Programmeringsolympiadens final 2008

TÄVLINGSREGLER

- Tävlingen äger rum den 13 mars. Tävlingstiden är sex timmar effektiv tid.
- Tävlingen består av sju uppgifter som samtliga ska lösas genom datorprogram.
- Uppgifterna ska lösas i valfritt programmeringsspråk. Vi rekommenderar något av språken C, C++, Pascal eller Java. Du får byta språk mellan olika uppgifter.
- Tävlingsbidragen lämnas som exekverbara filer i Windows-format (EXE). Dessutom ska källkoden bifogas. För Linux/Mac-användare som använder gcc kan vi göra undantag från regeln med EXE-filer och istället kompilera källkodsfilen själva. Inkludera en kommentar i källkoden med den fullständiga kommandoraden för att kompilera programmet. Detsamma gäller om du använder ett interpreterande språk, t.ex. PHP (ange även version). Om du använder Java, så fungerar class-filen som exekverbar.
- I varje källkodsfil ska finnas en kommentar innehållande namn och skola.
- Lösningarna poängsätts med max 5 poäng per uppgift. Fem tester, med varierande krav hos ditt program, kommer att göras vid rättningen (undantag kan finnas). Möjlighet till delpoäng finns om programmet klarar endast en del av dessa tester. Ingen närmare bedömning av programkoden görs.
- Ingen test av indata behöver göras. Alla testdata följer de specifikationer som givits i uppgiften. Om det trots detta, vid rättningen, uppstår exekveringsfel vid körning av programmet bedöms programmet som felaktigt för det testexemplet.
- Samtliga uppgifter leder fram till program vars exekveringstid bör understiga 5 sekunder på en modern dator. Skulle exekveringstiden för ditt program överskrida denna tid bedöms programmet med 0 poäng för detta testexempel.
- Du har tillgång till de indatafiler som används i uppgiftens exempel.
- Deltagandet är individuellt vilket bland annat innebär att inget utbyte av idéer eller filer får ske under tävlingstiden. Självklart får din dator inte vara kopplad till vare sig internt eller externt nät.
- Hjälpmedel förutom dator med installerade programspråk: Programspråkets manualer, vanlig formelsamling och räknedosa.
- I flera av uppgifterna ska indata läsas från en vanlig ASCII-fil (se nästa sida).
- Tävlingsbidragen ska läggas i roten på utdelad diskett eller i en av läraren angiven hårddiskkatalog. Filerna ska döpas till uppg1...uppg7 med passande filtillägg. Ingen hänsyn tas till andra filer. Var noga med med att lämna in den korrekta versionen av ditt program.
- Tips: Det kan vara värt att göra egna indata för att testa ditt program. Även om programmet klarar testexemplet behöver det inte vara korrekt.

LATHUND FÖR INLÄSNING FRÅN FIL

Exempel på hur man i fem språk kan läsa in följande indata från filen `fil.txt`:

```
4 6
3.22 Text
```

Observera att fel kan förekomma.

C

```
#include <stdio.h>
...
int a1, a2;
char word[100];
double d;
FILE *fil=fopen("fil.txt", "rt");
fscanf(fil, "%d %d", &a1, &a2);
fscanf(fil, "%lf %s", &d, word);
```

C++

```
#include <iostream>
using namespace std;
...
int a1, a2;
char word[100];
double d;
ifstream fil("fil.txt");
fil >> a1 >> a2;
fil >> d >> word;
```

Java (J2SE)

```
import java.util.Scanner;
import java.io.File;
...
Scanner sc=null;
sc = new Scanner(new File("fil.txt"));
int a1=sc.nextInt(), a2=sc.nextInt();
double d=sc.nextDouble();
String word=sc.next();
```

Pascal

```
VAR
infile : Text;
a1, a2 : Integer;
d : Double;
word : string[100];
...
Assign(infile, 'fil.txt');
Reset(infile);
Readln(infile, a1, a2);
Readln(infile, d, word);
Close(infile);
```

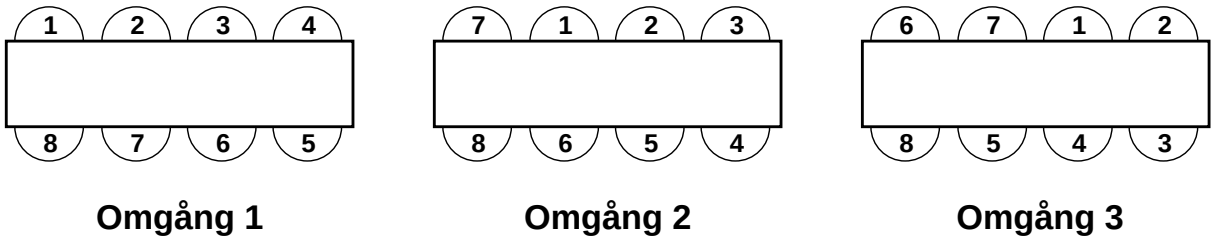
Basic

```
Dim s,word As String
Dim a1, a2 As Integer
Dim d As Double
Dim sar() As String
...
Open "fil.txt" For Input As #1
Line Input #1, s
sar = Split(s," ")
a1 = val(sar(0))
a2 = val(sar(1))
Line Input #1, s
sar = Split(s," ")
d = val(sar(0))
word = sar(1)
```

Lycka till!

UPPGIFT 1 – TURNERING

Om man vill ordna t.ex. en bordshockeyturnering där alla möter alla kan man använda sig av ett praktiskt rotationsschema som kallas *round robin*. Det går till så att spelarna i den första omgången möter varandra enligt figur 1 (vi antar att antalet spelare n är jämnt). När första omgången är klar förflyttar sig alla spelare ett steg medurs, utom spelaren i det nedre vänstra hörnet som hoppas över (därav namnet, man förflyttar sig "runt" Robin, d.v.s. den sista spelaren). Med detta rotationsschema är man garanterad att alla har mött alla precis en gång efter $n - 1$ omgångar.



FIGUR 1. *Spelarnas placering i de tre första omgångarna i fallet med 8 spelare.*

Din uppgift är att skriva ett program som skriver ut vilka spelare som ska möta vilka en viss omgång. Indata till programmet är antalet spelare i turneringen (ett jämnt tal n mellan 2 och 100) och omgången (mellan 1 och $n - 1$). Utdata är en lista över $n/2$ matcher där du anger vilka som möter vilka. Använd utdataformatet i exemplet nedan. Ordningen på matcherna spelar ingen roll.

Körningsexempel:

```
Antal spelare ? 8
Omgång ? 3

6-8
7-5
1-4
2-3
```

UPPGIFT 2 – KÖPA POTATIS

När man köper potatis är det ofta inte så viktigt hur stora potatisarna är, däremot är det smidigt om de är ungefär jämnstora så att man kan koka dem lika länge.

Edward har uppmärksammat det här problemet och funnit en lösning som kan verka lite opraktisk. När han ska köpa potatis väger han först varje potatis som finns tillgänglig i affären. Därefter väljer han ut potatisar så att

- de totalt väger minst V gram
- viktskillnaden mellan den tyngsta och lättaste potatisen är så liten som möjligt.

Skriv ett program som hjälper Edward att räkna ut den minsta möjliga viktskillnaden.

På första raden i filen `potatis.dat` står två heltal separerade med blanksteg: antal tillgängliga potatisar ($1 \leq N \leq 1000$) och den minsta acceptabla totalvikten ($1 \leq V \leq 10000$) i gram. Därefter följer N rader med ett heltal på varje rad, vikten för varje potatis (mellan 1 och 1000 gram). Summan av alla vikterna är minst V, så det finns alltid en lösning.

Programmet ska skriva ut den minsta möjliga skillnaden i vikt mellan den tyngsta och lättaste potatisen i ett optimalt urval av potatisar som totalt väger minst V gram.

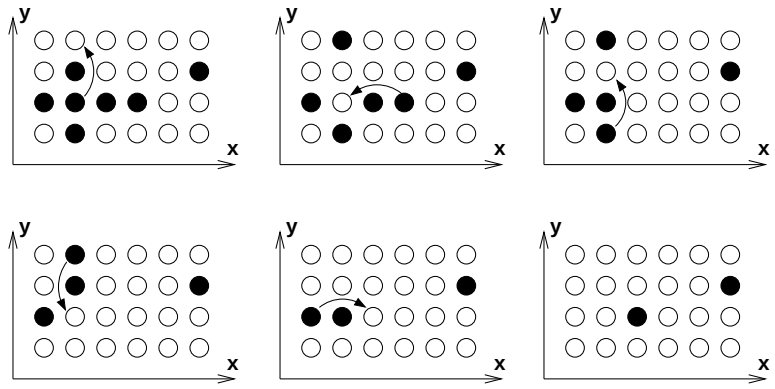
Exempel: Filen `potatis.dat` med innehållet

```
5 800
316
765
456
614
240
```

ska ge svaret

```
Minsta skillnad: 151
```

UPPGIFT 3 – SOLITÄR



FIGUR 2. *Dragserien som leder fram till två kvarvarande kulor från utgångsställningen i exemplet. Fylld ring symboliserar kula, ofylld ring grop utan kula.*

Solitär är ett franskt enpersonsspel som spelas på ett bräde med ett rutnät av gropar där det i varje grop kan ligga en kula. I varje drag väljer man en kula och flyttar två steg, antingen uppåt, neråt, åt höger eller åt vänster. För att draget ska vara godkänt krävs att gropen där kulan hamnar är tom och att det i den mellanliggande gropen finns en kula. Den kula som därmed blir "överhoppad" tas bort. Exempel på drag visas i figur 2.

Orginalversionen spelas på ett bräde med 37 hål och 36 kulor, och det går att utföra 35 drag så att det bara blir en kula kvar. I den här uppgiften är brädet ett obegränsat rutnät och kulorna kan vara utplacerade hur som helst. Skriv ett program som beräknar det minsta antalet kvarvarande kulor som kan uppnås från en given utgångsställning.

På första raden i filen `solitar.dat` står ett heltal N: antalet kulor (minst 2 och högst 10). Därefter följer N rader med två heltal på varje rad separerade med blanksteg, koordinater (x och y) för varje kula. Alla kulor har olika koordinater och från början ligger dessa i intervallet $0..9$ (men detta behöver inte gälla när kulorna flyttats). Programmet ska skriva ut det minsta antalet kulor som kan uppnås efter en serie godkända drag.

Exempel: Filen `solitar.dat` med innehållet

```
7
2 2
2 3
2 1
1 2
3 2
4 2
6 3
```

ska ge svaret

```
Minsta antal kulor: 2
```

UPPGIFT 4 – TERRÄNGLOPPET

Henning ska springa ett terränglopp där han vill minimera sin tid. Hans fysiska förmåga har (efter mycket datorspelande) diskretiserats så att han har fem "växlar" att välja mellan men ingen möjlighet att kontinuerligt välja ansträngningsnivå. Vidare är hans "energi" också diskretiserad till ett heltal mellan 0 (helt utpumpad) och K (full-laddad). På varje hundrametersträcka ändras energin med ett visst belopp D, som beror på vilken växel Henning springer på. För de fem växlarna är $D = -3, -2, -1, 0$ och $+1$. Med andra ord blir han tröttare på de tre första växlarna ($D < 0$), ändrar inte energi på den fjärde växeln, samt återhämtar sig något på den sista växeln ($D > 0$). Han kan naturligtvis inte välja en växel som ger negativ energi efter sträckan. Vidare kan han aldrig få högre energi än K; skulle han välja återhämtningsväxeln när han redan har energin K så förblir energin oförändrad.

Henning har detaljtränat på banan där loppet ska gå. Han har lyckats göra upp en tabell för hur snabbt han kan springa varje hundrametersträcka av loppet på var och en av sina växlar, med hänsyn tagen till lutning, underlag o.s.v. Skriv ett program som beräknar den minimala tiden Henning kan springa hela loppet på om han vid loppets början är full-laddad, d.v.s. har energin K.

På första raden i filen `loppet.dat` står två heltal: antalet hundrametersträckor ($1 \leq N \leq 1000$) samt maxenergin ($1 \leq K \leq 100$). Därefter följer N rader som beskriver hundrametersträckorna i ordning från start till mål. Varje rad innehåller fem heltal separerade med blanksteg, tiden i sekunder det tar att springa sträckan på var och en av växlarna. Det första talet gäller växeln med $D = -3$, det andra $D = -2$ o.s.v. Tiden för en växel med högre D är aldrig lägre än den för ett lägre D och alla tiderna ligger mellan 10 och 75 sekunder. Programmet ska skriva ut den minimala totaltiden (i sekunder) för hela loppet.

Exempel: Filen `loppet.dat` med innehållet

```
8 10
22 25 26 28 30
22 24 26 27 28
25 26 29 35 42
25 27 30 38 44
22 24 26 28 31
20 23 26 28 30
19 21 24 26 30
23 25 27 30 30
```

ska ge svaret

```
Minsta totaltid: 203
```

Kommentar: Den optimala taktiken har D-värdena -1, +1, -2, -2, 0, -3, -2 och -1.

UPPGIFT 5 – BUDBILEN

Veronica kör budbil mellan N städer, numrerade från 1 till N. Varje morgon får hon en lista över leveranser hon ska göra under dagen. Varje leverans innebär att ett paket ska hämtas i en stad och lämnas i en annan stad. Skriv ett program som, givet denna lista samt en avståndstabell för städerna, räknar ut den minimala sträckan Veronica behöver köra under dagen.

Den första raden i filen `budbil.dat` innehåller två heltal separerade med blanksteg: antalet städer ($2 \leq N \leq 8$) och antalet leveranser ($1 \leq M \leq 100$). Därefter följer N rader med N heltal på varje rad, separerade med blanksteg. Talen utgör en "avståndstabell" där det i:te talet på den j:te raden, $D(i, j)$, anger hur långt det är (i kilometer) mellan städerna i och j. Talen i tabellen uppfyller $D(i, i) = 0$, $D(i, j) = D(j, i)$ samt $D(i, j) + D(j, k) \geq D(i, k)$ (triangelolikheten) och alla avstånd mellan städer ligger i intervallet $1..100$. Slutligen följer M rader som vardera beskriver en leverans. Varje rad innehåller två heltal separerade med blanksteg. Det första talet anger staden där paketet ska hämtas och det andra staden där paketet ska lämnas. Veronica måste starta och sluta i stad 1 (även om hon inte ska hämta eller lämna paket där) och har obegränsat med plats i sin bil. Programmet ska skriva ut den minimala sträckan (i kilometer) som hon måste köra för att kunna leverera alla paketen.

Exempel

Filen `budbil.dat` med innehållet

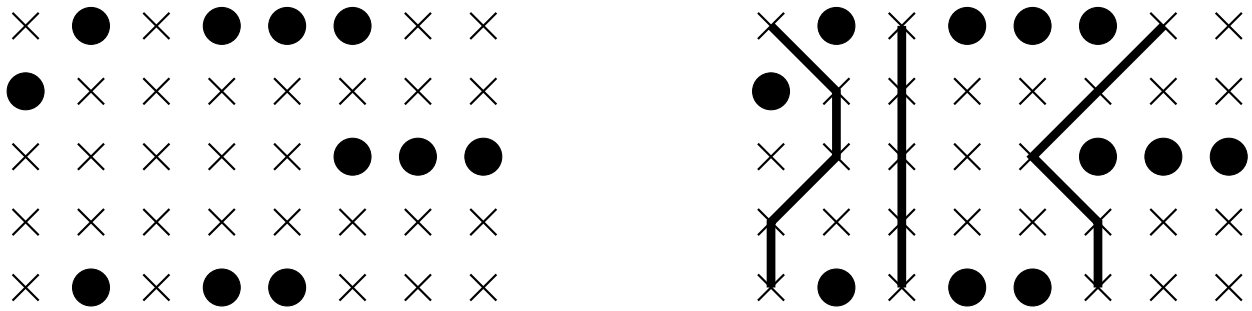
```
5 4
0 5 4 1 2
5 0 2 4 5
4 2 0 5 5
1 4 5 0 1
2 5 5 1 0
2 5
5 2
2 3
5 1
```

ska ge svaret

```
Minsta körsträcka: 16
```

Kommentar: Det bästa körschemat är $1 \to 5 \to 2 \to 3 \to 5 \to 1$

UPPGIFT 6 – KRETSKORTET



FIGUR 3. *Vänster: Skruvhålens placering i exemplet nedan. Höger: Ett möjligt sätt att dra tre ledningar (vilket är optimalt) från toppen till botten.*

Du har fått till uppgift att designa en del av ett kretskort. Du måste sammanbinda den övre delen av kortet med den undre delen genom att dra så många ledningar som möjligt. Det finns dock redan ett antal skruvhål på kortet som inte kan flyttas och måste undvikas. Skriv ett program som tar en beskrivning av skruvhålens position på kretskortet och beräknar det maximala antalet ledningar som kan dras. De lediga positionerna i den översta raden motsvarar möjliga startpunkter för ledningar, och de lediga positionerna i den nedersta raden motsvarar möjliga slutpunkter.

Ledningarna måste vara korta; med det menas att de varken kan gå i sidled eller bakåt på kortet. De börjar på den översta raden och förflyttar sig i varje steg antingen rakt ned, ned och ett steg åt vänster, eller ned och ett steg åt höger. Två dataledningar får inte korsa eller beröra varandra.

På första raden i filen `kretskort.dat` står två heltal: höjden ($2 \leq H \leq 100$) och bredden ($2 \leq B \leq 100$) på kretskortet, separerade med blanksteg. Därefter följer H rader som vardera innehåller en sträng med B tecken. Varje tecken beskriver en position på kretskortet och är antingen '.' (tom plats) eller 'o' (skruvhål). Programmet ska skriva ut det maximala antalet ledningar som kan dras från översta till nedersta raden så att skruvhålen undviks.

Exempel: Filen `kretskort.dat` med innehållet

```
5 8
.o.ooo..
o.......
.....ooo
........
.o.oo...
```

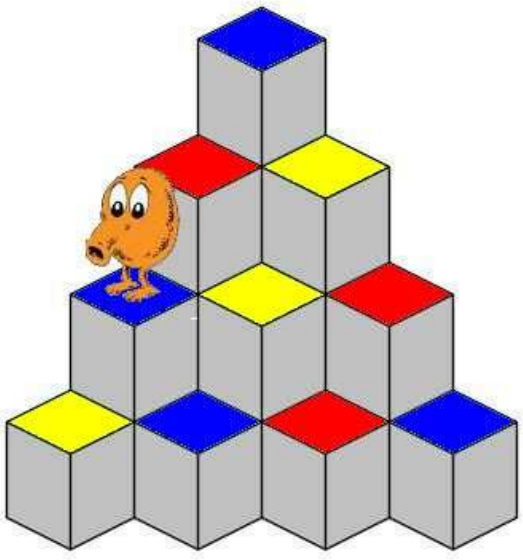
ska ge svaret

```
Antal ledningar: 3
```

UPPGIFT 7 – PYRAMIDEN

I ett klassiskt dataspel ska man styra en liten varelse som hoppar omkring på en pyramid (se figur nedan). Varelsen kan hoppa till "trappstegen" direkt ovanför eller under honom. Det är förstås inte tillåtet att hoppa utanför pyramiden.

När varelsen landar på ett trappsteg byter steget färg; från röd till gul, gul till blå eller blå till röd. Skriv ett program som läser in hur trappstegen på pyramiden är färgade vid starttillfället och skriver ut en hoppsekvens på max 5000 hopp som färgar om alla trappsteg till blåa. Du får starta på vilket trappsteg du vill i pyramiden (detta garanterar att en lösning alltid finns). Det första trappsteg som ändrar färg är det som nås efter första hoppet från starttrappsteget.



FIGUR 4. *Färgningen (i starttillståndet) av pyramiden i exemplet nedan. Varelsen står på trappsteg (3,1).*

Första raden i `pyramid.dat` består av ett heltal: pyramidens höjd ($2 \leq n \leq 40$). Därefter följer n rader som beskriver hur pyramiden är färgad vid starttillfället, uppifrån och ner. Bokstäverna 'R' (röd), 'G' (gul) och 'B' (blå) används för att beskriva trappstegen. Åtminstone ett trappsteg är inte blått från början.

Programmet ska skriva ut två rader. Den första raden ska innehålla två heltal: raden (där 1 är översta raden) och kolumnen (där 1 är kolumnen längst till vänster) som varelsen ska börja hoppa ifrån. Den andra raden ska bestå av en sträng (max 5000 tecken) som beskriver en hoppsekvens. Tecknen för att beskriva hoppen ska vara '7' (uppåt-vänster), '9' (uppåt-höger), '1' (nedåt-vänster) eller '3' (nedåt-höger).

Notera att ditt program inte behöver hitta den kortaste hoppsekvensen som färgar pyramiden blå; vilken sekvens som helst med högst 5000 hopp är godkänd.

Exempel: Filen `pyramid.dat` med innehållet

```
4
B
RG
BGR
GBRB
```

kan exempelvis ge svaret

```
3 1
193919193919373737717191991919373737
```