Rješenja Codeforces Problema

Dokumentacija

2024-12-21

Sadržaj

1 Uvod

Ovaj dokument sadrži detaljna rješenja i objašnjenja za 10 odabranih problema sa Codeforces platforme. Svaki problem je pažljivo analiziran i implementiran u C++ programskom jeziku. Za svako rješenje predstavljamo:

- Detaljnu analizu problema i pristup rješenju
- Matematičku pozadinu i dokaze gdje je to potrebno
- Optimizovanu implementaciju sa jasnim objašnjenjima
- Analizu vremenske i prostorne složenosti

Svako rješenje je testirano na više test primjera kako bi se osigurala korektnost implementacije.

2 Problem 1: Cipher (Codeforces 156 C - 2000 hard)

Link to problem

2.1 Postavka Problema

Sherlock Holmes je pronašao tajnu prepisku između dva VIP-a i odlučio je pročitati. Međutim, prepiska je šifrirana. Detektiv je pokušao dešifrirati prepisku, ali nije uspio razumjeti ništa.

Na kraju, nakon razmišljanja, smislio je nešto. Recimo da postoji riječ s, koja se sastoji od $|s|$ malih latiničnih slova. Tada za jednu operaciju možete odabrati određenu poziciju p ($1 \leq p < |s|$) i izvršiti jednu od sljedećih radnji:

- zamijeniti slovo s_p s onim koje ga abecedno slijedi i zamijeniti slovo s_{p+1} s onim koje ga abecedno prethodi;
- ili zamijeniti slovo s_p s onim koje ga abecedno prethodi i zamijeniti slovo s_{p+1} s onim koje ga abecedno slijedi.

Napominjemo da slovo "z" nema definirano sljedeće slovo, a slovo "a" nema definirano prethodno slovo. Zato odgovarajuće promjene nisu prihvatljive. Ako operacija zahtijeva izvođenje barem jedne neprihvatljive promjene, tada se takva operacija ne može izvesti.

Dve riječi se podudaraju u značenju ako se jedna od njih može transformirati u drugu kao rezultat nula ili više operacija.

Sherlock Holmes treba brzo naučiti odrediti sljedeće za svaku riječ: koliko riječi može postojati koje se podudaraju s njom u značenju, ali se razlikuju od nje u barem jednom znaku? Izračunajte ovaj broj za njega modulo 1000000007 ($10^9 + 7$).

2.1.1 Ulaz

- Prvi red sadrži jedan cijeli broj t ($1 \leq t \leq 10^4$) — broj testova.
- Sljedećih t redova sadrži riječi, po jednu u svakom redu. Svaka riječ se sastoji od malih latiničnih slova i ima dužinu od 1 do 100, uključivo. Dužine riječi mogu se razlikovati.

2.1.2 Izlaz

- Za svaku riječ ispišite broj različitih drugih riječi koje se podudaraju s njom u značenju — ne iz riječi navedenih u ulaznim podacima, već iz svih mogućih riječi. Kako traženi broj može biti vrlo velik, ispišite njegovu vrijednost modulo 1000000007 ($10^9 + 7$).

2.2 Pristup Rješenju

Problem se rješava koristeći dinamičko programiranje. Ključna opservacija je da možemo pratiti sumu pozicija slova u riječi i koristiti DP za izračunavanje broja mogućih transformacija.

2.3 Dinamičko Programiranje

Definišemo stanje dp[i][s] gdje je:

- i - dužina riječi
- s - suma pozicija slova u riječi

2.4 Tranzicije Stanja

Za svaku poziciju i i trenutnu sumu s, imamo sljedeće mogućnosti:

1. Dodati slovo: dp[i+1][s + k] += dp[i][s] za svako slovo k

2.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(n \cdot 25n)$
  - Za svaku dužinu riječi (n)
  - Za svaku moguću sumu ($25n$)
- Prostorna Kompleksnost: $O(n \cdot 25n)$
  - DP tabela: $O(n \cdot 25n)$

2.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Korištenje modularne aritmetike za velike brojeve
- Efikasno računanje suma pozicija slova
- Pravilno rukovanje ulaznim i izlaznim podacima

3 Problem 2: Clear The String (Codeforces 1132F - 2000 hard)

Link to problem

3.1 Postavka Problema

Dat je string s koji se sastoji od malih slova engleske abecede. U jednoj operaciji možemo izbrisati neki kontinualni podstring stringa s ako su sva slova u tom podstringu jednaka. Na primjer, nakon brisanja podstringa bbbb iz stringa abbbaccda dobijamo string aaccda.

Zadatak je izračunati minimalan broj operacija potrebnih da se izbriše cijeli string s.

3.1.1 Ulaz

- Prvi red sadrži jedan cijeli broj n ($1 \leq n \leq 500$) - dužinu stringa s
- Drugi red sadrži string s ($|s| = n$) koji se sastoji od malih slova engleske abecede

3.1.2 Izlaz

- Ispisati jedan cijeli broj - minimalan broj operacija potrebnih da se izbriše string s

3.2 Pristup Rješenju

Problem rješavamo koristeći dinamičko programiranje. Ključni uvid je da za svaki podstring možemo:

- Izbrisati prvo slovo zasebno
- Izbrisati prvo slovo zajedno sa nekim drugim istim slovom koje se nalazi kasnije u stringu

3.3 Dinamičko Programiranje

Definišemo stanje dp[l][r] kao minimalan broj operacija potrebnih da se izbriše podstring od pozicije l do pozicije r.

Imamo sljedeće slučajeve:

- Ako je l > r: dp[l][r] = 0 (prazan string)
- Ako je l = r: dp[l][r] = 1 (jedno slovo)
- Inače, dp[l][r] je minimum od:
  - 1 + dp[l+1][r] (brisanje prvog slova zasebno)
  - dp[l+1][i-1] + dp[i][r] za svako i gdje je s[l] = s[i] (brisanje prvog slova sa nekim kasnijim istim slovom)

3.4 Tranzicije Stanja

Za svaki podstring [l,r]:

1. Prvo postavimo dp[l][r] = 1 + dp[l+1][r] (slučaj zasebnog brisanja)
2. Zatim za svako i od l+1 do r:
   - Ako je s[l] = s[i], pokušamo spojiti slovo na poziciji l sa slovom na poziciji i
   - U tom slučaju moramo riješiti dva podproblema:
     - dp[l+1][i-1]: podstring između uparenih slova
     - dp[i][r]: podstring od drugog uparenog slova do kraja

3.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(n^3)$
  - Imamo $O(n^2)$ stanja (za svaki par l,r)
  - Za svako stanje prolazimo kroz $O(n)$ mogućih pozicija za uparivanje
- Prostorna Kompleksnost: $O(n^2)$ za DP tabelu

3.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Inicijalizacija DP tabele sa -1 za označavanje neposjećenih stanja
- Pravilno rukovanje baznim slučajevima (prazan string i pojedinačna slova)
- Efikasno računanje minimalnog broja operacija koristeći bottom-up pristup

4 Problem 3: Red-Green Towers (Codeforces 478D - 2000 hard)

Link to problem

4.1 Postavka Problema

4.1.1 Opis

Dat je broj r crvenih i g zelenih kocki. Potrebno je izgraditi toranj koji zadovoljava sljedeća pravila:

- Toranj se sastoji od nekoliko nivoa
- Ako toranj ima h nivoa, prvi nivo treba imati h kocki, drugi nivo $h-1$ kocki, i tako dalje do vrha gdje zadnji nivo ima 1 kocku
- Svaki nivo mora biti izgrađen od kocki iste boje (ili sve crvene ili sve zelene)

4.1.2 Ulaz

- Prvi i jedini red sadrži dva cijela broja r i g ($0 \leq r, g \leq 2 \cdot 10^5$, $r+g \geq 1$)

4.1.3 Izlaz

- Ispisati jedan broj - broj različitih načina na koje se može izgraditi toranj maksimalne moguće visine, po modulu $10^9 + 7$

4.2 Analiza Rješenja

4.2.1 Ključna Opažanja

- Za toranj visine h potrebno je tačno $\frac{h(h+1)}{2}$ kocki
- Maksimalna visina tornja je najveće h za koje vrijedi $\frac{h(h+1)}{2} \leq r + g$
- Dva tornja su različita ako postoji barem jedan nivo koji je u jednom tornju crven, a u drugom zelen

4.2.2 Pristup Rješenju

Problem rješavamo u dva koraka:

1. Prvo određujemo maksimalnu moguću visinu tornja h
2. Zatim koristimo dinamičko programiranje za računanje broja različitih načina izgradnje tornja te visine

4.3 Implementacija

4.3.1 Dinamičko Programiranje

Definišemo stanje dp[t][r] kao broj načina da se izgradi prvih t nivoa tornja koristeći tačno r crvenih kocki.

Tranzicije su:

- Dodavanje crvenog nivoa: dp[t+1][r + (t+1)] += dp[t][r]
- Dodavanje zelenog nivoa: dp[t+1][r] += dp[t][r]

4.3.2 Optimizacije

1. **Memorijska Optimizacija**: Čuvamo samo dva reda DP tabele
2. **Preskakanje Nula**: Preskačemo računanje tranzicija iz stanja gdje je dp[t][r] = 0
3. **Modularna Aritmetika**: Sve operacije se izvode po modulu $10^9 + 7$

4.4 Analiza Složenosti

- **Vremenska složenost**: $O(h \cdot r)$
  - Za svaki nivo t od 0 do h-1
  - Za svaki broj iskorištenih crvenih kocki r
- **Prostorna složenost**: $O(r)$
  - Samo dva reda DP tabele
  - Svaki red ima r+1 elemenata

5 Problem 4: Lucky Common Subsequence (Codeforces 346B - 2000 hard)

Link to problem

5.1 Postavka Problema

Data su dva stringa s_1 i s_2 i treći string koji se zove virus. Zadatak je pronaći najdužu zajedničku podsekvenciju stringova s_1 i s_2 koja ne sadrži virus kao podstring.

Podsekvencija stringa je sekvenca koja se može dobiti iz originalnog stringa brisanjem nekih elemenata bez mijenjanja redoslijeda preostalih elemenata. Podstring je kontinuirana podsekvencija stringa.

5.1.1 Ulaz

- Prvi red sadrži string s_1 ($1 \leq |s_1| \leq 100$)
- Drugi red sadrži string s_2 ($1 \leq |s_2| \leq 100$)
- Treći red sadrži string virus ($1 \leq |virus| \leq 100$)
- Svi stringovi se sastoje samo od velikih slova engleske abecede

5.1.2 Izlaz

- Ispisati najdužu zajedničku podsekvenciju stringova s_1 i s_2 koja ne sadrži virus kao podstring
- Ako takva podsekvencija ne postoji, ispisati 0

5.2 Pristup Rješenju

Problem kombinuje dva klasična problema dinamičkog programiranja:

- Najduža zajednička podsekvencija (LCS)
- Izbjegavanje podstringa (virus)

Ključni uvid je da moramo pratiti ne samo pozicije u s_1 i s_2, već i koliko smo virusa "izgradili" do sada.

5.3 Dinamičko Programiranje

Koristimo 3-dimenzionalni pristup dinamičkog programiranja:

$$dp[i][j][k]$$

Gdje je:

- i: Pozicija u prvom stringu s_1
- j: Pozicija u drugom stringu s_2
- k: Dužina prefiksa virusa koji se podudara sa sufiksom trenutne podsekvencije

5.4 Tranzicije Stanja

Za svaku poziciju (i, j, k), imamo nekoliko mogućnosti:

1. Ako se karakteri $s_1[i]$ i $s_2[j]$ podudaraju:
    - Ako se karakter podudara sa sljedećim karakterom virusa, povećavamo k
    - Inače, tražimo najduži prefiks virusa koji se podudara sa sufiksom + novi karakter
2. Preskočiti karakter iz s_1 $(i + 1)$

3. Preskočiti karakter iz s_2 $(j + 1)$

4. Preskočiti karaktere iz oba stringa $(i + 1, j + 1)$

5.5 Rekonstrukcija Rješenja

Za rekonstrukciju rješenja koristimo dodatnu strukturu *previous_state* koja pamti prethodno stanje za svaku poziciju u DP tabeli. Kada pronađemo optimalno rješenje, pratimo put unazad da rekonstruišemo podsekvenciju.

5.6 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(|s_1| \cdot |s_2| \cdot |virus|)$
- Prostorna Kompleksnost: $O(|s_1| \cdot |s_2| \cdot |virus|)$

5.7 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Korištenje memset za brzu inicijalizaciju DP tabele
- Efikasno poređenje podstringova za virus pattern matching
- Rano odbacivanje nemogućih stanja
- Optimizovano praćenje prethodnih stanja za rekonstrukciju rješenja

6 Problem 5: Cards and Joy (Codeforces 999F - 2000 hard)

Link to problem

6.1 Postavka Problema

Dato je n igrača koji sjede za stolom. Svaki igrač ima svoj omiljeni broj. Omiljeni broj j-tog igrača je f_j.

Na stolu se nalazi $k \cdot n$ karata. Svaka karta sadrži jedan cijeli broj: i-ta karta sadrži broj c_i. Također, data je sekvenca $h_1, h_2, \ldots, h_k$ čije će značenje biti objašnjeno u nastavku.

Igrači moraju raspodijeliti sve karte tako da svaki od njih dobije tačno k karata. Nakon što su sve karte raspodijeljene, svaki igrač prebroji koliko

karata ima sa svojim omiljenim brojem. Nivo sreće igrača je h_t ako igrač ima t karata sa svojim omiljenim brojem. Ako igrač nema nijednu kartu sa svojim omiljenim brojem (tj. $t = 0$), njegov nivo sreće je 0.

6.1.1 Ulaz

- Prvi red sadrži dva cijela broja n i k ($1 \leq n \leq 500, 1 \leq k \leq 10$)
- Drugi red sadrži $k \cdot n$ cijelih brojeva $c_1, c_2, \ldots, c_{k \cdot n}$ ($1 \leq c_i \leq 10^5$)
- Treći red sadrži n cijelih brojeva $f_1, f_2, \ldots, f_n$ ($1 \leq f_j \leq 10^5$)
- Četvrti red sadrži k cijelih brojeva $h_1, h_2, \ldots, h_k$ ($1 \leq h_t \leq 10^5$)

6.1.2 Izlaz

- Ispisati jedan cijeli broj – maksimalan mogući ukupni nivo sreće igrača nakon raspodjele karata

6.2 Pristup Rješenju

Problem se može riješiti korištenjem dinamičkog programiranja. Ključni uvid je da možemo nezavisno rješavati za svaki omiljeni broj koji se pojavljuje među igračima.

6.3 Dinamičko Programiranje

Za svaki jedinstveni omiljeni broj x, rješavamo podproblem:

- Imamo p igrača koji žele broj x
- Imamo c karata sa brojem x
- Svaki igrač mora dobiti između 0 i k karata

Definiramo stanje dinamičkog programiranja:

$$dp[i][j]$$

Gdje je:

- i: Broj igrača kojima smo dodijelili karte (0 do p)
- j: Broj karata koje smo iskoristili (0 do $\min(c, p \cdot k)$)

6.4 Tranzicije Stanja

Za svako stanje (i, j), razmatramo sve moguće načine da dodijelimo karte sljedećem igraču:

$$dp[i+1][j+t] = \max(dp[i+1][j+t], dp[i][j] + h_t)$$

Gdje je:

- t: Broj karata koje dajemo sljedećem igraču (0 do $\min(k, c-j)$)
- h_t: Nivo sreće za dobijanje t karata

6.5 Rekonstrukcija Rješenja

Za svaki omiljeni broj x:

1. Izračunamo optimalno rješenje za igrače koji žele taj broj
2. Dodamo maksimalnu moguću sreću na ukupni rezultat

6.6 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(C \cdot N \cdot K \cdot N)$, gdje je C broj različitih omiljenih brojeva
- Prostorna Kompleksnost: $O(N \cdot NK)$

6.7 Detalji Implementacije

Implementacija koristi nekoliko optimizacija:

- Brojanje karata i igrača umjesto praćenja pojedinačnih dodjela
- Ograničavanje broja dostupnih karata na $\min(card\_count[x], players \cdot k)$
- Korištenje lokalnog DP-a za svaki omiljeni broj
- Rano odbacivanje nemogućih stanja

7 Problem 6: Supermarket Shopping (Codeforces 815C - 2400 hard)

Link to problem

7.1 Postavka Problema

Karen je studentica koja želi kupiti namirnice u supermarketu sa ograničenim budžetom. Supermarket nudi n proizvoda i posebne kupone za svaki proizvod. Problem je optimizirati kupovinu kako bi se kupio maksimalan broj proizvoda uz data ograničenja.

7.1.1 Ulaz

- Prvi red sadrži dva cijela broja n i b ($1 \leq n \leq 5000$, $1 \leq b \leq 10^9$)
  - n - broj proizvoda u supermarketu
  - b - Karenin budžet
- Sljedećih n redova opisuju proizvode. Za svaki proizvod i:
  - Dva cijela broja c_i i d_i ($1 \leq d_i < c_i \leq 10^9$)
    - c_i - cijena proizvoda
    - d_i - iznos popusta sa kuponom
  - Za $i \geq 2$, dodatni broj x_i ($1 \leq x_i < i$) - indeks kupona koji mora biti iskorišten prije kupona i

7.1.2 Izlaz

- Jedan cijeli broj - maksimalan broj proizvoda koji Karen može kupiti sa budžetom b

7.2 Pristup Rješenju

Problem se može riješiti korištenjem dinamičkog programiranja i DFS pristupa. Ključni uvidi su:

- Za svaki proizvod imamo dva stanja: kupovina sa kuponom i bez kupona
- Kuponi formiraju usmjereni aciklički graf (DAG) zavisnosti
- Za korištenje kupona i, moramo koristiti kupon x_i

7.3 Dinamičko Programiranje

Koristimo tri stanja za svaki čvor u grafu:

$$dp[state][node]$$

Gdje je:

- *state* može biti:
  - 0: bez korištenja kupona za trenutni čvor
  - 1: sa korištenjem kupona za trenutni čvor
  - 2: najbolji rezultat kombinujući oba stanja
- *node*: trenutni proizvod koji razmatramo

7.4 DFS i Spajanje Rezultata

Za svaki čvor u grafu:

1. Inicijaliziramo bazne slučajeve:
   - Bez kupona: $\{0, cijena\}$
   - Sa kuponom: $\{INF, cijena - popust\}$
2. Rekurzivno obrađujemo sve zavisne čvorove
3. Spajamo rezultate koristeći funkciju merge:
   - Za stanje bez kupona: merge(dp[0][node], dp[0][child])
   - Za stanje sa kuponom: merge(dp[1][node], dp[2][child])

7.5 Analiza Složenosti

- **Vremenska složenost**: $O(n \cdot m)$
  - n - broj proizvoda
  - m - maksimalan broj mogućih kombinacija proizvoda
- **Prostorna složenost**: $O(n)$
  - Prostor za graf zavisnosti
  - DP tabela za svaki čvor

7.6 Detalji Implementacije

- Efikasno spajanje rezultata koristeći min operaciju
- Pamćenje samo neophodnih međurezultata
- Korištenje konstante `0x3f3f3f3f` kao beskonačnosti umjesto `INT_MAX`
- Rano odbacivanje nemogućih stanja

8 Problem 7: Divide by Three (Codeforces 792C - 2000 hard)

Link to problem

8.1 Postavka Problema

Na tabli je napisan pozitivan cijeli broj n. Potrebno je transformisati taj broj u lijep broj brisanjem nekih cifara, pri čemu želimo obrisati što je moguće manje cifara.

Broj se smatra lijepim ako se sastoji od najmanje jedne cifre, nema vodećih nula i djeljiv je sa 3. Na primjer, 0, 99, 10110 su lijepi brojevi, dok 00, 03, 122 nisu.

Potrebno je napisati program koji će za dati broj n pronaći lijep broj takav da se n može transformisati u taj broj brisanjem najmanjeg mogućeg broja cifara. Možete obrisati proizvoljni skup cifara, ne moraju biti uzastopne u broju n.

Ako nije moguće dobiti lijep broj, ispisati -1. Ako postoji više rješenja, ispisati bilo koje od njih.

8.1.1 Ulaz

- Prvi red sadrži pozitivan cijeli broj n bez vodećih nula ($1 \leq n < 10^{100000}$)

8.1.2 Izlaz

- Ispisati jedan broj – bilo koji lijep broj dobijen brisanjem najmanjeg mogućeg broja cifara
- Ako ne postoji rješenje, ispisati -1

8.2 Pristup Rješenju

Problem se rješava koristeći dinamičko programiranje. Ključna opservacija je da možemo pratiti ostatak pri dijeljenju sa 3 za svaki prefiks i dužinu formiranog broja. Također, moramo voditi računa o vodećim nulama i posebnim slučajevima.

8.3 Dinamičko Programiranje

Definišemo stanje dp[i][r] kao maksimalnu dužinu lijepog broja koji se može formirati koristeći prvih i cifara sa ostatkom r pri dijeljenju sa 3. Dodatno, koristimo niz choice[i][r] za rekonstrukciju rješenja.

8.4 Tranzicije Stanja

Za svaku poziciju i i trenutnu cifru d, imamo sljedeće mogućnosti:

1. Ako d nije 0, možemo početi novi broj: dp[i+1][d mod 3] = 1
2. Za svaki postojeći ostatak r:
   - Dodaj cifru: dp[i+1][(r $\cdot$ 10 + d) mod 3] = dp[i][r] + 1
   - Ne dodaj cifru: dp[i+1][r] = dp[i][r]

8.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: O(n), gdje je n dužina ulaznog stringa
  - Za svaku poziciju razmatramo 3 moguća ostatka
  - Svaka operacija je O(1)
- Prostorna Kompleksnost: O(n)
  - dp tabela: O(n)
  - choice niz: O(n)

8.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Korištenje pomoćne funkcije updateMax za efikasno ažuriranje dp stanja

- Pažljivo rukovanje vodećim nulama u izlazu
- Pravilno računanje ostataka tokom rekonstrukcije
- Optimizovano praćenje prethodnih stanja za rekonstrukciju rješenja

9 Problem 8: Bear and Company (Codeforces 790 C- 2500 hard)

Link to problem

9.1 Postavka Problema

Bear Limak priprema zadatke za programersko takmičenje. Naravno, bilo bi neprofesionalno spomenuti ime sponzora u tekstu zadatka. Limak to shvata ozbiljno i želi promijeniti neke riječi. Da bi tekst i dalje bio čitljiv, pokušat će modificirati svaku riječ što je manje moguće.

Limak ima string s koji se sastoji od velikih slova engleske abecede. U jednom potezu može zamijeniti mjesta dvama susjednim slovima u stringu. Na primjer, može transformirati string "ABBC" u "BABC" ili "ABCB" u jednom potezu.

Limak želi dobiti string bez podstringa "VK" (tj. ne smije biti slova 'V' neposredno praćenog slovom 'K'). Može se lako dokazati da je to moguće za bilo koji početni string s.

9.1.1 Ulaz

- Prvi red sadrži jedan cijeli broj n ($1 \leq n \leq 75$) – dužinu stringa
- Drugi red sadrži string s koji se sastoji od velikih slova engleske abecede, dužine n

9.1.2 Izlaz

- Ispisati jedan cijeli broj – minimalan broj poteza koje Limak mora napraviti da bi dobio string bez podstringa "VK"

9.2 Pristup Rješenju

Problem se rješava koristeći dinamičko programiranje. Ključna opservacija je da možemo pratiti pozicije slova 'V', 'K' i ostalih slova odvojeno, te računati minimalan broj poteza potrebnih za njihovo preuređivanje.

9.3 Dinamičko Programiranje

Definišemo stanje dp[i][j][k][l] gdje je:

- i - broj obrađenih 'V' slova
- j - broj obrađenih 'K' slova
- k - broj obrađenih ostalih slova (označenih sa `#`)
- l - flag koji označava da li je prethodno slovo bilo 'V' (1) ili nije (0)

9.4 Tranzicije Stanja

Za svako stanje imamo tri mogućnosti:

1. Uzeti sljedeće 'V': dp[i+1][j][k][1]
2. Uzeti sljedeće 'K' (samo ako prethodno nije bilo 'V'): dp[i][j+1][k][0]
3. Uzeti sljedeće drugo slovo: dp[i][j][k+1][0]

Cijena svake tranzicije se računa kao broj slova koja moramo preskočiti da bismo došli do željene pozicije.

9.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(n^3)$
  - Za svaku kombinaciju i, j, k (maksimalno n)
  - Za svako stanje l (2 mogućnosti)
  - Računanje cijene je $O(1)$ zbog binary searcha
- Prostorna Kompleksnost: $O(n^3)$
  - DP tabela: $O(n^3)$
  - Pozicije slova: $O(n)$

9.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Pretprocesiranje stringa zamjenom svih slova osim 'V' i 'K' sa `#`
- Korištenje binary searcha za efikasno računanje cijene tranzicija
- Pamćenje pozicija slova u sortiranim nizovima
- Korištenje upmin funkcije za ažuriranje minimalnih vrijednosti

10 Problem 9: Antimatter (Codeforces 383D - 2300 hard)

Link to problem

10.1 Postavka Problema

Iahub je slučajno otkrio tajni laboratorij. U njemu je pronašao n uređaja poredanih u liniju, numerisanih od 1 do n slijeva nadesno. Svaki uređaj i ($1 \leq i \leq n$) može proizvesti ili a_i jedinica materije ili a_i jedinica antimaterije.

Iahub želi odabrati neki kontinuirani podniz uređaja u laboratoriji, odrediti način proizvodnje za svaki od njih (proizvodnja materije ili antimaterije) i na kraju napraviti fotografiju. Međutim, bit će uspješan samo ako su količine proizvedene materije i antimaterije u odabranom podnizu jednake (u suprotnom bi došlo do prekomjerne materije ili antimaterije na fotografiji).

10.1.1 Ulaz

- Prvi red sadrži jedan cijeli broj n ($1 \leq n \leq 1000$) – broj uređaja
- Drugi red sadrži n cijelih brojeva $a_1, a_2, ..., a_n$ ($1 \leq a_i \leq 1000$)
- Suma $a_1 + a_2 + ... + a_n$ će biti manja ili jednaka 10000

10.1.2 Izlaz

- Ispisati jedan cijeli broj – broj različitih načina na koje Iahub može napraviti fotografiju, po modulu $10^9 + 7$

10.2 Pristup Rješenju

Problem se rješava koristeći dinamičko programiranje. Ključna opservacija je da za svaki uređaj imamo dva izbora: proizvesti materiju ($+a_i$) ili antimateriju ($-a_i$). Cilj je pronaći sve kontinuirane podnizove gdje je suma odabranih vrijednosti jednaka nuli.

10.3 Dinamičko Programiranje

Definišemo stanje dp[i][s] gdje je:

- i - pozicija do koje smo obradili uređaje (0-bazirano)
- s - trenutna razlika između količine materije i antimaterije

Koristimo tehniku "rolling array" za optimizaciju memorije, čuvajući samo dvije vrste stanja:

- dp[current_row][s] - stanja za trenutnu poziciju
- dp[next_row][s] - stanja za sljedeću poziciju

10.4 Tranzicije Stanja

Za svaku poziciju i i trenutnu sumu s, imamo sljedeće mogućnosti:

1. Dodati materiju: dp[next_row][s + a[i]] += dp[current_row][s]
2. Dodati antimateriju: dp[next_row][s - a[i]] += dp[current_row][s]
3. Započeti novi podniz od trenutne pozicije: dp[next_row][0] += 1

10.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(n \cdot MAXS)$
  - Za svaku poziciju (n)
  - Za svaku moguću sumu ($2 \cdot MAXS$)
  - Gdje je MAXS = 10000
- Prostorna Kompleksnost: $O(MAXS)$
  - Koristimo samo dva reda DP tabele
  - Svaki red ima veličinu $O(MAXS)$

10.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Korištenje "rolling array" tehnike za optimizaciju memorije
- Efikasno rukovanje negativnim indeksima pomoću pomaka u sredinu niza
- Pravilno rukovanje modularnom aritmetikom za velike brojeve
- Optimizovano čišćenje niza za sljedeću iteraciju

11 Problem 10: Minesweeper 1D (Codeforces 404 D- 1900 hard)

Link to problem

11.1 Postavka Problema

Igra "Minesweeper 1D" se igra na liniji kvadrata, visine 1 kvadrat i širine n kvadrata. Neki od kvadrata sadrže mine. Ako kvadrat ne sadrži minu, onda sadrži broj od 0 do 2 – ukupan broj mina u susjednim kvadratima.

Na primjer, ispravno polje za igru izgleda ovako: 001*2***101*. Ćelije označene sa "*" sadrže mine. Primijetite da na ispravnom polju brojevi predstavljaju broj mina u susjednim ćelijama. Na primjer, polje 2* nije ispravno, jer ćelija sa vrijednošću 2 mora imati dvije susjedne ćelije sa minama.

Valera želi napraviti ispravno polje za igru "Minesweeper 1D". Već je nacrtao kvadratno polje širine n ćelija, postavio nekoliko mina na polje i napisao brojeve u neke ćelije. Sada se pita na koliko načina može popuniti preostale ćelije minama i brojevima tako da na kraju dobije ispravno polje.

11.1.1 Ulaz

- Prvi red sadrži niz karaktera bez razmaka $s_1 s_2 ... s_n$ $(1 \leq n \leq 10^6)$
- Niz sadrži samo karaktere "*", "?" i cifre "0", "1" ili "2"
- Ako je karakter s_i jednak "*", tada i-ta ćelija polja sadrži minu
- Ako je karakter s_i jednak "?", tada Valera još nije odlučio šta staviti u i-tu ćeliju
- Karakter s_i koji je jednak cifri predstavlja cifru napisanu u i-tom kvadratu

11.1.2 Izlaz

- Ispisati jedan cijeli broj – broj načina na koje Valera može popuniti prazne ćelije i dobiti ispravno polje
- Kako odgovor može biti velik, ispisati ga po modulu 1000000007 ($10^9 + 7$)

11.2 Pristup Rješenju

Problem se rješava koristeći dinamičko programiranje. Ključna opservacija je da za svaku ćeliju moramo pratiti broj mina u susjednim ćelijama kako bismo osigurali da brojevi u ćelijama budu ispravni.

11.3 Dinamičko Programiranje

Definišemo stanje dp[i][j][k] gdje je:

- i - pozicija do koje smo popunili polje
- j - stanje prethodne ćelije (0-2 za broj, 3 za minu)
- k - stanje trenutne ćelije (0-2 za broj, 3 za minu)

11.4 Tranzicije Stanja

Za svaku poziciju i stanje imamo sljedeće mogućnosti:

1. Ako je trenutna ćelija "?", možemo staviti:
    - Broj (0-2) ako je validan s obzirom na susjedne mine
    - Minu (3) ako je validno s obzirom na susjedne brojeve
2. Ako je trenutna ćelija već popunjena, provjeravamo samo validnost

11.5 Analiza Kompleksnosti

- Vremenska Kompleksnost: $O(n \cdot 4 \cdot 4 \cdot 4)$
    - Za svaku poziciju (n)
    - Za svako stanje prethodne ćelije (4)
    - Za svako stanje trenutne ćelije (4)
    - Za svaku moguću vrijednost nove ćelije (4)
- Prostorna Kompleksnost: $O(n \cdot 4 \cdot 4)$
    - DP tabela: $O(n \cdot 4 \cdot 4)$

11.6 Detalji Implementacije

Implementacija koristi nekoliko ključnih optimizacija:

- Korištenje vrijednosti 3 za predstavljanje mine radi lakše obrade
- Efikasno računanje broja susjednih mina
- Provjera validnosti stanja prije ažuriranja DP tabele
- Pravilno rukovanje modularnom aritmetikom za velike brojeve