



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Toni Brajko, Ivan Janjić, Fran Škarica, Sofija Velkovska i Jakov Celin.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Djelitelj

Pripremio: Toni Brajko

Potrebno znanje: naredbe učitavanja i ispisivanja

Broj 1 je djelitelj svakog prirodnog broja i ujedno je najmanji takav broj, stoga je najmanji zajednički djelitelj bilo kojih dvaju prirodnih brojeva 1. Potrebno je učitati dva broja s ulaza te ispisati 1.

Programski kod (pisan u Python 3):

```
input()
input()

print(1)
```

Zadatak 1.2

Pripremio: Toni Brajko

Potrebno znanje: naredba odlučivanja (if), tipovi podataka

Mogući ulazi su 1.1, 1.2, 2, 3, 4, 5, 6 i 7, a za pojedini slučaj potrebno je ispisati 1, 2, 3, 4, 5, 6, 7 ili 8, redom. Za oznake 1.1 i 1.2 dovoljno je ispisati zadnju znamenku, dok je za ostale potrebno ispisati taj broj uvećan za jedan.

Jedan od načina razlikovanja slučaja je da ulaz učitamo kao niz znakova. Ako je niz duljine 3, ispisujemo zadnju znamenku. U suprotnom znamo da smo učitali broj te ga trebamo uvećati za 1. Učitani znak pretvaramo u broj naredbom `int()` u Pythonu.

Programski kod (pisan u Python 3):

```
s = input()

if len(s) == 3:
    print(s[2])
else:
    print(int(s[0]) + 1)
```



Zadatak Papiga

Pripremio: Fran Škarica

Potrebno znanje: stringovi, naredba odlučivanja (if), naredba ponavljanja (for)

Zadanu riječ promatramo kao niz slova numeriranih od 1 nadalje. Papiga Dino izgovara slova na parnim pozicijama (2, 4, 6, ...) velikim slovom, dok slova na neparnim pozicijama ostaju mala.

Prolazimo kroz riječ slovo po slovo te za svako slovo provjeravamo nalazi li se na parnoj poziciji. Ako se nalazi, pretvaramo ga u veliko slovo, a u suprotnom ga ostavljamo nepromijenjenim.

Na kraju ispisujemo tako dobivenu riječ.

Programski kod (pisan u Pythonu 3):

```
s = list(input())

for i in range(len(s)):
    if i % 2 == 1:
        s[i] = s[i].upper()

print("".join(s))
```

Zadatak Kist

Pripremio: Jakov Celin

Potrebno znanje: matrice, naredba ponavljanja (for), naredba odlučivanja (if), računanje udaljenosti polja

Za potpuno rješenje zadatka održavati ćemo 2 varijable x i y koje će predstavljati u kojem retku i stupcu matrice se nalazimo. Na početku se nalazimo u polju u retku i stupcu $(n + 1)/2$. U slučaju da naiđemo na slovo koje nas pomiče u nekom od smjerova, pomaknemo se u tom smjeru **ako** možemo tj. ako nakon poteza ne izađemo iz matrice. U slučaju da naiđemo na slovo koje boja neka polja, prolazimo po cijeloj matrici te računamo udaljenost svakog polja od polja u retku x i stupcu y .

Jedino nam preostaje izračunati udaljenost između neka 2 polja matrice. Primjetimo da je udaljenost 2 polja matrice jednaka zbroju udaljenosti redaka i udaljenosti stupaca ta dva polja. Razlog tomu jest da u 1 koraku u 4 smjera se možemo približiti drugom polju **za** točno 1 redak ili 1 stupac. Udaljenost redaka a i b jednaka je $|a - b|$ tj. apsolutnoj razlici redaka a i b . Drugi način računanja udaljenosti redaka bio je oduzimajući manjeg retka od većeg. Isti postupak računanja koristimo za računanje udaljenosti 2 stupca. Za implementacijske detalje, pogledajte priloženo službeno rješenje.

Zadatak Festival

Pripremila: Sofija Velkovska

Potrebno znanje: dinamičko programiranje

Označimo bombone od najvećeg do najmanjeg brojevima od 1 do n , pri čemu je bombon broj 1 najveći, a bombon broj n najmanji. Ako je $k = 1$, svi bomboni moraju biti u jednoj kutiji, a bombon broj 1 mora biti prvi. Možemo staviti bilo koji preostali bombon na drugo mjesto u kutiji, pa imamo $n - 1$ opcija. Slično tome, možemo staviti bilo koji od preostalih $n - 2$ bombona na treće mjesto - to je $n - 2$ opcija, $n - 3$



opcija za četvrto mjesto, ..., 1 opcija za posljednje mjesto. Ukupno postoji $1 \cdot (n-1) \cdot (n-2) \cdot (n-3) \cdot \dots \cdot 1$ odnosno $(n-1)!$ načina na koje možemo složiti bombone u kutiju.

Kada je $n \leq 10$, možemo generirati sve particije skupa $\{1, 2, \dots, n\}$ i ako je particija sastavljena od točno k podskupova, izračunamo broj načina za slaganje bombona u svakom od njezinih podskupova koristeći istu logiku koju smo objasnili za slučaj $k = 1$. Množenjem brojeva za svaki podskup dobivamo odgovor za cijelu particiju.

Sada, pretpostavimo da imamo dvije kutije. Procesirat ćemo bombone jedan po jedan od bombona broj 1 do bombona broj n . Označimo broj načina za slaganje prvih m bombona s $dp(m)$. Pretpostavimo da smo već izračunali broj $dp(m-1)$ i želimo izračunati $dp(m)$. Za m -ti bombon postoje dvije mogućnosti: već smo složili prvih $m-1$ bombona u dvije kutije te zatim dodajemo m -ti bombon u jednu od njih, ili m -ti bombon stavljamo u zasebnu kutiju, dok se svih ostalih $m-1$ bombona nalazi u drugoj kutiji.

U prvom slučaju, mogli smo uzeti bilo koji raspored prvih $m-1$ bombona u dvije kutije, a zatim dodati m -ti bombon **nakon** bilo kojeg od već postavljenih bombona. Ne možemo ga staviti prije prvog bombona ni u jednoj od kutija jer je m -ti bombon i svi bomboni koji bismo kasnije mogli dodati u kutiju manji od svih bombona koji se trenutno nalaze u kutiji. To bi bilo protivno pravilu koje kaže da prvi bombon mora biti najveći, pa nije moguće staviti novi bombon prije trenutno prvog bombona u niti jednoj od obje kutije. Dakle, ukupno imamo $m-1$ mogućih pozicija za njega. To se može učiniti za bilo koji početni raspored prvih $m-1$ bombona - to je ukupno $(m-1) \cdot dp(m-1)$ načina.

U drugom slučaju, svi ostali $m-1$ bomboni moraju biti u drugoj kutiji - to se može učiniti na $(m-1)!$ načina kao što smo vidjeli u slučaju gdje je $k = 1$. Dakle, $dp(m) = (m-1) \cdot dp(m-1) + (m-1)!$ za $m \geq 2$ i $dp(1) = 0$ (ne možemo rasporediti jedan bombon u dvije kutije).

Ovu ideju možemo proširiti na slučaj gdje je $k > 2$. Koristit ćemo $dp(m, p)$ za označavanje broja načina za raspored prvih m bombona u točno p kutija. Ako pretpostavimo da je prvih $m-1$ bombona već složeno u p kutija i želimo dodati novi bombon u jednu od tih kutija, to bismo mogli učiniti na $(m-1) \cdot dp(m-1, p)$ načina. Ako želimo m -ti bombon u kutiji zasebno, trebamo izračunati broj načina na koje smo mogli rasporediti prethodnih $m-1$ bombona u $p-1$ kutija - to je točno broj $dp(m-1, p-1)$. Stoga bi prijelaz dinamičkog programiranja trebao biti $dp(m, p) = (m-1) \cdot dp(m-1, p) + dp(m-1, p-1)$. Za bazne slučajeve imamo $dp(1, 1) = 1$ (postoji jedan način da se jedan bombon složi u jednu kutiju), $dp(0, p) = 0$ za $p > 0$ (kutije ne smiju biti prazne prema pravilima) i $dp(m, 0) = 0$ za $m > 0$ (ne možemo složiti neki pozitivan broj bombona u 0 kutija).

Zadatak Domjenak

Pripremio: Ivan Janjić

Potrebno znanje: teorija grafova, bipartitni grafovi

U terminima teorije grafova, treba pronaći najdulji alternirajući put s obzirom na jedinstveno savršeno sparivanje u bipartitnom grafu. Iz svakog para u savršenom sparivanju uzmimo jedan vrh tako da je izabran skup nezavisan. Nazovimo taj skup "lijeva strana", a vrhove koje nismo odabrali "desna strana".

Pretpostavimo da smo pronašli savršeno sparivanje te zanemarimo na trenutak uvjet jedinstvenosti. Simetrična razlika sa nekim drugim savršenim sparivanjem je unija alternirajućih ciklusa i izoliranih vrhova.

Možemo fiksirati smjer kojim obilazimo alternirajuće cikluse. Drugim riječima odlučimo se da, kada prolazimo bridom u sparivanju, uvijek idemo s npr. lijeve strane na desnu. To znači da ako prolazimo bridom koji nije dio sparivanja idemo s desne strane na lijevu.

Prethodne tvrdnje motiviraju sljedeću promjenu: sve bridove koji su dio savršenog sparivanja usmjerimo da pokazuju s lijeve strane na desnu, a bridove koji nisu s desne strane na lijevu. Dobili smo usmjeren graf gdje usmjereni putovi/ciklusi odgovaraju alternirajućim putovima/ciklusima (s obzirom na izabrano sparivanje) u originalnom grafu.



Vratimo li se na uvjet jedinstvenosti, nije teško primijetiti da je postojanje usmjerenog ciklusa u grafu ekvivalentno postojanju još barem jednog savršenog sparivanja. Zaključak: ovaj usmjeren graf je acikličan.

Traženje najduljeg puta u takvom grafu poznat je problem koji se može riješiti u složenosti $O(N + M)$. Preostaje dovoljno brzo pronaći savršeno sparivanje. Primijetimo da u acikličkom usmjerenom grafu postoji vrh iz kojeg ne izlazi niti jedan brid. To povlači postojanje vrha(s lijeve strane) u originalnom grafu koji ima samo jednog susjeda. Maknimo taj vrh i njegovog susjeda iz grafa te ih označimo kao sparene. Postupak ponavljamo dok ne spojimo sve vrhove. Ovaj algoritam može se implementirati u složenosti $O(N + M)$. Konačno, ukupna složenost rješenja je $O(N + M)$. Dodatno, najdulji put može se računati paralelno sa pronalaženjem savršenog sparivanja. Za implementacijske detalje pogledajte službeno rješenje.

Zadatak Država

Pripremio: Jakov Celin

Potrebno znanje: dinamičko programiranje, reroot, računanje složenosti

Zadatak rješavamo fiksirajući glavni grad P (korijenujemo stablo u čvoru P) te brojeći za svaku veličinu koliko država s tom veličinom u glavnom gradu P postoji. Neka je $dp(x, k)$ broj država veličine k takvih da se sporedni gradovi nalaze u podstablu od x , a glavni grad države je x . Valja napomenuti da je P fiksirani glavni grad država koje ćemo prebrojavati, no svejedno promatramo x kao glavni grad u stanju dinamike.

Promatrajmo sada spajanje podstabla djeteta a s ostatkom podstabla roditelja b . Neka je veličina podstabla čvora a jednaka s_a , a veličina podstabla čvora b jednaka s_b . Tada prijelaze možemo zapisati (ignorirajući modularnu aritmetiku):

```
for (int i = s[b]; i >= 0; i--) {  
    for (int j = s[a]; j >= 1; j--) {  
        dp[b][i + j] += dp[b][i] * dp[a][j];  
    }  
}
```

Ovo spajanje stanja možemo zamisliti kao biranje nekih od s_a čvorova iz podstabla od a te spajajući ih sa svim mogućim brojem čvorova u podstablu od b . Primijetimo da je vremenska složenost ovog prijelaza jednaka $O(s_b \cdot s_a)$.

Izračunajmo vremensku složenost računanja stanja dinamika ako fiksiramo grad P . Iako se čini da je vremenska složenost računanja stanja $O(n^3)$, ona je zapravo $O(n^2)$. Za dokaze složenosti pogledajte 7. odjeljak u [blogu](#) te rješenje zadatka [Džumbus](#). Ukupna vremenska složenost ovog rješenja je $O(n^3)$ jer je složenost rješenja $O(n^2)$ po fiksiranom glavnom gradu. Ovo rješenje je bilo dovoljno brzo za riješiti prva 3 podzadatka.

Zadnji podzadatak rješavamo koristeći tehniku dp reroot-a. Primijetimo da "prebacivanjem" glavnog grada P na neko njegovo dijete a mijenja samo vrijednosti stanja u čvorovima P i a . Umjesto spajanja podstabla djeteta na roditelja, sada ga trebamo odrezati te roditelja spojiti na podstablo djeteta.

Neka je s_a veličina podstabla djeteta na koje ćemo prebaciti glavni grad P . Tada je $s_b = n - s_a$. Vremensku složenost "novih" spajanja i rezanja možemo promatrati na sličan način kao i vremensku složenost u prethodnom primjeru. Postupak spajanja identičan je kao i u prethodno napisanom odjeljku koda, a postupak rezanja postiže se obrnutim izvršavanjem operacija spajanja (pogledati kod za implementacijske detalje). Složenosti oba postupka su $O(s_b \cdot s_a)$ tj. $O((n - s_a) \cdot s_a)$.

Primijetimo da je za brid koji spaja čvorove P i a vrijednost $(n - s_a) \cdot s_a$ jednaka broju puteva u stablu koji prolaze tim bridom. Tada je ukupna složenost opisanog rješenja jednaka sumi udaljenosti svih uređenih parova čvorova stabla. Iz uvjeta zadatka znamo da je udaljenost bilo koja dva čvora manja ili jednaka od 36 (definirajmo ovu vrijednost kao d). Slijedi da je ukupna vremenska složenost rješenja jednaka $O(n^2 \cdot d)$.



Zadatak Ravnalo

Pripremio: Jakov Celin

Potrebno znanje: rad s razlomcima

Trebamo izračunati najmanji broj dužina da nacrtamo zid opisan u zadatku.

Primjetimo da najmanji broj dužina crtamo jednostavnom strategijom: sve dužine iz prethodnog stupca koje se mogu produžiti, produžimo u trenutni stupac, a ostale započnemo crtati s početkom u trenutnom stupcu.

Neka je a_i visina i -tog stupca, a b_i broj dijelova na koje je on podijeljen. Tada se i -ti stupac sastoji od $b_i + 1$ dužina gdje se j -ta ($0 \leq j \leq b_i$) nalazi na visini $\frac{a_i}{b_i} \cdot j$ tj. razmak u visinama uzastopnih dužina u stupcu je $\frac{a_i}{b_i}$. Definirajmo $c_i = \frac{a_i}{b_i}$.

Sada nam preostaje pronaći broj dužina koje možemo produžiti između neka 2 uzastopna stupca zida (nazovimo ih i i $i + 1$) pomoću razlomaka c_i i c_{i+1} . Jedan od načina pronalaska presjeka dužina je pronaći "najmanji zajednički višekratnik" od 2 razlomka uzimajući omjer najmanjeg zajedničkog višekratnika brojnika i najvećeg zajedničkog djelitelja nazivnika. Drugi način bio je postaviti razlomke c_i i c_{i+1} u omjer, izraziti dužine koje čine presjek pomoću tog omjera te izračunati njihov broj.

Oba načina računaju broj dužina u presjeku u $O(\log M)$ gdje M predstavlja najveću vrijednost u nizovima a i b . Ukupna vremenska složenost rješenja je $O(n \log M)$.