21. veljače 2014.

Infokup 2013

Županijsko natjecanje / Osnovna škola (5. i 6. i 7. i 8. razred)

Algoritmi (Basic/Python/Pascal/C/C++)

OBJAŠNJENJA ZADATAKA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

MINISTARSTVO ZNANOSTI, OBRAZOVANJA I ŠPORTA REPUBLIKE HRVATSKE

5.1. Zadatak: UV

Autor: Nikola Dmitrović[1]

Ovo je zadatak jednostavne strukture te se u sličnim oblicima već pojavljivao na natjecanjima. Za svaku od ponuđenih vrijednosti, naredbom odlučivanja treba definirati odgovarajući ispis. Samo je dodatno trebalo paziti i provjeriti što se događa kada UV indeks prijeđe preko vrijednosti 11.

Ovdje je dobar trenutak upozoriti na nužnost ispisa ponuđenih poruka u identičnom obliku kako su zadane. Jer svaka automatska evaluacija putem Evaluatora za pogreške tog tipa neće imati razumijevanja.

Programski kod (pisan u psudojeziku)

```
ulaz (UV);
ako je UV == 1 onda
      izlaz ('Niska opasnost');
ako je UV == 2 onda
      izlaz ('Niska opasnost');
ako je UV == 3 onda
      izlaz ('Umjerena opasnost');
ako je UV == 4 onda
      izlaz ('Umjerena opasnost');
ako je UV == 5 onda
      izlaz ('Umjerena opasnost');
ako je UV == 6 onda
      izlaz ('Visoka opasnost');
ako je UV == 7 onda
      izlaz ('Visoka opasnost');
ako je UV == 8 onda
      izlaz ('Vrlo visoka opasnost');
ako je UV == 9 onda
      izlaz ('Vrlo visoka opasnost');
ako je UV == 10 onda
      izlaz ('Vrlo visoka opasnost');
ako je UV >= 11 onda
      izlaz ('Ekstremna opasnost');
```

Potrebno znanje: naredba učitavanja i ispisivanja, naredba odlučivanja

Kategorija: ad hoc

---

[1] član Državnog povjerenstva, voditelj natjecanja u kategoriji primjena algoritama (programski jezik Basic/Python/Pascal/C/C++), profesor informatike u XV. gimnaziji, Zagreb

5.2. Zadatak: Tarifa — Autor: Nikola Dmitrović

Analizirajmo zadatak tako što ćemo ga čitati i prevoditi u programski kod pisan u pseudojeziku. Prvo ćemo učitati broj poslanih poruka i cijenu jedne poslane poruke.

```
ulaz (N);
ulaz (C);
```

Prvi uvjet iz zadatka kaže: „**prvih 100** poslanih SMS poruka tijekom jednog mjeseca je **besplatno**“. To znači:

```
ako je N <= 100 onda
      ukupno := 0 * C;
```

Drugi uvjet iz zadatka kaže: „od **sljedećih 100** poslanih poruka operater će naplatiti slanje **svake treće** poruke“. Ovaj se uvjet provjerava kada je poslano između 101 i 200 poruka te znači da će se naplatiti slanje 103., 106., 109., 112., 115. itd. poruke. Općenito, ukupan broj poslanih poruka koji prelazi preko 100 (jer je prvih 100 besplatno) treba cjelobrojno podijeliti s tri.

```
ako je (N >= 101) i (N <= 200) onda
      ukupno := (0 + (N - 100) div 3) * C;
```

Treći uvjet iz zadatka kaže: „od **sljedećih 100** poslanih poruka operater će naplatiti slanje **svake druge** poruke“. Ovaj se uvjet provjerava kada je poslano između 201 i 300 poruka. Prvih 100 poruka bilo je besplatno, od sljedećih 100 poruka naplaćeno je slanje njih 33 (100 div 3 = 33) i za kraj ukupan broj poruka koje prelaze preko 200 treba cjelobrojno podijeliti s dva.

```
ako je (N >= 201) i (N <= 300) onda
      ukupno := (0 + 33 + (N - 200) div 2) * C;
```

Posljednji uvjet iz zadatka kaže: „nakon **navedenih 300** poslanih poruka, operater će naplatiti slanje **svake** SMS poruke“. Treba uočiti da će prvih 100 poslanih poruka biti besplatno, od drugih 100 naplatit će se slanje njih 33 (100 div 33), od sljedećih 100 naplatit će se slanje njih 50 (100 div 2) i zatim svaka sljedeća poslana poruka.

```
ako je (N >= 301) onda
      ukupno := (0 + 33 + 50 + (N - 300)) * C;
```

I za kraj, ispišemo ukupnu cijenu.

```
izlaz (ukupno);
```

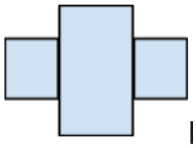
Potrebno znanje: naredba učitavanja i ispisa, naredba odlučivanja, logička analiza uvjeta

Kategorija: ad hoc

5.3. Zadatak: Ploča

Autor: Adrian Satja Kurdija[2]

Zadatak je moguće riješiti na više načina; ovdje opisujemo dva.



Podijelimo ploču na tri pravokutnika kao na sljedećoj shemi:

Za **svaki** od ovih pravokutnika izračunat ćemo ukupan broj polja u njemu te broj bijelih i broj crnih polja u njemu; na koncu zbrajanjem ovih brojeva za sva tri pravokutnika dobivamo tražene brojeve za cijelu ploču.

Da bismo riješili problem za pravokutnik, potrebno je imati sljedeće podatke:

- njegove dimenzije,
- boju njegova gornjeg-lijevog polja.

Sa slike iz teksta zadatka zaključujemo:

- dimenzije **lijevog** pravokutnika su C i E, dimenzije **srednjeg** su A i (B + E + F), a dimenzije **desnog** su D i E;
- gornje-lijevo polje **srednjeg** pravokutnika u zadatku je označeno zvjezdicom i uvijek je bijelo, a za bočne pravokutnike vrijedi sljedeće:
    - u **lijevom** pravokutniku, gornje-lijevo polje bijelo je ako je B + C paran broj (inače je crno),
    - u **desnom** pravokutniku, gornje-lijevo polje bijelo je ako je B + A paran broj (inače je crno).

Zašto je tako? Zato što su B + C i B + A duljine putova od polja označenog zvjezdicom do traženih gornjih-lijevih polja lijevog i desnog pravokutnika. Na takvom putu naizmjence se pojavljuju bijela i crna polja. Ako je put parne duljine, onda smo krenuvši od bijelog polja ponovno stigli u bijelo polje, a inače smo na crnome polju.

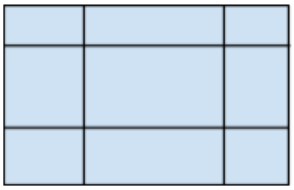
U redu, sada za pojedini pravokutnik znamo potrebne podatke; kako u njemu izbrojiti tražena polja? Ukupan broj polja, jasno, umnožak je dimenzija. Ako je ukupan broj polja u pravokutniku paran, broj bijelih i broj crnih polja u njemu međusobno su jednaki (pa iznose polovinu ukupnog broja). Ako je pak ukupan broj polja u pravokutniku neparan, za jedan je veći broj polja one boje koja se javlja u gornjem-lijevom polju pravokutnika.

Ovaj je pristup najlakše implementirati uz pomoć funkcije koja kao parametre prima dimenzije pravokutnika i boju gornjeg-lijevog polja, a vraća tražena tri broja za taj pravokutnik.

Drugi pristup.

Ovaj je pristup malo jednostavniji, ali koristi dvije dosjetke. Nadopunimo ploču tako da dobijemo veliku pravokutnu ploču sa B + E + F redaka i C + A + D stupaca, kao na sljedećoj shemi:

[2] niv. bacc. math., student PMF-MO, višestruki državni prvak iz matematike i informatike, osvajač srebrne i brončane medalje na IOI-ju te dviju brončanih medalja na IMO olimpijadi znanja



Dvostrukom for-petljom proći ćemo po svim poljima (R, S) velike ploče (R ide od 1 do B + E + F; unutar te petlje S ide od 1 do C + A + D). Za svako od tih polja pitamo se:

- pripada li polje originalnoj ploči?
- ako pripada, koje je boje?

Ako malo razmislimo, zaključujemo:

- polje (R, S) pripada originalnoj ploči ako je R između B + 1 i B + E, ili ako je S između C + 1 i C + A;
- boja bilo kojeg polja (R, S) ovisi samo o parnosti zbroja R + S. To vrijedi zato što je za sva bijela polja zbroj R + S paran, a za crna polja neparan, ili obrnuto. Ako je dakle zbroj R + S iste parnosti kao zbroj 1 + (C + 1) koji odgovara polju označenom zvjezdicom, onda je polje (R, S) bijele boje; inače je crne.

U dvjema varijablama pratimo broj bijelih i broj i crnih polja koje smo prošli. Na koncu ih zbrojimo da dobijemo i ukupan broj polja.

Programski kod (pisan u psudojeziku)

```
ulaz(A); ulaz(B); ulaz(C); ulaz(D); ulaz(E); ulaz(F);
bijelih := 0;
crnih := 0;
za R := 1 do B + E + F činiti
    za S := 1 do C + A + D činiti
        ako je (B + 1 <= R i R <= B + E) ili (C + 1 <= S i S <= C + A)
        {
            ako je (R + S) mod 2 = (1 + C + 1) mod 2 onda
                bijelih := bijelih + 1;
            inače
                crnih := crnih + 1;
        }
izlaz(bijelih + crnih);
izlaz(bijelih, crnih);
```

Na kraju spomenimo da na ploči iz zadatka, ukupan broj bijelih polja može od broja crnih polja biti veći za 3, 2 ili 1; mogu biti i jednaki, ali i broj crnih polja može od broja bijelih polja biti veći za 1 ili 2. Natjecatelji su u slučajevima u kojima je broj crnih polja veći od broja bijelih polja imali najviše netočnih rezultata. Pozivamo čitatelje da se zabave i pronađu primjere za svaki od navedenih šest slučajeva.

Potrebno znanje: uočavanje pravilnosti, naredba odlučivanja, naredba ponavljanja

Kategorija: ad hoc

6.1. Zadatak: Doba

Autor: Nikola Dmitrović

Jednostavnim postavljanjem uvjeta na zadani dan i mjesec iz datuma, možemo odrediti godišnje doba kojem taj datum pripada. Ovdje je dobar trenutak upozoriti na nužnost ispisa ponuđenih poruka u identičnom obliku kako su zadane. Jer svaka automatska evaluacija putem Evaluatora za pogreške tog tipa neće imati razumijevanja.

Programski kod (pisan u psudojeziku)

```
ulaz (D);
ulaz (M);
ako je (M < 3) onda izlaz ('Zima');
ako je (M = 3) i (D < 20) onda
      izlaz ('Zima')
inače
      izlaz ('Proljece');
ako je (M > 3) i (M < 6) onda izlaz ('Proljece');
ako je (M = 6) i (D < 21) onda
      izlaz ('Proljece')
inače
      izlaz ('Ljeto');
ako je (M > 6) i (M < 9) onda izlaz ('Ljeto');
ako je (M = 9) i (D < 22) onda
      izlaz ('Ljeto')
inače
      izlaz ('Jesen');
ako je (M > 9) i (M < 12) onda izlaz ('Jesen');
ako je (M = 12) i (D < 21) onda
      izlaz ('Jesen')
inače
      izlaz ('Zima');
```

Potrebno znanje: naredba odlučivanja, logička analiza uvjeta

Kategorija: ad hoc

6.2. Zadatak: Kraljica

Autor: Matija Milišić[3]

Zadatak je moguće riješiti na nekoliko načina, ovdje ćemo opisati dva.

Stupci u tekstu zadatka označeni slovima a-h u ovom opisu bit će označeni brojevima 1-8, tim redom. Dodatno, 1. dijagonalu iz teksta zadatka zvat ćemo sporednom, a 2. dijagonalu glavnom. U oba pristupa potrebno je prvo string iz ulaza (početno polje) razdvojiti na dva broja: broj retka i broj stupca.

Prvi pristup.

U ovom pristupu do rješenja dolazimo formulama. Izvedimo stoga zasebno formule za broj napadnutih polja u istom retku, stupcu te istoj glavnoj i sporednoj dijagonali.

Primijetimo da kraljica uvijek (bez obzira na početno polje) napada točno sedam polja u istom retku i sedam polja u istom stupcu. Jedino polje koje ne napada je upravo ono na kojem se već nalazi. To daje ukupno 14 polja u ova dva slučaja.

Da bi odredili broj napadnutih polja na istoj glavnoj dijagonali, moramo odrediti ukupan broj polja na toj dijagonali. Glavne dijagonale su dijagonale koje se protežu u smjeru gore-lijevo prema dolje-desno. Vrijedi: zbroj retka i stupca svakog polja glavne dijagonale je isti. Budući da je taj zbroj za svaku dijagonalu jedinstven nazovimo ga kodom glavne dijagonale.

Tako recimo glavna dijagonala koda devet proteže se od polja (8,1) do polja (1,8). Ukupan broj polja na toj dijagonali je osam, najveći od svih glavnih dijagonala. Broj polja glavnih dijagonala smanjuje se udaljavanjem od te najveće. Također, možemo primijetiti da se broj polja na glavnoj dijagonali smanji točno za apsolutnu razliku koda najveće i trenutne dijagonale. Uočavanjem tih karakteristika lako je doći do broja napadnutih polja na istoj glavnoj dijagonali.

Za određivanje broja napadnutih polja na istoj sporednoj dijagonali imat ćemo slična zapažanja. Sporedne dijagonale su dijagonale koje se protežu u smjeru dolje-lijevo prema gore-desno. Vrijedi: razlika retka i stupca svakog polja sporedne dijagonale je ista i taj broj možemo nazvati kodom sporedne dijagonale.

Sporedna dijagonala koda nula proteže se od polja (1,1) do polja (8,8). Ukupan broj polja na toj dijagonali je osam, najveći od svih sporednih dijagonala. Broj polja sporednih dijagonala također se smanjuje udaljavanjem od te najveće. Taj broj smanji se točno za apsolutnu razliku koda najveće i trenutne sporedne dijagonale (za koju određujemo broj polja).

Ukupan broj napadnutih polja jednak je: 14 (u istom retku ili stupcu) + broj polja na istoj glavnoj dijagonali - 1 (ne brojimo polje na kojem se kraljica nalazi) + broj polja na istoj sporednoj dijagonali - 1 (ne brojimo polje na kojem se kraljica nalazi).

[3] student Fakulteta elektrotehnike i računarstva, osvajač brončane medalje na IOI 2011., srebrne medalje na CEOI 2011. te dviju brončanih medalja na IMO 2011. i IMO 2012, dvostruki državni prvak iz informatike (2. i 4. razred)

Programski kod (pisan u psudojeziku)

```
ulaz(stupac, redak);
brojac := 14;
kod_glavne := redak + stupac;
kod_sporedne := redak - stupac;
brojac := brojac + 7 - apsolutna_vrijednost(9 - kod_glavne);
brojac := brojac + 7 - apsolutna_vrijednost(0 - kod_sporedne);
izlaz(brojac);
```

Drugi pristup.

Ovaj pristup je jednostavniji, ali zahtijeva poznavanje višestrukih naredbi ponavljanja. Proći ćemo po svim poljima šahovske ploče i za svako od njih odredit ćemo je li napadnuto. Koristit ćemo zapažanja iz prvog pristupa koja kažu da polja na istoj glavnoj dijagonali imaju isti zbroj retka i stupca, a polja na istoj sporednoj dijagonali imaju istu razliku retka i stupca.

Na početku brojač napadnutih polja postavimo na nulu. Sada jednom petljom prolazimo po retcima, a drugom (ugniježđenom) po stupcima kako bi prošli kroz sva polja šahovske ploče. Neka je početno polje (P_x,P_y), a trenutno (T_x,T_y). Trenutno polje je napadnuto ako zadovoljava neki od sljedećih uvjeta:

- nalazi se u istom retku ($P_x = T_x$)
- nalazi se u istom stupcu ($P_y = T_y$)
- nalazi se na istoj glavnoj dijagonali ($P_x+P_y = T_x+T_y$)
- nalazi se na istoj sporednoj dijagonali ($P_x-P_y = T_x-T_y$)

Ako trenutno polje zadovoljava nekih od navedenih uvjeta povećamo brojač napadnutih polja. Po završetku petlji ispišemo vrijednost tog brojača.

Potrebno znanje: višestruka naredba ponavljanja, naredba odlučivanja

Kategorija: ad hoc

6.3. Zadatak: Liftovi

Autor: Antun Razum[4]

Uvodne napomene: U opisu rješenja će se povremeno u zagradama pojavljivati riječi u kurzivu. To su imena varijabli (koja će se koristiti u programskom kodu) na koje se odnosi opisani pojam prije zagrade.

Za rješavanje ovog zadatka bit će nam potrebne dvije pomoćne varijable u kojima ćemo držati katove na kojima se nalaze liftovi (*mali* i *veliki*). Na početku ćemo obje varijable postaviti na nulu (što označava da se liftovi nalaze na prizemlju). Uz te dvije varijable bit će nam potrebna još jedna varijabla (*udaljenost*) u kojoj će biti ukupna udaljenost koju su prešla oba lifta. Promotrimo sada kako ćemo riješiti zadatak.

Proći ćemo petljom kroz sve osobe. Koristit ćemo sada još dvije pomoćne varijable koje će nam označavati odakle (*odakle*) i dokle (*dokle*) osoba putuje. Ako osoba putuje prema gore varijabla *odakle* biti

[4] student Fakulteta elektrotehnike i računarstva, osvajač srebrnih medalja na IOI 2012. i CEOI 2012., dvostruki državni prvak iz informatike (1. i 3. razred)

će postavljena na nulu, a *dokle* na kat na koji osoba putuje, ako pak putuje prema dolje varijabla *odakle* biti će postavljena na kat na kome se osoba trenutno nalazi, a *dokle* bit će postavljena na nulu. Možemo primijetiti da će ona varijabla od te dvije koja nije postavljena na nulu biti postavljena na učitani broj *K* (iz ulaznih podataka) za tu osobu. Sada možemo jednostavno utvrditi koji je lift bliže osobi tako što samo izračunamo apsolutne razlike katova liftova (pomoćne varijable *mali* i *veliki*) i varijable *odakle*. Ako su razlike iste uzet ćemo manji lift (kao što i piše u zadatku). Primijetimo da nam ne smeta ako se neki lift nalazi na istom katu kao i osoba koja ga poziva jer je tada promatrana razlika nula što je odgovarajuća interpretacija zadatka. Sada ćemo za lift koji je bliže izračunati koliku udaljenost treba prevaliti kako bi došao do kata na kome se nalazi osoba. To je upravo ova razlika koju smo maloprije računali. Osim ove udaljenosti ukupnoj pređenoj udaljenosti (*udaljenost*) treba dodati i udaljenost kata na kome se osoba nalazi i kata na koji lift treba odvesti tu osobu. Primijetimo da je to upravo apsolutna razlika varijabli *odakle* i *dokle*. Nakon dodavanja ovih udaljenosti ukupnoj pređenoj udaljenosti moramo pomaknuti lift koji smo koristili tj. trebamo postaviti pomoćnu varijablu koja označava kat na kome se on nalazi (*mali* ili *veliki*) na varijablu *dokle*.

Nakon što smo na gore opisan način prošlo kroz sve osobe potrebno je samo ispisati ukupnu prijeđenu udaljenost (*udaljenost*).

Programski kod (pisan u psudojeziku)

```
mali := 0;
veliki := 0;
udaljenost := 0;
ulaz(n);
za i := 1 do n činiti
{
    ulaz(k, smjer);
    ako je smjer = gore onda
    {
        odakle := 0;
        dokle := k;
    }
    inače
    {
        odakle := k;
        dokle := 0;
    }
    daljina_malog := apsolutna_vrijednost(mali - odakle);
    daljina_velikog := apsolutna_vrijednost(veliki - odakle);
    daljina_puta := apsolutna_vrijednost(dokle - odakle);
    ako je daljina_malog <= daljina velikog onda
    {
        udaljenost := udaljenost + daljina_malog + daljina_puta;
        mali := dokle;
    }
```

```
    inače
    {
        udaljenost := udaljenost + daljina_velikog + daljina_puta;
        veliki := dokle;
    }
}
izlaz(udaljenost);
```

Potrebno znanje: naredba ponavljanja, naredba odlučivanja

Kategorija: ad hoc simulacija

7.1. Zadatak: Boje — Autor: Adrian Satja Kurdija

Definirajmo niz od osam stringova koji predstavljaju retke ploče, pri čemu znakovi "y", "b", "r", "g" predstavljaju žutu, plavu, crvenu i zelenu boju. Nakon toga rješenje je jednostavno ispisati boju koja odgovara S-tom znaku R-tog stringa.

Naravno, zadatak se mogao riješiti i korištenjem naredbi odlučivanja, što je mnogo složenije jer valja pokriti mnogo slučajeva.

Programski kod (pisan u psudojeziku)

```
a[1] := "yyyyyyyr";
a[2] := "bbbbbbyr";
a[3] := "bggggbyr";
a[4] := "bgrrgbyr";
a[5] := "bgrybbyr";
a[6] := "bgryyyyr";
a[7] := "bgrrrrrr";
a[8] := "bgggggggg";

boja['y'] := 'ZUTA';
boja['b'] := 'PLAVA';
boja['r'] := 'CRVENA';
boja['g'] := 'ZELENA';

ulaz(R);
ulaz(S);
znak_boje := a[R][S - 1];  // S - 1 jer znakove u stringu brojimo od nule
izlaz(boja[znak_boje]);
```

Potrebno znanje: stringovi ili dvodimenzionalno polje ili naredba odlučivanja

Kategorija: ad hoc

7.2. Zadatak: Imenik

Autor: Nikola Dmitrović

Zadatak možemo riješiti promatrajući zadane brojeve kao stringove, ali i kao prirodne brojeve. Ako se odlučimo za prirodne brojeve, trebamo osmisliti algoritam kojim možemo odrediti je li dva broja **počinju istim slijedom** znamenki. To je jednostavno odrediti ako se sjetimo da do vodećih znamenki u broju možemo doći gledajući količnik koji dobijemo kada cjelobrojno podijelimo taj broj s odgovarajućom potencijom broja 10.

Npr: za zadani broj 325643 vrijedi: 3 = 325643 div 100000; 32 = 325643 div 10000; 325 = 325643 div 1000; 3256 = 325643 div 100; 32564 = 325643 div 10; 325643 = 325643 div 1.

Kako je u zadatku definirano da će zadani brojevi biti šesteroznamenkasti, tada je moguće i rješenje u kojem će se pojedinačno provjeriti svaka situacija. Ali, ovdje ćemo pokazati rješenje koje uključuje nizive i naredbu ponavljanja.

Programski kod (pisan u psudojeziku)

```
ulaz (B);
ulaz (N);
ulaz (imenik); // imenik je niz duljine N u kojem su zapisani telefonski brojevi
pot = 100000; // početna potencija od deset s kojom ćemo dijeliti brojeve
za i := 1 do 6 činiti // kako bi provjerili svaku kombinaciju početnog slijeda
{
      dio_od_B := B div pot; // početni slijed upisanih brojeva od broja B
      koliko := 0; // broj brojeva koji zadovoljavaju zadani uvjet
      za j := 1 do N činiti
      {
            dio_od_broja := imenik[j] div pot; //početni slijed od broja B
            ako je dio_od_B = dio_od_broja onda
                  koliko := koliko + 1;
      }
      izlaz(koliko);
      pot := pot div 10;
}
```

Potrebno znanje: naredba odlučivanja, naredba ponavljanja, niz, algoritam određivanja početnog slijeda znamenki u broju

Kategorija: ad hoc

7.3. Zadatak: Brodovi

Autor: Adrian Satja Kurdija

Blokirana polja (polja na koja ne možemo staviti dio broda) nisu samo ona koja su već zauzeta, nego i njima susjedna polja (zbog uvjeta o dodirivanju). Ako znamo sva polja koja su u ovom smislu blokirana, ne moramo više paziti na dodirivanje, nego samo na to da stavimo brod na **K** uzastopnih neblokiranih polja.

Načinit ćemo stoga novu ploču, u kojoj ćemo staviti prepreke ne samo na mjesta koja su zauzeta u originalnoj ploči, nego i na njima susjedna polja. Da bismo to učinili, moramo biti u stanju za svako zauzeto polje originalne ploče pronaći njegove susjede (kojih može biti osam, ili manje ako je polje na rubu). Kako to na jednostavan način učiniti? Za polje (redak, stupac) kandidati su za susjede sljedeći:

(redak - 1, stupac - 1)	(redak, stupac - 1)	(redak + 1, stupac)
(redak - 1, stupac)	(redak, stupac + 1)	(redak + 1, stupac + 1)
(redak - 1, stupac + 1)	(redak + 1, stupac - 1)	

Za svako od ovih polja valja provjeriti je li unutar ploče (tj. jesu li mu koordinate veće od nule i ne premašuju **R** odnosno **S**); ako jest, valja ga u novoj ploči označiti blokiranim. Pisanje svih ovih uvjeta naporno je i kompliciranо. Srećom, posao možemo veoma skratiti. U tome će nam pomoći dva pomoćna niza:

pomak_retka = (-1, -1, -1, 0, 0, 1, 1, 1),

pomak_stupca = (-1, 0, 1, -1, 1, -1, 0, 1).

For-petljom proći ćemo varijablom **smjer** od 1 do 8 te za svaki takav **smjer** izračunati:

redak_susjeda = redak + pomak_po_retku[**smjer**],

stupac_susjeda = stupac + pomak_po_stupcu[**smjer**].

Ako malo razmislite, vidjet ćete da na ovaj način stvarno biramo svih osam gore navedenih susjeda. Unutar petlje još valja provjeriti spomenute uvjete.

Nakon ovoga postupka zanima nas na koliko načina možemo zauzeti **K** uzastopnih neblokiranih polja u novoj ploči.

Postavljamo li brod vodoravno, odabrat ćemo sva moguća početna polja broda (redak, stupac) i potom, for-petljom za **k** od 0 do **K** - 1, provjeriti je li polje (redak, stupac + **k**) unutar ploče i je li blokirano. Ako nijedno od tih **K** polja nije blokirano, rješenje povećavamo za jedan jer brod možemo postaviti na ovaj način.

Za okomiti smjer činimo posve jednako, osim što provjeravamo polja (redak + **k**, stupac). Za jednomotorac okomiti smjer ne valja provjeravati jer taj brod zauzima samo jedno polje.

Programski kod (pisan u psudojeziku)

```
POMAK_RETKA := [-1, -1, -1, 0, 0, 1, 1, 1];
POMAK_STUPCA := [-1, 0, 1, -1, 1, -1, 0, 1];
ulaz(R, S);
ulaz(ploca);  // dvodimenzionalno polje znakova
ulaz(K);
```

```
za redak := 1 do R činiti
      za stupac := 1 do S činiti
            ako je ploca[redak][stupac] = '#' onda
            {
                  ploca2[redak][stupac] := '#';
                  za smjer := 1 do 8 činiti
                  {
                        redak_susjeda := redak + POMAK_RETKA[smjer];
                        stupac_susjeda := stupac + POMAK_STUPCA[smjer];
                        ako redak_susjeda >= 1 i redak_susjeda <= R i
                  stupac_susjeda >= 1 i stupac_susjeda <= S onda
                              ploca2[redak_susjeda][stupac_susjeda] := '#';
                  }
            }
rjesenje := 0;
za redak := 1 do R činiti
      za stupac := 1 do S činiti
      {
            slobodnih := 0;
            za k := 0 do K - 1 činiti
                  ako je stupac + k <= S i ploca2[redak][stupac + k] <> '#'
                        slobodnih := slobodnih + 1;
            ako je slobodnih = K onda
                  rjesenje := rjesenje + 1;
            slobodnih := 0;
            za k := 0 do K - 1 činiti
                  ako je redak + k <= R i ploca2[redak + k][stupac] <> '#'
                        slobodnih := slobodnih + 1;
            ako je slobodnih = K i K <> 1 onda
                  rjesenje := rjesenje + 1;
      }
izlaz(rjesenje);
```

Potrebno znanje: dvodimenzionalno polje, višestruka naredba ponavljanja, naredba odlučivanja

Kategorija: ad hoc

8.1. Zadatak: Veslanje Autor: Nikola Dmitrović

Najvažniji dio zadatka je rečenica da u prvih devet finala nastupa **jednak, najveći mogući** broj veslača. To znači da se ukupan broj natjecatelja treba cjelobrojno podijeliti s devet. Na taj način ćemo saznati broj natjecatelja u svakom od prvih 9 finala. Za J finale nije potrebno odrediti broj natjecatelja u njemu jer zadatak garantira da će ulazni podaci realno opisivati moguće rezultate te će za to finale biti dovoljno gledati trenutnu poziciju natjecatelja u njemu.

Sada ostaje samo odrediti kako oznaku finala pretvoriti u brojčanu oznaku. To možemo uraditi naredbama odlučivanja ili korištenjem svojstava ASCII tablice.

Programski kod (pisan u psudojeziku)

```
ulaz (N);
ulaz (Z);
ulaz (P);
koliko_u_finalu := N div 9;
izlaz ((ord(Z)-65) * koliko_u_finalu + P)
// ord(Z) – redni broj znaka Z u ASCII tablici
```

Potrebno znanje: rad sa znakovima

Kategorija: ad hoc

8.2. Zadatak: Kraljice Autor: Antun Razum

Zadatku možemo pristupiti na više načina. Ovdje ćemo opisati dva moguća.

Uvodne napomene: U opisima rješenja nećemo se upuštati u objašnjavanje učitavanja podataka budući da se ono razlikuje za različite programske jezike. U oba opisa rješenja govorit ćemo o stupcima u numeričkim oznakama (kao i kod redaka), dakle stupci u zadatku označeni redom od "a" do "h" u rješenjima će biti označeni od "1" do "8".

Prvi pristup

U ovom ćemo pristupu rješavanja promatrati matricu dimenzija šahovske ploče. Svako polje u matrici biti će postavljeno na jedan ako je napadnuto od neke od kraljica. Opišimo sada kako ćemo iskonstruirati takvu matricu.

Prije svega postavimo sva polja matrice na nulu (čime označavamo da polja nisu napadnuta). Sada prođimo po pozicijama na kojima se nalaze kraljice. Pozicije kraljica možemo držati u polju tako da si ovo možemo olakšati korištenjem petlje ili, još jednostavnije, pozicije kraljica možemo učitavati na početku petlje u kojoj promatramo napadnuta polja (ovaj način je korišten u programskom kodu). Na ovaj način ne moramo isti postupak pisati za svaku kraljicu. Za svaku kraljicu sada je potrebno proći u svih 8 osnovnih smjerova krenuvši od polja susjednog polju na kojem se nalazi sama kraljica pa sve do ruba šahovske ploče. Ovo jednostavno možemo implementirati koristeći pomoćne nizove za pomak retka i stupca (vidi programski kod). Za svaku kraljicu dakle prođemo jednom petljom po 8 smjerova, a drugom (ugniježđenom) petljom prolazimo po poljima u tom smjeru sve dok ne dođemo do ruba ploče. Dok prolazimo po poljima postavljamo ih na vrijednost jedan (što označava da je polje napadnuto).

Nakon što smo na gore opisan način iskostruirali traženu matricu samo prođemo po njoj i izbrojimo napadnuta polja tj. polja koja su postavljena na jedan. Broj tih polja rješenje je koje ispisujemo.

Drugi pristup

U ovom ćemo pristupu proći po svim poljima šahovske ploče i za svako od njih provjerit ćemo je li napadnuto od neke od kraljica.

Na početku brojač napadnutih polja postavimo na nulu. Sada jednom petljom prolazimo po retcima, a drugom (ugniježđenom) po stupcima kako bi prošli kroz sva polja šahovske ploče. Za svako polje sada prođemo po sve tri kraljice. Ovo, kao i u prvom načinu, jednostavno možemo napraviti tako da pozicije kraljica držimo u polju kako bi po njima mogli jednostavno proći petljom. Sada za svaku kraljicu moramo provjeriti nalazi li se promatrano polje u istom retku, stupcu ili na istoj dijagonali kao ta kraljica. Ako smo ustanovili da je neki od ovih uvjeta zadovoljen (objašnjeno u daljnjem tekstu), povećamo brojač napadnutih polja i možemo odmah ići na iduće polje na šahovskoj ploči.

Razmotrimo sada kako ćemo provjeriti spomenute uvjete za napadnutost polja. Slučajevi kada je polje u istom retku i stupcu kao i kraljica jednostavni su - samo provjerimo je li redak odnosno stupac polja isti kao i polja na kojem se nalazi promatrana kraljica. Za dijagonale je postupak malo složeniji. Nazovimo dijagonale koje se protežu u smjeru gore-lijevo prema dolje-desno glavnim dijagonalama, a one okomite na njih nazovimo sporednim. Možemo sada primijetiti da je zbroj oznake retka i stupca na glavnim dijagonalama konstantan. Isto tako možemo primijetiti da je razlika oznake retka i stupca na sporednim dijagonalama također konstantna. Ovim primjedbama uvjet za dijagonale postaje vrlo jednostavan. Da bi se neko polje nalazilo na istoj glavnoj dijagonali kao i neka kraljica zbroj oznaka retka i stupca mora mu biti jednak zbroju oznaka retka i stupca polja na kome se nalazi ta kraljica. Isto vrijedi i za uvjet za sporednu dijagonalu uz razliku da ne promatramo zbroj oznaka retka i stupca nego njihovu razliku.

Naposlijetku samo ispišemo brojač napadnutih polja koji smo koristili kroz zadatak.

Programski kod (pisan u psudojeziku) – prvi pristup

```
POMAK_RETKA := [-1, -1, 0, 1, 1, 1, 0, -1];
POMAK_STUPCA := [0, 1, 1, 1, 0, -1, -1, -1];
// postavimo sve vrijednosti matrice na 0
za i := 1 do 3 činiti
{
    ulaz(stupac, redak);
    za smjer := 1 do 8 činiti
    {
        udaljenost := 1;
        dok je (1)
        {
            novi_redak := redak + udaljenost * POMAK_RETKA[smjer];
            novi_stupac := stupac + udaljenost * POMAK_STUPCA[smjer];
```

```
            ako je (novi_redak < 1) ili (novi_redak > 8) ili (novi_stupac < 1)
      ili (novi_stupac > 8)
                    prekini petlju;
                matrica[novi_redak][novi_stupac] := 1;
                udaljenost := udaljenost + 1;
            }
        }
}
broj_napadnutih := 0;
za i := 1 do 8 činiti
    za j := 1 do 8 činiti
        broj_napadnutih := broj_napadnutih + matrica[i][j];
izlaz(broj_napadnutih);
```

Potrebno znanje: dvodimenzionalno polje, višestruka naredba ponavljanja, naredba odlučivanja

Kategorija: ad hoc simulacija

8.3. Zadatak: Igra — Autor: Matija Milišić

Opišimo najprije rješenje koje nosi 80% ukupnog broja bodova, kada su sva slova u nizu Markovih uvjeta različita.

Prvi dio rješenja je Markove uvjete prikazati u drukčijem obliku, u kojem će nam biti lakše vršiti provjere nad Ivanovim stringovima. Koristit ćemo tri niza: *znak, donja, gornja*.

Niz *znak* dobivamo tako da iz svakog uvjeta uzmemo prvi znak, slovo X.

Niz *donja* za odgovarajuće slovo niza *znak* kazuje koliko je najmanje puta zaredom napisano.

Niz *gornja* za odgovarajuće slovo niza *znak* kazuje koliko je najviše puta zaredom napisano.

Svaki Markov uvjet zauzet će po jedno polje u svakom nizu. Konstruiranje nizova iz učitanih uvjeta pokazano je u tablici.

tip	Markov uvjet	*znak*	*donja*	*gornja*
1.	"X"	X	1	1
2.	"X{A}"	X	A	A
3.	"X{A,}"	X	A	1000
4.	"X{A,B}"	X	A	B

Uvjet "X{A,}" kaže da je slovo X barem A puta zaredom napisano i pritom ne kazuje ništa o gornjoj granici. Međutim, broj ponavljanja tog slova možemo ograničiti nekim dovoljno velikim brojem, većim od najveće duljine stringa u ulazu (ovdje je odabran broj 1000).

Konstruiranje nizova radimo na sljedeći način. Prolazimo petljom po Markovim uvjetima, za svaki uvjet određujemo tip i zatim odgovarajuće vrijednosti polja *znak*, *donja*, *gornja*.

Drugi dio rješenja je za svaku Ivanovu riječ odrediti odgovara li uvjetima koji se nalaze u tri niza. Na početku kažemo da riječ odgovara Markovim uvjetima (*odgovara := 1*). Zatim prolazimo kroz Ivanovu riječ na način da sva ista slova pređemo u jednom (istom) koraku. U tom koraku odredimo o kojem se slovu radi i koliko puta je zaredom napisano (*broj_istih*). Nakon toga slijedi provjera.

Provjerimo je li to slovo jednako trenutnom slovu u nizu znak te je li broj ponavljanja (broj_istih) unutar donje i gornje granice. Ako neki od tih uvjeta nije zadovoljen, kažemo da riječ ne odgovara Markovim uvjetima (*odgovara := 0*). Prije prelaska na sljedeće slovo Ivanove riječi, pomaknemo trenutnu poziciju u nizu znak da i ona pokazuje na sljedeće slovo.

Nakon prolaska kroz riječ ispišemo "DA" ako odgovara Markovim uvjetima, a "NE" u suprotnom.

Preostalih 20% bodova odnosi se na test primjere u kojima se u nizu Markovih uvjeta isto slovo može pojaviti više puta. Mogući problem nastaje kada se isto slovo pojavi u više uvjeta zaredom. Međutim, tada možemo zbrojiti donje granice istog slova u jednu novu donju, te gornje granice istog slova u jednu novu gornju. To je u programskom kodu implementirano pomicanjem brojaca jedino u slucaju kada je novo (*trenutno*) slovo različito od prethodnog. Primijetimo da će takvi novi zapisi uvjeta gdje više njih stopimo u jedan i dalje odgovarati početnim Markovim uvjetima. Slovo se ne smije pojaviti manje puta od zbroja donjih granica, niti više puta od zbroja gornjih granica. Također, lako se vidi da će uvjetima odgovarati bilo koji broj ponavljanja slova između nove donje i gornje granice. Time smo riješili i taj slučaj.

Ovaj zadatak je slojevit i omogućuje dobivanje parcijalnih bodova bez rješavanja općenitog slučaja. Tako se 10 bodova (1. red u bodovanju) moglo dobiti uspoređivanjem znak po znak stringa Markovih uvjeta sa stringovima Ivanovih riječi. Dodatnih 10 bodova moglo se dobiti rješenjem koji prepoznaje samo 2. tip uvjeta (2. i 5. red u bodovanju). Kada se to rješenje dopuni na način da prepoznaje i višeznamenkaste brojeve, dobiva se dodatnih 10 bodova.

Programski kod (pisan u psudojeziku)

```
ulaz(uvjeti);
i := 1;
brojac := 1;
dok je i < duljina_uvjeta činiti
    ako je trenutno_slovo_različito_od_prethodnog onda
            brojac := brojac+1;
    prepoznaj_tip_uvjeta;
    odredi znak[brojac];
    odredi donja[brojac];
    odredi gornja[brojac];
    postavi_'i'_da_pokazuje_na_sljedeći_uvjet;

ulaz(n);
```

```
za i := 1 do n činiti
      ulaz(Ivanova_riječ);
      odgovara := 1;
      trenutni := 1;
      j := 1;
      dok je j < duljina_riječi činiti
            broj_istih := koliko_je_puta_isto_slovo_zaredom_napisano();
            ako je Ivanov_niz[j] != znak[trenutni] onda
                  odgovara := 0;
            ako je broj_istih < donja[trenutni] onda
                  odgovara := 0;
            ako je broj_istih > gornja[trenutni] onda
                  odgovara := 0;
            trenutni := trenutni+1;
            j := j + broj_istih;

      ako je odgovara = 1 onda
            izlaz(DA);
      inače
            izlaz(NE);
```

Potrebno znanje: stringovi, polje, višestruka naredba ponavljanja, naredba odlučivanja

Kategorija: ad hoc stringovi