Zadatke, test primjere i rješenja pripremili: Frane Kurtović i Ante Đerek. Primjeri implementiranih rješenja su dani u priloženim izvornim kodovima koji nužno ne odgovaraju u svim detaljima ovdje opisanim algoritmima.

## TEHLE

*Predložio: Ante Đerek*
*Potrebno znanje: naredba if, for petlja, stringovi*

Najjednostavnije je znakove učitati u string, te zatim obrađivati string znak po znak. Ako je znak znamenka onda odmah ispišemo taj znak, a ako nije onda je potrebno napisati 8 if naredbi kojima se provjerava kojoj znamenci odgovara taj znak. Još jednostavnije rješenje, primjerice u Pythonu je napraviti rječnik koji preslikava string znakova na jednoj tipki u broj koji je na toj tipki, npr. map = {'ABC': 2, 'DEF': 3, ...}. Za trenutno slovo je potrebno proći po elementima mape, te provjeriti nalazi li se znak unutar ključa trenutnog elementa i ispisati pripadajuću vrijednost ako se nalazi.

### Za one koji žele više

Za zadani telefonski broj i rječnik, pronaćite sve moguće načine da se neke znamenke zamjene slovima tako da se dobije riječ iz rječnika.

## MPEG

*Predložio: Ante Đerek*
*Potrebno znanje: for petlja, niz brojeva*

Jedan od pristupa je zasebno riješiti naredbe I, P i B. Izračunate vrijednosti niza zapisujemo u zajednički niz A. Prvo i najjednostavnije za izračunati su vrijednosti na mjestima naredbi I, samo zapišemo vrijednost iz naredbe u odgovarajuće mjesto u nizu. Nakon toga je tek moguće izračunati vrijednosti na ostalim pozicijama jer naredbe P i B uvijek koriste postojeće vrijednosti u nizu za izračun vrijednosti na svojoj poziciji.

Sve vrijednosti na mjestima naredbi P se sada mogu izračunati obilaskom niza s lijeva na desno te računajući vrijednost za trenutnu poziciju iz prethodne jer će prethodna pozicija već biti izračunata. Nakon toga se isti postupak koristi i za računanje vrijednosti na pozicijama naredbi B samo što je potrebno niz obilaziti s desna na lijevo (od kraja).

# TOČKE

*Predložio: Ante Đerek*
*Potrebno znanje: dvodimenzionalni niz, jednostavno dinamičko programiranje*

Definirajmo funkciju $L[r,s]$ kao najveću duljinu puta ako put počinje u točki $[r,s]$ i u trenutnom retku ide samo lijevo, a kad pređe u idući redak se smije kretati u svim smjerovima. Analogno tome definiramo funkciju $R[r,s]$ koja ide u desno u trenutnom retku. Neka matrica B označava boje u ulaznom polju.

Trivijalno je izračunati vrijednosti ovih funkcija u zadnjem retku. Iz tih vrijednosti računamo vrijednosti od predzadnjeg retka do prvog retka prema sljedećoj formuli.

$$L(r,s) = max\{$$
$$1,$$
$$L(r,s-1) + 1, (ako\ je\ B[r][s] = B[r][s-1])$$
$$L(r+1,s) + 1, (ako\ je\ B[r][s] = B[r+1][s])$$
$$R(r+1,s) + 1, (ako\ je\ B[r][s] = B[r+1][s])$$
$$\}$$

Funkcija $R$ se računa na sličan način.

Rješenje je najveća vrijednost u svakoj od izračunatih funkcija L i R. Složenost ovog rješenja je $O\ (N * N)$.

# BOJE

*Predložili: Frane Kurtović i Ante Đerek*
*Potrebno znanje: dvodimenzionalna polja, stringovi, for petlja, while petlja*

Zadatak rješavamo direktnom implementacijom postupka brisanja boja i padanja točaka koja je opisana u tekstu zadatka. Ploču za igru možemo pamtiti kao dvodimenzionalno polje znakova te problem rješavamo u tri koraka.

- U prvom koraku brišemo sve točke zadane boje tako da pomoću dvije for petlje prolazimo kroz svaku poziciju te svaki znak boje koja se briše zamijenimo sa posebnim znakom koji označava pobrisanu točku (npr. znakom $).
- U drugom koraku simuliramo padanje točaka. Postupak padanja jedne točke za jedno mjesto dolje je jednostavan: Ako se na nekoj poziciji u polju nalazi slovo, a direktno ispod znak $, onda zamjenjujemo ta dva znaka. Kako bi simulirali jedan trenutak padanja ovaj postupak napravimo za svaku poziciju u polju, nužno od zadnjeg reda prema prvome. Prethodni postupak je dovoljno ponoviti četiri puta (jer je točno pet redaka u polju) kako bi u potpunosti odredili konačni izgled ploče.
- U zadnjem koraku zamijenimo znakove $ crnom bojom X.

# PREPORUKE

*Predložio: Frane Kurtović*
*Potrebno znanje: polja, stringovi, skupovi, sortiranje*

Zadatak rješavamo direktnom implementacijom postupka preporuke. Za svakog korisnika, listu njegovih omiljenih filmova možemo pamtiti u polju ili u nekoj sličnoj strukturi podataka (skup, lista).

- U prvom koraku za svakog korisnika računamo broj zajedničkih omiljenih filmova sa prvim korisnikom. Broj zajedničkih elemenata u dvije liste možemo jednostavno naći pomoću dvije for-petlje (sa svakom petljom prolazimo kroz jednu listu). Sve ove brojeve zapamtimo u polju.
- U drugom koraku pronađemo najveći broj zajedničkih elemenata.
- U trećem koraku stvaramo popis preporuka tako da razmatramo samo korisnike koji imaju najveći broj elemenata. Filmove koji su na njihovoj listi, a nisu na listi prvog korisnika opet tražimo pomoću dvije for-petlje slično kao i presjek.
- Na kraju, rezultat sortiramo upotrebom nekog od jednostavnih algoritama (npr. selection sort) ili korištenjem sort funkcije iz biblioteka.

# CODEC

*Predložio: Ante Đerek*
*Potrebno znanje: jednostavno dinamičko programiranje*

Zadatak rješavamo dinamičkim programiranjem. Označimo zadani niz brojeva sa $x_1$, …, $x_n$. Neka $S[a,b]$ označava minimalnu veličinu niza naredbi koja kodira brojeve od $x_a$ do $x_b$ tako da su prva i zadnja naredba tipa `I`, a sve ostale naredbe tipa `P` ili `B`. Obzirom da nakon naredbe `B` ne može doći naredba `P` niz naredbi je oblika `IPP…PBB…BI` pa vrijednost $S[a,b]$ možemo naći tako da ispitamo sve mogućnosti za broj `P` naredbi.

Neka $B[a,b]$ označava minimalnu veličinu niza naredbi koja kodira brojeve od $x_a$ do $x_b$ tako da su opet prva i zadnja naredba tipa `I`, ali nema dodatnih ograničenja. Rekurzivnu relaciju za $B$ možemo dobiti tako da razmatramo sve mogućnosti za poziciju prve sljedeće naredbe `I`:

$$B[a,b] = max(S[a,i] + B[i,b] - C[i])$$

gdje $i$ ide od $a+1$ do $b$, a $C[i]$ označava veličinu naredbe `I` $x_i$ (ovo je potrebno oduzeti jer ga dodajemo i u jednom i u drugom pribrojniku).

Pomoću ove rekurzivne relacije možemo implementirati rješenje dinamičkim programiranje, primjerice koristeći rekurzivnu funkciju sa memoizacijom.

**Za one koji žele više**

Postoji vrlo jednostavno linearno rješenje za ovaj zadatak – pronađite ga.