Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Domagoj Bradač, Marin Kišić, Adrian Satja Kurdija, Vedran Kurdija, Ivan Paljak, Tonko Sabolčec i Paula Vidas. Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak: Bitstring

Inspiracija: Rusija 2015., zadatak Пингвиноведение[1]
Pripremio: Adrian Satja Kurdija
Potrebno znanje: dinamičko programiranje, strukture podataka

Trebamo pronaći niz koji se sastoji od najviše k blokova istih znamenki, a među njima onaj koji se od početnog razlikuje u minimalnom broju pozicija. Ako duljina izvornog niza nije velika, tada možemo proći kroz sve moguće nizove u $O(2^n)$ i provjeriti jesu li podijeljeni u potreban broj blokova. Za one koji jesu pamtimo minimalnu cijenu – broj znakova u kojima se taj niz razlikuje od izvornika. Ovo rješenje nosi 20 bodova.

Neka $ans[i][j]$ označava optimalan odgovor za prefiks izvornog niza duljine i, ako ga podijelimo u j blokova.
Inicijalizacija: prazan prefiks može se podijeliti na bilo koji broj blokova, cijena bilo kojeg dijeljenja je 0.
Prijelaz: iteracijom po pozicijama $x \leq i$ određujemo mjesto početka posljednjeg bloka, a zatim

$$ans[i][j] = \min_{x=1}^{i}(ans[x-1][j-1] + trosak[x][i]),$$

gdje $trosak[x][i]$ odgovara optimalnom razdvajanju intervala $[x, i]$ u jedan blok, a računa se kao manji od broja nula i broja jedinica u tom intervalu.
Odgovor na problem sadržan je u $ans[n][k]$. Ovo rješenje nosi 40 bodova i ima složenost $O(N^2K)$.

Neka $ans[i][j][c]$ označava optimalan odgovor ako prefiks duljine i podijelimo na j blokova, a zadnji blok čine znakovi c.
Prijelaz: sljedeći znak može započeti novi blok ili nastaviti stari, tj.

$$ans[i][j][c] = (s[i] \neq c) + \min(ans[i-1][j][c], ans[i][j-1][0], ans[i][j-1][1])$$

Ovo rješenje nosi 60 bodova i ima složenost $O(NK)$.

Zamislimo da smo cijeli izvorni niz pretvorili u jedan blok nula. Sada ćemo odabrati neke blokove (najviše $k/2$ njih) koji će se sastojati od jedinica. Odaberemo li neki blok i promijenimo ga u jedinice, cijena particije mijenja se za $-$broj jedinica + broj nula. Dakle, ako izvorni niz brojeva 0 i 1 zamijenimo nizom brojeva -1 i 1, potrebno je riješiti sljedeći problem: u nizu treba odabrati najviše $k/2$ disjunktnih intervala s maksimalnim zbrojem.

Pretpostavimo da u nizu trebamo odabrati k disjunktnih intervala s maksimalnim zbrojem. Za $k = 1$ potrebno je pronaći jedan interval s maksimalnim zbrojem, to je standardni problem. Odgovor ćemo konstruirati postupno: pretpostavimo da smo već odabrali k intervala, a želimo odabrati $k + 1$. Tvrdimo da treba ili dodati interval disjunktan sa svima već odabranima, ili jedan od odabranih intervala podijeliti na dva dijela, "izrezujući" neki dio iz njega. Za obje svrhe potrebno je moći pronaći podinterval s maksimalnim/minimalnim zbrojem na intervalu, što možemo tournament stablom. Budući da u svakom koraku postoji $O(k)$ intervala za navedenu provjeru, ovo rješenje može se implementirati u složenosti $O(k^2 + n \log n)$ i nosi 80 bodova.

Da bismo optimirali rješenje, valja primijetiti da se naše mogućnosti vrlo malo mijenjaju iz koraka u korak: za upit se pojavljuju tri nova intervala a uklanja se jedan. Dakle, s pomoću strukture prioritetnog reda možemo odabrati optimalnu radnju za svaki korak. Ovo je rješenje složenosti $O(n \log n)$.

[1] http://neerc.ifmo.ru/school/archive/2014-2015/ru-olymp-roi-2015-day1.pdf

Zadatak: Mnogokut

Autor: Paula Vidas
Pripremili: Adrian Satja Kurdija i Paula Vidas
Potrebno znanje: metoda dvaju pokazivača (two pointers)

Trojku (različitih) vrhova mnogokuta zvat ćemo *dobrom* ako je moguće dani mnogokut saviti u trokut s tim vrhovima. U suprotnom, kažemo da je trojka *loša*.

20% bodova moguće je ostvariti jednostavnim *brute force* algoritmom. Za svaku trojku vrhova provjerimo je li dobra. Trojka je dobra ako i samo ako za duljine stranica, tj. zbrojeve duljina štapića između vrhova, vrijedi da je duljina svake stranice manja od zbroja duljina druge dvije. Algoritam je moguće implementirati u složenosti $\mathcal{O}(n^3)$.

Za ostatak rješenja ključna ideja je sljedeća: umjesto da brojimo dobre trojke, brojat ćemo loše. Na kraju od ukupnog broja trojki, koji je jednak $\binom{n}{3}$, samo oduzmemo broj loših.

Trojka je loša ako i samo je duljina najveće stranice veća ili jednaka zbroju duljina druge dvije stranice. To vrijedi ako i samo ako je duljina najveće stranice veća ili jednaka polovici opsega.

Fiksirajmo vrhove najdulje stranice. Ako su zbrojevi duljina s obje strane jednaki, treći vrh možemo odabrati proizvoljno. Inače, treći vrh može biti bilo koji vrh na "kraćoj" strani. Ovaj pristup moguće je implementirati u složenosti $\mathcal{O}(n^2)$, što je dovoljno za 50% bodova.

Sada ćemo opisati puno rješenje. Za vrh x označimo s $f(x)$ prvi vrh u smjeru kazaljke na satu za koji je zbroj duljina štapića od x do $f(x)$ veći ili jednak polovici opsega. Neka je k jednak broju vrhova strogo između x i $f(x)$ (u smjeru kazaljke na satu). Tada vrijedi da je broj loših trojki u kojima je x jedan od vrhova najdulje stranice, a drugi vrh je onaj koji mu je bliži u smjeru kazaljke na satu, jednak $\binom{n-k-1}{2}$. Naime, za druga dva vrha možemo uzeti bilo koje vrhove između $f(x)$ i x, uključujući $f(x)$, ali naravno ne i x.

Kako efikasno naći $f(x)$? Koristit ćemo tzv. metodu dva pokazivača. Vrhove mnogokuta označimo s $1, 2, \ldots n$ u smjeru kazaljke na satu. Na početku oba pokazivača pokazuju na vrh 1. Pomičemo drugi pokazivač u smjeru kazaljke na satu sve dok je zbroj duljina između pokazivača manji od polovice opsega. Kad stanemo, našli smo $f(1)$. Pomaknimo sad prvi pokazivač u vrh 2. Primjetimo da se $f(2)$ ne može nalaziti strogo između 2 i $f(1)$. Opet pomičemo drugi pokazivač dok zbroj ne postane veći ili jednak polovici opsega, odnosno dok ne nađemo $f(2)$. Zatim isti postupak ponavljamo redom za sve vrhove $3, 4, \ldots n$. Na kraju, prvi pokazivač je napravio jedan, a drugi pokazivač najviše dva kruga oko mnogokuta, pa je složenost ovog rješenja $\mathcal{O}(n)$.

Alternativno, $f(x)$ možemo naći binarnim pretraživanjem. U tom slučaju složenost je $\mathcal{O}(n \log n)$.

Zadatak: Razmaci

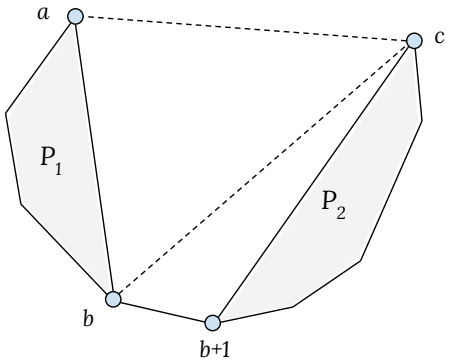
Autor: Adrian Satja Kurdija
Pripremili: Adrian Satja Kurdija i Ivan Paljak

Zadatak možemo riješiti tako da za svako pravilo prođemo po danom stringu i pobrinemo se da je to pravilo zadovoljeno. Tako je implementirano službeno rješenje. Prilikom svakog prolaza stvaramo novi string koji će zamijeniti stari. On će, naravno, sadržavati sve znakove koji nisu razmaci, no kad naiđemo na razmak valja odlučiti hoćemo li ga ubaciti ili preskočiti, a možda ga treba i ubaciti negdje gdje se prije nije nalazio. U tu svrhu održavamo varijablu *nospace* koja nam govori treba li preskočiti razmak, a mijenjamo je u skladu s pravilom koje u određenom prolazu primjenjujemo. Nove razmake ubacujemo kada zaključimo da je to potrebno, ovisno o prethodnom i idućem znaku u skladu s pravilom.

Zadatak: Torta

Autor: Tonko Sabolčec
Pripremili: Tonko Sabolčec i Marin Kišić
Potrebno znanje: strukture podataka, računanje površine mnogokuta

Pretpostavimo da su svi upiti tipa 1. Upit je zapravo određeni interval točaka na konveksnoj ljusci za koji moramo pronaći dvije površine — ovisno o tome idemo li od prve do druge točke u smjeru kazaljke na satu ili u suprotnom smjeru. Da si olakšamo i izbjegnemo *cirkularne* slučajeve, uvijek ćemo tražiti površinu idući od točke manje oznake prema točki veće oznake (dakle, u smjeru suprotnom smjeru kazaljki na satu).



Pretpostavimo da želimo izračunati površinu za interval $[a, c]$ i da smo prethodno izračunali površine za susjedne intervale $[a, b]$ i $[b + 1, c]$, kao na slici. Tada površinu za interval $[a, c]$ možemo dobiti zbrajanjem $P = P_1 + P_2 + P(\Delta a, b, c) + P(\Delta b, b + 1, c)$. Ako unaprijed odredimo površine za određene intervale, možda bismo mogli efikasno odrediti površinu za proizvoljni interval. Uz navedeni način spajanja intervala, zadatak se može riješiti standardnom primjenom turnirskog stabla (eng. *segment tree* — vidi `https://en.wikipedia.org/wiki/Segment_tree`).

Upitom tipa 2 zapravo *brišemo* točku mnogokuta. Možemo održavati sortirani skup (npr. `std::set` u C++-u) svih aktivnih točaka kako bismo efikasno odredili točke trokuta. Strukturu spomenutog turnirskog stabla zbog brisanja točke nije potrebno mijenjati — potrebno je samo pamtiti sumu svih trokuteva koje smo izrezali u promatranom intervalu i oduzeti od inicijalne površine za taj interval. Dakle, treba nam još jedno turnirsko stablo u kojem ćemo pamtiti površine obrisane u nekom intervalu (kod ažuriranja drugog turnirkog stabla, dovoljno je ažurirati samo točku koju brisemo iz mnogokuta). Ukupna složenost algoritma iznosi $\mathcal{O}(n \log n + q \log n)$.