Cribbage Count

In Cribbage scoring, there are several ways of accumulating points in a pile of playing cards. Among these are

15's. Any combination of cards whose values sum to 15 scores 2 points. (Aces have value 1 and picture cards have value 10.)

Pairs. Any pair of cards of the same rank scores 1 point.

Runs. A maximal sequence of length $m \geq 3$ of cards of consecutive rank scores m points.

In this problem, you will be given a sequence of up to 7 cards and should report the total number of points scored according to these rules.

Input will be a series of piles of cards, one per line. Each line will consist of at most 7 characters from the set $\{A, 2, 3, 4, 5, 6, 7, 8, 9, 10, J, Q, K\}$. Output should be the number of points for that pile. Since each pile is selected from a single deck of cards, there will be at most 4 cards of any rank.

Sample Input	Sample Output
`2 A 3 9`	`5`
`5 5 5`	`5`
`9 J 10 A Q`	`4`
`2 Q 9 8 K`	`0`
`4 6 4 5`	`11`

(*Explanations.*

Pile 1: There is run of length 3 and one way of making 15

Pile 2: There is way of making 15 and there are 3 possible pairs.

Pile 3: There is a run of length 4.

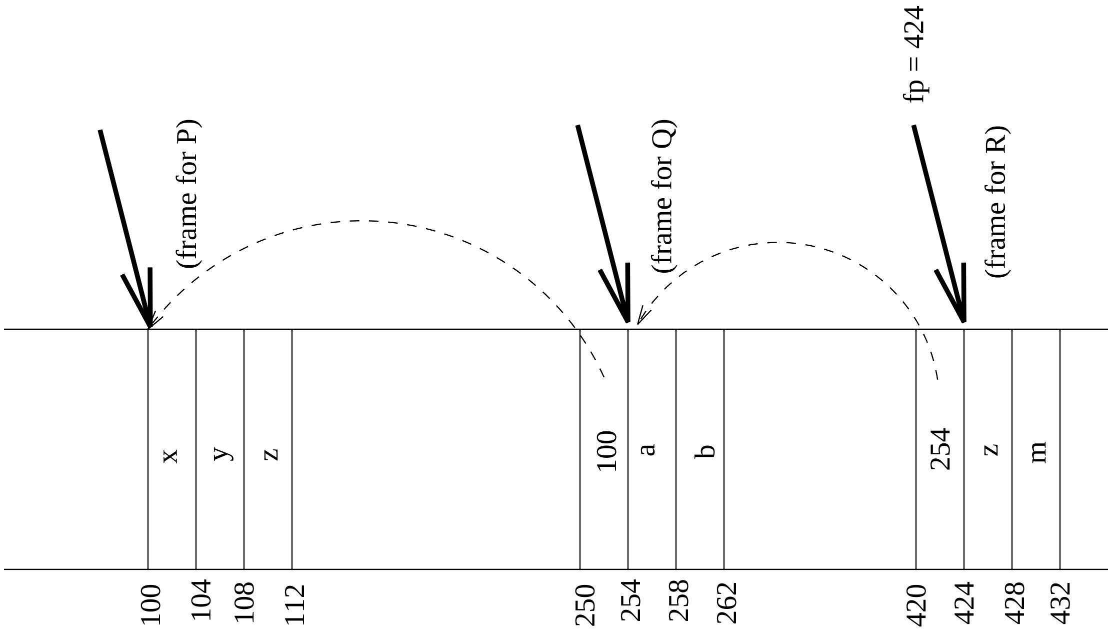
Pile 4: No score.

Pile 5: There are two ways of making 15, one pair, and two runs of length 3.)

Static Links

Compilers for many high-level languages use a stack for runtime storage management. In the simplest case, every procedure invocation is allocated a portion of the stack called a *frame*, which holds the local variables defined by the procedure. In more advanced languages where procedures may be nested, each frame has a pointer, called a *static link*, which points to the frame of the parent procedure. The static link is used to access variables defined in outer scopes. For the purposes of this problem, we assume that the static link is always stored at offset `-4` of the frame pointer, and that the local variables are stored at offsets 0, 4, 8, etc.

For example, consider the situation depicted in the figure.



A procedure P defines three variables x, y, and z. Within P, a procedure Q defines two variables a and b. As the stack organization shows, Q may access the variables defined in P by following the static link (the dotted line). Within Q, another procedure R defines two more variables z and m. If procedure R is currently active, a global variable `fp` points to its frame. The variables local to R are stored at addresses `(fp+0)` and `(fp+4)`. To access these variables we write `(*(fp+0))` and `(*(fp+4))` respectively. The variable a can be accessed by following the static link to reach the frame for Q and then using offset 0. We express this computation as `(*((*(fp-4))+0))`. The variable y can be accessed by following two static links, and then using offset 4. We express this computation as `(*((*((*(fp-4))-4))+4))`. The variable z in procedure P cannot be accessed from R since the declaration of z in R shadows the one in P.

You are to implement the part of the compiler that generates intermediate code for accessing variables.

The input will be sets of nested procedure declarations. The first line of each set gives the number n (≤ 10) of nested procedure declarations that follow. The next n lines list the variables declared at each level. (Each line will have at most 10 variables where each variable name will be one character long.) An additional line then lists the variables for which you should generate code assuming that last frame is the current one.

The output should list the code for accessing each variable on a separate line. The syntax of the output code is shown in the samples.

Sample Input

```
3
x y z
a b
z m
z m a y
3
x y z
x y
x
x y z
```

Sample Output

```
(* (fp + 0))
(* (fp + 4))
(* ((* (fp - 4)) + 0))
(* ((* ((* (fp - 4)) - 4)) + 4))

(* (fp + 0))
(* ((* (fp - 4)) + 4))
(* ((* ((* (fp - 4)) - 4)) + 8))
```

Multicast Routing

For efficient support of one-to-many communication, a network must be able to send a packet from a source node to a set of destination nodes, called a *multicast group*. Thus, the network builds a *multicast tree*, rooted at the source and extending to the group members. When the source *multicasts* a packet, the network sends it along the tree, duplicating the packet only when it reaches a node that has more than one child. This saves considerable bandwidth compared to *unicast* routing, where the source sends a separate packet to each group member. To reduce delay, the network typically will construct a multicast tree using the shortest (by weight) paths from the source to the group members. These are the same paths that would be used by unicast routing.

In this problem, you will be given a set of networks along with source and group members. You should compute, in each case, a single-source shortest-path tree and the number of packet transmissions required when using *this* tree for multicast and for unicast. (A *packet transmission* is defined as the act of sending a packet from a node to its neighbor via a direct link.)

The first line for each network will give a pair n, s of integers where n (≤ 15) indicates the number of nodes in the network and s $(1 \leq s \leq n)$ is the label of the source node. (We assume the nodes are labelled $1, 2, \ldots, n$). The next n lines give the adjacency matrix of a positively *weighted* graph representing the network (if the row i, column j entry is $w > 0$ then there is a direct link from i to j with weight w). An additional line then lists the set of nodes forming the multicast group. (Inputs on a single line are separated by white space.)

Your output for each network should have the following form.

- The shortest-path multicast tree rooted at the source and extending to the members. For each parent node i in the tree, there should be one listing "i :" followed by the children of i (see sample output).
- The next two lines should give the number of multicast transmissions and then the number of unicast transmissions (see sample output).

Leave a space between elements on a single line and skip a line between networks.

Sample Input

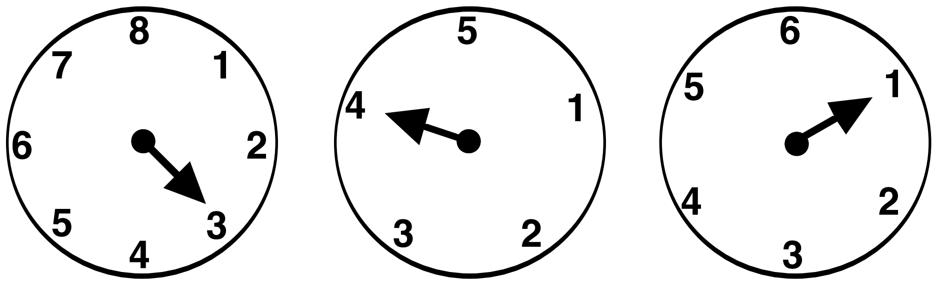
```
3 1
0 1 3
1 0 1
3 1 0
2 3
3 1
0 1 3
1 0 1
3 1 0
3
5 2
0 3 1 0 2
3 0 1 4 4
1 1 0 2 2
0 4 2 0 1
2 4 2 1 0
1 4 5
5 4
0 5 2 0 0
5 0 0 1 0
2 0 0 7 0
0 1 7 0 3
0 0 0 3 0
1 2 5 3
```

Sample Output

```
Shortest-path tree:
1: 2
2: 3
Multicast transmissions: 2
Unicast transmissions: 3

Shortest-path tree:
1: 2
2: 3
Multicast transmissions: 2
Unicast transmissions: 2

Shortest-path tree:
2: 3
3: 1 4 5
Multicast transmissions: 4
Unicast transmissions: 6

Shortest-path tree:
2: 1
4: 2 3 5
Multicast transmissions: 4
Unicast transmissions: 5
```

Galactic Days

Although Starfleet Academy is located on Earth, cadets traditionally wear watches that record time in the fashion of their home world. To keep class time uniform, the length of an "hour" is standard. However, the number of hours around the watch faces vary. In the following figure, it is 3 o'clock or 4 o'clock or 1 o'clock according to which alien watch one reads.



Academy "days" for a student squadron can be quite long as they begin when all cadets' watch-hands are directed straight up and end when they are next simultaneously straight up.

In this problem, you are given the times (on the hour) shown on the watches of squadrons and need to determine the number of hours left in the squadron days.

Input to your program will be sets of watch times for squadrons. The first line for each squadron will be the number of watches n in the set ($1 \le n \le 10$). The following n lines will each consist of a pair of integers w, h where w ($2 \le w \le 25$) is the number of integers around a watch dial and h is the current position of the hour hand ($1 \le h \le w$ with w always straight up).

Output for each squadron should be the number of hours remaining until the end of the squadron's current "day".

Sample Input

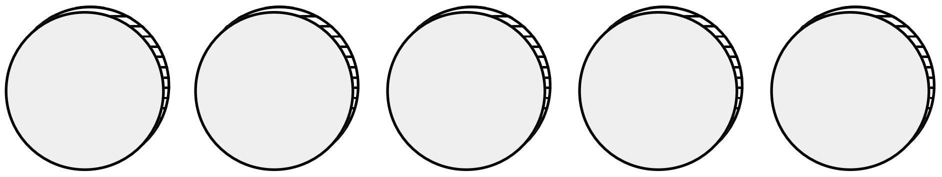
```
2
2 1
3 1
5
4 3
5 4
6 5
7 6
8 7
1
10 4
3
8 3
5 4
6 1
```

Sample Output

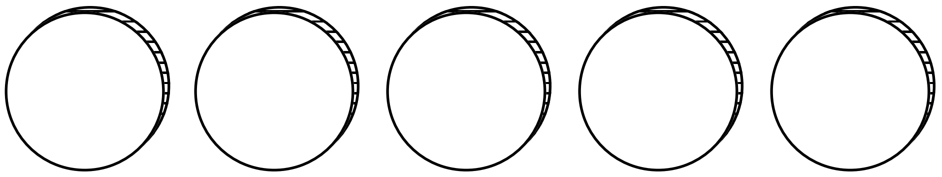
```
5
1
6
101
```

Capsized Coins

A puzzle starts with a line of coins, face down



and the objective is to turn them all face up.



The puzzle can vary in the number n of coins and the legal moves. A legal move is specified by a subset of $\{1, 2, \ldots, n\}$ that indicates the positions of the coins to be turned by that move.

In this problem, you are given a set of legal moves and must determine the moves that solve the puzzle or else determine that it is unsolvable.

Input will be a sequence of puzzle specifications. A specification begins with a line containing two integers, the number n of coins ($n \leq 6$) and the number m of legal moves ($m \leq 6$), separated by a space. The legal moves then follow, one to a line; each consists of a single-character label followed by a single space and then the positions (numbers between 1 and n) of the coins turned by that move, separated by spaces.

Output for each puzzle is a set of labels of moves that solve the puzzle or else "`Unsolvable`".

Sample Input

```
4 2
a 1 2
b 3 4
5 4
a 1 3 5
b 1 2 4
c 1 2
d 1
4 2
a 1 2 3
b 1 2 4
```

Sample Output

```
a b
a b d
Unsolvable
```