Univerza v Ljubljani

Prvenstvo
UNIVERZE V LJUBLJANI
v
programiranju

2002

4. krog

Pokrovitelji:

UNIVERZA V LJUBLJANI
Fakulteta za matematiko in fiziko

Prvenstvo v programiranju 2002

4. krog 14. oktober 2002

NAVODILA

Prijava: Prijavite se lahko le na en računalnik. Če bodo med tekmovanjem z njim slučajno težave, javite vodstvu tekmovanja. V primeru, da boste hkrati prijavljeni na dveh računalnikih, bo ekipa diskvalificirana.

Uporabniško ime je sestavljeno iz besede acm in številke računalnika.

Primer: uporabniško ime na računalniku št. 1 je: acm01

uporabniško ime na računalniku št. 12 je: acm12

Geslo je sporočeno pred začetkom tekmovanja.

Vaše domače področje na disku je `/home/acmXX` kjer je XX številka računalnika

Priprava delovnega okolja

Po uspešni prijavi morate odpreti dva terminala in program Netscape.

Terminal št.1: v njem boste pisali programe, prevajali programe in pošiljali programe sodniku.

Terminal št.2: v njem boste dobivali obvestila o pravilnosti rešitev in pošiljali morebitna vprašanja sodniku.

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Pisanje programov:

Na voljo so urejevalniki besedil: *vi, joe, jed, mcedit, emacs*, *gedit in kedit*

Ime programov je točno določeno pri opisu nalog. Pazite na `return 0,` ki mora zaključiti vsak uspešno izveden C program.

Prevajanje programov:

Za prevajanje programov morate uporabljati naslednje prevajalnike:

```
C       (*.c)         gcc    primer:   gcc -o program program.c
C++     (*.cpp)       g++    primer:   g++ -o program program.cpp
Pascal (*.pas)        gpc    primer:   gpc -o program program.pas
```

Pošiljanje programov:

Za pošiljanje programov sodniku uporabljate ukaz `submit`.

Primer: `submit program.c`

Sporočila o napakah – PE in WA

Zaradi različnih razlogov je zelo težko razlikovati med `Presentation Error` in `Wrong Answer`. Ker preverjanje poteka več ali manj avtomatsko, program pogosto ne zna razlikovati med tem, ali ste le pozabili piko v odgovoru, ali pa ste napačno zračunali. Zato tudi `Wrong Answer` včasih lahko pomeni

le manjkajoč znak v odgovoru. Praviloma naj bi sporočilo `PE` dobili le v primeru napačno postavljenih presledkov ali ločil in praznih vrstic, ni pa to nujno!

Komunikacija sodniki – ekipa:

V terminalu 2 izvedete ukaz: `telnet svarun`

Za prijavo na računalnik svarun uporabite isto uporabniško ime in geslo, kot ste ga uporabili za prijavo na vaš računalnik. V tem terminalskem oknu se bodo izpisovala sodnikova sporočila vam. Če želite poslati sporočilo sodniku, morate v tem terminalu napisati ukaz:

```
write sodnik
```

in potem sporočilo, ki ga zaključite s kombinacijo tipk CTRL + D na začetku prazne vrstice. Prosimo, da **ne** pošiljate sporočil tipa `... Sedaj smo poslali program ...` in `... Naš program zagotovo dela, pa dobimo WA ...` Ekipa, ki bo pošiljala tovrstna sporočila, je lahko tudi diskvalificirana.

Trenutni rezultati:

Program Netscape: bo omogočal pregled trenutne uspešnosti ekip

Po zagonu programa greste na naslov:

http://svarun.fmf.uni-lj.si/index.html

Kadar želite videti trenutno stanje, morate narediti »reload«.

Pritožbe:

So možne le takoj po zaključku tekmovanja. Kasneje se rezultati lahko spremenijo le v primeru drastične sodniške napake.

Preklapljanje med angleško in slovensko tipkovnico:

(Velja samo za okolje X windows)

Prednastavljena je slovenska tipkovnica. Za preklop na angleško tipkovnico v terminalu izvedete ukaz

```
setxkbmap us
```

za preklop nazaj na slovensko pa:

```
setxkbmap sl
```

Izbrana tipkovnica potem velja za celotno X okolje.

Stockbroker Grapevine

Program BROKER.C, BROKER.CPP, BROKER.PAS

Stockbrokers are known to overreact to rumours. You have been contracted to develop a method of spreading disinformation amongst the stockbrokers to give your employer the tactical edge in the stock market. For maximum effect, you have to spread the rumours in the fastest possible way.

Unfortunately for you, stockbrokers only trust information coming from their 'trusted sources'. This means you have to take into account the structure of their contacts when starting a rumour. It takes a certain amount of time for a specific stockbroker to pass the rumour on to each of his colleagues. Your task will be to write a program that tells you which stockbroker to choose as your starting point for the rumour, as well as the time it will take for the rumour to spread throughout the stockbroker community. This duration is measured as the time needed for the last person to receive the information.

Input

Your program will input data for different sets of stockbrokers. Each set starts with a line with the number of stockbrokers. Following this is a line for each stockbroker which contains the number of people who they have contact with, who these people are, and the time taken for them to pass the message to each person. The format of each stockbroker line is as follows: The line starts with the number of contacts (n), followed by n pairs of integers, one pair for each contact. Each pair lists first a number referring to the contact (e.g. a `1' means person number one in the set), followed by the time in minutes taken to pass a message to that person. There are no special punctuation symbols or spacing rules. For each set of data, your program must output a single line containing the person who results in the fastest message transmission and how long before the last person will receive any given message after you give it to this person, measured in integer minutes.

Each person is numbered 1 through to the number of stockbrokers. The time taken to pass the message on will be between 1 and 10 minutes (inclusive), and the number of contacts will range between 0 and one less than the number of stockbrokers. The number of stockbrokers will range from 1 to 100. The input file is terminated by a set of stockbrokers containing 0 (zero) people.

It is possible that your program will receive a network of connections that excludes some persons, i.e. some people may be unreachable. If your program detects such a broken network, simply output the message "disjoint". Note that the time taken to pass the message from person A to person B is not necessarily the same as the time taken to pass it from B to A, if such transmission is possible at all.

SAMPLE INPUT

```
3
2 2 4 3 5
2 1 2 3 6
2 1 2 2 2
5
3 4 4 2 8 5 3
1 5 8
4 1 6 4 10 2 7 5 2
0
2 2 5 1 5
0
```

SAMPLE OUTPUT

```
3 2
3 10
```

Parallel Deadlock

Program PARALLEL.C, PARALLEL.CPP, PARALLEL.PAS

A common problem in parallel computing is establishing proper communication patterns so that processors do not deadlock while either waiting to receive messages from other processors, or waiting for the sending of messages to other processors to complete. That is, one processor will not complete sending a message until it is received by the destination processor. Likewise, a receive cannot complete until a message is actually sent.

There are two modes of communication: blocking and non-blocking. A blocking send will not complete until a matching receive is performed at the destination processor. Likewise, a blocking receive will not complete until the matching send is performed by the source processor. Non- blocking actions will "return" immediately (i.e., allow the program to continue), but will not actually complete until the matching action is performed at the target. The matching action of a send is a receive (either blocking or non-blocking), and similarly, the matching action of a receive is a send (either blocking or non-blocking).

At the start of each timestep, each processor that is not blocked starts to run its next instruction. Processors that execute blocking instructions become blocked. Messages can be received at the end of the timestep in which they are sent, but may need to wait several timesteps until the recipient performs a matching receive. If the recipient of a message is waiting to receive from the sender, then the message is received in the same timestep. Messages are received in the order that they are sent. If all of the actions for a particular blocking instruction complete at the end of the timestep, then the processor that ran the instruction will be unblocked before the next timestep.

A correct program will terminate only when all of its actions have completed. Pending non- blocking operations must be completed before a program can terminate. Your program will take in a list of processors and actions (no more than 100 for each processor), and determine if each processor finishes its program. If a given processor does not finish, it must print out which other processors are preventing it from finishing.

Input

There will be several cases. The first line of each case will be a single positive integer that tells how many processors will be listed. For each processor there will be one line containing the name of the processor (a single capital letter) followed by a positive integer, N. The following N lines will contain the instructions that comprise the program for that processor. The input ends with number 0.

An instruction is of the form "Mode Action Target(s)" where "Mode" can be "B" or "N", for blocking or non-blocking, respectively. "Action" can be "S" or "R", for send or receive, respectively. "Target(s)" will be one or more processor names to which the action is to be addressed. No processor will be listed twice and a processor will never attempt any sort of communication with itself. A send to multiple targets will not complete until matching receives have been performed by all of the targets (and vice versa).

Output

Given that instruction 1 occurs at t = 1, your program will output at which timestep each processor finishes. If a processor does not finish, you must output which processor are preventing that processor from finishing. Processors should be listed in alphabetical order, both for the list of processors and the sets of processors that prevent a processor from finishing. The list of processors preventing termination should list processors at most once and

separate multiple processors with "and". Output for each case should begin with "`Case #n:`". Leave an empty line after each case.

Sample Input

```
4
I 5
B S B P C
N S B P C
N R B
B R P
B R C
B 2
B R I
B S I
P 3
N S I
N R I
B R I
C 4
N S I
B R I
B S P
B R I
0
```

Sample Output

```
Case #1:
B finishes at t=4
C never finishes because of P
I never finishes because of B and C
P finishes at t=5
```

Notice how C's final blocking receive would be matched by a send on I if both instructions were executed. However, it never gets executed because it is stuck in the blocking send to P (that has no matching receive on P), therefore causing deadlock on I.

Molecules

Program MOLECULE.C, MOLECULE.CPP, MOLECULE.PAS

In this abstraction from a molecular engineering problem associated with developing a synthetic fuel, we are given four, equal length, molecular chains that are to form a super molecule. In the simplified two-dimensional model used here, the super molecule is formed as an interlocking rectangular arrangement of the four given molecular chain strands. The interlocking feature is the sharing of a common molecule between pairs of chains.

To illustrate, suppose we have the four, length-twelve, molecular chains:

```
OIMDIHEIAFNL
CHJDBJMHPJKD
LCBJOJGIEKBO
KAINLHLOLBEJ
```

These can be placed in the interlocking arrangements:

```
    O       L                               O   C
    I       C                               I   H
    M       B                               M   J
CHJDBJMHPJKD                                D   D
    I       O                       LCBJOJGIEKBO
    H       J           or                  H   J
    E       G                               E   M
    I       I                       KAINLHLOLBEJ
    A       E                               A   P
    F       K                               F   J
KAINLHLOLBEJ                                N   K
    L       O                               L   D
```

In this problem, we have some constraints on the arrangements being sought:

1) Any of the four chains can be placed in any of the super molecule's four, general, horizontal or vertical slots, as in the illustrations above.

2) If a chain is placed in one of the two horizontal slots, it must keep the same left-to-right orientation it had in the original chain listing. That is, it can't be flipped end-for-end.

3) If a chain is placed in one of the two vertical slots, its left-to-right orientation in the original chain listing must match its top-to-bottom orientation in the slot. It can't be flipped end-for-end from this orientation.

4) The enclosed rectangular region at the center of the super molecule must have as large an area as possible, and the area cannot be zero. (The large area constraint arises from a fuel-volatility criterion for the arrangement. The non-zero area constraint arises because neither the vertically nor the horizontally oriented chains can lie immediately next to each other without producing side effects we're not considering.)

The area is measured as the count of vacant character positions within the enclosed rectangle of the super molecule. The area counts of the two super molecules illustrated above are thirty (30) and four (4).

5) The fore and aft tails of each chain extending beyond the super molecule's central interlocked rectangle must have a minimum length of one chain element. That is, none of the four original chains can have either its first or its last element as part of the interlocking-rectangle boundary.

Input

The input consists of a series of data sets. Each data set consists of four molecular chains of 12 fixed elements each. These 12 elements are given as contiguous capital letters. The molecule designators within the chains will be restricted to the sixteen letters, A...P. The first letter of a chain will appear as the first character on an input line.

The first molecule designator within the first chain of a data set will be the letter "Q" to indicate the end of data.

Output

A line with a single integer is to be emitted for each input data set encountered. This integer is the maximum area enclosed by any legitimate arrangement of the four chains. Use the output value zero (0) to indicate that no legitimate super molecule could be formed for a given data set. The first digit of an output value should be the first character on a line.

Sample Input

```
CDBADCBBEFEF
DACCBADAFEAB
EFBDCAADBDCD
ABCDABCDABCD
DACCBADAFEAB
EFBDCAADBDCD
ABCDABCDABCD
CDBADCBBEFEF
ABABABABABAB
CDCDCDCDCDCD
EEEEEEEEEEEE
FFFFFFFFFFFF
ABAAAAAAAABA
CBCCCCCCCCBC
DBDDDDDDDDBD
EBEEEEEEEEBE
ABBBBBBBBBBA
ACCCCCCCCCCA
ADDDDDDDDDA
AEEEEEEEEEA
BBBABBBABBBB
CCACCCACCCCC
DDDDADDADDDD
EEAEEEAEEEEE
Q
```

Sample Output

```
48
48
0
64
0
6
```

Coconuts revisited

Program COCONUTS.C, COCONUTS.CPP, COCONUTS.PAS

The short story titled *Coconuts*, by Ben Ames Williams, appeared in the *Saturday Evening Post* on October 9, 1926. The story tells about five men and a monkey who were shipwrecked on an island. They spent the first night gathering coconuts. During the night, one man woke up and decided to take his share of the coconuts. He divided them into five piles. One coconut was left over so he gave it to the monkey, then hid his share and went back to sheep.

Soon a second man woke up and did the same thing. After dividing the coconuts into five piles, one coconut was left over which he gave to the monkey. He then hid his share and went back to bed. The third, fourth, and fifth man followed exactly the same procedure. The next morning, after they all woke up, they divided the remaining coconuts into five equal shares. This time no coconuts were left over.

An obvious question is “how many coconuts did they originally gather?” There are an infinite number of answers, but the lowest of these is 3121. But that's not our problem here.

Suppose we turn the problem around. If we know the number of coconuts that were gathered, what is the maximum number of persons (and one monkey) that could have been shipwrecked if the same procedure could occur?

Input

The input will consist of a sequence of integers, each representing the number of coconuts gathered by a group of persons (and a monkey) that were shipwrecked. The sequence will be followed by a negative number.

Output

For each number of coconuts, determine the largest number of persons who could have participated in the procedure described above. Display the results similar to the manner shown below, in the Sample Output. There may be no solution for some of the input cases; if so, state that observation.

Sample Input

```
25
30
3121
-1
```

Sample Output

```
25 coconuts, 3 persons and 1 monkey
30 coconuts, no solution
3121 coconuts, 5 persons and 1 monkey
```

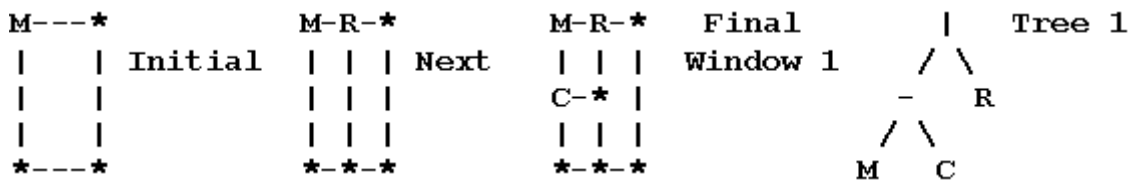
Split Windows

Program WINDOWS.C, WINDOWS.CPP, WINDOWS.PAS

The Dotty Software Company makes software that is displayed on inexpensive text based terminals. One application for this system has a main window that can be subdivided into further subwindows. Your task is to take a description of the screen layout after a sequence of window splits and draw the minimum sized window grid that is consistent with the description.

In this problem we will concentrate on the boundaries of windows, so all the characters inside of windows will be left blank. Each window that is not further subdivided has a label. Each label is a distinct uppercase letter. For a text terminal the boundaries of windows must be drawn with characters, chosen as follows: A capital letter label is placed in the upper left-hand corner of each undivided window. Asterisks,'*', appear in corners of windows where there is not a label. Dashes, '−', appear on upper and lower boundaries where there are not corners. Vertical bars, '|', appear on side boundaries where there are not corners.

For example, the sequence of splits below would generate Window 1: Initially there could be an application window labeled `M`, that is split next into left and right subwindows, adding label `R`, and the left subwindow is split into top and bottom subwindows, adding the label `C`.

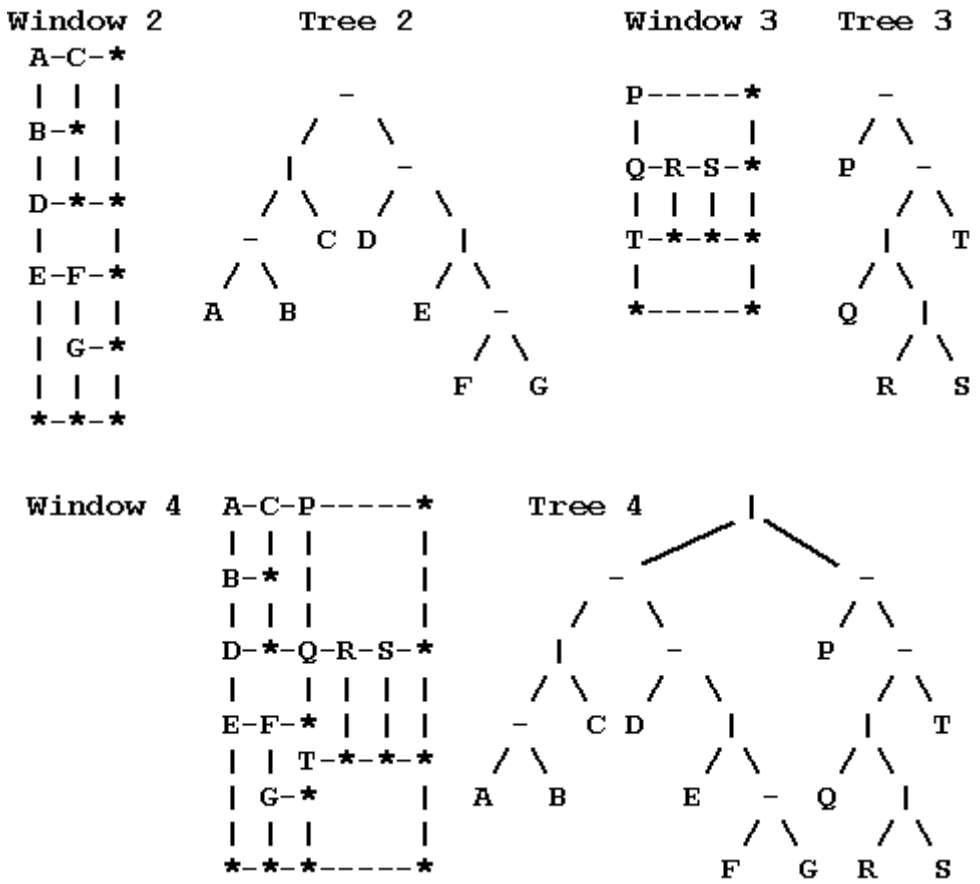


For each pattern of splits there is a binary tree of characters that can describe it. The window splitting and tree structures are described together, building up from the simplest cases.

1) A window may be an undivided rectangle. Such a window has a capital letter as label. The tree for the window contains just the label.

2) A window may either be split into left and right subwindows or into top and bottom subwindows, and the corresponding trees have as root the boundary character for the split: a vertical line '|' or a horizontal dash '−' respectively. The root has left and right subtrees corresponding to the top and bottom or left and right subwindows respectively.

Tree 1, above, and Trees 2-4, below, would be consistent with Windows 1-4. Note that Tree 4 contains Trees 2 and 3.



The trees may be more succinctly expressed via a preorder traversal:

1) The preorder traversal of a tree with just one node (containing a letter) is that letter.

2) The preorder traversal of a tree with a left and a right subtree is the character from the root of the tree ('-' or '|') followed by the preorder traversal of the left subtree, and then the preorder traversal of the right subtree.

The preorder traversals for Trees 1 through 4 are

```
|-MCR     -|-ABC-D|E-FG    -P-|Q|RST     |-|-ABC-D|E-FG-P-|Q|RST
```

Each undivided window must have space for at least one character inside. Hence each tree of splits will be associated with a minimum window size. Windows 1-4 are minimum sized windows for Trees 1-4. Each window illustrates the fact that even in a minimum sized window, not all undivided windows contain only one character.

Consider Tree 4 and Window 4. The main window is split into a left window with Tree 2 and right window with Tree 3. The left window is like Window 2, but the right window is not just like Window 3. The heights of left and right subwindows must match, so the right window must be stretched.

The stretching rule depends on a definition of the size of windows. For dimension calculations it is easiest to imagine that a window contains its interior and a half character wide boundary on all sides, so the total dimensions of a window are one more than the dimensions of the interior. Hence the minimum dimensions of a window are 2 by 2, since a window must contain one character inside, and we add one for the boundary. This definition also

means that the sum of the widths of left and right subwindows is the width of their enclosing window. The sum of the heights of top and bottom subwindows is the height of their enclosing window.

The right window in Window 4 must be stretched to match the height 10 of the left window. The right window is split into a top with tree `P` having minimum height 2 and a bottom with tree `-|Q|RST` having minimum height 4. The rule for the dimensions in the stretched window is that the heights of the subwindows expand in proportion to their minimum heights, if possible. Some symbols may help here: Let $D = 10$ be the height of the combined stretched window. We want to determine D_1 and D_2, the stretched heights of the top and bottom subwindow. Call the corresponding minimum dimensions $d = 6$, $d_1 = 2$, and $d_2 = 4$. If the window were expanded from a total height d to D in proportion, we would have $D_1 = d_1*(D/d) = 2*(10/6) = 3.333...$and $D_2 = d_2*(D/d) = 6.666....$ Since the results are not integers we increase D_1 to 4 and decrease D_2 to 6.

There is a similar calculation for the bottom window with tree `-|Q|RST`. It is further subdivided into a top with tree `|Q|RS` and a bottom with tree `T`, each having minimum height $2 = d_1 = d_2$. The heights need to add up to $D = 6$, so they are increased proportionally to $D_1 = D_2 = 2*(6/4) = 3$ (exact integers).

The final dimensions of an enclosing window are always determined before the final dimensions of its subwindows. In this example only heights needed to be apportioned. If all horizontal and vertical splits were interchanged in this example, producing a tree `-|-|ABC|D-E|FG|P|-Q-RST`, then widths would be apportioned correspondingly, as shown in the third part of the sample output below. If the proportion calculations do not work out to integers, it is always the top or left subwindow whose dimension is increased to the next integer.

The first line of input contains one integer, which is the total number of preorder traversals describing window structures. This line is followed by one line for each preorder traversal. Each preorder traversal will contain appropriate dividers '|' and '-' and from 1 to 26 uppercase letters.

For each preorder traversal, print the number of the preorder traversal on one line followed by the minimum sized window grid that the traversal could represent. The total number of rows or columns in output grids will be no more than 53.

Sample input:

```
3
|-MCR
|-|-ABC-D|E-FG-P-|Q|RST
-|-|ABC|D-E|FG|P|-Q-RST
```

Sample output:

```
1
M-R-*
| | |
C-* |
| | |
*-*-*
2
A-C-P-----*
| | |     |
B-* |     |
| | |     |
D-*-Q-R-S-*
|   | | | |
E-F-* | | |
| | T-*-*-*
| G-*     |
| | |     |
*-*-*-----*
3
A-B-D-E---*
| | | |   |
C-*-* F-G-*
|   | | | |
P---Q-*T*-*
|   |   |   |
|   R--*   |
|   |   |   |
|   S--*   |
|   |   |   |
*---*--*--*
```