2. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

24. januarja 2025

NASVETI ZA TEKMOVALCE

Naloge na tem šolskem tekmovanju pokrivajo širok razpon težavnosti, tako da ni nič hudega, če ne znaš rešiti vseh. V okviru svoje rešitve obvezno poleg izvorne kode v nekaj stavkih opiši, kako deluje tvoja rešitev in na kakšni ideji temelji.

Pri ocenjevanju so vse naloge vredne enako število točk. Svoje odgovore dobro utemelji. Prizadevaj si predvsem, da bi bile tvoje rešitve pravilne, ob tem pa je zaželeno, da so tudi čim bolj učinkovite (take dobijo več točk kot manj učinkovite). Za manjše sintaktične napake se načeloma ne odbije veliko točk. Priporočljivo in zaželeno je, da so tvoje rešitve napisane pregledno in čitljivo. Če je na listih, ki jih oddajaš, več različic rešitve za kakšno nalogo, jasno označi, katera je tista, ki naj jo ocenjevalci upoštevajo.

Če naloga zahteva branje ali obdelavo vhodnih podatkov, lahko tvoja rešitev (če v nalogi ni drugače napisano) predpostavi, da v vhodnih podatkih ni napak (torej da je njihova vsebina in oblika skladna s tem, kar piše v nalogi).

Nekatere naloge zahtevajo branje podatkov s standardnega vhoda in pisanje na standardni izhod. Za pomoč je tu nekaj primerov programov, ki delajo s standardnim vhodom in izhodom:

- Program, ki prebere s standardnega vhoda dve števili in izpiše na standardni izhod njuno vsoto:

```
program BranjeStevil;
var i, j: integer;
begin
  ReadLn(i, j);
  WriteLn(i, ' + ', j, ' = ', i + j);
end. {BranjeStevil}
```

```
#include <stdio.h>
int main() {
  int i, j;
  scanf("%d %d", &i, &j);
  printf("%d + %d = %d\n", i, j, i + j);
  return 0;
}
```

- Program, ki bere s standardnega vhoda po vrsticah, jih šteje in prepisuje na standardni izhod, na koncu pa izpiše še skupno dolžino:

```
program BranjeVrstic;
var s: string; i, d: integer;
begin
  i := 0; d := 0;
  while not EOF do begin
    ReadLn(S); i := i + 1;
    d := d + Length(s);
    WriteLn(i, '. vrstica: "', s, '"');
  end; {while}
  WriteLn('Skupaj ', i, ' vrstic, ', d,
      ' znakov.');
end. {BranjeVrstic}
```

```
#include <stdio.h>
#include <string.h>

int main() {
  char s[101]; int i = 0, d = 0;
  while (scanf("%[^\n]%*c", s) == 1) {
    i++; d += strlen(s);
    printf("%d. vrstica: \"%s\"\n", i, s); }
  printf("Skupaj %d vrstic, %d znakov.\n",
         i, d);
  return 0;
}
```

Opomba: C-jevska različica gornjega programa predpostavlja, da ni nobena vrstica vhodnega besedila daljša od sto znakov. V praksi je bolje uporabljati funkcije oz. načine, ki dovoljujejo omejitev branja tudi glede na velikost tabele, vendar imamo na programerskih tekmovanjih omejitve vhodnih podatkov in zagotovila glede njih, tako da bo tudi tak scanf zadoščal.

- Program, ki bere s standardnega vhoda po znakih, jih prepisuje na standardni izhod, na koncu pa izpiše še število prebranih znakov (ne vštevši znakov za konec vrstice):

```
program BranjeZnakov;
var i: integer; c: char;
begin
  while not EOF do
    begin
      Read(c); Write(c); i := i + 1
    end; {while}
  WriteLn('Skupaj ', i, ' znakov.');
end. {BranjeZnakov}
```

```
#include <stdio.h>
int main() {
  int i = 0, c;
  while ((c = getchar()) != -1){
    i++; putchar(c);
  }
  printf("Skupaj %d znakov.\n", i);
  return 0;
}
```

Še isti trije primeri v pythonu:

```
# Branje dveh \v stevil in izpis vsote:
a, b = input().split()
a = int(a)
b = int(b)
print(a, b, a + b)
```

```
# Branje standardnega vhoda po vrsticah:
import sys

idx_vrstice = st_znakov = 0
for vrstica in sys.stdin:
  vrstica = vrstica.rstrip('\n') # odrežemo znak za konec vrstice
  idx_vrstice += 1
  st_znakov += len(vrstica)
  print(f"{idx_vrstice}. vrstica: '{vrstica}'")
print(f"{idx_vrstice} vrstic, {st_znakov} znakov.")
```

```
# Branje standardnega vhoda znak po znak:
import sys

st_znakov = 0
while True:
  znak = sys.stdin.read(1)
  if znak == "":
    break # Konec vhoda
  sys.stdout.write(znak) # Izpišemo znak
  if znak != '\n':
    st_znakov += 1
print(f"Skupaj {st_znakov} znakov.")
```

Še isti trije primeri v javi:

```java
// Branje dveh števil in izpis vsote:
import java.io.*;
import java.util.Scanner;
public class Primer1 {
    public static void main(String[ ] args) throws IOException {
        Scanner fi = new Scanner(System.in);
        int i = fi.nextInt(); int j = fi.nextInt();
        System.out.println(i + " + " + j + " = " + (i + j));
    }
}
```

```java
// Branje standardnega vhoda po vrsticah:
import java.io.*;
public class Primer2 {
    public static void main(String[ ] args) throws IOException {
        BufferedReader fi = new BufferedReader(new InputStreamReader(System.in));
        int i = 0, d = 0; String s;
        while ((s = fi.readLine()) != null) {
            i++; d += s.length();
            System.out.println(i + ". vrstica: \"" + s + "\""); }
        System.out.println(i + " vrstic, " + d + " znakov.");
    }
}
```

```java
// Branje standardnega vhoda znak po znak:
import java.io.*;
public class Primer3 {
    public static void main(String[ ] args) throws IOException {
        InputStreamReader fi = new InputStreamReader(System.in);
        int i = 0, c;
        while ((c = fi.read()) >= 0) {
            System.out.print((char) c); if (c != '\n' && c != '\r') i++; }
        System.out.println("Skupaj " + i + " znakov.");
    }
}
```

2. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

24. januarja 2025

NALOGE ZA ŠOLSKO TEKMOVANJE

Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši idejo, na kateri temelji tvoja rešitev. Če ni v nalogi drugače napisano, lahko tvoje rešitve predpostavljajo, da so vhodni podatki brez napak (da ustrezajo formatu in omejitvam, kot jih podaja naloga). Zaželeno je, da so tvoje rešitve poleg tega, da so pravilne, tudi učinkovite (bolj učinkovite rešitve dobijo več točk). Naloge so štiri in pri vsaki nalogi lahko dobiš od 0 do 25 točk.

Rešitve bodo objavljene na `https://rtk.ijs.si/`.

1. Priden študent

Študenti morajo po bolonjski reformi vsako leto za uspešno opravljen letnik zbrati 60 kreditnih točk ECTS. Pri tem naj bi vsaka kreditna točka predstavljala med 25 in 30 urami dela.

Pomagaj ubogim študentom ugotoviti, koliko dela imajo še pred sabo, če želijo uspešno zaključiti leto.

Napiši program, ki bo prebral število kreditnih točk T, ki jih je študent Stane letos že pridobil, in število ur H, ki jih je že vložil v svoje preostale obveznosti. Potem naj program izpiše, koliko ur dela Staneta še čaka, preden gre lahko brezskrbno na počitnice. Ker je Stane hitre narave, za vsako kreditno točko obveznosti potrebuje le 25 ur.

Tvoj program lahko bere podatke s standardnega vhoda ali pa iz datoteke `vhod.txt` (karkoli ti je lažje). Vedno bo veljalo $0 \leq T \leq 60$ in $0 \leq H \leq 1000$.

Primer: če je $T = 15$ in $H = 37$, mora program izpisati 1088. Stane mora pridobiti še 45 točk, za katere je vložil že 37 ur dela.

2. Največkrat citirano

V znanstvenih člankih se pogosto sklicujemo na druge članke, knjige in drugačne vire. Običajno imamo zato na koncu članka seznam literature, vsakič, ko se v samem besedilu sklicujemo na določen članek, pa napišemo samo `[n]`, kjer je `n` številka citiranega članka.

Napiši program, ki v danem besedilu poišče največkrat citiran članek. Številke citiranih člankov bodo vedno med 1 in 9. Izpiši številko največkrat citiranega članka. Tvoj program lahko bere podatke s standardnega vhoda ali pa iz datoteke `vhod.txt` (karkoli ti je lažje). Prav tako lahko privzameš obliko vhoda, ki ti je najbolj všečna – lahko imaš ves članek v eni vrstici, lahko na začetku prebereš še število vrstic, na katere je razdeljen članek ...

Primer:

> Pojem self-motion manifold je vpeljal že Burdick [2]. Ti »manifolds« pa niso nujno zares mnogoterosti in zato jih bomo v tem delu imenovali sukališča kinematične preslikave (v prihodnje sukališča). Pojem izpeljujemo iz dejstva, da se izvajalo robota pri premikanju sklepov po sukališču ne premika, sklepi pa se »sučejo« okoli te ene točke delovnega prostora. Poleg sukališč je Burdick vpeljal še nekaj topoloških konceptov in oznak [2], ki so ključne za razumevanje predstavljene metode za iskanje pregrad delovnih prostorov.
>
> Lück in Lee v [3], [4] raziskujeta, kako se spremeni kinematika robota, ko vpeljemo kinematične omejitve, ki jih lahko izrazimo kot prepovedana območja konfiguracijskega prostora K.

V besedilu se največkrat pojavi citat 2. Če je v besedilu nekaj različnih člankov citiranih enako pogosto, naj tvoj program izpiše številko kateregakoli izmed njih.

3. Toti pravokotnik

Napiši program, ki ugotovi, ali so podane točke oglišča pravokotnika, poravnanega s koordinatnimi osmi. Na vhodu so v štirih vrsticah podane koordinate točk, ki so cela števila. Točke so podane v smeri urinega kazalca. Program naj izpiše `DA`, če so točke oglišča pravokotnika, sicer pa `NE`. Pri tem upoštevaj tudi, da naj ima pravokotnik neničelno površino (ni izrojen primer, kjer se dve stranici prekrivata).

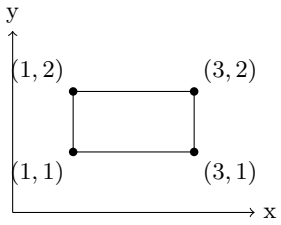
Primer:

Vhod	*Izhod*

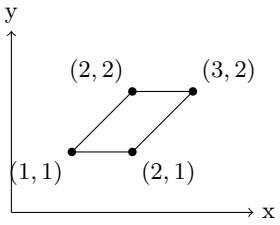
`1 2`
`3 2`
`3 1`
`1 1`
	`DA`

`3 2`
`2 1`
`1 1`
`2 2`
	`NE`

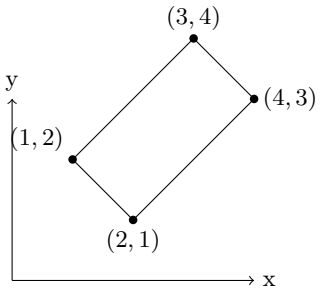
`2 1`
`2 1`
`1 1`
`2 2`
	`NE`



Prvi primer - točke opisujejo pravokotnik.



Drugi primer - točke opisujejo paralelogram, ki ni pravokotnik.



Tretji primer - točke sicer tvorijo pravokotnik, vendar je zavrten, in zato ne ustreza našim zahtevam.

4. Matrika v LaTeX-u

Vsi resni matematiki svoje tabele izrisujejo z LaTeX-om. Pri tem pa so te tabele v izvorni kodi pogosto videti neurejene in zato te komisija Mladega RTK prosi za program, ki bo olepšal že napisane tabele v izvorni kodi. Vsebina tabele v LaTeX-u izgleda tako, da je najprej vsaka vrstica tabele v svoji vrstici izvorne kode in na njenem koncu sta dve levi poševnici oz. znaka backslash (\\). Stolpce v vrstici pa razločujemo z znakom *in* oz. *ampersand* (&).

Izvorno kodo tabele LaTeX prevede v lepo tabelo, pri urejanju pa nam pride prav, če je tabela pregledna. To lahko opišemo s tremi pridevniki – da je lepa, urejena in negovana.

- Tabela je *lepa*, če ima vsaka vrstica isto število stolpcev kot ostale.
- Tabela je *urejena*, če ima vsak stolpec konstantno širino (torej vsaka celica iz določenega stolpca ima isto širino). V tem primeru širina pomeni število znakov.
- Tabela je *negovana*, če so znaki *in* (&) od vsebine celice ločeni s presledkom.

Napiši program, ki prebere vsebino *lepe* tabele in izpiše to isto vsebino, le da bo sedaj tabela *lepa*, *urejena*, *negovana*. Če v kakšni celici manjka presledkov (poleg tistih okoli &), jih dodajamo pred vsebino celice. Povedano drugače, vsebino celic želimo imeti poravnano na desno. Za nazoren pogled si oglej primere izhodov spodaj; pri zadnjem v srednjem stolpcu se vidi tudi želeno poravnavo.

V prvi vrstici vhoda se nahaja število V, ki predstavlja število vrstic v tabeli. Nato sledi V vrstic, ki vse vsebujejo $S-1$ znakov & (saj je tabela lepa) in v vsaki celici (med vsakima &-oma, pred prvim in za zadnjim) do 50 poljubnih drugih znakov. Predpostaviš lahko $1 \leq V \leq 20$ in $2 \leq S \leq 10$. Tvoj program naj izpiše V vrstic, ki vsebujejo preoblikovano tabelo.

Primer:

Vhod

```
3
4 & 9 & 2 \\
3 & 5 & 7 \\
8 & 1 & 6 \\
```

Izhod

```
4 & 9 & 2 \\
3 & 5 & 7 \\
8 & 1 & 6 \\
```

Vhod

```
4
16&3&2&13 \\
5&10&11&8 \\
9&6&7&12 \\
4&15&14&1 \\
```

Izhod

```
16 &  3 &  2 & 13 \\
 5 & 10 & 11 &  8 \\
 9 &  6 &  7 & 12 \\
 4 & 15 & 14 &  1 \\
```

Vhod

```
8
1 &&& 1 &&&& \\
& \ddots && \vdots &&&& \\
&& n-1 & n-1 &&&& \\
1 & \cdots & n-1 & n & n+1 & n+2 & \cdots & 2n \\
&&& n+1 & n+1 &&& \\
&&& n+2 && n+2 && \\
&&& \vdots &&& \ddots & \\
&&& 2n &&&& 2n \\
```

Izhod

```
    1 &        &     &      1 &     &     &        &    \\
      & \ddots &     & \vdots &     &     &        &    \\
      &        & n-1 &    n-1 &     &     &        &    \\
    1 & \cdots & n-1 &      n & n+1 & n+2 & \cdots & 2n \\
      &        &     &    n+1 & n+1 &     &        &    \\
      &        &     &    n+2 &     & n+2 &        &    \\
      &        &     & \vdots &     &     & \ddots &    \\
      &        &     &     2n &     &     &        & 2n \\
```

2. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

24. januarja 2025

REŠITVE NALOG ŠOLSKEGA TEKMOVANJA

1. Priden študent

V nalogi moramo prebrati števili T in H, ugotoviti, da Stanetu manjka še $60 - T$ kreditnih točk, in izpisati $25 \cdot (60 - T) - H$.

```c
#include <stdio.h>

int main() {
  int T; // Število že pridobljenih kreditnih točk
    int H; // Število že vloženih ur
    scanf("%d %d", &T, &H);
    printf("%d\n", 25 * (60 - T) - H);
    return 0;
}
```

Zapišimo takšno rešitev še v pythonu:

```python
T, H = input().split()
T = int(T)
H = int(H)
print(25 * (60 - T) - H)
```

Pri taj nalogi je 10 točk vredno pravilno branje podatkov, 15 točk pa pravilen izračun in izpis rezultata. Tekmovalcem ni treba preverjati, ali je rezultat slučajno negativen, ni pa nič narobe, če to naredijo.

2. Največkrat citirano

V tej nalogi moramo prebrati vse besedilo in v njem poiskati števila podnizov tipa `[n]`, kjer je `n` med 1 in 9. Izpisati moramo najpogostejše od teh števil. Pristopa sta načeloma dva, lahko preberemo cel niz in nato v njem iščemo vzorce `[n]` ali pa beremo znak po znak in beležimo zadnje tri prebrane znake. Tako izgleda rešitev v C-ju:

```c
#include <stdio.h>

bool je_stevka(char c) {
    // Vrne true, če je znak c število, sicer false.
    return '0' <= c && c <= '9';
}

int main() {
    char ta, prejsnji, predprejsnji;
    char c;
    int i = 0;

    // Števila pojavitev [1], [2], ..., [9].
    // Seznam se začne z indeksom 0. Da se indeksi ujemajo s številkami,
    // preskočimo mesto 0. Seznam je zato dolg 10.
    int kolicine[] = { 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 };

    while ((c = getchar()) != EOF) {
        // Se sprehodimo po znakih vhoda in beležimo zadnje 3.
        // Zadnje 3 zamaknemo za eno mesto nazaj in dodamo c.
        predprejsnji = prejsnji; prejsnji = ta; ta = c;
        i++;
        if (i >= 3) {
            // Če smo prebrali vsaj 3 znake, preverimo, ali je to vzorec [n].
```

```
            if (predprejsnji == '[' && je_stevka(prejsnji) && ta == ']') {
                // Na ustreznem mestu v tabeli povečamo število pojavitev.
                kolicine[prejsnji - '0']++;
            }
        }
    }

    // Najdemo število, ki se pojavi največkrat.
    int najvec = 1;
    for (int i = 2; i <= 9; i++) {
        if (kolicine[i] > kolicine[najvec]) {
            najvec = i;
        }
    }
    printf("%d\n", najvec);
    return 0;
}
```

Zapišimo takšno rešitev še v pythonu:

```
import sys

# Preberemo ves vhod v text
text = sys.stdin.read()
najboljsi = 0
for i in range(10):
  # Prestejemo vse pojavitve '[i]' za vse mozne i
  c = text.count(f"[{i}]")
  if c > najboljsi:
    najboljsi = c
print(najboljsi)
```

V tej nalogi je

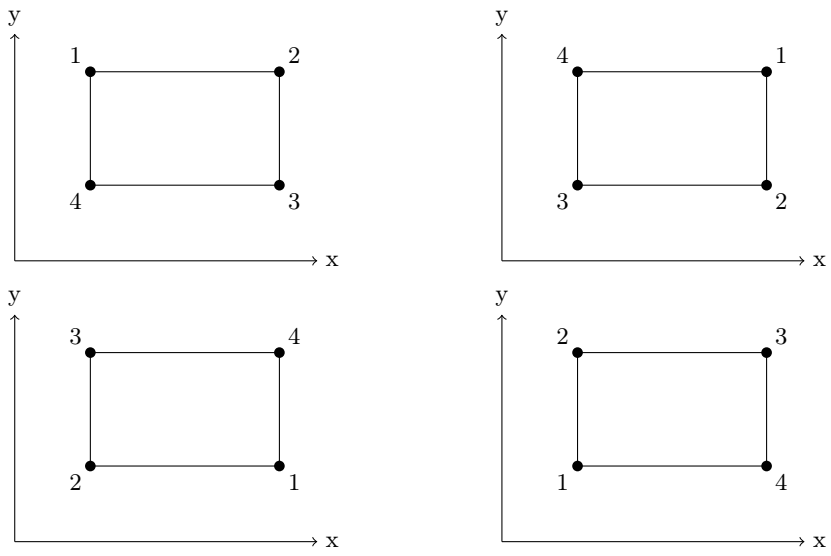
- 5 točk za pravilno branje podatkov,
- 15 točk za pravilno štetje pojavitev števil in
  Če tekmovalec le prešteje pojavitve števil in ne skrbi, da se ta pojavijo v oglatih oklepajih, za to izgubi 8 točk.
  Če tekmovalec ne skrbi, da ne gleda znakov pred začetkom ali po koncu besedila, izgubi 4 točke.
- 5 točk za pravilno iskanje največkrat pojavljenega števila.

3. Toti pravokotniki

Pravokotnik je lik, ki ima dva para vzporednih stranic, ki se sekajo pod pravim kotom. Ker je pravokotnik poravnan s koordinatnimi osmi, bosta imeli točki enega roba bodisi enako x-koordinato, bodisi enako y-koordinato.

Nalogo lahko rešujemo na veliko načinov, tukaj pa bomo opisali dva izmed njih. Rešitev v programskem jeziku C pa poda še tretji, bolj splošen način.

Prvi je, da preverimo, ali se zaporedne točke ujemajo po x ali y koordinati. Ker so točke podane v smeri urinega kazalca, se lahko zgodijo štirje primeri:



- V primerih na levi imata prvi dve točki enako y-koordinato, druga in tretja enako x-koordinato, tretja in četrta enako y-koordinato, četrta in prva enako x koordinato.
- V primerih na desni je ravno obratno, torej prva in druga enako ter tretja in četrta enako x-koordinato, druga in tretja ter četrta in prva enako y-koordinato.

Drugi način je, da med točkami najdemo tiste z najmanjšo x-koordinato in med njimi tisto z najmanjšo y-koordinato. Tako dobimo spodnji levi kot, nakar lahko preverimo, ali si točke za tem kotom sledijo v smeri urinega kazalca (torej druga nad najdenim kotom, tretja na višini druge in zadnja na višini najdenega kota in oddaljenosti druge). Tu upoštevamo, da je prva točka naslednja (v smeri urinega kazalca) zadnji.

Tretji način, zapisan v C-ju, je sledeč:

```c
#include <stdio.h>

int main(){
    int i, j, numx, numy, x[4], y[4], ok=1;
    for(i=0;i<4;++i)
        scanf("%d %d", x+i, y+i);
    for(i=0;i<4;++i){
        //    Presteje, koliko drugih tock se ujema s trenutno v kateri koordinati.
        //    Ce se kaka tocka ujema v obeh koordinatah, sta si enaki in ne tocke ne
        //    bodo predstavljale pravokotnika. Ce vsaka tocka deli tocno eno
        //    koordinato s tocno eno drugo tocko, imamo pravokotnik.
        for(numx=numy=j=0;j<4;++j){
            if(i==j) continue;
            if(x[i]==x[j]) ++numx;
            if(y[i]==y[j]) ++numy;
            if(y[i]==y[j] && x[i]==x[j]) ok=0;
        }
        if(numx!=1 || numy!=1) ok=0;
    }
    if(ok) puts("DA");
    else puts("NE");
    return 0;
}
```

Drugi način, zapisan v pythonu:

```python
# Preberemo x in y koordinate točk
x_koord = []; y_koord = []
for i in range(4):
    x, y = input().split()
    x_koord.append(int(x))
    y_koord.append(int(y))

# Najdemo indeks točke, ki je spodnji levi kot
```

```
levi_rob = []
for i in range(4):
    if x_koord[i] == min(x_koord):
        levi_rob.append(i)
spodnji_levi = levi_rob[0]
for i in levi_rob:
    if y_koord[i] < y_koord[spodnji_levi]:
        spodnji_levi = i

# Ker vemo, da so točke urejene v smeri urinega kazalca, vemo, da mora biti
# (i+1)-a (naslednja) levi zgornji kot, (i+2)-a zgornji desni kot in (i+3)-a
# spodnji desni kot. Ker želimo na seznam točk gledati kot cikličen 'neskončen'
# seznam, vzamemo indekse 'po modulu 4'.
# Če sta si nasprotna vogala delita katero od kordinat je pravokotnik izrojen.
if (x_coord[i] == x_coord[(i+1)%4] and y_coord[(i+1)%4] == y_coord[(i+2)%4] and
    x_coord[(i+2)%4] == x_coord[(i+3)%4] and y_coord[(i+3)%4] == y_coord[i] and
    x_coord[i] != x_coord[(i+2)%4] and y_coord[i] != y_coord[(i+2)%4]):
    print("DA")
else:
    print("NE")
```

Pri tej nalogi je

- 3 točke za pravilno branje podatkov in pravilen izpis,
- 3 točke za pravilno preverjanje izrojenosti pravokotnika,
- 19 točk za pravilno preverjanje, ali so točke pravokotnik

Če tekmovalec predpostavi samo eno možnost zaporedja točk (npr. prva je spodnja leva, druga zgornja leva, ...), naj ne dobi več kot 11 točk pri tej alineji. Če pa samo ni imel časa izpisati vseh primerov v kodi ali psevdokodi, je pa komentiral, da bi moral obravnavati še ostale primere, lahko za to dobi tudi do 15 točk.

4. Matrika v LaTeX-u

Če želimo, da bo tabela izgledala poravnano, morajo imeti vse celice v enem stolpcu enako širino. Seveda celic ne moremo krajšati, lahko pa jih podaljšamo tako, da jim na začetek dodajamo presledke. Najprej poiščemo širino vsake od celic v tabeli. Za vsak stolpec izračunamo širino najširše celice v njem. Ko bomo celice iz tega stolpca izpisovali, bomo morali s presledki zagotoviti, da bodo vse celice imele to širino.

Število presledkov v začetni tabeli je relevantno, zato jih želimo ohraniti. Zato pri računanju širine upoštevamo, če se celica začne in konča na presledek – če se ne, bomo morali tam presledka še vriniti, sicer bosta pa obstoječa morda že dovolj.

- izpišemo toliko presledkov, da bo celica imela enako širino kot najširša celica v tem stolpcu,
- izpišemo vsebino celice, s tem da preskočimo presledka spredaj in zadaj, če obstajata,
- izpišemo znak & s presledkom pred in po njim.

Zgornje ponovimo za vse celice v vrstici. Za \\ na koncu vrstice lahko opazimo, da je naloga enaka, če ju tretiramo kot del zadnje celice v vsaki vrstici.
Tako izgleda rešitev v C-ju:

```
#include <stdio.h>
#include <string.h>

int max(int a, int b){
    return a<b?b:a;
}

char vrstice[100][(100+1)*50+3];

int main(){
    int v, s, i, j, sirina[101]={0}, trenutnasirina;
    char *pchl, *pchd;
    scanf("%d%*c", &v);

    for(i=0;i<v;++i){
        scanf("%[^\n]%*c", vrstice[i]);
        //    pchl in pchd predstavljata pointerja na levi in desni konec
```

```
        //    posamezne celice. Premikamo ju vsakic z iskanjem naslednjega znaka
        //    '&'.
        pchl=vrstice[i];
        pchd=strchr(pchl, '&');

        //    pchl[0] je prvi znak na levi strani naslednje celice. Ko je ta
        //    znak 0 (na koncu vrstice), nehamo z iskanjem naslednjega konca.
        for(s=0;pchl[0];++s){

            //    Izracunamo trenutno sirino. Pri tem upostevamo, da, ce ima
            //    celica ze presledek spredaj ali zadaj, ga nam ni treba dodajati
            //    oz. v programu, ce presledka ni, ga bomo morali v izhodu dodati.
            trenutnasirina=pchd-pchl-1;
            if(s!=0 && pchl[0]!=' ') ++trenutnasirina;
            if(pchd[0]!=0 && pchd[-1]!=' ') ++trenutnasirina;

            //    Sproti racunamo najvecjo sirino za vsak stolpec.
            sirina[s]=max(sirina[s], trenutnasirina);

            //    Poiscemo konca naslednje celice, ce obstaja naslednja celica.
            pchl=pchd+1;
            pchd=strchr(pchl, '&');
            if(pchd==NULL) pchd=pchl+strlen(pchl);
        }
    }

    for(i=0;i<v;++i){
        pchl=vrstice[i];
        pchd=strchr(pchl, '&');
        for(s=0;pchl[0];++s){
            //    Glede na sirino trenutnega stolpca in celice, izpisemo toliko
            //    presledkov, da bo celica siroka toliko, kot je stolpec. Pri tem
            //    upostevamo morebitni manjkajoci presledek na koncu celice.
            trenutnasirina=pchd-pchl-1;
            if(pchd[0]!=0 && pchd[-1]!=' ') ++trenutnasirina;
            for(j=trenutnasirina;j<sirina[s];++j) putchar(' ');

            //    Izpisemo vsebino celice.
            pchd[0]=0;
            printf("%s", pchl);
            pchd[0]='&';

            //    Ce na koncu celice ni bilo presledka, ga dodamo.
            if(pchd[0]!=0 && pchd[-1]!=' ') putchar(' ');

            //    Ce smo v zadni celici vrstice izpisemo znak za novo vrstico
            //    '\n', sicer pa '&' za mejo do naslednje celice.
            pchl=pchd+1;
            if(pchl[0]!=0) putchar('&');
            else putchar('\n');
            pchd=strchr(pchl, '&');
            if(pchd==NULL) pchd=pchl+strlen(pchl);
        }
    }
    return 0;
}
```

Tako pa še v pythonu z uporabo nekaterih vgrajenih funkcij za delo s nizi:

```
import sys

tabela = []

V = int(input()) # Preberemo število vrstic
for line in sys.stdin:
```

```
    line = line
    ta_vrstica = list(line.split("&"))
    tabela.append(ta_vrstica)

# Izračunamo širine celic v tabeli.
# Ker iščemo največjo, bomo vrednosti na začetku nastavili na 0.
sirine = [0 for _ in range(tabela[0])]
for vrstica in tabela:
    for i, celica in enumerate(vrstica):
        # Racunamo sirino dejanske celice, zato upostevamo, ce ze imamo spredaj
        # in zadaj presledka. Ce ju imamo ju odstranimo, saj ju bomo kasneje
        # v vsakem primeru izpisali.
        sirina = len(celica)
        if celica[0] == ' ':
            sirina -= 1
            vrstica[i] = celica[1:]
        if celica[-1] == ' ':
            sirina -= 1
            vrstica[i] = celica[:-1]
        sirine[i] = max(sirine[i], sirina)

for vrstica in tabela:
    for i, celica in enumerate(vrstica):
        # Izpišemo celico z dodanimi presledki, da je širina enaka maksimalni
        # širini celice v stolpcu.
        presledekzadaj = celica[-1] == ' '
        print(" " * (sirine[i] - len(celica)) + celica, end="")
        # Vse celice razen zadnje končamo z " & "
        if i != len(vrstica) - 1:
            print(" & ", end="")
        else:
            print()
```

Pri pythonu smo upoštevali, da funkcija `print` sama doda znak za novo vrstico na koncu, če ne podamo argumenta `end=""`.

Rešitev te naloge je posebej dolga in vse prej kot enostavna, zato od tekmovalcev ne pričakujemo, da bo popolna. Tekmovalec naj dobi tudi do 15 točk, če njegova rešitev primerno opiše grobo idejo prave rešitve (torej računanje širine stolpca kot največjo širino vseh celic v njem), pa tudi, če njegov program ne bi zares rešil naloge v celoti. Sicer pa okvirno razdelimo 25 točk na sledeč način:

- 5 točk, če tekmovalec na smiseln način prebere in si shrani vhodne nize.
- 10 točk, če tekmovalec pravilno izračuna širine celic in največje širine celic po stolpcih.
- 10 točk za pravilno izpisovanje celic.

  5 točk za to, da se celice izpišejo v pravilni širini.

  5 točk za to, da je med celicami pravilno postavljen znak &, in da ima na obeh straneh presledek.

2. osnovnošolsko tekmovanje ACM v znanju računalništva

Šolsko tekmovanje

24. januarja 2025

NASVETI ZA MENTORJE O IZVEDBI TEKMOVANJA IN OCENJEVANJU

Tekmovalci naj pišejo svoje odgovore na papir ali pa jih natipkajo z računalnikom; ocenjevanje teh odgovorov poteka v vsakem primeru tako, da jih pregleda in oceni mentor (in ne npr. tako, da bi se poskušalo izvorno kodo, ki so jo tekmovalci napisali v svojih odgovorih, prevesti na računalniku in pognati na kakšnih testnih podatkih). Čas reševanja je omejen na 120 minut.

Glede tega, katere programske jezike tekmovalci uporabljajo, naše tekmovanje ne postavlja posebnih omejitev, niti pri nalogah, pri katerih je rešitev v nekaterih jezikih znatno krajša in enostavnejša kot v drugih (npr. uporaba perla ali pythona pri problemih na temo obdelave nizov).

Kjer se v tekmovalčevem odgovoru pojavlja izvorna koda, naj bo pri ocenjevanju poudarek predvsem na vsebinski pravilnosti, ne pa na sintaktični. Pri ocenjevanju na državnem tekmovanju zaradi manjkajočih podpičij in podobnih sintaktičnih napak odbijemo mogoče kvečjemu eno točko od dvajsetih; glavno vprašanje pri izvorni kodi je, ali se v njej skriva pravilen postopek za rešitev problema. Ravno tako ni nič hudega, če npr. tekmovalec v rešitvi v C-ju pozabi na začetku **#include**ati kakšnega od standardnih headerjev, ki bi jih sicer njegov program potreboval; ali pa če podprogram `main()` napiše tako, da vrača **`void`** namesto **`int`**.

Pri vsaki nalogi je možno doseči od 0 do 25 točk. Od rešitve pričakujemo predvsem to, da je pravilna (= da predlagani postopek ali podprogram vrača pravilne rezultate), poleg tega pa je zaželeno tudi, da je učinkovita (manj učinkovite rešitve dobijo manj točk).

Če tekmovalec pri neki nalogi ni uspel sestaviti cele rešitve, pač pa je prehodil vsaj del poti do nje in so v njegovem odgovoru razvidne vsaj nekatere od idej, ki jih rešitev tiste naloge potrebuje, naj vendarle dobi delež točk, ki je približno v skladu s tem, kolikšen delež rešitve je našel.

Če v besedilu naloge ni drugače navedeno, lahko tekmovalčeva rešitev vedno predpostavi, da so vhodni podatki, s katerimi dela, podani v takšni obliki in v okviru takšnih omejitev, kot jih zagotavlja naloga. Tekmovalcem torej načeloma ni treba pisati rešitev, ki bi bile odporne na razne napake v vhodnih podatkih.

Če oblika vhodnih podatkov ni natančno določena, si lahko podrobnosti tekmovalec izbere sam. Na primer, če naloga pravi, da dobimo seznam parov, je to lahko v praksi tabela (*array*), vektor, *linked list* ali še kaj drugega, pari pa so lahko bodisi strukture, ki jih je deklarirala tekmovalčeva rešitev, ali pa kaj iz standardne knjižnice (kot je `pair` v C++ ali `tuple` v pythonu).

Težavnost nalog

Državno tekmovanje ACM v znanju računalništva poteka v dveh težavnostnih skupinah (prva je lažja, druga pa težja); na tem šolskem tekmovanju pa je skupina ena sama, vendar naloge v njej pokrivajo razmeroma širok razpon zahtevnosti. Za občutek povejmo, s katero skupino državnega tekmovanja so po svoji težavnosti primerljive posamezne naloge letošnjega šolskega tekmovanja:

Naloga	Kam bi sodila po težavnosti na državnem tekmovanju ACM
1. Priden študent	lahka naloga v prvi skupini
2. Največkrat citirano	težka naloga v prvi skupini
3. Toti pravokotniki	težka v prvi ali srednje težka v drugi
4. Matrika v LaTeX-u	srednje težka do težka v drugi skupini

Če torej na primer neki tekmovalec reši le eno ali dve lažji nalogi, pri ostalih pa ne naredi (skoraj) ničesar, to še ne pomeni, da ni primeren za udeležbo na državnem tekmovanju; pač pa je najbrž pametno, če se državnega tekmovanja udeleži v prvi skupini.

Podobno kot prejšnja leta si tudi letos želimo, da bi čim več tekmovalcev s šolskega tekmovanja prišlo tudi na državno tekmovanje in da bi bilo šolsko tekmovanje predvsem v pomoč tekmovalcem in mentorjem pri razmišljanju o tem, v kateri težavnostni skupini državnega tekmovanja naj kdo tekmuje.