AZ ACM NEMZETKÖZI PROGRAMOZÓI VERSENYE

Kuki Attila, *kuki@math.klte.hu*
Kossuth Lajos Tudományegyetem, Információ Technológia Tanszék

Abstract

This paper is dedicated to the Scholastic Programming Contest of ACM. This is a contest for teams of three persons from a higher educational institute. After a short introduction we give a summary of the contest, the rules and the method of judgement. The main types of the given problems are prompted and the last part of this paper we give some sample promblems.

1. Egy kis történelem

1947-ben alapították az Association for Computing Machinery (ACM) társaságot, mely azóta a legnagyobbá és legpatinásabbá nõtte ki magát. Szervezetei jelen vannak mind az oktatás, mind a kutatás területén. Alapítóinak egyike John Mauchly, aki az ENIAC szülõatyja volt. Az ACM azóta is aktívan részt vesz a technikai fejlõdés elõsegítésében, tevékenysége végigkövethetõ a számítógépgenerációk fejlõdésén. Mûködése mégis a számítástudományban és információ technológiában való fajsúlyos részvétele alapján ismert igazán.

A 78 fõs csoportból napjainkra közel 100.000 fõt számláló társaságra duzzadt, több, mint 600 szervezete (chapter) mellett diákszervezettel is rendelkezik. 12 folyóiratot jelentet meg, s emellett évente mintegy 70 speciális irányultságú tudományos konferenciát rendez.

1995-ben rendezte meg 13-adik alkalommal a felsõfokú oktatási intézmények számára kiírt iskolai programozó bajnokságát. Magyarországról eddig elég szûk körbõl indultak versenyzõk ezen a rendezvényen, ezért reméljük, hogy ezen tájékoztató alapján megismerve a versenyt, egyre több résztvevõ lesz felsõoktatási intézményeinkbõl.

2. A verseny

Az Európában mutatkozó egyre bõvülõ érdeklõdés hatására a korábbi egy selejtezõ helyszín évrõl évre szaporodik, megkönnyítve ezzel a csapatok kijutását a rendezvény helyszínére.

1995-ben Közép-Európában Pozsony adott otthont a versenynek, s utolsó információink szerint 1996-ban is ott lesz. Az idõpontja általában egy október végi, november elejei dátum szokott lenni, maga a rendezvény két napig tart.

Az elsõ nap programja között van a rendezõ intézmény bemutatkozása, a versenyzõk regisztrálása, pontos tudnivalók ismertetése.

A második nap kerül lebonyolításra maga a verseny, s utána rögtön az eredményhirdetés.

3. Lebonyolítási rend

A versenyre 3 fõs csapatokkal lehet nevezni, az utóbbi évtõl intézményenként már több csapatot is fogadnak. Minden csapatnak egy IBM kompatibilis személyi számítógép áll a rendelkezésére, s a feladatokat vagy Pascal vagy C nyelven kell megoldani. A verseny idõtartama általában 6 óra.

4. Értékelés

A legtöbb esetben 8 problémát kell a versenyzõknek a rendelkezésre álló idõ alatt megoldani. A problémák mind olyanok, hogy a problémára adott programnak adott input adatokra jól meghatározott output adatokat kell szolgáltatni. A megoldás értékelése csak ezen alapul, helyességének nincsenek fokozatai, mint pl. 'az elgondolás jó volt, csak a megvalósításba csúszott kis hiba'. Ha a program a zsûri input adataira a megfelelõ tartalmú és formátumú outputot szolgáltatja, akkor a megoldás helyes, minden egyéb esetben a megoldás hibás. Az eredmény ellenõrzése az output file-ok karakterenkénti összehasonlítását is jelentheti, ezért nagyon lényeges, hogy a helyes eredményt pontosan a leírt formában állítsák elõ a csapatok.

Ha a megoldás hibás, akkor a csapat egy néhány szavas hibaüzenetet kap, melyek a következõk valamelyike lehet:

- szintaktikai hiba - meglepõ de ilyen is elõfordult már,
- futási hiba,
- hibás output - a program rossz eredményt szolgáltat,
- hibás output formátum - a jó eredmény nem a megfelelõ formában van,
- idõtúllépés - a futás túllépte a feladathoz rendelt idõkorlátot; ezt a korlátot a versenyzõk nem ismerik.

Ez utóbbi hiba kényszeríti arra a versenyzõket, hogy a problémára minél hatékonyabb algoritmust keressenek.

Az értékelés ezek után egyszerû. Az a csapat nyeri a versenyt, amelyik a legtöbb helyes megoldással rendelkezik a verseny zárásakor. Holtverseny esetén a megoldásokra fordított idõ alapján döntenek. Ez az idõ úgy alakul ki, hogy összeadják a helyes problémák beadási idõpontjait. Tehát ha egy csapat egy helyes megoldást a verseny indítása után 2 óra 20 prckor ad be, egy másik helyes megoldást 3 óra 30 perckor, akkor az idejük 5 óra 50 perc. Ha egy helyes megoldásnak korábban voltak sikertelen próbálkozásai, akkor annyiszor 20 perc büntetõidõt adnak a csapat idejéhez, amennyi a sikertelen próbálkozások száma volt.

Ezzel a rendszerrel már a holtverseny esélye minimális, s az eredmény a verseny zárásakor rögtön rendelkezésünkre áll.

5. Feladatok

Általános leírást nehéz adni a versenyeken szereplõ problémákról. Nagyon sokrétûek , de a többségük megoldásának alapját szolgáló algoritmusokat be lehet sorolni néhány jól meghatározott osztályba. Ezek közül a legjellemzõbbek:

- **Kombinatorikai algoritmusok**
- Ismétlés nélküli permutációk,
- Ismétléses permutációk,

– Kombinációk,

– Egy halmaz részhalmazai, particionálás.

– **Geometriai algoritmusok**

– Szakaszok metszete,

– Poligon és pont,

– Ponthalmaz konvex burka.

– **Gráfalgoritmusok**

– Mélységi keresés,

– Optimális keresés.

A megoldások során az említett hatékonyság mellett a problémák speciális eseteire kell ügyelni, hiszen a zsûri a tesztadatokban elõszeretettel szerepeltet ‘necces’ eseteket, melyeken egy egyszerû, kevés esetet vizsgáló algoritmus esetleg megbukhat.

6. Motiváció

A résztvevõk díjazása esetenként változik. Kisebb-nagyobb ajándékok mellett a helyezettek általában tárgyjutalomban részesülnek. A legcsábítóbb lehetõség az, hogy a gyõztes részt vehet a verseny döntõjén, melyet minden esetben az Egyesült Államokban rendeznek. Az ACM jelentõs utazási hozzájárulást biztosít a döntõn résztvevõ csapatoknak, de itt az ACM tagság már feltétel.

7. Mintafeladatok

A verseny nyelve minden selejtezõn az angol, ezért a hazai felkészítõ versenyeken is érdemes a problémákat angolul kitûzni. Mellékletként közlünk néhány, korábbi európai döntõkön szereplõ problémát:

Urban Elevations

An elevation of a collection of buildings is an orthogonal projection of the buildings onto a vertical plane. An external elevation of a city wouldshow the skyline and the faces of the "visible" buildings of the city as viewedfrom outside the city from a certain direction. A southern elevation shows no sides; it shows the perfectly rectangular faces of buildings or parts of faces of buildings not obstructed on the south by taller buildings. For this problem, you must write a program that determines whicb buildings of a city are visiblein a southern elevation. For simplicity, assume all the buildings for the elevation are perfectrectangular solids, each with two sides that run directly east-west and two running diretly north-south. Your program will find tbe buildings that appear in a southern elevation based on knowing the positions and heights of each city building. That data can be illustrated by a map of the city as in tbe diagram on the left below. The southern elevation for the city is illustrated in the diagram on the right.

Input

Input for your program consists of tbe numeric description of maps of several cities. The first line of each map contains the number of buildings in the city (a non-negative integer less than 101). Each subsequent line of a map contains data for a single building - 5 real numbers separated by spaces in the following order:

- x-coordinate of tbe southwest corner
- y-coordinate of the southwest corner
- width of tbe building (length of the south side)
- depth of the building (length of the west side)
- height of tbe building

Each map is oriented on a rectangular coordinate system so that the positive x-axis points east and the positive y-axis points north. Assume that all input for each map corresponds to a legitimate map (the number of buildings is tbe same as tbe number of subsequent lines of input for the map; no two buildings in a single map overlap). Input is terminated by the number 0 representing a map witb no buildings.

Output

Buildings are numbered according to where their data lines appear in the map's input data - building #1 corresponding to the first line of building data, building #2 data to tbe next line and building #n to the *n*th line of building data for that map. (Buildings on subsequent maps also begin their numbering with 1.)

For each map, output begins with line identifying the map (map #l, map #2, etc.). On the next line the numbers of the visible buildings as they appear in the southern elevation, ordered south-to-north, west-to-east. (Separated by single spaces.) This means that if building *n* and building *m* are visible buildings and if the southwest corner of building *n* is west of the southwest corner of building *m*, then number *n* is printed before number *m*. If building *n* and building *m* have the same x-coordinate for their southwest corners and if building *n* is south of building *m*, then the number *n* is printed before the number *m*. For this program, a building is considered visible whenever tbe part of its southern face that appears in the elevation has strictly positive area. One blank line must separate output from consecutive input records.

Sample Input

```
14
160 0 30 60 30
125 0 32 28 60
95 0 27 28 40
70 35 19 55 90
0 0 60 35 80
0 40 29 20 60
35 40 25 45 80
0 67 25 20 50
0 92 90 20 80
95 38 55 12 50
95 60 60 13 30
95 80 45 25 50
165 65 15 15 25
165 85 10 15 35
0
```

Sample Output

```
map #1
5 9 4 3 10 2 1 14
```

Moth Eradication

Entomologists in the Northeast have set out traps to determine the influx of Jolliet moths into the area. They plan to study eradication programs that have some potential to control the spread of the moth population.

The study calls for organizing the maps in which moths have been caught into compact regions, which will then be used to test each eradication program. A region is defined as the polygon with tbe minimum length perimeter that can enclose all traps within that region. For example, the traps (represented by dots) of a particular region and its associated polygon are illustrated below.

You must write a program that can take as input the locations of traps in a region and output the loations of traps that lie on the perimeter of the region as well as the length of the perimeter.

Input

The input file will contain records of data for several regions. The first line of each record contains the number (an integer) of traps for that region. Subsequent lines of the records contain 2 real numbers that are the x- and y- coordinates of the trap locations. Data within a single record will not be duplicated. End of input is indicated by a region with 0 traps.

Output

Output for a single region is displayed on at least 3 lines:

First line: The number of the region. (The first record corresponds to region #1, the second to region #2, etc.)

Next line(s): A listing of all the points that appear on the perimeter of the region. The points must be identified in the standard form "(x-coordinate,y-coordinate)" rounded to a single decimal place. The sarting point for this listing is irrelevant, but the listing must be oriented *clockwvise* and begin and end with the same point.

Last line: The length of the perimeter of the region rounded to 2 decmal places. One blank line must separate output from consecutive input records. A sample input file with records for 3 regions followed by correct output for the sample input:

Sample Input

```
3
1 2
4 10
5 12.3
6
0 0
1 1
3.1 1.3
3 4.5
6 2.1
2 -3.2
7
1 0.5
5 0
4 1.5
```

3 -0.2
2.5 -1.5
0 0
2 2
0

Sample Output

Region #l:
(1.0,2.0)-(4.0,10.0)-(5.0,12.3)-(1.0,2.0)
Perimeter length = 22.10
Region #2:
(0.0,0.0)-(3.0,4.5)-(6.0,2.1)-(2.0,-3.2)-(O.O,O.0)
Perimeter length = 19.66
Region #3:
(0.0,0.0)-(2.0,2.0)-(4.0,1.5)-(5.0,0.0)-(2.5,-1.5)-(0.0,0.0)
Perimeter length = 12.52