog niza odgovara uhu. Može se dokazati da ako uha mićemo tim redoslijedom, u svakom trenutku će prvi preostali brid u nizu odgovarati uhu kojega trebamo maknuti u ovom koraku. (Naravno, ako dani brid ne čini uho, triangulacija nije ispravna.)

Za detalje implementacije pogledajte službeno rješenje, gdje koristimo vezanu listu za održavanje trenutnog poretka vanjskih vrhova mnogokuta. Tada su provjere je li neki brid čini uho i dobre obojenosti uha jednostavne i mogu se raditi paralelno s danim algoritmom.

Zadatak: Popcount

Pripremio: Ivan Paljak
Potrebno znanje: konstruktivni algoritmi, turnirsko stablo

Nepoznat programski jezik, dopuštena samo jedna varijabla, ograničenja na broj naredbi i duljinu naredbe, zvuči kao savršena kombinacija za rješavanje zadatka "podzadatak po podzadatak".

U prvom podzadatku dopušteno nam je koristiti broj naredbi koji je (skoro) jednak broju bitova ulazne vrijednosti. Ovo nam sugerira da ćemo zadatak rješavati "bit po bit". Pretpostavimo da smo sufiks od X bitova uspješno riješili, odnosno, taj sufiks varijable A sadrži vrijednost koja odgovara broju jedinica u tom istom sufiksu ulazne vrijednosti dok ostali bitovi nisu promijenjeni u odnosu na bitove ulazne vrijednosti. Primijetimo da sama ulazna vrijednost u program već ima riješen sufiks veličine 1. Sufiks od $X+1$ bitova ćemo riješiti tako da vrijednost $(X+1)$-og bita pribrojimo varijabli A te taj bit postavimo na 0. To možemo napraviti naredbom `A=((A&(((1<<N)-1)-(1<<X)))+((A&(1<<X))>>X))`. Koristeći naredbe ovog oblika za svaki bit mogli ste riješiti prvi podzadatak.

Za osvajanje bodova na drugom podzadatku dovoljno je iskoristiti ideju iz prvog podzadatka zajedno sa svojstvom jezika da se varijabla A smije pojavljivati najviše 5 puta. Naravno, umjesto "bit po bit", zadatak ćemo rješavati tehnikom "četiri bita po četiri bita" i tako smanjiti broj naredbi 4 puta.

Za rješavanje preostalih podzadataka bilo je potrebno sjetiti se jedne poznate strukture podataka – *turnirskog stabla*. Zamislimo da je nad našim bitovima izgrađeno turnirsko stablo u kojem svaki čvor pamti sumu bitova u svom podstablu. Umjesto u dekadskom brojevnom sustavu, zamislimo da je u svakom čvoru turnirskog stabla napisana vrijednost u binarnom sustavu te da je prikazana u onoliko bitova kolika je veličina intervala kojeg taj čvor pokriva.

Slijepimo li vrijednosti svih listova (slijeva nadesno), dobit ćemo ulaz u program. Slijepimo li vrijednosti njihovih roditelja (slijeva nadesno), dobit ćemo željenu vrijednost varijable A nakon prve naredbe. Ponavljamo li ovaj postupak sve do korijena stabla, u končnici ćemo u varijabli A imati točno rješenje, a broj naredbi bit će reda $\mathcal{O}(\log n)$. Ostaje nam samo osmisliti prijelaz između redaka u turnirskom stablu.

U prvoj naredbi mogli bismo uzeti sve bitove na parnim pozicijama te im pribrojiti posmaknute bi-

tove na neparnim pozicijama. To možemo dobiti izrazom `((...01010101&A)+((...10101010&A)>>1))`. U sljedećem koraku algoritma potrebno je riješiti intervale duljine četiri pa je odgovarajući izraz `((...00110011&A)+((...11001100&A)>>2))`. Ponavljanjem ovog postupka dobivamo konačno rješenje zadatka.

Ovisno o načinu na koji ste izračunali vrijednosti potrebnih konstanti (...0101010, ...10101010, ...) mogli ste osvojiti različit broj bodova. Naime, neki načini implicitnog generiranje ovih konstanti će prekršiti uvjet da se svaka naredba smije sastojati od najviše 1000 znakova.

U ovom zadatku, radi lakše implementacije, preporučamo korištenje programskih jezika `Python` ili `Java`.

Zadatak: Zvijezda

Pripremio: Marin Kišić
Potrebno znanje: ccw funkcija, binarna pretraga, analiza problema

Neka je s A_i označena i-ta točka N-terokuta, a s X točka iz upita. Indekse kod točaka u N-terokutu gledamo modulo N. Za 20 bodova bilo je dovoljno znati kako provjeriti nalazi li se točka s neke strane pravca. Funkcija koja daje odgovor na to popularno se zove *ccw*. Dakle, proći ćemo po svim dužinama i ako za neki i vrijedi $ccw(A_i, A_{i+1}, X) > 0$ i $ccw(A_{i+\frac{n}{2}}, A_{i+\frac{n}{2}+1}, X) > 0$, to znači da se točka nalazi u obojenom dijelu površine.

Za dodatnih 30 bodova mogli ste implementirati takozvano *offline* rješenje. To je rješenje koje prvo učita sve upite, obradi u nekom poretku (ne nužno onom u kojem su zadani u inputu) te onda redom odgovori na njih.

Za svih 110 bodova bilo je potrebno osmisliti *online* rješenje, odnosno rješenje koje kada učita upit odmah odgovori na njega.

Ako je $ccw(A_i, A_{i+1}, X) > 0$, na i-tu stranicu N-terokuta ćemo napisati 1, a inače 0. Sada je pitanje postoji li par nasuprotnih stranica na kojima pise 1. Očito ne možemo za svaki upit provjeriti što piše na svakoj stranici jer je vremenska složenost takvog algoritma $\mathcal{O}(nq)$, ali možemo nizom zamjedbi doći do efikasnijeg rješenja.

Zamjedba 1: Ako neke dvije stranice imaju 1 na sebi, onda i sve stranice između njih s neke strane N-terokuta imaju isto 1 na sebi. Ovo možemo lagano uočiti promatranjem stranica na kojima je napisana 0. Zamislimo da je točka iz upita izvor svjetlosti te da su stranice zidovi kroz koje ne prodire svjetlost. Možemo uočiti da, ako svjetlost može doći do zida direktno iz izvora svijetlosti, onda je na tom zidu napisana 0. Sada lako vidimo da sve nule tvore interval na tom cikličnom nizu. Jasno, iz toga slijedi da i jedinice također tvore interval.

Zamjedba 2: odgovor je `DA` ako i samo ako je interval jedinica veličine barem $\frac{n}{2} + 1$. Ovo je očito jer u $\frac{n}{2} + 1$ uzastopnih stranica moraju postojati dvije koje su nastuprotne.

Sada uzmimo proizvoljnu stranicu i njezinu nasuprotnu stranicu te pogledajmo koje vrijednosti pišu na njima. Ako na objema piše 1 možemo odmah ispisati `DA`. Ako na objema pise 0 možemo odmah ispisati `NE`. Ako na jednoj stranici piše 1, a na drugoj 0 onda možemo prelomiti niz na dvije polovice, svaka započinje stranicom za koju smo pogledali koja je vrijednost gore. Jedna polovica je oblika `111...000`, a druga `000...111`. U jednoj želimo pronaći zadnju jedinicu, u drugoj prvu jedinicu, a to možemo s binarnom pretragom. Kad smo to pronašli zapravo smo pronašli i rubove našeg intervala jedinica pa sad možemo lako odgovoriti na zadano pitanje.

Vremenska složenost: $\mathcal{O}(q \log n)$