20. veljače 2015.

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

Primjena algoritama (Basic/Python/Pascal/C/C++)

OPISI ALGORITAMA

Programski kod za većinu zadataka bit će napisan u programskom jeziku Python. Na taj način želimo učenike upoznati s novim jezikom te ih motivirati da se dodatno posvete njegovom proučavanju.

U posebnoj mapi nalaze se dodatni kodovi koji predstavljaju neka dodatna, neslužbena rješenja autora zadataka.

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

HRVATSKI SAVEZ
INFORMATIČARA

Ministarstvo znanosti,
obrazovanja i sporta

5.1. Zadatak: Mladi — Ideja: Nikola Dmitrović

Kako su u zadatku strogo definirani odnosi između varijabli A, B, C i D, tada jednostavnim provjerama možemo utvrditi između koje dvije vrijednosti se nalazi vrijednost varijable E te ispisati prigodne poruke. U zadatku je bila jedna zamka. Naime, postavlja se pitanje u kojoj je mijeni Mjesec ako je zadani datum koji promatramo strogo manji od vrijednosti varijable A. Kako se izmjena mjesečevih mijena "neprestano odvija zadnjih nekoliko eona godina" tada je jasno da je Mjesec prije ulaska u mijenu "MLADI" bio u mijeni "ZADNJA".

Programski kod (pisan u Pythonu 3.4)

```
A = int(input())
B = int(input())l
C = int(input())
D = int(input())
E = int(input())
if E >= A and E < B: print("MLADI")
if E >= B and E < C: print("PRVA")
if E >= C and E < D: print("PUNI")
if E >= D and E <= 31: print("ZADNJA")
if E < A: print("ZADNJA")
```

Potrebno znanje: naredba odlučivanja

Kategorija: ad-hoc

5.2. Zadatak: Stube — Ideja: Adrian Satja Kurdija

For-petljom prolazimo po koracima, od prvog do posljednjeg. Unutar petlje, u varijablu koja će biti konačni rezultat pribrajamo broj preskočenih stuba, tj. duljinu trenutnoga koraka. Kolika je ta duljina?

Ona može biti 1, 2 ili 3 te ovisi o ostatku brojača petlje pri dijeljenju s tri. Dakle, možemo je odrediti korištenjem naredbi odlučivanja koje provjeravaju spomenuti ostatak. Ako brojač petlje ide od 0 do N - 1 (umjesto od 1 do N), duljinu koraka možemo računati i brže, kao ostatak brojača petlje pri dijeljenju s tri uvećan za jedan.

Duljinu trenutnoga koraka možemo računati i tako da je pamtimo u pomoćnoj varijabli koja na početku iznosi 1, a na kraju svakog koraka povećamo je za jedan, osim ako iznosi 3 pa je tada ponovno postavljamo na 1.

Zadatak je moguće riješiti i bez petlje, korištenjem formula, što ostavljamo čitateljima za vježbu.

Programski kod (pisan u Pythonu 3.4)

```
n = int(input())
s = 0
for i in range(n):
```

```
    s += i % 3 + 1
print(s)
```

Potrebno znanje: naredba ponavljanja

Kategorija: ad-hoc

5.3. Zadatak: Izbori — Ideja: Adrian Satja Kurdija

Zadatak je moguće elegantno riješiti sortiranjem niza parova (oznaka kandidata, broj glasova). Ali budući da je takvo rješenje izvan propisanoga opsega znanja za ovu razinu natjecanja, opisat ćemo rješenje koje koristi samo razmatranje slučajeva.

Najprije provjeravamo koji su kandidati prošli u drugi krug. Postoji šest slučajeva: to mogu biti AB, AC, AD, BC, BD ili CD. Provjera svakog od ovih slučajeva svodi se na provjeru imaju li odabrani kandidati više glasova od preostalih kandidata. Na primjer, da bismo provjerili jesu li u drugi krug prošli kandidati B i D, provjeravamo ima li B više glasova od A i C te ima li D više glasova od A i C.

Za svaki od gornjih slučajeva, ako se dogodio, valja pronaći pobjednika u drugome krugu, što činimo jednostavnom naredbom odlučivanja.

Programski kod (pisan u Pythonu 3.4)

```
a, b, c, d = map(int, input().split())
p, q = map(int, input().split())
if a > c and a > d and b > c and b > d:
    if p > q: print('A')
    else: print('B')
if a > b and a > d and c > b and c > d:
    if p > q: print('A')
    else: print('C')
if a > b and a > c and d > b and d > c:
    if p > q: print('A')
    else: print('D')
if b > a and b > d and c > a and c > d:
    if p > q: print('B')
    else: print('C')
if b > a and b > c and d > a and d > c:
    if p > q: print('B')
    else: print('D')
if c > a and c > b and d > a and d > b:
    if p > q: print('C')
    else: print('D')
```

Potrebno znanje: složena naredba odlučivanja

Kategorija: ad-hoc

6.1. Zadatak: Rub — Ideja: Adrian Satja Kurdija

Ako je R ≤ 2 ili S ≤ 2, lako je vidjeti da su sva polja na rubu pa ispisujemo R * S. Inače, uočimo da polja koja nisu na rubu (nego u unutrašnjosti) čine manju tablicu dimenzija (R - 2) x (S - 2), pa ako njihov broj oduzmemo od ukupnoga broja polja, dobivamo traženi broj rubnih polja.

Programski kod (pisan u Pythonu 3.4)

```
r = int(input())
s = int(input())
if r < 3 or s < 3:
    print(r * s)
else:
    print(r * s - (r - 2) * (s - 2))
```

Potrebno znanje: if-then-else, osnovne računske operacije

Kategorija: ad-hoc

6.2. Zadatak: Puni — Ideja: Nikola Dmitrović

Prvo treba uočiti da datumi početaka pojedinih mijena ne moraju biti poredani po redu, ali da se mijene izmjenjuju točno zadanim redoslijedom. Zbog toga ćemo zadatak riješiti u nekoliko koraka: prvo trebamo na neki način povezati datum i mijenu koja započinje na taj datum. Zatim trebamo datume poredati po veličini i zadržati povezivanje datuma i mijene. Na kraju, s nekoliko naredbi odlučivanja lako ćemo provjeriti u kojoj se mijeni trenutno nalazimo. Drugi dio zadatka sada nije problem i svodi se na jednostavno oduzimanje.

U zadatku je bila jedna zamka. Naime, postavlja se pitanje u kojoj je mijeni Mjesec ako je zadani datum koji promatramo strogo manji od datuma koji ima najmanju vrijednost. Kako se izmjena mjesečevih mijena "neprestano odvija zadnjih nekoliko eona godina" tada je jasno da je Mjesec u mijeni koja je započela na datum koji je zadnji po vrijednosti u mjesecu.

Programski kod (pisan u Pythonu 3.4)

```
A = int(input())
B = int(input())
C = int(input())
D = int(input())
E = int(input())
datumi = [A, B, C, D]
```

```
mijene = ["MLADI", "PRVA", "PUNI", "ZADNJA"]
# sortirajmo niz datumi pomoću osnovnog algoritma za sortiranje zamjenom
# svaki put kada treba napraviti zamjenu, istu zamjenu radimo i u nizu mijene
for i in range(3):
    for j in range(i, 4):
        if datumi[i] > datumi[j]:
            t = mijene[i]; t1 = datumi[i];
            mijene[i] = mijene[j]; datumi[i] = datumi[j]
            mijene[j] = t; datumi[j] = t1
if E >= datumi[0] and E < datumi[1]: print(mijene[0]); print(datumi[1] - E)
if E >= datumi[1] and E < datumi[2]: print(mijene[1]); print(datumi[2] - E)
if E >= datumi[2] and E < datumi[3]: print(mijene[2]); print(datumi[3] - E)
if E >= datumi[3] and E <= 31:       print(mijene[3]); print(31 - E)
if E < datumi[0]:                    print(mijene[3]); print(datumi[0] - E)
```

Pokažimo i jedno dodatno rješenje koje ne uključuje sortiranje, već samo jednostavno uspoređivanje.

Programski kod (pisan u Pythonu 2.7)

```
a = input()
b = input()
c = input()
d = input()
e = input()

if e >= a and e < b: print 'MLADI'
elif e >= b and e < c: print 'PRVA'
elif e >= c and e < d: print 'PUNI'
elif e >= d and e < a: print 'ZADNJA'
else:
    M = max(a, b, c, d)
    if M == a: print 'MLADI'
    if M == b: print 'PRVA'
    if M == c: print 'PUNI'
    if M == d: print 'ZADNJA'
if (e == 31):
    print 0
else:
    dana_do_promjene = 1
```

```
    e += 1
    while e != a and e != b and e != c and e != d and e != 31:
        dana_do_promjene += 1
        e += 1
    print dana_do_promjene
```

Potrebno znanje: naredba odlučivanja, naredba ponavljanja, sortiranje (za prvo rješenje)

Kategorija: ad-hoc

6.3. Zadatak: Isprika — Ideja: Nikola Dmitrović

U ovom zadatku imali smo tri dijela. U takvim situacijama, ako nije moguće riješiti cijeli zadatak, treba se koncentrirati na to da se točno riješe neki od dijelova cijelog zadatka. Npr. ukupno kašnjenje vlaka moglo se jednostavno riješiti na sljedeći način:

```
N = int(input())
kasnjenje_ukupno = 0
for i in range(N):
    S1, M1, S2, M2 = map(int,input().split())
    kasnjenje_ukupno += (S2 * 60 + M2) - (S1 * 60 + M1)
print(kasnjenje_ukupno)
```

Općenito, cjelokupno rješenje zadatka dobivamo pažljivim implementiranjem uvjeta koji se nalaze u tekstu zadatka.

Programski kod (pisan u Pythonu 3.4)

```
N = int(input()) # učitamo broj dana
# inicijaliziramo varijable
kasnjenje_ukupno = najvece = kada = prosjek = isprika = 0
for i in range(N): # ponovimo N puta
    S1, M1, S2, M2 = map(int,input().split()) # učitamo ulazne podatke
    # odredimo kašnjenje vlaka za i-ti dan
    kasnjenje_dan = (S2 * 60 + M2) - (S1 * 60 + M1)
    # ako je kašnjenje vlaka i-tog dana veće od prosjeka povećaj broj isprika
    if kasnjenje_dan >= prosjek:
        isprika += 1
    if kasnjenje_dan > najvece:
        najvece = kasnjenje_dan # zapamtimo to kašnjenje
            kada = i + 1 # zapamtimo koji se dan to dogodilo
    # povećamo ukupno kašnjenje za kašnjenje i-tog dana
```

```
    kasnjenje_ukupno += kasnjenje_dan
    # odredimo prosjek dosadašnjih kašnjenja
    prosjek = kasnjenje_ukupno / (i + 1)
print(kasnjenje_ukupno)
print(kada)
print(isprika)
```

Potrebno znanje: naredba ponavljanja, određivanje maksimalne vrijednosti

Kategorija: ad-hoc

7.1. Zadatak: Bingo — Ideja: Mislav Bradač[1]

Ovaj zadatak najlakše je riješiti koristeći jedan niz u koji ćemo bilježiti koji su brojevi prozvani. Elementi tog niza na početku su inicijalizirani na nulu, a tijekom izvođenja programa jedinicom obilježavamo brojeve koji su pročitani.

Nakon što smo označili koji su brojevi pročitani, za svakog od Goranovih prijatelja izbrojit ćemo koliko on ima zaokruženih brojeva na kartici. To lako nalazimo tako da izbrojimo koliko ima jedinica u nekom intervalu našeg niza.

Još nam ostaje provjeriti koja osoba ima najviše zaokruženih brojeva, a to možemo s pomoću nekoliko naredbi grananja. Za detalje pogledajte implementaciju.

Mnogo učenika na natjecanju je imalo sljedeći algoritam: za svaki izvučeni broj provjeri nalazi li se na listiću pojedine osobe te ako se nalazi, povećaj brojač kojim brojimo koliko je ta osoba dosad zaokružila brojeva. Primijetimo da je taj algoritam pogrešan jer u slučaju da se više puta izvuče neki broj, njega ne smijemo više puta brojiti. Ovaj algoritam nosio je 30 bodova.

Programski kod (pisan u Pythonu 3.4)

```
def zaokruzeni(procitan, a, b) :
  zaokruzenih = 0
  for i in range(a, b) :
    zaokruzenih += procitan[i]
  return zaokruzenih
n = int(input())
i, s, g = map(int, input().split())
m = int(input())
procitan = [0] * 25
for j in range(m):
```

[1] student Fakulteta elektrotehnike i računarstva, osvajač dviju srebrnih i jedne brončane medalje s međunarodnih informatičkih olimpijada

```
  x = int(input())
  procitan[x] = 1
ni = zaokruzeni(procitan, i, i + n)
ns = zaokruzeni(procitan, s, s + n)
ng = zaokruzeni(procitan, g, g + n)
if ni > ns and ni > ng :
  print("Ivan")
elif ns > ni and ns > ng :
  print("Stjepan")
elif ng > ni and ng > ns :
  print("Gustav")
else :
  print("Goran")
```

Potrebno znanje: nizovi

Kategorija: ad-hoc

7.2. Zadatak: Papavski — Ideja: Matija Milišić

Pri rješavanju ovog zadatka potrebno je za svaku riječ odrediti je li papavska ili nije, a zatim izvršiti odgovarajuću pretvorbu.

Je li riječ papavska određujemo na sljedeći način. Prolazimo redom po slovima te riječi i kada naiđemo na samoglasnik provjerimo jesu li odmah nakon njega: malo slovo p i ponovno taj samoglasnik. Ako nisu, otkrili smo da riječ nije papavska i prekidamo daljnju provjeru. Ako jesu, preskočimo ta sljedeća dva slova u riječi i dalje vršimo istu provjeru. Ako dođemo do kraja riječi, ona je papavska.

Ako je riječ papavska, potrebno je ispisati polaznu riječ koju dobivamo tako da iz riječi izbacimo ubačene dodatke. Prolazimo redom po slovima te riječi i ispisujemo ono slovo na kojem se nalazimo. Ako je to slovo samoglasnik, onda nakon ispisivanja preskačemo sljedeća dva slova (malo slovo p i ponovno taj samoglasnik) te na taj način izbacujemo ubačene dodatke - budući da ih preskočimo, oni se ne ispišu.

Ako riječ nije papavska, potrebno je ispisati odgovarajuću papavsku riječ koju dobivamo tako da u riječ, prema opisanom postupku, ubacimo dodatke. Prolazimo redom po slovima te riječi i ispisujemo ono slovo na kojem se nalazimo. Ako je to slovo samoglasnik, onda nakon ispisivanja dodatno ispisujemo i sljedeća dva slova: malo slovo p i ponovno taj samoglasnik. Na taj način ubacujemo potrebne dodatke i na kraju dobivamo odgovarajuću papavsku riječ.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <cstring>
using namespace std;
```

```
int n;
char rijec[11];
bool isVowel(char letter) {
  return letter == 'a' || letter == 'e' || letter == 'i' || letter == 'o' ||
letter == 'u';
}
int main(void) {
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) {
    scanf("%s", rijec);
    int len = strlen(rijec);
    bool papavski = true;
    for (int j = 0; j < len; ++j) {
      if (isVowel(rijec[j])) {
        if (j >= len-2 || rijec[j+1] != 'p' || rijec[j+2] != rijec[j]) {
          papavski = false;
          break;
        }
        j += 2;
      }
    }
    if (papavski) {
      for (int j = 0; j < len; ++j) {
        printf("%c", rijec[j]);
        if (isVowel(rijec[j])) {
          j += 2;
        }
      }
    } else {
      for (int j = 0; j < len; ++j) {
        printf("%c", rijec[j]);
        if (isVowel(rijec[j])) {
          printf("p%c", rijec[j]);
        }
      }
    }
    printf("\n");
```

```
    }
    return 0;
}
```

Potrebno znanje: stringovi

Kategorija: ad-hoc

7.3. Zadatak: Memorija — Ideja: Adrian Satja Kurdija

Za prvi dio zadatka (prva dva retka izlaznih podataka) treba imati dvije liste stringova: u jednu ćemo zapisivati Ivine, a u drugu Sandrine osvojene slike. Na početku su ove liste prazne. Učitavajući poteze, u pomoćnoj varijabli pamtimo tko je na potezu. Ako su u trenutnome potezu otkrivene dvije jednake slike, dotičnu sliku dodajemo u listu koja pripada osobi koja je na potezu, a inače mijenjamo varijablu koja označava osobu na potezu. Na koncu ispisujemo popunjene liste.

Za drugi dio zadatka (treći redak izlaznih podataka) potrebna je matrica koja pamti koliko je puta koje polje otkriveno. Na početku je ta matrica, naravno, ispunjena nulama jer nijedno polje još nije otkriveno. U svakom potezu potrebno je u toj matrici odgovarajuća dva polja povećati za jedan. Međutim, ako su u potezu odgovarajuće karte uklonjene, potrebno je to zapisati u matricu, npr. tako da vrijednosti odgovarajućih polja postavimo na -1. Na koncu prođemo matricom brojeći polja čija je vrijednost barem 2.

Za posljednji dio zadatka treba nam matrica stringova u čija polja zapisujemo slike koje u potezima otkrivamo. Nakon obrađenih poteza valja vidjeti koje se slike u toj matrici pojavljuju dvaput, a nisu uklonjene: to su traženi parovi koje je moguće ukloniti. Prolazeći po matrici, u pomoćnu listu spremamo otkrivene slike, pa kada naiđemo na sliku koja se već nalazi u pomoćnoj listi, to znači da se ona pojavljuje dvaput.

Sve parove koje smo na ovaj način otkrili treba smatrati uklonjenima i vidjeti koliko je karata preostalo. Ako su preostale samo dvije karte, jasno je da čine par pa i njega pribrajamo u traženi rezultat.

Programski kod (pisan u Pythonu 3.4)

```
def ispis(niz):
    if len(niz) == 0:
        print(0)
        return
    for i in niz:
        print(i, end=" ")
    print()
r, s = map(int, input().split())
p = int(input())

slika = [['' for j in range(s)] for i in range(r)]
```

```
otvoreno = [[0 for j in range(s)] for i in range(r)]
osvojeno = [[], []]

na_potezu = 0
for potez in range(p):
    r1, s1, slika1, r2, s2, slika2 = input().split()
    r1 = int(r1) - 1
    s1 = int(s1) - 1
    r2 = int(r2) - 1
    s2 = int(s2) - 1
    slika[r1][s1] = slika1
    slika[r2][s2] = slika2
    if slika1 == slika2:
        otvoreno[r1][s1] = otvoreno[r2][s2] = -1
        osvojeno[na_potezu].append(slika1)
    else:
        otvoreno[r1][s1] += 1
        otvoreno[r2][s2] += 1
        na_potezu = 1 - na_potezu
ispis(osvojeno[0])
ispis(osvojeno[1])

poznato = []
moguce_skupiti = []
bar_dvaput = 0

for i in range(r):
    for j in range(s):
        bar_dvaput += (otvoreno[i][j] > 1)
        if otvoreno[i][j] > 0:
            if slika[i][j] in poznato:
                moguce_skupiti.append(slika[i][j])
            else:
                poznato.append(slika[i][j])
print(bar_dvaput)
```

```
preostalo = 0
for i in range(r):
    for j in range(s):
        if otvoreno[i][j] != -1 and \
                slika[i][j] not in moguce_skupiti:
            preostalo += 1
print(len(moguce_skupiti) + (preostalo == 2))
```

Potrebno znanje: matrice

Kategorija: ad-hoc

8.1. Zadatak: Krug — Ideja: Dominik Gleich[2]

Prvi problem koji nam se pojavljuje u zadatku je cirkularnost prstena. Taj problem je najlakše riješiti tako da niz zalijepimo na samoga sebe. Time smo sveli problem na običan niz (koji sadrži 2N brojeva), ali s uvjetom da ne smijemo pročitati više od N brojeva, zbog uvjeta da nijedan broj ne pročitamo dvaput.

Pokušajmo sada riješiti zadatak na običnom nizu. Neka nam varijabla maks označava najveće dosad pronađeno rješenje.

S obzirom da moramo provjeriti sve podnizove duljine manje ili jednake N, moramo pokušati fiksirati dvije pozicije, i te j, tako da i označava lijevi kraj intervala koji razmatramo, a j desni kraj tog intervala, te da je razlika između krajeva manja ili jednaka N (kako ne bismo dvaput isti broj koristili). Za svaki odabrani podniz potrebno je izračunati sumu njegovih brojeva i usporediti to s varijablom maks, u kojoj nam je spremljeno trenutno najbolje rješenje.

Na kraju jednostavno ispišemo sadržaj varijable maks, jer je upravo to traženo rješenje.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <algorithm>
#include <cstring>
using namespace std;
int n;
int a[20];
int main (void){
  scanf("%d", &n);
  for (int i = 0; i < n; ++i) scanf("%d", &a[i]);
```

[2] student Fakulteta elektrotehnike i računarstva, osvajač tri srebrne i dvije brončane medalje s međunarodnih informatičkih olimpijada te četverostruki državni prvak iz informatike

```
  int ans = 0;
  for (int i = 0; i < n; ++i){
    int sum = 0;
    for (int j = 0; j < n; ++j){
      sum += a[(i + j)%n];
      ans = max(ans, sum);
    }
  }
  printf("%d\n", ans);
  return 0;
}
```

Potrebno znanje: nizovi, algoritam za traženje maksimuma

Kategorija: ad-hoc

8.2. Zadatak: Križić — Ideja: Antun Razum

Prvo što možemo primijetiti u ovom zadatku je da nam uzastopni nizovi istih oznaka duljine veće od tri ne predstavljaju poseban problem. Ako samo tražimo nizove od tri uzastopne oznake i svaki takav računamo kao jedan bod, ispravno ćemo izračunati bodove.

Opišimo sada postupak pronalaska nizova od tri uzastopne oznake koji počinju na nekom polju. Budući da svaki od osam osnovnih smjerova ima svoj suprotni smjer unutar tih istih osam, provjeravat ćemo samo četiri smjera kako ne bi iste nizove brojali dva puta. Pokažimo to na primjeru. Kada bi provjeravali smjer lijevo i njemu suprotan smjer, desno, niz od tri uzastopne oznake u retku brojali bi na njegovom lijevom polju kada bi provjeravali smjer desno i na njegovom desnom polju kada bi provjeravali smjer lijevo. U rješenju ćemo stoga provjeravati smjerove: desno, dolje-desno, dolje i dolje-lijevo. Primijetite da niti jedan od ovih smjerova nije suprotan nekom drugom.

Pokažimo sada kako izračunati broj bodova za svakog igrača. U rješenju će nam za to trebati niz u kojem ćemo pamtiti trenutni broj bodova svakog igrača. Prođemo kroz sva polja tablice i za svako polje radimo sljedeći postupak. Prođemo kroz svaki od četiri spomenuta smjera. Ako su sva polja u trenutnom smjeru unutar tablice i imaju istu oznaku kao i trenutno polje, igraču s tom oznakom dodajemo jedan bod.

Kada smo izračunali broj bodova svih igrača, pronađemo maksimalni broj bodova. Nakon toga prođemo po bodovima igrača i ispišemo oznake svih igrača koji imaju maksimalni broj bodova. Naposljetku ispišemo maksimalni broj bodova.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <vector>
using namespace std;const int MAXN = 10, MAXK = 20;
```

```
const int RP[] = {0, 1, 1, 1}, SP[] = {1, 1, 0, -1};
int n, m, k, bodovi[MAXK];
char tablica[MAXN][MAXN];

int main() {
	scanf("%d%d%d", &n, &m, &k);
	for (int i = 0; i < n; i++)
		scanf("%s", tablica[i]);
	for (int i = 0; i < n; i++)
		for (int j = 0; j < m; j++) {
			char oznaka = tablica[i][j];
			for (int smjer = 0; smjer < 4; smjer++) {
				bool bod = true;
				for (int odmak = 1; odmak < 3; odmak++) {
					int redak = i + RP[smjer] * odmak, stupac = j + SP[smjer] * odmak;
					if (redak < 0 || redak >= n || stupac < 0 || stupac >= m || tablica[redak][stupac] != oznaka) {
						bod = false;
						break;
					}
				}
				bodovi[oznaka - 'A'] += bod;
			}
		}
	vector <int> pobjednici;
	int max = 0;

	for (int i = 0; i < k; i++) {
		int trenutni = bodovi[i];
		if (trenutni > max) {
			max = trenutni;
			pobjednici.clear();
		}
		if (trenutni == max)
			pobjednici.push_back(i);
	}
```

```
        for (int i = 0; i < pobjednici.size(); i++) {
              printf("%c", pobjednici[i] + 'A');

              if (i < pobjednici.size() - 1)
                    putchar(' ');
        }
        putchar('\n');
        printf("%d\n", max);
        return 0;
}
```

Potrebno znanje: matrice

Kategorija: ad-hoc

8.3. Zadatak: Promet — Ideja: Nikola Dmitrović

Promet je kompleksan zadatak koji simulira jednu stvarnu situaciju iz realnog života. Prema bodovanju iz zadatka, 36 bodova moglo se dobiti ako se simulirao samo prvi prolazak kroz raskrižje. To se moglo postići višestrukim naredbama odlučivanja i provjeravanjem slučajeva koji se mogu dogoditi. Nema ih malo, ali ima konačan broj.

Jedno od općih rješenja za sve bodove svodilo se na simuliranje svih prolazaka kroz raskrižje upravo onako kako to piše u zadatku. Pokažimo na jednoj konkretnoj implementaciji kako se to moglo izvesti. Još jednu implementaciju možete pronaći u priloženim kodovima.

Programski kod (pisan u Pythonu 3.4)

```
# promotrimo kada vozilu iz i-te ulice smeta vozilo iz j-te ulice
def presijecaju(i, si, j, sj):
    # na istoj su cesti
    if i % 2 == j % 2:
        if si != 'L' and sj != 'L': return False
        if si == sj: return False
        return True
    # na različitim su cestama
    if si == 'R' and sj == 'R': return True
    if si == 'L' and sj == 'L': return True
    if si == 'D' and sj == 'D': return False
    if si == 'R':
        if sj == 'D': return ((j + 1 - i + 4) % 4 == 0)
        if sj == 'L': return True
    if sj == 'R':
        if si == 'D': return ((i + 1 - j + 4) % 4 == 0)
```

```
        if si == 'L': return True
    return False

# promotrimo ima li vozilo iz i-te ulice prednost
def prednost(i, si, j, sj):
    # ima li si prednost?
    if i % 2 == 0 and j % 2 == 1: return True
    if i % 2 == 1 and j % 2 == 0: return False
    if si != 'L' and sj == 'L': return True
    return False

# ----------------------------------------

# učitamo niz stringova
promet = [input(),input(),input(),input()]

# zamijenimo '*' s '' zbog lakše implementacije i prebrojimo vozila
vozila = 0
for i in range(4):
    if promet[i] == '*': promet[i] = ''
    vozila += len(promet[i])

sekundi = 0
while vozila > 0: # dok ima vozila na raskrižju
    sekundi += 1  # promatramo jedan prolaz kroz raskrižje koji traje jednu sekundu
    prolaze = []  # pamtimo koja sve vozila u jednom prolazu mogu proći
    for i in range(4): # za svaku ulicu pogledajmo vozilo pred raskrižjem
        if promet[i] != '': # naravno, ako u toj ulici ima vozila
            smeta = 0 # brojimo koliko vozila smeta vozilu iz i-te ulice
            for j in range(4): # za svaku ulicu pogledajmo vozilo pred rakrižjem
                if i != j and promet[j] != '': # naravno, ako to nije ista ulica i ako u njoj ima
vozila
                    # u zadatku piše upravo ovo što ćemo sada napisati
                    smeta += presijecaju(i,promet[i][0],j,promet[j][0])and prednost(j, promet[j][0],
i, promet[i][0])
            if smeta == 0: # ako vozilu iz i-te ulice nitko ne smeta
                prolaze = prolaze + [i] # dadaj to vozilo na popis onih koji u toj sekundi mogu
istovremeno proći

    for i in prolaze: # obriši smjerove za ona vozila koja su prošla
        promet[i] = promet[i][1:]

    vozila -= len(prolaze) # smanji ukupan broj vozila
```

```
    if sekundi == 1: # ako je riječ o prvom prolazu ispiši vozila u sortiranom poretku
        prolaze.sort()
        for i in prolaze:
            print(chr(i + 65), end = "")
        print()

print(sekundi) # ispiši ukupan broj sekundi
```

Potrebno znanje: nizovi, naredba ponavljanja, rad sa znakovima, sortiranje

Kategorija: simulacija