18. ožujka 2015.

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

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

OPISI ALGORITAMA

Agencija za odgoj i obrazovanje
Education and Teacher Training Agency

HRVATSKI SAVEZ INFORMATIČARA

Ministarstvo znanosti, obrazovanja i sporta

5.1. Zadatak: Horoskop

Ideja: Nikola Dmitrović

Za zadani datum jednostavnim provjerama tražimo kojem horoskopskom znaku on pripada.

Programski kod (pisan u Pythonu 3)

```
D = int(input())
M = int(input())
if D >= 21 and M == 3: print("OVAN")
if D <= 20 and M == 4: print("OVAN")
if D >= 21 and M == 4: print("BIK")
if D <= 20 and M == 5: print("BIK")
if D >= 21 and M == 5: print("BLIZANCI")
if D <= 20 and M == 6: print("BLIZANCI")
if D >= 21 and M == 6: print("RAK")
if D <= 20 and M == 7: print("RAK")
if D >= 21 and M == 7: print("LAV")
if D <= 21 and M == 8: print("LAV")
if D >= 22 and M == 8: print("DJEVICA")
if D <= 22 and M == 9: print("DJEVICA")
if D >= 23 and M == 9: print("VAGA")
if D <= 22 and M == 10: print("VAGA")
if D >= 23 and M == 10: print("SKORPION")
if D <= 22 and M == 11: print("SKORPION")
if D >= 23 and M == 11: print("STRIJELAC")
if D <= 21 and M == 12: print("STRIJELAC")
if D >= 22 and M == 12: print("JARAC")
if D <= 20 and M == 1: print("JARAC")
if D >= 21 and M == 1: print("VODENJAK")
if D <= 19 and M == 2: print("VODENJAK")
if D >= 20 and M == 2: print("RIBE")
if D <= 20 and M == 3: print("RIBE")
```

Potrebno znanje: naredba odlučivanja

Kategorija: ad-hoc

5.2. Zadatak: Chelsea — Ideja: Nikola Dmitrović

Kako znamo rezultat cijele utakmice i rezultat prvog poluvremena, lako možemo odrediti rezultat drugog poluvremena. Njegovo određivanje svodi se na jednostavno oduzimanje. U tri pomoćne varijable pratimo stanje bodova za svaki od navedenih perioda igre.

Programski kod (pisan u Pythonu 3)

```
cijela = prvo = drugo = 0
koliko = int(input())
for i in range(koliko):
    Du, Gu, D1, G1 = map(int, input().split())

    D2 = Du - D1
    G2 = Gu - G1

    if Du > Gu: cijela += 3
    if Du == Gu: cijela += 1

    if D1 > G1: prvo += 3
    if D1 == G1: prvo += 1

    if D2 > G2: drugo += 3
    if D2 == G2: drugo += 1
print(cijela, prvo, drugo, sep = '\n')
```

Potrebno znanje: naredba ponavljanja, naredba odlučivanja

Kategorija: ad-hoc

5.3. Zadatak: Kamera — Ideja: Matija Milišić

Prvo ćemo opisati rješenje zadatka u kojem se ne pojavljuju vozila hitne pomoći. Simuliramo pomicanje vozila u redu pomoću dvije ugniježđene for petlje.

Prva petlja prolazi po svim vozilima u redu – početno_vozilo označava vozilo koje je u tom trenutku prvo u redu, dok su vozila prije njega već prošla naplatu.

Druga petlja služi za pronalaženje svih vozila koja su barem jednim dijelom u vidokrugu kamere. Ona prolazi po vozilima koja su još uvijek u redu (početno_vozilo i sva iza njega) i računa njihovu udaljenost od naplatne kućice. Udaljenost trenutnog_vozila od početnog_vozila spremljena je u varijabli udaljenost.

Udaljenost lijevog kraja vozila (lijeva_udaljenost) jednaka je udaljenost, a udaljenost desnog kraja (desna_udaljenost) jednaka je zbroju lijeve_udaljenosti i duljine trenutnog_vozila. Vozilo je barem

jednim dijelom unutar vidokruga kamere ako je bilo lijeva_udaljenost bilo desna_udaljenost unutar vidokruga – između X i Y.

Za svako trenutno_vozilo na taj način odredimo je li unutar vidokruga. Prebrojimo ona koja jesu i to ispišemo kao rješenje za početno_vozilo.

Sada ćemo opisati rješenje cijelog zadatka, uzimajući u obzir i vozila hitne pomoći.

Promatrajmo hitne koje će nadzornik uočiti na kameri – to su hitne koje već jesu ili će biti u vidokrugu. Te hitne ni na koji način ne utječu na broj vozila u vidokrugu – one koje još nisu u vidokrugu ne utječu jer su iza tih vozila, a one koje su u njemu isto tako jer odmah izlaze. Stoga, ne moramo čekati da hitna dođe u vidokrug da bi je izbacili, nego to možemo napraviti odmah na početku. Na taj si način uvelike olakšavamo zadatak. Takve hitne prepoznajemo po tome što imaju desnu_udaljenost veću od X.

Nakon što smo izbacili hitne, zadatak rješavamo na sličan način kao i bez hitni. Preostale hitne nadzornik će uočiti na naplatnoj kućici. Stoga, kada početno_vozilo bude hitna, ona odmah izlazi s autoceste i samo ju je potrebno preskočiti.

Programski kod (pisan u C-u)

```
#include <cstdio>
using namespace std;
int x, y, ucitani_n, n;
int udaljenost, brojac;
int ucitana_duljina_vozila[100], duljina_vozila[100];

int main(void) {
	scanf("%d%d", &x, &y);
	scanf("%d", &ucitani_n);
	for (int i = 0; i < ucitani_n; ++i) {
		scanf("%d", &ucitana_duljina_vozila[i]);
	}

	// izbacivanje hitni
	udaljenost = 0;
	for (int vozilo = 0; vozilo < ucitani_n; ++vozilo) {
		udaljenost += ucitana_duljina_vozila[vozilo];

		if (ucitana_duljina_vozila[vozilo] == 5 && udaljenost > x) {
			// hitna koja ili vec je ili ce biti u vidokrugu kamere
		} else {
			duljina_vozila[n] = ucitana_duljina_vozila[vozilo];
			++n;
```

```
            }
      }
      for (int pocetno_vozilo = 0; pocetno_vozilo < n; ++pocetno_vozilo) {
            if (duljina_vozila[pocetno_vozilo] == 5) {
                  // hitna uocena na naplatnoj kucici
            } else {
                  udaljenost = 0;
                  brojac = 0;
                  for (int trenutno_vozilo = pocetno_vozilo; trenutno_vozilo <
n; ++trenutno_vozilo) {
                        if (udaljenost+duljina_vozila[trenutno_vozilo] > x &&
udaljenost < y) {
                              // vozilo u vidokrugu kamere
                              brojac = brojac+1;
                        }
                        udaljenost += duljina_vozila[trenutno_vozilo];
                  }
                  printf("%d\n", brojac);
            }
      }
      return 0;
}
```

Potrebno znanje: dvostruka naredba ponavljanja, niz

Kategorija: ad-hoc

6.1. Zadatak: Kineski — Ideja: Nikola Dmitrović

U osnovi, oba dijela zadatka rješavaju se provjeravanjem slučajeva. Drugi dio zadatka svodi se na vrlo jednostavne provjere svih slučajeva. Prvi dio je malo kompliciraniji. Znamo da je 1960. bila godina Štakora. Tada moramo promatrati dva slučaja, godine koje su bile prije 1960 i godine koje su bile nakon nje. Jednostavnom provjerom vidimo da vrijedi opisani kod. Zadatak je moguće riješiti bez početne naredbe odlučivanja ako primijetimo da je 1900. isto bila godina Štakora.

Programski kod (pisan u Pythonu 3.4)

```
G = int(input())
S = int(input())


if G < 1960:
```

```
    odabir = (12 - ((1960 - G) % 12)) % 12
else:
    odabir = (G - 1960) % 12

if odabir == 0: print("STAKOR")
if odabir == 1: print("BIK")
if odabir == 2: print("TIGAR")
if odabir == 3: print("ZEC")
if odabir == 4: print("ZMAJ")
if odabir == 5: print("ZMIJA")
if odabir == 6: print("KONJ")
if odabir == 7: print("KOZA")
if odabir == 8: print("MAJMUN")
if odabir == 9: print("PIJETAO")
if odabir == 10: print("PAS")
if odabir == 11: print("SVINJA")

if S == 23 or S == 0: print("STAKOR")
if S == 1 or S == 2: print("BIK")
if S == 3 or S == 4: print("TIGAR")
if S == 5 or S == 6: print("ZEC")
if S == 7 or S == 8: print("ZMAJ")
if S == 9 or S == 10: print("ZMIJA")
if S == 11 or S == 12: print("KONJ")
if S == 13 or S == 14: print("KOZA")
if S == 15 or S == 16: print("MAJMUN")
if S == 17 or S == 18: print("PIJETAO")
if S == 19 or S == 20: print("PAS")
if S == 21 or S == 22: print("SVINJA")
```

Potrebno znanje: naredba odlučivanja

Kategorija: ad-hoc

6.2. Zadatak: Zmija — Ideja: Mislav Bradač

Opišimo algoritam za slučaj kada se zmija kreće udesno. U svakom trenutku moramo pamtiti koji je najdesniji vidljivi kvadratić koji pripada zmiji, kolika je vidljiva duljina zmije, je li zmija živa te ima li zmija super moć. Je li zmija živa možemo pamtiti u istoj varijabli kao i duljinu zmije, tj. ako zmija više nije živa samo postavimo duljinu zmije na 0. Spomenute informacije pamtimo u varijablama pos, length i power. Length je u početku 1, power je false, a pos određujemo jednim prolaskom kroz polje i pronalaskom pozicije zmije u početnom trenutku.

Sada kada smo odredili što moramo pamtiti možemo krenuti sa simulacijom kretnje zmije. U svakom koraku ćemo promotriti kako se mijenjaju parametri koje moramo pratiti. Bitno je napomenuti da simulaciju prekidamo kada duljina zmije padne na nulu ili kada simuliramo svih T sekundi. U svakom koraku vrijedi da se pos poveća za 1, osim u slučaju da je zmija na rubu retka - tada pos ne mijenjamo. Primijetimo da određena pravila Mirkove igre možemo definirati i ovako:

Ako je redak[pos] jednak '.', onda se length ne mijenja.

Ako je redak[pos] jednak '#', postavljamo length na 0.

Ako je redak[pos] jednak '+', povećavamo length za 1.

Ako je redak[pos] jednak '!', postavljamo power na true.

Ako je redak[pos] jednak 'x', a power jednak false, onda povećavamo length.

Ako je redak[pos] jednak 'x', a power jednak true, onda smanjujemo length.

Sve kvadratiće kroz koje je zmija prošla i ostala živa tijekom simulacije možemo postaviti na '.' (može se dogoditi da se u na kraju u nekom od tih kvadratića nalazi zmija, ali o tome brinemo na kraju simulacije).

Kad je simulacija gotova moramo još označiti polja u kojima se nalazi zmija. To radimo prolaskom petlje od pos - length + 1 do pos i stavljamo te kvadratiće na vrijednost 'o'.

Da ne bismo morali dvaput implementirati ovaj algoritam, kretanje ulijevo ćemo riješiti trikom: možemo obrnuti cijeli niz, pozvati funkciju koja riješi zadatak ako se zmije kreće udesno te ispisati unatrag dobiveni izgled retka.

Programski kod (pisan u Pythonu 3)

```
def solve(to_left, n, a, t) :
   if to_left :
      a = list(reversed(a))
   b = a[:]

   pos = a.index('o')
   b[pos] = '.'
   length = 1
   super_power = False
```

```
    for i in range(t) :
        pos += 1
        if pos >= n :
            pos = n - 1
            length -= 1
            if len == 0 :
                break
            continue
        if a[pos] == '#' :
            length = 0
        elif a[pos] == '+' :
            b[pos] = '.'
            length += 1
        elif a[pos] == 'x' :
            b[pos] = '.'
            if super_power :
                length += 1
            else :
                length -= 1
        elif a[pos] == '!' :
            b[pos] = '.'
            super_power = True
        if length == 0 :
            break

    for i in range(length) :
        b[pos-i] = 'o'

    if to_left :
        b = list(reversed(b))
    print(''.join(b))

n = int(input())
a = list(input())
t = int(input())
```

```
solve(False, n, a, t)
solve(True, n, a, t)
```

Potrebno znanje: string

Kategorija: ad-hoc

6.3. Zadatak: Vlakovi — Ideja: Antun Razum

Kako bismo riješili ovaj zadatak, zamislimo kolosijeke kao brojevne pravce. Brojke na brojevnom pravcu biti će sekunde u danu. Svaki vlak na nekom kolosijeku predstavljat će jedna dužina na pripadnom brojevnom pravcu koja će se protezati od sekunde kada vlak počinje prolazak do sekunde kada ga završava. Budući da svakom vagonu treba točno jedna sekunda da prođe, vlak završava svoj prolazak onoliko sekundi poslije početka prolaska koliko vagona ima. Kada smo ovo ustanovili, na svakom brojevnom pravcu možemo vidjeti u kojim sekundama odgovarajući kolosijek tokom dana nije slobodan. Ako neku brojku prekriva dužina, to znači da kolosijek tokom te sekunde nije slobodan.

Iduće što valja primijetiti je da sve dužine koje se nalaze na ovim brojevnim pravcima možemo smjestiti na jedan brojevni pravac tako da svaka od njih ima isti položaj koja je imala i prije. Iz ovog brojevnog pravca sada možemo iščitati kada prijelaz nije slobodan uzimajući sve kolosijeke istovremeno u obzir, a to je upravo ono što se traži u zadatku. Iz toga slijedi da nam informacija na kojem se kolosijeku neki vlak nalazi zapravo uopće nije potrebna jer sve vlakove možemo promatrati kao da prolaze jednim jedinim kolosijekom. Preslikajmo sada ovu matematičku ideju u programsko rješenje.

Prvi problem s kojim se susrećemo prilikom rješavanja ovog zadatka je pretvorba vremena zadanih u obliku HH:MM:SS u broj sekundi od početka dana. To rješavamo na vrlo jednostavan način. Podijelimo niz znakova zadan u ulazu na tri dijela: HH, MM i SS. Pomoću njih možemo lako odrediti broj sati, minuta i sekundi. Zatim kako bismo dobili ukupni broj sekundi broj sati pomnožimo s brojem sekundi u satu, a broj minuta s brojem sekundi u minuti te to sve zajedno zbrojimo s brojem sekundi.

Sada pripremimo pomoćni niz koji će nam predstavljati brojevni pravac na kojemu se nalaze svi vlakovi. Svako polje tog niza predstavljat će odgovarajuću sekundu u danu i moći će biti slobodno ili ne. Za početak postavimo sva polja da budu slobodna. Sada za svaki vlak prođemo po svim sekundama u danu kojima on prolazi te odgovarajuća polja niza postavimo tako da ne budu slobodna. Prisjetimo se da neki vlak može početi prolaziti krajem dana te dovršiti svoj prolazak početkom idućeg dana. Kako bismo se pobrinuli za ovaj slučaj moramo paziti da kada dođemo do posljednje sekunde u danu nastavimo s prvom. Ovo jednostavno možemo riješiti tako da koristimo ostatak pri dijeljenju s brojem sekundi u danu. Primijetite također da slučaj u kojem više vlakova prolazi u istoj sekundi ovdje ne predstavlja nikakav problem.

Kada smo prošli sve vlakove jednostavno prođemo kroz sve sekunde u danu i prebrojimo polja pomoćnog niza koja nisu slobodna. Broj takvih polja je rješenje zadatka.

Programski kod (pisan u C-u)

```
#include <cstdio>
const int MAXM = 110, MAXS = 100, MAXD = 24 * 3600;
int n, m, vremena[MAXM], brojevi[MAXM], prolazaka[MAXD];
```

```
int main() {
    scanf("%d%d", &n, &m);
    for (int i = 0; i < n; i++) {
        int k;
        scanf("%d", &k);
        for (int j = 0; j < k; j++) scanf("%*d");
    }
    for (int i = 0; i < m; i++) {
        char vrijeme[MAXS];
        scanf("%d%s", &brojevi[i], vrijeme);
        int hh = (10 * (vrijeme[0] - '0') + vrijeme[1] - '0');
        int mm = (10 * (vrijeme[3] - '0') + vrijeme[4] - '0');
        int ss = (10 * (vrijeme[6] - '0') + vrijeme[7] - '0');
        vremena[i] = 3600 * hh + 60 * mm + ss;
    }
    for (int i = 0; i < m; i++)
        for (int j = 0; j < brojevi[i]; j++)
            prolazaka[(vremena[i] + j) % MAXD]++;
    int sekundi = 0;
    for (int i = 0; i < MAXD; i++)
        if (prolazaka[i])
            sekundi++;
    printf("%d\n", sekundi);
    return 0;
}
```

Potrebno znanje: nizovi, naredba ponavljanja

Kategorija: ad-hoc simulacija

7.1. Zadatak: Superniz — Ideja: Mislav Balunović

Održavat ćemo pomoćni niz koji možemo nazvati brojač i u kojemu ćemo pamtiti koliko puta se koji broj pojavljuje u tablici. Na početku postavimo sve vrijednosti u nizu brojač na 0. Sada učitavamo n brojeva. Kada učitamo broj x, povećamo vrijednost brojača na poziciji x za 1. Na kraju, imamo još jednu for petlju koja ide od 1 do K i imamo if koji za svaki broj x provjerava pojavljuje li se taj broj točno x puta.

Programski kod (pisan u Pythonu 3)

```
N = int(input())
a = list(map(int, input().split()))
K = int(input())

cnt = [0 for x in range(0, 1000010)]
for x in a:
    cnt[x] += 1

flag = False
for x in range(1, K + 1):
    if (cnt[x] != x):
        print(x)
        flag = True
        break
if (not flag):
    print("SUPER")
```

Potrebno znanje: nizovi

Kategorija: ad-hoc

7.2. Zadatak: Sandra — Ideja: Nikola Dmitrović

Sandra je zadatak za čije rješenje treba biti vješt u kodiranju. Ideja rješenja može se iščitati iz samog teksta zadatka, ali je dobro posložen kod ključna stvar u rješenju.

Ulazni podaci su daljine bacanja natjecatelja po serijama. Uočimo da je nakon učitavanja bolje zamijeniti retke i stupce i dalje promatrati daljine bacanja po natjecateljima. Ovo lukavstvo nije nužno za rješenje, ali će uvelike olakšati kodiranje rješenja.

Zadatak se sastoji od dva dijela. Proučimo i analizirajmo jedno od mogućih rješenja. Neki dijelovi koda mogući su zbog samog programskog jezika Python u kojem je kod pisan. Za ostale jezike, ideja ostaje ista, samo implementacija treba biti prilagođena tom jeziku.

Programski kod (pisan u Python 3)

```
#funkcija koja zamjenjuje retke i stupce
def obrni(orginal, koliko):
    kopija = [[0 for j in range(3)] for i in range(koliko)]
    for i in range(len(orginal)):
        for j in range(len(orginal[i])):
            kopija[j][i] = orginal[i][j]
    return(kopija)

N = int(input()) # broj natjecatelja
# učitavanje podataka, i-ti redak su daljine bacanja natjecatelja u i-toj seriji
serija = [[0] for i in range(3)]
for i in range(3):
    serija[i] = list(map(int,input().split()))

# zamjenjujemo retke i stupce, sada je u listi bacaci i-ti redak daljine bacanja
# i-tog natjecatelja po serijama
bacaci = obrni(serija, N)

# u listu best_skok upisujemo parove (max_daljina_i_tog_bacača, i)
prosli_dalje, best_skok = [], []
for i in range(N):
    best_skok += [(max(bacaci[i]), i)]

# sortiramo listu best_skok
best_skok.sort(reverse = True)

# u listu prosli_dalje upisujemo oznake najboljih N//2 natjecatelja
for i in range(N // 2):
    prosli_dalje += [best_skok[i][1]]

# sortirajmo oznake natjecatelja koji su prošli dalje i ispisujemo
prosli_dalje.sort()
for i in prosli_dalje:
    print(i + 1, end = ' ')
print()
# ovdje završava prvi dio zadatka
# učitavamo daljine bacanja u naredne tri serije
```

```
nastavak = [[0] for i in range(3)]
for i in range(3):
    nastavak[i] = list(map(int,input().split()))

# lista finale je lista nastavak sa zamijenjenim retcima i stupcima
finale = obrni(nastavak, N // 2)

# u listu finale objedinjuje daljine bacanja u prve i druge tri serije bacanja
for i in range(len(finale)):
    finale[i] += bacaci[prosli_dalje[i]]
    finale[i].sort(reverse = True) # sortiramo daljine bacanja i-tog bacača

# sortiramo natjecatelje (retke) prema daljinama bacanja, u prosli_dalje pamtimo
# koji redak odgovara kojem bacaču
for i in range(N // 2):
    for j in range(i, N // 2):
        k = 0
        while k < 6 and finale[i][k] == finale[j][k]:
            k += 1
        if k == 6 or finale[i][k] > finale[j][k]:
            continue
        finale[i], finale[j] = finale[j], finale[i]
        prosli_dalje[i], prosli_dalje[j] = prosli_dalje[j], prosli_dalje[i]

# određivanje osvajača medalja
k = 0
for medalja in range(3):
    medaljasi = [prosli_dalje[k] + 1]
    while k + 1 < N // 2 and finale[k] == finale[k + 1]:
        k += 1
        medaljasi.append(prosli_dalje[k] + 1)
    medaljasi.sort()
    print(' '.join(map(str, medaljasi)))
    k += 1
```

Potrebno znanje: dvodimenzionalni nizovi, sortiranje

Kategorija: ad-hoc

7.3. Zadatak: Koncert — Ideja: Dominik Gleich

Pokušajmo prvo riješiti zadatak za 25 bodova, tj. za slučaj kada je N <= 2.

U takvom slučaju broj se sastoji od 4 znamenke. Naivnim isprobavanjem svih 10^(2N) mogućnosti možemo provjeriti za koju mogućnost vrijede gore navedeni uvjeti, te u slučaju da vrijede pribrajamo 1 na rješenje za svaku mogućnost. Mogućnosti možemo generirati for petljom od 0 do 10^(2N)-1, uključivo, zapišemo li svaki broj od 0 do 10^(2N)-1 kao string dulijne 2N, dodajući nule na početak u slučaju da broj ima manje od 2N znamenaka. Zatim uspoređujemo taj string znak po znak sa stringom iz ulaza i u slučaju da odgovara vršimo provjeru zbroja znamenaka na obje polovice, te provjeru broja parnih znamenaka na obje polovice za drugi broj u ispisu. Takav pristup donosio je 25 bodova. Složenost ovog pristupa je O(10^(2N) * N)

U skupu test primjera vrijednima 60 bodova vrijedi da je broj nepoznatih znamenaka <= 5.

Kako nam to pomaže? Iz definicije valjanog broja ulaznice u zadatku slijedi da sve ulaznice imaju iste sve znamenke koje nisu ‘?’ u originalnoj ulaznici. Što znači da su samo znakovi ‘?’ promjenjive znamenke. S obzirom da ih ima maksimalno 5, radimo sličan algoritam kao u prvom slučaju, samo ovaj put pokušavamo postaviti sve moguće kombinacije nad upitnicima direktno. Neka je broj upitnika L, onda prolaskom po svim brojevima od 0 do 10^L - 1, uključivo, prolazimo kroz sve kombinacije za te upitnike, i pridjeljujemo znamenke tim upitnicima slično kao i u prvom slučaju. Nakon pridjeljivanja broja svakom upitniku vršimo provjeru sume i broja parnih znamenaka na obje polovice, te odlučujemo dodajemo li 1 na rješenje. O(10^L* N)

Za potpuno rješenje ovog zadatka potrebno je fiksirati sumu znamenaka i broj parnih znamenaka u svakoj polovici. Prvo ćemo riješiti prvi ispis u zadatku, dakle broj različitih brojeva s jednakom sumom znamenaka na polovicama. Jasno je kako zbroj znamenaka jedne polovice ne može biti veći od 9*N, jer je svaka znamenka manja ili jednaka 9. 9*N, za N <= 5 ima maksimalnu vrijednost 45, što je dovoljno malo da pokušamo postaviti svaku sumu znamenaka. Dakle, za svaki S od 0 do 9*N, uključivo, pretpostavimo da je to suma znamenaka jedne polovice. Što nas zanima kada to pretpostavimo? Zanima nas koliko postoji brojeva u lijevoj polovici s tom sumom znamenaka, nazovimo taj broj A, i koliko postoji brojeva u desnoj polovici s tom sumom znamenaka, nazovimo taj broj B. U slučaju da imamo te dvije informacije znamo da postoji točno A*B različitih brojeva sa sumom znamenaka S. Za sada nam preostaje kako pronaći te brojeve A i B za svaku sumu. S obzirom da je potrebno generirati broj načina za postići neku sumu na obje polovice brojeva, a polovica broja je veličine N, gdje je N <= 5, dovoljno je proći po svim kombinacijama koristeći prvi algoritam, i za svaki broj zapisati u polje A[x], na koliko načina je moguće postići sumu x u prvoj polovici, i u polje B[x], na koliko načina je moguće postići sumu x u drugoj polovici.

Za drugi dio zadatka, za broj takvih brojeva s jednakim brojem parnih znamenaka, radimo gotovo jednaku stvar. Ponovno fiksiramo zbroj znamenaka, ali ovaj put fiksiramo i broj parnih znamenaka, neka je x zbroj znamenaka, a y broj parnih znamenaka, onda je rješenje suma svih parova

A[x][y] * B[x][y], gdje je A[x][y] broj kombinacija za konstruirati broj koji odgovara lijevoj polovici, a B[x][y] broj kombinacija za konstruirati broj koji odgovara drugoj polovici. Ta dva polja ponovno tražimo algoritmom koji je objašnjen u prvom dijelu opisa ovog rješenja, algoritmom koji prolazi po svim kombinacijama. Složenost ovog pristupa je O(10^N * N).

Za detalje i implementaciju pogledajte priloženi kod u kojem su implementirana sva tri dijela ovog algoritma, u pripadajućim funkcijama.

Programski kod (pisan u Pythonu 2)

```
n = int(raw_input())
broj = raw_input()
def numToStr(x, req):
 ret = str(x)
 while len(ret) < req:
   ret = '0' + ret
 return ret

def countEven(x):
 ret = 0
 for i in x:
   if int(i)%2 == 0: ret += 1
 return ret

def countSum(x):
 ret = 0
 for i in x:
   if i == '?': continue
   ret += int(i)
 return ret

def compare(x, y):
 for i in range(len(x)):
   if x[i] == '?': continue
   if x[i] != y[i]: return 0

 return 1

def solve30():
 ans0 = 0
 ans1 = 0

 for i in range(10**(2*n)):
```

```
    curr = numToStr(i, 2*n)
    if compare(broj, curr):
      if countSum(curr[:n]) == countSum(curr[n:]):
     ans0 += 1
     if countEven(curr[:n]) == countEven(curr[n:]):
      ans1 += 1

 return (ans0, ans1)

def solve70():
 ans0 = 0
 ans1 = 0
 cnt = 0

 a = []
 for i in range(len(broj)):
   if broj[i] == '?': a.append(i)

 for i in range(10**len(a)):
   p = list(broj)
   curr = numToStr(i, len(a))
   for j in range(len(curr)):
     p[a[j]] = curr[j]

   curr = ''.join(p)
   if countSum(curr[:n]) == countSum(curr[n:]):
     ans0 += 1
     if countEven(curr[:n]) == countEven(curr[n:]):
    ans1 += 1

 return (ans0, ans1)

def genCombinations(broj):
 n = len(broj)
 MAXSUM = 9 * 2 * 5 + 10
 MAXEVEN = 5 + 1
```

```
 sumCombinations = [0] * MAXSUM
 sumEvenCombinations = [[0] * MAXEVEN for x in range(MAXSUM)]

 for i in range(10**n):
   curr = numToStr(i, n)
   if compare(broj, curr):
     sumCombinations[countSum(curr)] += 1
     sumEvenCombinations[countSum(curr)][countEven(curr)] += 1

 return (sumCombinations, sumEvenCombinations)

def solve100():
 ans0 = 0
 ans1 = 0

 a = genCombinations(broj[:n])
 b = genCombinations(broj[n:])
 for i in range(len(a[0])):
   ans0 += a[0][i] * b[0][i]
   for j in range(len(a[1][0])):
     ans1 += a[1][i][j] * b[1][i][j]

 return (ans0, ans1)

#ret = solve30()
#ret = solve70()
ret = solve100()
print ret[0]
print ret[1]
```

Potrebno znanje: dvodimenzionalni nizovi

Kategorija: ad-hoc

8.1. Zadatak: San — Ideja: Adrian Satja Kurdija

Da bismo pojednostavnili računanje, potrebno je svako vrijeme iz ulaza pretvoriti u minute (npr. 05:30 → 5 * 60 + 30 = 330). Kako to učiniti? Najprije, vrijeme oblika AB:CD učitamo kao string, iz njega izdvojimo četiri znamenke: A, B, C, D i potom računamo:

broj_sati = A * 10 + B,

broj_minuta = C * 10 + D,

ukupan_broj_minuta = **broj_sati** * 60 + **broj_minuta**.

Posljednji je broj ono što nam zapravo treba i čime ćemo dalje računati.

Sada imamo vrijeme lijeganja **L** i vrijeme buđenja **B** izraženo u minutama. Koliko je ukupno vrijeme spavanja? Na prvi pogled ono je **B** - **L**. Ipak, nije uvijek tako: ako je vrijeme lijeganja prije ponoći, a vrijeme buđenja nakon ponoći, onda je formula drugačija: do ponoći smo spavali 24 * 60 - **L** minuta, a nakon ponoći **B** minuta i to treba zbrojiti.

Konačno, ukupno vrijeme spavanja treba podijeliti na cikluse. Broj čitavih (neprekinutih) ciklusa možemo dobiti npr. cjelobrojnim dijeljenjem ukupnog broja minuta s 90. Iz svakog ciklusa treba pribrojiti 20 minuta dubokom snu i 20 minuta REM snu. Iz preostalog, prekinutog ciklusa (čije je trajanje ostatak dijeljenja ukupnog broja minuta s 90) valja uz pomoć if-then-else naredbi zaključiti koliko je minuta osoba spavala dubokim, a koliko minuta REM snom i to pribrojiti rješenju.

Programski kod (pisan u Python 3)

```
l = input()
l = 60 * int(l[:2]) + int(l[3:])
b = input()
b = 60 * int(b[:2]) + int(b[3:])
if l >= 21 * 60 and b <= 9 * 60:
   ukupno = 24 * 60 - l + b
else:
   ukupno = b - l
duboki = ukupno // 90 * 20
rem = ukupno // 90 * 20
zadnji = ukupno % 90
if zadnji > 50:
   duboki += min(zadnji - 50, 20)
   rem += max(zadnji - 70, 0)
print(duboki)
print(rem)
```

Potrebno znanje: string

Kategorija: ad-hoc

8.2. Zadatak: Excel — Ideja: Adrian Satja Kurdija

Zadatak je moguće riješiti na više načina.

Očekivano rješenje. Najprije simulirajmo bojenje danih dvaju pravokutnika, tj. u matrici označimo obojena poja npr. brojem 1, a ostala polja npr. brojem 0.

Budući da je opseg zapravo duljina ruba, pogledajmo gdje se u našoj matrici nalazi rub obojenog područja. On se nalazi između obojenih i neobojenih polja.

Preciznije, za svako obojeno polje promotrimo njegove susjede (gore, dolje, lijevo, desno). Ako je neki od tih susjeda neobojen, tada je granica između njega i promatranog obojenog polja dio ruba koji treba uračunati u opseg. Ta je granica duljine **W** ili **H**, ovisno o tome je li susjedstvo vertikalno ili horizontalno.

Valja pritom voditi računa i o slučaju kada se obojeno polje nalazi na rubu tablice (npr. u prvom retku), jer njegov rub tada pridonosi opsegu iako s druge strane ne postoji susjedno polje. To možemo riješiti bilo dodatnom provjerom, bilo dodavanjem neobojenih polja oko čitave tablice (nulti i jedanaesti redak i stupac).

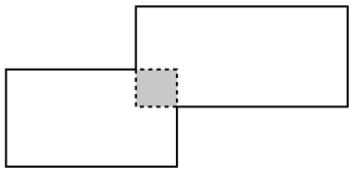
Programski kod (pisan u Python 3)

```
w, h = map(int, input().split())
a = [[0 for j in range(15)] for i in range(15)]
for p in range(2):
    polje1, polje2 = input().split()
    r1 = int(polje1[1:])
    r2 = int(polje2[1:])
    s1 = ord(polje1[0]) - ord('A') + 1
    s2 = ord(polje2[0]) - ord('A') + 1
    for i in range(r1, r2 + 1):
        for j in range(s1, s2 + 1):
            a[i][j] = 1
opseg = 0
for i in range(11):
    for j in range(11):
        opseg += w * (a[i][j] != a[i + 1][j])
        opseg += h * (a[i][j] != a[i][j + 1])
print(opseg)
```

Matematičko rješenje. Zadatak zapravo traži opseg unije dvaju pravokutnika. Njega možemo računati formulom uključivanja-isključivanja:

$$\text{opseg}(\mathbf{A} \text{ unija } \mathbf{B}) = \text{opseg}(\mathbf{A}) + \text{opseg}(\mathbf{B}) - \text{opseg}(\mathbf{A} \text{ presjek } \mathbf{B}),$$

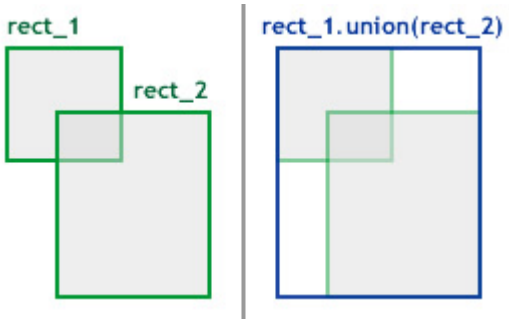
pri čemu su **A** i **B** dani pravokutnici, a (**A** presjek **B**) njihov zajednički dio, na crtežu označen tamnom bojom:



Zaista, iz crteža se vidi: zbrojimo li opsege danih dvaju pravokutnika i od toga oduzmemo opseg presjeka, ostat će nam traženi opseg unije. Primijetimo da je presjek, ako postoji, također pravokutnik čije duljine stranica valja izračunati.

"Tricky" slučaj ovog rješenja nastupa kada se **A** i **B** dodiruju, kao u prvom primjeru iz teksta zadatka. Tada je presjek pravokutnik širine 0, a njegovu visinu treba dvaput uračunati kao i inače u formuli za opseg, iako na slici vidimo samo jednu dužinu.

Rješenje natjecatelja Daniela Širole. Možda i najjednostavnije rješenje dobivamo sljedećom dosjetkom: ako se pravokutnici dodiruju ili preklapaju, dovoljno je izračunati opseg njihovog "bounding boxa", tj. većeg pravokutnika koji ih obuhvaća. Na sljedećoj slici taj je pravokutnik označen plavom bojom i lako se vidi da je njegov opseg jednak traženom opsegu obojenog područja:



Potrebno znanje: stringovi, matrice

Kategorija: ad-hoc

8.3. Zadatak: Otok — Ideja: Adrian Satja Kurdija

Ovaj zadatak može se riješiti na mnogo načina koji nose visok broj bodova. Ovdje ćemo prikazati jedno od tih rješenja i način razmišljanja koji je do njega doveo.

Najprije uočimo da se ne isplati raditi polja odluke u obliku križića (polja koja imaju četiri susjeda) jer ono ne vrijedi ništa više od polja odluke u obliku slova T (s trima susjedima), a zauzima više prostora.

Nakon što smo ovo primijetili, brzo se nameće ideja o rješenju koje sadrži oblike slične ovome:

```
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
##################################
.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
```

Svako polje u drugom retku ovog oblika jest polje odluke, osim prvog i zadnjeg polja u retku. Time je gotovo svako treće polje polje odluke. Za proizvoljne dimenzije otoka iscrtat ćemo ovakve oblike jedan ispod drugoga pazeći da, zbog uvjeta o ciklusima, polja jednog oblika ne budu susjedna poljima drugog, kao što je vidljivo na idućoj slici.

```
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
###################################
.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
###################################
.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
```

Preostaje nam spojiti ove oblike kako bismo zadovoljili uvjet povezanosti iz zadatka, ali tako da nema ciklusa. Jedna od prvih ideja koja nam može pasti na pamet jest jednostavno spojiti središnje retke oblika jednim vertikalnim stupcem na jednoj strani matrice. Pokazalo se kako je upravo ovaj način spajanja jedan od boljih. Iduća slika pokazuje primjer takvog spajanja s vertikalnim stupcem na lijevoj strani matrice.

```
#..#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
###################################
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
#..#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
###################################
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
```

Svi dosadašnji primjeri prikazivali su matrice s brojem redaka djeljivim s tri. Što dodati u zadnjim redcima kada broj redaka nije djeljiv s tri? Ako imamo dva retka "viška", možemo jednostavno ispisati gornji dio našeg elementarnog oblika kao da je odrezan odozdo. U slučaju kada je jedan redak "viška", ne možemo postupiti na ovaj način jer staze ne bi bile povezane. Kao dobro rješenje pokazao se postupak prikazan na idućoj slici.

```
#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#
###################################
#..#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.#.
#.###.###.###.###.###.###.###.###.###
```

Važno je također primijetiti da je ovo rješenje bolje ako je broj stupaca veći od broja redaka. Kako bi rješenje bilo jednako dobro i kad je broj redaka veći od broja stupaca, jednostavno matricu okrenemo za 90 stupnjeva, ispišemo staze na prethodno opisan način te dobivenu matricu prije ispisa ponovno okrenemo za 90 stupnjeva. Na taj način ne moramo raditi posebna rješenja za ova dva analogna slučaja.

Pokazalo se da gore opisano rješenje daje dobre rezultate za veće matrice, no za matrice manjih dimenzija postoje specifične konfiguracije koji su znatno bolje od onih koje generira ovo rješenje. Ovo

možemo riješiti tako da matrice malih dimenzija riješimo ručno i te staze zapišemo u kod. Tada program za manje matrice ispiše naše ručno napisane staze, a za veće koristi gore opisano rješenje.

Programski kod (pisan u C++u)

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

const int MAXN = 200;

int n, m;
bool polje[MAXN][MAXN];

int main() {
    scanf("%d%d", &n, &m);
    bool transposed = false;
    if (n > m) {
        transposed = true;
        swap(n, m);
    }
    // stablo
    for (int i = 0; i < n; i++)
        polje[i][0] = true;

    // grane
    for (int i = 1; i < n; i += 3) {
        // drvo
        for (int j = 1; j < m; j++)
            polje[i][j] = true;
        // gornje lisce
        for (int j = 2; j < m; j += 2)
            polje[i - 1][j] = true;
        // donje licse
        if (i + 1 < n)
            for (int j = 3; j < m; j += 2)
                polje[i + 1][j] = true;
    }
```

```
    // 2 x m
    if (n == 2)
        for (int i = 0; i < m; i++)
            polje[0][i] ^= true;

    // 3 x 3
    if (n == 3 && m == 3)
        polje[2][2] = true;

   // 4 x 4
    if (n == 4 && m == 4) {
        polje[2][2] = polje[3][1] = true;
        polje[2][3] = polje[3][0] = false;
   }

    // 3n + 1 x m
    if (n % 3 == 1 && n > 1)
        for (int i = 2; i < m; i += 4)
            if (i + 1 < m)
                for (int j = 0; j < 3 && i + j < m; j++)
                    polje[n - 1][i + j] = true;

    if (transposed)
        swap(n, m);

    for (int i = 0; i < n; i++) {
        for (int j = 0; j < m; j++)
            printf("%c", (transposed ? polje[j][i] : polje[i][j]) ? '#' : '.');
        printf("\n");
    }
    return 0;
}
```

Potrebno znanje: dizajn algoritma

Kategorija: ad-hoc