1. tekmovanje IJS v znanju računalništva

Institut Jožef Stefan, Ljubljana, 6. maja 2006

Bilten

Bilten 1. tekmovanja IJS v znanju računalništva
Institut Jožef Stefan, 2006

Uredil Janez Brank

Avtorji nalog: Gorazd Božič, Primož Gabrijelčič, Boris Gašperin, Uroš Jovanovič, Aleš Košir, Mark Martinec, Mojca Miklavec, Miha Vuk, Janez Brank.

Razdelek o tekmovanju programov je prispeval Blaž Novak.

Tisk: Present d. o. o.
Naklada: 300 izvodov

Ta bilten je dostopen tudi v elektronski obliki na domači strani tekmovanja:
`http://rtk.ijs.si/`

Vprašanja, pripombe, komentarji, popravki ipd. v zvezi z biltenom so dobrodošli. Pišite nam na naslov `rtk-info@ijs.si`.

CIP — Kataložni zapis o publikaciji
Narodna in univerzitetna knjižnica, Ljubljana

371.27:004(497.4)

TEKMOVANJE IJS v znanju računalništva (1 ; 2006 ; Ljubljana)
Bilten / 1. tekmovanje IJS v znanju računalništva, Ljubljana, 6. maja 2006 ; [avtorji nalog Gorazd Božič ... [et al.] ; uredil Janez Brank]. — Ljubljana : Institut Jožef Stefan, 2006

ISBN 961-6303-76-7
1. Brank, Janez, 1979–
226985216

KAZALO

PREDGOVOR

Srednješolska računalniška tekmovanja imajo v Sloveniji že dolgo tradicijo — okoli tri desetletja se dogajajo vsako leto spomladi tekmovanja raznih tipov. Sprva so bila to le pisna tekmovanja, ki so preverjala sposobnosti algoritmičnega razmišljanja, kasneje, v devetdesetih letih, pa smo jim dodali druge discipline, ki so bile in so nekatere še aktualne za trenutni razvoj računalništva. Kaj je rezultat vseh teh aktivnosti? Glavni rezultat zagotovo niso nagrade in naslovi, ki jih posamezniki ali šole dosežejo na teh tekmovanjih, ampak pridobljeno znanje in izkušnje, s katerimi se obogatijo posamezni tekmovalci in z njimi vred vsa slovenska računalniška skupnost. To vidimo predvsem preko tega, da večina uspešnejših tekmovalcev iz preteklih tekmovanj danes soustvarja hrbtenico slovenske računalniške industrije, mnogo pa jih na univerzah poučuje mlade in na znanstvenih inštitutih ustvarja nova znanja. Z drugimi besedami — le stežka bi v Sloveniji našli kakšno pomembnejše okolje, povezano z računalništvom, ne da bi se zaleteli v katerega od bivših tekmovalcev.

Tekmovanja pa so v vseh teh letih doživela marsikakšno preobrazbo — strokovno in organizacijsko. V trenutni preobleki tekmovanje, ki ga pripravlja komisija, sestavljena skoraj izključno iz bivših tekmovalcev, domuje na Institutu Jožef Stefan in se je formalno preoblikovalo v dogodek „Tekmovalni dnevi IJS“. Njegova ambicija je računalništvo gojiti predvsem kot tehnično in znanstveno panogo ter tako vzgajati in izobraževati računalniške entuziaste, ki želijo prerasti okvire običajnega znanja računalništva. V tem duhu smo tudi pripravili tekmovanje, ki ga predstavlja ta bilten — v naslednjih letih nameravamo s to dejavnostjo nadaljevati in jo prilagajati aktualnim tokovom.

Marko Grobelnik, Janez Brank

STRUKTURA TEKMOVANJA

Tekmovanje poteka v treh težavnostnih skupinah. Tekmovalec se lahko prijavi v katerokoli od teh treh skupin ne glede na to, kateri letnik srednje šole obiskuje. Prva skupina je najlažja in je namenjena predvsem tekmovalcem, ki se ukvarjajo s programiranjem šele nekaj mesecev ali mogoče kakšno leto. Druga skupina je malo težja in predpostavlja, da tekmovalci osnove programiranja že poznajo; primerna je za tiste, ki se učijo programirati kakšno leto ali dve. Tretja skupina je najtežja, saj od tekmovalcev pričakuje, da jim ni prevelik problem priti do dejansko pravilno delujočega programa; koristno je tudi, če vedo kaj malega o algoritmih in njihovem snovanju.

V lažjih dveh skupinah traja tekmovanje tri ure; tekmovalci rešujejo naloge na papir, nato pa njihove odgovore oceni temovalna komisija. Naloge v teh dveh skupinah večinoma zahtevajo, da tekmovalec opiše postopek ali pa napiše program ali podprogram, ki reši določen problem. Pri pisanju izvorne kode programov ali podprogramov načeloma ni posebnih omejitev glede tega, katere programske jezike smejo tekmovalci uporabljati.

V tretji skupini pa rešujejo tekmovalci naloge na računalnikih, za kar imajo pet ur časa. Pri vsaki nalogi je treba napisati program, ki prebere podatke iz vhodne datoteke, izračuna nek rezultat in ga izpiše v izhodno datoteko. Programe se potem ocenjuje tako, da se jih na ocenjevalnem računalniku izvede na več testnih primerih, število točk pa je sorazmerno s tem, pri koliko testnih primerih je izpisal pravilni rezultat. (Podrobnosti točkovanja v 3. skupini so opisane na strani 20.) Letos so bili v 3. skupini dovoljeni programski jeziki pascal, C, C++ in java.

Nekaj težavnosti tretje skupine izvira tudi od tega, da je pri njej mogoče dobiti točke le za delujoč program, ki vsaj nekaj testnih primerov reši pravilno; če imamo le pravo idejo, v delujoč program pa nam je ni uspelo preliti (npr. ker nismo znali razdelati vseh podrobnosti, odpraviti vseh napak, ali pa ker smo ga napisali le do polovice), ne bomo dobili pri tisti nalogi nič točk.

Tekmovalci vseh treh skupin si lahko pri reševanju pomagajo z zapiski in literaturo, v tretji skupini pa tudi z dokumentacijo raznih prevajalnikov in razvojnih orodij, ki so nameščena na tekmovalnih računalnikih.

Na začetku smo tekmovalcem razdelili tudi list z nekaj nasveti in navodili (str. 7–8 za 1. in 2. skupino, str. 19–21 za 3. skupino).

Omenimo še, da so rešitve, objavljene v tem biltenu, večinoma obsežnejše od tega, kar na tekmovanju pričakujemo od tekmovalcev, saj je namen tukajšnjih rešitev pogosto tudi pokazati več poti do rešitve naloge in bralcu omogočiti, da bi se lahko iz razlag ob rešitvah še česa novega naučil.

Poleg tekmovanja v znanju računalništva smo organizirali tudi tekmovanje programov; to je podrobneje predstavljeno na straneh 68–69.

NASVETI ZA 1. IN 2. SKUPINO

Nekatere naloge so tipa **napiši program** (ali **napiši podprogram**), nekatere pa tipa **opiši postopek**. Pri slednjih ti ni treba pisati programa ali podprograma v kakšnem konkretnem programskem jeziku, ampak lahko postopek opišeš tudi kako drugače: z besedami (v naravnem jeziku), psevdokodo (glej spodaj), diagramom poteka, itd. Glavno je, da je tvoj opis dovolj natančen, jasen in razumljiv, tako da je iz njega razvidno, da si dejansko našel in razumel pot do rešitve naloge.

Psevdokodi pravijo včasih tudi strukturirani naravni jezik. Postopek opišemo v naravnem jeziku, vendar opis strukturiramo na podoben način kot pri programskih jezikih, tako da se jasno vidi strukturo zank, vejitev in drugih programskih elementov.

Primer opisa postopka v psevdokodi: recimo, da imamo zaporedje besed in bi ga radi razbili na več vrstic tako, da ne bo nobena vrstica preširoka.

```
naj bo trenutna vrstica prazen niz;
pregleduj besede po vrsti od prve do zadnje:
    če bi trenutna vrstica z dodano trenutno besedo (in presledkom
    pred njo) postala predolga,
        izpiši trenutno vrstico in jo potem postavi na prazen niz;
    dodaj trenutno besedo na konec trenutne vrstice;
če trenutna vrstica ni prazen niz, jo izpiši;
```

Če pa v okviru neke rešitve pišeš izvorno kodo programa ali podprograma, obvezno poleg te izvorne kode v nekaj stavkih opiši, kako deluje (oz. naj bi delovala) 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.

Če naloga zahteva branje ali obdelavo kakšnih 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(i, ' vrstic, ', d, ' znakov.');
end. {BranjeVrstic}
```

```
#include <stdio.h>
#include <string.h>
int main() {
  char s[101]; int i = 0, d = 0;
  while (gets(s)) {
    i++; d += strlen(s);
    printf("%d. vrstica: \"%s\"\n", i, s);
  }
  printf("%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. Funkciji gets se je v praksi bolje izogibati, ker pri njej nimamo zaščite pred primeri, ko je vrstica daljša od naše tabele s. Namesto gets bi bilo bolje uporabiti fgets; vendar pa za rešitev naših tekmovalnih nalog v prvi skupini zadošča tudi gets.

- 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
  i := 0;
  while not Eof do begin
    while not Eoln do
      begin Read(c); Write(c); i := i + 1 end;
    if not Eof then begin ReadLn; WriteLn end;
  end; {while}
  WriteLn('Skupaj ', i, ' znakov.');
end. {BranjeZnakov}
```

```
#include <stdio.h>

int main() {
  int i = 0, c;
  while ((c = getchar()) != EOF) {
    putchar(c); if (i != '\n') i++;
  }
  printf("Skupaj %d znakov.\n", i);
  return 0;
}
```

NALOGE ZA PRVO SKUPINO

Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši, na kakšni ideji temelji tvoja rešitev.

1. Odstavki

V nekem besedilu so odstavki ločeni s praznimi vrsticami. **Napiši program**, ki prebira besedilo s standardnega vhoda in ga izpisuje na standardni izhod, pri tem pa, če se kdaj pojavi več zaporednih praznih vrstic, takšno skupino praznih vrstic nadomesti z eno samo prazno vrstico. (Za prazne vrstice štejemo pri tej nalogi le tiste vrstice, ki res ne vsebujejo nobenega znaka, niti presledkov. Predpostaviš lahko, da ni nobena vrstica daljša od sto znakov.)[1]

2. Sneg

Letošnja zima je bila radodarna s snegom. Da bi opazovali časovno spreminjanje količine zapadlega snega, lahko za vsak dan posebej merimo, koliko snega je na novo zapadlo tisti dan in koliko se ga je stalilo (ali pa se je snežna odeja stanjšala zaradi sesedanja). Iz razlike med tema dvema količinama lahko ugotovimo, za koliko se je povečala ali zmanjšala debelina snežne odeje.

Od toplih jesenskih dni naprej vsak dan spremljamo dve meritivi:

- Debelino na novo zapadlega snega na ta dan (v milimetrih).
- Znižanje debeline snežne odeje na ta dan zaradi taljenja in sesedanja (v milimetrih).

Napiši program, ki s standardnega vhoda prebira podatke za 365 zaporednih dni. Za vsak dan dobi vrstico z dvema številoma: prvo je debelina novozapadlega snega, drugo pa znižanje snežne odeje zaradi taljenja in sesedanja. Za vsak dan naj program na standardni izhod izpiše debelino snežne odeje na koncu tega dneva. Predpostavi, da ob začetku merjenja še ni snega in da so vhodni podatki taki, kot bi se res lahko zgodili (da iz njih npr. ne sledi, da je bila debelina snežne odeje kdaj manjša od 0 mm).

Primer: če bi se podatki na vhodu začeli z vrsticami

```
0 0
12 3
14 2
2 10
0 13
```

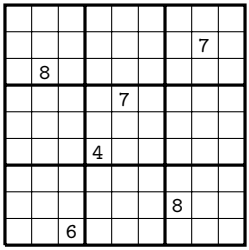
bi se moral izpis programa začeti z vrsticami

[1] V gornjem besedilu (ki je tako, kot smo ga na tekmovanju razdelili tekmovalcem) smo pozabili eksplicitno povedati, koliko besedila naj program prebere. Mišljeno je, naj prebere celotno vsebino standardnega vhoda, torej vse do EOF.

```
0
9
21
13
0
```

3. Sudoku

Sudoku je številčna križanka. Igralno polje velikosti 9×9 kvadratkov je dodatno razdeljeno na devet manjših kvadratov velikosti 3×3 (na spodnji sliki so predstavljeni z debelejšimi črtami), vanj pa je vpisanih nekaj števil (od 1 do 9). Primer:



V igralno polje igralec vpisuje števila od 1 do 9 in to tako, da so na koncu izpolnjeni naslednji trije pogoji: (1) v vsakem stolpcu se mora vsako število od 1 do 9 pojavljati natanko enkrat; (2) v vsaki vrstici se mora vsako število od 1 do 9 pojavljati natanko enkrat; (3) v vsakem od devetih malih kvadratov velikosti 3×3 se mora vsako število od 1 do 9 pojavljati natanko enkrat.

Na spodnji sliki levo vidimo primer pravilno izpolnjenega polja. Desno polje pa je izpolnjeno napačno (med drugim zato, ker v zadnjem stolpcu ni števila 6, se pa število 2 v njem pojavlja kar dvakrat).

7	6	9	3	1	4	5	8	2
1	4	2	6	5	8	3	7	9
3	8	5	7	2	9	1	6	4
6	9	3	1	7	5	2	4	8
8	1	4	2	9	3	6	5	7
5	2	7	4	8	6	9	3	1
2	7	8	5	6	1	4	9	3
4	5	1	9	3	7	8	2	6
9	3	6	8	4	2	7	1	5

7	6	9	3	1	4	5	8	2
1	4	2	6	5	8	3	7	9
3	8	5	7	2	9	1	6	4
6	9	3	1	7	5	2	4	8
8	1	4	2	9	3	6	5	7
5	2	7	4	8	6	9	3	1
2	7	8	5	6	1	4	9	3
4	5	1	9	3	7	8	6	2
9	3	6	8	4	2	7	1	5

Napiši program, ki bo sprejel izpolnjeno polje in izpisal niz `PRAVILNA`, če je rešitev pravilna, in `NAPACNA`, če ni pravilna. Če ti je naloga pretežka, lahko poskusiš napisati program, ki bo preverjal le pogoja (1) in (2), ne pa tudi pogoja (3) (z drugimi besedami: preveri naj, če so prisotna vsa števila od 1 do 9 v vsaki vrstici in v vsakem stolpcu, ni pa se mu treba ukvarjati z malimi kvadrati velikosti 3×3). Za takšno rešitev dobiš pri tej nalogi polovico vseh možnih točk.

Igralno polje je deklarirano kot dvodimenzionalna tabela. Primer deklaracije v programskem jeziku pascal:

```
type SudokuPoljeT = array [1..9, 1..9] of 1..9;
```

Predpostavi, da že obstaja nek podprogram Preberi, ki ga lahko pokličeš, da ti bo prebral izpolnjeno polje iz neke vhodne datoteke:

```
procedure Preberi(var T: SudokuPoljeT);
```

Še deklaraciji v C/C++:

```
typedef int SudokuPoljeT[9][9];
void Preberi(SudokuPoljeT T);
```

4. Naraščajoče besede

V nekaterih besedah so črke že urejene naraščajoče po abecedi: vsaka črka take besede pride v abecedi kasneje kot prejšnja črka te besede. Takšnim besedam pravimo *naraščajoče besede*. Primer naraščajoče besede je `AGILNOST` — `G` je v abecedi kasneje kot `A`, `I` je kasneje kot `G` in tako naprej.

Napiši program, ki prebere zaporedje besed s standardnega vhoda in na koncu izpiše najdaljšo naraščajočo besedo v njem.[2] (Če je najdaljših več enako dolgih naraščajočih besed, je vseeno, katero izmed njih izpiše.) Predpostaviš lahko, da je vsaka beseda v svoji vrstici, v besedah nastopajo samo velike črke angleške abecede in nobena beseda ni daljša od 100 znakov.

5. Podnapisi

Nek predvajalnik filmov bi radi dopolnili tako, da bi znal prikazovati tudi podnapise. Te imamo podane v samostojnih datotekah, ločeno od filma, tako da lahko k istemu filmu pritaknemo podnapise v različnih jezikih. Ob predvajanju je treba, tik preden se prikaže posamezna sličica filma, ugotoviti, kateri podnapis pripada tej sličici (če sploh kakšen). Tvoja naloga je **napisati podprogram** PodnapisZaSlicico, ki ga bo sistem poklical pred prikazom vsake sličice, tvoj podprogram pa bo vrnil podnapis, ki ga je treba prikazati na tej sličici (oz. prazen niz, če ni treba prikazati nobenega podnapisa):

```
function PodnapisZaSlicico(StevilkaSlicice: integer): string;
```

Sličice filma so oštevilčene z zaporednimi celimi števili od 1 naprej. Če bi rad izvedel kakšne operacije še pred prvim klicem podprograma PodnapisZaSlicico, lahko napišeš tudi podprogram Inicializacija, ki naj bo takšne oblike:

```
procedure Inicializacija;
```

Sistem ga bo poklical na začetku predvajanja filma (ko so že znani podnapisi za celoten film, vendar pred prikazom prve sličice). V tem podprogramu lahko na primer inicializiraš svoje globalne spremenljivke, če jih boš uporabljal (v tem primeru seveda tudi ne pozabi navesti deklaracij teh spremenljivk).

Predpostavi, da so ti na voljo naslednji podprogrami, s katerimi dobiš od sistema podatke o podnapisih:

[2] Podobno kot pri nalogi 1.1 tudi tu mogoče ni dovolj jasno, kdaj naj program neha brati vhodno zaporedje. Mišljeno je, naj prebere vse besede, kar jih lahko dobi s standardnega vhoda, torej vse do EOF.

- **function** SteviloPodnapisov: integer;
  Vrne število podnapisov v celem filmu.

- **function** PrvaSlicica(StPodnapisa: integer): integer;
  function ZadnjaSlicica(StPodnapisa: integer): integer;
  function Vsebina(StPodnapisa: integer): string;
  Vrnejo podatke o podnapisu s številko StPodnapisa: številko prve in zadnje sličice filma, na katerih naj se vidi ta podnapis, in besedilo podnapisa. Podnapisi so oštevilčeni od 1 do SteviloPodnapisov v takšnem vrstnem redu, v kakršnem naj bi se prikazovali v filmu. Predpostaviš lahko, da se intervali PrvaSlicica..ZadnjaSlicica različnih podnapisov med sabo ne prekrivajo (torej nobeni sličici ne pripada več kot en podnapis).

Še deklaracije v C/C++:

```
int SteviloPodnapisov();
int PrvaSlicica(int StPodnapisa);
int ZadnjaSlicica(int StPodnapisa);
const char* Vsebina(int StPodnapisa);

void Inicializacija();
const char* PodnapisZaSlicico(int StevilkaSlicice);
```

(Opomba: „**const**“ v gornjih deklaracijah pomeni, da funkcija vrača kazalec na niz znakov, ki ga klicatelj ne sme spreminjati. Če te ta **const** kaj mede, se delaj, kot da ga ni.)

Zaželeno je, da je tvoja rešitev čim hitrejša, predvsem pa ne sme biti preveč potratna s pomnilnikom. Predpostavi, da predvajalnik nima dovolj pomnilnika, da bi si lahko npr. privoščil tabelo, v kateri bi bil po en celoštevilski element za vsako sličico filma (lahko pa bi imeli nekaj takšnih elementov za vsak podnapis v filmu — podnapisov je vendarle večdesetkrat manj kot sličic). (Rešitve, ki porabijo več pomnilnika, bodo dobile malo manj točk.)

NALOGE ZA DRUGO SKUPINO

Svoje odgovore dobro utemelji. Če pišeš izvorno kodo programa ali podprograma, **OBVEZNO** tudi v nekaj stavkih z besedami opiši, na kakšni ideji temelji tvoja rešitev.

1. 1337ovščina

Med mladimi uporabniki komunikacijskih storitev (IRC, igrice, SMS) sproti nastaja žargon in tudi poseben način pisanja, ki služi več namenom: prikrivanju vsebine pred neposvečenimi tujci, poudarjanju drugačnosti kroga posvečenih uporabnikov (elitizem), hitremu tipkanju ipd.

Eden od načinov pisanja je zamenjava nekaterih črk (naključno, ne vseh in ne vedno) z enim od nadomestnih nizov, ki so videti podobno. Nadomestek je lahko en znak, lahko pa je sestavljen tudi iz več znakov. Tule je nekaj primerov možnih zamenjav:

```
a  →  4  /\  @  ^
b  →  8  6  13  )3
c  →  (  <
d  →  )  [)
e  →  3  &  [-
f  →  ph
g  →  6  9  &  gee
h  →  #  )-(
i  →  1  !  |
.  .  .  .  .  .  .  .  .
```

Napiši program, ki bo prepisal neko besedilo s standardnega vhoda na standardni izhod tako, da bo v njem naključno zamenjal nekatere (ne nujno vseh) znake z eno od možnih zamenjav za ta znak. Predpostaviš lahko, da že obstaja funkcija Nakljucje, ki jo lahko uporabiš kot vir naključnih števil:

```
function Nakljucje(n: integer): integer;
```

Parameter `n` mora biti večji od 0, funkcija pa ob vsakem klicu vrne neko naključno izbrano celo število, večje ali enako 1 in manjše ali enako `n`.

Tabela možnih preslikav je podana podobno, kot je prikazano zgoraj. Da ti prihranimo branje in organizacijo preslikovalne tabele, si lahko pomagaš s funkcijo MozneZamenjave, za katero predpostavi, da že obstaja:

```
function MozneZamenjave(c: char): string;
```

Ta funkcija za podani znak `c` vrne niz, v katerem so vsebovana vsa možna nadomestila tega znaka; tako na primer za znak `'b'` dobimo niz `'8 6 13 )3'`. Pri tem so alternative med seboj ločene z natanko enim presledkom. Če za nek znak ni določenega nobenega nadomestila, nam funkcija MozneZamenjave pri tem znaku vrne prazen niz, kar le pomeni, da se takega znaka ne da zamenjati.

Primer rezultata (ob uporabi dopolnjene tabele iz primera):

```
Ta13e7a m0z|\|i# presl1|</\v je podana po)0bno, k0+ je pr1k@z4n0 29024j.
D@ ti |"ri#24n!mo b2anje i|\| o/2ga|\|!z@(1jo pr3sl!Xov/\lne 746313, $1
la#ko |"omag@$ s f00n|<<!j0 Mo2|\|3Z@AAenj/\v&,
```

Še deklaraciji v C/C++:

```
int Nakljucje(int n);
const char* MozneZamenjave(char c);
```

2. Predpone

Telefonski promet poteka prek telefonskih central in vse telefonske številke, za katere skrbi neka centrala, imajo skupnih prvih nekaj števk — to je torej *predpona* (prefiks), ki pripada tisti centrali.

Recimo, da imaš klicni center, na katerem se vsak dohodni klic zabeleži s polno številko klicatelja. Rad bi opravil pregled klicev po telefonskih centralah klicateljev, zato si od telefonskih operaterjev prejel podatke o številčnih predponah in pripadajoči centrali. Primer (podatki so izmišljeni!):

01212	Trzin
012125	Domžale
01213	Mengeš
0121342	Kamnik
. . . . .	. . .

Kot je razvidno iz zgornjega primera, so lahko daljše predpone določene svojim lastnim centralam, čeprav je krajši del že dodeljen neki drugi. Za vsako telefonsko številko, ki jo je zabeležil tvoj klicni center, moraš torej najti najdaljšo ujemajočo predpono, da lahko določiš telefonsko centralo klicatelja. (Ni pa se ti treba ukvarjati z določanjem imena centrale.)

Predpostavi, da obstajata naslednji funkciji, prek katerih lahko prideš do vseh predpon:

```
function StPredpon: integer; { Vrne število predpon. }
function PovejPredpono(ZapSt: integer): string;
  { Vrne predpono z zaporedno številko ZapSt, ki mora biti od 1 do StPredpon. }
```

Opiši postopek, ki bo za dano telefonsko številko poiskal in vrnil najdaljšo predpono, ki se ujema z začetkom te telefonske številke. Računaj na to, da bo moral tvoj postopek hitro najti najdaljšo predpono za veliko različnih telefonskih številk (recimo: na tisoče predpon, na milijone telefonskih številk). Zato si lahko poskusiš pred prvim izvajanjem tvojega postopka podatke preurediti in organizirati tako, da bo potem tvoj postopek za iskanje najdaljše predpone deloval čim hitreje. Če se odločiš za to možnost, opiši tudi svoje globalne spremenljivke, ki bi jih v ta namen uporabil, in opiši postopek za inicializacijo teh spremenljivk (ta postopek za inicializacijo bi poklical sistem še pred prvim klicem postopka za iskanje najdaljše predpone).

Če ti je to kaj v pomoč, si lahko postopek za iskanje najdaljše predpone predstavljaš kot podprogram takšne oblike:

```
function NajdaljsaPredpona(TelSt: string): string;
```

Postopek za inicializacijo pa kot podprogram takšne oblike:

```
procedure Inicializacija;
```

Vendar pa ni nujno, da svoja dva postopka zapišeš kot podprograma; lahko ju opišeš tudi kako drugače (v naravnem jeziku, s psevdokodo, ipd.), samo da bo opis dovolj natančen in čim bolj jasen in razumljiv.

Še deklaracije v C/C++:

```
int StPredpon();
const char* PovejPredpono(int ZapSt);
const char* NajdaljsaPredpona(const char* TelSt);
void Inicializacija();
```

3. Cestne lučke

Cestno podjetje ima lučke, ki jih postavlja ob delih na cesti, da zavaruje mesto del, da ne bi prišlo do nesreč. Lučke lahko utripajo ali pa potujejo levo ali desno, v odvisnosti od postavitve na cesti.

Cestni delavec ima na kontrolni napravi na voljo štiri gumbe, na katerih piše: „Ugasni“, „Utripaj“, „V levo“ in „V desno“.

Napiši štiri **podprograme** (Ugasni, Utripaj, VLevo in VDesno), ki skrbijo za to, da se lučke obnašajo tako, kot zahteva posamezni izmed teh štirih gumbov. Sistem bo ob pritisku na posamezni gumb poklical ustreznega od tvojih štirih podprogramov. Predpostaviš lahko, da sta na voljo naslednja sistemska podprograma:

Čas po pritisku	Ugasni	Utripaj	V levo	V desno
0 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ☼ ⊜ ⊜ ☼	☼ ⊜ ☼ ☼ ⊜	☼ ⊜ ☼ ☼ ⊜
0,5 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ☼ ☼ ⊜ ☼	⊜ ☼ ⊜ ☼ ☼
1 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ☼ ⊜ ⊜ ☼	☼ ☼ ⊜ ☼ ⊜	☼ ⊜ ☼ ⊜ ☼
1,5 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ⊜ ⊜ ⊜ ⊜	☼ ⊜ ☼ ⊜ ☼	☼ ☼ ⊜ ☼ ⊜
2 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ☼ ⊜ ⊜ ☼	⊜ ☼ ⊜ ☼ ☼	⊜ ☼ ☼ ⊜ ☼
2,5 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ⊜ ⊜ ⊜ ⊜	☼ ⊜ ☼ ☼ ⊜	☼ ⊜ ☼ ☼ ⊜
3 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ☼ ⊜ ⊜ ☼	⊜ ☼ ☼ ⊜ ☼	⊜ ☼ ⊜ ☼ ☼
3,5 s	⊜ ⊜ ⊜ ⊜ ⊜	⊜ ⊜ ⊜ ⊜ ⊜	☼ ☼ ⊜ ☼ ⊜	☼ ⊜ ☼ ⊜ ☼

Ilustracija k nalogi 2.3. Recimo, da imamo pet lučk in je POTUJE $= 22_{10} = 10110_2$ in UTRIPA $= 9_{10} = 01001_2$. Gornja slika prikazuje, kakšno naj bi bilo stanje lučk v prvih osmih korakih (štirih sekundah) po pritisku na posameznega od gumbov.

- **procedure** Nastavi(BitnaSlika: integer) — nastavi stanje vseh lučk. Lučke so oštevilčene od 0 naprej (0 je najbolj desna lučka v vrsti, 1 je druga z desne strani itd.); lučka i se prižge, če je bit i v številu BitnaSlika enak 1, sicer pa se ugasne. (Predpostaviš lahko, da je lučk manj, kot je bitov v številu tipa integer.)

- **procedure** Pocakaj — počaka pol sekunde. Približno toliko časa naj mine med pomiki lučk pri gumbih „V levo“ in „V desno“ in med vklopom in izklopom lučk pri gumbu „Utripaj“.

Predpostavi, da se ob pritisku na gumbe na kontrolni napravi pokliče neka zunanja procedura, ki v trenutku prekine izvajanje trenutnega od tvojih štirih podprogramov (če se kateri od njih trenutno izvaja) in potem pokliče tistega, ki ustreza pravkar pritisnjenemu gumbu.

Predpostavi tudi, da so definirane tri celoštevilske konstante:

- SteviloLuck pove število lučk v sistemu; oštevilčene so od 0 (najbolj desna) do SteviloLuck − 1 (najbolj leva);

- POTUJE je vzorec bitov (celo število), ki pove, v kakšnem vzorcu naj bodo prižgane in ugasnjene lučke pri potovanju v levo/desno (ta vzorec lučk se v vsakem koraku ciklično zamakne za eno mesto v levo/desno; glej tudi spodnji primer);

- UTRIPA je vzorec bitov (celo število), ki pove, katere lučke naj utripajo pri utripajočih lučkah (ostale naj bodo ugasnjene).

Pomen bitov v konstantah POTUJE in UTRIPA je enak kot zgoraj pri parametru BitnaSlika podprograma Nastavi; vsi biti, ki ne sodijo med najnižjih SteviloLuck bitov, pa so ugasnjeni.

Še deklaracije v C/C++:

```
const int SteviloLuck = ..., POTUJE = ..., UTRIPA = ...;
void Nastavi(int BitnaSlika);
void Pocakaj();
```

4. Drevo končnic

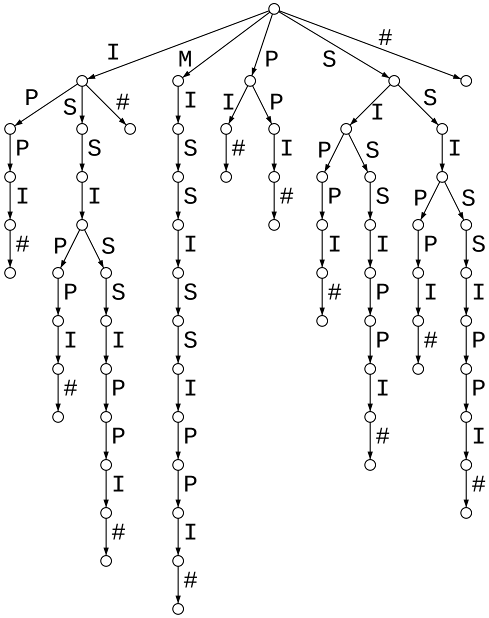
Človeški genom je sestavljen iz dolgega zaporedja nukleinskih kislin: adenina (A), gvanina (G), timina (T) in citozina (C). V genetiki nas pogosto zanimajo zaporedja, ki se v genomu ponavljajo na več mestih, ali pa taka, ki se pojavljajo pri več različnih organizmih hkrati.

Zaporedje nukleinskih kislin lahko predstavimo z nizom črk A, C, T in G; včasih pa je koristno takšen niz predstaviti tudi s pomočjo *drevesa končnic* (suffix tree).

Končnice (sufiksi) niza $s = s_1 s_2 \ldots s_n$ so nizi $s_i s_{i+1} \ldots s_n$ za $i = 1, 2, \ldots, n$. Drevo končnic niza s zgradimo takole. Najprej na konec niza s dodamo nek poseben znak za konec niza, ki se drugače ne pojavlja nikjer v nizu (pri tej nalogi bomo uporabljali znak „#“). Potem za vsako končnico tako podaljšanega niza, torej za vsak niz $s_i s_{i+1} \ldots s_n$# in še za niz # ustvarimo od korena drevesa navzdol zaporedje povezav z oznakami s_i, s_{i+1}, s_{i+2} in tako naprej, vse do s_n in nazadnje #. Če

ima več končnic prvih nekaj črk skupnih, uporabljajo od začetka tudi iste povezave in vozlišča. Tako gre iz vsakega vozlišča naprej največ po ena povezava za vsako črko abecede (v abecedo štejemo zdaj tudi posebni znak `#`). Povezave do poddreves imamo vedno urejene po abecedi (mislimo si, da pride `#` na konec abecede).

Takšno drevo lahko zgradimo za poljuben niz s, ne le za nize iz črk `A`, `C`, `T` in `G`. Naslednja slika prikazuje na primer drevo končnic niza `MISSISSIPPI`.



Ilustracija k nalogi 2.4: drevo končnic niza `MISSISSIPPI`.

(a) Po zgornjem zgledu **nariši** drevo končnic za besedo `RABARBARA`.

(b) V genomiki se pogosto srečujemo s kratkimi ponavljajočimi se nukleotidnimi zaporedji. V zaporedju `CATCATCATGGCATTCAT` se na primer podzaporedje `CAT` pojavi petkrat; v zaporedju `CGCGCGCGCGCTTTCGCGC` se podzaporedje `CGC` pojavi šestkrat, podzaporedje `CGCG` petkrat in tako naprej. Čeprav takšna zaporedja navadno ne kodirajo proteinov, imajo za organizem druge pomembne funkcije. **Opiši postopek**, ki ti za podano drevo končnic poljubnega niza in za dani števili m in k izpiše vsa (strnjena) podzaporedja dolžine m ali več, ki se v originalnem nizu pojavijo vsaj k-krat. (Primer: za niz `MISSISSIPPI` in $m = k = 2$ bi dobili podnize `IS`, `SI`, `SS`, `ISS`, `SSI` in `ISSI`.)

(*Opomba:* podvprašanje (a) je vredno pri tej nalogi tretjino točk, podvprašanje (b) pa dve tretjini.)

5. Razvajeni zvezdniki

Najel te je producent novega reality showa, ki bi rad na samotni otok za nekaj tednov poslal skupino tretjerazrednih zvezdnikov, nato pa snemal, kako bodo shajali drug z drugim. Pripravil si je seznam potencialnih udeležencev in jih oštevilčil od 1 do n. Težava je v tem, da so zvezdniki zelo muhasti in so mu postavili kup pogojev. Vsak zvezdnik je pripravil dva seznama in pravi, da bo sodeloval le, če bodo v oddaji sodelovali tudi vsi zvezdniki z njegovega prvega seznama in nobeden od zvezdnikov z njegovega drugega seznama. (Možno je tudi, da je kateri od teh seznamov prazen; lahko sta prazna celo oba.) Na samotni otok bomo poslali le prvih nekaj zvezdnikov s producentovega spiska, torej zvezdnike $\{1, 2, \ldots, k\}$ za nek k. **Opiši postopek**, ki ugotovi, kateri je največji k, za katerega lahko pošljemo na otok zvezdnike $\{1, 2, \ldots, k\}$ (ne pošljemo pa nobenega izmed zvezdnikov $\{k+1, k+2, \ldots, n\}$), ne da bi prekršili katero od omejitev, ki so jih ti zvezdniki postavili.

PRAVILA TEKMOVANJA ZA TRETJO SKUPINO

Vsaka naloga zahteva, da napišeš program, ki prebere neke vhodne podatke, izračuna odgovor oz. rezultat ter ga izpiše v izhodno datoteko. Programi naj berejo vhodne podatke iz datoteke *imenaloge*.`in` in izpisujejo svoje rezultate v *imenaloge*.`out`. Natančni imeni datotek sta podani pri opisu vsake naloge. V vhodni datoteki je vedno po en sam testni primer. Vaše programe bomo pognali po večkrat, vsakič na drugem testnem primeru. Besedilo vsake naloge natančno določa obliko (format) vhodnih in izhodnih datotek. Tvoji programi lahko predpostavijo, da se naši testni primeri ujemajo s pravili za obliko vhodnih datotek, ti pa moraš zagotoviti, da se bo izpis tvojega programa ujemal s pravili za obliko izhodnih datotek.

Delovno okolje

Na začetku boš dobil mapo s svojim uporabniškim imenom ter navodili, ki jih pravkar prebiraš. Ko boš sedel pred računalnik, boš dobil nadaljnja navodila za prijavo v sistem.

Na vsakem računalniku imaš na voljo enoto (disk) `U:`, na kateri lahko kreiraš svoje datoteke (datoteke, ki so tam že od prej, pusti pri miru). Programi naj bodo napisani v programskem jeziku Pascal, C, C++ ali Java, mi pa jih bomo preverili z 32-bitnimi prevajalniki FreePascal, GNU C/C++ in GCJ. Za delo lahko uporabiš `FP` oz. `ppc386` (FreePascal), `GCC`/`G++` (GNU C/C++ — command line compiler), `GCJ`, `GPC` in Java 2 SDK.

Oglej si tudi spletno stran: `http://rtk/`, kjer boš dobil nekaj testnih primerov in program `RTK.EXE`, ki ga lahko uporabiš za preverjanje svojih rešitev. Tukaj si lahko tudi ogledaš anonimizirane rezultate ostalih tekmovalcev.

Preden boš oddal prvo rešitev, boš moral programu za preverjanje nalog sporočiti svoje ime, kar bi na primer Janez Novak storil z ukazom

```
rtk -name JNovak
```

(prva črka imena in priimek, brez presledka, brez šumnikov).

Za oddajo rešitve uporabi enega od naslednjih ukazov:

```
rtk imenaloge.pas
rtk imenaloge.c
rtk imenaloge.cpp
rtk ImeNaloge.java
```

Program `rtk` bo prenesel izvorno kodo tvojega programa na testni računalnik, kjer se bo prevedla in pognala na desetih testnih primerih. Na spletni strani boš dobil za vsak testni primer obvestilo o tem, ali je program pri njem odgovoril pravilno ali ne. Če se bo tvoj program s kakšnim testnim primerom ukvarjal več kot deset sekund, ga bomo prekinili in to šteli kot napačen odgovor pri tem testnem primeru.

Da se zmanjša možnost zapletov pri prevajanju, ti priporočamo, da ne spreminjaš privzetih nastavitev svojega prevajalnika. Tvoji programi naj uporabljajo le standardne knjižnice svojega programskega jezika in naj ne delajo z datotekami na disku, razen s predpisano vhodno in izhodno datoteko. Dovoljena je uporaba literature (papirnate), ne pa računalniško berljivih pripomočkov (razen tega, kar je že

na voljo na tekmovalnem računalniku), prenosnih računalnikov, prenosnih telefonov itd.

Ocenjevanje

Vsaka naloga lahko prinese tekmovalcu od 0 do 100 točk. Vsak oddani program se preizkusi na desetih testnih primerih; pri vsakem od njih dobi od 0 do 10 točk (praviloma 10, če je izpisal popolnoma pravilen odgovor, sicer pa 0; izjemi sta 1. in 5. naloga, kjer dobijo boljše rešitve več točk kot slabše), nato pa se te točke po vseh testnih primerih seštejejo v skupno število točk tega programa. Če si oddal N programov za to nalogo in je najboljši med njimi dobil M (od 100) točk, dobiš pri tej nalogi $\max\{0, M - 3(N-1)\}$ točk. Z drugimi besedami: za vsako oddajo (razen prve) pri tej nalogi se ti odbijejo tri točke. Pri tem pa ti nobena naloga ne more prinesti negativnega števila točk. Če nisi pri nalogi oddal nobenega programa, ti ne prinese nobenih točk. Če se poslana izvorna koda ne prevede uspešno, to ne šteje kot oddaja.

Skupno število točk tekmovalca je vsota po vseh nalogah. Tekmovalce razvrstimo po skupnem številu točk.

Vsak tekmovalec se mora sam zase odločiti o tem, katerim nalogam bo posvetil svoj čas, v kakšnem vrstnem redu jih bo reševal in podobno. Verjetno je priporočljivo najprej reševati lažje naloge.

Poskusna naloga (ne šteje k tekmovanju) (`poskus.in`, `poskus.out`)

Napiši program, ki iz vhodne datoteke prebere eno celo število (le-to je v prvi vrstici, okoli njega ni nobenih dodatnih presledkov ipd.) in izpiše njegov desetkratnik v izhodno datoteko.

Primer vhodne datoteke:

```
123
```

Ustrezna izhodna datoteka:

```
1230
```

Primer rešitve:

```
program PoskusnaNaloga;
var T: text; i: integer;
begin
  Assign(T, 'poskus.in'); Reset(T); ReadLn(T, i); Close(T);
  Assign(T, 'poskus.out'); Rewrite(T); WriteLn(T, 10 * i); Close(T);
end.
```

```
#include <stdio.h>
int main() {
  FILE *f = fopen("poskus.in", "rt");
  int i; fscanf(f, "%d", &i); fclose(f);
  f = fopen("poskus.out", "wt"); fprintf(f, "%d\n", 10 * i);
  fclose(f); return 0;
}
```

```
#include <fstream>
int main() {
   std::ifstream ifs("poskus.in"); int i; ifs >> i;
   std::ofstream ofs("poskus.out"); ofs << 10 * i;
   return 0;
}
```

```
import java.io.*;
public class Poskus {
   public static void main (String[] args) {
      try {
         StreamTokenizer st = new StreamTokenizer(new FileReader("poskus.in"));
         st.nextToken(); int i = (int) st.nval;
         PrintWriter os = new PrintWriter(new FileOutputStream("poskus.out"));
         os.println(10 * i); os.close();
      } catch (Exception e) { }
   }
}
```

NALOGE ZA TRETJO SKUPINO

1. Optična miška (`miska.in`, `miska.out`)

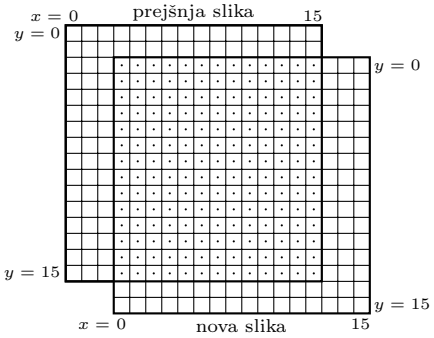
Optične računalniške miške, ki ne potrebujejo posebne podlage, vsebujejo preprosto, a hitro videokamero, ki si stalno ogleduje del podlage pod miško, vgrajeni procesor pa zaporedne slike primerja med seboj. Glede na to, da uporabnik premika miško dokaj počasi in neskokovito glede na kratek čas med zaporednima slikama (pod tisočinko sekunde), sta si zaporedni sliki precej podobni, naslednja slika je kvečjemu nekoliko premaknjena glede na prejšnjo (sukanje lahko zanemarimo, dviga miške s podlage pri tej nalogi ne bomo predvideli).

Vsaka slika je sestavljena iz kvadratne mreže $n \times n$ slikovnih elementov (pikslov). Ker zajema kamera le sivinske slike, je vsak slikovni element predstavljen s svetlostjo, ki je celo število med 0 in 63. Točka s koordinatama $(0, 0)$ je v zgornjem levem vogalu slike. Predpostavimo, da so premiki miške med dvema zaporednima slikama veliki kvečjemu tri piksle v vsaki smeri ($-3 \leq dx \leq 3$, $-3 \leq dy \leq 3$).

S primerjanjem dveh zaporednih slik bi radi ocenili, kakšen je najverjetnejši premik miške v času med trenutkoma, ko sta bili sliki posneti.

Nek možni premik (dx, dy) ocenimo takole: za vsak slikovni element, ki bi bil pri tem premiku viden tako na prejšnji kot na trenutni sliki, izračunamo absolutno vrednost razlike v svetlosti te točke na prejšnji in na trenutni sliki. Nato izračunamo povprečje teh absolutnih vrednosti po vseh slikovnih elementih, ki so vidni tako na prejšnji kot na trenutni sliki. Manjše ko je to povprečje, bolj verjetno je, da je (dx, dy) res pravi premik.

Na primer: če delamo s slikami velikosti 16×16 in se miška premakne za $dx = 3$, $dy = 2$ (glej spodnjo sliko), je prejšnji in trenutni sliki skupnih $13 \cdot 14$ slikovnih elementov in po njih bi računali povprečje iz gornjega odstavka.



Napiši program, ki bo prebral nekaj parov zaporednih slik iz vhodne datoteke in za vsak par izpisal najverjetnejši premik miške v horizontalni in vertikalni smeri.

Vhodna datoteka: v prvi vrstici je celo število m ($0 < m \leq 30$), ki pove, da v tem testnem primeru nastopa m parov slik. Sledijo podatki o posameznih parih slik, pred vsakim od njih pa je prazna vrstica.

Za vsak par slik je v vhodni datoteki najprej vrstica, ki vsebuje celo število n ($4 \leq n \leq 64$), ki pove, da sta sliki v tem paru veliki $n \times n$ slikovnih elementov. Sledi prazna vrstica, nato pa n vrstic, ki opisujejo sliko pred premikom (od zgoraj navzdol: prva vrstica je za $y = 0$, zadnja za $y = n - 1$); v vsaki od teh vrstic je n celih števil (med vključno 0 in vključno 63), ki povedo svetlost slikovnih elementov te vrstice slike (od leve proti desni: prvo število je za $x = 0$, zadnje za $x = n - 1$). Nato pride še ena prazna vrstica in nato še nadaljnjih n vrstic, ki na enak način podajajo drugo sliko (tisto po premiku miške).

Izhodna datoteka: za vsak par slik poišči zamik (dx, dy), pri katerem je dosežena najmanjša vrednost zgoraj opisane ocene (povprečna razlika absolutnih vrednosti razlik svetlosti tistih slikovnih elementov, ki so skupni stari in novi sliki). Če je ta najmanjša vrednost dosežena pri več zamikih, je vseeno, katerega izpišeš. Veljavni zamiki so le tisti, pri katerih sta dx in dy celi števili in velja $-3 \leq dx \leq 3$, $-3 \leq dy \leq 3$. Najdeni zamik za vsak par slik izpiši v svojo vrstico (najprej dx, nato dy, vmes pa presledek) in to v enakem vrstnem redu, v kakršnem so pari slik podani v vhodni datoteki. Vmes ne izpisuj praznih vrstic ali česa podobnega.

Točkovanje: če izhodna datoteka ni oblikovana tako, kot je opisano v prejšnjem odstavku, dobi tvoj program pri tem testnem primeru 0 točk. Drugače pa, če je od m premikov pravilno prepoznal k premikov, dobi pri tem testnem primeru $(10 \cdot k)$ div m točk.

Primer vhodne datoteke za nalogo 3.1:

```
2

4

56 52 47 40
57 48 43 39
56 50 44 36
56 50 41 35

53 50 45 40
56 54 49 46
52 46 42 34
51 45 38 32

5

53 42 33 26 17
45 36 30 23 13
37 30 23 20 12
27 25 19 15 12
19 16 12 11 9

36 45 43 36 28
35 46 36 31 23
37 47 27 24 18
38 47 19 16 13
39 45 9 9 9
```

Pripadajoča izhodna datoteka:

```
1 -2
-2 1
```

Pri prvem paru slik lahko na primer vidimo, da se slikovni elementi

```
52 47 40
48 43 39
```

na stari sliki

precej lepo ujemajo z elementi

```
52 46 42
51 45 38
```

na novi sliki.

Podobno se pri drugem paru lepo ujemajo

```
45 36 30
37 30 23
27 25 19
19 16 12
```

na stari in

```
43 36 28
36 31 23
27 24 18
19 16 13
```

na novi sliki.

2. Spletne knjigarne (`knjigarne.in`, `knjigarne.out`)

Prek spletnih knjigarn bi radi kupili po en izvod vsake izmed n knjig. Na voljo imamo k spletnih knjigarn; posamezno knjigo lahko kupimo od katerekoli od njih, vendar pa so lahko cene knjig v različnih knjigarnah različne. Plačati moramo tudi dostavo knjig na naš domači naslov. Knjigarna i ($1 \leq i \leq k$) računa a_i denarnih enot za dostavo prve naročene knjige in b_i za dostavo vsake naslednje naročene knjige (če iz neke knjigarne ne naročimo nobene knjige, ji seveda ni treba plačati ničesar). Vedno velja $a_i \geq b_i$. Cena knjige j ($1 \leq j \leq n$) pri knjigarni i je c_{ji} denarnih enot.

Skupna cena, ki jo bomo morali plačati za želenih n knjig, je torej v splošnem lahko odvisna od tega, pri kateri knjigarni bomo naročili katero knjigo. **Napiši program**, ki izračuna najmanjšo skupno ceno knjig (s stroški dostave vred), za katero lahko nakupimo vseh n knjig.

Vhodna datoteka: v prvi vrstici sta celi števili n in k, ločeni s presledkom. Zanju velja $1 \leq n \leq 10000$ in $1 \leq k \leq 10$. Sledi še $n + 2$ vrstic; v vsaki je po k celih števil, ločenih s po enim presledkom. V prvi od teh vrstic so cene $a_1, a_2, \ldots, a_k$; v drugi so cene $b_1, b_2, \ldots, b_k$; v ostalih vrsticah pa so cene posamezne knjige v vseh k knjigarnah: najprej je tu vrstica s cenami $c_{11}, c_{12}, \ldots, c_{1k}$ (cena prve knjige v vseh k knjigarnah), nato vrstica s cenami $c_{21}, c_{22}, \ldots, c_{2k}$ (cena druge knjige v vseh k knjigarnah) in tako naprej. Zadnja vrstica torej vsebuje cene $c_{n1}, c_{n2}, \ldots, c_{nk}$. Vse cene, a_i, b_i, c_{ji} za vse i od 1 do k in vse j od 1 do n, so cela števila, večja ali enaka 1 in manjša ali enaka 10000.

Izhodna datoteka: vanjo izpiši celo število, ki je najnižja skupna cena (s stroški dostave vred), za katero je mogoče dobiti vseh n knjig.

Primer vhodne datoteke:

```
10 3
8 10 10
5 7 10
10 8 11
10 9 12
9 9 12
8 7 11
9 9 12
9 9 14
9 9 10
9 10 11
8 8 11
10 8 3
```

Pripadajoča izhodna datoteka:

```
142
```

Za ta denar lahko pridemo do knjig, če kupimo vse razen zadnje knjige v prvi knjigarni, zadnjo pa v tretji.

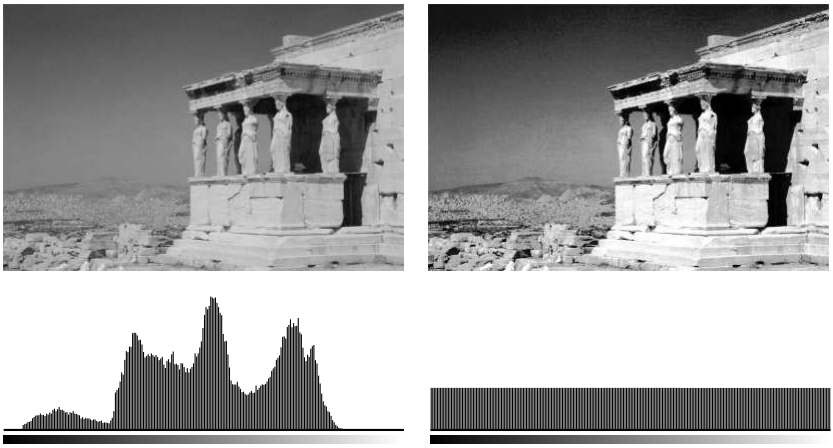
3. Izravnavanje histogramov (`histogrami.in`, `histogrami.out`)

Imamo sivinsko sliko, široko w slikovnih elementov (pikslov) in visoko h slikovnih elementov. Barva posameznega slikovnega elementa je opisana z enim od celih števil od 0 do 255 (večja ko je, svetlejši je ta slikovni element — 0 pomeni črno, 255 pa belo barvo).

Če za vsako svetlost od 0 do 255 preštejemo, koliko slikovnih elementov na sliki ima ravno to svetlost, in ta števila predstavimo s stolpci, dobimo *histogram* naše slike.

Izravnavanje histograma (*histogram equalization*) je postopek, pri katerem sliko popravimo tako, da postane njen histogram čim bolj „raven“ — torej da so vsi stolpci približno enako visoki.

Primer slike pred in po izravnavanju histograma (pod vsako različico slike je narisan tudi histogram):



Napiši program, ki prebere vhodno sliko I in izpiše izhodno sliko I' (enake velikosti kot I), za katero velja:

- Število slikovnih elementov najpogostejše barve na sliki I' je kvečjemu za 1 večje od števila slikovnih elementov najredkejše barve (tiste, ki ji pripada najmanj slikovnih elementov).

- Za vsak par slikovnih elementov (x_1, y_1) in (x_2, y_2) velja: če je slikovni element (x_1, y_1) na sliki I svetlejši od slikovnega elementa (x_2, y_2), je na sliki I' slikovni element (x_1, y_1) svetlejši ali pa enake barve kot (x_2, y_2), ni pa temnejši.

Če obstaja več slik I', ki ustrezajo gornjima pogojema, je vseeno, katero izpišeš.

Vhodna datoteka: v prvi vrstici sta celi števili h (višina slike) in w (širina slike), ločeni s presledkom. Obe sta večji ali enaki 1 in manjši ali enaki 200. Sledi h vrstic, v vsaki od njih pa je w celih števil, ločenih s po enim presledkom; i-to število v j-ti vrstici pove barvo slikovnega elementa na preseku i-tega stolpca in j-te vrstice slike. Vse barve so večje ali enake 0 in manjše ali enake 255.

Izhodna datoteka: vanjo zapiši h vrstic, v vsaki od njih pa naj bo w celih števil, ločenih s po enim presledkom. Ta števila naj opisujejo barve slikovnih elementov izhodne slike na enak način, kot je opisana vhodna slika v vhodni datoteki. Izhodna slika mora ustrezati zahtevam, navedenim v besedilu naloge.

Primer vhodne datoteke:

```
5 5
105 105 99 101 101
104 102 103 104 103
105 9 103 105 103
9 105 105 101 102
102 9 103 101 101
```

Ena od možnih pripadajočih izhodnih datotek:

```
19 20 3 4 5
17 9 12 18 13
21 0 14 22 15
1 23 24 6 10
11 2 16 7 8
```

4. Mafija (`mafija.in`, `mafija.out`)

V neki mafijski družini so člani urejeni hierarhično: vsakdo razen vrhovnega šefa ima natanko enega nadrejenega človeka in če sledimo povezavam od podrejenega do nadrejenega, lahko od vsakega člana družine sčasoma pridemo do vrhovnega šefa.

Denar v družini kroži takole. Tisti, ki nimajo nobenih podrejenih, morajo zbirati denar z raznimi kriminalnimi dejavnostmi in ves tako prisluženi denar oddati svojemu nadrejenemu. Ta mora ves denar, ki ga prejme od podrejenih, oddati *svojemu* nadrejenemu, on ga pošlje spet svojemu nadrejenemu in tako naprej, vse dokler ves denar ne konča v rokah vrhovnega šefa.

Vendar pa šef sumi, da nekateri člani družine goljufajo in ne pošljejo naprej svojemu nadrejenemu vsega denarja, ki so ga prejeli od svojih podrejenih. Šef je z izsiljevanjem in vohunjenjem uspel od vsakega člana pridobiti podatek o tem, koliko denarja je poslal svojemu nadrejenemu, zdaj pa prosi tebe, da mu **napišeš program**, ki bo ugotovil, kolikšna je največja vsota denarja, ki jo je utajil kakšen od članov družine (torej največja razlika med tem, koliko denarja je prejel od podrejenih, in tem, koliko ga je posredoval svojemu nadrejenemu). Za tiste, ki nimajo nobenega podrejenega, predpostavimo, da niso utajili nič svojega zaslužka (ker ne znamo oceniti, koliko denarja so v resnici zbrali s svojimi kriminalnimi dejavnostmi).

Vhodna datoteka: v prvi vrstici je celo število n ($0 < n \leq 100000$), ki pove, koliko članov ima družina. Člani družine so oštevilčeni s celimi števili od 1 do n. Sledi še n vrstic datoteke, ki za vsakega člana družine povedo, kdo je njegov nadrejeni in koliko denarja je ta član posredoval temu svojemu nadrejenemu. Tako torej $(i+1)$-va vrstica datoteke vsebuje dve celi števili, ločeni s presledkom; prvo od teh števil je številka člana, ki je neposredno nadrejen članu i, drugo pa je znesek denarja, ki ga je i oddal svojemu nadrejenemu (ta znesek je večji ali enak 0, obenem pa manjši ali enak vsoti zneskov, ki jih je član i prejel od svojih podrejenih). Pri vrhovnem šefu, ki nadrejenega nima, sta navedeni dve ničli. Vsota zneskov, ki jih posamezni član družine prejme od svojih podrejenih, pri nobenem članu ne presega 10^9 denarnih enot.

Izhodna datoteka: za vsakega člana (razen vrhovnega šefa) izračunaj razliko med zneskom, ki ga je prejel od podrejenih, in zneskom, ki ga je oddal nadrejenemu. Izpiši največjo vrednost te razlike po vseh članih.

Primer vhodne datoteke:

```
8
3 100
3 50
4 120
0 0
8 30
8 40
8 50
4 95
```

Pripadajoča izhodna datoteka:

```
30
```

Član 3 je od svojih podrejenih (1 in 2) prejel 150 denarnih enot, oddal pa 120, torej jih je utajil 30. Član 8 pa je od svojih podrejenih (5, 6 in 7) prejel 120 denarnih enot, oddal pa jih je 95, torej jih je utajil 25.

5. Tovornjaki (`tovornjaki.in`, `tovornjaki.out`)

Gneče na trajektih so običajen pojav in Jadrolinija vlaga veliko naporov v to, da bi jih zmanjšala. Običajno na obali stoji kolona tovornjakov in potrebno je čimprej ugotoviti, kako velik trajekt potrebujemo, da jih lahko naekrat prepeljemo.

Dano imamo torej zaporedje tovornjakov, ki bi jih radi razporedili na trajekt. Na trajektu bodo tovornjaki stali v treh pasovih in mi si lahko poljubno izberemo, na katerem pasu bo stal kateri tovornjak. Seveda pa skupna dolžina tovornjakov na nobenem od pasov ne sme presegati dolžine trajekta. **Napiši program**, ki bo za dano zaporedje dolžin tovornjakov poskusil ugotoviti, kako dolg je najkrajši trajekt, pri katerem bi se še dalo razporediti te tovornjake v tri pasove, ne da bi skupna dolžina tovornjakov na katerem od pasov presegla dolžino trajekta.

Vhodna datoteka: v prvi vrstici je celo število n ($1 \leq n \leq 100$), ki pove število tovornjakov. Sledi še n vrstic, v katerih so dolžine tovornjakov. Vsaka dolžina tovornjaka je celo število, večje ali enako 1 in manjše ali enako 100.

Izhodna datoteka: V prvo vrstico izpiši dolžino najkrajšega trajekta, na katerega je tvojemu programu uspelo razporediti tovornjake iz vhodne datoteke v skladu z zahtevami naloge. (Testni primeri bodo zasnovani tako, da ta dolžina najkrajšega trajekta nikoli ne bo večja od 100.) Sledi naj še n vrstic, ki opišejo nek konkreten razpored tovornjakov na ta najkrajši trajekt: v i-ti izmed teh vrstic naj bo število 1, 2 ali 3, ki pove, na kateri pas je bil razporejen i-ti tovornjak iz vhodne datoteke.

Točkovanje: če tvoja izhodna datoteka ne ustreza zgoraj opisanim zahtevam, dobiš pri tistem testnem primeru 0 točk, drugače pa je število točk odvisno od tega, kako dolg je tvoj trajekt v primerjavi z najkrajšim možnim. Recimo, da je pri nekem testnem primeru najkrajši možni trajekt dolg t^* enot, v tvoji izhodni datoteki pa je trajekt dolžine t enot. Potem dobiš pri tem testnem primeru $\max\{1, 10 - (t - t^*)\}$ točk. Z drugimi besedami, 10 točk dobiš le, če je tvoja rešitev najboljša možna ($t = t^*$); sicer pa se ti odbije toliko točk, za kolikor je tvoj trajekt daljši od najkrajšega možnega, vendar največ devet točk (tako da zagotovo dobiš vsaj eno točko).

Primer vhodne datoteke:

```
7
26
15
29
30
30
16
10
```

Pripadajoča izhodna datoteka:

```
55
1
2
1
2
3
3
2
```

Na trajekt dolžine 55 lahko razporedimo tovornjake takole: $26 + 29$ na en pas, $10 + 15 + 30$ na drugega in $16 + 30$ na tretjega. To je pri tem zaporedju tovornjakov tudi najkrajši možni trajekt.

REŠITVE NALOG ZA PRVO SKUPINO

1. Odstavki

V neki spremenljivki (v spodnji rešitvi je to PrejPrazna) si zapomnimo, če je bila prejšnja vrstica prazna ali ne. Na začetku postavimo PrejPrazna na false, ker bi se drugače naš program obnašal tako, kot da je pred prvo zares prebrano vrstico še neka prazna vrstica; če bi bila potem prva vrstica tudi prazna, bi jo program pobrisal, ker bi mislil, da je to zdaj že druga prazna vrstica zaporedoma (v resnici pa ni in je ne bi smel pobrisati).

Kakorkoli že, če sta trenutna in prejšnja vrstica prazni, trenutne vrstice ne bomo izpisovali. To bo zagotovilo, da bomo od vsake skupine dveh ali več zaporednih praznih vrstic izpisali le prvo, ostale pa preskočili in tako izpolnili zahtevo naloge. Potem pa si podatek o tem, ali je bila trenutna vrstica prazna ali ne, zapomnimo v spremenljivki PrejPrazna, saj bomo ta podatek potrebovali v naslednji iteraciji zanke, ko bomo prebrali naslednjo vrstico.

```
program Odstavki;
var S: string; PrejPrazna: boolean;
begin
  PrejPrazna := false;
  while not Eof do begin
    ReadLn(S);
    if not ((S = '') and PrejPrazna) then WriteLn(S);
    PrejPrazna := S = '';
  end; {while}
end. {Odstavki}
```

Oglejmo si še primer rešitve v C-ju. Spodnji program kaže malo drugačen pristop k reševanju te naloge: namesto po vrsticah lahko beremo vhodno datoteko tudi po znakih. Prazno vrstico prepoznamo po tem, da se pojavita dva zaporedna znaka za konec vrstice (`'\n'`); če pa se pojavijo trije taki znaki zapored, pomeni, da imamo dve zaporedni prazni vrstici in nam druge ni treba izpisati.

```
#include <stdio.h>
int main()
{
  int c, c1 = -1, c2 = -1;
  while ((c = getchar()) != EOF) {
    if (! (c == '\n' && c1 == c && c2 == c)) putchar(c);
    c2 = c1; c1 = c; }
  return 0;
}
```

To nalogo lahko zelo na kratko in elegantno rešimo tudi v perlu, kar iz ukazne vrstice:

```
perl -ne 'print unless /^$/ && $prejPrazna; $prejPrazna = /^$/'
```

Stikalo `-n` mu naroči, naj bere vhod po vrsticah in pri vsaki izvede dani program, `-e` pa mu pove, da je program podan kar kot naslednji parameter v ukazni vrstici. S pogojem `/^$/` preverimo, če je trenutna vrstica prazna — v regularnih izrazih pomeni `^` začetek, `$` pa konec vrstice, tako da se z izrazom `^$` ujema le prazna vrstica.

2. Sneg

V neki spremenljivki (v spodnji rešitvi je to Debelina) hranimo trenutno debelino snežne odeje. Za vsak dan moramo prebrati podatek o tem, za koliko se poveča in za koliko zmanjša, potem pa lahko izračunamo novo debelino in jo izpišemo.

```
program Sneg;
var i, Debelina, Povecanje, Zmanjsanje: integer;
begin
  Debelina := 0;
  for i := 1 to 365 do begin
    ReadLn(Povecanje, Zmanjsanje);
    Debelina := Debelina + Povecanje − Zmanjsanje;
    WriteLn(Debelina);
  end; {for i}
end. {Sneg}
```

Še primer rešitve v C-ju:

```
#include <stdio.h>

int main()
{
  int i, debelina = 0, povecanje, zmanjsanje;
  for (i = 0; i < 365; i++) {
    scanf("%d %d", &povecanje, &zmanjsanje);
    debelina = debelina + povecanje − zmanjsanje;
    printf("%d\n", debelina); }
  return 0;
}
```

3. Sudoku

Lahko si pomagamo z množicami (**set** v pascalu). Za vsako vrstico izračunamo množico vseh števil v tej vrstici; če ta množica ni enaka [1..9], mora v tej vrstici kakšno od števil od 1 do 9 manjkati. Pogoja, da se v vrstici nobeno število ne sme pojavljati več kot enkrat, pa nam ni treba še dodatno preverjati: če se kakšno pojavlja več kot enkrat, mora kakšno drugo manjkati (ker imamo devet števil, v vrstici pa devet kvadratkov) in bomo dano polje zavrnili že zaradi tega. Enako kot za vrstice naredimo tudi za stolpce in za male kvadrate velikosti 3×3.

```
program Sudoku;

  type SudokuPoljeT = array [1..9, 1..9] of 1..9;
  procedure Preberi(var T: SudokuPoljeT); external;

var T: SudokuPoljeT; V, S, K: array [0..8] of set of 1..9; i, j, a: integer; Ok: boolean;
begin
  for i := 0 to 8 do begin V[i] := []; S[i] := []; K[i] := [] end;
  Preberi(T); Ok := true;
  for i := 0 to 8 do for j := 0 to 8 do begin
    a := T[i + 1, j + 1]; V[i] := V[i] + [a]; S[j] := S[j] + [a];
    K[(i div 3) * 3 + j div 3] := K[(i div 3) * 3 + j div 3] + [a];
  end; {for i}
  for i := 0 to 8 do if (V[i] <> [1..9]) or (S[i] <> [1..9]) or (K[i] <> [1..9]) then
    Ok := false;
  if Ok then WriteLn('PRAVILNA') else WriteLn('NAPACNA');
end. {Sudoku}
```

Namesto množic bi lahko tudi prižigali bite v kakšni celoštevilski spremeljivki ali pa bi uporabili tabelo devetih booleanov. S prižiganjem bitov deluje na primer tale program v C-ju:

```
#include <stdio.h>

typedef int SudokuPoljeT[9][9];
extern void Preberi(SudokuPoljeT T);

int main()
{
  SudokuPoljeT T; int x, y, i, v[9], s[9], k[9];
  Preberi(T);
  for (i = 0; i < 9; i++) v[i] = 0, s[i] = 0, k[i] = 0;
  for (y = 0; y < 9; y++) for (x = 0; x < 9; x++) {
    i = 1 << (T[y][x] - 1);
    v[y] |= i; s[x] |= i; k[(y / 3) * 3 + x / 3] |= i; }
  for (i = 0; i < 9 && v[i] == 511 && s[i] == 511 && k[i] == 511; i++) ;
  if (i == 9) printf("PRAVILNA\n"); else printf("NAPACNA\n");
  return 0;
}
```

Še ena možnost pa je na primer ta, da si napišemo podprogram, ki preveri, če se neko število pojavi v vsaki vrstici, stolpcu in kvadratu 3×3. V spodnji rešitvi je to podprogram Preveri. Potem ga moramo le še poklicati po enkrat za vsako število od 1 do 9:

```
program Sudoku;

  type SudokuPoljeT = array [1..9, 1..9] of 1..9;
  procedure Preberi(var T: SudokuPoljeT); external;

  function Preveri(var T: SudokuPoljeT; n: integer): boolean;
  var i, j, x, y: integer; Nasel: boolean;
  begin
    Preveri := false;
    for i := 1 to 9 do begin       { Preverimo i-to vrstico in i-ti stolpec. }
      Nasel := false; for j := 1 to 9 do if T[i, j] = n then Nasel := true;
      if not Nasel then exit;
      Nasel := false; for j := 1 to 9 do if T[j, i] = n then Nasel := true;
      if not Nasel then exit;
    end; {for i}
    for i := 0 to 2 do for j := 0 to 2 do begin          { Preverimo mali kvadrat 3 × 3 }
      Nasel := false;                { z zgornjim levim kotom v (3 * i + 1, 3 * j + 1). }
      for y := 1 to 3 do for x := 1 to 3 do
        if T[3 * i + y, 3 * j + x] = n then Nasel := true;
      if not Nasel then exit;
    end; {for j, i}
    Preveri := true;
  end; {Preveri}

var n: integer; T: SudokuPoljeT; Ok: boolean;
begin {Sudoku}
  Ok := true; Preberi(T);
  for n := 1 to 9 do if not Preveri(T, n) then begin Ok := false; break end;
  if Ok then WriteLn('PRAVILNA') else WriteLn('NAPACNA');
end. {Sudoku}
```

4. Naraščajoče besede

Najdaljšo doslej znano naraščajočo besedo hranimo v spremenljivki Naj. Ko prebiramo besede, lahko tiste, ki niso daljše od Naj, kar preskočimo — tudi če so naraščajoče, očitno ne morejo biti najdaljše naraščajoče besede v celem vhodnem zaporedju. Za tiste besede pa, ki so dovolj dolge, preverimo, če so naraščajoče. S števcem i se sprehodimo po besedi in za vsako črko preverimo, če je v abecednem vrstnem redu za prejšnjo; če ni, se ustavimo. Če na ta način pridemo do konca besede, pomeni, da je vsaka črka po abecedi res za prejšnjo, torej je beseda naraščajoča. Ker smo že prej ugotovili, da je tudi daljša od Naj, si jo zapomnimo kot novo najdaljšo doslej znano naraščajočo besedo. Ko na ta način obdelamo celotno vhodno zaporedje, moramo le še izpisati Naj.

```
program NarascajoceBesede;
var Beseda, Naj: string; i: integer; Narasca: boolean;
begin
  Naj := '';
  while not Eof do begin
    ReadLn(Beseda);
    if Length(Beseda) > Length(Naj) then begin
      i := 2; Narasca := true;
      while i <= Length(Beseda) do
        if Beseda[i] > Beseda[i - 1] then i := i + 1
        else begin Narasca := false; break end;
      if Narasca then Naj := Beseda;
    end; {if}
  end; {while}
  WriteLn('Najdaljša naraščajoča beseda: "', Naj, '"');
end. {NarascajoceBesede}
```

Oglejmo si še primer rešitve v C-ju. Spodnji program je še malo robustnejši, kot naloga zahteva — deloval bi tudi, če ne bi bila vsaka beseda v svoji vrstici (dovolj mu je že, da so besede ločene s poljubnimi nečrkovnimi znaki).

```
#include <stdio.h>
#define MaxDolz 100
int main()
{
  char naj[MaxDolz + 1] = "", beseda[MaxDolz + 1];
  int c = 0, cp, dolz = 0, najDolz = 0, narasca;
  while (1)
  {
    cp = c; c = getchar();
    if ('A' <= c && c <= 'Z')
      { narasca = narasca && (cp < c); beseda[dolz++] = c; }
    else {
      if (dolz > najDolz && narasca)
        { beseda[dolz] = 0; strcpy(naj, beseda); najDolz = dolz; }
      dolz = 0; narasca = 1; if (c == EOF) break; else c = 0; }
  }
  printf("%s\n", naj);
  return 0;
}
```

Kot se pri nalogah na temo obdelovanja besedil pogosto zgodi, lahko tudi to nalogo veliko krajše rešimo v nekaterih drugih programskih jezikih. Spodaj je primer kratke rešitve v pythonu:

```
print max((len(t), t) for s in __import__("sys").stdin for t in [s.strip()]
                                                    if "".join(sorted(t)) == t)[1]
```

Argument funkcije max je v tem primeru generator — funkcija, ki v zanki prebira standardni vhod po vrsticah, za vsako vrstico s ustvari seznam [s.strip()] (ki torej vsebuje le en element — to isto vrstico brez znaka za konec vrstice), nato pa z notranjo zanko pregleda ta seznam, iz njega dobi niz t (torej trenutno vrstico brez znaka za konec vrstice) in, če je izpolnjen pogoj za besedo **if**, vrne urejeni par (len(t), t). Funkcija max med vsemi temi pari vrne največjega, torej tistega, pri katerem je t najdaljša naraščajoča beseda (če je več enako dolgih, bomo dobili tisto, ki je zadnja po abecedi). Druga komponenta tega para (do katere pridemo z [1]) je torej najdaljša naraščajoča beseda.

Pogoj **if** "".join(sorted(t)) == t preveri, če je beseda t naraščajoča. Pri tem si pomaga z dejstvom, da se zna niz obnašati kot zaporedje črk, zato ga lahko pošljemo funkciji sorted, ki vrne urejen seznam elementov danega zaporedja. V našem primeru torej sorted(t) vrne seznam črk t-ja, urejen po abecedi; če te črke spet staknemo, dobimo niz, ki je tak kot t, le črke ima urejene po abecedi; če je ta niz enak t-ju, pomeni, da je imel že t črke urejene po abecedi, torej je bil naraščajoča beseda.

5. Podnapisi

Preprosta rešitev lahko vsakič pregleda vse podnapise in poišče ustreznega:

```
function PodnapisZaSlicico(StevilkaSlicice: integer): string;
var i: integer;
begin {PodnapisZaSlicico}
  for i := 1 to SteviloPodnapisov do
    if (StevilkaSlicice >= PrvaSlicica(i)) and (StevilkaSlicice <= ZadnjaSlicica(i)) then
      begin PodnapisZaSlicico := Vsebina(i); exit end;
  PodnapisZaSlicico := '';
end; {PodnapisZaSlicico}
```

Učinkovitejša rešitev upošteva dejstvo, da sličice filma ponavadi pregledujemo po vrsti; zato je pogosto dober kar isti podnapis kot ob zadnjem klicu naše funkcije; zapomnimo si ga v neki globalni spremenljivki. Če ni pravi ta, bo pa pogosto dober naslednji ali pa prejšnji (če predvajamo film nazaj); precej mogoče je tudi, da smo na eni od sličic med dosedanjim in naslednjim podnapisom (in tam podnapisa ni, saj jih imamo podane po vrsti). Če nič od tega ne uspe, lahko pravi podnapis poiščemo z bisekcijo.

```
var ZadnjiPodnapis: integer;

procedure Inicializacija;
begin
  ZadnjiPodnapis := 0;
end; {Inicializacija}
```

```
function PodnapisZaSlicico(StevilkaSlicice: integer): string;

  procedure Vrni(StPodnapisa: integer);
  begin
    ZadnjiPodnapis := StPodnapisa;
    PodnapisZaSlicico := Vsebina(StPodnapisa);
  end; {Vrni}

var L, R, M: integer;
begin {PodnapisZaSlicico}
  PodnapisZaSlicico := '';
  if ZadnjiPodnapis > 0 then
    if StevilkaSlicice < PrvaSlicica(ZadnjiPodnapis) then begin
      if ZadnjiPodnapis = 1 then exit
      else if StevilkaSlicice > ZadnjaSlicica(ZadnjiPodnapis - 1) then exit
      else if StevilkaSlicice >= PrvaSlicica(ZadnjiPodnapis - 1) then
        begin Vrni(ZadnjiPodnapis - 1); exit end;
    end else if StevilkaSlicice > ZadnjaSlicica(ZadnjiPodnapis) then begin
      if ZadnjiPodnapis = SteviloPodnapisov then exit
      else if StevilkaSlicice < PrvaSlicica(ZadnjiPodnapis + 1) then exit
      else if StevilkaSlicice <= ZadnjaSlicica(ZadnjiPodnapis + 1) then
        begin Vrni(ZadnjiPodnapis + 1); exit end
    end else
      begin Vrni(ZadnjiPodnapis); exit end
  L := 1; R := SteviloPodnapisov;
  while L < R do begin
    { Na tem mestu velja: če je treba na tej sličici sploh prikazati kakšen podnapis,
      je to eden od podnapisov L..R. }
    M := (L + R) div 2;
    if StevilkaSlicice < PrvaSlicica(M) then R := M - 1
    else if StevilkaSlicice > ZadnjaSlicica(M) then L := M + 1
    else begin Vrni(M); exit end;
  end; {while}
  if L <= R then if (StevilkaSlicice >= PrvaSlicica(L))
                    and (StevilkaSlicice <= ZadnjaSlicica(L)) then Vrni(L);
end; {PodnapisZaSlicico}
```

Prostorsko malo potratnejša rešitev (besedilo naloge pravzaprav že pravi, da je takšna poraba prostora nesprejemljiva) pa je ta, da si ob inicializaciji v neki tabeli za vsako sličico zapomnimo ustrezni indeks podnapisa. Potem je PodnapisZaSlicico res zelo hitra, saj mora le pogledati v pravi element te tabele.

```
type TabelaT = packed array [1..1000000] of integer;
var Tabela: ↑TabelaT; StSlicic: integer;

procedure Inicializacija;
var i, t: integer;
begin
  if SteviloPodnapisov <= 0 then begin StSlicic := 0; exit end;
  StSlicic := ZadnjaSlicica(SteviloPodnapisov);
  GetMem(Tabela, SizeOf(integer) * StSlicic);
  for t := 1 to StSlicic do Tabela↑[t] := 0;
  for i := 1 to SteviloPodnapisov do
    for t := PrvaSlicica(i) to ZadnjaSlicica(i) do Tabela↑[t] := i;
end; {Inicializacija}

function PodnapisZaSlicico(StevilkaSlicice: integer): string;
var i: integer;
```

```
begin
  i := 0;
  if (StevilkaSlicice >= 1) and (StevilkaSlicice <= StSlicic)
  then i := Tabela↑[StevilkaSlicice];
  if i = 0 then PodnapisZaSlicico := ''
  else PodnapisZaSlicico := Vsebina(i);
end; {PodnapisZaSlicico}
```

Oglejmo si še dve rešitvi v C-ju. Najprej preprosta in manj učinkovita:

```
const char *PodnapisZaSlicico(int StevilkaSlicice)
{
  int i;
  for (i = 1; i <= SteviloPodnapisov(); i++)
    if (PrvaSlicica(i) <= StevilkaSlicice && StevilkaSlicice <= ZadnjaSlicica(i))
      return Vsebina(i);
  return 0;
}
```

Pa še malo boljša rešitev, ki si zapomni zadnji prikazani podnapis (oz. indeks nekega podnapisa, ki je blizu zadnje prikazane sličice, tudi če prav na tisti sličici ni nobenega podnapisa), če pa ta ni dober tudi ob naslednjem klicu, pregleduje zaporedje podnapisov naprej ali pa nazaj od tistega podnapisa:

```
int ZadnjiPodnapis;
void Inicializacija()
  { ZadnjiPodnapis = 1; }
const char *PodnapisZaSlicico(int StevilkaSlicice)
{
  int i = ZadnjiPodnapis;
  while (i > 1 && StevilkaSlicice < PrvaSlicica(i)) i--;
  while (i < SteviloPodnapisov() && StevilkaSlicice > ZadnjaSlicica(i)) i++;
  ZadnjiPodnapis = i;
  if (PrvaSlicica(i) <= StevilkaSlicice && StevilkaSlicice <= ZadnjaSlicica(i))
    return Vsebina(i); else return 0;
}
```

REŠITVE NALOG ZA DRUGO SKUPINO

1. 1337ovščina

Vhodno besedilo berimo znak za znakom. Naj bo c trenutni znak in s niz, v katerem so vse možne zamenjave tega znaka (ločene s po enim presledkom). Če je na primer možnih pet zamenjav, vsebuje niz s štiri presledke. Če torej presledke preštejemo in prištejemo 1, dobimo ravno število možnih zamenjav znaka c. (Poseben primer je, če je s prazen niz; tedaj možnih zamenjav pač ni. Če bi tu šteli presledke in prišteli 1, bi dobili napačen vtis, da je možna ena zamenjava.)

Zdaj se moramo nekako odločiti, katero od naših n možnih zamenjav bi uporabili. Skupaj z možnostjo, da znak c pustimo kar v prvotni obliki (ga ne zamenjamo z ničimer), imamo torej $n+1$ možnosti. Spodnja rešitev poskuša ravnati tako, da bi bile vse te možnosti enako verjetne. S stavkom m := Nakljucje(n + 1) − 1 dobimo neko naključno število od 0 do n. Pri $m = n$ bomo pustili vhodni znak c pri miru, drugače pa ga bomo zamenjali z eno od možnih zamenjav, in sicer s tisto, ki v nizu s stoji med m-tim in $(m+1)$-vim presledkom. Tako pri $m = 0$ izberemo prvo možno zamenjavo znaka c, pri $m = 1$ drugo in tako naprej. Zdaj se moramo le še sprehoditi po nizu s, šteti presledke in izpisati pravo zamenjavo.

```
program Leet;
var c: char; s: string; i, n, m: integer;
begin
  while not Eof do
    if Eoln then begin ReadLn; WriteLn; end
    else begin
      Read(c);
      s := MozneZamenjave(c);
      if Length(S) = 0 then n := 0 else n := 1;
      for i := 1 to Length(s) do if s[i] = ' ' then n := n + 1;
      m := Nakljucje(n + 1) − 1;
      if m = n then Write(c)
      else for i := 1 to Length(s) do
        if s[i] = ' ' then if m = 0 then break else m := m − 1
        else if m = 0 then Write(s[i]);
    end; {if}
end. {Leet}
```

Zapišimo to rešitev še v C-ju:

```
#include <stdio.h>

extern const char *MozneZamenjave(char c);
extern int Nakljucje(int n);

int main()
{
  int c, n, m; const char *p;
  while ((c = getchar()) != EOF)
  {
    n = 0; p = MozneZamenjave(c);
    if (p && *p) for (n += 1; *p; ) if (*p++ == ' ') n++;
    m = Nakljucje(n + 1) − 1;
    if (m == n) putchar(c);
```

```
    else for (p = MozneZamenjave(c); *p && m >= 0; p++)
      if (*p == ' ') m--; else if (m == 0) putchar(*p);
  }
  return 0;
}
```

Učinkovitejša rešitev ne bi vsakič sproti prežvekovala celega niza MozneZamenjave(c), štela presledkov v njem in tako naprej, pač pa bi to naredila na začetku za vse možne znake c; za vsakega bi si zapomnila, koliko možnih zamenjav ima, in posamezne zamenjave shranila v neko tabelo. Potem bi morali po vsakem branju nekega znaka c le izpisati naključno izbran element c-jeve tabele možnih zamenjav.

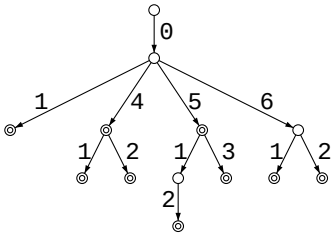
2. Predpone

Preprosta in ne najbolj učinkovita rešitev je, da gremo z zanko po vseh predponah in za vsako preverimo, če je to res predpona naše telefonske številke. Med tistimi, ki so, si zapomnimo najdaljšo. Spodnja rešitev primerja predpono z začetnimi znaki telefonske številke v notranji zanki **while**; če ta zazna neujemanje, se ustavi, če pa srečno pride skozi vseh DP znakov, kolikor jih je v trenutni predponi, vemo, da je to res predpona naše telefonske številke.

Da si prihranimo nekaj dela, si zapomnimo tudi dolžino najdaljše doslej znane predpone; če neka kasnejša predpona ni daljša od nje, jo lahko preskočimo, saj vemo, da četudi je to res predpona naše številke, gotovo ne bo postala nova najdaljša znana predpona. Podobno preskočimo tudi morebitne predolge predpone (daljše od cele telefonske številke).

```
function NajdaljsaPredpona(TelSt: string): string;
var i, pi, DP, DS, MaxD: integer; Predpona: string;
begin
  MaxD := 0; DS := Length(TelSt); NajdaljsaPredpona := '';
  for pi := 1 to StPredpon do begin
    Predpona := PovejPredpono(pi); DP := Length(Predpona);
    if (DP > DS) or (DP <= MaxD) then continue;
    i := 1; while i <= DP do if TelSt[i] = Predpona[i] then i := i + 1 else break;
    if i > DP then begin NajdaljsaPredpona := Predpona; MaxD := DP end;
  end; {for i}
end; {NajdaljsaPredpona}
```

Slabost te rešitve je, da mora iti pri vsaki telefonski številki po celem seznamu predpon. Če je predpon veliko, je to zamudno, pa še nepotrebno, saj večina predpon praviloma sploh ne nastopa na začetku naše telefonske številke (torej niso zares njene predpone). Hitreje bo šlo, če si ob inicializaciji vse predpone uredimo v drevo. Vsako vozlišče drevesa ima lahko do deset poddreves, ki ustrezajo posameznim števkam od 0 do 9. Vsaka pot od korena drevesa do nekega vozlišča tako ustreza nekemu zaporedju števk, v vozlišču pa imejmo podatek o tem, ali je to zaporedje števk tudi res ena od predpon ali ne. Primer drevesa za predpone 01, 04, 041, 042, 05, 0512, 053, 061 in 062 (z dvojnimi krožci so označena tista vozlišča, pri katerih se res konča neka predpona):



Podprogram NajdaljsaPredpona lahko zdaj jemlje števke z začetka dane telefonske številke in se spušča po drevesu s korena navzdol; vsakič gre v tisto poddrevo, na katerega kaže povezava, označena s trenutno števko. (Če take povezave ni, vemo, da ni nobene predpone, ki bi pokrivala ves doslej prebrani del naše telefonske številke, zato lahko nehamo.) Na vsakem koraku pogledamo, če se pri trenutnem vozlišču končuje kakšna predpona, in če se, si jo zapomnimo kot najdaljšo doslej znano predpono naše dane telefonske številke.

Drevo zgradi podprogram Inicializacija, ki začne s praznim drevesom (takim, ki vsebuje le koren) in potem vanj eno za drugo dodaja vse predpone. Za vsako predpono moramo poskrbeti, da obstaja v drevesu pot od korena navzdol, pri kateri se števke na povezavah zložijo ravno v to predpono. Če ustrezne povezave še ne obstajajo, bo pač treba dodati kakšno novo vozlišče.

```
type VozlisceP = ↑VozlisceT;
     VozlisceT = record
       JePredpona: boolean;
       Otroci: array [0..9] of VozlisceP;
     end; {VozlisceT}

var Koren: VozlisceP;

{ Ta podprogram ustvari novo vozlišče in ga doda med otroke vozlišča Oce (če Oce ni nil). }
function DodajVozlisce(Oce: VozlisceP; Stevka: integer): VozlisceP;
var V: VozlisceP; i: integer;
begin
  New(V); V↑.JePredpona := false;
  for i := 0 to 9 do V↑.Otroci[i] := nil;
  if Oce <> nil then Oce↑.Otroci[Stevka] := V;
  DodajVozlisce := V;
end; {DodajVozlisce}

procedure Inicializacija;
var i, j, Stevka: integer; Predpona: string; V: VozlisceP;
begin
  Koren := DodajVozlisce(nil, 0);
  for i := 1 to StPredpon do begin
    Predpona := PovejPredpono(i);
    V := Koren;
    for j := 1 to Length(Predpona) do begin
      Stevka := Ord(Predpona[j]) - Ord('0');
      if V↑.Otroci[Stevka] <> nil then V := V↑.Otroci[Stevka]
      else V := DodajVozlisce(V, Stevka);
    end; {for j}
```

```
      V↑.JePredpona := true;
  end; {for i}
end; {Inicializacija}

function NajdaljsaPredpona(TelSt: string): string;
var Predpona: string; V: VozlisceP; i: integer;
begin
  V := Koren; Predpona := ''; NajdaljsaPredpona := '';
  for i := 1 to Length(TelSt) do begin
      V := V↑.Otroci[Ord(TelSt[i]) - Ord('0')];
      if V = nil then break;
      Predpona := Predpona + TelSt[i];
      if V↑.JePredpona then NajdaljsaPredpona := Predpona;
  end; {for i}
end; {NajdaljsaPredpona}
```

Še rešitev v C-ju:

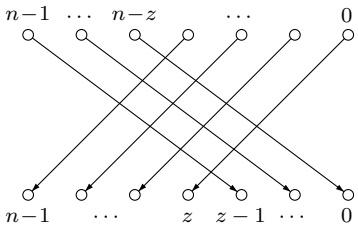
```
extern int StPredpon();
extern const char *PovejPredpono(int ZapSt);

typedef struct VozlisceT_{
  int stevka, predpona;
  struct VozlisceT_ *otroci[10];
} VozlisceT, *VozlisceP;

VozlisceP koren;

VozlisceP NovoVozlisce(int stevka)
{
  int i; VozlisceP p = malloc(sizeof(VozlisceT));
  p->stevka = stevka; p->predpona = -1;
  for (i = 0; i < 10; i++) p->otroci[i] = 0;
  return p;
}

void DodajPredpono(int stPredpone)
{
  VozlisceP p = koren; int stevka;
  const char *predpona = PovejPredpono(stPredpone);
  while (*predpona) {
    stevka = *predpona++ - '0';
    if (! p->otroci[stevka]) p->otroci[stevka] = NovoVozlisce(stevka);
    p = p->otroci[stevka]; }
  p->predpona = stPredpone;
}

void Inicializacija()
{
  int i; koren = NovoVozlisce(-1);
  for (i = 1; i <= StPredpon(); i++) DodajPredpono(i);
}

const char *NajdaljsaPredpona(const char *TelSt)
{
  VozlisceP p = koren, naj = koren;
  while (p && *TelSt) {
    if (p && p->predpona >= 0) naj = p;
    p = p->otroci[*TelSt++ - '0']; }
  return naj->predpona >= 0 ? PovejPredpono(naj->predpona) : 0;
}
```

3. Cestne lučke

Podprogram Ugasni mora le ugasniti vse lučke, torej postaviti stanje na 0. Podprogram Utripaj lahko v neskončni zanki prižiga in ugaša lučke, vmes pa čaka; ko bo treba z utripanjem prenehati, ga bo že prekinil sistem (tako obljublja besedilo naloge.)

```
procedure Ugasni;
begin
  Nastavi(0);
end; {Ugasni}

procedure Utripaj;
begin
  while true do begin
    Nastavi(UTRIPA); Pocakaj;
    Nastavi(0); Pocakaj;
  end; {while}
end; {Utripaj}
```

Nekaj več dela pa je s potovanjem lučk v levo in desno. Naj bo n število vseh lučk; mislimo si neko n-bitno število, v katerem vsak bit pove stanje ene lučke (bit 0 predstavlja skrajno desno lučko, bit $n-1$ pa skrajno levo). Kako se to število spremeni, če ciklično zamaknemo vzorec na lučkah za z mest v levo? Tule je primer za $n = 7$, $z = 3$:



Vidimo torej, da se vsebina spodnjih $n-z$ bitov (od 0 do $n-z-1$) premakne v bite od z do $n-1$, vsebina zgornjih z bitov (od $n-z$ do $n-1$) pa se premakne v bite od 0 do $z-1$.

Do spodnjih nekaj bitov nekega števila x lahko pridemo s pomočjo operatorja **and**. Tako na primer z izrazom x **and** 1111_2 dobimo spodnje štiri bite števila x, ostali pa so ugasnjeni (ker so bili ugasnjeni tudi v desnem operandu, 1111_2). V splošnem, če hočemo dobiti spodnjih k bitov števila x, moramo x z**and**ati z operandom, ki ima spodnjih k bitov prižganih, ostale pa ugasnjene. Če bi tako število $111\ldots1_2$ povečali za 1, vidimo, da pride povsod do prenosa in dobili bi rezultat $10\ldots0_2$ (enica in k ničel), to pa je ravno 2^k. Naš operand $111\ldots1_2$ je torej enak 2^k-1. Do števila 2^k pa lahko pridemo z izrazom oblike 1 **shl** k. Na ta način si pripravi spodnji podprogram v spremenljivki Spodnji spodnjih $n-z$ bitov števila POTUJE.

Do zgornjih z bitov pridemo še enostavneje — število POTUJE preprosto zamaknemo za $n-z$ bitov v desno (biti od n-tega naprej so bili, kot pravi naloga, v konstanti POTUJE tako ali tako ugasnjeni in se nam zdaj ni treba posebej ubadati s tem, kako bi jih ugasnili). Tako dobimo v spodnjih z bitih spremenljivke Zgornji vsebino zgornjih z bitov konstante POTUJE.

Zdaj nam ostane le še, da spodnjih $n - z$ bitov prvotnega vzorca POTUJE (ki so zdaj v Spodnji) zamaknemo za z mest v levo in na tako izpraznjena mesta bite postavimo vsebino zgornjih z bitov vzorca POTUJE (ki so zdaj na spodnjih z bitih spremenljivke Zgornji). To potem ponavljamo v neskončni zanki in zamik bodisi povečujemo (če hočemo premikanje v desno) ali pa zmanjšujemo (če hočemo premikanje v levo).

```
procedure PotujoceLucke(Smer: integer);
var Zamik, Spodnji, Zgornji: integer;
begin
  Zamik := 0;
  while true do begin
    { Spodnjih SteviloLuck − Zamik bitov. }
    Spodnji := POTUJE and ((1 shl (SteviloLuck − Zamik)) − 1);
    { Zgornjih Zamik bitov. }
    Zgornji := POTUJE shr (SteviloLuck − Zamik);
    { Nastavimo novo stanje. Zgornjih Zamik bitov pride na dno. }
    Nastavi((Spodnji shl Zamik) or Zgornji); Pocakaj;
    { Premaknimo Zamik. }
    Zamik := Zamik + Smer;
    if Zamik < 0 then Zamik := SteviloLuck − 1;
    if Zamik = SteviloLuck then Zamik := 0;
  end; {while}
end; {PotujoceLucke}

procedure VLevo; begin PotujoceLucke(1) end;
procedure VDesno; begin PotujoceLucke(−1) end;
```

Gornja rešitev uporablja nestandardne razširitve pascala (ki so sicer v dandanašnjih narečjih zelo razširjene): **and** in **or** uporablja kot aritmetična operatorja (na bitih), medtem ko sta v standardnem pascalu definirana le kot logična operatorja; operatorjev **shl** in **shr** pa standardni pascal sploh ne pozna.

Oglejmo si zdaj še primer rešitve, ki ne uporablja takšnih nestandardnih razširitev. Namesto tega si bomo pomagali z množenjem in deljenjem s potencami števila 2. Če število delimo z 2^k, ga v bistvu zamaknemo za k bitov v desno; če ga množimo z 2^k, ga s tem zamaknemo za k bitov v levo. Če izračunamo njegov ostanek po deljenju z 2^k, pa dobimo pravzaprav spodnjih k bitov prvotnega števila. Tako lahko premikamo kose nekega števila za določeno število mest v levo in desno.

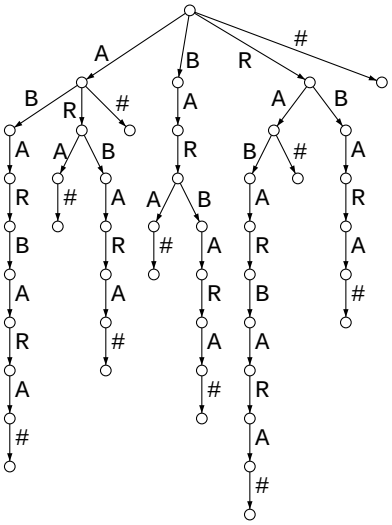
```
procedure PotujoceLucke(Smer: integer);
var Stanje, i, m: integer;
begin
  Stanje := POTUJE; m := 1;
  for i := 1 to SteviloLuck − 1 do m := m * 2;
  while true do begin
    Nastavi(Stanje);
    if Smer < 0 then { v desno }
      Stanje := m * (Stanje mod 2) + (Stanje div 2)
    else { v levo }
      Stanje := 2 * (Stanje mod m) + (Stanje div m);
    Pocakaj;
  end; {while}
end; {PotujoceLucke}
```

Za konec si oglejmo še rešitev v C-ju. Tu so operatorji za zamike standarden del jezika.

```
extern int Nastavi(int BitnaSlika);
extern void Pocakaj();

const int SteviloLuck = ..., POTUJE = ..., UTRIPA = ...;

void Ugasni()
   { Nastavi(0); }

void Utripaj()
{
   while (1) {
      Nastavi(UTRIPA); Pocakaj();
      Nastavi(0); Pocakaj(); }
}

void VLevo() {
   int stanje = POTUJE;
   while (1) {
      Nastavi(stanje); Pocakaj();
      stanje = (stanje & (1 << SteviloLuck - 1) - 1) << 1 | stanje >> SteviloLuck - 1; }
}

void VDesno() {
   int stanje = POTUJE;
   while (1) {
      Nastavi(stanje); Pocakaj();
      stanje = stanje >> 1 | (stanje & 1) << SteviloLuck - 1; }
}
```

4. Drevo končnic

(a) Drevo končnic za niz `RABARBARA` je takšno:



(b) Naj bo v neko vozlišče našega drevesa končnic; označimo število končnic, ki se končujejo v poddrevesu s korenom pri v (v to poddrevo šteje tudi vozlišče v sámo), s $f(v)$. To je ravno enako številu listov v tem poddrevesu. Vrednost $f(v)$ lahko za vsa vozlišča izračunamo tako, da rekurzivno pregledamo drevo od spodaj navzgor in seštevamo vrednosti f-ja po otrocih vsakega vozlišča (v listih pa je $f(v) = 1$).

Mislimo si zdaj nek niz $p = p_1 p_2 \ldots p_t$, ki se pojavlja r-krat kot podniz niza s. To pomeni, da za r različnih začetnih indeksov $i_1, i_2, \ldots, i_r$ velja $s_{i_j} s_{i_j+1} \ldots s_{i_j+t-1} = p_1 \ldots p_t$. To pa pomeni, da se vse končnice $s_{i_j} s_{i_j+1} \ldots s_n$ (za $j = 1, \ldots, r$) začnejo na niz p. Z drugimi besedami: število pojavitev p-ja kot podniza v s-ju je ravno enako številu s-jevih končnic, ki se začnejo na p. Te končnice pa bomo našli tako, da se po našem drevesu sprehodimo od korena navzdol in sledimo povezavam, kot jih določajo črke p-ja. Če v nekem trenutku ni primerne povezave, pomeni, da se sploh nobena s-jeva končnica ne začne na p, zato se p v s-ju sploh ne pojavlja. Če pa se naš sprehod ustavi v nekem vozlišču v, vemo, da se vsaka končnica s-ja, ki se končuje nekje v v-jevem poddrevesu (in nobena, ki se ne nahaja v njem!), začenja na niz p; takih končnic je, kot vemo, $f(v)$, to pa je potemtakem tudi število pojavitev p-ja v s-ju.

Če torej hočemo najti vse nize, ki se pojavljajo v s-ju vsaj k-krat, se moramo sprehajati od korena drevesa navzdol v vse smeri, dokler ne pridemo do kakšnega vozlišča, ki ima $f(v) < k$; tam se ustavimo. Za vsa vozlišča, ki pa imajo $f(v) \geq k$, pa lahko izpišemo niz, ki ga tvorijo oznake povezav od korena do vozlišča v. No, preden ga izpišemo, preverimo še, če je ta niz dolg vsaj m znakov, saj naloga pravi, da nas krajši nizi ne zanimajo.

Opisani postopek prikazuje spodnji podprogram, ki pa ne predpostavlja, da so vrednosti $f(v)$ že izračunane in shranjene v drevesu; zato jih sproti računa sam, kar pa med drugim pomeni, da mora v vsakem primeru pregledati celotno drevo.

```
const MaxOtrok = 5; { C, G, A, T, # }
type VozlisceP = ↑VozlisceT;
     VozlisceT = record
       Oznaka: char; { oznaka na povezavi, ki kaže na to vozlišče }
       StOtrok: integer; { število otrok }
       Otroci: array [1..MaxOtrok] of VozlisceP; { kazalci na otroke }
     end; {VozlisceT}

procedure IzpisiPogostePodnize(Koren: VozlisceP; MinPogostost, MinDolzina: integer);

  function Rekurzija(V: VozlisceP; PotDoslej: string): integer;
  var i, StKoncnic: integer; Otrok: VozlisceP;
  begin
    if V↑.StOtrok = 0 then begin Rekurzija := 1; exit end;
    StKoncnic := 0;
    for i := 1 to V↑.StOtrok do begin
      Otrok := V↑.Otroci[i];
      StKoncnic := StKoncnic + Rekurzija(Otrok, PotDoslej + Otrok↑.Oznaka);
    end; {for i}
    if StKoncnic >= MinPogostost then
      if Length(PotDoslej) >= MinDolzina then WriteLn(PotDoslej, ' (', StKoncnic, ')');
    Rekurzija := StKoncnic;
  end; {Rekurzija}
```

```
begin {IzpisiPogostePodnize}
  if Koren <> nil then Rekurzija(Koren, '');
end; {IzpisiPogostePodnize}
```

V gornji rešitvi podprogram Rekurzija pregleda poddrevo, ki se začne v vozlišču V, in vrne število listov v njem. V parametru PotDoslej mu je treba podati niz, dobljen s stikanjem oznak vseh povezav na poti od korena do vozlišča V. Če je V list, vemo, da je oznaka na povezavi do njega kar posebni znak #, torej se PotDoslej tudi konča na # in je nima smisla izpisovati, saj to ni podniz tistega prvotnega niza, ki uporabnika zares zanima (torej tistega pred dodajanjem znaka #). Če pa je V notranje vozlišče, moramo z rekurzivnimi klici ugotoviti, koliko listov je v njegovem poddrevesu; to pa nam tudi pove, kolikokrat se PotDoslej pojavi v prvotnem nizu. Če je to število pojavitev dovolj veliko in če je niz PotDoslej dovolj dolg, ga izpišemo.

Še rešitev v C-ju:

```
#define MaxOtrok 5
typedef struct VozlisceT_{
  char oznaka; int stOtrok;
  struct VozlisceT_*otroci[MaxOtrok], *oce;
} VozlisceT, * VozlisceP;
void Izpisi(VozlisceP p, int eol) /* Izpiše oznake po poti od korena do p. */
{
  if (p->oce == 0) return; /* p je koren drevesa */
  Izpisi(p->oce, 0); putchar(p->oznaka); if (eol) putchar('\n');
}
int Rekurzija(VozlisceP p, int minPogostost, int minDolzina, int globina)
{
  int i, stListov = 0;
  if (p->stOtrok == 0) return 1;
  for (i = 0; i < p->stOtrok; i++) if (p->otroci[i])
    stListov += Rekurzija(p->otroci[i], minPogostost, minDolzina, globina + 1);
  if (stListov >= minPogostost && globina >= minDolzina) Izpisi(p, 1);
  return stListov;
}
void IzpisiPogostePodnize(VozlisceP koren, int minPogostost, int minDolzina)
{
  if (koren) Rekurzija(koren, minPogostost, minDolzina, 0);
}
```

5. Razvajeni zvezdniki

Naloga pravi, da je vsak zvezdnik navedel dve množici in kot pogoj za svoje sodelovanje zahteval, da v oddaji sodelujejo vsi zvezdniki iz prve množice in nobeden od zvezdnikov iz druge množice. Označimo pri zvezdniku i prvo od teh dveh množic z A_i, drugo pa z B_i.

Če bi radi v oddajo vključili zvezdnike $\{1, 2, \ldots, k\}$, iz gornjih omejitev sledi, da morajo biti v oddajo vključeni tudi vsi zvezdniki iz $A_1 \cup A_2 \cup \ldots \cup A_k$ (označimo to unijo z $\hat{A}_k$) in nobeden od zvezdnikov iz $B_1 \cup B_2 \cup \ldots \cup B_k$ (označimo to unijo z $\hat{B}_k$). Da bomo res lahko uporabili le zvezdnike od 1 do k in nobenih drugih, mora biti torej množica zahtevanih zvezdnikov $\hat{A}_k$ podmnožica množice $\{1, \ldots, k\}$;

množica „prepovedanih“ zvezdnikov $\hat{B}_k$ pa ne sme vsebovati nobenega od zvezdnikov iz $\{1, \ldots, k\}$.

Pogoj, da je $\hat{A}_k$ podmnožica množice $\{1, \ldots, k\}$, je enakovreden pogoju, da je največji element množice $\hat{A}_k$ manjši ali enak k. Podobno tudi vidimo, da je pogoj, da $\hat{B}_k$ nima skupnih elementov z množico $\{1, \ldots, k\}$, enakovreden pogoju, da je najmanjši element množice $\hat{B}_k$ večji od k.

Naš postopek lahko pregleduje različne k-je v naraščajočem vrstnem redu. Ko se k povečuje, pridobivata množici $\hat{A}_k$ in $\hat{B}_k$ nove elemente, starih pa nikoli ne izgubljata; zato se najmanjši element $\hat{B}_k$ sčasoma zgolj zmanjšuje. Zato, če pri nekem k opazimo, da je najmanjši element množice $\hat{B}_k$ manjši ali enak k, bo to veljalo tudi pri vseh večjih k (ker se bo tisti najmanjši element le še zmanjševal, k pa se bo povečeval); čim je torej ta pogoj prekršen, vemo, da niti prvih k zvezdnikov na otok ne more odpotovati skupaj, še toliko manj pa lahko zato potuje skupaj kakšna večja skupina zvezdnikov, ki vsebuje vseh prvih k zvezdnikov. Zato lahko v takem primeru naš postopek takoj prekinemo.

Če pa s tem pogojem ni težav, nam ostane le še pogoj, da je največji element množice $\hat{A}_k$ manjši ali enak k; če je to res, pomeni, da oddajo lahko organiziramo z zvezdniki $\{1, \ldots, k\}$. Med k-ji, ki ustrezajo temu pogoju, si zapomnimo največjega; to je rešitev naše naloge.

Zdaj tudi vidimo, da množic $\hat{A}_k$ in $\hat{B}_k$ sploh ni treba shranjevati v celoti, ampak je dovolj že, če si zapomnimo največji element $\hat{A}_k$ (MaxZaht v spodnjem programu) in najmanjši element $\hat{B}_k$ (MinPrep v spodnjem programu).

```
var z, k, MaxK, MaxZaht, MinPrep: integer;
begin
  MaxK := 0; MaxZaht := 0; MinPrep := n + 1;
  for k := 1 to n do begin
    za vsakega z ∈ B_k do if z < MinPrep then MinPrep := z;
    if MinPrep <= k then break;
    za vsakega z ∈ A_k do if z > MaxZaht then MaxZaht := z;
    if MaxZaht <= k then MaxK := k;
  end; {for k}
  WriteLn(MaxK);
end.
```

Enak postopek v C-ju:

```
int main()
{
  int z, k, MaxK = 0, MaxZaht = 0, MinPrep = n + 1;
  for (k = 1; k <= n; k++) {
    for (za vsakega z ∈ B_k) if (z < MinPrep) MinPrep = z;
    if (MinPrep <= k) break;
    for (za vsakega z ∈ A_k) if (z > MaxZaht) MaxZaht = z;
    if (MaxZaht <= k) MaxK = k; }
  printf("%d\n", MaxK); return 0;
}
```

REŠITVE NALOG ZA TRETJO SKUPINO

1. Optična miška

Ker so naše slike majhne, je dovolj dobra že najpreprostejša rešitev: pregledamo vse možne zamike, torej vse pare (dx, dy), za katere sta dx in dy med -3 in $+3$ (takih parov je $7 \times 7 = 49$) in pri vsakem pregledamo vse slikovne elemente, ki so pri tem zamiku vidni na obeh slikah, stari in novi. Točka, ki jo na stari sliki vidimo na koordinatah (x, y), je na novi sliki vidna na koordinatah $(x - dx, y - dy)$. Na primer: če se miška premakne v desno ($dx > 0$), se stvari na sliki premaknejo v levo, torej k manjšim x-koordinatam; zato tisto, kar smo prej videli pri x, zdaj vidimo $x - dx$. Podobno je s premiki v navpični smeri.

V poštev pridejo seveda le tisti (x, y) s stare slike, ki so pri trenutnem zamiku tudi zares vidni na novi sliki. Za koordinati na novi sliki, $(x-dx, y-dy)$, mora torej veljati: $0 \le x-dx < n$ in $0 \le y-dy < n$, če imamo opravka s slikami velikosti $n \times n$. Zanimale nas bodo torej x-koordinate od $\max\{0, -dx\}$ do $\min\{n-1, n+dx-1\}$, podobno pa je tudi pri y-koordinatah.

Pri vsakem slikovnem elementu, ki ga vidimo na obeh slikah, izračunajmo absolutno vrednost razlike v svetlosti tega elementa na obeh slikah. Na koncu nas bo zanimalo povprečje teh absolutnih vrednosti po vseh slikovnih elementih, ki se jih vidi na obeh slikah. Da ne bomo imeli težav s kakšnimi zaokrožitvenimi napakami, povprečij ne bomo izračunali eksplicitno, ampak jih bomo hranili v obliki ulomka a/b, pri čemer je a vsota absolutnih vrednosti razlik, b pa število slikovnih elementov, ki so prisotni na obeh slikah. Če je najmanjše doslej znano povprečje a^*/b^*, moramo potem pri vsakem naslednjem zamiku preveriti, če je a/b mogoče manjše od a^*/b^*. Pogoj $a/b < a^*/b^*$ lahko preoblikujemo v $ab^* < a^*b$ in se tako izognemo deljenju.

```
program OpticnaMiska;
const cxMax = 64; cyMax = 64; dxMax = 3; dyMax = 3;
var cx, cy, dx, dy, dxNaj, dyNaj, x, y: integer; T, U: text;
    OcenaA, OcenaB, NajA, NajB, StPrimerov: integer;
    SP, SN: array [0..cyMax - 1, 0..cxMax - 1] of integer;
begin
  Assign(T, 'miska.in'); Reset(T); ReadLn(T, StPrimerov);
  Assign(U, 'miska.out'); Rewrite(U);
  while StPrimerov > 0 do begin
    ReadLn(T); StPrimerov := StPrimerov - 1; Read(T, cx); cy := cx;
    { Preberimo obe sliki. }
    ReadLn(T); for y := 0 to cy - 1 do
      begin for x := 0 to cx - 1 do Read(T, SP[y, x]);   ReadLn(T) end;
    ReadLn(T); for y := 0 to cy - 1 do
      begin for x := 0 to cx - 1 do Read(T, SN[y, x]);   ReadLn(T) end;
    { Primerjajmo sliki in določimo premik. }
    NajA := -1; NajB := -1;
    for dy := - dyMax to dyMax do for dx := - dxMax to dxMax do begin
      OcenaA := 0; OcenaB := 0;
      y := dy; if y < 0 then y := 0;
      while (y < cy) and (y - dy < cy) do begin
        x := dx; if x < 0 then x := 0;
        while (x < cx) and (x - dx < cx) do begin
```

```
            OcenaA := OcenaA + Abs(SP[y, x] − SN[y − dy, x − dx]);
            OcenaB := OcenaB + 1; x := x + 1;
          end; {while x}
          y := y + 1;
        end; {while y}
        { Ocena trenutnega premika je OcenaA / OcenaB, ocena najboljšega doslej znanega
          pa je NajA / NajB. Če je trenutni boljši, si ga zapomnimo. }
        if (NajB < 0) or (OcenaA * NajB < OcenaB * NajA) then
          begin NajA := OcenaA; NajB := OcenaB; dxNaj := dx; dyNaj := dy end;
      end; {for dx, dy}
      WriteLn(U, dxNaj, ' ', dyNaj);
    end; {while}
    Close(T); Close(U);
end. {OpticnaMiska}
```

Še program v C-ju:

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

#define cxMax 64
#define cyMax 64

int main()
{
  FILE *f = fopen("miska.in", "rt"), *g = fopen("miska.out", "wt");
  int StPrimerov, x, y, cx, cy, dx, dy, dxNaj, dyNaj;
  int OcenaA, OcenaB, NajA, NajB, SP[cyMax][cxMax], SN[cyMax][cxMax];

  fscanf(f, "%d", &StPrimerov);
  while (StPrimerov−− > 0)
  {
    /* Preberimo obe sliki. */
    fscanf(f, "%d", &cx); cy = cx;
    for (y = 0; y < cy; y++) for (x = 0; x < cx; x++) fscanf(f, "%d", &SP[y][x]);
    for (y = 0; y < cy; y++) for (x = 0; x < cx; x++) fscanf(f, "%d", &SN[y][x]);

    /* Primerjajmo obe sliki pri vseh možnih premikih. */
    NajA = −1; NajB = −1;
    for (dy = −3; dy <= 3; dy++) for (dx = −3; dx <= 3; dx++) {
      OcenaA = 0; OcenaB = 0;
      for (y = dy < 0 ? 0 : dy; y < cy && y − dy < cy; y++)
      for (x = dx < 0 ? 0 : dx; x < cx && x − dx < cx; x++)
        OcenaA += abs(SP[y][x] − SN[y − dy][x − dx]), OcenaB++;

      /* Ali je trenutni premik boljši od najboljšega doslej znanega? */
      if (NajA < 0 || OcenaA * NajB < OcenaB * NajA)
        NajA = OcenaA, NajB = OcenaB, dxNaj = dx, dyNaj = dy; }

    /* Izpišimo najverjetnejši premik. */
    fprintf(g, "%d %d\n", dxNaj, dyNaj);
  }
  fclose(f); fclose(g); return 0;
}
```

Postopki, ki jih miške res uporabljajo za ugotavljanje premikov, so omenjeni npr. v Wikipedijinem članku "Optical Flow".

2. Spletne knjigarne

Če bi knjigarne zaračunavale enako poštnino za prvo knjigo kot za vse ostale, bi

bila ta naloga zelo preprosta. Dovolj bi bilo, če bi za vsako knjigo j in vsako knjigarno i pogledali, koliko bi nas stala ta knjiga z dostavo vred, če bi jo naročili pri tej knjigarni: to je skupaj $b_i + c_{ji}$ enot. Vsako knjigo bi potem naročili pri tisti knjigarni, kjer je ta vsota najmanjša.

Pri naši nalogi pa so stvari malo bolj zapletene, ker je poštnina za prvo knjigo pri knjigarni i lahko dražja, namreč a_i namesto b_i. Na ta strošek lahko pogledamo tudi takole: za to, da bomo od knjigarne i sploh lahko kaj naročili, ji moramo plačati konstantno vsoto $a_i - b_i$ denarja; potem pa bomo za vsako naročeno knjigo (tudi prvo) plačali tej knjigarni še b_i denarnih enot za poštnino.

Vnaprej je težko reči, od katerih knjigarn se splača naročati knjige. Mogoče ima neka knjigarna sicer ugodno nizke zneske $b_i + c_{ji}$, vendar ima tako visok $a_i - b_i$, da je bolje, če od nje sploh ničesar ne naročamo. Ker je knjigarn malo, si lahko privoščimo pregledati kar vse možne množice knjigarn. Pri k knjigarnah imamo $2^k - 1$ nepraznih množic knjigarn. Za vsako tako množico M si mislimo, da smo se že sprijaznili s tem, da bomo od vsake od teh knjigarn mogoče kaj naročili, kar nas bo skupaj stalo $\sum_{i \in M}(a_i - b_i)$ enot denarja. Potem moramo ugotoviti le še to, pri kateri od teh knjigarn je posamezna knjiga najcenejša (z dostavo vred). Tako bomo za knjige plačali še nadaljnjih $\sum_{j=1}^{n} \min_{i \in M}(b_i + c_{ji})$ enot denarja.

Ker bomo sčasoma pregledali vse možne množice knjigarn, bomo prej ali slej prišli tudi do tiste, pri kateri je res dosežena optimalna rešitev. Spodnji program predstavi množico M kar s celim številom, v katerem bit $i-1$ pove, ali je knjigarna i v tej množici ali ne. V tabeli Cj si pripravi vrednosti $\min_{i \in M}(b_i + c_{ji})$, ki povedo najmanjšo ceno (z dostavo vred) za posamezno knjigo j, gledano po vseh knjigarnah i iz M.

```
program SpletneKnjigarne;
const MaxK = 10; MaxN = 10000; MaxCena = 10000;
var Ai, Bi: array [1..MaxK] of integer;
    Cji: array [1..MaxN, 1..MaxK] of integer;
    Cj: array [1..MaxN] of integer;
    i, j, k, n, M, Cena, NajCena: integer; T: text;
begin
  { Preberimo vhodno datoteko. }
  Assign(T, 'knjigarne.in'); Reset(T); ReadLn(T, n, k);
  for i := 1 to k do Read(T, Ai[i]);  ReadLn(T);
  for i := 1 to k do Read(T, Bi[i]);  ReadLn(T);
  for j := 1 to n do
    begin for i := 1 to k do Read(T, Cji[j, i]);  ReadLn(T) end;

  { Preglejmo vse neprazne množice knjigarn. }
  NajCena := (MaxCena + 1) * (n + n + k);
  for M := 1 to (1 shl k) - 1 do begin
    Cena := 0;
    for j := 1 to n do Cj[j] := 2 * MaxCena + 1;
    for i := 1 to k do if (M and (1 shl (i - 1))) <> 0 then begin
      Cena := Cena + Ai[i] - Bi[i];
      for j := 1 to n do if Cj[j] > Bi[i] + Cji[j, i] then
        Cj[j] := Bi[i] + Cji[j, i];
    end; {for i}
    for j := 1 to n do Cena := Cena + Cj[j];
    if Cena < NajCena then NajCena := Cena;
  end; {for M}
```

```
  { Izpišimo rezultat. }
  Assign(T, 'knjigarne.out'); Rewrite(T); WriteLn(T, NajCena); Close(T);
end. {SpletneKnjigarne}
```

Še program v C-ju:

```
#include <stdio.h>

#define MaxK 10
#define MaxN 10000
#define MaxCena 10000

int main()
{
  int Ai[MaxK], Bi[MaxK], Cji[MaxN][MaxK], Cj[MaxN];
  int i, j, k, n, M, Kand, Cena, NajCena; FILE *f;

  /* Preberimo vhodno datoteko. */
  f = fopen("knjigarne.in", "rt");
  fscanf(f, "%d %d", &n, &k);
  for (i = 0; i < k; i++) fscanf(f, "%d", &Ai[i]);
  for (i = 0; i < k; i++) fscanf(f, "%d", &Bi[i]);
  for (j = 0; j < n; j++) for (i = 0; i < k; i++) fscanf(f, "%d", &Cji[j][i]);
  fclose(f);

  /* Preglejmo vse neprazne množice knjigarn. */
  NajCena = (MaxCena + 1) * (n + n + k);
  for (M = 1; M < (1 << k); M++)
  {
    for (j = 0; j < n; j++) Cj[j] = 2 * MaxCena + 1;
    for (i = 0, Cena = 0; i < k; i++) if ((M >> i) & 1) {
      Cena += Ai[i] - Bi[i];
      for (j = 0; j < n; j++) if ((Kand = Bi[i] + Cji[j][i]) < Cj[j]) Cj[j] = Kand; }
    for (j = 0; j < n; j++) Cena += Cj[j];
    if (Cena < NajCena) NajCena = Cena;
  }

  /* Izpišimo rezultat. */
  f = fopen("knjigarne.out", "wt"); fprintf(f, "%d\n", NajCena); fclose(f); return 0;
}
```

3. Izravnavanje histogramov

Naj bo h_c (za $0 \le c \le 255$) število slikovnih elementov (pikslov) barve c na vhodni sliki I, h'_c pa število slikovnih elementov barve c na izhodni sliki I'. Histogram $h = (h_0, \ldots, h_{255})$ lahko določimo s pregledom vhodne slike I, histogram $h' = (h'_0, \ldots, h'_{255})$ pa določimo na podlagi zahteve, da se smeta najvišji in najnižji stolpec razlikovati največ za 1. Če ima naša slika vsega skupaj n slikovnih elementov, lahko za začetek postavimo vsak stolpec histograma na višino n div 256. Tako smo že porabili $256 \cdot (n \text{ div } 256)$ od vseh n slikovnih elementov; ostalo jih je še n mod 256, kar je manj kot 256. Torej lahko n mod 256 stolpcev dvignemo za 1 in tako porabimo vse slikovne elemente, naš izhodni histogram h' pa še vedno ustreza zahtevi, da najvišji stolpec ni več kot za 1 višji od najnižjega.

Recimo, da smo že našli primerno sliko I', ki ima histogram h' in ustreza tudi drugi od zahtev naše naloge, torej: če je nek slikovni element na sliki I svetlejši od nekega drugega, mora biti na sliki I' tudi svetlejši ali pa iste barve, nikakor pa ne temnejši. Ta zahteva v bistvu pravi, da morajo najtemnejši slikovni elementi

vhodne slike postati tudi najtemnejši elementi izhodne slike, najsvetlejši vhodne slike pa najsvetlejši izhodne slike in podobno.

Mislimo si, da bi za vsak slikovni element naše slike zapisali urejen par $(b_\bullet, b'_\bullet)$, pri čemer je $b_\bullet$ barva tega elementa na vhodni sliki, $b'_\bullet$ pa na izhodni sliki. Vse tako urejene pare uredimo naraščajoče po $b_\bullet$, znotraj vsake vrednosti $b_\bullet$ pa še naraščajoče po $b'_\bullet$. V tako urejenem zaporedju pare oštevilčimo od 0 do $n-1$; i-ti par je torej (b_i, b'_i). Zaradi načina urejanja vemo, da je zaporedje $b = (b_0, \ldots, b_{n-1})$ nepadajoče. Toda ker I' ustreza zahtevam naloge, vemo, da je tudi zaporedje $b' = (b'_0, \ldots, b'_{n-1})$ nepadajoče: če to ne bi bilo res, bi bil torej nekje nek $b'_i > b'_{i+1}$, kar pa je zaradi našega načina urejanja zaporedja parov $(b_\bullet, b'_\bullet)$ možno le, če sta b_i in b_{i+1} različna; to pa je možno le v obliki $b_i < b_{i+1}$, saj smo zaporedje parov uredili po $b_\bullet$-jih. Tedaj bi torej imeli dva slikovna elementa, od katerih bi bil prvi na vhodni sliki temnejši od drugega ($b_i < b_{i+1}$), na drugi pa svetlejši ($b'_i > b'_{i+1}$), kar bi bilo v protislovju z zahtevo iz naloge.

Ker poznamo histograma obeh slik, seveda tudi vemo, da je prvih h_0 členov zaporedja b enakih 0, naslednjih h_1 členov tega zaporedja je enakih 1 in tako naprej; podobno je prvih h'_0 členov zaporedja b' enakih 0, naslednjih h'_1 členov je enakih 1 in tako naprej. Obe zaporedji torej pravzaprav že v celoti poznamo in ju lahko v mislih napišemo eno nad drugim, na primer takole:

	$h_0 = 3$			$h_1 = 1$	$h_2 = 7$														
b	0	0	0	1	2	2	2	2	2	2	2	3	·	·	·	·	·	·	255
b'	0	0	0	1	1	1	2	2	2	3	3	3	·	·	·	·	·	·	255
	$h'_0 = 3$			$h'_1 = 3$			$h'_2 = 3$			$h'_3 = 3$									

Obenem pa zaradi načina, kako smo do teh zaporedij prišli, še vedno velja, da vsak par istoležnih elementov (b_i, b'_i) opisuje spremembo barve v enem od slikovnih elementov naše slike (torej pove, da se je nek slikovni element barve b_i spremenil v barvo b'_i). Ker je na vhodni sliki h_0 slikovnih elementov barve 0, imamo v našem zaporedju b vrednosti $b_i = 0$ pri $i = 0, 1, 2, \ldots, h_0 - 1$, pripadajoče vrednosti b'_i pa nam povedo, kako moramo pobarvati te slikovne elemente na izhodni sliki. Naslednjih h_1 členov zaporedja b (torej za $i = h_0, \ldots, h_0 + h_1 - 1$) ima $b_i = 1$ in pripadajoče vrednosti b'_i nam povedo, kako pobarvati na izhodni sliki tiste slikovne elemente, ki so na vhodni sliki barve 1. Tako lahko nadaljujemo, dokler ne pobarvamo cele izhodne slike:

```
1   pripravi zaporedji b in b', kot je bilo opisano zgoraj;
2   i := 0;
3   za vsako barvo c od 0 do 255 ponovi:
4       za vsak slikovni element (x, y), ki je na stari sliki I barve c:
5           vemo, da je b_i = c; na izhodni sliki pobarvaj ta element z barvo b'_i;
6           i := i + 1;
```

Pri tem postopku že vidimo, da zaporedja b ni treba hraniti ekspliticno, saj ga pravzaprav sploh nikjer ne uporabljamo. Še lepše pa je, da tudi zaporedja b' ni treba hraniti eksplicitno, saj dostopamo do njega po naraščajočih indeksih i. Ker vemo, da ima prvih h'_0 členov tega zaporedja vrednost 0, naslednjih h'_1 členov vrednost 1

in tako naprej, je dovolj že, če si zapomnimo, na katerem od teh intervalov leži trenutni i in kako daleč mu še manjka do konca tega intervala. Tako pridemo do naslednjega postopka:

```
1    i := 0; c' := 0; ν := h'_0;
2    za vsako barvo c od 0 do 255 ponovi:
3        za vsak slikovni element (x, y), ki je na stari sliki I barve c:
4            vemo, da je indeks i v zaporedju b na območju barve c
                 (torej da je b_i = c);
5            vemo tudi, da je indeks i v zaporedju b' na območju barve c'
                 in da se naslednje območje začne pri indeksu i + ν;
6            while ν = 0 do begin c' := c' + 1; ν := h'_c' end;
7            pobarvaj (x, y) na izhodni sliki I' z barvo c';
8            i := i + 1; ν := ν − 1;
```

Notranja zanka **while** torej poskrbi za to, da c' povečamo, ko pridemo v zaporedju b' do indeksa, kjer se vrednost členov tega zaporedja poveča. Tako je c' po koncu te zanke zagotovo enak b'_i.

Gornji postopek bi načeloma za vsako barvo od 0 do 255 izvedel po en prehod čez celo sliko, da bi našel slikovne elemente te barve in jih ustrezno prebarval. Še elegantnejši in hitrejši postopek pa dobimo, če vse barve prebarvamo istočasno. Gornji postopek, ki pregleduje barve eno za drugo od temnejših k svetlejšim, bi imel opravka z barvo c takrat, ko bi imel števec i vrednosti od $h_0 + h_1 + \ldots + h_{c-1}$ do $h_0 + h_1 + \ldots + h_{c-1} + (h_c - 1)$. Če bi torej znali za vsakega od začetnih indeksov $h_0 + \ldots + h_{c-1}$, pri katerih se začne naše delo s slikovnimi elementi barve c, ugotoviti, kakšna je takrat v gornjem postopku vrednost spremenljivk c' in ν (recimo tema vrednostma $c'[c]$ in $\nu[c]$), bi lahko potem v enem samem prehodu čez sliko vzporedno izvajali vseh 256 iteracij glavne zanke gornjega postopka — ko naletimo na slikovni element barve c, izvedemo eno iteracijo notranje zanke (vrstice 4–8) znotraj tiste iteracije zunanje zanke, ki se nanaša na to barvo c.

Do vrednosti $c'[c]$ in $\nu[c]$ pa tudi ni težko priti; še enkrat moramo v mislih izvesti gornji postopek, le da iz njega pobrišemo vse ukvarjanje s konkretnimi slikovnimi elementi in iskanje le-teh po sliki: zdaj je pomembno le, kolikokrat bi se notranja zanka (vrstice 4–8) izvedla pri posameznem c in kako bi se zaradi tega spremenili vrednosti c' in ν. Tako pridemo do naslednjega postopka za inicializacijo tabel $c'[c]$ in $\nu[c]$:

```
1    c' := 0; ν := h'_0;
2    za vsako barvo c od 0 do 255 ponovi:
3        c'[c] := c'; ν[c] := ν;
4        ν := ν − h_c;
5        while ν ≤ 0 and c' < 255 do begin c' := c' + 1; ν := ν + h'_c' end;
```

Barvanje stare slike v novo lahko zdaj poteka takole:

```
1    za vsak slikovni element (x, y):
2        naj bo c barva tega elementa na stari sliki;
3        while ν[c] = 0 do begin c'[c] := c'[c] + 1; ν[c] := h'_c'[c] end;
4        pobarvaj (x, y) na izhodni sliki I' z barvo c'[c];
5        ν[c] := ν[c] − 1;
```

Tako smo dobili učinkovit in eleganten postopek, ki potrebuje le dva prehoda po sliki (enega, da določi histogram h, in drugega, da sliko prebarva oz. ustvari izhodno sliko) in ima tako na sliki z n slikovnimi elementi in B barvami (v našem primeru je $B = 256$) časovno zahtevnost $O(n + B)$.

Zapišimo našo rešitev še v obliki programa. Tabeli Hist in CiljHist hranita histograma h in h', tabeli Nova in NovaKoliko pa hranita vrednosti $c'[c]$ in $\nu[c]$.

```
program IzravnavanjeHistograma;
const MaxSirina = 200; MaxVisina = 200;
var T: text;
    Slika: array [0..MaxVisina − 1, 0..MaxSirina − 1] of integer;
    Hist, CiljHist, Nova, NovaKoliko: array [0..255] of integer;
    x, y, c, cNova, Koliko, Sirina, Visina: integer;
begin
  { Preberimo prvotno sliko. }
  Assign(T, 'histogram.in'); Reset(T); ReadLn(T, Visina, Sirina);
  for y := 0 to Visina − 1 do for x := 0 to Sirina − 1 do Read(T, Slika[y, x]);
  Close(T);
  { Izračunajmo histogram prvotne slike. }
  for c := 0 to 255 do Hist[c] := 0;
  for y := 0 to Visina − 1 do for x := 0 to Sirina − 1 do
    Hist[Slika[y, x]] := Hist[Slika[y, x]] + 1;
  { V tabeli CiljHist pripravimo enega od možnih histogramov nove slike. Druge možne
    histograme, ki bi tudi ustrezali zahtevam naloge, bi lahko dobili tako, da bi
    druga zanka povečala za 1 kakšne druge elemente, ne pa ravno prvih nekaj. }
  for c := 0 to 255 do CiljHist[c] := (Sirina * Visina) div 256;
  for c := 0 to ((Sirina * Visina) mod 256) − 1 do CiljHist[c] := CiljHist[c] + 1;
  { Vsaka barva prvotne slike se preslika v eno ali več zaporednih barv nove slike.
    Izračunajmo za začetek najtemnejšo barvo Nova[c], v katero se bo preslikala posamezna
    barva c prvotne slike, pa še to, koliko pikslov barve c se bo preslikalo v barvo Nova[c]. }
  cNova := 0; Koliko := CiljHist[cNova];
  for c := 0 to 255 do begin
    Nova[c] := cNova; NovaKoliko[c] := Koliko;
    Koliko := Koliko − Hist[c];
    while (Koliko <= 0) and (cNova < 255) do
      begin cNova := cNova + 1; Koliko := Koliko + CiljHist[cNova] end;
  end; {for c}
  { Pojdimo po sliki in popravljajmo barve pikslov. Barva c stare slike postane barva Nova[c]
    na novi sliki. Če pa ima ta barva že dovolj pikslov, začnimo uporabljati naslednjo. }
  for y := 0 to Visina − 1 do for x := 0 to Sirina − 1 do begin
    c := Slika[y, x];
    while NovaKoliko[c] = 0 do
      begin Nova[c] := Nova[c] + 1; NovaKoliko[c] := CiljHist[Nova[c]] end;
    Slika[y, x] := Nova[c]; NovaKoliko[c] := NovaKoliko[c] − 1;
  end; {for x, y}
  { Izpišimo rezultat. }
  Assign(T, 'histogrami.out'); Rewrite(T);
  for y := 0 to Visina − 1 do for x := 0 to Sirina − 1 do begin
    Write(T, Slika[y, x]); if x = Sirina − 1 then WriteLn(T) else Write(T, ' ');
  end; {for x, y}
  Close(T);
end. {IzravnavanjeHistograma}
```

Še program v C-ju:

```
#include <stdio.h>

#define MaxSirina 200
#define MaxVisina 200
#define StBarv 256

int main()
{
  int Slika[MaxVisina][MaxSirina];
  int Hist[StBarv], CiljHist[StBarv], Nova[StBarv], NovaKoliko[StBarv];
  int x, y, c, cNova, Koliko, Sirina, Visina; FILE *f;

  /* Preberimo prvotno sliko. */
  f = fopen("histogram.in", "rt");
  fscanf(f, "%d %d", &Visina, &Sirina);
  for (y = 0; y < Visina; y++) for (x = 0; x < Sirina; x++)
    fscanf(f, "%d", &Slika[y][x]);
  fclose(f);

  /* Izračunajmo histogram prvotne slike. */
  for (c = 0; c < StBarv; c++) Hist[c] = 0;
  for (y = 0; y < Visina; y++) for (x = 0; x < Sirina; x++) Hist[Slika[y][x]]++;

  /* Določimo ciljni (izravnani) histogram. */
  for (c = 0; c < StBarv; c++) CiljHist[c] = (Sirina * Visina) / StBarv;
  for (c = 0; c < (Sirina * Visina) % StBarv; c++) CiljHist[c]++;

  /* Za vsako barvo c naj bo Nova[c] najtemnejša barva, v katero se preslikajo
     piksli te barve, in sicer naj se jih vanjo preslika NovaKoliko[c]. */
  cNova = 0; Koliko = CiljHist[cNova];
  for (c = 0; c < StBarv; c++) {
    Nova[c] = cNova; NovaKoliko[c] = Koliko;
    Koliko -= Hist[c];
    while (Koliko <= 0 && cNova < StBarv)
      Koliko += CiljHist[++cNova]; }

  /* Prebarvajmo prvotno sliko in ji s tem izravnajmo histogram. */
  for (y = 0; y < Visina; y++) for (x = 0; x < Sirina; x++) {
    c = Slika[y][x];
    while (NovaKoliko[c] == 0)
      NovaKoliko[c] = CiljHist[++Nova[c]];
    Slika[y][x] = Nova[c]; NovaKoliko[c]--; }

  /* Izpišimo rezultat. */
  f = fopen("histogram.out", "wt");
  for (y = 0; y < Visina; y++) for (x = 0; x < Sirina; x++)
    fprintf(f, "%d%c", Slika[y][x], (int) (x == Sirina - 1 ? '\n' : ' '));
  fclose(f); return 0;
}
```

4. Mafija

Za vsakega člana družine vodimo podatek o razliki med njegovimi prejemki in tem, kar je dal nadrejenemu. Vsaka vrstica vhodne datoteke nam pove, da je član i dal svojemu nadrejenemu (recimo mu j) nek znesek denarja; ta znesek torej prištejemo j-jevemu elementu tabele in odštejemo od i-jevega elementa. Na koncu imamo tako v tej tabeli za vsakega člana ravno količino denarja, ki ga je utajil. Poiskati moramo največji element tabele in ga izpisati; pri tem pa moramo paziti, da ne upoštevamo

elementa, ki se nanaša na šefa družine: on nima nadrejenega, zato le prejema in v naši tabeli utaj se mu lahko nabere zelo velik znesek (ki pa seveda ne pomeni utaje).

```
program Mafija;
const MaxN = 100000;
var Utajil: array [1..MaxN] of integer;
    i, j, Znesek, n, MaxUtaja, Sef: integer; T: text;
begin
  { Preberimo vhodne podatke in izračunajmo, koliko je kdo utajil. }
  Assign(T, 'mafija.in'); Reset(T); ReadLn(T, n);
  for i := 1 to n do Utajil[i] := 0;
  for i := 1 to n do begin
    ReadLn(T, j, Znesek); Assert(0 <= Znesek);
    if j = 0 then Sef := i
    else begin Utajil[i] := Utajil[i] - Znesek; Utajil[j] := Utajil[j] + Znesek end;
  end; {for i}
  Close(T);
  { Poiščimo največjo utajo. }
  MaxUtaja := 0;
  for i := 1 to n do if (i <> Sef) and (Utajil[i] > MaxUtaja) then MaxUtaja := Utajil[i];
  { Izpišimo rezultat. }
  Assign(T, 'mafija.out'); Rewrite(T); WriteLn(T, MaxUtaja); Close(T);
end. {Mafija}
```

Še rešitev v C-ju:

```
#include <stdio.h>
#define MaxN 100000
int main()
{
  int i, j, Znesek, n, MaxUtaja, Sef, Utajil[MaxN];
  /* Preberimo vhodne podatke in izračunajmo, koliko je kdo utajil. */
  FILE *f = fopen("mafija.in", "rt"); fscanf(f, "%d", &n);
  for (i = 0; i < n; i++) Utajil[i] = 0;
  for (i = 0; i < n; i++) {
    fscanf(f, "%d %d", &j, &Znesek);
    if (j == 0) Sef = i;
    else Utajil[i] -= Znesek, Utajil[j - 1] += Znesek; }
  fclose(f);
  /* Poiščimo največjo utajo. */
  for (i = 0, MaxUtaja = 0; i < n; i++)
    if (i != Sef && Utajil[i] > MaxUtaja) MaxUtaja = Utajil[i];
  /* Izpišimo rezultat. */
  f = fopen("mafija.out", "wt"); fprintf(f, "%d\n", MaxUtaja); fclose(f); return 0;
}
```

5. Tovornjaki

Pri tej nalogi ni nujno priti do najboljše možne rešitve; dovolj za nekaj točk je že, da se ji vsaj čim bolj približamo. Izkaže se, da lahko blizu dobrim rešitvam pridemo že z naključnim razporejanjem tovornjakov na pasove. Vsak tovornjak naključno razporedimo na enega od treh pasov in na koncu pogledamo, kako dolg trajekt

potrebujemo. To velikokrat ponovimo in na koncu izpišemo tisti razpored, ki je dal najkrajši trajekt.

```
program Tovornjaki;
const MaxStTov = 100; StPasov = 3;
var
  Dolzina, Pas, NajPas: array [1..MaxStTov] of integer;
  DolPasu: array [1..StPasov] of integer;
  Trajekt, NajTrajekt, i, t, StTov: integer; T: text;
begin
  { Preberimo vhodno datoteko. }
  Assign(T, 'tovornjaki.in'); Reset(T); ReadLn(T, StTov);
  NajTrajekt := 0;
  for i := 1 to StTov do begin
    ReadLn(T, Dolzina[i]);
    NajTrajekt := NajTrajekt + Dolzina[i]; NajPas[i] := 1;
  end; {for i}
  Close(T);
  { Preizkusimo veliko naključnih razporedov. }
  for t := 1 to 10000 do begin
    for p := 1 to StPasov do DolPasu[p] := 0;
    for i := 1 to StTov do begin
      Pas[i] := Random(StPasov) + 1;
      DolPasu[Pas[i]] := DolPasu[Pas[i]] + Dolzina[i];
    end; {for i}
    Trajekt := 0;
    for p := 1 to StPasov do if DolPasu[p] > Trajekt then Trajekt := DolPasu[p];
    if Trajekt < NajTrajekt then
      begin NajTrajekt := Trajekt; for i := 1 to StTov do NajPas[i] := Pas[i] end;
  end; {for t}
  { Izpišimo najboljši razpored. }
  Assign(T, 'tovornjaki.out'); Rewrite(T); WriteLn(T, NajTrajekt);
  for i := 1 to StTov do WriteLn(T, NajPas[i]);
  Close(T);
end. {Tovornjaki}
```

Ali pa v C-ju:

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

#define MaxStTov 100
#define StPasov 3

int main()
{
  int Dolzina[MaxStTov], Pas[MaxStTov], NajPas[MaxStTov];
  int DolPasu[StPasov], Trajekt, NajTrajekt, i, t, StTov;

  /* Preberimo dolžine tovornjakov. Ena rešitev je že, da postavimo vse na prvi pas. */
  FILE *f = fopen("tovornjaki.in", "rt"); fscanf(f, "%d", &StTov);
  for (i = 0, NajTrajekt = 0; i < StTov; i++) {
    fscanf(f, "%d", &Dolzina[i]);
    NajTrajekt += Dolzina[i]; NajPas[i] = 0; }
  fclose(f);

  for (t = 0; t < 10000; t++) /* Preizkusimo veliko naključnih razporedov. */
  {
```

```
    /* Vsak tovornjak postavimo na naključno izbran pas. */
    for (i = 0; i < StPasov; i++) DolPasu[i] = 0;
    for (i = 0; i < StTov; i++)
      Pas[i] = rand() % StPasov, DolPasu[Pas[i]] += Dolzina[i];
    for (i = 0, Trajekt = 0; i < StPasov; i++)
      if (DolPasu[i] > Trajekt) Trajekt = DolPasu[i];
    if (Trajekt < NajTrajekt) { /* To je najkrajši trajekt doslej, zapomnimo si ga. */
      NajTrajekt = Trajekt;
      for (i = 0; i < StTov; i++) NajPas[i] = Pas[i]; }
  }
  /* Izpišimo najboljši najdeni razpored. */
  f = fopen("tovornjaki.out", "wt"); fprintf(f, "%d\n", NajTrajekt);
  for (i = 0; i < StTov; i++) fprintf(f, "%d\n", NajPas[i] + 1);
  fclose(f); return 0;
}
```

Pri naših testnih primerih je tovornjakov malo in njihove dolžine so majhna naravna števila, tako da gornji postopek praktično vedno najde trajekt, ki je le malo daljši od najkrajšega možnega; pogosto pa najde sploh najkrajši možni trajekt.

Še bolje se obnese različica gornje rešitve, ki razporeja tovornjake „požrešno“ namesto naključno: vsak tovornjak pošljemo na tisti pas, na katerem je skupna dolžina dosedanjih tovornjakov najmanjša. Dolžina dobljenega trajekta je pri tem postopku v splošnem odvisna od tega, v kakšnem vrstnem redu pregledujemo tovornjake, zato je pametno preizkusiti več naključnih vrstnih redov in izpisati najboljšo izmed vseh dobljenih rešitev:

```
program Tovornjaki;
const MaxStTov = 100; StPasov = 3;
var Dolzina, Pas, NajPas, VrstniRed: array [1..MaxStTov] of integer;
    DolPasu: array [1..StPasov] of integer;
    Trajekt, NajTrajekt: integer; i, j, k, p, StTov: integer; T: text;
begin
  { Preberimo vhodno datoteko. }
  Assign(T, 'tovornjaki.in'); Reset(T); ReadLn(T, StTov);
  NajTrajekt := 0;
  for i := 1 to StTov do begin
    ReadLn(T, Dolzina[i]);
    NajTrajekt := NajTrajekt + Dolzina[i]; NajPas[i] := 1;
  end; {for i}
  Close(T);
  { Preizkusimo veliko naključnih vrstnih redov. }
  for k := 1 to 1000 do begin
    { Sestavimo nek naključen vrstni red tovornjakov. }
    for i := 1 to StTov do begin
      j := Random(i) + 1;
      if j < i then VrstniRed[i] := VrstniRed[j];
      VrstniRed[j] := i;
    end; {for i}
    { Preglejmo tovornjake v tem vrstnem redu, vsakega dajmo na najkrajši pas. }
    for i := 1 to StPasov do DolPasu[i] := 0;
    for i := 1 to StTov do begin
      j := VrstniRed[i]; Pas[j] := 1;
      for p := 2 to StPasov do if DolPasu[p] < DolPasu[Pas[j]] then Pas[j] := p;
      DolPasu[Pas[j]] := DolPasu[Pas[j]] + Dolzina[j];
```

```
      end; {for i}
      { Če je to najboljši razpored doslej, si ga zapomnimo. }
      Trajekt := 0;
      for i := 1 to 3 do if DolPasu[i] > Trajekt then Trajekt := DolPasu[i];
      if Trajekt < NajTrajekt then
        begin NajTrajekt := Trajekt; for i := 1 to StTov do NajPas[i] := Pas[i] end
    end; {for k}
    { Izpišimo najboljši razpored. }
    Assign(T, 'tovornjaki.out'); Rewrite(T); WriteLn(T, NajTrajekt);
    for i := 1 to StTov do WriteLn(T, NajPas[i]);
    Close(T);
end. {Tovornjaki}
```

In v C-ju:

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

#define MaxStTov 100
#define StPasov 3

int main()
{
  int Dolzina[MaxStTov], Pas[MaxStTov], NajPas[MaxStTov], VrstniRed[MaxStTov];
  int DolPasu[StPasov], Trajekt, NajTrajekt, i, j, p, t, StTov;

  /* Preberimo dolžine tovornjakov. Ena rešitev je že, da postavimo vse na prvi pas. */
  FILE *f = fopen("tovornjaki.in", "rt"); fscanf(f, "%d", &StTov);
  for (i = 0, NajTrajekt = 0; i < StTov; i++) {
    fscanf(f, "%d", &Dolzina[i]);
    NajTrajekt += Dolzina[i]; NajPas[i] = 0; }
  fclose(f);

  for (t = 0; t < 1000; t++) /* Preizkusimo veliko naključnih vrstnih redov. */
  {
    /* Pripravimo naključen vrstni red tovornjakov. */
    for (i = 0; i < StTov; i++) {
      j = rand() % (i + 1);
      VrstniRed[i] = VrstniRed[j]; VrstniRed[j] = i; }
    for (p = 0; p < StPasov; p++) DolPasu[p] = 0;

    /* Tovornjake pregledujmo v izbranem vrstnem redu. */
    for (i = 0; i < StTov; i++) {
      j = VrstniRed[i];
      /* Postavimo tovornjak j na tisti pas, ki je zaenkrat najkrajši. */
      for (Pas[j] = 0, p = 1; p < StPasov; p++)
        if (DolPasu[p] < DolPasu[Pas[j]]) Pas[j] = p;
      DolPasu[Pas[j]] += Dolzina[j]; }

    /* Kako dolg trajekt potrebujemo za tak razpored? */
    for (p = 0, Trajekt = 0; p < StPasov; p++)
      if (DolPasu[p] > Trajekt) Trajekt = DolPasu[p];
    if (Trajekt < NajTrajekt) { /* To je najkrajši trajekt doslej, zapomnimo si ga. */
      NajTrajekt = Trajekt;
      for (i = 0; i < StTov; i++) NajPas[i] = Pas[i]; }
  }

  /* Izpišimo najboljši najdeni razpored. */
  f = fopen("tovornjaki.out", "wt"); fprintf(f, "%d\n", NajTrajekt);
  for (i = 0; i < StTov; i++) fprintf(f, "%d\n", NajPas[i] + 1);
```

```
  fclose(f); return 0;
}
```

Ta postopek je pri naših poskusih in na naših testnih primerih vedno odkril najkrajši možni trajekt. V splošnem pa nam niti naključni niti požrešni postopek ne dajeta zagotovila, da bodo njuni trajekti res najkrajši možni.

Oglejmo si še postopek, ki je časovno in prostorsko znatno zahtevnejši od gornjih dveh, vendar pa vedno zagotovo najde najkrajši možni trajekt.

Stanje trajekta lahko predstavimo z urejeno trojico števil (p_1, p_2, p_3), ki povedo skupno dolžino vseh tovornjakov na vsakem od treh pasov. V naši vhodni datoteki je neko zaporedje n tovornjakov z dolžinami $d_1, \ldots, d_n$. Če je mogoče prvih i tovornjakov razporediti v tri pasove tako, da je vsota dolžin tovornjakov na prvem pasu p_1, na drugem p_2 in na tretjem p_3, bomo rekli, da je stanje (p_1, p_2, p_3) *dosegljivo* s prvimi i tovornjaki. (Takoj lahko opazimo, da nobeno stanje ne more biti dosegljivo s prvimi i tovornjaki za več različnih vrednosti i — če je dosegljivo s prvimi i, to pomeni, da je $p_1 + p_2 + p_3$ enako vsoti dolžin prvih i tovornjakov; če bi bilo hkrati dosegljivo tudi s prvimi i' tovornjaki za nek $i' > i$, bi to pomenilo, da je $p_1 + p_2 + p_3$ enako tudi vsoti dolžin prvih i' tovornjakov, torej je vsota dolžin tovornjakov od $i + 1$ do i' enaka 0, kar pa je nemogoče.)

Stanje (p_1, p_2, p_3) je možno le na trajektu, ki je dolg vsaj $\max\{p_1, p_2, p_3\}$ enot. Če bi pregledali vsa stanja, dosegljiva z n tovornjaki, bi torej lahko za vsako ugotovili, kako dolg trajekt zahteva, in izpisali tisto z najkrajšim trajektom.

S preprostim rekurzivnim podprogramom (Razvij v spodnjem programu) si lahko v neki tabeli (StTov) pripravimo podatke o tem, s koliko tovornjaki je dosegljivo posamezno stanje (če je sploh dosegljivo). Rekurzija temelji na opažanju, da če je stanje (p_1, p_2, p_3) dosegljivo s prvimi $i-1$ tovornjaki, potem so stanja (p_1+d_i, p_2, p_3), (p_1, p_2+d_i, p_3) in (p_1, p_2, p_3+d_i) dosegljiva s prvimi i tovornjaki. Ob vsakem stanju si tudi zapomnimo (tabela KjeZadnji), na kateri pas smo pri tem stanju uvrstili zadnji (i-ti) tovornjak. Začnemo pa lahko ta postopek pri $i = 0$ tovornjakih, ko vemo, da je dosegljivo le stanje $(0, 0, 0)$.

S tabelo KjeZadnji si pomagamo, da na koncu rekonstruiramo celoten razpored tovornjakov po pasovih. Če je (p_1, p_2, p_3) stanje, ki nam da najkrajši trajekt za vseh n tovornjakov, lahko s pogledom v KjeZadnji ugotovimo, na katerem pasu stoji zadnji (n-ti) tovornjak v razporedu, ki pripelje do tega stanja; dolžino tega tovornjaka potem odštejemo od enega od števil p_1, p_2, p_3 (odvisno od pasu, na katerem je bil tovornjak) in tako dobimo stanje pred prihodom zadnjega tovornjaka; zanj spet pogledamo v KjeZadnji in tako ugotovimo, kje mora stati $(n-1)$-vi tovornjak in tako naprej.

```
program Tovornjaki;
const MaxDolzTraj = 100; MaxDolzTov = MaxDolzTraj; MaxStTov = 100;
type TabelaP = ↑TabelaT;
     TabelaT = packed array [0..MaxDolzTraj, 0..MaxDolzTraj, 0..MaxDolzTraj] of integer;
var n: integer; StTov, KjeZadnji: TabelaP; Dolzine: array [1..MaxStTov] of integer;

  { Klic tega podprograma pomeni, da je stanje (d1, d2, d3) dosegljivo s prvimi k tovornjaki,
    pri čemer je zadnji od njih na pasu p. }
  procedure Razvij(d1, d2, d3, k, p: integer);
  var d: integer;
  begin
```

```
    if (d1 > MaxDolzTraj) or (d2 > MaxDolzTraj) or (d3 > MaxDolzTraj) then exit;
    if StTov↑[d1, d2, d3] >= k then exit; { Do tega stanja smo že prišli nekoč prej. }
    StTov↑[d1, d2, d3] := k; KjeZadnji↑[d1, d2, d3] := p;
    if k < n then begin
      d := Dolzine[k + 1]; Razvij(d1 + d, d2, d3, k + 1, 1);
      Razvij(d1, d2 + d, d3, k + 1, 2); Razvij(d1, d2, d3 + d, k + 1, 3);
    end; {if}
  end; {Razvij}

  procedure NajdiResitev(var d1, d2, d3: integer);
  var i1, i2, i3: integer;
  begin
    for i1 := 0 to MaxDolzTraj do for i2 := 0 to i1 do for i3 := 0 to i2 do
      if StTov↑[i1, i2, i3] = n then begin d1 := i1; d2 := i2; d3 := i3; exit end;
  end; {NajdiResitev}

var d1, d2, d3, i, r, Dolzina: integer; Pasovi: array [1..MaxStTov] of integer; T: text;
begin
  Assign(T, 'tovornjaki.in'); Reset(T); ReadLn(T, n);
  for i := 1 to n do ReadLn(T, Dolzine[i]);
  Close(T);
  New(StTov); New(KjeZadnji);
  for d1 := 0 to MaxDolzTraj do for d2 := 0 to MaxDolzTraj do
    for d3 := 0 to MaxDolzTraj do StTov↑[d1, d2, d3] := -1;
  Razvij(0, 0, 0, 0, 0);
  NajdiResitev(d1, d2, d3); r := d1;
  for i := n downto 1 do begin
    Pasovi[i] := KjeZadnji↑[d1, d2, d3];
    if Pasovi[i] = 1 then d1 := d1 - Dolzine[i]
    else if Pasovi[i] = 2 then d2 := d2 - Dolzine[i]
    else d3 := d3 - Dolzine[i];
  end; {for i}
  Dispose(StTov); Dispose(KjeZadnji);
  Assign(T, 'tovornjaki.out'); Rewrite(T); WriteLn(T, r);
  for i := 1 to n do WriteLn(T, Pasovi[i]);
  Close(T);
end. {Tovornjaki}
```

Zapišimo to rešitev še v C-ju:

```
#include <stdio.h>
#define MaxDolzTraj 100
#define MaxStTov 100
typedef int TabelaT[MaxDolzTraj + 1][MaxDolzTraj + 1][MaxDolzTraj + 1];
/* StTov[d1][d2][d3] = število tovornjakov, ki jih je mogoče razporediti
   v tri pasove tako, da je dolžina prvega pasu d1, drugega d2 in tretjega d3.
   KjeZadnji = pas, na katerem mora biti zadnji tovornjak pri takem razporedu. */
TabelaT StTov, KjeZadnji;
int n, Dolzina[MaxStTov]; /* število tovornjakov in njihove dolžine */

/* Spodnji podprogram zapiše podatek, da je stanje (d1, d2, d3) dosegljivo s k tovornjaki,
   pri čemer je zadnji od njih na pasu p. Nato se pokliče rekurzivno za stanja,
   ki jih lahko dosežemo tako, da trenutnemu dodajamo še ostale tovornjake. */
void Razvij(int d1, int d2, int d3, int k, int p)
```

```
{
  int d;
  if (d1 > MaxDolzTraj || d2 > MaxDolzTraj || d3 > MaxDolzTraj) return;
  if (StTov[d1][d2][d3] >= k) return;
  StTov[d1][d2][d3] = k; KjeZadnji[d1][d2][d3] = p;
  if (k < n) {
    d = Dolzina[k]; Razvij(d1 + d, d2, d3, k + 1, 1);
    Razvij(d1, d2 + d, d3, k + 1, 2); Razvij(d1, d2, d3 + d, k + 1, 3); }
}

/* Najde najkrajši trajekt, dosegljiv z n tovornjaki. */
void NajdiResitev(int *d1, int *d2, int *d3)
{
  int i1, i2, i3;
  for (i1 = 0; i1 <= MaxDolzTraj; i1++)
    for (i2 = 0; i2 <= i1; i2++) for (i3 = 0; i3 <= i2; i3++)
      if (StTov[i1][i2][i3] == n) { *d1 = i1; *d2 = i2; *d3 = i3; return; }
}

int main()
{
  int i, d1, d2, d3, r, Nasel, Pasovi[MaxStTov];
  /* Preberimo dolžine tovornjakov. */
  FILE *f = fopen("tovornjaki.in", "rt"); fscanf(f, "%d", &n);
  for (i = 0; i < n; i++) fscanf(f, "%d", &Dolzina[i]);
  fclose(f);
  /* Raziščimo vsa dosegljiva stanja in poiščimo najkrajši trajekt. */
  for (d1 = 0; d1 <= MaxDolzTraj; d1++) for (d2 = 0; d2 <= MaxDolzTraj; d2++)
    for (d3 = 0; d3 <= MaxDolzTraj; d3++) StTov[d1][d2][d3] = -1;
  Razvij(0, 0, 0, 0, 0);
  NajdiResitev(&d1, &d2, &d3); r = d1;
  /* Rekonstruirajmo razpored na najkrajšem trajektu. */
  for (i = n - 1; i >= 0; i--) {
    Pasovi[i] = KjeZadnji[d1][d2][d3];
    if (Pasovi[i] == 1) d1 -= Dolzina[i];
    else if (Pasovi[i] == 2) d2 -= Dolzina[i];
    else d3 -= Dolzina[i]; }
  /* Izpišimo rešitev. */
  f = fopen("tovornjaki.out", "wt"); fprintf(f, "%d\n", r);
  for (i = 0; i < n; i++) fprintf(f, "%d\n", Pasovi[i]);
  fclose(f); return 0;
}
```

Naloge so sestavili: predpone — Gorazd Božič; sudoku — Primož Gabrijelčič; cestne lučke — Boris Gašperin; mafija — Uroš Jovanovič; sneg — Aleš Košir; odstavki, 1337ovščina, optična miška — Mark Martinec; drevo končnic — Mojca Miklavec; tovornjaki — Miha Vuk; naraščajoče besede, podnapisi, razvajeni zvezdniki, spletne knjigarne, izravnavanje histogramov — Janez Brank. Hvala Blažu Novaku za implementacijo rešitev nalog za 3. skupino. Primeri zamenjav v tabeli pri nalogi „1337ovščina“ so iz Wikipedijinega članka “Leet”.

REZULTATI

Spodnje tabele prikazujejo vrstni red vseh tekmovalcev, ki so sodelovali na letošnjem tekmovanju. Poleg skupnega števila doseženih točk je za vsakega tekmovalca navedeno tudi število točk, ki jih je dosegel pri posamezni nalogi. V prvi in drugi skupini je mogoče pri vsaki nalogi doseči največ 20 točk, v tretji skupini pa največ 100 točk.

Načeloma se v prvi skupini podeli tri nagrade za prvo mesto, tri za drugo in tri za tretje; v drugi skupini dve nagradi za prvo, dve za drugo in dve za tretje mesto; v tretji skupini pa po eno nagrado za prvo, drugo in tretje mesto. Letos smo v tretji skupini izjemoma podelili dve nagradi za prvo mesto, ker je bila razlika med najboljšima tekmovalcema tako majhna.

PRVA SKUPINA

Mesto	Mesto po točkah	Ime	Letnik	Šola	Točke (po nalogah in skupaj) 1	2	3	4	5	$\sum$
1	1	Borut Ajdič	4	ŠC Novo mesto	20	20	10	19	14	83
1	2	Matjaž Kraševec	4	ŠC Novo mesto	8	20	19	20	15	82
1	2	Nace Hudobivnik	1	Šk. klas. gim. Ljubljana	15	20	17	17	13	82
2	4	Nejc Ramovš	3	ŠC Novo mesto	18	15	19	8	18	78
2	4	Matic Redek	3	ŠC Novo mesto	20	15	20	5	18	78
2	4	Matjaž Payrits	1	Gimnazija Bežigrad	8	15	20	20	15	78
3	7	Mitja Lapajne	3	Gim. J. Vege Idrija	15	13	17	13	15	73
3	8	Miha Sedej	4	Gim. J. Vege Idrija	20	20	2	12	17	71
3	9	Gašper Rugelj	1	Gimnazija Bežigrad	17	20	0	20	10	67
	10	Amela Rakanović	3	SŠ S. Kosovela, Sežana	10	20	20	8	3	61
	11	Robert Slak	3	ŠC Novo mesto	12	15	3	17	8	55
	12	Andrej Krebs	1	Gimnazija Bežigrad	19	5	0	14	15	53
	13	Anja Višnikar	3	Gimnazija Poljane	0	15	8	15	0	38
	14	Aleš Bolka	4	Gimnazija Kranj	3	10	20	0	0	33
	14	Sandi Verdev	2	ŠC Celje, Gim. Lava	1	0	10	17	5	33
	16	Daniel Brulc	3	ŠC Novo mesto	0	15	9	7	0	31
	16	Rožle Bregar	3	TŠC Kranj	3	20	5	3	0	31
	18	Klemen Kobetič	3	ŠC Novo mesto	0	15	5	3	0	23
	19	Rok Keglevič	1	Gimnazija Bežigrad	0	18	0	4	0	22
	20	Nace Kranjc	1	Gim. J. Vege Idrija	0	19	2	0	0	21
	21	Marko Blagojevič	3	TŠC Kranj	0	5	0	5	1	11
	22	Jurij Zibler	3	Gimnazija Kranj	0	2	0	0	0	2
	23	Gašper Peklaj	1	Gimnazija Poljane	0	0	0	0	0	0
	23	Boštjar Luzar	3	ŠC Novo mesto	0	0	0	0	0	0
	23	Jernej Drobež	1	Gimnazija Bežigrad	0	0	0	0	0	0

DRUGA SKUPINA

Mesto	Mesto po točkah	Ime	Letnik	Šola	Točke (po nalogah in skupaj) 1	2	3	4	5	$\sum$
1	1	Andraž Žagar	4	ŠC Novo mesto	18	15	19	12	13	77
1	2	Jan Berčič	2	ZRI + Gim. Bežigrad	18	13	19	10	11	71
2	3	Aljaž Osojnik	3	ZRI + Gim. Bežigrad	3	20	15	18	9	65
2	4	Mitja Ferder	3	ŠC Velenje, PTERŠ	15	3	15	11	19	63
3	5	Matej Jakop	3	ŠC Celje, Gim. Lava	19	10	19	2	8	58
3	6	Jaka Ivančič	4	ŠC Postojna	14	11	18	7	0	50
	7	Miha Lunar	2	SŠER Ljubljana	20	8	5	7	7	47
	8	Matej Prepelič	4	SERŠ Maribor	20	12	0	13	0	45
	9	Damir Srpčič	3	ŠC Novo mesto	16	10	5	13	0	44
	10	Primož Korošec	2	SŠER Ljubljana	17	12	0	2	0	31
	11	Jakob Drešar	3	Gimnazija Kranj	17	5	0	4	4	30
	12	David Božjak	2	SŠER Ljubljana	10	4	5	0	8	27
	13	Matej Lenarčič	4	ŠC Postojna	4	0	1	7	10	22
	14	Tilen Marko	4	SERŠ Maribor	8	0	5	7	0	20
	15	Miha Dodič	?	SŠER Ljubljana	0	10	1	3	3	17
	16	Aljaž Čeru	2	SŠER Ljubljana	8	8	0	0	0	16
	17	Rok Meglič	4	TŠC Kranj	3	4	1	3	4	15
	18	Mario Jurišič	3	ŠC Celje, Gim. Lava	4	3	0	1	4	12
	19	Denis Aleksander Križman	2	SŠER Ljubljana	0	0	5	6	0	11
	19	Aleš Kalan	4	TŠC Kranj	4	0	1	5	1	11
	19	Boštjan Štor	3	ŠC Celje, Gim. Lava	4	3	1	0	3	11
	22	Dejan Učakar	2	SŠER Ljubljana	2	0	1	7	0	10
	23	Aleksander Kozlar	2	SPTŠ Murska Sobota	1	1	0	1	5	8
	24	Patrick Zver	2	SPTŠ Murska Sobota	2	0	5	0	0	7
	25	David Karlaš	2	SŠER Ljubljana	4	0	2	0	0	6
	26	Matej Grujić	2	SŠER Ljubljana	0	5	0	0	0	5
	27	Gregor Slana	2	SPTŠ Murska Sobota	1	1	0	1	1	4
	28	Gašper Mlinar	2	SŠER Ljubljana	3	0	0	0	0	3
	29	Roman Šuster	2	SPTŠ Murska Sobota	1	0	0	1	0	2

TRETJA SKUPINA

Mesto	Mesto po točkah	Ime	Letnik	Šola	Točke (po nalogah in skupaj) 1	2	3	4	5	$\sum$
1	1	Primož Koželj	4	ZRI + Gim. Bežigrad	100	100	94	100	100	494
1	2	Nino Bašić	4	ZRI + Gim. Bežigrad	100	97	100	100	94	491
2	3	Matjaž Gomilšek	4	II. gimn. Maribor	100	54	0	97	83	334
3	4	Tomaž Hočevar	3	ZRI + Gim. Vič	100	97		100	18	315
	5	Matic Vrščaj	3	SŠER Ljubljana	100		30	100	71	301
	6	Jan Berdajs	3	Gimnazija Bežigrad	96	0		100	79	275
	7	Blaž Tomažič	4	SŠER Ljubljana	87			100	79	266
	8	Boštjan Bogataj	4	TŠC Kranj	94		0	100		194
	9	David Vodnik	2	Gimnazija Kranj	0			100	86	186
	10	Žiga Osolin	3	Gimnazija Bežigrad	35			90	24	149
	11	Matjaž Debelak	3	TŠC Kranj	4			100	16	120
	12	Rok Močnik	4	Gim. J. Vege Idrija	24		0	60	20	104
	13	Gašper Ažman	3	Gimnazija Vič		10		88	0	98
	14	Franci Kalan	4	TŠC Kranj					77	77
	15	Urh Srečnik	4	TŠC Kranj				60		60
	16	Matic Potočnik	4	TŠC Kranj					0	0
	16	Oto Brglez	4	ŠC Velenje, PTERŠ		0			0	0
	16	Tomaž Treven	1	Šk. klas. gim. Ljubljana			0			0
	16	Miha Miklič	3	Kmetijska šola Grm				0		0
	16	Simon Kralj	4	SŠER Ljubljana				0		0
	16	Blaž Kraker	4	TŠC Kranj						0

NAGRADE

Za nagrado so najboljši tekmovalci vsake skupine prejeli naslednjo strojno opremo in knjižne nagrade:

Skupina	Mesto	Nagrajenec	Nagrade
1	1	Borut Ajdič	320 GB zunanji disk
			S. Hawking: *Črne luknje in otroška vesolja*
1	1	Matjaž Kraševec	Olympus FE-120
			J. Weeks: *Oblika prostora*
1	1	Nace Hudobivnik	Olympus FE-120
			J. Weeks: *Oblika prostora*
1	2	Nejc Ramovš	160 GB zunanji disk
			J. Weeks: *Oblika prostora*
1	2	Matic Redek	160 GB zunanji disk
			J. Weeks: *Oblika prostora*
1	2	Matjaž Payrits	miška Logitech G5
			K. Devlin: *Nova zlata doba matematike*
1	3	Mitja Lapajne	miška Logitech G5
			K. Devlin: *Nova zlata doba matematike*
1	3	Miha Sedej	predvajalnik DVDjev
			K. Devlin: *Nova zlata doba matematike*
1	3	Gašper Rugelj	predvajalnik DVDjev
			K. Devlin: *Nova zlata doba matematike*
2	1	Andraž Žagar	320 GB zunanji disk
			Wilson, Watkins: *Uvod v teorijo grafov*
			N. Morris *et al.*: *Ilustrirana zgodovina sveta*
2	1	Jan Berčič	320 GB zunanji disk
			Wilson, Watkins: *Uvod v teorijo grafov*
			N. Morris *et al.*: *Ilustrirana zgodovina sveta*
2	2	Aljaž Osojnik	Olympus FE-120
			S. Hawking: *Kratka zgodovina časa*
2	2	Mitja Ferder	Olympus FE-120
			S. Hawking: *Kratka zgodovina časa*
2	3	Matej Jakop	160 GB zunanji disk
			S. Hawking: *Črne luknje in otroška vesolja*
			Z. Xinxin, S. Ye: *Ko je Kitajska še upala*
2	3	Jaka Ivančič	160 GB zunanji disk
			S. Hawking: *Črne luknje in otroška vesolja*
			Z. Xinxin, S. Ye: *Ko je Kitajska še upala*
3	1	Primož Koželj	2 GB iPod nano
			R. Banks: *Ledene gore, padajoče domine*
3	1	Nino Bašić	2 GB iPod nano
			R. Banks: *Ledene gore, padajoče domine*
3	2	Matjaž Gomilšek	320 GB zunanji disk
			Wilson, Watkins: *Uvod v teorijo grafov*
3	3	Tomaž Hočevar	200 GB zunanji disk
			S. Hawking: *Kratka zgodovina časa*
			B. Hvala: *Matematika in šport*
Mačka in miš	1	Tomaž Hočevar	160 GB zunanji disk
			B. Hvala: *Matematika in šport*
			T. Mihelič: *Andi: beli vrhovi zelene celine*

Vsak od nagrajencev je prejel tudi izvod knjige *Enajsta šola računalništva: rešene naloge z republiških tekmovanj 1977–1987* (uredili Vladimir Batagelj *et al.*, DMFA, 1988).

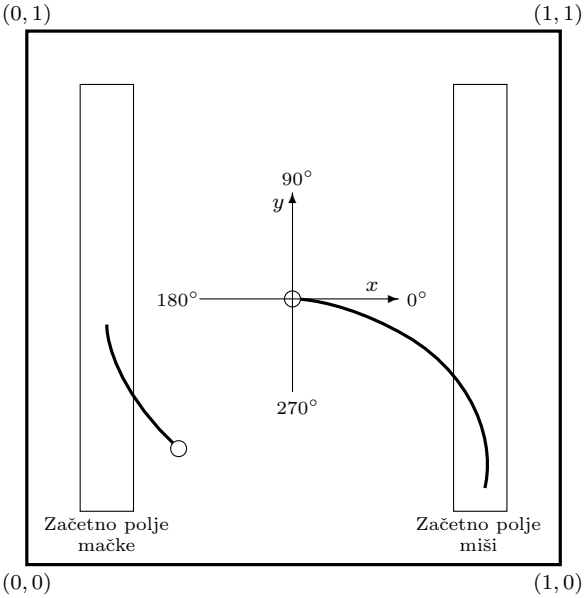
SODELUJOČE ŠOLE IN MENTORJI

Gimnazija Bežigrad, Ljubljana	Andrej Šuštaršič, Jurij Železnik
Srednja šola za elektrotehniko in računalništvo (SŠER), Ljubljana	Nataša Makarovič, Darjan Toth
Gimnazija Kranj	Primož Zevnik, Matevž Jekovec
Šolski center Velenje, Poklicna in tehniška elektro in računalniška šola (PTERŠ)	Damjan Borovnik, Uroš Sonjak
Šolski center Novo mesto	Mile Božič, Albert Zorko, Andrej Jakobčič
Zavod za računalniško izobraževanje (ZRI), Ljubljana	Jelko Urbančič
Tehniški šolski center (TŠC) Kranj	Gabrijela Kranjc
Gimnazija Jurija Vege, Idrija	Danica Bogataj
Gimnazija Poljane, Ljubljana	Boštjan Žnidaršič
Srednja poklicna in tehniška šola (SPTŠ) Murska Sobota	Simon Horvat
Šolski center Celje, Splošna in strokovna gimnazija Lava	Borut Slemenšek
Škofijska klasična gimnazija, Ljubljana	
Srednja šola Srečka Kosovela, Sežana	
Srednja elektro-računalniška šola Maribor (SERŠ)	
Šolski center Postojna, srednja šola, gimnazija	Stanislav Rupar
II. gimnazija Maribor	
Kmetijska šola Grm, Novo mesto	Ivan Kočevar
(zunajšolska mentorja)	Kazimir Gomilšek, Demir Rakanović

TEKMOVANJE PROGRAMOV — MAČK IN MIŠI

V okviru računalniškega tekmovanja je bilo izvedeno tudi tekmovanje programov. Za razliko od tekmovanja v znanju računalništva, kjer imajo tekmovalci na voljo le nekaj ur časa za reševanje, je v tej kategoriji problem objavljen nekaj mesecev pred tekmovanjem, naloga tekmovalcev pa je, da do tekmovanja napišejo delujoč program, ki ta problem čim bolje reši.

Letošnja naloga je od tekmovalcev zahtevala, da napišejo nadzorno logiko za igralca v preprosti računalniški igri: lovu mačke in miši. Mačka in miš sta predstavljeni s točkama, lovita pa se po kvadratnem igralnem polju velikosti 1×1. Začetni položaj obeh živali je izbran naključno, vendar tako, da je mačka znotraj pravokotnika $[\frac{1}{10}, \frac{2}{10}] \times [\frac{1}{10}, \frac{9}{10}]$, miš pa znotraj $[\frac{8}{10}, \frac{9}{10}] \times [\frac{1}{10}, \frac{9}{10}]$. Začetna hitrost obeh živali je 0, začetno smer pa si lahko vsaka žival izbere sama.



Igra poteka v korakih. V vsakem koraku se obe živali odločita o svojem premiku, ki ga izrazita z želeno spremembo trenutne smeri in hitrosti. Obe živali sta enakovredni — imata enake omejitve pri spreminjanju smeri in hitrosti. Žival lahko v vsakem koraku spremeni svojo smer za največ 5°, hitrost pa za največ 0,001 (hitrost se meri v dolžinskih enotah na časovni korak; spomnimo se, da je celotno igralno polje kvadrat, čigar stranice so dolge ravno po eno dolžinsko enoto).

Mačkin cilj je uloviti miš, naloga miši pa je uspešno bežati pred mačko. Mačka ujame miš, če se ji približa na zadosti majhno razdaljo (0,01 ali manj), miš pa mora uspešno bežati 3000 korakov, preden ji priznamo zmago. Vsaka žival mora paziti tudi na to, da ostane znotraj meja igralne površine; v nasprotnem primeru takoj izgubi. Za računanje želenih premikov je bilo na voljo precej časa in pomnilnika, vendar so imele vse prejete rešitve zanemarljivo majhno porabo sistemskih virov.

Vsak tekmovalec je moral napisati po en podprogram za krmiljenje miši in en podprogram za krmiljenje mačke. Po roku za oddajo rešitev smo izvedli navzkrižno tekmovanje: vsaka mačka je poskusila svojo iznajdljivost z vsako nasprotnikovo mišjo, za večje zaupanje v rezultate pa smo posamezno tekmo ponovili dvajsetkrat.

Rezultati

Kljub precejšnjemu zanimanju ob objavi naloge smo do tekmovanja prejeli le tri pare živali, od tega je eno poslal študent, dve pa dijaka. Najpogosteje se je igra končala z mačkino prekoračitvijo mej igralnega polja, nekatere kombinacije živali pa so povzročile precej nepričakovano obnašanje.

V obeh kategorijah (pri mačkah in pri miših) je vrstni red po številu dobljenih iger enak: na prvem mestu sta rešitvi Tomaža Hočevarja (ZRI), drugi je Jurij Kodre (FMF) in tretji Nino Bašić (ZRI).

ANKETA

Tekmovalcem vseh treh skupin smo na tekmovanju skupaj z nalogami razdelili tudi naslednjo anketo. Rezultati ankete so predstavljeni na str. 73–79.

Spol: ☐ M ☐ Ž
Letnik: ☐ 1 ☐ 2 ☐ 3 ☐ 4 ☐ 5

Kako si izvedel za tekmovanje?
☐ od mentorja ☐ na spletni strani (kateri? _____)
☐ od prijatelja/sošolca ☐ drugače (kako? _____)
Koliko let že sodeluješ na tekmovanjih iz računalništva? _____
Koliko let že sodeluješ na srednješolskih tekmovanjih iz računalništva? _____
Koliko časa že programiraš? _____
Kje si se naučil? ☐ sam ☐ v šoli pri pouku ☐ na krožkih ☐ na tečajih ☐ poletna šola
Katere programske jezike znaš in kako dobro? _____
_____

[Le pri 1. in 2. skupini.] V besedilu nalog trenutno objavljamo deklaracije tipov in podprogramov v pascalu in C/C++.
— Ali razumeš kakšnega od teh jezikov dovolj dobro, da razumeš te deklaracije v besedilu naših nalog? ☐ da ☐ ne
— Ali bi raje videl, da bi objavljali deklaracije (tudi) v kakšnem drugem programskem jeziku? Če da, v katerem? _____

V rešitvah nalog trenutno objavljamo izvorno kodo v pascalu.
— Ali razumeš pascal dovolj dobro, da si lahko kaj pomagaš s pascalsko izvorno kodo v naših rešitvah? ☐ da ☐ ne
— Ali bi raje videl, da bi izvorno kodo rešitev namesto v pascalu pisali v kakšnem drugem jeziku? Če da, v katerem? _____

Katere od naslednjih jezikovnih konstruktov in programerskih prijemov znaš uporabljati?
☐ branje in pisanje s standardnega vhoda in izhoda (ReadLn, WriteLn, ...)
☐ branje in pisanje iz datotek
☐ tabele (**array**)
☐ znaš napisati svoj podprogram (**procedure**, **function**)
☐ poznaš rekurzijo (podprogram, ki kliče samega sebe)
☐ kazalce, dinamično alokacijo pomnilnika (New/Dispose, GetMem/FreeMem, malloc/free, **new**/**delete**, ...)
☐ zanka **for**
☐ zanka **while**
☐ gnezdenje zank (ena zanka znotraj druge)
☐ naštevni tipi (*enumerated types* — **type** ImeTipa = (Ena, Dve, Tri) v pascalu, **typedef enum** v C/C++)
☐ strukture (**record** v pascalu, **struct**/**class** v C/C++)
☐ **and**, **or**, **xor**, **not** kot aritmetični operatorji (nad biti celoštevilskih operandov namesto nad logičnimi vrednostmi tipa boolean) (v C/C++: &, |, ^, ~)
☐ operatorja **shl** in **shr** (v C/C++: <<, >>)

[Le pri 3. skupini.] Doslej smo v tretji skupini podpirali reševanje nalog v pascalu, C, C++ in javi. Bi rad uporabljal kakšen drug programski jezik? Če da, katerega? _____

[Naslednja skupina vprašanj se je ponovila za vsako nalogo po enkrat.]

Zahtevnost naloge: ☐ prelahka ☐ lahka ☐ primerna ☐ težka ☐ pretežka ☐ ne vem
Naloga je (ali: bi) vzela preveč časa: ☐ da ☐ ne ☐ ne vem
Predolgo/prezahtevno/nejasno besedilo: ☐ da ☐ ne ☐ ne vem
Naloga je bila: ☐ zanimiva ☐ dolgočasna ☐ že znana ☐ povprečna
Si jo rešil?
- ☐ zmanjkalo mi je časa za reševanje
- ☐ zmanjkalo mi je volje za reševanje
- ☐ nisem prišel daleč
- ☐ rešil sem le delno
- ☐ rešil sem skoraj v celoti
- ☐ rešil sem celo

Ostali komentarji o tej nalogi: _____

Katera naloga ti je bila najbolj in katera najmanj všeč? Zakaj? _____

Na letošnjem tekmovanju ste imeli tri ure [1. in 2. skupina] / pet ur [3. skupina] časa za pet nalog.
Bi imel raje: ☐ več časa ☐ manj časa ☐ časa je bilo ravno prav
Bi imel raje: ☐ več nalog ☐ manj nalog ☐ nalog je bilo ravno prav

Kakršne koli druge pripombe in predlogi. Kaj bi spremenil(a), popravil(a), odpravil(a), ipd., da bi postalo tekmovanje zanimivejše in bolj privlačno? _____

Kaj ti je bilo pri tekmovanju všeč? _____

Kaj te je najbolj motilo? _____

Če imaš kaj vrstnikov, ki se tudi zanimajo za programiranje, pa se tega tekmovanja niso udeležili, kaj bi bilo po tvojem mnenju treba spremeniti, da bi jih prepričali k udeležbi?

Poleg tekmovanja bi radi tudi v preostalem delu leta organizirali razne aktivnosti, ki bi vas zanimale, spodbujale in usmerjale pri odkrivanju računalništva. Prosimo, da nam pomagate izbrati aktivnosti, ki vas zanimajo in bi se jih zelo verjetno udležili.

Udeležil bi se oz. z veseljem bi spremljal:
- ☐ izlet v kak raziskovalni laboratorij v Evropi (po možnosti za dva dni)
- ☐ poletna šola računalništva (1 teden na IJS, spanje v dijaškem domu)
- ☐ poletna praksa na IJS
- ☐ predstavitve novih tehnologij (.NET, mobilni portali, programiranje „vgrajenih računalnikov“, strojno učenje, itd.) (1× mesečno)
- ☐ predavanja o algoritmih in drugih temah, ki pridejo prav na tekmovanju (1× mesečno)
- ☐ reševanje tekmovalnih nalog (naloge se rešuje doma in bi bile delno povezane s temo, predstavljeno na predavanju; rešitve se preveri na strežniku) (1× mesečno)
- ☐ tvoji predlogi: _____

Vesel bi bil pomoči pri:
- ☐ iskanju štipendije
- ☐ iskanju podjetij, ki dijakom ponujajo njim prilagojene poletne prakse in druge projekte, kjer se ob mentorstvu lahko veliko naučijo.

Hvala za sodelovanje in lep pozdrav! Tekmovalna komisija

REZULTATI ANKETE

Anketo je izpolnilo 24 tekmovalcev prve skupine, 28 tekmovalcev druge in 17 tekmovalcev tretje. Ker se marsikomu ne da pisati odgovorov na vprašanja, smo pri letošnji anketi dali večji poudarek na vprašanjih z vnaprej ponujenimi odgovori, pri katerih je treba le odkljukati okence pred želenim odgovorom.

Mnenje tekmovalcev o nalogah

Tekmovalce smo spraševali: kako zahtevna se jim zdi posamezna naloga; ali se jim zdi, da jim vzame preveč časa; ali je besedilo predolgo, prezahtevno ali nejasno; ali se jim zdi naloga zanimiva; in ali so jo rešili (oz. zakaj ne).

Rezultate vprašanj o zahtevnosti nalog kažejo grafi na str. 74. Tam so tudi podatki o povprečnem številu točk, doseženem pri posamezni nalogi, tako da lahko primerjamo mnenje tekmovalcev o zahtevnosti naloge in to, kako dobro so jo zares reševali.

Edina naloga, ki se je res veliko ljudem zdela pretežka, so podnapisi (1.5) v prvi skupini. V drugi skupini so se jim zdele precej težke lučke (2.3); predvsem zato, ker niso navajeni operacij na bitih. Splošni vtis tekmovalcev o drugi skupini je, da bi bila mogoče lahko malenkost lažja. Tretja skupina letos ni bila kaj posebej zahtevna in to se pozna tudi na odgovorih v anketi. Da bi bila kakšna naloga prelahka, se tudi ne pritožujejo, razen deloma pri snegu (1.2), ki je res zelo lahka.

Pri nekaterih nalogah je razpon ocen zahtevnosti nenavadno širok: isto nalogo imajo nekateri za prelahko, nekateri za pretežko. Primera sta: lučke (2.3), miška (3.1).

Rezultate ostalih vprašanj o nalogah pa kažejo grafi na str. 75. Nad razumljivostjo besedil ni veliko pripomb, razen pri nalogi 2.4 (drevo končnic). Razmeroma veliko pripomb je tudi na 2.2 (predpone) in 2.3 (lučke).

V tretji skupini se jim nobena naloga ni zdela preveč zamudna. Še največ pripomb, da sta prezamudni, je bilo na podnapise (1.5) in končnice (2.4).

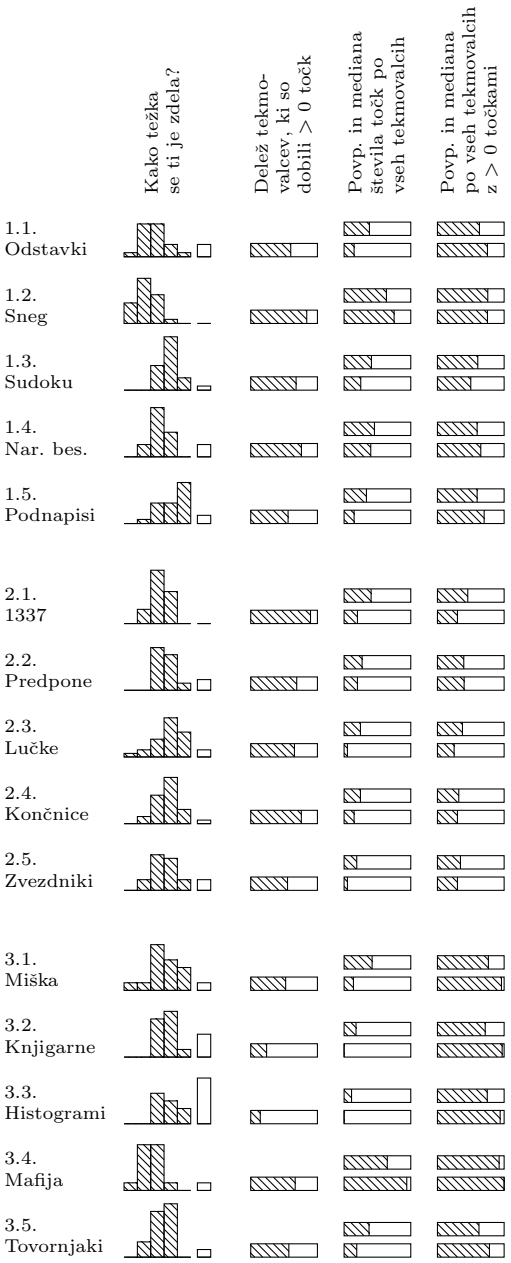
Večini tekmovalcev so se zdele praktično vse naloge zanimive. Manj navdušeni so le nad odstavki (1.1).

Pri nalogi 1.1 (odstavki) je nekaj tekmovalcev upravičeno pripomnilo, da je besedilo nejasno, ker pravi le, da naj program prebira besedilo s standardnega vhoda, ne pove pa, kako dolgo naj to počne. Mišljeno je bilo, naj program bere vse do konca besedila (EOF), vendar bi morali to v nalogi navesti eksplicitno. Podobna težava se pojavi tudi v besedilu naloge 1.4 (naraščajoče besede).

Pri 1337ovščini (2.1) se nekateri pritožujejo, da so nekatere izmed zamenjav, prikazanih v primeru v besedilu nalog, preveč neživljenjske in jih ne bi nihče uporabil v praksi. Mnogi nalogo tudi pohvalijo, ker jim je ta tematika očitno blizu.

Nalogo z drevesi končnic (2.4) je precej ljudi pohvalilo, češ da je zanimiva; žal pa večini reševalcev ni šla dobro od rok, še posebej podvprašanje (b).

Nekaj tekmovalcev v prvi skupini je opozorilo tudi na to, da so bile vse naloge v prvi skupini tipa „napiši (pod)program“, nobena pa „opiši postopek“. Ta očitek je čisto upravičen in bomo poskusili prihodnje leto bolj paziti na uravnoteženost celotnega nabora nalog.

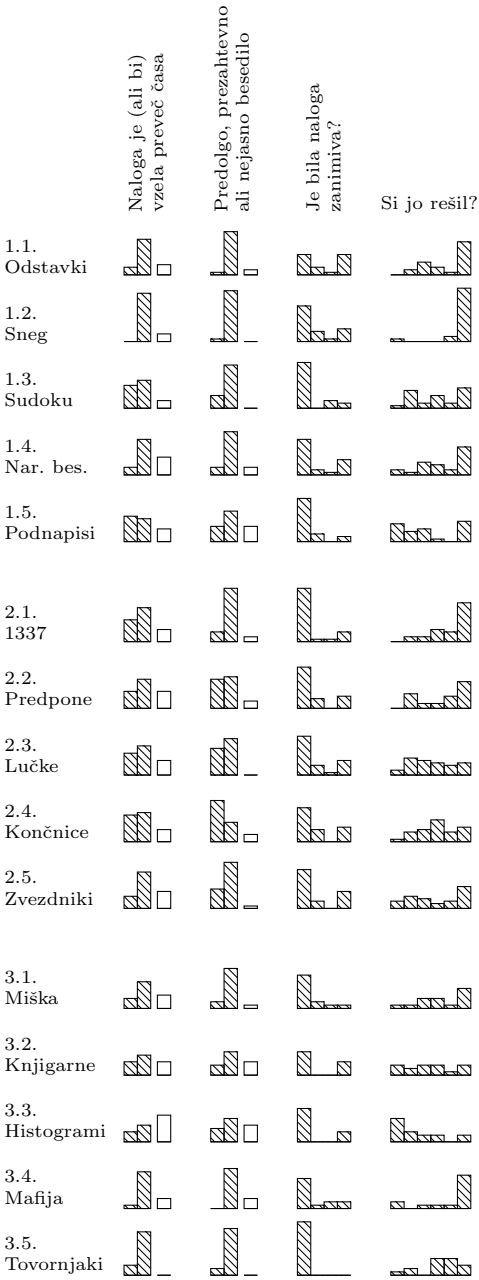


Mnenje tekmovalcev o zahtevnosti nalog in število doseženih točk

Pomen stolpcev v vsaki vrstici:

Na levi je skupina šestih stolpcev, ki kažejo, kako so tekmovalci v anketi odgovarjali na vprašanje o zahtevnosti naloge. Stolpci po vrsti pomenijo odgovore „prelahka“, „lahka“, „primerna“, „težka“, „pretežka“ in „ne vem“. Višina stolpca pove, koliko tekmovalcev je izrazilo takšno mnenje o zahtevnosti naloge.

Sledi stolpec, ki pokaže, kolikšen delež tekmovalcev je pri tej nalogi dobil več kot 0 točk. Naslednji par stolpcev pokaže povprečje (zgornji stolpec) in mediano (spodnji stolpec) števila točk pri vsej nalogi. Zadnji par stolpcev pa kaže povprečje in mediano števila točk, gledano le pri tistih tekmovalcih, ki so dobili pri tisti nalogi več kot nič točk.



Mnenje tekmovalcev o nalogah

Višina stolpcev pove, koliko tekmovalcev je dalo določen odgovor na neko vprašanje.

Stolpci se od leve proti desni nanašajo na naslednja vprašanja in možne odgovore:

Naloga je (ali: bi) vzela preveč časa:

- da
- ne
- ne vem

Predolgo/prezahtevno/nejasno besedilo:

- da
- ne
- ne vem

Naloga je bila:

- zanimiva
- dolgočasna
- že znana
- povprečna

Si jo rešil?

- zmanjkalo mi je časa za reševanje
- zmanjkalo mi je volje za reševanje
- nisem prišel daleč
- rešil sem le delno
- rešil sem skoraj v celoti
- rešil sem celo

Programersko znanje

Ko sestavljamo naloge, še posebej tiste za prvo skupino, nas pogosto skrbi, če tekmovalci poznajo ta ali oni jezikovni konstrukt ali programerski prijem. Zato smo se letos odločili, da jih v anketi za nekaj stvari kar naravnost vprašamo, če jih poznajo in bi jih znali uporabiti v svojih programih.

	Prva skupina	Druga skupina	Tretja skupina
zamikanje s **shl**, **shr**	46 %	14 %	65 %
operatorji na bitih	71 %	75 %	82 %
strukture	54 %	82 %	88 %
naštevni tipi	42 %	68 %	59 %
gnezdenje zank	88 %	96 %	100 %
zanka **while**	96 %	100 %	100 %
zanka **for**	96 %	100 %	100 %
kazalci	29 %	39 %	71 %
rekurzija	46 %	50 %	100 %
podprogrami	88 %	93 %	100 %
tabele (**array**)	92 %	93 %	100 %
delo z datotekami	46 %	68 %	94 %
str. vhod/izhod	88 %	93 %	94 %

Tabela kaže, kolikšen delež tekmovalcev je izjavil, da pozna določen konstrukt ali prijem.

Rezultati ankete so po tej plati kar spodbudni, saj jih velika večina tudi v prvi skupini pravi, da bi znala uporabljati tabele in napisati svoj podprogram; tudi gnezdenih zank se ne ustrašijo. (Seveda ni nujno, da imajo prav — mogoče se precenjujejo.)

Rekurzije v prvih dveh skupinah ne poznajo prav dosti. Na letošnjem tekmovanju je prišla rekurzija prav v drugi skupini (naloga 2.4, drevo končnic, podvprašanje (b)). Tudi operatorje za zamikanje bitov pozna zelo malo ljudi (ti bi prišli prav v nalogi 2.3 (cestne lučke), čeprav gre čisto dobro tudi brez njih).

Razmeroma malo tekmovalcev pozna naštevne tipe. Mogoče se je torej pametno izogibati uporabi naštevnih tipov v deklaracijah, ki jih vključujemo v besedila nalog. Pred nekaj leti smo v prvi skupini (2004.1.4) ponudili definiciji

```
type SmerT = (Gor, Dol, Ustavi);
typedef enum { Gor, Dol, Ustavi } SmerT;
```

in vsaj tista v C-ju je nekaj tekmovalcev pošteno zmedla — bolje bi bilo namesto posebnega tipa SmerT uporabljati navaden **int** in definirati tri konstante z imeni Gor, Dol in Ustavi. V pascalu je to mogoče manjši problem, ker je sintaksa malo manj zastrašujoča.

Uporaba programskih jezikov

Večina tekmovalcev uporablja programske jezike pascal, C in C++. Letos je bil pascal v izraziti manjšini, kar je pomembna razlika v primerjavi s prejšnjimi leti, ko je večina tekmovalcev uporabljala pascal.

	Leto in skupina						
	2006			2004			2003
Jezik	1	2	3	1	2	3	3
pascal	6	5	5	23	20	13	17
C	4	16	$1\frac{1}{2}$	13	$7\frac{1}{2}$	1	4
C++	13	5	$10\frac{1}{2}$	5		6	5
java			3		$\frac{1}{2}$		
PHP	1						
basic		1					
nič	1	2		3	2		

Število tekmovalcev, ki so uporabljali posamezni programski jezik.
Dva sta uporabljala po dva različna jezika (pri različnih nalogah) in sta šteta polovično k vsakemu jeziku. „Nič“ pomeni, da tekmovalec ni napisal nič izvorne kode.

Glede štetja C in C++ v gornji tabeli je treba pripomniti, da je razlika med njima tako ali tako zelo majhna: tisti, ki delajo v C++, uporabljajo večinoma le malo stvari, ki jih C++ ima, C pa ne. To so ponavadi <iostream> in vhodno/izhodna tokova cin in cout namesto C-jevih funkcij scanf, printf in podobnih.

V besedilu nalog za 1. in 2. skupino objavljamo deklaracije tipov, spremenljivk, podprogramov ipd. v pascalu in C/C++. Tekmovalce smo v anketi vprašali, če te deklaracije razumejo ali pa bi morale biti še v kakšnem drugem jeziku; veliki večini sedanje deklaracije zadostujejo (22/24 v prvi skupini in 27/28 v drugi). Trije predlagajo, da bi dodali še deklaracije v C#, eden v pythonu in eden v PHPju.

V rešitvah nalog pa smo doslej objavljali le izvorno kodo v pascalu. Tekmovalce vseh treh skupin smo v anketi vprašali, če pascal razumejo dovolj, da si lahko kaj pomagajo s to izvorno kodo, in če bi radi videli izvorno kodo rešitev še v kakšnem drugem jeziku. S pascalom jih je zadovoljnih 11/24 v prvi skupini, 16/28 v drugi in 15/17 v tretji skupini. Večina tekmovalcev, tudi tistih, ki pravijo, da pascal sicer razumejo dovolj dobro, je priporočila še objavo rešitev v C ali C++ (točne številke: 13 jih je predlagalo C, 7 C/C++, 22 C++, 2 C#, 2 PHP in 2 javo).

Zaključek tega dela ankete je, da je poznavanje pascala letos precej manjše kot prejšnja leta in bo treba v bodoče objavljati rešitve tudi v C ali C++. Verjetno je C vendarle primernejši kot C++ (tisti, ki sicer ponavadi delajo v C++, najbrž bolje razumejo C kot pa razumejo C++ tisti, ki ponavadi delajo v C). Zato smo tudi v letošnji bilten dodali rešitve v C-ju za vse tri skupine. Predvidoma bomo tudi prihodnja leta objavljali rešitve tako v pascalu kot v C-ju.

Letnik

Po pričakovanjih so tekmovalci zahtevnejših skupin v povprečju v višjih letnikih kot tisti iz lažjih skupin. Niso pa te razlike ravno grozno velike in tudi v prvi skupini je precej tekmovalcev iz četrtega letnika.

Skupina	Št. tekmovalcev po letnikih					
	1	2	3	4	neznan	povprečje
prva	7	1	11	4	1	2,5
druga		12	8	7	1	2,8
tretja	1	1	4	10		3,4

Druga vprašanja

Velikanska večina tekmovalcev je za tekmovanje izvedela prek svojih mentorjev (za kar se slednjim ob tej priložnosti še enkrat prav lepo zahvaljujemo). Ostali so

večinoma izvedeli za tekmovanje prek spletnih strani; dva omenjata našo stran (`rtk.ijs.si`), skoraj vsi ostali pa portal Slo-Tech (`www.slo-tech.com`), ki je prijazno objavil novico o našem tekmovanju in reklamno pasico s povezavo na našo spletno stran.

Programirati so se naučili večinoma sami, v drugi skupini pa je še več takih, ki so se programirati naučili v šoli. Verjetno je razmerje med številom enih in drugih odvisno v precejšnji meri od tega, koliko tekmovalcev prihaja s splošnih srednjih šol (ki v šoli nimajo veliko programiranja, če sploh kaj), koliko pa z računalniških.

Pri času reševanja in številu nalog je največ takih, ki so s sedanjo ureditvijo zadovoljni. V tretji in še zlasti drugi skupini je precej tudi takih, ki si želijo manj nalog. Glede časa pa, kolikor si že želijo sprememb, si želijo več časa za reševanje nalog, ne manj.

	Kje si izvedel za tekmovanje				Kje si se naučil programirati					Čas reševanja			Število nalog			Potekmovalne dejavnosti							
Skupina	od mentorja	na spletni strani	od prijatelja/sošolca	drugače	sam	pri pouku	na krožkih	na tečajih	poletna šola	hočem več časa	hočem manj časa	je že v redu	hočem več nalog	hočem manj nalog	je že v redu	izlet v tuji laboratoriji	poletna šola	praksa na IJS	predstavitve tehnologij	predavanja o algoritmih	reševanje nalog	iskanje štipendije	iskanje podjetij
I	19	7	1	1	19	8	5	1	0	5	2	16	4	5	14	14	14	12	12	13	12	7	15
II	21	5	2	1	14	17	4	2	1	5	4	19	1	13	14	10	12	8	11	9	7	12	12
III	14	2	1	3	13	6	5	0	2	3	1	11	1	7	7	9	11	8	8	10	6	5	9

Iz odgovorov na vprašanje, kakšne potekmovalne dejavnosti bi jih zanimale, je težko zaključiti kaj posebej konkretnega. Večina je odkljukala kar vse možnosti ali pa nobene (ker npr. tistega dela ankete sploh niso izpolnjevali).

V prvih dveh skupinah je več tekmovalcev predlagalo, da bi imeli možnost pisati odgovore z računalnikom namesto na papir. Ta zamisel bi bila zelo všeč tudi komisiji in jo bomo tudi izvedli, če bodo za to izpolnjeni organizacijski pogoji (predvsem dovolj računalnikov za tekmovalce prve in druge skupine). Tekmovalci bi tako lažje urejali svoje odgovore, komisiji pa se pri ocenjevanju ne bi bilo treba mučiti z razvozlavanjem načečkanih in nakracanih pisnih odgovorov. Seveda bi možnost pisanja odgovorov na papir še vedno dopustili, saj nočemo šikanirati tistih, ki jim gre tipkanje mogoče bolj počasi od rok.

Veliko tekmovalcev si tudi želi, da bi bilo tekmovanje med tednom, tako da bi imeli zaradi njega možnost izostati od pouka. Žal bo tekmovanje tudi v bodoče najbrž še vedno ob sobotah.

Nekaj tekmovalcev je opozorilo tudi na to, da se je naše tekmovanje prekrivalo z državnim tekmovanjem iz kemije. To je res, bojda pa je bil januarja, ko smo izbirali datum svojega tekmovanja, šesti maj še prost in so se kemiki nanj naselili kasneje. Takšnim trkom se je težko izogniti, ker je spomladi veliko tekmovanj in premalo sobot.

Z organizacijo tekmovanja je drugače velika večina tekmovalcev zadovoljna in nimajo posebnih pripomb.

Še nekaj utrinkov iz anket

Katera naloga ti je najbolj všeč? (1. skupina)
— Sneg, ker je vzeta iz življenja.
— Zadnja (podnapisi), ker je vsakodnevno uporabna.

Kaj te je pri tekmovanju najbolj motilo? (2. skupina)
— Da je potekalo na listih... In to je kao rač. tekmovanje.

Druge pripombe na tekmovanje. (3. skupina)
— Zaboga, uporabljajte XML in ne tekstovnih datotek.
— Več nalog brez standardnega vhoda in izhoda.[3]
— manjša vloga algoritmov, večjo vlogo „hard core“.

Posebno omembo za najbolj bizarno izpolnjeno anketo si zasluži naslednja anketa (iz 3. skupine):

(*mnenje o 1. nalogi*) `=)`

(*mnenje o 2. nalogi*) Lepa, pošteno povedano `:-*`

(*mnenje o 3. nalogi*) Dejte naslednjič raj kake network-flowe namest tega `;-P`

(*mnenje o 4. nalogi*) Tut take lahke morjo bit za warming-up.

(*mnenje o 5. nalogi*) Lepa kot satan `:-*` Poklon sestavljalcu.

(*katera naloga ti je bila najbolj všeč?*) Vse naloge so lepe. —— je lep.

(*kaj ti je bilo pri tekmovanju všeč?*) Da je naloge sestavu —— + ostale legende; tipkovnice so bolše [*sic*] kot na FRI-ju

(*kako bi prepričal znance k udeležbi*) Hm, hm, hm... če človek ni enthusiast, ga je težko prepričat; Pavle pa bi pomojem pršu, sam je bla isti dan (dons) kemija `=P`

(*potekmovalne dejavnosti: praksa na IJS*) `=)`

(*potekm. dejavnosti: predavanja o algoritmih*) `=) =) =)` temu se ni moč odreči

(*potekm. dejavnosti: reševanje nalog*) A s tem so mišljene priprave? Obvezno!

[3] Pri čemer so bile *vse* v 3. skupini brez standardnega vhoda in izhoda.

CVETKE

V tem razdelku je zbranih nekaj zabavnih odlomkov iz rešitev, ki so jih napisali tekmovalci. V oklepajih pred vsakim odlomkom sta skupina in številka naloge.

(1.1) Nenavadno ime spremenljivke (program vanjo prebere vrstico besedila) in komentar:

```
char vrabec[101];
/* declaramo naš buffer in zastavico in se opravičimo za neknjižni jezik */
```

(1.1) Rešitev z veliko spremenljivkami:

> Najprej v programu deklariramo vse spremenljivke, ki jih bomo potrebovali v programu. Za ta program bi rabili veliko spremenljivk.
>
> Torej, ko deklariramo vse spremenljivke, začnemo program pisati. Ko napišemo neko besedilo oz. besedo v program nam program mora to besedo oz. besedilo si raztaviti na vsako črko posebej. Primer če vnesemo besedo `"Test"`, si program vsako črko shrani pod določeno spremenljivko. Boljše bi sicer bilo, da bi uporabili tabelo, ki bi si to besedo zapomnila (npr. $\overset{1}{\boxed{\mathtt{t}}}\overset{2}{\boxed{\mathtt{e}}}\overset{3}{\boxed{\mathtt{s}}}\overset{4}{\boxed{\mathtt{t}}}$ in bi kasneje v programu klicali vsako okence v tabeli. Potem bi napisali funkcijo, ki bi nam vsako okence primerjala z črkami in bi jo zamejala in si jo shranila pod drugo spremenljivko.
>
> [...]
>
> Zdaj pa napišemo podprogram, ki si zapomni, katere spremenljivke smo uporabili in jih zamenjali ter nakoncu te spremenljivke izpiše (v spremenljivke so pa shranjeni znaki, torej izpiše znake). Tu program zaključimo.

(1.2) Komentar v eni od rešitev:

```
/* se opravičujem za neestetsko prečrtano vrstico */
```

(1.2) Komentar ob rešitvi, ki je najprej prebrala vse podatke, v tabeli izračunala debelino snežne odeje za vsak dan in šele na koncu izpisala vse debeline:

> Predpostavil sem, da želimo vse podatke izpisane hkrati po končanem vnašanju. Zato sem debelino definiral kot kar obsežno polje. Če bi podatke vnašal in izpisoval sproti, bi dobil precej manj spominsko požrešen program, a obenem tudi manj pregleden. Tudi ta izpis ni preveč pregleden, saj ima v vsaki vrstici izpisan le en podatek.

(1.3) Še en zabaven komentar:

```
/* na nadvse zoprn način, ki bi ga bilo še zoprnejše ubesediti, preverimo
   še ustreznost 3 × 3 kvadratkov */
```

(1.4) Eden od načinov, kako deklarirati spremenljivko v C-ju (isti program je deklariral tudi nekaj spremenljivk na pravilen način):

```
#define c = 0; // definiramo spremenljivko za naraščajoče število
```

(1.5) Rešitev poskuša v podprogramu Inicializacija deklarirati spremenljivko, ki bi jo uporabljal tudi PodnapisZaSlicico. Reševalec je, kot kaže, razumel, da ne bi bilo dobro, če bi Inicializacija ustvarila to spremenljivko lokalno pri sebi na skladu, zato je naredil tole:

```
int *count = malloc(sizeof(int), 0);
*count = 0;
```

Ker pa je ta kazalec še vedno lokalna spremenljivka podprograma Inicializacija, si PodnapisZaSlicico ne bo mogel pomagati z njim. Če pa bi ga spremenili v globalno spremenljivko, bi odpadla potreba po alokaciji s kopice...

(2.1) Kako pritakniti niz c na konec niza b, ki je trenutno dolg j znakov:

```
k = strlen(c);
switch (k)
{
   case 1:
      b[j] = c[0]; j++;
      break;
   case 2:
      b[j] = c[0]; j++;
      b[j] = c[1]; j++;
      break;
   case 3:
      b[j] = c[0]; j++;
      b[j] = c[1]; j++;
      b[j] = c[2]; j++;
      break;
   default:
      printf("Nisem vedel da obstajajo daljši od tri.");
}
```

(2.1) Celotna rešitev te naloge pri enem od reševalcev (vključno z navpičnimi tropičji):

```
#include <stdio.h>
#include <stdlib.h>
int Nakljucje(int n)
⋮
char MozneZamenjave(char c)
⋮
void main()
```

(2.3) Iz ene od rešitev te naloge:

> iz tabel ne znam brati števil, tako da bom samo opisal potek programa

(2.5) Nek reševalec pri nalogi 2.5 dosledno piše „zvezniki“ namesto „zvezdniki“.

(3.3) Nagrado za najglobljo indentacijo dobi...

```
procedure FindSolution;
:
  procedure Solve;
  :
  begin
    repeat
      Done := false;
      for I := Hi downto Lo do
        begin
          if Hist[I] < Mean
            then
              begin
                if I <= Lo
                  then
                    begin
                      Done := true;
                      break;
                    end;
```

(3.3) Nagrada za najbolj neumestno poimenovanje spremenljivk:

```
if not Krneki then
begin
  for fak := 5 to 256 do
  begin
    if Slika[j, i] = fak then
    begin
      Slika[j, i] := fak + 1;
    end;
  end;
  Krneki := true;
end;
```

(3.4) Kako prebrati dve celi števili iz ene vrstice v datoteki? Z ReadLn(T, a, b) že ne...

```
var T: text; s: string;
:
ReadLn(T, s);
Mafija[i].sur := StrToInt(Copy(s, 1, Pos(' ', s) - 1));
Delete(s, 1, Pos(' ', s));
Mafija[i].oddan := StrToInt(s);
```

(3.4) Ne pustimo heap managerju lenariti:

```
typedef struct node { int poslan, dobljen, nad; } clan;
:
clan *mafijci[100002];
for (i = 1; i <= n; i++) { mafijci[i] = (clan *) malloc(sizeof(clan)); ... }
```

SODELUJOČE INŠTITUCIJE

Institut Jožef Stefan
Institut je največji javni raziskovalni zavod v Sloveniji s skoraj 800 zaposlenimi, od katerih ima približno polovica doktorat znanosti. Več kot 150 naših doktorjev je habilitiranih na slovenskih univerzah in sodeluje v visokošolskem izobraževalnem procesu. V zadnjih desetih letih je na Institutu opravilo svoja magistrska in doktorska dela več kot 550 raziskovalcev. Institut sodeluje tudi s srednjimi šolami, za katere organizira delovno prakso in jih vključuje v aktivno raziskovalno delo. Glavna raziskovalna področja Instituta so fizika, kemija, molekularna biologija in biotehnologija, informacijske tehnologije, reaktorstvo in energetika ter okolje.

Poslanstvo Instituta je v ustvarjanju, širjenju in prenosu znanja na področju naravoslovnih in tehniških znanosti za blagostanje slovenske družbe in človeštva nasploh. Institut zagotavlja vrhunsko izobrazbo kadrom ter raziskave in razvoj tehnologij na najvišji mednarodni ravni.

Institut namenja veliko pozornost mednarodnemu sodelovanju. Sodeluje z mnogimi uglednimi institucijami po svetu, organizira mednarodne konference, sodeluje na mednarodnih razstavah. Poleg tega pa po najboljših močeh skrbi za mednarodno izmenjavo strokovnjakov. Mnogi raziskovalni dosežki so bili deležni mednarodnih priznanj, veliko sodelavcev IJS pa je mednarodno priznanih znanstvenikov.

Tekmovanje so podprli naslednji odseki IJS:

CT3 — Center za prenos znanja na področju informacijskih tehnologij
Center za prenos znanja na področju informacijskih tehnologij izvaja izobraževalne, promocijske in infrastrukturne dejavnosti, ki povezujejo raziskovalce in uporabnike njihovih rezultatov. Z uspešnim vključevanjem v evropske raziskovalne projekte se Center širi tudi na raziskovalne in razvojne aktivnosti, predvsem s področja upravljanja z znanjem v tradicionalnih, mrežnih ter virtualnih organizacijah. Center je partner v več EU projektih.

Center razvija in pripravlja skrbno načrtovane izobraževalne dogodke kot so seminarji, delavnice, konference in poletne šole za strokovnjake s področij inteligentne analize podatkov, rudarjenja s podatki, upravljanja z znanjem, mrežnih organizacij, ekologije, medicine, avtomatizacije proizvodnje, poslovnega odločanja in še kaj. Vsi dogodki so namenjeni prenosu osnovnih, dodatnih in vrhunskih specialističnih znanj v podjetja ter raziskovalne in izobraževalne organizacije. V ta namen smo postavili vrsto izobraževalnih portalov, ki ponujajo že za več kot 500 ur posnetih izobraževalnih seminarjev z različnih področij.

Center postaja pomemben dejavnik na področju prenosa in promocije vrhunskih naravoslovno-tehniških znanj. S povezovanjem vrhunskih znanj in dosežkov različnih področij, povezovanjem s centri odličnosti v Evropi in svetu, izkoriščanjem različnih metod in sodobnih tehnologij pri prenosu znanj želimo zgraditi virtualno učečo se skupnost in pripomoči k učinkovitejšemu povezovanju znanosti in industrije ter večji prepoznavnosti domačega znanja v slovenskem, evropskem in širšem okolju.

E2 — Odsek za sisteme in vodenje
Poslanstvo odseka, ki smo si ga postavili kot vodilno nit ob začetku delovanja, smo opredelili kot „premoščanje prepada med teorijo in prakso“.

Osrednji poudarek pri načinu našega dela je na zahtevi, da je treba izhajati iz konkretnih (industrijskih) problemov ter raziskave prilagajati temu, ne pa nasprotno. To dejstvo seveda potegne za seboj potrebo po širokem območju zelo različnih znanj, tako po tipu dela (raziskave, razvoj, inženirstvo itd.) kot tudi stroki (avtomatika, elektronika, računalništvo itd.).

Poslanstvo je izšlo iz potreb domačega okolja, saj so izkušnje pokazale, da spoznanj iz raziskav praktično ne moremo uporabiti pri reševanju konkretnih aplikativnih problemov in je s tem tudi opredelilo način našega dela.

Dejavnosti odseka obsegajo raziskave, razvoj in aplikacije na širšem področju računalniško podprtega vodenja in regulacije (predvsem) tehničnih sistemov, skupaj z ustreznimi storitvami, inženirstvom in izobraževanjem

S svojim znanjem vam lahko pomagamo na področju elektronike, merilne in regulacijske tehnike, računalniške avtomatizacije in informatizacije procesov.

Samo v času od svoje formalne ustanovitve (1986) je odsek delal večje ali manjše projekte za okoli 100 slovenskih in tujih podjetij, večinoma iz sektorja industrijske proizvodnje.

E5 — Laboratorij za odprte sisteme in mreže
Laboratorij za odprte sisteme in mreže je bil ustanovljen leta 1992. Aktivnosti laboratorija so usmerjene v raziskave in razvoj omrežij naslednje generacije, telekomunikacijskih tehnologij, komponent in integriranih sistemov ter storitev in aplikacij informacijske družbe. Laboratorij izvaja raziskovalni program Tehnologije, storitve in poslovanje v omrežjih naslednje generacije, raziskave pa potekajo še v mednarodnih projektih iz 6. okvirnega programa EU, projektu PHARE ter v nekaj nacionalnih projektih.

Raziskovalni cilji so usmerjeni v zagotovitev znanstvenih, raziskovalnih in razvojnih rezultatov na področju omrežij naslednje generacije in na njih temelječih storitvah in aplikacijah. Znanstvene raziskave potekajo v smeri zagotavljanja pomembnega mesta slovenskim raziskovalnim dosežkom v svetovnem in evropskem merilu na področju generičnih tehnologij in aplikacij, ki so jedro ekonomije znanj.

Glavna področja dela so tehnološko podprto učenje, varnost in zasebnost v informacijskih sistemih ter napredna omrežja naslednje generacije. Raziskovalne vsebine so osredinjene na ključne teme tehnologij in aplikacij informacijske družbe:

- telekomunikacijske tehnologije, komponente in omrežni integrirani sistemi
- storitve in aplikacije
- znanstvenoraziskovalne aktivnosti za spremljanje razvoja telekomunikacij in sprejemljivosti tehnologij v družbeno infrastrukturo.

E8 — Odsek za tehnologije znanja
Poslanstvo Odseka za tehnologije znanja IJS je razvoj naprednih informacijskih tehnologij za zajemanje, shranjevanje, upravljanje in odkrivanje znanja s poudarkom na rudarjenju prodatkov oz. strojnem učenju, podpori odločanja in razvoju jezikovnih tehnologij, katerih cilj je prispevati vrhunske znanstvene rezultate v svetovno

zakladnico znanja ter pospeševati aplikacije teh tehnologij za razvoj e-znanosti in družbe znanja.

Dolgoročni cilji odseka so razvoj metod inteligentne analize podatkov, upravljanja znanja, podpore odločanja in računalniškega jezikoslovja ter njihova uporaba za reševanje praktičnih problemov na področju ekologije, medicine, zdravstvenega varstva, ekonomije in tržništva. V raziskave vključujemo tudi novejša področja informacijskih tehnologij: semantični splet in upravljanje mrežnih organizacij.

E9 — Odsek za inteligentne sisteme

Osnovni cilji Odseka za inteligentne sisteme so raziskave računalniških osnov inteligence in razvoj naprednih aplikacij s področja inteligentnih informacijskih storitev, analize podatkov, inteligentnega preiskovanja spleta, podpore odločanja, inteligentnih agentov, medicine, ekologije, jezikovnih tehnologij, inteligentne proizvodnje in ekonomije. Z več kot 20-letno tradicijo pri raziskavah in razvoju na širšem področju umetne inteligence, inteligentnih sistemov, medicinske informatike, procesiranja naravnega jezika in kognitivnih znanosti se je Odsek za inteligentne sisteme uveljavil v evropskem in svetovnem merilu. Sodelavci odseka so razvili več delujočih sistemov, ki so pomembni tako v slovenskem kot mednarodnem merilu.

Raziskovalna področja Odseka za inteligentne sisteme:

- induktivno logično programiranje
- evolucijsko računanje
- večstrategijsko učenje in principi mnogoterega znanja
- rudarjenje spletnih podatkov — sinteza znanja za modeliranje in vodenje sistemov
- sistemi za podporo odločanja
- principi inteligence in kognitivnih znanosti
- inteligentni agenti in večagentni sistemi
- umetna inteligenca v medicini
- sinteza govora
- ontologije in semantični splet
- analiza igranja iger.

*

Fakulteta za matematiko in fiziko

Fakulteta za matematiko in fiziko je članica Univerze v Ljubljani. Sestavljata jo Oddelek za matematiko in Oddelek za fiziko. Izvaja dodiplomske univerzitetne študijske programe matematike, računalništva in informatike ter fizike na različnih smereh od pedagoških do raziskovalnih.

Prav tako izvaja tudi podiplomski specialistični, magistrski in doktorski študij matematike, fizike, mehanike, meteorologije in jedrske tehnike.

Poleg rednega pedagoškega in raziskovalnega dela na fakulteti poteka še vrsta obštudijskih dejavnosti v sodelovanju z različnimi institucijami od Društva matematikov, fizikov in astronomov do Inštituta za matematiko, fiziko in mehaniko ter Instituta Jožef Stefan. Med njimi so tudi tekmovanja iz programiranja, kot sta Programerski izziv in Slovensko univerzitetno prvenstvo v programiranju.

DONATORJI

Advansys

Advansys razvija, proizvaja in trži informacijske sisteme za potrebe igralništva. Podjetje je bilo ustanovljeno konec leta 2003. Pri dejavnosti podjetja je danes aktivnih 16 sodelavcev in vrsta podizvajalcev.

Advansys svojo konkurenčno prednost gradi na tehnološkem znanju in inovativnosti, posebno pozornost pa namenja odnosom s strankami.

Advansysov produkt SlotScanner™ je sistem, ki zagotavlja popoln nadzor nad delovanjem igralnih avtomatov in blagajn, obenem pa nudi nepretrgano podporo vsem operativnim procesom oddelka igralnih avtomatov. SlotScanner™ zagotavlja preglednost in varnost vseh denarnih transakcij, omogoča pa tudi analize finančnega poslovanja igralnih avtomatov ter meritve njihove zasedenosti.

SlotScanner™Pro omogoča nadzor in upravljanje nad celotnim igralniškim poslovanjem z vidika trženja in odnosov z gosti. Sistem igralniškemu osebju omogoča identificiranje najdonosnejših strank, povečevanje njihove zvestobe in vrednosti za igralnico. SlotScanner™Pro je eno ključnih orodij marketinških oddelkov v igralnici.

S sistemom SlotScanner™ Advansys v sloveniji dosega 75 % tržni delež. Sistem doma in v tujini podpira delovanje skoraj 5.000 igralnih avtomatov, kar podjetje v svetovnem merilu uvršča med deset najuspešnejših podjetij, proizvajalcev informacijskih sistemov za podporo igralnih avtomatov.

Aldata

Aldata je mednarodno podjetje s podružnicami v Evropi, Ameriki in na daljnem Vzhodu. Specializirali smo se za izdelavo poslovnih rešitev na področju maloprodaje in logistike. Aldata sporočilo je enostavno — *maloprodaja, danes in jutri.* Več kot 200 velikih svetovnih verig in preko 20.000 inštalacij naše programske opreme v več kot 40 državah po svetu je dokaz, da razumemo potrebe naših strank, zahteve trga in da ponujamo sodobno in preverjeno informacijsko rešitev.

Slovenska Aldata je ena najmanjših članic v skupini Aldata, s svojim teamom 30 sodelavcev pa znotraj skupine in pri strankah igra ključno vlogo že skoraj 10 let. V Sloveniji imamo močan oddelek razvoja produktnega portfolia G.O.L.D.®, pohvalimo pa se lahko z bogatimi izkušnjami in znanjem, ki smo si ga pridobili z delom na domačih in velikih mednarodnih projektih v 24 državah širom po svetu.

Delo v našem razvojnem in implementacijskem teamu predstavlja odlično priložnost na poti do uresničitve načrtovane kariere. Sodelovanje pri razvoju jedra produkta G.O.L.D.®, pri katerem tesno sodelujemo z našim francoskim partnerjem, se lepo dopolnjuje s razvojem in adaptacijo številnih specifičnih zahtev naših strank doma in v tujini. Sodelovanje je možno na različne načine: od analize in načrtovanja, do izvedbe in integracije v različna okolja, svetovanja in izobraževanja pa tudi vodenja razvojnih in implementacijskih projektov. Skupina Aldata, ki skupaj zaposluje več kot 500 ljudi, ima organizirano mednarodno bazo sodelavcev, preko katere zagotavlja kadre za potrebe izmenjave najustreznejših kadrov za določene projekte, kakor tudi za uvajanje novih kadrov. Več: `http://www.aldata-solution.com`.

ARNES

Javni zavod ARNES je bil ustanovljen z namenom, da skrbi za načrtovanje, organiziranje in upravljanje računalniških povezav med organizacijami s področja raziskovanja, razvoja, izobraževanja in kulture, za povezovanje v izobraževalna in raziskovalna omrežja v drugih državah in s tem posredno tudi v svetovni Internet. ARNES mora opravljati tudi določene centralne aktivnosti, ki omogočajo, da uporabniki lahko uporabljajo storitve na omrežju.

Javni zavod ARNES opravlja aktivnosti na naslednjih strokovnih področjih:

- zagotavljanje povezljivosti,
- priključevanje novih uporabnikov,
- višjenivojske storitve,
- pomoč uporabnikom,
- razvojne aktivnosti,
- izobraževanje in promocija.

Osnovne storitve, ki jih nudi omrežje uporabnikom, so:

- elektronska pošta,
- elektronski imenik,
- prenos datotek med računalniki,
- dostop do oddaljenih računalnikov,
- elektronske konference,
- multimedijske storitve,
- dostop do podatkovnih baz po celem svetu,
- vključitev lastnih podatkovnih baz v svetovno omrežje

Za zagotavljanje teh storitev je potrebno upravljati domenski in naslovni prostor, vzdrževati domenski strežnik in opravljati druge storitve, ki jih uporabnik direktno ne vidi. Pri vzpostavljanju povezav ARNES svetuje, kasneje pri delu pa pomaga pri reševanju problemov. Vedno bolj je pomembna ARNES-ova koordinatorska funkcija pri varnostnih incidentih na omrežju.

Business Solutions

Vabimo vas, da si pobliže ogledate in preizkusite naše rešitve na področju poslovne informatike in jih primerjate s konkurenčnimi. Verjamemo, da boste videli naš trud in znanje, ki smo ga s ponosom vložili v proizvode in storitve, ki vam bodo pomagali doseči in preseči vašo poslovno vizijo.

Veliko je načinov, kako prihraniti nekaj tolarjev in priti do nižje cene investicije: Enostavno uporabite rešitve z zastarelimi tehnologijami in metodami, rešitve ki onemogočajo možnost nadgradnje in prilagajanja specifičnim potrebam kupcev, vlagate manj denarja v raziskave in razvoj, ali pa enostavno preskočite male podrobnosti, ki so lahko ključne za uspeh.

V podjetju Business Solutions d.o.o. smo se od teh idej ogradili, saj je naše poslanstvo, da vam prinesemo največjo vrednost, ne najcenejše možne rešitve.

Hvala vam za vaše zanimanje za vrhunske izdelke in storitve, ki jih ponujamo. Veselimo se sodelovanja z vami pri doseganju vaše poslovne odličnosti.

Center za uporabno matematiko in teoretično fiziko
Center za uporabno matematiko in teoretično fiziko (CAMTP) je bil ustanovljen junija 1990 kot nov raziskovalni institut pod vodstvom profesorja dr. Marka Robnika. Neodvisna akademska institucija in pridružena članica Univerze v Mariboru je postal julija 1991. Financiranje centra je večinoma zagotovljeno z izvajanjem raziskovalnih projektov Ministrstva za šolstvo, znanost in šport. Raziskovalne aktivnosti in sodelovanje je močno mednarodno usmerjeno. Center ima veliko zunanjih sodelavcev in obiskovalcev, ki so vodilni znanstveniki v Sloveniji in tujini, posebej iz Nemčije, Japonske, Francije, Italije, Belorusije in ZDA.

Raziskovalni program obsega štiri glavna področja:

- nelinearna dinamika klasičnih in kvantnih neintegrabilnih in kaotičnih sistemov,
- hidrodinamika in teorija plazme,
- računalniška algebra,
- inteligentni sistemi in računalniško modeliranje kompleksnih sistemov.

Sodelujemo tudi na področju teoretične astrofizike v skupnem projektu z raziskovalci Univerze v Bonnu. CAMTP sodeluje tudi z Univerzo v Ljubljani. Center ima dvostranske sporazume o sodelovanju (Memorandum of Understanding) z Waseda Univerzo v Tokyu, Tehniško Univerzo v Münchnu, Institutom Rudjer Bošković iz Zagreba in še nekaterimi institucijami iz Minska v Belorusiji. Center je tudi ustanovitelj in pokrovitelj relativno nove mednarodne revije Nonlinear Phenomena in Complex Systems (NPCS), ki jo izdajajo v Minsku, njeno elektronsko verzijo pa vzdržujemo na CAMTP.

Glavne naloge centra so izvajanje teoretičnih raziskav, vzdrževanje mednarodnih akademskih povezav in sodelovanja na teh področjih in vzgoja mladih raziskovalcev. CAMTP ima zelo dobre računalniške zmogljivosti. Center redno organizira mednarodno poletno šolo/konferenco Let's Face Chaos through Nonlinear Dynamics.

CDE
CDE d.d. je začel delovati kot skupen projekt v poletnih mesecih leta 1993, in sicer z namenom razviti večpredstavnostne medijske programske opreme. Skupina desetih ljudi je postala majhna družba, ki je želela prodreti na razvijajoči se trg komunikacij. Prvi rezultati so bili prikazani na največjem slovenskem računalniškem sejmu INFOS 93. Glede na svetovni trend razvoja industrije se je CDE d.d. že leta 1994 začel usmerjati na področja informacijskih tehnologij in telekomunikacij ter integratorja le-teh.

Zavedamo se pomembnosti lastnega razvoja za vsako proizvodno podjetje z dolgoročno usmeritvijo. Zato je družba CDE d.d. ravno v preteklih letih veliko časa in sredstev vlagala v razvoj lastnih programskih orodij in aplikacij. Vzporedno je družbi uspelo skleniti dolgoročne dogovore o poslovnem sodelovanju z nekaterimi strateškimi partnerji s področja telekomunikacij, razvoja programskih aplikacij in večpredstavnosti. Ti uspehi zagotavljajo družbi dolgoročen in kontinuiran razvoj in utrditev položaja na slovenskem trgu računalništva.

V zadnjih letih smo skupaj z našimi partnerji in predstavniki predstavljali naše programske rešitve potencialnim strankam na najpomembnejših evropskih sejmih. Redno skrbimo za izobraževanje naših predstavnikov in sodelavcev, tako da so naše razvojne skupine seznanjene s tokom dogajanja s področja sodobnih tehnologij.

Tako se lahko CDE d.d. predstavi z uspehi tudi v tujini s spiskom tujih poslovnih partnerjev, zlasti iz sosednjih držav in republik nekdanje Jugoslavije.

Cosylab
V podjetju Cosylab (Control System Laboratory) razvijamo kontrolne sisteme za velike fizikalne eksperimentalne naprave, kot so jedrski pospeševalniki in radijski teleskopi. Ukvarjamo se tudi z drugimi področji: geografskimi informacijskimi sistemi, avtomobilsko elektroniko in telekomunikacijami. Bolj kot samo področje dela nas namreč motivira izziv, ki nam ga delo ponuja. Več informacij o dejavnosti podjetja si lahko poiščeš na spletnem naslovu `http://www.cosylab.com`.

Podjetje je bilo ustanovljeno leta 2001, zdaj pa nas je že več kot 40. Veliko izmed nas je s podjetjem pričelo sodelovati, ko smo bili še študenti ali celo dijaki. Tudi danes z veseljem v svojo sredino sprejmemo nadarjene mlade računalničarje, fizike, elektrotehnike, matematike, ... Če te sodelovanje z nami zanima, si več o tem preberi na spletni strani `http://www.cosylab.com/Cosylab/Employment`.

Za zabavo pa si lahko poiščeš svoje bivališče na zračnem posnetku Slovenije s pomočjo spletne aplikacije, ki smo jo razvili za kmetijsko ministrstvo: `http://rkg.gov.si/GERK/viewer.jsp`.

Infonet
Naša dejavnost:

- razvoj in vpeljava računalniških informacijskih sistemov v zdravstveni in lekarniški dejavnosti v skladu z evropskimi in svetovnimi standardi,
- svetovanje pri razvoju in izgradnji informacijskih sistemov v zdravstvu,
- povezovanje različnih zdravstvenih informacijskih sistemov na osnovi mednarodnih standardov za komunikacijo in izmenjavo sporočil,
- vzdrževanje sistemov.

Smo vodilni proizvajalec informacijskih sistemov v zdravstveni in lekarniški dejavnosti v Sloveniji, nekaj naših rešitev pa uspešno uporabljajo tudi v Srbiji in Makedoniji. Naši programi nudijo podporo strokovnjakom v večini slovenskih bolnišnic, polovici zdravstvenih domov, dveh tretjinah lekarn, nekaj socialnih domovih in domovih ostarelih in v več kot 400 zasebnih ambulantah. Naši programi tako tečejo na približno 4000 računalnikih, uporablja jih nekaj manj kot 6000 delavcev v zdravstvu.

Pri delu se nenehno srečujemo z novimi izzivi, tako domenskimi kot tehnološkimi. Delo pri nas je izredno zanimivo in nudi veliko osebnega zadovoljstva in možnost nenehnega nadgrajevanja. Glavni vrlini v našem kolektivu sta kvalitetno delo in pristni osebni odnosi.

Klika
Podjetje Klika je ponudnik inovativnih lokacijskih storitev, kot sta TripTracker in FriendLocator. Nudimo tudi storitve napodročju razvoja programske opreme. Specializiramo se za projekte, ki so tehnično in vsebinsko na visokem nivoju in pri izvedbi uporabljajo napredne tehnologije. Med drugim lahko nanizamo reference s področij, kot so spletne aplikacije in spletni uporabniški vmesniki, razvojna okolja za poslovne aplikacije (Enterprise Java, Microsoft .NET), elektronsko poslovanje in „embedded“ sistemi (Symbian, Windows CE).

Podjetje Klika d.o.o. je bilo ustanovljeno leta 2003. Podjetje vodi motivirana in uigrana skupina računalniških strokovnjakov, ki se lahko pohvali z dolgoletnimi skupnimi izkušnjami pri razvoju programske opreme. Osredotočamo se na moderne spletne in mobilne tehnologije. Naše glavne stranke so končni uporabniki, mobilni operaterji, potovalne agencije, in tudi podjetja s programsko opremo, ki pri lastnem razvoju koristijo naše strokovne storitve.

Marand — napredna računalniška hiša
Podjetje Marand Inženiring d. o. o. je eno vodilnih slovenskih podjetij na področju informacijsko-komunikacijskih tehnologij in informacijske podpore poslovnih procesov. Naša osnovna prednost je stalno izpopolnjevano lastno znanje in izkušnje, ki nam omogočajo, da lahko naročnikom ponudimo tako izdelke vodilnih svetovnih ponudnikov informacijskih tehnologij kot tudi lastne rešitve.

Naročnikom zagotavljamo celovito podporo vsem informacijskim funkcijam in procesom, od osebnih računalnikov, do največjih strežniških sistemov, od svetovanja pri izbiri opreme do sistemske integracije in razvoja prilagojenih programskih rešitev, od izobraževanja uporabnikov do celovite ponudbe vzdrževanja vseh vrst informacijskih sistemov in opreme. Kupcem ponujamo celovite informacijske rešitve, prilagojene njihovim željam in potrebam. Razvili smo lastne informacijske rešitve za kompleksne poslovne sisteme in procese:

- sistemi za zaračunavanje storitev
- sisteme za upravljanje z vsebinami
- zdravstveni informacijski sistem

Uspeh podjetja dopolnjuje naša ponudba storitev vzdrževanja in podpore, izobraževanja in svetovanja, sistemske integracije ter razvoja celovitih informacijskih sistemov. Posebna prednost so naše večletne izkušnje sistemske integracije na specializiranih področjih sporočilnih sistemov ter spletnih in govornih portalov.

Marg
Smo mlado podjetje za izdelavo vrhunske programske opreme, ki so ga leta 2004 ustanovili investitorji Andrej Marčič, Branko Šaletič (ustanovitelja podjetja Marand) ter Andrej Kuščer (ustanovitelj podjetja HERMES SoftLab).

- Vizija
  Na trgu programskih rešitev postati znanilec sprememb in vodilni ponudnik generičnih produktov.

- Poslanstvo
  MARG svoje poslanstvo vidi v produktizaciji rešitev, ki povečujejo uporabnost tehnologij. Svoje produkte ponujamo pod lastnimi in tujimi blagovnimi znamkami.

- Ekipa
  Vodstvena in razvojna ekipa se lahko pohvalita z izkušnjami pri trženju produktov, razvoju in uporabi kvalitetnih tehnologij in dolgoročnim izvajanjem rasti uspešnosti v mednarodnih okvirih.

Najdi.si

Najdi.si — srce slovenskega spleta!

Najdi.si je osrednje slovensko internetno križišče, saj se dnevno vrača tisoče in tisoče uporabnikov, ki iščejo na spletu. Ključna naloga Najdi.si je, da iz nepregledne množice slovenskih spletnih dokumentov, za vsakega uporabnika v vsakem trenutku, na podlagi kratke iskalne poizvedbe, vrne najbolj relevantne zadetke.

Najdi.si je v zadnjih nekaj letih postal daleč najbolj obiskana spletna stran. Uporablja ga že preko 800.000 Slovencev, kar je večji doseg kot vodilni dnevnoinformativni časopisi v Sloveniji.

Najdi.si uporabnikom ponuja tudi uporabna brezplačna orodja (družinski filter, iskalni okvirji, orodna vrstica itd.) in razvijajoči se spletni centri (Novice, Avtomobili, E-razglednice...) z bogato in zanimivo vsebino. Z Najdi.si pa lahko uporabnik išče, tudi kadar ne sedi za računalnikom — mobilni iskalnik Najdi.si je prilagojen iskanju na mobilnem telefonu.

Geslo spletnega iskalnika Najdi.si je „Išči pametneje“, ki sporoča prednost za uporabnika, da z iskalnikom Najdi.si najde več, hitreje in natančneje kot z drugimi iskalniki.

Najdi.si — Išči pametneje.

Quintelligence

Obstoječi informacijski sistemi podpirajo predvsem procesni in organizacijski nivo pretoka podatkov in informacij. Biti lastnik informacij in podatkov pa ne pomeni imeti in obvladati znanja in s tem zagotavljati konkurencne prednosti. Obvladovanje znanja je v razumevanju, sledenju, pridobivanju in uporabi novega znanja. IKT (informacijsko-komunikacijska tehnologija) je postavila temelje za nemoten pretok in hranjenje podatkov in informacij. S primernimi metodami je potrebno na osnovi teh informacij izpeljati ustrezne analize in odločitve. Nivo upravljanja in delovanja se tako seli iz informacijske logistike na mnogo bolj kompleksen in predvsem nedeterminističen nivo razvoja in uporabe metodologij. Tako postajata razvoj in uporaba metod za podporo KM vedno pomembnejši segment razvoja.

Podjetje Quintelligence je in bo usmerjeno predvsem v razvoj in izvedbo metod in sistemov za pridobivanje, analizo, hranjenje in prenos znanja. S kombiniranjem delnih — problemsko usmerjenih rešitev, gradimo kompleksen in fleksibilen sistem za podporo KM, ki bo predstavljal osnovo globalnega informacijskega centra znanja.

Obvladovanje znanja je v razumevanju, sledenju, pridobivanju in uporabi novega znanja.

SUN d. o. o.

SUN d. o. o. — kreativne rešitve, podjetje s sedežem v Novi Gorici, je specializirano za nudenje kreativnih poslovnih rešitev, ki združujejo najnovejše metodologije in tehnologije tako na področju razvoja produktov kakor tudi njihove realizacije. Naše veščine so podprte s tehničnim znanjem ter večletnimi izkušnjami, ki nam zagotavljajo izdelavo poslovnih platform in aplikacij, ki se skladajo z načini modernega poslovanja. Skupaj z zunanjimi sodelavci ustvarimo skupine, ki zagotavljajo učinkovite kreativne rešitve. Naše delo obsega:

- poslovno svetovanje — razvoj poslovnih strategij, analiza poslovnih potreb, tržna analiza, izdelava poslovnih in investicijskih programov, finančno svetovanje
- tehnološko svetovanje — definiranje IT strategij, nadzor in vodenje procesov, združevanje tehnologij
- projektno upravljanje — razvoj projektne pisarne, razvoj in uvajanje metod, planiranje in nadzor, upravljanje procesov, finančni inženiring, inženiring finančnih spodbud
- integracija sistemov — programske in podatkovne povezave, upravljanje poslovnih procesov

Za doseganje trajnostnega razvoja tako podjetja kot tudi okolja je potrebno slediti novim znanjem in tehnologijam, ki vodijo k lastnemu napredku. Pri nas spremembam ne sledimo ampak jih skupaj s partnerji kreiramo.

Temida

Podjetje Temida je bilo ustanovljeno leta 1991. Osnovni dejavnosti podjetja sta:

- izdelava zahtevne programske opreme po naročilu in
- svetovanje uporabnikom pri rešitvah na področju informacijske tehnologije.

Pri tem uporabljamo sodobne tehnološke rešitve in platforme. Dolgoročna strategija našega podjetja je zagotavljati visoko kvaliteto svojih storitev in izdelkov, ohranjati strokovnost in vzpostavljati dobre poslovne odnose s svojimi uporabniki in poslovnimi partnerji v domačem in mednarodnem okolju.

Izdelava informacijskih rešitev:

- posnetek stanja, sistemska analiza in funkcionalna specifikacija,
- načrtovanje računalniških programskih sistemov,
- izdelava računalniških aplikacij
- zagotavljanje kakovosti pri izvedbi projektov,
- integriranje projektov v obstoječe okolje,
- načrtovanje vzdrževanja in vzdrževanje programskih sistemov.

Svojo ponudbo oblikujemo na podlagi stalnega spremljanja zahtev naših strank, globalnih usmeritev, ter sodelovanja s partnerji. Naše delovanje je dolgoročno in zanesljivo.

Svetovanje uporabnikom:

- izdelava študij in analiz obstoječih informacijskih sistemov,
- svetovanje pri nadgradnjah, razvoju in vzdrževanju informacijskih sistemov,
- izobraževanje uporabnikov.

Trg informacijskih tehnologij narekuje hitre spremembe, zato je pridobivanje znanja in izkušenj na tem področju naš stalen proces. Našim sedanjim in bodočim uporabnikom želimo ponuditi vedno več in bolje od tistega, s čimer se vsakodnevno srečujemo.

XLAB
XLAB je mlado visokotehnološko podjetje, ustanovljeno leta 2001 kot spin-off Instituta Jožef Stefan. Glavna dejavnost podjetja so raziskave in razvoj programske opreme. XLAB smo mednarodno priznana raziskovalno-razvojna ekipa, vedno pripravljena sprejeti nove izzive.

Primarna področja delovanja so:

- razvoj aplikacij v omrežjih peer-to-peer,
- načrtovanje in razvoj storitveno usmerjenih sistemov,
- razvoj omrežnih protokolov za prenos podatkov v realnem času,
- 3D vizualizacija.

Raziskovalni projekti in aktivnosti:

- nacionalni raziskovalni projekti (omrežja Grid),
- izobraževanje mladih raziskovalcev v gospodarstvu.

Raziskovalni projekti 6. okvirnega programa Evropske unije:

- zanesljivi porazdeljeni sistemi (`http://www.dedisys.org`),
- aktivnosti za promocijo odprtokodnih rešitev (`http://www.tossad.org`),
- podpora široko porazdeljenim sistemom na nivoju operacijskega sistema (v zagonu),
- porazdeljena vizualizacijska knjižnica (v zagonu).

Izdelki:

- ISL Light (družina izdelkov ISL) ponuja širok nabor orodij za sodelovanje in tehnično podporo preko interneta. Stranka podjetja, ki uporablja rešitve ISL, lahko kadarkoli zaprosi svetovalca v podjetju za pomoč, ta pa problem rešuje s pomočjo video konference, tekstovnih sporočil ali celo z neposrednim dostopom na računalniško namizje stranke.
- MedicImaging zdravnikom olajša delo pri posameznih kirurških posegih, predvsem na področju nuklearne medicine. Z obdelavo medicinskih slik zdravnik pridobi podatke, ki omogočijo natančno preučevanje posameznih organov in tkiv, npr. pregled ledvic ali prostorskega modela srca.


RTK
2006
Design by: Anja Štifter