16. tekmovanje ACM v znanju računalništva za srednješolce

27. marca 2021

NALOGE ZA DRUGO SKUPINO

Naloge rešuj samostojno; ne sprašuj drugih ljudi za nasvete ali pomoč pri reševanju (niti v živo niti prek interneta ali kako drugače), ne kopiraj v svoje odgovore tuje izvorne kode in podobno. Tekmovalna komisija si pridržuje pravico, da tekmovalca diskvalificira, če bi se kasneje izkazalo, da nalog ni reševal sam. Internet lahko uporabljaš, če ni v nasprotju s prejšnjimi omejitvami (npr. za branje dokumentacije), vendar za reševanje nalog ni nujno potreben. Tvoje odgovore bomo pregledali in ocenili ročno, 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 (bo)ste dobili po elektronski pošti. Pri oddaji preko računalnika rešitev natipkaš neposredno v brskalniku. 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 urejevalniku na svojem računalniku 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 in zapri“ 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), izjemoma pa tudi po elektronski pošti na `rtk-info@ijs.si`. 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 pri oddaji odgovorov prek spletnega strežnika kakšne težave, lahko izjemoma pošlješ svoje odgovore po elektronski pošti na `rtk-info@ijs.si`, vendar nas morajo doseči pred koncem tekmovanja; odgovorov, prejetih po koncu tekmovanja, ne bomo upoštevali.

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.

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

1. Sredinec

V šoli je pri športni vzgoji navada, da se pred začetkom šolske ure učenci postavijo v vrsto od najmanjšega do največjega. Prav tako je navada, da učenci zamujajo. Vsak učenec, ki vstopi v telovadnico, se vrine na svoje mesto v vrsti glede na velikost. Učitelj športne vzgoje se med tem zamudnim procesom zabava z opazovanjem, kdo se po vsakem novem prihodu nahaja na sredini vrste. Učenci vstopajo posamično, učitelja pa zanima, kako visok je tisti izmed n prisotnih učencev, ki se trenutno nahaja na $\lceil n/2 \rceil$-tem mestu v vrsti od najmanjšega do največjega. (Zapis $\lceil n/2 \rceil$ pomeni, da rezultat po deljenju n z 2 zaokrožimo navzgor. Na primer: pri $n = 5$ in $n = 6$ ga zanima tretji po vrsti, pri $n = 7$ in $n = 8$ četrti po vrsti in podobno.)

Opiši postopek, ki to nalogo reši čim bolj učinkovito (recimo, da učencev ni le nekaj deset, ampak na milijone): prebira naj višine učencev v takem vrstnem redu, kakor vstopajo v telovadnico, in po vsakem prebranem učencu sproti izpiše višino tistega, ki je zdaj srednji po višini. Oceni tudi časovno zahtevnost svoje rešitve, torej kako se povečuje čas izvajanja v odvisnosti od števila učencev. Višine učencev so podane v obliki seznama po vrsti, tako kot vstopajo v telovadnico. Višine niso večje od dveh metrov in so podane s celimi števili, ki predstavljajo višino v centimetrih.

2. Svetilka

Žepna baterijska svetilka je lahko ugasnjena ali pa sveti v dveh možnih načinih: sveti stalno ali pa utripa tako, da vsako sekundo posveti za eno desetinko sekunde (in je potem devet desetink sekunde ugasnjena). Za preklop med temi tremi stanji služi tipka.

Takoj ko pritisnemo tipko (t.j. ob začetku pritisnjenosti tipke) naj se svetilka vklopi: če je bila prej ugasnjena, naj se vklopi v stalni način, če je bila v stalnem načinu, naj se preklopi v utripanje, in če je utripala, naj se preklopi v stalni način. Če je tipka pritisnjena tri sekunde ali več, naj se po teh treh sekundah svetilka izklopi.

Podana je funkcija Luc(vklop), s katero lahko program upravlja svetilo: vrednost **true** vklopi svetilo, **false** ga izklopi.

Napiši naslednji dve **funkciji**, ki ju bo operacijski sistem malega računalnika v svetilki avtomatsko klical takole:

- Tiktak() — ta funkcija bo poklicana vsako desetinko sekunde;
- Tipka(pritisnjena) — ta funkcija bo poklicana vsakokrat, ko se bo stanje pritisnjenosti tipke spremenilo; vrednost argumenta bo **true**, če je bila tipka pravkar pritisnjena (t.j. začetek pritiska), in **false**, če je bila tipka pravkar spuščena.

Za ohranitev stanja programa lahko uporabiš poljubne globalne spremenljivke in jih tudi po svoje inicializiraš. Na začetku delovanja programa je luč ugasnjena, tipka pa spuščena.

Glede na to, da nimamo možnosti merjenja časa z večjo ločljivostjo od desetinke sekunde, ne bo nič narobe, če ob preklopu na utripanje prvi blisk ne traja točno eno desetinko sekunde, prav tako lahko trosekundni interval (za ugašanje svetilke) odstopa za malenkost.

Če si želiš poenostaviti nalogo, lahko opustiš stanje utripanja in poskrbiš le za vklop in izklop svetilke — pri tem boš dobil največ polovico točk naloge.

3. Pletenje puloverja

Neža se je med karanteno lotila novega konjička. Naučila se je plesti. Nekaj preglavic pa ji povzročajo sheme za pletenje vzorcev. V knjigah so pogosto narisane velike sheme, na primer:

```
---0-0---0-0---0-0---0-0---0-0
---000---000---000---000---000
---0-0---0-0---0-0---0-0---0-0
---000---000---000---000---000
---0-0---0-0---0-0---0-0---0-0
---000---000---000---000---000
---0-0---0-0---0-0---0-0---0-0
---000---000---000---000---000
```

Zgornja shema predstavlja osnovni vzorec

```
---0-0
---000
```

Neža je hitro ugotovila, da si mora pri pletenju izdelka zapomniti oziroma zapisati le osnovni vzorec in ne celotne velike sheme iz knjige. Prosi te, da **napišeš program** ali podprogram (funkcijo), ki v poljubni dani shemi poišče osnovni vzorec. Osnovni vzorec je najmanjši (po površini) tak vzorec, iz katerega lahko s ponavljanjem sestavimo celotno shemo (pri čemer se mora vzorec lepo zaključiti na vseh robovih sheme). Program naj izpiše širino in višino osnovnega vzorca. Za primer zgoraj je rešitev `6 2`. Če je možnih več enako dobrih rešitev, je vseeno, katero od njih izpišeš. Shemo lahko tvoj program prebere iz datoteke ali s standardnega vhoda ali pa predpostavi, da je že podana v neki tabeli ali seznamu nizov (ali dvodimenzionalni tabeli znakov). Shemo sestavljajo le znaki „`-`“ in „`0`“.

4. Pangramski podniz

Pangram je niz, ki vsebuje vsako črko abecede vsaj enkrat; pri tej nalogi pa nas bodo zanimali malo bolj posebni pangrami — taki, ki vsebujejo vsako črko abecede vsaj k-krat. **Napiši podprogram** oz. funkcijo, ki za dani niz s in naravno število k vrne dolžino najkrajšega takega strnjenega podniza niza s, v katerem se vsaka črka abecede pojavi vsaj k-krat. Če takega podniza sploh ni, naj funkcija vrne -1. Niz s je sestavljen le iz malih črk angleške abecede, lahko pa je zelo dolg, zato naj bo tvoja rešitev čim bolj učinkovita.

Primer: če bi namesto cele abecede gledali le črke $\{\texttt{a}, \texttt{b}, \texttt{c}\}$ in če bi imeli $k = 2$, bi bil najkrajši primerni podniz v nizu $s = \texttt{aabaaccabaacccbabb}$ dolg 7 znakov. Taki podnizi so celo trije: `baaccab`, `baacccb`, `acccbab`. Poudarimo pa, da je to samo primer in da mora tvoja rešitev delovati za celotno abecedo in za poljuben k in poljubno dolg niz s.

5. Tetris

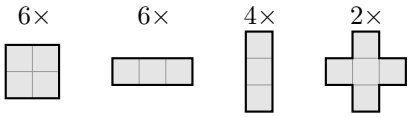
Imamo ploščo, sestavljeno iz 8×8 kvadratnih polj, ki bi jo radi pokrili s ploščki v obliki raznih likov, podobnih tistim iz igre Tetris. **Napiši program** ali podprogram (funkcijo), ki bo ploščo v celoti pokril s ploščki, pri čemer se le-ti med seboj ne smejo prekrivati ali štrleti čez rob plošče. Na voljo so ploščki n različnih oblik, ki so oštevilčene od 1 do n; v vsaki obliki pa je na voljo le omejeno število ploščkov. Za delo s ploščki naj tvoj program uporablja naslednje funkcije (zanje torej predpostavi, da že obstajajo in ni mišljeno, da jih ti implementiraš sam):

- **int** StOblik() — vrne n, torej število, ki pove, koliko različnih oblik ploščkov je na voljo.
- **int** StPlosckov(**int** oblika) — vrne število razpoložljivih ploščkov oblike oblika. To je celo število, večje od 0, vanj pa so všteti tudi tisti ploščki, ki jih je tvoj program mogoče že postavil na ploščo.
- **bool** JePokrito(**int** x, **int** y) — vrne logično vrednost, ki pove, ali je polje (x, y) na plošči trenutno pokrito (torej ali ga pokriva kakšen od že doslej postavljenih ploščkov). Na začetku izvajanja tvojega programa je plošča prazna (torej ni na njej še nobenega ploščka). Koordinate polj na plošči gredo od $x = 0$ (levo) do $x = 7$ (desno) in od $y = 0$ (zgoraj) do $y = 7$ (spodaj).
- **bool** PreveriPloscek(**int** oblika, **int** x, **int** y) — vrne logično vrednost, ki pove, ali je mogoče na ploščo dodati plošček oblike oblika tako, da najbolj levo polje v najbolj zgornji vrstici tega ploščka pokrije polje (x, y) na plošči. (Funkcija preverja le obliko, ne pa tudi tega, ali imaš še na voljo kaj ploščkov te oblike ali pa si jih morda že vse postavil na ploščo.)
- **void** PostaviPloscek(**int** oblika, **int** x, **int** y, **bool** b) — če je b == **true**, ta funkcija položi plošček oblike oblika na ploščo tako, da najbolj levo polje v najbolj zgornji vrstici tega ploščka pokrije polje (x, y) na plošči. Če je b == **false**, pa funkcija ta plošček s tega položaja odstrani.

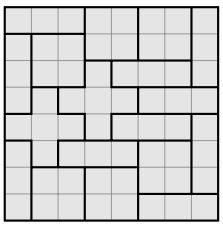
Funkcija PostaviPloscek prekine izvajanje tvojega programa, če zahtevaš od nje operacijo, ki je ni mogoče izvesti (npr. dodajanje ploščka neke oblike, če si vse razpoložljive ploščke te oblike že položil na mrežo; ali dodajanje ploščka tako, da bi se prekrival z že obstoječimi ali štrlel čez rob mreže; ali brisanje ploščka, ki ga v resnici ni tam).

Ploščkov se pri tej nalogi ne dá obračati ali vrteti in funkciji PreveriPloscek in PostaviPloscek tega tudi ne poskušata početi. To pomeni, da štejeta na primer  in za dve različni obliki in ploščkov ene oblike ne moremo zasukati in uporabiti kot ploščke druge oblike.

Primer. Recimo, da imamo ploščke naslednjih štirih oblik v naslednjih količinah:



Potem lahko ploščo pokrijemo takole:



Še deklaracije gornjih funkcij v drugih jezikih:

{ *V pascalu:* }
function StOblik: integer;
function StPlosckov(oblika: integer): integer;
function JePokrito(x, y: integer): boolean;
function PreveriPloscek(oblika, x, y: integer): boolean;
procedure PostaviPloscek(oblika, x, y: integer; b: boolean);

// *V javi: deklaracije so kot v besedilu naloge, le z* **boolean** *namesto* **bool**.

V pythonu:
def StOblik() –> int: ...
def StPlosckov(oblika: int) –> int: ...
def JePokrito(x: int, y: int) –> bool: ...
def PreveriPloscek(oblika: int, x: int, y: int) –> bool: ...
def PostaviPloscek(oblika: int, x: int, y: int, b: bool) –> None: ...