# Odprto ekipno prvenstvo UNIVERZE V LJUBLJANI v programiranju

2003

finalno tekmovanje

# Glavni pokrovitelji








UNIVERZA V LJUBLJANI

Fakulteta za matematiko in fiziko

# Pokrovitelji














Šolski servis
Janez Vrankar



# 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, kwrite, 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)      fpc    primer:   fpc program.pas
Java    (*.java)     gcj    primer:   gcj -o program --main=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: `ssh svarun.fmf.uni-lj.si`

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:**

Z ukazom v terminalu:

setxkbmap si

setxkbmap us

Nastavitve veljajo za novo odprte programe.

# ID Codes

Program codes.c, codes.cpp, codes.java, codes.pas

It is 2084 and the year of Big Brother has finally arrived, albeit a century late. In order to exercise greater control over its citizens and thereby to counter a chronic breakdown in law and order, the Government decides on a radical measure -- all citizens are to have a tiny microcomputer surgically implanted in their left wrists. This computer will contains all sorts of personal information as well as a transmitter which will allow people's movements to be logged and monitored by a central computer. (A desirable side effect of this process is that it will shorten the dole queue for plastic surgeons.)

An essential component of each computer will be a unique identification code, consisting of up to 50 characters drawn from the 26 lower case letters. The set of characters for any given code is chosen somewhat haphazardly. The complicated way in which the code is imprinted into the chip makes it much easier for the manufacturer to produce codes which are rearrangements of other codes than to produce new codes with a different selection of letters. Thus, once a set of letters has been chosen all possible codes derivable from it are used before changing the set.

For example, suppose it is decided that a code will contain exactly 3 occurrences of '`a`', 2 of '`b`' and 1 of '`c`', then three of the allowable 60 codes under these conditions are:

```
abaabc
abaacb
ababac
```

These three codes are listed from top to bottom in alphabetic order. Among all codes generated with this set of characters, these codes appear consecutively in this order.

Write a program to assist in the issuing of these identification codes. Your program will accept a sequence of no more than 50 lower case letters (which may contain repeated characters) and print the successor code if one exists or the message '`No Successor`' if the given code is the last in the sequence for that set of characters.

## *Input and Output*

Input will consist of a series of lines each containing a string representing a code. The entire file will be terminated by a line consisting of a single `#`.

Output will consist of one line for each code read containing the successor code or the words '`No Successor`'.

### Sample input

```
abaacb
cbbaa
#
```

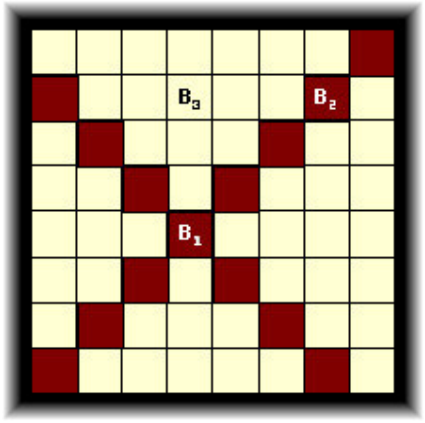
### Sample output

```
ababac
No Successor
```

# Little Bishops

Program bishop.c, bishop.cpp, bishop.java, bishop.pas

A bishop is a piece used in the game of chess which is played on a board of square grids. A bishop can only move diagonally from its current position and two bishops attack each other if one is on the path of the other. In the following figure, the dark squares represent the reachable locations for bishop $B_1$ form its current position. The figure also shows that the bishops $B_1$ and $B_2$ are in attacking positions whereas $B_1$ and $B_3$ are not. $B_2$ and $B_3$ are also in non-attacking positions.




Now, given two numbers $n$ and $k$, your job is to determine the number of ways one can put $k$ bishops on an $n \times n$ chessboard so that no two of them are in attacking positions.

## Input

The input file may contain multiple test cases. Each test case occupies a single line in the input file and contains two integers $n$ ($1 \le n \le 8$) and $k$ ($0 \le k \le n^2$).

A test case containing two zeros for $n$ and $k$ terminates the input and you won't need to process this particular input.

## Output

For each test case in the input print a line containing the total number of ways one can put the given number of bishops on a chessboard of the given size so that no two of them are in attacking positions. You may safely assume that this number will be less than $10^{15}$.

## Sample Input

```
8 6
4 4
0 0
```

### *Sample Output*

```
5599888
260
```

# Graph Coloring

Program coloring.c, coloring.cpp, coloring.java, coloring.pas

You are to write a program that tries to find an optimal coloring for a given graph. Colors are applied to the nodes of the graph and the only available colors are black and white. The coloring of the graph is called optimal if a maximum of nodes is black. The coloring is restricted by the rule that no two connected nodes may be black.

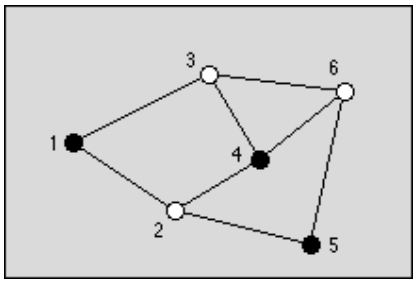



*Figure 1: An optimal graph with three black nodes*

## Input

The graph is given as a set of nodes denoted by numbers 1...$n$, $n \leq 100$, and a set of undirected edges denoted by pairs of node numbers ($n_1$, $n_2$), $n_1 \neq n_2$. The input file contains $m$ graphs. The number $m$ is given on the first line. The first line of each graph contains $n$ and $k$, the number of nodes and the number of edges, respectively. The following $k$ lines contain the edges given by a pair of node numbers, which are separated by a space.

## Output

The output should consist of $2m$ lines, two lines for each graph found in the input file. The first line of should contain the maximum number of nodes that can be colored black in the graph. The second line should contain one possible optimal coloring. It is given by the list of black nodes, separated by a blank.

### Sample Input:

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

### Sample Output:

```
3
1 4 5
```

# Predprevajalnik [KCi]

Program prevajalnik.c, prevajalnik.cpp, prevajalnik.java, prevajalnik.pas

Dobili ste nalogo, da s posebnim orodjem vzdržujete ključno poslovno aplikacijo v podjetju. Orodje je zastarelo in nima več podpore izdajatelja, vendar je še uporabno. Jedro orodja predstavlja poseben prevajalnik. Žal ta prevajalnik ne pozna makro ukazov in pogojnega prevajanja, katerega nujno potrebujete za predvidene spremembe.

Napišite predpravajalnik, ki bo omogočal definiranje in uporabo makro ukazov ter pogojno prevajanje. V ta namen naj predprevajalnik odpre podano vhodno datoteko in na standardni izhod izpiše obdelano izhodno datoteko.

Ukazi predprevajalnika so izključno v vrstica, ki se začnejo z nizom *V. V drugih vrsticah naj predprevajalnik zamenja morebitne makro ukaze z njihovimi definicijami. Predprevajalnik naj ne loči med velikimi in malimi črkami.

Ukazi predprevajalnika so naslednji:

`MACRO <ime> = <vrednost>`

Ukaz MACRO določi vrednost <vrednost> novemu makru z imenom <ime>. V kolikor makro s tem imenom že obstaja, naj predprevajalnik javi napako.

`ime` : Niz dolžine med 3 in 16 znakov; vsebuje lahko samo črke angleške abecede (A..Z)

`vrednost` : Niz poljubnih znakov poljubne dolžine brez presledkov. Morebitni začetni in končni presledki se izločijo iz vrednosti.

```
IFDEF <ime>
<vrstice izvorne kode in/ali drugi predprevajalniški ukazi>
ELSE
<vrstice izvorne kode in/ali drugi predprevajalniški ukazi>
ENDIF
```

Vrstice med ukazoma `IFDEF` ter `ENDIF` naj predpravajalnik obdela, če je makro `<ime>` definiran, sicer obdela vrstice med `ELSE` in `ENDIF`.

```
IFVALUE <ime> <vrednost>
<vrstice izvorne kode in/ali drugi predprevajalniški ukazi>
ELSE
<vrstice izvorne kode in/ali drugi predprevajalniški ukazi>
ENDIF
```

Vrstice med ukazoma `IFVALUE` ter `ELSE` naj predpravajalnik obdela, če je vrednost makra `<ime>` enaka podani vrednosti `<vrednost>`, sicer obdela vrstice med `ELSE` in `ENDIF`. V kolikor makra z imenom `<ime>` ni, javi napako in preneha s predprevajanjem.

Program naj omogoča gnezdenje `IFDEF` in/ali `IFVALUE` ukazov do (vsaj) 20 nivojev globoko. Ukaz `ELSE` ni obvezen niti v `IFDEF` niti v `IFVALUE`.

`ERROR <sporočilo>`

Izpiše <sporočilo> ter konča predprevajanje.

`@<ime>@`

Zaporedje `@<ime>@` na izhodu zamenja z vsebino makra z imenom `<ime>`. V kolikor je `<ime>` prazen niz, na izhod zapiši en znak »@«. Če makro `<ime>` ni definiran, izpiše napako in konča s predprevajanjem.

Napake se izpisujejo v obliki:
`*** Error: <sporočilo>`

pri čemer je `<sporočilo>` eno od naslednjih (mislim, da je očitno kaj pomenijo)

```
Unknown command 'cmd'
Macro 'name' not defined
ENDIF without IF
ELSE without IF
ENDIF missing
```

in seveda katerokoli sporočilo, ki ga izpiše ukaz ERROR.

## Input

```
*V MACRO platforma = Linux_RedHat_9
Poljubno besedilo
*V IFDEF platforma
*V IFVALUE platforma Linux_RedHat_9
Tole je vkljuceno
*V ELSE
Tole ni vkljuceno
*V ENDIF
*V ENDIF
Preostalo besedilo
Prevajamo na platformi @platforma@
*V ERROR Napaka kar tako
```

## Output

```
Poljubno besedilo
Tole je vkljuceno
Preostalo besedilo
Prevajamo na platformi Linux_RedHat_9
*** Error: Napaka kar tako
```

# Jack Straws

Program straws.c, straws.cpp, straws.java, straws.pas

In the game of Jack Straws, a number of plastic or wooden "straws" are dumped on the table and players try to remove them one-by-one without disturbing the other straws. Here, we are only concerned with if various pairs of straws are connected by a path of touching straws. You will be given a list of the endpoints for some straws (as if they were dumped on a large piece of graph paper) and then will be asked if various pairs of straws are connected. Note that touching is connecting, but also two straws can be connected indirectly via other connected straws.

## *Input:*

A problem consists of multiple lines of input. The first line will be an integer $n$ ($1 < n < 13$) giving the number of straws on the table. Each of the next n lines contain 4 positive integers, $x_1$ , $y_1$ , $x_2$ and $y_2$ , giving the coordinates, ($x_1$ ; $y_1$); ($x_2$ ; $y_2$) of the endpoints of a single straw. All coordinates will be less than 100. (Note that the straws will be of varying lengths.) The first straw entered will be known as straw #1, the second as straw #2, and so on. The remaining lines of input (except for the final line) will each contain two positive integers, $a$ and $b$, both between 1 and $n$, inclusive. You are to determine if straw $a$ can be connected to straw $b$. When $a = 0 = b$, the input is terminated. There will be no illegal input and there are no zero-length straws.

## *Output:*

You should generate a line of output for each line containing $a$ pair $a$ and $b$, except the final line where $a = 0 = b$. The line should say simply "`CONNECTED`", if straw a is connected to straw b, or "`NOT CONNECTED`", if straw $a$ is not connected to straw $b$. For our purposes, a straw is considered connected to itself.

### Sample Input:

```
7
1 6 3 3
4 6 4 9
4 5 6 7
1 4 3 5
3 5 5 5
5 2 6 3
5 4 7 2
1 4
1 6
3 3
6 7
2 3
1 3
0 0
```

### Sample Output:

```
CONNECTED
NOT CONNECTED
CONNECTED
CONNECTED
NOT CONNECTED
CONNECTED
```

# What's In A Name?

Program name.c, name.cpp, name.java, name.pas

The FBI is conducting a surveillance of a known criminal hideout which serves as a communication center for a number of men and women of nefarious intent. Using sophisticated decryption software and good old fashion wiretaps, they are able to decode any e-mail messages leaving the site. However, before any arrest warrants can be served, they must match actual names with the user ID's on the messages. While these criminals are evil, they're not stupid, so they use random strings of letters for their ID's (no dillingerj ID's found here). The FBI knows that each criminal uses only one ID. The only other information they have which will help them is a log of names of the people who enter and leave the hideout. In many cases, this is enough to link the names to the ID's.

## *Input*

Input consists of one problem instance. The first line contains a single positive integer *n* indicating the number of criminals using the hideout. The maximum value for *n* will be 20. The next line contains the *n* user ID's, separated by single spaces. Next will be the log entries in chronological order. Each entry in the log has the form *type arg* , where *type* is either **E**, **L** or **M**: **E** indicates that criminal *arg* has entered the hideout; **L** indicates criminal *arg* has left the hideout; **M** indicates a message was intercepted from user ID *arg*. A line containing only the letter **Q** indicates the end of the log. Note that not all user ID's may be present in the log but each criminal name will be guaranteed to be in the log at least once. At the start of the log, the hideout is presumed to be empty. All names and user ID's consist of only lowercase letters and have length at most 20. Note: The line containing only the user ID's may contain more than 80 characters.

## *Output*

Output consists of n lines, each containing a list of criminal names and their corresponding user ID's, if known. The list should be sorted in alphabetical order by the criminal names. Each line has the form *name* **:** *userid* , where *name* is the criminal's name and *userid* is either their user ID or the string **???** if their user ID could not be determined from the surveillance log.

### Sample Input

```
7
bigman mangler sinbad fatman bigcheese frenchie capodicapo
E mugsy
E knuckles
M bigman
M mangler
L mugsy
E clyde
E bonnie
M bigman
M fatman
M frenchie
L clyde
M fatman
E ugati
M sinbad
E moriarty
```

```
E booth
Q
```

### Sample Output

```
bonnie:fatman
booth:???
clyde:frenchie
knuckles:bigman
moriarty:???
mugsy:mangler
ugati:sinbad
```

# Spreadsheet [KCi]

Program spreadsheet.c, spreadsheet.cpp, spreadsheet.java, spreadsheet.pas

In 1979, Dan Bricklin and Bob Frankston wrote VisiCalc, the first spreadsheet application. It became a huge success and, at that time, was the killer application for the Apple II computers. Today, spreadsheets are found on most desktop computers.

The idea behind spreadsheets is very simple, though powerful. A spreadsheet consists of a table where each cell contains either a number or a formula. A formula can compute an expression that depends on the values of other cells. Text and graphics can be added for presentation purposes.

You are to write a very simple spreadsheet application. Your program should accept several spreadsheets. Each cell of the spreadsheet contains either a numeric value (integers only) or a formula, which only support sums. After having computed the values of all formulas, your program should output the resulting spreadsheet where all formulas have been replaced by their value.

| | | | | | | |
|---|---|---|---|---|---|---|
| A1 | B1 | C1 | D1 | E1 | F1 | ... |
| A2 | B2 | C2 | D2 | E2 | F2 | ... |
| A3 | B3 | C3 | D3 | E3 | F3 | ... |
| A4 | B4 | C4 | D4 | E4 | F4 | ... |
| A5 | B5 | C5 | D5 | E5 | F5 | ... |
| A6 | B6 | C6 | D6 | E6 | F6 | ... |
| ... | ... | ... | ... | ... | ... | ... |

*Figure 1: Naming of the top left cells*

## Input

The first line of the input file contains the number of spreadsheets to follow. A spreadsheet starts with a line consisting of two integer numbers, separated by a space, giving the number of columns and rows. The following lines of the spreadsheet each contain a row. A row consists of the cells of that row, separated by a single space.

A cell consists either of a numeric integer value or of a formula. A formula starts with an equal sign (=). After that, one or more cell names follow, separated by plus signs (+). The value of such a formula is the sum of all values found in the referenced cells. These cells may again contain a formula. There are no spaces within a formula.

You may safely assume that there are no cyclic dependencies between cells. So each spreadsheet can be fully computed.

The name of a cell consists of one to three letters for the column followed by a number between 1 and 999 (including) for the row. The letters for the column form the following series: A, B, C, ..., Z, AA, AB, AC, ..., AZ, BA, ..., BZ, CA, ... ZZ, AAA, AAB, AAC, ... AAZ, ABA, ..., ABZ, ACA, ..., ZZZ. These letters correspond to the number from 1 to 18278. The top left cell has the name A1. See figure 1.

## Output

The output of your program should have the same format as the input, except that the number of spreadsheets and the number of columns and rows are not repeated. Furthermore, all formulas should be replaced by their value.

Sample Input:

```
1
4 3
10 34 37 =A1+B1+C1
40 17 34 =A2+B2+C2
=A1+A2 =B1+B2 =C1+C2 =D1+D2
```

Sample Output:

```
10 34 37 81
40 17 34 91
50 51 71 172
```

# "Accordian" Patience

Program patience.c, patience.cpp, patience.java, patience.pas

You are to simulate the playing of games of "Accordian" patience, the rules for which are as follows:

Deal cards one by one in a row from left to right, not overlapping. Whenever the card matches its immediate neighbour on the left, or matches the third card to the left, **it** may be moved onto that card. Cards match if they are of the same suit or same rank. After making a move, look to see if it has made additional moves possible. Only the top card of each pile may be moved at any given time. Gaps between piles should be closed up as soon as they appear by moving all piles on the right of the gap one position to the left. Deal out the whole pack, combining cards towards the left whenever possible. The game is won if the pack is reduced to a single pile.

Situations can arise where more than one play is possible. Where two cards may be moved, you should adopt the strategy of always moving the leftmost card possible. Where a card may be moved either one position to the left or three positions to the left, move it three positions.

## *Input*

Input data to the program specifies the order in which cards are dealt from the pack. The input contains pairs of lines, each line containing 26 cards separated by single space characters. The final line of the input file contains a # as its first character. Cards are represented as a two character code. The first character is the face-value (A=Ace, 2-9, T=10, J=Jack, Q=Queen, K=King) and the second character is the suit (C=Clubs, D=Diamonds, H=Hearts, S=Spades).

## *Output*

One line of output must be produced for each pair of lines (that between them describe a pack of 52 cards) in the input. Each line of output shows the number of cards in each of the piles remaining after playing "Accordian patience" with the pack of cards as described by the corresponding pairs of input lines.

### Sample Input

```
QD AD 8H 5S 3H 5H TC 4D JH KS 6H 8S JS AC AS 8D 2H QS TS 3S AH 4H TH TD 3C 6S
8C 7D 4C 4S 7S 9H 7C 5D 2S KD 2D QH JD 6D 9D JC 2C KH 3D QC 6C 9S KC 7H 9C 5C
AC 2C 3C 4C 5C 6C 7C 8C 9C TC JC QC KC AD 2D 3D 4D 5D 6D 7D 8D TD 9D JD QD KD
AH 2H 3H 4H 5H 6H 7H 8H 9H KH 6S QH TH AS 2S 3S 4S 5S JH 7S 8S 9S TS JS QS KS
#
```

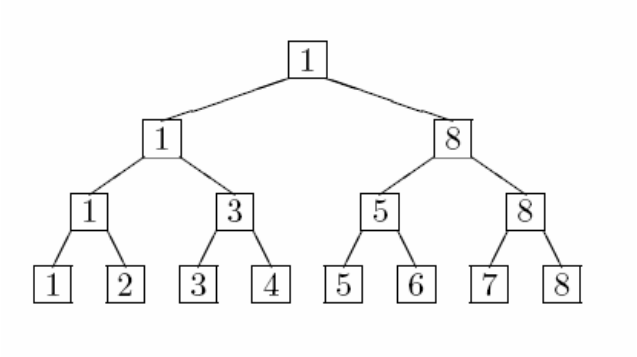
### Sample Output

```
6 piles remaining: 40 8 1 1 1 1
1 pile remaining: 52
```

# Knockout Tournament

Program tournament.c, tournament.cpp, tournament.java, tournament.pas

In a knockout tournament there are $2^n$ players. One loss and a player is out of the tournament. Winners then play each other with the new winners advancing until there is only one winner left. If we number the players 1, 2, 3,..., $2^n$, with the first round pairings 2k-1 vs 2k, for k = 1, 2,..., $2^{n-1}$, then we could give the results of the tournament in a complete binary tree. The winners are indicated in the interior nodes of the tree. Below is an example of a tournament with n = 3.




After the tournament, some reporters were arguing about the relative ranking of the players, as determined by the tournament results. It's assumed that if player A beats player B who in turn beats player C, that player A will also beat player C; that is, winning is transitive. Now there is no doubt who the best player is. The question is what is the highest ranking a player can reasonably claim as a result of the tournament and what is the worst ranking a player can have, as a result of the tournament? For example, in the above tournament player 2, having lost to the eventual winner, could claim to be the 2nd best player in the field, but could well be the worst (ranked 8th). Player 5 could claim to be as high as 3rd (having lost to someone who could be 2nd) but no worse than 7th (having beaten one player in the 1st round).

You are to determine the highest and lowest possible rankings of a set of players in the field, given the results of the tournament.

## *Input*

There will be multiple input instances. The input for each instance consists of three lines. The first line will contain a positive integer n < 8, indicating there are $2^n$ players in the tournament, numbered 1 through $2^n$, paired in the manner indicated above. A value of n = 0 indicates end of input. The next line will contain the results of each round of the tournament (listed left-to-right) starting with the 1st round. For example, the tournament above would be given by

```
1 3 5 8 1 8 1
```

The final line of input for each instance will be a positive integer $m$ followed by integers $k_1$,..., $k_m$, where each $k_i$ is a player in the field.

## *Output*

For each $k_i$, issue one line of output of the form:

`Player` $k_i$ `can be ranked as high as` $h$ `or as low as` $l$`.`

where you supply the appropriate numbers. These lines should appear in the same order as the $k_i$ did in the input. Output for problem instances should be separated with a blank line.

### Sample Input

```
3
1 3 5 8 1 8 1
2 2 5
4
2 3 6 7 9 11 14 15 3 6 9 15 6 9 6
4 1 15 7 6
0
```

### Sample Output

```
Player 2 can be ranked as high as 2 or as low as 8.
Player 5 can be ranked as high as 3 or as low as 7.
Player 1 can be ranked as high as 4 or as low as 16.
Player 15 can be ranked as high as 3 or as low as 13.
Player 7 can be ranked as high as 2 or as low as 15.
Player 6 can be ranked as high as 1 or as low as 1.
```