17. tekmovanje ACM v znanju računalništva za srednješolce

26. marca 2022

NALOGE ZA PRVO SKUPINO

Odgovore lahko pišeš/rišeš na papir ali pa jih natipkaš z računalnikom ali pa oddaš del odgovorov na papirju in del prek računalnika. Vse te možnosti so enakovredne. Če oddajaš kaj na papirju, napiši na vsak oddani list svoje ime. Pri delu si lahko pomagaš s prevajalniki in razvojnimi orodji, ki so na voljo na tvojem računalniku, vendar bomo tvoje odgovore v vsakem primeru pregledali in ocenili ročno (ne glede na to, ali si jih oddal prek računalnika ali na papirju), zato manjše napake v sintaksi ali pri klicih funkcij standardne knjižnice niso tako pomembne, kot bi bile na tekmovanjih z avtomatskim ocenjevanjem.

Tekmovanje bo potekalo na strežniku `https://rtk.fri.uni-lj.si/`, kjer dobiš naloge in oddajaš svoje odgovore. Uporabniška imena in gesla vas bodo čakala na mizi v učilnici. Pri oddaji preko računalnika odpreš dotično nalogo v spletni učilnici in rešitev natipkaš oz. prilepiš v polje za programsko kodo. Med tipkanjem se rešitev na približno dve minuti samodejno shrani. Poleg tega lahko sam med pisanjem rešitve izrecno zahtevaš shranjevanje rešitve s pritiskom na gumb „Shrani spremembe“. Ker je vgrajeni urejevalnik dokaj preprost in ne omogoča označevanja kode z barvami, predlagamo, da rešitev pripraviš v kakšnem drugem urejevalniku na računalniku (Visual Studio Code, Geany, Lazarus) in jo nato prekopiraš v okno spletnega urejevalnika. Naj te ne moti, da se bodo barvne oznake kode pri kopiranju izgubile.

Ko si bodisi zadovoljen z rešitvijo ter si zaključil nalogo ali ko želiš začasno prekiniti pisanje rešitve naloge ter se lotiti druge naloge, uporabi gumb „Shrani spremembe“ in nato klikni na „Nazaj na seznam nalog“, da se vrneš v glavni meni. (Oddano rešitev lahko kasneje še spreminjaš.) Za vsak slučaj priporočamo, da pred oddajo shraniš svoj odgovor tudi v datoteko na svojem lokalnem računalniku.

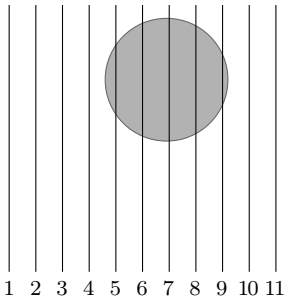
Med reševanjem lahko vprašanja za tekmovalno komisijo postavljaš prek zasebnih sporočil na tekmovalnem strežniku (ikona oblačka zgoraj desno) ali pa vprašaš člane komisije, ki bodo prisotni v učilnicah. Prek zasebnih sporočil bomo pošiljali tudi morebitna pojasnila in popravke, če bi se izkazalo, da so v besedilu nalog kakšne nejasnosti ali napake. Zato med reševanjem redno preverjaj, če so se pojavila kakšna nova zasebna sporočila. Če imaš težave z računalnikom ali s povezavo s spletnim strežnikom za oddajo nalog in komunikacijo s tekmovalno komisijo, se nemudoma obrni na nadzornika v učilnici, ki bo zagotovil drug računalnik. **Če zaradi morebitnih težav pri oddajanju rešitev na strežnik želiš, da ocenimo odgovore v datotekah na lokalnem disku tvojega računalnika, o tem obvezno obvesti nadzorno osebo v svoji učilnici, še preden odideš iz nje.**

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 (s tem je mišljeno predvsem, naj ima rešitev učinkovit algoritem; drobne tehnične optimizacije niso tako pomembne). **Nalog je pet** in pri vsaki nalogi lahko dobiš od 0 do 20 točk. Liste z nalogami lahko po tekmovanju obdržiš.

Rešitve bodo objavljene na `https://rtk.ijs.si/`. Predvidoma nekaj dni po tekmovanju bodo tam objavljeni tudi rezultati.

1. Snežinke

Radi bi imeli napravo za merjenje velikosti snežink. V ta namen imamo rešetko vzporednih toplih merilnih žic, razdalja med njimi je en milimeter. Žice so oštevilčene od 1 do 100. Predpostavimo, da so snežinke okrogle, cela snežinka je vedno znotraj območja merilnih žic in vedno se dotakne vsaj ene žice, nato pa spolzi skozi (snežinke se ne nabirajo na žicah). Med dvema snežinkama je vedno dovolj časovnega razmaka, da se prejšnja stopi in spolzi mimo merilnih žic.



Slika kaže prvih enajst od 100 žic. Snežinko, ki jo predstavlja sivi krog, zaznavajo žice 5, 6, 7, 8 in 9.

Na voljo imaš funkcijo Senzor(n), ki vrne vrednost **true**, če n-ta žica zaznava snežinko, sicer vrne **false**. Predpostavimo, da se snežinka dotakne vseh žic hkrati, ko pa se čez čas stopi, je v istem trenutku ne zaznava nobena žica več. Delovanje programa je dovolj hitro, da lahko v času obstoja ali odsotnosti snežinke na merilnih žicah večkrat odčitamo stanje merilnih žic. Snežinka lahko pade na žice kadarkoli, tudi med dvema zaporednima klicema funkcije Senzor.

Napiši program, ki stalno pregleduje stanje merilnih žic in ob vsaki novi snežinki izpiše (natanko enkrat), koliko žic se je dotaknila (in tako izvemo njeno približno velikost). V primeru snežinke z gornje slike mora program izpisati 5.

2. Semafor

Na nekatere semaforje so v zadnjih letih namestili dodatne prikazovalnike, ki prikazujejo čas v sekundah do vklopa zelene luči. Ker želimo čas čakanja izkoristiti za kaj koristnega, smo v avto vgradili inteligentno kamero, ki prepozna številko na prikazovalniku.

Predpostavi, da je za dostop do kamere na voljo funkcija BeriStevec(), ki počaka, da se stanje prikazovalnika spremeni, in poskuša prepoznati številko, ki je po novem prikazana na prikazovalniku. Žal pa se je pri uporabi pokazalo, da kamera pri prepoznavanju ni vedno najbolj natančna (sonce, megla, kót snemanja), zato funkcija BeriStevec ne vrne nujno ene same številke, ampak eno ali več možnih številk, ki jih je kamera prepoznala. Vse te številke so z območja od 0 do 99; prava številka je zagotovo med njimi. Ko prikazovalnik kaže številko 0, vrne tudi BeriStevec le številko 0 in nobene druge. Funkcija BeriStevec je takšne oblike:

```
vector<int> BeriStevec();          // v C++
public static int[] BeriStevec();  // v javi ali C#
def BeriStevec() -> list[int]: ... # v pythonu
```

```
int stevilke[100];   /* v C/C++: funkcija BeriStevec shrani prepoznane številke v globalno */
int BeriStevec();    /* tabelo „stevilke“, kot vrednost funkcije pa vrne število teh številk. */
```

```
var stevilke: array [1..100] of integer;   { v pascalu; deluje enako }
function BeriStevec: integer;              { kot tista v C/C++ }
```

Napiši program, ki s čim manj zaporednimi klici funkcije BeriStevec ugotovi, katera številka je trenutno na prikazovalniku, in jo izpiše.

Primer: spodnja tabela kaže možen potek dogajanja pri več zaporednih klicih. V prvem stolpcu je prava številka s prikazovalnika, v drugem pa je seznam številk, ki bi ga utegnila vrniti funkcija BeriStevec.

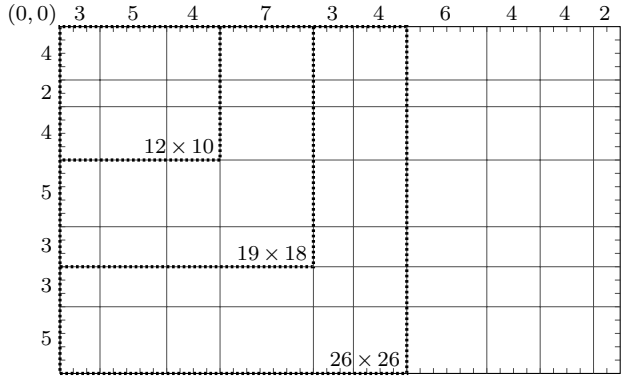
Prikazana številka	Ob klicu BeriStevec vrne
53	$[59, 93, 53, 99]$
52	$[52, 92, 56, 96]$
51	$[51, 57, 97]$
50	$[90, 50, 58, 98]$
49	$[43, 79, 45, 48, 49]$

Po klicih v prikazanem primeru bi se že dalo z gotovostjo zaključiti, da je številka na prikazovalniku res 49 in ne kakšna druga. Tvoj program mora torej takrat izpisati 49.

3. Iskanje kvadrata

Imamo tabelo oz. razpredelnico (*spreadsheet*) z različnimi širinami stolpcev in višinami vrstic. Če vzamemo presek zgornjih nekaj vrstic in levih nekaj stolpcev, lahko dobimo pravokotnike različnih oblik in velikosti (odvisno od tega, koliko vrstic in koliko stolpcev smo uporabili), vsi pa se začnejo v zgornjem levem kotu. Med vsemi takimi pravokotniki želimo izbrati tistega, ki je po obliki čim bližje kvadratu (ali pa je celo res kvadrat). Z drugimi besedami: radi bi, da bi bilo razmerje med dolžino daljše in krajše stranice pravokotnika čim bližje 1.

Opiši postopek (ali napiši program ali podprogram oz. funkcijo, če ti je lažje), ki kot vhodne podatke dobi zaporedje širin stolpcev in zaporedje višin vrstic ter izračuna širino in višino pravokotnika, o katerem govori prejšnji odstavek. Če je možnih več enako dobrih rešitev, je vseeno, katero izpiše. Širine stolpcev in višine vrstic so cela števila, večja od 0. Poleg tega tudi dobro **utemelji**, zakaj tvoj postopek vrača pravilne rešitve.



Primer: na sliki zgoraj imamo stolpce s širinami $[3, 5, 4, 7, 3, 4, 6, 4, 4, 2]$ in vrstice z višinami $[4, 2, 4, 5, 3, 3, 5]$. Izmed vseh možnih pravokotnikov z začetkom v zgornjem levem kotu so označeni trije (z debelimi črtkanimi črtami), od katerih je najprimernejši tisti z velikostjo 26×26.

4. Neprevidni poeti

Znani poet Gimon Sregorčič se je odločil, da bo v umetnosti pesnjenja izuril nekaj vajencev. Ti seveda niso njegove generacije, temveč so bistveno mlajši in imajo na Gimonovo žalost bistveno bolj moderne poglede na svet in, kar je sploh hudo, poezijo. Najbolj opazna stvar je, da se sploh ne držijo ritma. Celo noč je obupan prebiral njihove pesnitve in beležil, kdo je upošteval ritem in kdo ga ni. Zdaj je že tako utrujen, da tudi sam ne zaznava več poudarjenih in nepoudarjenih zlogov in potrebuje pomoč pri ugotavljanju, katere pesmi se držijo ritma.

V besedilu pesmi so nekateri zlogi naglašeni, drugi pa ne. Zaporedje naglašenih in nenaglašenih zlogov tvori ritem. V nalogi so naglašeni samoglasniki označeni z velikimi črkami, vse ostale črke pa so male. Sklop črk je zlog samo v primeru, da ima natanko en samoglasnik (*a*, *e*, *i*, *o* ali *u*) (v besedi „stržen“ je torej po našem štetju en zlog, zlogotvornega *r* ne upoštevamo). **Napiši program**, ki za dano pesem izpiše, ali je v vseh verzih pesmi isti ritem in če da, kakšen je. Tvoj program lahko prebere pesem s standardnega vhoda ali pa iz datoteke `vhod.txt` (kar ti je lažje). Predpostavi, da posamezne vrstice niso daljše od 100 znakov.

Primer vhoda:

```
od nEkdaj lepE so ljubljAnke slovEle
a lEpse od Urske bilO ni nobEne
nobEne ocEm bilo bOlj zazelEne
ne spOmnem se bEsedilA naprej sOri
```

Pripadajoči izhod:

```
DA
U-UU-UU-UU-U
```

Komentar: druga vrstica je ritem, `U` predstavlja nepoudarjen zlog, `-` pa poudarjen zlog.

Še en primer vhoda:

```
cuj vlAka zvIzg z vetrOvi gnAn
med mrAk oblAkov in poljAn
narascajOc, pojemajOc
takO moj klIc gre v nOc polnOc
```

Pripadajoči izhod:

```
NE
```

5. Stonoge

Malokdo ve, da je Alfred Hitchcock pred svojo uspešno grozljivko *Ptiči* posnel tudi *Stonoge*, ki pa iz različnih razlogov, med drugim zaradi izjemno nizkega filmskega proračuna, niso postale uspešnica. Ker stonoge niso pretirano ubogljiva bitja, so v filmu uporabljali izključno umetne stonoge, ki pa niso bile izdelane preveč prepričljivo. Za primer podajmo eno prepričljivo in eno očitno umetno stonogo:

```
.........\|||||\/\///||/.............
.........##############>.............
.........//||||||/\/\/\||\.............
.....................................
.....................\|||/||/.........
....................<######..........
....................//|||\||\.........
```

Trup stonoge torej tvori eden ali več zaporednih znakov „#“ (vsi v eni vrstici), glavo pa predstavlja bodisi znak „<“ tik levo ob trupu ali pa znak „>“ tik desno ob trupu. (V primeru zgoraj ima gornja stonoga glavo na desnem koncu, spodnja pa na levem.) Stonoge so prepričljive, če so vsi njihovi pari nog simetrični in če sta prvi in zadnji par nog usmerjena k trupu. V primeru zgoraj prva stonoga ni prepričljiva, ker peti par nog (gledano od spredaj) ni simetričen (leva noga kaže naprej, desna pa nazaj), druga stonoga pa je prepričljiva. V spodnjem primeru pa nobena stonoga ni prepričljiva, ker se prvi in/ali zadnji par nog pri nobeni ne drži trupa:

```
..............................////............
..||\//.............|\\||/...##>.....//\\|..
...###>.............<####...\\\\......###>..
..||/\\..../|||/....|//||\...........\\//|..
...........<###..............................
...........\|||\.............................
.............................................
```

Napiši program, ki iz položaja stonog v nekem trenutku filma ugotovi, koliko izmed njih je prepričljivih in koliko očitno umetnih. Stonoge se nahajajo na polju iz n vrstic in m stolpcev (n in m sta največ 100). Polje je sestavljeno zgolj iz znakov „\“, „|“, „/“, „<“, „>“, „#“ in „.“. Nobeni dve stonogi se ne prekrivata ali dotikata in nobena stonoga ni v kadru le delno. Predpostaviš lahko, da je z glavami stonog vse v redu in ti ni treba preverjati, ali ima res vsaka stonoga glavo na natanko enem koncu. Podrobnosti tega, v kakšni obliki tvoj program dobi ali prebere vhodne podatke, si izberi sam(a) in jih v svoji rešitvi tudi opiši.