



Opisi algoritama

Zadatke, testne primjere i rješenja pripremili: Ivan Janjić, Sofija Velkovska i Jakov Celin.

Primjeri implementiranih rješenja dani su u priloženim izvornim kodovima.

Zadatak Kvadrat i kub

Pripremila: Sofija Velkovska

Potrebno znanje: naredbe učitavanja i ispisivanja

Najprije učitamo broj x . Kvadrat broja x je po definiciji jednak $x \cdot x$, pa ispišemo $x \cdot x$. Kub broja x jednak je $x \cdot x \cdot x$ - ispišemo $x \cdot x \cdot x$ u drugom retku.

Programski kod (pisan u Python 3):

```
x = int(input())
```

```
print(x * x)
```

```
print(x * x * x)
```

Zadatak Lozinka

Pripremila: Sofija Velkovska

Potrebno znanje: naredba odlučivanja (if)

Na početku primijetimo da je n najviše 19 (pogledajte odjeljak "Ulazni podaci" u tekstu zadatka).

Nadalje, primijetimo da prvih 9 prirodnih brojeva sastavljenih od samo jedne znamenke su redom 1, 2, 3, 4, 5, 6, 7, 8 i 9. Dakle, ako je n između 1 i 9, Martinina lozinka bit će upravo broj n .

Sljedeće trebamo promatrati dvoznamenkaste brojeve sastavljene od samo jedne znamenke - to su brojevi 11, 22, 33, 44, 55, 66, 77, 88 i 99. Primijetimo da te brojeve možemo dobiti kao 11 pomnožen nekim brojem ($11 = 1 \cdot 11, 22 = 2 \cdot 11, \dots, 99 = 9 \cdot 11$). Za 10. broj sastavljen od samo jedne znamenke imamo $1 \cdot 11$, za 11. broj $2 \cdot 11$, ..., za 18. broj $9 \cdot 11$. S jednom formulom to možemo iskazati kao $(n - 9) \cdot 11$.

Ostaje nam još situacija kad je $n = 19$. Nakon 99 sljedeći broj sastavljen od samo jedne znamenke je 111 - ispišemo ga jer je taj broj upravo Martinina lozinka.

Programski kod (pisan u Python 3):

```
n = int(input())
```

```
if n <= 9:
```

```
    print(n)
```

```
if n > 9 and n <= 18:
```

```
    print((n - 9) * 11)
```

```
if n == 19:
```

```
    print(111)
```



Zadatak Igra

Pripremila: Sofija Velkovska

Potrebno znanje: naredba odlučivanja (if)

Ako je $a = b$, očito ne trebamo raditi nikakve operacije jer je a već jednak b . Dakle, potrebno je 0 operacija da se a pretvori u b .

Ako je $a < b$, možemo povećati a za prirodan broj $b - a$. Dobit ćemo $a = b$ nakon jedne operacije.

Ako je $a > b$, možemo a pretvoriti u b koristeći samo jednu operaciju ako postoji neki prirodan broj k takav da k točno dijeli a i vrijedi $\frac{a}{k} = b$, tj. $a = b \cdot k$. Takav broj postoji samo ako b točno dijeli a - to možemo provjeriti koristeći modulo operator.

Ako ne vrijedi nijedan od ovih uvjeta (tj. $a > b$ i b ne dijeli a), tada su nam potrebne dvije operacije da a pretvorimo u b . Svaki broj dijeli sam sebe pa možemo podijeliti a s a i dobiti broj 1. Zatim povećamo 1 za $b - 1$ i dobijemo broj b . Tako smo iskoristili dvije operacije da a pretvorimo u b .

Programski kod (pisan u Python 3):

```
a, b = map(int, input().split())

if a == b:
    print(0)
elif a < b or a % b == 0:
    print(1)
else:
    print(2)
```

Zadatak Čokolada

Pripremila: Sofija Velkovska

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

Pretpostavimo da se dvije susjedne kockice u istom retku razlikuju po boji, tj. jedna je crna, a druga bijela. Neka su to j -ta i $(j + 1)$ -va kockica u i -tom retku. Luka će sigurno morati napraviti rez između j -tog i $(j + 1)$ -vog stupca. U suprotnom, te će kockice na kraju postupka rezanja biti u istom komadu čokolade, što nije dopušteno jer je jedna kockica crna, a druga bijela.

Ako postoji još jedan redak, označimo ga s k , u kojem se j -ta i $(j + 1)$ -va kockica razlikuju po boji, one će biti razdvojene rezom koji smo zaključili da moramo napraviti (rez koji razdvaja j -tu i $(j + 1)$ -vu kockicu u i -tom retku čokolade). Dakle, Luka može jednim rezom riješiti sve susjedne različito obojene kockice u j -tom i $(j + 1)$ -vom stupcu.

Preciznije, ako su j -ta i $(j + 1)$ -va kockica u svim retcima po parovima iste boje, Luka ne mora napraviti rez između j -tog i $(j + 1)$ -vog stupca. Ako postoji barem jedan redak u kojem se njihove boje razlikuju, potrebno je napraviti rez.

Slično vrijedi i za vodoravne rezove koje Luka može napraviti (u tom slučaju promatramo j -tu kockicu u i -tom i $(i + 1)$ -tom retku). Lako je vidjeti da nakon izvođenja svih rezova nijedne dvije kockice različite boje neće biti u istom pravokutnom komadu.

Kako bismo izračunali minimalan broj rezova potrebnih za razdvajanje crnih i bijelih kockica, potrebno je provjeriti opisani uvjet za sve parove susjednih stupaca i redaka te pratiti ukupni broj potrebnih rezova.



Zadatak Džeparac

Pripremio: Jakov Celin

Potrebno znanje: dinamičko programiranje, brzo potenciranje

Zadatak se rješava pronalaskom formule na razne načine (formalnim dokazima, intuicijom...) pa ćemo opisati jedan od njih.

Označimo s $r = N - k$ iznos koji mama Antonija raspodjeljuje sinovima. Budući da svaki sin mora primiti isti ukupni iznos, r mora biti djeljivo s 2. Uvedimo $m = r/2$ - ukupan iznos koji prima svaki sin. Raspodjela iznosa m po danima odgovara *kompoziciji* broja m , tj. odabiru uređenog niza pozitivnih cijelih brojeva $(x_1, x_2, \dots, x_d)$ čiji je zbroj točno m . Broj takvih kompozicija za fiksni $m \geq 1$ jednak je 2^{m-1} , jer svaki od $m - 1$ razmaka između jediničnih dijelova iznosa može biti ili unutar istog dana ili na granici između dana.

Ukupan broj rasporeda stoga je

$$1 + \sum_{\substack{r \in \{2, 4, \dots\} \\ r \leq N}} 2^{r/2-1},$$

gdje prvi pribrojnik odgovara slučaju $k = N$, $d = 0$, a suma prolazi po svim parnim vrijednostima ostatka r . Supstitucijom $j = r/2$, gdje j ide od 1 do $\lfloor N/2 \rfloor$, dobivamo

$$1 + \sum_{j=1}^{\lfloor N/2 \rfloor} 2^{j-1} = 1 + (2^{\lfloor N/2 \rfloor} - 1) = 2^{\lfloor N/2 \rfloor},$$

gdje je iskorištena formula za zbroj geometrijskog niza $\sum_{j=1}^t 2^{j-1} = 2^t - 1$.

Konačan odgovor je dakle $2^{\lfloor N/2 \rfloor} \bmod (10^9 + 7)$. Budući da je $N \leq 10^{18}$, eksponent $\lfloor N/2 \rfloor \leq 5 \cdot 10^{17}$, pa je potrebno rješenje računati brzim potenciranjem u vremenu $O(\log N)$.

Zadatak Prepisivanje

Pripremio: Jakov Celin

Potrebno znanje: maksimalno bipartitno sparivanje, najveći nezavisni skup u bipartitnom grafu, Königov teorem

Uvjet zadatka kaže da nijedan učenik ne smije imati susjeda (u četiri smjera) koji je također učenik. Tražimo dakle najveći skup mjesta na kojima mogu sjediti učenici (uzorni ili vragolasti) takav da nikoja dva odabrana mjesta nisu susjedna - što je upravo definicija *najvećeg nezavisnog skupa* na nekom grafu.

Konstruiramo graf na sljedeći način. Čvorovi grafa su sva slobodna mjesta (polja s vrijednošću 0) koja **nisu** susjedna uzornim učenicima (poljima s vrijednošću 2). Brid između dva čvora postoji ako i samo ako odgovarajuća polja dijele stranicu. Polja s vrijednošću 1 zanemarujemo u potpunosti - na njima nitko ne može sjediti i ne utječu na uvjet prepisivanja. Broj uzornih učenika je fiksni i svi moraju biti uključeni u rješenje.

Graf susjednosti mreže polja prirodno je *bipartitan*: obojimo li polja matrice u šahovskom uzorku (crno-bijelo prema parnosti $i + j$), tada svaki brid u mreži spaja polje jedne boje s poljem druge. Polja s uzornim učenicima tretiramo kao unaprijed odabrane čvorove i uklanjamo sva susjedna prazna polja (ona ne mogu biti odabrana). Na preostalom grafu (koji se sastoji samo od preostalih slobodnih polja) tražimo najveći nezavisni skup.

Najveći nezavisni skup u bipartitnom grafu može se odrediti kombinacijom dvaju klasičnih zaključaka. Prvi od njih tj. Königov teorem kaže da je veličina *minimalnog pokrivanja vrhovima* (eng. *minimum vertex cover*) jednaka veličini *maksimalnog sparivanja* (eng. *maximum matching*). Drugi jest da je nezavisni skup komplement pokrivanja vrhovima, vrijedi

$$|\text{najveći nezavisni skup}| = |V| - |\text{minimalni pokrivač}| = |V| - |\text{maksimalno sparivanje}|,$$

gdje je $|V|$ broj čvorova grafa (preostalih slobodnih polja). Maksimalno bipartitno sparivanje pronalazi



se standardnim algoritmom koji na grafu s $|V| \leq n \cdot m \leq 6400$ čvorova i $O(nm)$ bridova radi unutar vremenskog ograničenja u složenosti $O(V * B)$ gdje je $B \leq 4 * nm$ broj bridova u bipartitnom grafu.

Konačan odgovor jest broj uzornih učenika (sva polja s vrijednošću 2) uvećan za veličinu najvećeg nezavisnog skupa na grafu preostalih slobodnih polja.

Zadatak Skijanje

Pripremio: Jakov Celin

Potrebno znanje: Li-Chao stablo, monotoni deque

Prije čitanja opisa ovog rješenja preporuča se pomno proučiti sljedeće blogove koji detaljno opisuju ključne tehnike korištene u rješenju:

- [Li Chao i Convex hull optimizacija](#)
- [Monotoni deque](#)

Svaka staza u stablu dolazi do točno jednog čvora $v > 1$, pa se spust od najviše k uzastopnih staza jednoznačno opisuje odabirom *prvog* čvora f i *zadnjeg* čvora l na spustu, gdje je f predak od l na udaljenosti između 1 i k (udaljenost 1 odgovara spustu od jedne staze). Uvedimo i prefik sume brzina duž puta od korijena: neka je $S(v) = \sum_u \text{pravi predak } v b_u$, s konvencijom $S(\text{korijen}) = 0$. Tada je zbroj brzina svih staza na spustu od f do l jednak

$$\sum_{\text{staze na spustu}} b_i = S(l) + b_l - S(f).$$

Hektičnost spusta može se zapisati kao

$$\begin{aligned} H(f, l) &= z_l \cdot (z_l + S(l) + b_l - S(f)) + z_f^2 \\ &= z_l^2 + z_l \cdot (S(l) + b_l) - z_l \cdot S(f) + z_f^2. \end{aligned}$$

Za fiksni čvor l , prve dvije komponente su konstante koje ne ovise o f , pa maksimiziranje $H(f, l)$ po svim dopuštenim precima f svodi se na maksimiziranje izraza

$$-z_l \cdot S(f) + z_f^2.$$

Svakom čvoru v pridružimo vektor $\mathbf{p}(v) = (S(v), -z_v^2)$, a upitni vektor za l definiramo kao $\mathbf{q}(l) = (z_l, 1)$. Vrijednost koju maksimiziramo tada je točno (uz promjenu predznaka)

$$-\mathbf{p}(f) \cdot \mathbf{q}(l) = -(S(f) \cdot z_l + (-z_f^2) \cdot 1) = -z_l \cdot S(f) + z_f^2,$$

što problem svodi na: za svaki čvor l , pronaći minimum skalarnog produkta $\mathbf{p}(f) \cdot \mathbf{q}(l)$ po skupu predaka f od l koji se nalaze na udaljenosti između 0 i $k - 1$ razina, te rezultat negirati.

Primijetimo da za fiksni l upitni vektor ima oblik $(z_l, 1)$, pa je

$$\mathbf{p}(f) \cdot \mathbf{q}(l) = S(f) \cdot z_l + (-z_f^2) \cdot 1.$$

Svaki vektor $\mathbf{p}(f)$ možemo interpretirati kao pravac $y = S(f) \cdot x + (-z_f^2)$, a upit u točki $x = z_l$ traži minimum vrijednosti tog pravca. Ovo je klasični problem koji *Li Chao stablo* rješava te na takve upite u vremenu $O(\log M)$, gdje je M raspon vrijednosti z_v , tj. $M = 10^5$.



Skup dopuštenih predaka mijenja se dok DFS obilazak stabla prolazi čvorovima: pri ulasku u čvor l u skup ulazi l sam (kao potencijalni f za potomke), a čvor koji je $k + 1$ razina iznad l ispada. Pri izlasku iz l taj čvor se opet vraća skup. Dakle, skup predaka kojima upravljamo treba podržavati operacije ubacivanja i izbacivanja s oba kraja, pri čemu svaka operacija mora ažurirati Li Chao stablo. Budući da Li Chao stablo **ne** podržava brisanje, koristi se monotoni Deque s Li Chao stablima koja podržavaju poništavanje **posljednjeg** dodavanja (rollback). Svaki element se kroz cijeli DFS doda i ukloni $O(1)$ puta, a svaka operacija dodavanja i brisanja ima vremensku složenost $O(\log M)$ zbog Li Chao stabla. Globalni odgovor dobiva se uzimanjem maksimuma po svim čvorovima l nad vrijednostima

$$\text{ans} \leftarrow \max \left(z_l^2 + z_l \cdot (S(l) + b_l) + \max_f (-z_l \cdot S(f) + z_f^2) \right).$$

Ukupna vremenska složenost algoritma je $O(N \log M)$, a prostorna $O(N + M)$.

Zadatak Učionica

Pripremila: Sofija Velkovska

Potrebno znanje: binarno pretraživanje

Konstruirajmo matricu *maxstupac* tako da je *maxstupac* _{i,j} jednak maksimumu svih $a_{l,j}$ za koje vrijedi $l \leq i$. Osoba visine h može sjesti na sjedalo $(i + 1, j)$ ako je sjedalo prazno ($a_{i+1,j} = 0$) i ako vrijedi $h > \text{maxstupac}_{i,j}$. U prvom retku osoba može sjesti na sjedalo $(1, j)$ ako je $a_{1,j} = 0$.

Za prvi podzadatak prolazimo kroz sve parove susjednih sjedala (i, j) i $(i, j + 1)$ te provjeravamo zadovoljava li uvjete neka od sljedeće dvije mogućnosti: prijatelj visine h_1 sjedi na (i, j) , a prijatelj visine h_2 na $(i, j + 1)$, ili obratno.

Za drugi podzadatak promatramo sve uzastopne skupove od k sjedala. Pretpostavimo da su sva ta sjedala prazna. Optimalno je više prijatelje smjestiti u stupce u kojima je najviši učenik ispred njih viši, a niže prijatelje iza nižih učenika. Stoga, za sjedala $(i, j), (i, j + 1), \dots, (i, j + k - 1)$ sortiramo vrijednosti *maxstupac* _{i,j} , *maxstupac* _{$i,j+1$} , ..., *maxstupac* _{$i,j+k-1$} i dodjeljujemo sjedala prijateljima prema tom poretku: najviši prijatelj dobiva stupac s najvećom vrijednošću, drugi najviši sljedeći itd. Zatim provjeravamo je li svaki prijatelj viši od najviše osobe ispred sebe. Ovo rješenje radi u vremenu $O(nmk \log k)$.

Sada zanemarimo zauzetost sjedala i fokusirajmo se samo na uvjete vezane uz visinu. Ako sjedala koja počinju na (i, j) čine pogodan skup, tada sjedala koja počinju na (l, j) za svaki $1 \leq l \leq i$ također čine pogodan skup. Dakle, za fiksni j postoji prag t takav da su redci 1 do t pogodni, dok redci $t + 1$ do n nisu. Vrijednost t možemo pronaći binarnim pretraživanjem. Nakon toga provjeravamo koji od tih pogodnih skupova sjedala imaju svih k sjedala prazna. Koristeći tehniku sliding window, to možemo napraviti u $O(1)$ po retku, što daje složenost $O(mk \log n \log k)$.

Za zadnji podzadatak uklanjamo faktor $\log k$. Promotrimo k vrijednosti *maxstupac* $ms_1, ms_2, \dots, ms_k$ i visine prijatelja $h_1, h_2, \dots, h_k$. Trebamo provjeriti vrijedi li nakon sortiranja da je svaka vrijednost ms manja od odgovarajuće vrijednosti h . To vrijedi ako postoji jedna vrijednost ms koja je manja od svih visina, druga koja je manja od barem $k - 1$ visina, treća koja je manja od barem $k - 2$ visina itd.

Prije glavnog algoritma za svaki (i, j) unaprijed izračunamo koliko je visina prijatelja veća od *maxstupac* _{i,j} . To možemo napraviti u vremenu $O(n^2)$. Sad kad znamo te vrijednosti, lako možemo provjeriti postoji li barem jedna vrijednost veća ili jednaka k , druga veća ili jednaka $k - 1$ itd., koristeći pomoćno polje. Budući da ova provjera traje $O(k)$ vremena, konačna vremenska složenost iznosi $O(mk \log n)$.