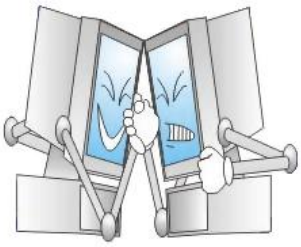


Prva runda kvalifikacija za Okružno takmičenje, 2013. godine
Analiza problema sa rešenjima

Počev od ove godine, uveden je novi nivo takmičenja – Kvalifikacije, kao najniži u godišnjem ciklusu takmičenja. Same kvalifikacije se sastoje iz dve runde. Runde traju nedelju dana i sadrže po pet problema, od koji svaki može doneti po 100 bodova. Uslov za učešće na okružnom takmičenju je osvojenih 100 ili više bodova u sumi runda kvalifikacija (drugim rečima 100 od mogućih 1000 bodova).

Ovaj dokument sadrži analizu problema sa rešenjima sa prve kvalifikacione runde za okružno takmičenje 2013. godine. Ukoliko imate dodatnih komentara ili sugestija, slobodno se obratite takmičarskoj komisiji.

Komisija za organizaciju srednjoškolskih takmičenja iz pogramiranja
e- mail: tak.prog@gmail.com
sajt: takprog.dms.rs

Problem 1. KDelioci

Dat je prirodni broj N. Za prirodni broj $D > 0$ kažemo da je delioc broj N ukoliko D deli N. Kolika je aritmetička sredina svih delioca datog broj N?

Aritmetička sredina brojeva $D_1, D_2, \dots, D_m$ jednaka je: $\frac{D_1+D_2+\cdots+D_m}{m}$.

Ulaz. Prvi i jedini red standardnog ulaza sadrži prirodni broj $N(1 \leq N \leq 10^9)$.

Izlaz. U prvi i jedini red standardnog izlaza ispisati aritmetičku sredinu deliloca datog prirodnog broja. Broj štampati sa tačnošću od dve decimale.

Primer

Ulaz	Izlaz
6	3.00

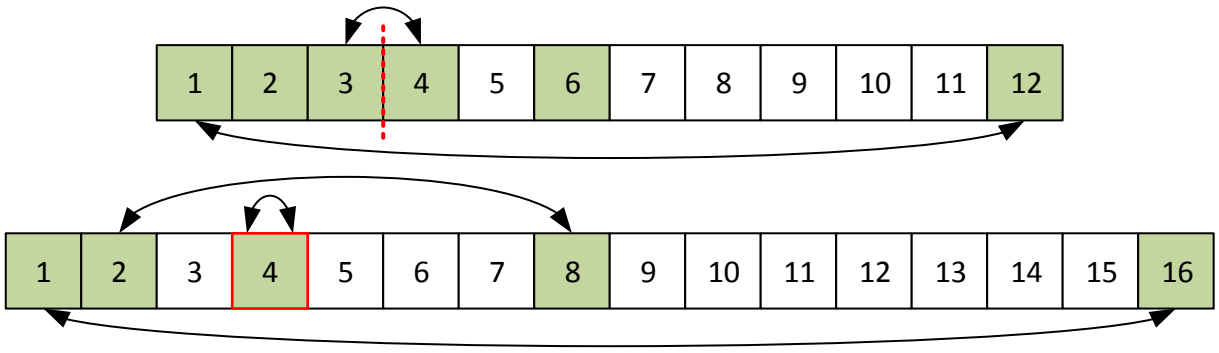
Objašnjenje primera. Delioci broj 6 su: $1, 2, 3$ i 6. Njihova aritmetička sredina je: $\frac{1+2+3+6}{4} = 3$.

Rešenje i analiza:

Prvi problem je bio najlakši problem na kvalifikacijama i predstavlja školski tip problema. Prosečan broj bodova osvojenih na ovom problemu bio je 70.77, dok je više od polovine takmičara imalo maksimalan skor.

Naivna implementacija koja za svaki prirodni broj od 1 od N ispituje da li deli broj N, nije dovoljno efikasna za ograničenja koja su data u samom problemu. Pristup jeste korektan ali je linearne složenosti. Takmičari čije je rešenje bazirano na ovaj ideji osvojili su 40 poena.

Za rešenje ovog problema koristi se slična ideja kao pri ispitivanje da li je dati broj prost. Naime, ukoliko želimo da ispitamo da li je prirodni broj N prost, dovoljno je da vidimo da li je deljiv nekim prirodnim brojem iz segmenta $[1, \sqrt{N}]$. Zašto je ovo dovoljno za proveru? Ukoliko broj D deli broj N, tada on ima svog "para" koji takođe deli broj N a to je $\frac{N}{D}$. Kako je $D \cdot \frac{N}{D} = N$, tada barem jedan od brojeva D ili $\frac{N}{D}$ manji ili jednak od korena broja N (u suprotnom bi proizvod bio veći od broja N). Kako barem jedan broj iz svakog para pripada datom segmentu, ovim pristupom ćemo sigurno pokriti sve delioce broja N.



Slika 1. Primer delioca broja 12 i broja 16 (popun kvadrat). Strelicama su označeni parovi delilaca, a crvenom brojm je označena granica segmenta odnosno $\sqrt{N}$.

Zbog gore navedene činjenice, možemo ispitati samo brojeve iz segmenta $[1, \sqrt{N}]$. U slučaju da neki od njih deli broj N, odmah znamo da je i broj $\frac{N}{D}$ delilac. Međutim postoji specijalan slučaj delioca – ukoliko je broj N potput kvadrat tj. $\sqrt{N}$ je ceo broj, tada je njegov par zapravo isti broj. Ovaj slučaj je potrebno posebno ispitati.

Pseudo kod opisanog rešenja:

```
================================================================================
   Ulaz: prirodni broj N
   Izlaz: aritmeticka sredina delilaca broj N

   01 suma = 0;
   02 num = 0;
   03 for D = 1 to √N do
   04     if D deli N then
   05            suma = suma + D + N / D;
   06            num = num + 2;
   07     endif
   08 endfor
   09 if (√N ceo broj) then
   10     suma = sum - √N;
   11     num = num - 1;
   12 endif
   13 return (suma / num);
================================================================================
```

Na kraju treba napomenuti da zbog velikih ograničenja ulaznih podataka treba koristiti realnu vrednost promenjive $suma$ i to sa što većom tačnošću – tip $doube$. Takmičari koji su koristili tipove kao što su $float$, za C++, odnosno $real$, za Pascal, nisu mogli da osvoje maksimalni broj bodova.

Problem 2. Menjanje Stringa

Dat je string S dužine N sastavljen od malih slova engleskog alfabeta. Nad njim se, redom, izvršava Q upita jednog od sledeća dva tipa:

$1\ i\ x$ - zameniti i-ti znak u trenutnom stringu znakom x ($1 \leq i \leq n$, x je malo slovo engleskog alfabeta).
2 - obrnuti trenutni string, tj. zameniti prvi i poslednji znak, drugi i pretposlednji itd.

Odrediti izgled stringa S posle svih upita.

Ulaz. Prvi red standardnog ulaza sadrži, redom, brojeve N i Q razdvojene razmakom ($1 \leq N, Q \leq 2 \cdot 10^5$). Naredni red sadrži string S dužine N sastavljen od malih slova engleskog alfabeta. Narednih Q redova sadrže upite u gore pomenutom obliku. Upiti se izvršavaju u datom redosledu.

Izlaz. U prvi i jedini red standardnog izlaza ispisati izgled stringa posle svih upita.

Primer

Ulaz	Izlaz
5 4 abcde 1 2 n 2 1 2 m 1 5 c	emcnc

Objašnjenje primera. String se menja na sledeći način: abcde → a**n**cde → edcna → e**m**cna → emcn**c.**

Napomena. U 50% test primera je $N \leq 100$.

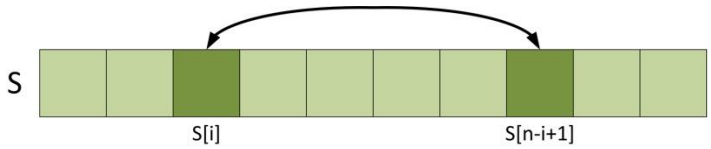
Rešenje i analiza:

Ovo je prilično standardan takmičarski problem gde jedna jednostavna ali lepa ideja daje elegantan kod i maksimalan broj poena. Po rezultatima takmičara, opravdao je svoje mesto drugog najlakšeg zadatka.

Kod mnogih problema sa upitima/operacijama/naredbama, najjednostavnije rešenje je direktna simulacija. Ovde je to vrlo jednostavno implementirati – upit $1\ i\ x$ je ekvivalentan dodeli $S[i] = x$, dok je za upit 2 (obrtanje stringa) dovoljno zameniti mesta karakterima na pozicijama i i $n - i + 1$ za svako $1 \leq i \leq \frac{n}{2}$. Međutim, iako je složenost upita tipa 1 konstantna, složenost obrtanja stringa je $O(n)$ po upitu što daje ukupnu složenost od $O(NQ)$ - ovo nije dovoljno efikasno. Opisana simulacija donosi 50 poena, što su takmičari mogli i očekivati na osnovu napomene.

Glavna ideja je primetiti da, iako moramo pratiti upite tipa 2 da bismo ispravno izvršavali upite tipa 1, direktna simulacija obrtanja stringa nije potrebna. Naime, prilikom izvršavanja upita tipa 2 string S naizmenično prelazi iz "početnog" u "obrnuto" stanje i iz obrnutog u početno stanje; proizvoljna pozicija i u stringu prilikom obrtanja prelazi u poziciju $n - i + 1$ dok se prilikom novog obrtanja vraća na isto mesto. Prema tome, ukoliko je string S u početnom stanju, upit $1\ i\ x$ ima značenje $S[i] = x$, dok ukoliko je string S u obrnutom stanju, upit $1\ i\ x$ znači da treba promeniti $(n - i + 1)$-vi karakter stringa, tj. $S[n - i + 1] = x$.

Opisani algoritam ne simulira obrtanje svaki put i radi u složenosti $O(1)$ po upitu što daje ukupnu složenost $O(N + Q)$ i osvaja sve poene. Za praćenje da li je S u početnom ili obrnutom stanju možemo koristiti standardno; u suprotnom štampamo ga unazad.



Slika 1. Pozicija i u obrnutom stringu odgovara poziciji $n - i + 1$ u početnom stringu.

Pseudo kod opisanog rešenja:

```
================================================================================
   Ulaz: string S dužine N i Q upita
   Izlaz: string S posle svih upita

   01 obrnut = false;
   02 for j = 1 to Q do
   03     if Q[j] naredba tipa 1 then
   04             if obrnut then
   05                     S[n − i + 1] = x;
   06             else
   07                     S[i] = x;
   08     endif
   09     else obrnut = not obrnut;
   10 endfor
   11 if obrnut then Obrni_String_S; // standardno obrtanje
   12 return S;

================================================================================
```

Napomenimo da su najčešće greške u Pascal-u bile deklarisanje S kao tip *string* (koji može da prihvati samo 255 karaktera dok S može imati do 200.000) a u C/C++-u deklaracija “char S[200000];” (koja ne ostavlja mesta za karakter ‘\0’ koji označava kraj stringa). Par rešenja sa tačnom idejom je ostalo bez maksimalnog broja poena zbog ovakvih grešaka.

Problem 3. Borovnice

Perica i Jovica sakupljaju gomile borovnica. N gomila borovnica je poređano u red. Perica sakuplja gomile s početka reda, a Jovica s kraja reda. Nakon što sakupe sve gomile, svako prebroji svoje borovnice.

Dato je Q upita. Svaki upit sadrži jedan broj K, a vi trebate da odgovorite da li je moguće da na kraju i Perica i Jovica imaju više od K borovnica.

Ulaz. Prvi red standardnog ulaza sadrži dva cela broja N i Q ($1 \leq N \leq 100.000, 1 \leq Q \leq 1.000.000$), broj gomila i broj upita, respektivno.
U drugom redu nalazi se N celih brojeva iz intervala $[1,\ 1.000.000]$, koji predstavljaju broj borovnica na svakoj od N gomila.
U sledećih Q redova nalazi se po jedan broj K ($1 \leq K \leq 1.000.000.000$).

Izlaz. Na standardan izlaz ispisati Q redova, koji predstavljaju odgovore na svaki od Q upita. Svaki red mora da sadrži jedan karakter: **d** - ako je moguće da obojica sakupe više od K borovnica, ili **n** - ako nije moguće da obojica sakupe više od K borovnica.

Primer 1

Ulaz	Izlaz
4 3 3 8 5 6 7 14 11	d n n

Primer 2

Ulaz	Izlaz
4 1 8 1 2 2 4	d

Objašnjenje primera 2. Ako Perica uzme prvu gomilu borovnica (8), a Jovica uzme preostale tri ($2 + 2 + 1 = 5$), obojica će na kraju imati više od 4 borovnice.

Rešenje i analiza:

Problem Borovnice je srednji zadatak po težini na ovim kvalifikacijama. Karakterističan je i po količini podataka na ulazu i izlazu, pa su takmičari morali da obrate pažnju da ne koriste spore funkcije prilikom baratanja sa ulazom i izlazom.

Neka je dati niz gomila označen sa $borovnice$. Problem, odnosno odgovor na određeni upit, se svodi na proveru da li postoji indeks p u nizu $borovnice$, za koji važi da je $\sum_{i=1}^{p} borovnice[i] > K \wedge \sum_{i=p+1}^{N} borovnice[i] > K$.

Jasno je da se jedna od ovih suma može neposredno izračunati ako smo prethodno izračunali ukupan broj borovnica B. Ako od ukupnog broja borovnica oduzmemo broj borovnica u jednom delu niza, dobićemo broj borovnica u drugom delu niza, tj. $\sum_{i=p+1}^{N} borovnice[i] = B - \sum_{i=1}^{p} borovnice[i]$. Ovo dovodi do naivnog

rešenja gde za svaki upit prolazimo kroz niz te tražimo indeks p, za koji važi da je $\sum_{i=1}^{p} borovnice[i] > K$, te zatim proveravamo da li važi $B - \sum_{i=1}^{p} borovnice[i] > K$. Prilikom ispitivanja nekog indeksa p, ukupan broj borovnica u prvom delu niza ne računamo prolazeći kroz sve gomile i sabirajući sve borovnice na gomilama od 1 do p, već na broj borovnica na gomilama od 1 do $p-1$ dodajemo broj borovnica na gomili p. To jest, $\sum_{i=1}^{p} borovnice[i] = \sum_{i=1}^{p-1} borovnice[i] + borovnice[p]$. Ovo rešenje ima složenost $O(QN)$ i donosi oko 50% bodova.

Rešenje koje donosi 80-100% bodova koristi binarnu pretragu. Prvo ćemo da prekalkulišemo niz $suma$, gde je $suma[p] = \sum_{i=1}^{p} borovnice[i]$, za $p = 1, 2, \dots, N$. Ovo radimo u linearnoj složenosti po N, na način opisan u prethodnom pasusu. Pošto je ovaj niz rastući, binarnom pretragom možemo da pronađemo najmanji indeks p, za koji važi da je $suma[p] > K$, a zatim proverimo da li je i $B - suma[p] > K$. Primetimo da se traži najmanji indeks, jer nema potrebe dodavati još borovnica u taj deo niza ako je njihov broj već veći od K, dok bismo dodavanjem samo smanjili broj borovnica u drugom delu niza. Pošto je složenost binarne pretrage $O(logN)$, a mi vršimo pretragu prilikom svakog upita, ukupna složenost je $O(N + QlogN)$.

Iako je prethodno rešenje sa pametnom implementacijom moglo da nam donese i maksimalan broj bodova, postoji još efikasnije rešenje. Primetimo da je uslov da su oba broja veća od K sigurno zadovoljen ako je manji broj veći K. Takođe, uslov nije zadovoljen ako manji broj nije veći od K. Ova činjenica dovodi do ideje da je potrebno maksimizovati broj borovnica u delu niza koji ima manje borovnica, tj. želimo da je manji broj što veći. Odnosno, mi tražimo takvu podelu niza, za koju važi da je $\min(P, J)$ maksimalno, gde je P broj borovnica u prvom delu niza, a J broj borovnica u drugom delu niza. Ovu podelu možemo da nađemo jednim prolazom kroz niz gomila, ispitujući svaku moguću podelu na način ranije opisan, i pamteći najbolju. Kad odredimo koji je to maksimalan broj, na svaki upit možemo da odgovorimo samo proverom da li je on veći od K. Pošto ovo rešenje ima složenost odgovaranja na upite $O(1)$, ukupna složenost algoritma je $O(N + Q)$.

Glavni deo koda ovog rešenja u C++:

```
scanf("%d %d", &n, &q);
for (int i = 0; i < n; i++)
{
    scanf("%d", &borovnice[i]);
    jovica += borovnice[i];
}
for (int i = 0; i < n; i++)
{
    perica += borovnice[i];
    jovica -= borovnice[i];
    maxBorovnica = max(maxBorovnica, min(perica, jovica));
}
for (int i = 0; i < q; i++)
{
    scanf("%d", &k);
    if (maxBorovnica > k)
        printf("d\n");
    else
        printf("n\n");
}
```

Problem 4. Razlika Stringova

Data su dva stringa - A i B ($length(A) > length(B)$) sastavljena od malih slova engleske abecede. Neki karakteri u stringu B su zamenjeni karakterom $'?'$.

Razlika izmedju dva stringa $S1$ i $S2$, iste dužine, je broj indeksa X tako da je $S1[X] \neq S2[X]$.

Ukupna razlika između stringova A i B je zbir razlika izmedju stringa B i svakog podstringa uzastopnih elemenata duzine $length(B)$ stringa A.

Primer:
Ukupna razlika izmedju stringova "$abataraba$" i "$baba$" je jednaka

$$razlika("baba","abat")\ [\mathbf{4}]\ +$$
$$razlika("baba","bata")\ [\mathbf{1}]\ +$$
$$razlika("baba","atar")\ [\mathbf{4}]\ +$$
$$razlika("baba","tara")\ [\mathbf{2}]\ +$$
$$razlika("baba","arab")\ [\mathbf{4}]\ +$$
$$razlika("baba","raba")\ [\mathbf{1}]\ =\ \mathbf{16}$$

Zameniti karaktere $'?'$ u stringu B tako da ukupna razlika bude najmanja moguća. Ukoliko ima više rešenja, ispisati leksikografski najmanje.

Ulaz. U prvom redu standardnog ulaza nalazi se string A. U drugom redu se nalazi string B. Duzina stringa A ce uvek biti veca od duzine stringa B.

Izlaz. U prvom redu standardnog izlaza ispisati string B nakon zamenjivanja karaktera $'?'$ tako da se dobija najmanja ukupna razlika izmedju stringova A i B. U drugom redu standardnog izlaza ispisati najmanju ukupnu razliku izmedju stringova A i B.

Primer

Ulaz	Izlaz
abbcd b?d	bbd 4
qwertytre d???z	detrz 20
abataraba ???	aaa 10

Objašnjenje 2. primera. Stringovi koji daju ukupnu razliku 20 su (leksikografski od najmanjeg ka najvećem): **detrz**, dettz, drtrz, drttz, dttrz, dtttz, dwtrz, dwttz, dytrz, dyttz.

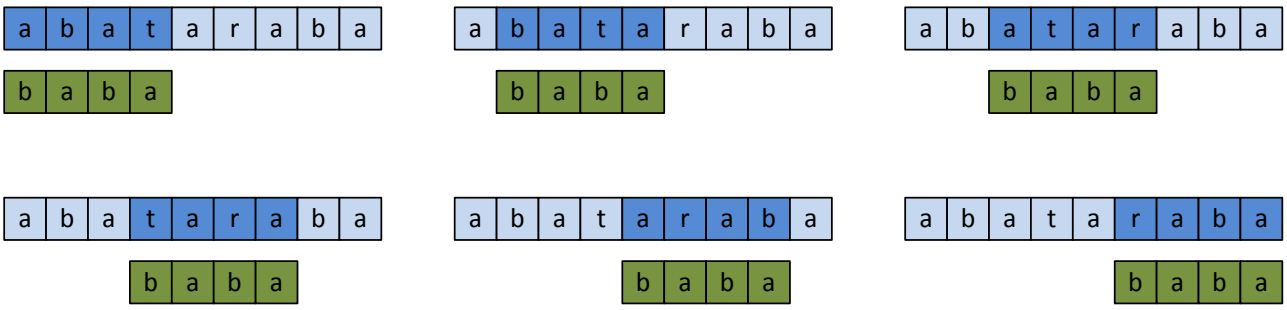
Ograničenja:
U 30% primera ce biti: $1 \leq length(B) < length\ (A) \leq 1.000$ i samo jedan karakter $'?'$ u stringu B.
U 50% primera ce biti: $1 \leq length(B) < length\ (A) \leq 1.000$
U 100% primera ce biti: $1 \leq length(B) < length\ (A) \leq 1.000.000$

Rešenje i analiza:

Ukoliko pogledamo ograničenja iz zadatka, vidimo da u 30% test primera u stringu B postoji samo jedan karakter '?', i dužine stringova A i B su najviše 1000. Jednostavno rešenje za ove slučajeve je da zamenimo '?' svakim slovom od 'a' – 'z', a ukupnu razliku izračunamo prolazeći kroz sve podstringove stringa A, dužine $length(B)$, upoređujući svaki karakter sa odgovarajućim karakterom stringa B. Iako je izgledalo kao lako rešenje, većina takmičara nije ni pokušala da ga uradi, pa zato dajemo pseudo kod ovog rešenja:

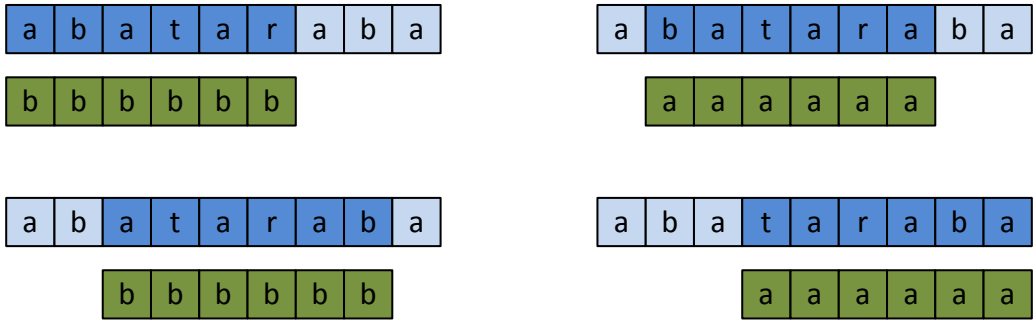
```
================================================================================
   Ulaz: Stringovi A i B
   Izlaz: Minimalna ukupna razlika i odgovarajuci string B

   01 min_razlika = +∞;
   01 for i = 1 to N do
   02     if (B[i] == '?') then
   03            for s = 'a' to 'z' do
   04                   B[i] = s;
   05                   razlika = ukupna_razlika(A,B);
   06                   if (razlika ≤ min_razlika) then
   07                          min_razlika = razlika;
   08                          resB = B;
   09                   endif
   10            endfor
   11     endif
   12 endfor
   13
   14 print resB, min_razlika


   Ukupna_razlika(string A, string B)

   01     ukupna_razlika = 0;
   02     for i = 1 to length(A)-length(B)+1 do
   03            for j = 1 to length(B) do
   04                   if (A[i+j-1] != B[j]) ukupna_razlika++;
   05            endfor
   06     endfor
   07
   08     return ukupna_razlika
================================================================================
```

Da bi došli do efikasnijeg rešenja, potrebno je da vidimo kako možemo brže da izračunamo ukupnu razliku između dva stringa. Posmatrajmo primer kada je $A = "abataraba"$ i $B = "baba"$.



Slika 1. Poredjenje podstringova kod računanja ukupne razlike za stringove A i B

Ono što je ovde bitno primetiti je sa kojim karakterima stringa A se upoređuju pojedinačni karakteri stringa B. Ovo možemo videti na sledećoj slici:



Slika 2. Prikaz sa kojim karakterima iz stringa A će se upoređivati svako pojedinačno slovo stringa B

Sada nije teško zaključiti da ćemo i-ti karakter stringa B upoređivati sa svim karakterima iz intervala

$[i, i + length(A) - length(B) + 1]$, stringa A, odnosno da ćemo na ukupnu razliku dodati broj karaktera iz grupe $(A[i], A[i+1], A[i+2], \ldots, A[i + lenA - lenB + 1])$ koji su različiti od $B[i]$, za svako $i \in [1, length(B)]$.

Postavlja se pitanje kojim slovima treba zameniti nepoznate karaktere u stringu B? Iz prethodno navedene činjenice vidi se da je na mestu i optimalno slovo baš ono koje se pojavljuje najviše puta u stringu A u intervalu $[i,\ i + length(A) - length(B) + 1]$, jer će tako biti najmanje drugih karaktera koji su različiti od njega. Ukoliko ima više mogućih slova, uzećemo ono koje prvo dolazi po abecednom redu kako bi na kraju dobili leksikografski najmanji string.

Naivno traženje slova koje se najviše puta pojavljuje u svakom intervalu posebno, dovodi do složenosti $O(length(B) \cdot (length(A) - length(B)))$, što nije dovoljno za maksimalan broj bodova, pa je zato potrebno primetiti još jednu stvar.

Označimo sa D broj $length(A) - length(B) + 1$. Sada je potrebno pronaći koje slovo se najviše puta pojavljuje u intervalima $[1, D+1]$, $[2, D+2]$, $[3, D+3]$, itd. u stringu A. Pošto su svi intervali jednake dužine D, i uzastopni su, to nam omogućuje da lako update-ujemo vrednosti koliko se koje slovo puta pojavljuje, bez potrebe da uvek računamo sve ispočetka. Naime, ako smo pronašli za svako slovo s koliko se puta ono javlja u intervalu $[i, D+i]$, i obeležili to sa $broj\_pojavljivanja[s]$, onda kada prelazimo na interval $[i+1,\ D+i+1]$ dovoljno je oduzeti jedno pojavljivanje slova $A[i]$ (odnosno umanjiti $broj\_pojavljivanja[A[i]]$ za 1), a povećati pojavljivanje slova $A[D+i+1]$.

Posle ovih zaključaka, implementacija rešenja u složenosti $O(length(A))$ ne bi trebalo da bude previše teška, a ovde ćemo prikazati pseudo kod opisanog rešenja:

```
================================================================================
   Ulaz: Stringovi A i B
   Izlaz: Minimalna ukupna razlika i odgovarajuci string B

   01 broj_pojavljivanja[s] = 0, za svako s ∈ ['a','z']
   02 D = length(A) - length(B) + 1;
   03 for i = 1 to length(A) do
   04     if (i > D) then
   05             broj_pojavljivanja[ A[i-D] ]--;
   06     endif
   07     broj_pojavljivanja[ A[i] ]++;
   08     if (i >= D) then
   09           if (B[i] == '?') then
   10                 s = najveci_element(broj_pojavljivanja);
   11                 B[i] = s;
   12           endif
   13           uk_razlika += D - broj_pojavljivanja[ B[i] ];
   14     endif
   15 endfor
   16
   17 print B, uk_razlika
================================================================================
```

Napomena. Ovde je bitno (kao i u svakom drugom zadatku) voditi računa o graničnim slučajevima i maksimalnim vrednostima koje mogu imati pojedine promenljive. Tačnije, najveća razlika stringova se može dobiti kada je npr. string $A = "aaa \dots aaa"$ (1.000.000 slova $'a'$), a string $B = "bbb \dots bbb"$ (500.000 slova $'b'$), i ona iznosi $25 \cdot 10^{10}$, što prelazi granicu longint-a u Pascalu ili int-a u C/C++, pa je potrebno za promenljivu `uk_razlika` koristiti 64-bitni tip podataka, tj. int64 u Pascalu i long long u C/C++. Nažalost, iako je nepažnja oko biranja tipa podataka više puta pomenuta kao standardna greška, i u ovim kvalifikacijama smo naišli na veliki broj takmičara koji su je ponovili i tako izgubili dragocen broj poena.

Problem 5. Plusevi i Minusevi

Data je šahovska dimenzije 8 x 8. U svakom polju table nalazi se ili znak '+' (plus) ili znak '−' (minus). U jednom potezu je dozvoljeno izabrati proizvoljnu vrstu ili kolonu i promeniti sve znake u izabranoj vrsti/koloni u suprotne (plus prelazi u minus a minus prelazi u plus). Koliko najviše pluseva može sadržati naša tabla posle tačno K poteza?

Ulaz. Prvih 8 redova standardnog ulaza sadrže po 8 znakova ('+' ili '−') - i-ti red predstavlja opis i-tog reda tablice. Deveti red sadrži prirodan broj K ($1 \leq K \leq 10^9$) - potreban broj poteza.

Izlaz. U prvi i jedini red standardnog izlaza ispisati koliko najviše znakova '+' može sadržati tabla posle **tačno** K poteza .

Primer

Ulaz	Izlaz
+ + + + + + − + − + − − − − − + + + + + + + + + + + + + + + − + + + + + + + + + − + + + + + − + + + + + + + + + − + + − − + + + 2	53

Objašnjenje primera. Jedno od rešenja je da u prvom potezu izaberemo 2. vrstu a u drugom potezu izaberemo 7. kolonu.

Napomena. U 30% test primera je $K \leq 5$.

Rešenje i analiza:

Male dimenzije šahovske table sugerišu da problem Plusevi i Minusevi pripada klasi problema koji se rešavaju metodom isprobavanja svih mogućnosti, tj. *backtracking*-om. Na koliko načina uopšte možemo izvršiti K poteza? Bez dublje analize broj poteza možemo ograničiti sa 16^K – zaista, u svakom potezu biramo neku od 8 vrsta i 8 kolona (ukupno 16 mogućnosti). Za male vrednosti broja K to izgleda prilično uradivo (zapravo, *backtrack* sa tom složenošću donosi 30 poena) ali K može biti i do 10^9. Ipak, ispostavlja se da većina tih poteza dovode do iste konfiguracije table (ovo je i bilo za očekivati jer je ukupan broj različitih tabli "samo" 2^{64}).

Za $1 \leq i \leq 8$, označimo sa $row[i]$ - broj promena znakova u i-toj vrsti (tj. koliko smo puta "izabrali" i-tu vrstu). Analogno, za $1 \leq j \leq 8$ označimo sa $col[j]$ - broj promena znakova u j-toj koloni. Po uslovu zadatka mora važiti *glavna jednačina:*

$$\sum_{i=1}^{8} row[i] + \sum_{j=1}^{8} col[j] = K.$$

Označimo sa $a[i][j]$ – znak koji se nalazio na početku u preseku i-te vrste i j-te kolone naše table. Kako se promenio taj znak posle K poteza? On se promenio tačno $row[i] + col[j]$ puta. Prema tome, ukoliko je broj $row[i] + col[j]$ paran, znak $a[i][j]$ je ostao isti a inače se promenio u suprotni. Zaključujemo da krajnji

izgled znaka **zavisi isključivo od parnosti brojeva** $row[i]$ i $col[j]$ dok **ne zavisi od redosleda kojim smo menjali znakove po vrstama/kolonama** (ovo je standardna i poznata osobina operacije koja samo "obrće" stanja). Kako je i, j proizvoljno, ovo važi za celu tablu, tj. izgled table posle K poteza zavisi isključivo od parnosti elementa nizova row i col. Svaki element niza row može biti paran ili neparan (2^8 mogućnosti) i svaki element niza col može biti paran ili neparan (2^8 mogućnosti) pa posle K poteza možemo dobiti najviše $2^8 \cdot 2^8 = 2^{16}$ različitih tabli. Ali nisu sve kombinacije par-nepar validne, moramo uzeti u obzir i glavnu jednačinu.

Neka je, za svako $1 \leq i \leq 8$, $row[i] = 2 \cdot row'[i] + r[i]$ i $col[i] = 2 \cdot col'[i] + c[i]$, gde su $r[i], col[i] \in \{0, 1\}$. Prema prethodnoj diskusiji, krajnji izgled table zavisi samo od brojeva $r[i]$ i $c[i]$. Ukoliko ovo zamenimo u glavnu jednačinu, dobijamo

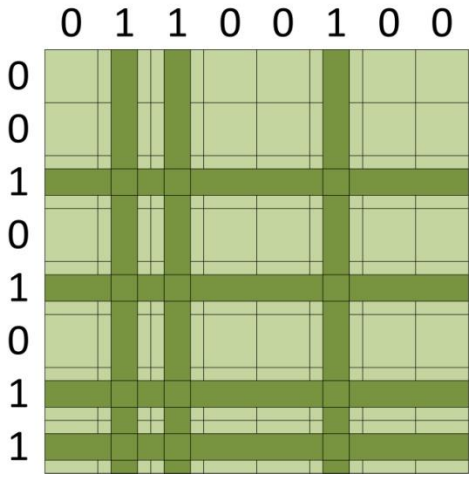
$$2\left(\sum_{i=1}^{8} row'[i] + \sum_{j=1}^{8} col'[j]\right) + \sum_{i=1}^{8} r[i] + \sum_{j=1}^{8} c[j] = K.$$

Kako su row', col' nizovi nenegativnih celih brojeva, iz prethodne jednačine zaključujemo da binarni nizovi r i c moraju zadovoljavati sledeća dva uslova:

- $\sum_{i=1}^{8} r[i] + \sum_{j=1}^{8} c[j] \equiv_2 K$ i
- $\sum_{i=1}^{8} r[i] + \sum_{j=1}^{8} c[j] \leq K$.

Obratno, ako su prethodna dva uslova zadovoljena, uvek možemo izabrati nizove row' i col' tako da glavna jednačina važi. Prema tome, sve moguće table možemo dobiti promenom svake vrste/kolone najviše jednom pri čemu ako i-tu vrstu menjamo $r[i]$ puta a j-tu kolonu $c[j]$ puta, binarni nizovi r i c moraju zadovoljavati prethodna 2 uslova.

Sada je algoritam prilično jasan: za sve moguće kombinacije binarnih nizova r i c, ukoliko zadovoljavaju gornja 2 uslova, izračunaćemo broj pluseva koji se dobijaju odgovarajućim promenama i uzećemo najveću od svih tako dobijenih vrednosti. Najlakše je nizove r i c kodirati po jednim brojem iz segmenta $[0, 2^8 - 1]$ a kasnije ih rekonstruisati iz binarnog zapisa – tehnika poznata kao korišćenje bitmaski (vidi Sliku 1).



Slika 1. Primer kodiranja nizova r i c: Situaciju kada vršimo promene u trećoj, petoj, sedmoj i osmoj vrsti i promene u drugoj, trećoj i šestoj koloni možemo predstaviti uredjenim parom brojeva $(R, C) = (43{,}100)$ jer je $43 = (00101011)_2$ i $100 = (01100100)_2$ pa nizove r i c jednostavno pročitamo iz binarnih zapisa brojeva R i C.

Pseudo kod opisanog rešenja:

```
================================================================================
   Ulaz: matrica a[i][j] i broj K
   Izlaz: najveći mogući broj pluseva posle K poteza

   01 max = 0;
   02 for R = 0 to 2^8 − 1 do
   03     for C = 0 to 2^8 − 1 do
   04           r[] = binarni zapis broja R;
   05           c[] = binarni zapis broja C;
   06           S = ∑r[i] + ∑c[i];
   07           if (S≤ K and (S-K) mod 2 = 0) then
   08                 plusevi = brojPlusevaPosleDatihPromena;
   09                 if plusevi > max then max = plusevi;
   10           endif
   11     endfor
   12 endfor
   13 return max;

================================================================================
```

Iskomentarišimo delove koda: u liniji 07 vršimo proveru pomenuta dva uslova. Linija 08 računa broj pluseva posle promena vrsta/kolona određenih nizovima r i c - složenost ove linije je $O(n^2)$ gde je n veličina table jer tu jednostavno prolazimo kroz celu tablu. Na polju $[i][j]$ se trenutno nalazi plus akko važi

$$(a[i][j] =' +' \textbf{ and } (r[i] + col[j])\ mod\ 2 = 0) \textbf{ or } (a[i][j] =' -' \textbf{ and } (r[i] + col[j])\ mod\ 2 \neq 0).$$

Ukupna složenost algoritma je $O(2^{2n}n^2)$ što za $n = 8$ osvaja maksimalan broj poena.

Zbog prirode problema i malih ograničenja, moguće je bilo razviti i drugačija rešenja koja su se zasnivala na raznim *greedy* algoritmima i sličnim heuristikama. Iako takva rešenja nisu tačna, solidan deo njih je uzeo solidan broj poena.

Pomenimo da postoji i algoritam bolje složenosti. Pretpostavimo da smo fiksirali niz c, tj. da smo odlučili u kojim kolonama ćemo menjati znakove a u kojim ne. Ukoliko te promene izvršimo odmah, ostaje nam da izvršimo izbor vrsta u kojima ćemo vršiti promene. U prethodnom algoritmu smo to radili tako što smo probali svih 2^8 mogućnosti. Međutim, ovde radi i grabljivi (*greedy*) metod: posmatrajmo vrstu i. Ukoliko je u njoj broj pluseva veći ili jednak od broja minusa, sigurno se ne isplati da vršimo promenu u ovoj vrsti (više ne vršimo promene po kolonama pa bi promena u i-toj vrsti dovela do smanjenja broja pluseva). U suprotnom, uvek se ispalti da izvršimo promenu u toj vrsti. Međutim, stalno treba voditi računa o pomenuta 2 uslova za nizove r i c - u opštem slučaju broj vrsti koje možemo da okrenemo će nam biti ograničen i u tom slučaju je najbolje okrenuti one sa najviše minusa. Složenost algoritma je $O(2^n n^2)$. Detalje implementacije i analizu slučajeva ostavljamo zainteresovanom čitaocu.