17. ožujka 2016.

Državno natjecanje / Osnovna škola
(5. / 6. / 7. / 8. razred)
Primjena algoritama OŠ

OPISI ALGORITAMA

ministarstvo znanosti obrazovanja i sporta
AGENCIJA ZA ODGOJ I OBRAZOVANJE
HRVATSKA ZAJEDNICA TEHNIČKE KULTURE
HRVATSKI SAVEZ INFORMATIČARA

5.1. Zadatak: Putuj — Ideja: Nikola Dmitrović

Za potpuno točno rješenje ovog zadatka dovoljno je pažljivo implementirati sve uvjete koji su postavljeni u zadatku.

Programski kod (pisan u Pythonu 3)

```
S = int(input())
M = int(input())
X = int(input())
if S == 6:
    X = X - (60 - M)
else:
    if M >= 30:
        X = X + (M - 30)
M += X
if M >= 60:
    S += 1
    M -= 60
print(X)
print(S, M)
```

Potrebno znanje: naredba odlučivanja

Kategorija: ad hoc

5.2. Zadatak: Kalendar — Ideja: Adrian Satja Kurdija

Zadatak rješavamo u dvjema fazama:

Računamo koliko je ukupno dana prošlo od početka godine do zadanog datuma. To činimo tako da zbrojimo trajanja (u danima) svih mjeseci prije zadanog i tome pribrojimo zadani dan u mjesecu. Npr. 15. svibnja je 135. dan u godini jer opisanim zbrajanjem dobivamo:

31 (siječanj) + 28 (veljača) + 31 (ožujak) + 30 (travanj) + 15 = 135.

Ovo zbrajanje olakšat ćemo ako prethodno trajanja svih mjeseci zapišemo u niz.

Budući da Cotsworthov mjesec ima 28 dana, dobiveni broj dijelimo s 28 i ovisno o količniku i ostatku dobivamo traženi datum; za detalje pogledajte priloženi kod.

Programski kod (pisan u Pythonu 3)

```
mj = [31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31]
D, M = map(int, input().split())
if D == 31 and M == 12:
    print('Dan nove godine')
    exit(0)
for m in range(M - 1):
    D += mj[m]
```

```
if D % 28 == 0:
    print(28, D // 28)
else:
    print(D % 28, D // 28 + 1)
```

Potrebno znanje: nizovi

Kategorija: ad hoc

5.3. Zadatak: Igra — Ideja: Mislav Balunović

Zamislimo da smo povezali brojeve koje je zaokružio Mirko na način opisan u zadatku.

Za svaki takav paran broj Slavko može raditi sljedeće:

On će redom zaokruživati znamenke koje se nalaze iza tog broja sve dok ne zaokruži neparnu znamenku ili dok ne dođe do neke već zaokružene znamenke ili dok ne dođe do kraja niza.

Primijetimo da je ovim postupkom na papiru ostao najviše jedan paran broj.

Nakon toga, Slavko ide redom po svim znamenkama i zaokruži znamenku ako ispunjava sljedeća 2 uvjeta:

1. Znamenka je neparna
2. Znamenka lijevo od nje nije zaokružena

Prateći opisanu strategiju Slavko će ostvariti maksimalan broj bodova.

Kompletan dokaz da je ovo optimalna strategija bazira se na sljedećem:

- Pretpostavimo da imamo neko optimalno rješenje
- Ako prije Slavkovog zaokruživanja postoji paran broj, onda zaokružene znamenke između njega i sljedećeg broja možemo namjestiti kao i u opisanom algoritmu
- Ako zanemarimo već prije zaokružene brojeve, Slavko će zaokružiti maksimalan mogući broj neparnih znamenaka čiji lijevi susjed nije zaokružen.

Razradu dokaza ostavljamo čitatelju za vježbu.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <iostream>
#include <cstring>

using namespace std;
const int MAXN = 110;
int n, a[MAXN], t[MAXN], s[MAXN];

int main(void) {
```

```
  scanf("%d",&n);
  for (int i = 0; i < n; ++i)
    scanf("%d",&a[i]);
  for (int i = 0; i < n; ++i)
    scanf("%d",&t[i]);

  for (int i = n - 1; i >= 0; --i)
    if (t[i] && a[i] % 2 == 0) {
      int j;
      for (j = i + 1; j < n && !t[j] && a[j] % 2 == 0; ++j)
      s[j] = 1;
      if (j < n && !t[j])
      s[j] = 1;
    }
  for (int i = 0; i < n; ++i) {
    if (t[i] || s[i]) continue;
    if ((i == 0 || (!t[i - 1] && !s[i - 1])) && a[i] % 2 == 1) {
      s[i] = 1;
      ++i;
    }
  }
  for (int i = 0; i < n; ++i)
    printf("%d ",s[i]);
  printf("\n");
  return 0;
}
```

Potrebno znanje: nizovi

Kategorija: simulacija

6.1. Zadatak: Jonathon — Ideja: Dominik Gleich

Zadatak rješavamo jednostavnim kretanjem po polju sve dok ne dođemo do tražene vrijednosti. Zbog jednostavnosti na početak niza dodajemo vrijednost 0 kako bismo sve pozicije i indekse imali numerirane od 1, te kako bismo mogli pristupati elementu s[A]. Dok se krećemo nizom i ne dođemo do vrijednosti koju tražimo povećavamo rješenje za 1.

Programski kod (pisan u Pythonu 2)

```
N = input()
A = input()
s = [0]
for i in range(N-1):
  x = input()
  s.append(x)
ans = 0
while A != N:
  A = s[A]
  ans += 1
print ans
```

Potrebno znanje: nizovi

Kategorija: ad hoc

6.2. Zadatak: Sunce — Ideja: Nikola Dmitrović

Zahtjevan zadatak, ne toliko idejno koliko koderski. Nužno je analizirati svih 6 mogućih slučajeva predstavljaju li dobar datum u budućnosti. Za to nam je potrebno napisati funkciju koja provjerava je li nešto uopće datum, je li godina prestupna i nalazi li se datum u budućnosti.

Programski kod (pisan u Pythonu 3)

```
days = [31,28,31,30,31,30,31,31,30,31,30,31]
def leap(g):
    if (g%4==0):
        if (g%100==0):
            return (g%400==0)
        else:
            return True
    else:
        return False
def datum (d,m,g):
    if m == 0 or d == 0:
```

```
        return False
    return (m <= 12 and d <= (days[m-1]+leap(g)*(m==2)))
def future(d,m,g):
    if g > 2016:
        return True
    elif g == 2016:
        if m > 3:
            return True
        elif m == 3:
            return (d > 17)
    return False

D = input()
L = []

# ddmmgggg
dd = int(D[:2]); mm = int(D[2:4]); gggg = int(D[4:])
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')
# mmddgggg
mm = int(D[:2]); dd = int(D[2:4]); gggg = int(D[4:])
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')
# ddggggmm
dd = int(D[:2]); gggg = int(D[2:6]); mm = int(D[6:])
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')
# mmggggdd
mm = int(D[:2]); gggg = int(D[2:6]); dd = int(D[6:])
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')
# ggggmmdd
gggg = int(D[:4]); mm = int(D[4:6]); dd = int(D[6:])
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')
# ggggddmm
gggg = int(D[:4]); dd = int(D[4:6]); mm = int(D[6:])
```

```
if (datum(dd,mm,gggg) and future(dd,mm,gggg)):
    L.append(str(int(dd))+ '.' + str(int(mm)) + '.' + str(int(gggg))+ '.')


L = list(set(L))
for i in L:
    print(i)
```

Potrebno znanje: stringovi, funkcije

Kategorija: ad hoc

6.3. Zadatak: Traka — Ideja: Mislav Bradač

Kako bismo uspješno riješili ovaj zadatak potrebno je simulirati savijanje trake. Traku možemo predstaviti pomoću niza ili liste te na svakoj poziciji u listi ili nizu moramo pamtiti tri podatka: broj koji je vidljiv s gornje strane trake, broj koji je vidljiv s donje strane trake te broj slojeva papira na tom dijelu trake.

Možemo primijetiti da prilikom savijanja manje od pola trake, traku možemo podijeliti na tri dijela:

- dio koji savijamo
- dio na kojem će se nalaziti preklopljeni dio nakon savijanja
- dio koji nikako ne sudjeluje u savijanju

Savijanje trake simuliramo tako da iz niza u kojem čuvamo izgled trake u drugi niz kopiramo drugi i treći dio te nakon toga spojimo s prvim dijelom. Prilikom spajanja primijetimo da će novi gornji vidljiv broj biti donji vidljiv broj iz prvog dijela, novi donji vidljiv broj biti donji vidljiv broj iz drugog dijela te da će broj slojeva biti jednak zbroju broja slojeva iz prvog i drugog dijela. Sličan algoritam možemo napraviti kod savijanja više od pola trake te se ostavlja čitatelju za vježbu. Savijanje je dobro implementirati u jednoj funkciji, npr. fold_paper kako bismo ju mogli koristiti u raznim dijelovima programa.

Nakon prethodno opisane simulacije savijanja možemo odgovoriti na prva tri pitanja iz zadatka. Kako bismo odgovorili na četvrto pitanje potrebno je isprobati sve kombinacije savijanja u 2 poteza te zapamtiti najbolji rezultat. S obzirom da već imamo implementiranu funkciju fold_paper za savijanje papira, isprobavanje kombinacija se svodi na dvije petlje koje biraju smjer savijanja te dvije petlje koje biraju broj polja koje savijamo i dva poziva funkcije fold_paper.

Programski kod (pisan u Pythonu 3)

```
def copy_paper(a, s, b, os, l):
  for i in range(l):
    b[os + i] = a[s + i]


def flip_and_merge_cells(a, b):
  return (b[1], b[0] if a[1] == -1 else a[1], a[2] + b[2])
```

```
def add_flipped(a, s, b, os, l):
  for i in range(l):
    b[os + l - 1 - i] = flip_and_merge_cells(b[os + l - 1 - i], a[s + i])

def fold_paper(t, v, a):
  n = len(a)
  b = [(-1, -1, 0)] * max(v, n - v)
  if t == 'L':
    copy_paper(a, v, b, 0, n - v)
    add_flipped(a, 0, b, 0, v)
  else:
    copy_paper(a, 0, b, max(0, 2 * v - n), n - v)
    add_flipped(a, n - v, b, max(0, n - 2 * v), v)
  return b

def find_sum(a):
  ret = 0
  for x in a:
    ret += x[0]
  return ret

def find_min(t1, t2, a):
  ret = 1000000000
  for i in range(1, len(a)):
    b = fold_paper(t1, i, a)
    ret = min(ret, find_sum(b))
    for j in range(1, len(b)):
      c = fold_paper(t2, j, b)
      ret = min(ret, find_sum(c))
  return ret

def main():
  n = int(input())
  input_up = list(map(int, input().split()))
  input_down = list(map(int, input().split()))
  input_count = [1] * n
  a = list(zip(input_up, input_down, input_count))
```

```
  k = int(input())
  for i in range(k):
    t, v = input().split()
    a = fold_paper(t, int(v), a)

  print(len(a))
  cnt = 0
  for x in a:
    cnt = max(cnt, x[2])
  print(cnt)
  paper_sum = find_sum(a)
  print(paper_sum)
  paper_sum = min(paper_sum,
      find_min('L', 'L', a),
      find_min('L', 'R', a),
      find_min('R', 'L', a),
      find_min('R', 'R', a)
  )
  print(paper_sum)
main()
```

Potrebno znanje: nizovi

Kategorija: simulacija

7.1. Zadatak: Zagrijavanje Ideja: Dominik Gleich

U ovom zadatku treba biti oprezan s implementacijom kako ne bismo pristupali ilegalnim pozicijama u stringu, i točno pretvorili sva mala slova u velika, ukoliko se nalaze ispred crtice. Poseban slučaj na koji neka rješenja nisu pazila je slučaj kada je donja crtica na kraju stringa.

Najjednostavniji način za riješit zadatak je prolaziti originalnim stringom s lijeva na desno i dodavati znak iz njega u novi string, string koji ćemo ispisati kao rješenje. Postoje dva glavna slučaja, od kojih se drugi dijeli na još dva slučaja:

1) Znak je donja crtica ('_'), u tom slučaju označavamo da se crtica pojavila, nije se "iskoristila" da neko slovo postane veliko slovo i stavljamo ju normalno u novi string.
2) Znak nije donja crtica ('_')
   a) Postoji neiskorištena donja crtica na zadnjem mjestu u stringu rješenja. Tu crticu brišemo iz stringa rješenja, i na kraj stringa rješenja dodajemo trenutno slovo, ali veliko. Također označavamo da ne postoji crtica lijevo od sljedećeg slova.
   b) Ne postoji neiskorištena crtica lijevo od nas, u tom slučaju slovo jednostavno dodajemo u rješenje, onakvo kakvo je.

Programski kod (pisan u Pythonu 2)

```
S = raw_input()
ans = ""
up = 0
for x in S:
  if x == '_':
    up = 1
    ans += x
  else:
    if up:
      ans = ans[:-1]
      ans += x.upper()
      up = 0
    else: ans += x

print ans
```

Potrebno znanje: stringovi

Kategorija: ad hoc

7.2. Zadatak: Simetrija — Ideja: Adrian Satja Kurdija

Os simetrije prolazi ili vrhom, ili polovištem stranice mnogokuta. Za svaki vrh mnogokuta, kao i za svaku stranicu, provjeravamo može li os simetrije tuda proći tako da for-petljom prođemo lijevo i desno od odabranog vrha ili stranice te redom usporedimo boje vrhova koji bi, u slučaju simetrije, morali biti jednaki na lijevoj i desnoj strani. Da bismo implementirali ovu ideju, budući da je dobiveni string cikličan, možemo ga slobodno umnožiti dva ili tri puta da prilikom provjere ne izađemo izvan granica stringa.

Programski kod (pisan u Pythonu 3)

```
n = int(input())
a = input()
a = a + a + a
osi = 0
for i in range(n, n + n):
    # Provjera za vrh i.
    sim = True
    for j in range(n):
        if a[i - j] != a[i + j]:
            sim = False
    osi += sim

    # Provjera za stranicu [i, i+1].
    sim = True
    for j in range(n):
        if a[i - j] != a[i + 1 + j]:
            sim = False
    osi += sim
# Svaku os brojili smo dvaput, s obiju strana, pa dijelimo s 2.
print(osi // 2)
```

Potrebno znanje: stringovi

Kategorija: ad hoc

7.3. Zadatak: Papir — Ideja: Mislav Bradač

Zadatak rješavamo simulacijom savijanja papira. Papir u programu možemo prikazati pomoću dvodimenzionalnog polja čije ćelije predstavljaju vidljive kvadratiće papira. U svakoj ćeliji ćemo čuvati zbroj brojeva ispod pripadajućeg kvadratića papira. Svako savijanje možemo napraviti u dva koraka. U prvom koraku u novo polje kopiramo dio papira koji se ne presavija, a u drugom koraku dodajemo

vrijednosti ćelija dijela papira koji se presavija na ćelije s kojima će se preklopiti. Opisana simulacija je implementirana u rješenjima papir.py i papir.cpp.

U ovom zadatku se moglo olakšati implementaciju simulacije savijanja na mnoge načine, npr. možemo primijetiti da nam je potrebno implementirati samo savijanje u jednom smjeru, recimo prema desno, te funkciju za rotaciju sadržaja polja. U tom slučaju svako presavijanje možemo postići tako da polje rotiramo određeni broj puta te onda izvršimo savijanje prema desno i ponovno rotiramo polje dok ne dođe u pravu poziciju. Presavijanje u lijevo bi se postiglo nizom funkcija: rotiraj(), rotiraj(), savini_udesno(), rotirat(), rotiraj(). Ova implementacija nije prikazana u u priloženim kodovima te se ostavlja čitatelju za vježbu.

Postoji i drugi pristup zadatku koji ne uključuje simulaciju cijelog polja savijanje po savijanje nego za svaki početni kvadratić određuje gdje će se nalaziti nakon svih savijanja. Možemo odabrati jedan kvadratić te izračuanti njegovo poziciju nakon svakog savijanja. Nakon zadnjeg savijanja dodajemo vrijednost koja piše na tom kvadratiću na njegovu konačnu poziciju u polje u kojem pamtiti rješenje. Detalji ove implementacije se mogu vidjeti u kodovima papir2.py i papir2.cpp.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <cstring>
#include <algorithm>
#include <cassert>
using namespace std;
const int MAXN = 110;

int n, m, k;
int a[2][MAXN][MAXN];

void copy_paper(int in[MAXN][MAXN], int isr, int isc,
    int out[MAXN][MAXN], int osr, int osc, int lr, int lc) {
  for (int i = 0; i < lr; ++i) {
    for (int j = 0; j < lc; ++j) {
      out[osr + i][osc + j] = in[isr + i][isc + j];    }   }     }

void add_flipped_horizontal(int in[MAXN][MAXN], int isr, int isc,
    int out[MAXN][MAXN], int osr, int osc, int lr, int lc) {
  for (int i = 0; i < lr; ++i) {
    for (int j = 0; j < lc; ++j) {
      out[osr + lr - 1 - i][osc + j] += in[isr + i][isc + j];    }   } }

void add_flipped_vertical(int in[MAXN][MAXN], int isr, int isc,
    int out[MAXN][MAXN], int osr, int osc, int lr, int lc) {
```

```
  for (int i = 0; i < lr; ++i) {
    for (int j = 0; j < lc; ++j) {
      out[osr + i][osc + lc - 1 - j] += in[isr + i][isc + j];    }  }  }

void fold_paper(char t, int v, int curr) {
  int next = 1 - curr;
  memset(a[next], 0, sizeof a[next]);
  if (t == 'U') {
    copy_paper(a[curr], v, 0, a[next], 0, 0, n - v, m);
    add_flipped_horizontal(a[curr], 0, 0, a[next], 0, 0, v, m);
    n = max(v, n - v);
  } else if (t == 'D') {
    copy_paper(a[curr], 0, 0, a[next], max(0, 2 * v - n), 0, n - v, m);
    add_flipped_horizontal(a[curr], n - v, 0, a[next], max(0, n - 2 * v), 0, v,
m);
    n = max(v, n - v);
  } else if (t == 'L') {
    copy_paper(a[curr], 0, v, a[next], 0, 0, n, m - v);
    add_flipped_vertical(a[curr], 0, 0, a[next], 0, 0, n, v);
    m = max(v, m - v);
  } else if (t == 'R') {
    copy_paper(a[curr], 0, 0, a[next], 0, max(0, 2 * v - m), n, m - v);
    add_flipped_vertical(a[curr], 0, m - v, a[next], 0, max(0, m - 2 * v), n,
v);
    m = max(v, m - v);
  } else {
    assert(false);
  }
}
int main() {
  scanf("%d%d", &n, &m);
  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      scanf("%d", &a[0][i][j]);
    }
  }
  scanf("%d", &k);
  for (int i = 0; i < k; ++i) {
```

```
    char t;
    int v;
    scanf(" %c%d", &t, &v);
    fold_paper(t, v, i % 2);
  }

  for (int i = 0; i < n; ++i) {
    for (int j = 0; j < m; ++j) {
      printf("%d%c", a[k % 2][i][j], j == m - 1 ? '\n' : ' ');
    }
  }
  return 0;
}
```

Potrebno znanje: nizovi

Kategorija: ad hoc, simulacija

8.1. Zadatak: Red — Ideja: Dario Babojelić

Najprije učitane podatke pretvorimo u sekunde kako bismo lakše uspoređivali vremena dolazaka. Treba primijetiti da osoba na poziciji i nije nezadovoljna ako i samo ako ispred nje ne postoji osoba s većim vremenom dolaska, tj. ako je vrijeme dolaska osobe i maksimum od svih vremena dolazaka osoba na pozicijama od prve do i-te.

Iz toga slijedi da je osoba i nezadovoljna ako je max(v[1], v[2], …, v[i]) različit od v[i] gdje je v[i] vrijeme dolaska osobe koja stoji na i-toj poziciji. Jednim prolaskom po ulaznim podacima lako održavamo maksimalno vrijeme dolaska do trenutne pozicije. Ako je taj maksimum do trenutne pozicije različit od vremena dolaska na toj poziciji povećavamo brojač nezadovoljnih ljudi za jedan.

Programski kod (pisan u Pythonu 3)

```
n = int(input())
nezadovoljni = 0
maksi = -1
for i in range(n):
     h, m, s = map(int, input().split(":"))
     s += h * 3600 + m * 60
     maksi = max(maksi, s)
     if s != maksi:
          nezadovoljni += 1
print(nezadovoljni)
```

Potrebno znanje: održavanje maksimuma

Kategorija: ad hoc

8.2. Zadatak: Sretan — Ideja: Adrian Satja Kurdija

Ključno je pitanje sljedeće: koji sretan broj odabrati u trenutku kada dosadašnji sretan broj ne pokriva broj autobusa koji slijedi? Jasno je da treba odabrati onaj sretan broj koji će pokriti što više sljedećih brojeva. Koliko ih je moguće pokriti? Da bismo to efikasno saznali, prolazimo redom po brojevima koje slijede i pamtimo najmanji i najveći od njih. U trenutku kad njihova razlika postane veća od raspona koji može pokriti jedan sretan broj, povećavamo broj promjena za jedan, "resetiramo" minimum i maksimum te ponovno promatramo brojeve koje slijede.

Programski kod (pisan u Pythonu 3)

```
n = int(input())
s, k = map(int, input().split())
promjene = 0
mini = 1000 * 1000
maxi = 0
for i in range(n):
```

```
    x = int(input())
    if s > 0:
        if abs(x - s) < k:
            continue
        promjene += 1
        s = -2 * k
    mini = min(mini, x)
    maxi = max(maxi, x)
    if maxi - mini > 2 * k - 2:
        promjene += 1
        mini = maxi = x
print(promjene)
```

Potrebno znanje: nizovi

Kategorija: ad hoc

8.3. Zadatak: Mat — Ideja: Mislav Balunović

Za uspješno rješavanje ovog zadatka bez greške program je bilo poželjno razdvojiti u nekoliko komponenata.

Koristimo nizove dr i dc koji pamte pomake za svaki od mogućih 8 smjerova.

Struktura Figura za članove ima redak, stupac i tip figure.

Za svaki tip figure znamo u kojem podskupu dozvoljenih smjerova se može kretati te koliko najviše poteza može napraviti u tom smjeru (1 za kralja, a 8 za ostale)

Implementiramo sljedeće ključne funkcije:

- *provjeriSah* - za svaku figuru, pomaknemo ju na sva moguća polja do kojih ona može doći te ako je jedno od njih polje na kojemu se nalazi protivniči kralj imamo šah
- *provjeriMat* - za svako pomicanje protivničkog kralja pozovemo provjeriSah te na kraju pozovemo provjeriSah i za početnu konfiguraciju te ako je u svim slučajevima šah onda imamo mat
- *kraljeviSusjedni* - provjerava jesu li kraljevi u trenutnoj konfiguraciji susjedni

U glavnom dijelu rješenja, iteriramo po svim figurama i za svaku figuru probamo ju pomaknuti na bilo koje mjesto na koje ona može doći. Ako za neki od tih poteza funkcija *provjeriMat* vraća istinu i *kraljeviSusjedni* vraća laž pronašli smo potez kojim matiramo protivničkog kralja.

Programski kod (pisan u C++)

```
#include <cstdio>
#include <iostream>
#include <cstring>
```

```
#include <vector>

#define r first
#define c second
using namespace std;
typedef pair<int, int> field;

const int N = 8;
const int dr[] = {-1, 1, 0, 0, 1, -1, 1, -1};
const int dc[] = {0, 0, 1, -1, 1, 1, -1, -1};
const int from[] = {0, 0, 4};
const int to[] = {7, 3, 7};
const int maxMoves[] = {1, 8, 8};

struct Figura {
  int r, c, tip;
  Figura(int _r = 0, int _c = 0, int _tip = 0) { r = _r; c = _c; tip = _tip;}};

char a[N + 2][N + 2];
vector<Figura> figs;
Figura mir, sla;
bool attack[N + 2][N + 2];

inline int myabs(int x) { return x < 0 ? -x : x; }

Figura nadjiFiguru(char z, int tip) {
  for (int r = 0; r < N; ++r)
    for (int c = 0; c < N; ++c)
      if (a[r][c] == z)
      return Figura(r, c, tip);
  return Figura(-1, -1, -1);}

bool provjeriSah(int kr, int kc) {
  for (int i = 0; i < figs.size(); ++i) {
    Figura fig = figs[i];
    for (int j = from[fig.tip]; j <= to[fig.tip]; ++j) {
      int r = fig.r, c = fig.c;
```

```
      for (int i = 0; i < maxMoves[fig.tip]; ++i) {
      r += dr[j];
      c += dc[j];
      if (r < 0 || c < 0 || r >= N || c >= N) break;
      if (kr == r && kc == c) return true;
      if (fig.tip != 0 && a[r][c] != '.') break;      }    }  }
  return false; }

bool provjeriMat(void) {
  for (int j = 0; j < 8; ++j) {
    int nr = mir.r + dr[j], nc = mir.c + dc[j];
    if (nr < 0 || nc < 0 || nr >= N || nc >= N) continue;
    char tmp = a[nr][nc];
    a[nr][nc] = 'M';
    a[mir.r][mir.c] = '.';
    bool sah = provjeriSah(nr, nc);
    a[nr][nc] = tmp;
    a[mir.r][mir.c] = 'M';
    if (!sah)
      return false;
  }
  return provjeriSah(mir.r, mir.c);
}

bool kraljeviSusjedni(void) {
  Figura mk = nadjiFiguru('M', 0);
  Figura sk = nadjiFiguru('S', 0);
  return myabs(mk.r - sk.r) <= 1 && myabs(mk.c - sk.c) <= 1;}

void ispisiMatricu(void) {
  for (int r = 0; r < N; ++r)
    printf("%s\n",a[r]);}

int main(void) {
  for (int r = 0; r < N; ++r)
    scanf("%s",a[r]);
```

```
mir = nadjiFiguru('M', 0);
figs.push_back(sla = nadjiFiguru('S', 0));
if (nadjiFiguru('T', 1).r != -1) figs.push_back(nadjiFiguru('T', 1));
if (nadjiFiguru('L', 2).r != -1) figs.push_back(nadjiFiguru('L', 2));

for (int i = 0; i < figs.size(); ++i)  {
  Figura& fig = figs[i];
  for (int j = from[fig.tip]; j <= to[fig.tip]; ++j) {
    int tr = fig.r, tc = fig.c;
    char fig_znak = a[fig.r][fig.c];
    for (int i = 0; i < maxMoves[fig.tip]; ++i) {
    fig.r += dr[j];
    fig.c += dc[j];
    if (fig.r < 0 || fig.c < 0 || fig.r >= N || fig.c >= N) break;
    if (fig.tip != 0 && a[fig.r][fig.c] != '.') break;
    if (fig.tip == 0 && a[fig.r][fig.c] != '.') continue;

    a[fig.r][fig.c] = fig_znak;
    a[tr][tc] = '.';

    if (provjeriMat() && !kraljeviSusjedni()) {
      ispisiMatricu();
      return 0;
    }
    a[fig.r][fig.c] = '.';
    a[tr][tc] = fig_znak;
    }
    fig.r = tr;
    fig.c = tc;    }  }
return 0;}
```

Potrebno znanje: matrice

Kategorija: Simulacija