

Sokić

```
n = int(input())
x = int(input())
print(n // x)
```

Umnožak

```
n = int(input())

def umn(x):
    u = 1
    for z in str(x):
        u *= int(z)
    return u

i = 0
for i in range(1, 100000):
    if umn(i) == n:
        print(i)
        break
else:
    print(-1)
```

Šašave operacije

```
n = int(input().strip())
expr = input().strip()

groups = expr.split('*')

values = []
for g in groups:
    parts = g.split('+')
    total = sum(map(int, parts))
```

```
values.append(total)

result = 1
for v in values:
    result *= v

print(result)
```

Permutacija

-1- Strategija za prvu grupu se svodi na dva koraka. Prvo je potrebno postaviti $n - 1$ upita tipa "1 x" za $x \in [2, n]$. Sada dobivamo niz odgovora b_i u kojem zasigurno postoji maksimum mx te se on pojavljuje najviše 2 puta.

Sada možemo biti sigurni da su minimum i maksimum na pozicijama/poziciji gdje je $b_i = mx$. Pošto imamo 2 pokušaja, možemo fiksirati neki od njih npr. y kao minimum.

Sada je samo potrebno još postaviti n upita oblika "y x" za $x \in [1, n]$. Ne zaboravite da je y mjesto minimuma te u slučaju minimuma (1) odgovori će biti od 0 do $n - 1$ i svi će se pojaviti jednom, jednako će biti i za maksimum.

Nadalje, brojevi su jednoznačno definirani razlikom od minimuma/maksimuma te za oba slučaja možemo konstruirati permutaciju. (ako je razlika k onda je na tom mjestu $k + 1$ ili $n - k$ ovisno o tome je li gledamo minimum ili maksimum.)

-2- Prvi korak strategije ostaje isti te je potrebno pronaći y .

Za izmjenu 2. koraka potrebno je primijetiti da se u listi odgovora b svaki broj može najviše 2 puta pojaviti. To proizlazi iz toga da ako je na 1. mjestu a_1 , on može imati razliku b_i s nekim brojem ako i samo ako vrijedi $|a_i - a_1| = b_i$ i to se može dogoditi samo za brojeve $(a_1 + b_i)$ i $(a_1 - b_i)$, označimo ta 2 kandidata kao par, a one koji nemaju 2 ponavljanja u b kao jedince.

Nadalje treba primijetiti da se prvo pojavljuju parovi, a kako b_i raste u vrijednosti, bit će samo jedinci. Jedince obrađujemo na način da onaj s najvećom vrijednošću bude minimum te onaj s idućom najvećom bude min+1. Takvo obrađivanje funkcionira uvijek kada je u pitanju jedinac.

U slučaju para, nismo sigurni o kojem se broju radi te je potrebno dodatno ispitati koji je koji tako da saznamo razliku između 1. iz para i y . Recimo da dobijemo odgovor k , tada je na tom mjestu

broj $k + 1$ te je drugi u paru jednak $2a_1 - (k + 1)$.

Primijetite da parova najviše može biti $\lfloor n/2 \rfloor$ te to zadovoljava broj upita s $n - 1$ upita u 1. koraku i $\lfloor n/2 \rfloor$ u 2. koraku.

Oba rješenja rade u $O(n)$.

Bitstring

Za prvi podzadatak rješenje se dobije potpunim pretraživanjem. Nakon svakog upita tipa 2 izdvojimo sve pozicije s nulom i isprobamo sve podskupove tih pozicija veličine najviše k . Za svaki takav izbor privremeno ih promijenimo u jedinice i prebrojimo susjedne parove jedinica. Maksimum po svim mogućnostima je odgovor. Upiti tipa 1 se izravno simuliraju. Složenost je $O(2^n \cdot n)$ po upitu tipa 2.

Za sljedeće podzadatke koristi se pohlepna ideja s “rupama”, tj. nizovima nula između dviju jedinica. Ako je rupa duljine g , s g promjena može se potpuno ispuniti i spojiti dvije skupine jedinica, čime se dobiva jedan dodatni susjedni par. Optimalno je uvijek prvo popunjavati kraće rupe.

Za podzadatke ($n, q \leq 5000$) nakon svakog upita tipa 2 možemo u $O(n)$ izračunati sve rupe, sortirati ih po duljini i pohlepno ih popunjavati dok ima budžeta k . Upiti tipa 1 su $O(1)$, a upiti tipa 2 $O(n \log n)$.

Za zadnji podzadatak sve se održava dinamički. Drži se skup pozicija jedinica, trenutni broj susjednih parova i dvije Fenwickove strukture: jedna broji koliko ima rupa određene duljine, a druga zbroj njihovih duljina. Upit tipa 1 mijenja najviše dvije susjedne veze i najviše tri rupe, pa se sve ažurira u $O(\log n)$.

Kod upita tipa 2 uzme se $t = \min(k, \text{brojnula})$. Cilj je odrediti koliko se rupa može potpuno ispuniti s budžetom t . To se radi binarnim pretraživanjem po duljini rupa: traži se najveća duljina idx takva da je zbroj duljina svih rupa duljine $\leq idx$ manji ili jednak t . Taj zbroj i broj rupa dobivaju se iz Fenwickovih stabala. Nakon toga se eventualno djelomično popuni još rupa duljine $idx + 1$, koliko to preostali budžet dopušta. Ukupna složenost po upitu uz naivnu implementaciju je $O(\log^2 n)$ što je dovoljno brzo za ograničenja iz zadatka. Uz pažljivu implementaciju može se postići složenost od $O(\log n)$

(<https://codeforces.com/blog/entry/61364>).

Ključna je interpretacija uvjeta zadatka u terminima linearne algebre nad XOR-om. Ako za segmente označimo njihove XOR vrijednosti s x_1, x_2, \dots, x_k , tada zahtjev da nijedan neprazan podskup nema XOR jednak nuli znači da su ti brojevi linearno nezavisni nad binarnim poljem. Drugim riječima, nijedan XOR nekog podskupa segmenata ne smije se moći dobiti kao XOR ostalih, odnosno ne smije postojati netrivialna kombinacija koja daje nulu.

Prvo primijetimo nužan uvjet: XOR svih elemenata svitka mora biti različit od nule. Ako je ukupni XOR jednak nuli, tada bez obzira na podjelu, XOR svih segmenata zajedno bit će nula, što je zabranjeni neprazan podskup, pa rješenje ne postoji.

Pretpostavimo dalje da ukupni XOR nije nula. Svitak čitamo slijeva nadesno i gradimo segmente pohlepno. Održavamo trenutni XOR *cur* koji predstavlja XOR elemenata od posljednjeg reza do trenutne pozicije. Istovremeno održavamo bazu već dobivenih segmenata, odnosno skup njihovih XOR vrijednosti, tako da baza uvijek sadrži linearno nezavisne elemente.

Kako prolazimo niz, svaki put kad proširimo trenutni segment, ažuriramo *cur*. Čim se dogodi da je *cur* linearno nezavisan u odnosu na sve prethodne segmente, možemo sigurno napraviti rez na toj poziciji. Time dobivamo novi segment čiji XOR povećava rang baze, pa ne može sudjelovati ni u jednom nepravaznom podskupu koji daje nulu. Nakon reza, *cur* vraćamo na nulu i nastavljamo graditi sljedeći segment.

Ako se dogodi da je *cur* linearnom kombinacijom već postojećih segmenata, tada rez na tom mjestu ne smijemo napraviti jer bi novi segment bio ovisan, pa bi postojao neprazan podskup segmenata s XOR-om jednakim nuli. U tom slučaju samo nastavljamo širiti trenutni segment.

Pohlepnost ovog postupka je ispravna jer svaki put kada možemo povećati rang baze, to vodi prema većem broju segmenata, a kasnije takvu priliku ne bismo mogli nadoknaditi bez narušavanja nezavisnosti. Na kraju prolaska dobijemo maksimalan broj segmenata s međusobno linearnom nezavisnim XOR vrijednostima, što je točno ono što zadatak traži.

Linearno nezavisnost provjeravamo pomoću XOR baze, u koju svaki novi kandidat pokušavamo umetnuti standardnim postupkom eliminacije po bitovima. Svaki pokušaj umetanja traje $O(\log A)$, gdje je A maksimalna vrijednost elemenata.

Ukupna vremenska složenost algoritma je $O(n \log A)$. Ako je ukupni XOR jednak nuli, ispisujemo -1 , inače ispisujemo broj dobivenih segmenata.

Daske

Rješenje se može promatrati geometrijski. Svaki stup odgovara točki u ravнини, a daska koja spaja stupove dopuštena je ako pravac između tih točaka leži iznad svih stupova između njih. Na trenutak ignorirajmo uvjet da paketi mogu klizati niz daske, odnosno samo se trebamo pobrinuti da je cijela dužina prekrivena s daskama. U ovom slučaju, potrebno je uvidjeti da će svi stupovi koji se nalaze na gornjoj konveksnoj ljusci svih stupova morati biti krajnje točke neke daske (stup koji je na gornjoj konveksnoj ljusci ne može biti nadkriven daskom, jer bi to značilo da nije na konveksnoj ljusci). Iz toga slijedi da je minimalan broj potrebnih daski jednak broju bridova gornje konveksne ljuske.

Vratimo se sada na cijeli problem. Postoje nužni uvjeti nemogućnosti: ako je $h_0 < h_1$, novac koji padne između prva dva stupa ne može se zaustaviti te analogno $h_{n-2} > h_{n-1}$ za kraj niza. U tim slučajevima rješenje ne postoji.

U suprotnom, prva i posljednja daska nam osiguravaju da paket ne može isklizati s ruba, što znači da nakon što smo odabrali prvu i posljednju dasku, ostatak daski mora biti konveksna ljuska između desnog kraja prve daske (nazovimo taj stup l) i lijevog kraja posljednje daske (nazovimo taj stup r). Ostaje još pitanje kako odabrati l i r .

Riješit ćemo problem kako odabrati l , jer je slučaj za r analogan. Neka je s pozicija prvog stupa koji je viši od h_0 . Znamo da je $0 < l < s$. Tvrdimo da je za l optimalno uzeti najlijeviji stup koji se može direktno povezati sa stupom s . Ova pohlepna ideja radi jer ako se stupovi na pozicijama l_1 i l_2 ($l_1 < l_2$) oba mogu povezati s stupom s , onda se l_1 može povezati s svakim stupom desno od s s kojim se može povezati l_2 (jer će daska od l_1 uvijek biti potpuno iznad daske od l_2).

Pronaći l i r koji zadovoljavaju gornji uvjet nije komplicirano, jer se povezanost s s može jednostavno provjeriti ako prolazimo po stupovima od $s - 1$ prema 1 i pamtimo zadnji stup koji je bio povezan s s . Onda samo treba provjeriti siječe li taj zadnji stup dasku između trenutnog stupa i s .

Na kraju je potrebno pronaći gornju konveksnu ljusku između l i r i izbrojati broj bridova.