UNIVERSITY OF CALIFORNIA
Department of Electrical Engineering
and Computer Sciences
Computer Science Division

Programming Contest **P. N. Hilfinger**
Fall 2013

2013 Programming Problems

Please make sure your electronic registration is up to date, and that it contains the correct account you are going to be using to submit solutions (we connect names with accounts using the registration data).

To set up your account, execute

```
source ~ctest/bin/setup
```

in all shells that you are using. (This is for those of you using csh-like shells. Those using `bash` should instead type

```
source ~ctest/bin/setup.bash
```

Others will have to examine this file and do the equivalent for their shells.)

This booklet should contain 9 problems on 16 pages. You have 5 hours in which to solve as many of them as possible. Put each complete C solution into a file N`.c`, each complete C++ solution into a file N`.cc`, and each complete Java program into a file N`.java`, where N is the number of the problem. Each program must reside entirely in a single file. In Java, the class containing the main program for problem N must be named `P`N (yes, it is OK to have a Java source file whose base name consists of a number, even though it doesn't match the name of the class). Do not make class `P`N public, or the Java compiler will complain. Each C/C++ file should start with the line

```
#include "contest.h"
```

and must contain no other `#include` directives, except as indicated below. Upon completion, each program *must* terminate by calling `exit(0)` (or `System.exit(0)` in Java).

Aside from files in the standard system libraries and those we supply, you may not use any pre-existing computer-readable files to supply source or object code; you must type in everything yourself. Selected portions of the standard `g++` class library are included among of the standard libraries you may use: specifically, the headers `string`, `vector`, `iostream`, `iomanip`, `sstream`, `fstream`, `map`, `set`, `unordered_map`, `unordered_set`, and `algorithms`. Likewise, you can use the standard C I/O libraries (in either C or C++), and the math library (header `math.h`). In Java, you may use the standard packages `java.lang`, `java.io`,

`java.text`, and `java.util` and their subpackages (but this year, *not* java.math). You may not use utilities such as `yacc`, `bison`, `lex`, or `flex` to produce programs. Your programs may not create other processes (as with the `system`, `popen`, `fork`, or `exec` series of calls or their Java-library equivalents). You may use any inanimate reference materials you desire, but no people. You can be disqualified for breaking these rules.

There are two ways to submit solutions: by a command-line program, and over the web. Submit from the command line on the instructional machines. When you have a solution to problem number N that you wish to submit, use the command

`submit` N

from the directory containing N`.c`, N`.cc`, or N`.java`. Before actually submitting your program, `submit` will first compile it and run it on one sample input file. No submission that is sent after the end of the contest will count. You should be aware that `submit` takes some time before it actually sends a program. In an emergency, you can use

`submit -f` N

which submits problem N without compiling or running it.

To submit from the web, go to our contest announcement page:

`http://inst.cs.berkeley.edu/~ctest/contest/index.html`

and click on the "web interface" link. You will go to a page from which you can upload and submit files from your local computer (at home or in the labs). On this page, you can also find out your score, and look at error logs from failed submissions.

Regardless of the method you use for submission, your results are also mailed back to you at the account from which you submitted (in the case of web submission, that is the instructional account you used to validate yourself). Use the `https://imail.eecs.berkeley.edu` page to retrieve this mail.

You will be penalized for incorrect submissions that get past the simple test administered by `submit`, so be sure to test your programs (if you get a message from `submit` saying that it failed, you will *not* be penalized). All tests (for any language) will use the compilation command

`contest-gcc` N

followed by one or more execution tests of the form (Bourne shell):

`./`N `<` *test-input-file* `>` *test-output-file* `2>` *junk-file*

which sends normal output to *test-output-file* and error output to *junk-file*. The output from running each input file is then compared with a standard output file, or tested by a program in cases where the output is not unique. In this comparison, leading and trailing blanks are ignored and sequences of blanks are compressed to single blanks. Otherwise, the comparison is literal; be sure to follow the output formats *exactly*. It will do no good to argue about how trivially your program's output differs from what is expected; you'd be

arguing with a program. Make sure that the last line of output ends with a newline. Your program must not send any output to `stderr`; that is, the temporary file *junk-file* must be empty at the end of execution. Each test is subject to a time limit of about 10 seconds (on torus.cs). You will be advised by mail whether your submissions pass (use the imail account at

`https://imail.eecs.berkeley.edu`

and log in with the account you registered to use for the contest.) You can also view this information using the web interface described above.

In the actual ACM contests, you will not be given nearly as much information about errors in your submissions as you receive here. Indeed, it may occur to you to simply take the results you get back from our automated judge and rewrite your program to print them out verbatim when your program receives the corresponding input. Be warned that I will feel free to fail any submission in which I find this sort of hanky-panky going on (retroactively, if need be).

The command `contest-gcc` N, where N is the number of a problem, is available to you for developing and testing your solutions. For C and C++ programs, it is roughly equivalent to

`gcc -Wall -o` N `-O2 -g -I`*our-includes* N`.*` *our-libraries* `-lm`

For Java programs, it is equivalent to

`javac -g -classpath .:`*our-classes* N`.java`

followed by a command that creates an executable file called N that runs the command

`java -cp .:`*our-classes* `P`N

when executed (so that it makes the execution of Java programs look the same as execution of C/C++ programs). The *our-includes* directory (typically `~ctest/include`) contains `contest.h` for C/C++, which also supplies the standard header files. The *our-libraries* and *our-packages* files and directories provide the additional tools we've provided this year. The files in `~ctest/submission-tests/`N, where N is a problem number, contain the input files and standard output files that `submit` uses for its simple tests.

All input will be placed in `stdin`. You may assume that the input conforms to any restrictions in the problem statement; you need not check the input for correctness. Consequently, you C/C++ programmers are free to use `scanf` to read in numbers and strings and `gets` to read in lines.

Terminology. The terms *free format* and *free-format input* indicate that input numbers, words, or tokens are separated from each other by arbitrary whitespace characters. By standard C/UNIX convention, a whitespace character is a space, tab, return, newline, formfeed, or vertical tab character. A *word* or *token,* accordingly, is a sequence of non-whitespace characters delimited on each side by either whitespace or the beginning or end of the input file.

Scoring. Scoring will be according to the ACM Contest Rules. You will be ranked by the number of problems solved. Where two or more contestants complete the same number of problems, they will be ranked by the *total time* required for the problems solved. The total time is defined as the sum of the *time consumed* for each of the problems solved. The time consumed on a problem is the time elapsed between the start of the contest and successful submission, plus 20 minutes for each unsuccessful submission, and minus the time spent judging your entries. Unsuccessful submissions of problems that are not solved do not count. As a matter of strategy, you can derive from these rules that it is best to work on the problems in order of increasing expected completion time.

Protests. Should you disagree with the rejection of one of your problems, first prepare a file containing the explanation for your protest, and then use the `protest` command (without arguments). It will ask you for the problem number, the submission number (submission 1 is your first submission of a problem, 2 the second, etc.), and the name of the file containing your explanation. Do not protest without first checking carefully; groundless protests will result in a 5-minute penalty (see Scoring above). The Judge will *not* answer technical questions about C, C++, Java, the compilers, the editor, the debugger, the shell, or the operating system.

Notices. During the contest, the Web page at URL

`http://inst.cs.berkeley.edu/~ctest/contest/index.html`

will contain any urgent announcements, plus a running scoreboard showing who has solved what problems. Sometimes, it is useful to see what problems others are solving, to give you a clue as to what is easy.

1. [Miguel Revilla; adapted from the Universidad de Valladolid's collection]
Write a program to compute the volume of the intersection of a set of cubes.

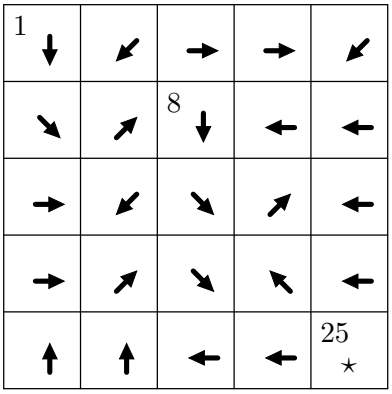
The input will consist of several sets of cubes in free format. Each set begins with a positive integer, which indicates the number of cube descriptions that follow. Each description consists of four integers, x, y, z, and s, indicating the coordinates of a corner of the cube and the distance ($s > 0$) that the three edges of the cube extend from that corner in the positive direction.

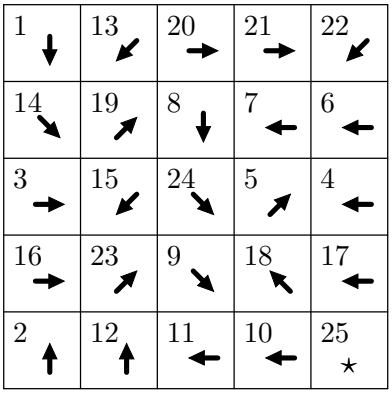
For each set of cubes, output one line containing the volume of their intersection.

Example:

Input	Output
2	25
0 0 0 10	9
9 1 1 5	
3	
0 0 0 10	
9 1 1 5	
8 2 2 3	

2. A puzzle game variously known as "Signpost" and "Pfeilpfad" presents you with an $N \times N$ square grid of cells, each of which, except the final goal square, contains an arrow pointing in one of the eight compass directions, and some of which contain integers between 1 and N^2, inclusive. The goal of the puzzle is to number the rest of the squares so that the grid contains each of the integers between 1 and N^2 and the arrow in any given square (again, except the goal square, which is number N^2) points toward the next integer in sequence. For example, the puzzle on the left below has the solution on the right.





The goal square is marked with a $\star$ rather than an arrow, and for this problem will always be in the lower right corner. The starting square will always be in the upper left. Both squares will have numbers initially, and one or more other squares *may* have numbers initially. The solution will always exist and be unique.

Input will consist of one or more sets of data in free format, each representing one puzzle. Each set begins with a positive integer, $N \leq 5$, giving the size of the puzzle (which is an $N \times N$ grid). There will then follow N^2 pairs of integers indicating the arrows and numbers in the cells, from left to right, top to bottom. Each pair—k, d—will indicate the number in the square (or 0 if no number is given), and the direction, given as an integer in the range -1 to 7, inclusive. A direction of -1 indicates the goal, 0 indicates "north" (up), 1 indicates "northeast", 2 indicates east, and so on clockwise. The last set of data is followed by an integer 0, indicating the end.

Output the solution for each set by giving just the numbers in a grid, using the format of the example. Each solution is unique.

Example:

Input	Output

`5`
`1 4  0 5  0 2  0 2  0 5`
`0 3  0 1  8 4  0 6  0 6`
`0 2  0 5  0 3  0 1  0 6`
`0 2  0 1  0 3  0 7  0 6`
`0 0  0 0  0 6  0 6  25 -1`
`1`
`1 -1`
`0`
	`Set 1:`
	`  1 13 20 21 22`
	`14 19  8  7  6`
	`  3 15 24  5  4`
	`16 23  9 18 17`
	`  2 12 11 10 25`
	
	`Set 2:`
	`  1`

3. [Adapted from the Universidad de Valladolid's collection] There are many ways to extrapolate a sequence of values. One very old technique uses a *difference table.* To extrapolate a sequence of N values (such as 3, 6, 10, and 15), one first creates a triangular $N \times N$ table in which the first column is the sequence to be extrapolated and each subsequent column contains the differences of adjacent entries in the previous column. For example:

3			
	3		
6		1	
	4		0
10		1	
	5		
15			

The last column will always contain only a single value (not always 0).

To extrapolate using a difference table we extend the last column with copies of the (single) value in that column and then work backwards to add values to the previous columns, reading the extrapolated values from the first column. For our example, the next value would be 21:

3			
	3		
6		1	
	4		0
10		1	
	5		0
15		1	
	6		
21			

The input for this problem will be a set of extrapolation requests in free format. For each request the input will contain first an integer N, with $0 < N \leq 100$, which specifies the number of values in the sequence to be extended. It will be followed by N integers representing the given elements in the sequence. The last item in the input for each extrapolation request is $K \geq 1$, the index of the value in the sequence to be returned (numbering from 1).

For each input set, report the Kth value in the sequence, using the format shown in the example on the next page. You may actually use any method you want to compute this value, but if you use the method shown above, all intermediate results and the answer will be representable as 32-bit signed integers.

Example:

Input	Output
`4 3 6 10 15 5`	`Term 5 of the sequence is 21`
`4 3 6 10 15 2`	`Term 2 of the sequence is 6`
`4 3 6 10 15 6`	`Term 6 of the sequence is 28`
`3 2 4 6 23`	`Term 23 of the sequence is 46`
`6 3 9 12 5 18 -4 16`	`Term 16 of the sequence is -319268`

4. The coefficients of polynomials do not have to be ordinary real or complex numbers. In this problem, the coefficients come from the smallest field, GF(2), which has two elements (0 and 1). Addition in GF(2) is defined as $0 + 0 = 1 + 1 = 0$, $0 + 1 = 1 + 0 = 1$ (so that $1 \equiv -1$). Multiplication is the same as for integers. Since the coefficients of polynomials over GF(2) are all 0s and 1s, they may be represented as bits, and we can therefore denote any univariate polynomial over GF(2) as an integer numeral in base 2. For example, $x^8 + x^4 + x^3 + x + 1$ can be denoted by the integer 283 (100011011_2).

As you know from high-school algebra, there are division and modulo operations on polynomials, which can be carried out using a form of long division. This works for polynomials over GF(2) as well as it does for polynomials over the reals. For example, to compute

$$\frac{x^5 + 1}{x^2 + x + 1}$$

we can proceed like this (as it would be written in the U.S):

$$\begin{array}{rl}
 & x^3 + x^2 + 1 \\
\hline
x^2 + x + 1 \;) & x^5 + 1 \\
 & x^5 + x^4 + x^3 \\
\hline
 & x^4 + x^3 + 1 \\
 & x^4 + x^3 + x^2 + 1 \\
\hline
 & x^2 + 1 \\
 & x^2 + x + 1 \\
\hline
 & x
\end{array}$$

Giving $x^3 + x^2 + 1$ with a remainder of x. Thus, $x^5 + 1 \bmod x^2 + x + 1 = x$. (If it seems confusing to subtract, e.g., x^4 from 0 and get x^4, just remember that $-x^4 = x^4$ for these polynomials, since $-1 \equiv 1$ for the coefficients.)

Your program is to take pairs of positive integers a and b (in decimal), interpret them as polynomials as described above, and output the integer corresponding to $a \bmod b$. So, for example, for $a = 100001_2 = 33$ (standing for $x^5 + 1$) and $b = 111_2 = 7$ (standing for $x^2 + x + 1$), you would output $10_2 = 2$ (standing for x).

The input consists of a sequence of such (a, b) pairs in free format, where $0 \le a < 2^{31}, 0 < b < 2^{31}$. The output consists of a line for each input pair that echoes the inputs and reports the result in the format shown in the example.

Example:

Input	Output
`33 7 283`	`As polynomials, 33 mod 7 = 2`
`4`	`As polynomials, 283 mod 4 = 3`

5. Numerous programs provide some notation for specifying patterns that match strings. A common component of such pattern notations is the *character class,* meaning "one of the following characters." For example, in the Unix shell and in regular expressions in Perl, Java, and Python, `[adh-p]` means "any of the characters 'a', 'd', and 'h' through 'p'." For this problem, we'll use a slight variation having the following syntax:

```
charset: '[' items '|' charset_tail
items: ϵ
     | items char
     | items char '-' char
charset_tail : ']'
     | char charset_tail
char: any of the 26 lower-case alphabetic characters
```

(ϵ denotes the empty string.) Here, "items" represents a set of characters as in the example "`adh-p`" above. The notation "`[`S_1`|`S_2`]`" means "any character that is in S_1 and is *not* in S_2." For example,

`[a-cz|]`	Any of 'a', 'b', 'c', or 'z'.
`[a-z|g]`	Any lower-case letter other than 'g'.
`[agm|aefghijklmo]`	Doesn't match anything (the empty set).
`[abc|def]`	Any of 'a', 'b', or 'c'.

Given a list of lower-case letters (not necessarily in alphabetical order and possibly containing repetitions), you are to produce the *shortest* charset that matches exactly those characters. For example, given "`abdcp`", you should produce "`[a-dp|]`" and given "`fghiabcd`", you should produce "`[a-i|e]`."

The input will be a sequence of strings of lower-case letters in free format. For each one, produce a line of output that echoes the input sequence, followed by the shortest charset (see the example). In the case of ties, any correct charset will do, as long as it has minimal length. Letters appearing to the left of the '`|`' must appear in alphabetical order, as must those to the right of the '`|`' (for example, '`[a-ghi|bd]`' and not '`[ha-gi|bd]`' or '`[a-ghi|db]`').

Example:

Input	Output
`azbdc abcdefhijklmnopqrstuvwxyz`	`azbdc [a-dz|]`
`abdfhghi`	`abcdefhijklmnopqrstuvwxyz [a-z|g]`
	`abdfhghi [a-i|ce]`

6. These days, it seems that we expect computers to direct our social lives, even to the point of choosing our friends. Of course, as programmers, we get to implement these choice algorithms, as in this problem.

Given a "friend relation"—a set of pairs of people who designate each other as "friends"—and a particular person (C, "the client") your algorithm is to select any specified number (N) of people who are *not* friends of C, but who are friends of friends of C. Specifically, the algorithm picks those people who know the greatest numbers of C's friends. For example, if $N = 2$, C is Jack, and the friend relation is

```
Jack/Mary     Mary/Claire     Jack/Tom     Tom/Claire    Jack/Richard
Tod/Richard   Claire/Richard  Jack/Jill    Jill/June     June/Tod
Jill/Tod      Jill/Peter      Mary/Tom     Richard/Tom   Jill/Tom
```

the program would select Claire (who knows Jack's friends Mary, Tom, and Richard) and Tod (who knows Richard and Jill). Jack's other friends of friends are June and Peter (who both know Jill). The relationships between Jack's friends (such as Mary and Tom) are irrelevant. Implicitly, Jack is his own friend, so that he is not in the list of suggestions. The "friend" relation is symmetric: you are my friend iff I am yours.

The input will consist of multiple sets of data in free format. Each set begins with an integer, N, and a name (consisting of a string of non-whitespace characters), C. There then follows a sequence of an even number of names, each successive pair of which denotes a friend relationship. A given friend relationship will appear only one in the sequence (with either member first). The implicit friendship of a person with himself will not be included. The sequence terminates with two asterisks (separated by whitespace).

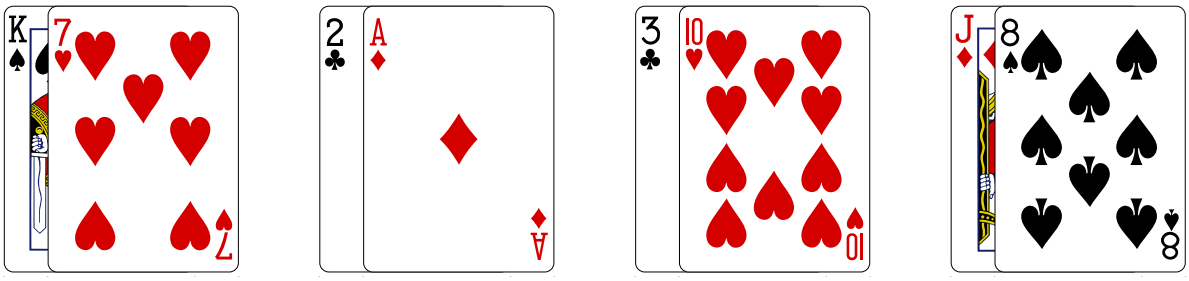
For each input set, the output consists of a list of whitespace-separated names in alphabetical order giving the N non-friends of C who are acquainted with the most friends of C. You may assume that there will always be at least N non-friends of C. In case too many people know the smallest qualifying number of friends, choose those that come earlier in alphabetical order. Thus, if $N = 2$ and Mike knows two of C's friends while Sam, Jill, and Mary know one, then choose Jill and Mike.

Example:

Input	Output
2 Jack Mary Claire Jack Tom Tom Claire Jack Richard Jack Mary Tod Richard Claire Richard Jack Jill Jill June June Tod Jill Tod Jill Peter Mary Tom Richard Tom Jill Tom * * 3 Jack Mary Claire Jack Tom Tom Claire Jack Richard Jack Mary Tod Richard Claire Richard Jack Jill Jill June June Tod Jill Tod Jill Peter Mary Tom Richard Tom Jill Tom * *	Claire Tod Claire June Tod

7. In this problem, you are to compute the maximum score that a player can achieve in a simple two-player card game, assuming the opponent plays perfectly. Four piles are dealt face up, so that all cards are visible. The players take turns removing one card from the top of one of the piles. When all the cards are taken, the winner is the player whose cards' values have the highest sum. The card values are 1 for ace through 13 for king (suits are irrelevant).

For example, starting from the position shown here, the player who moves first can collect no more than 26 out of 55 points. He gets the 10♡, but must eventually expose the K♠ and J♢ to the second player.



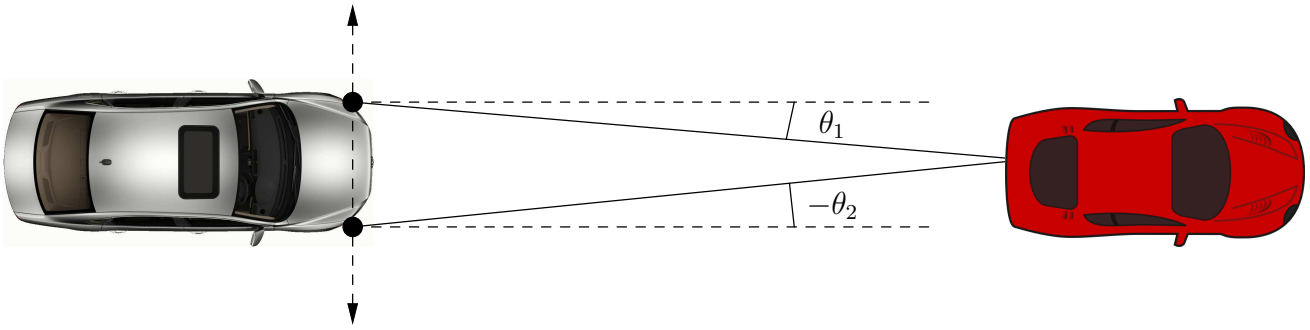
The input will consist of a sequence of one or more deals in free format. Each deal consists of four equal-length strings containing letters A, T (10), J, Q, K, and 2–9 (that is, suits are not included). There are as many as four of each rank of card, as in a normal deck, so the total number of cards in any pile can never be more than 13. The last character in each string denotes the top card.

For each deal, the output contains a line reporting the number of the deal, the score that the first player can achieve with perfect play, and the total points available to be won. See the example for the proper format.

Example:

Input	Output
`K7 2A 3T J8`	`Deal 1: First player wins 26 out of 55`
`K Q J T`	`Deal 2: First player wins 24 out of 46`
`A2 A2 A2 A2`	`Deal 3: First player wins 6 out of 12`
`2A 2A 2A 2A`	`Deal 4: First player wins 4 out of 12`

8. A car has an optical collision-detection device mounted in the vicinity of its headlights—one sensor on the left and one on the right. One of the electronics modules is able to pick out particular points on objects in front of the car and to match up points seen by the left sensor with the images of the same points as seen by the right, and report the angular positions of these points with respect to the two sensors, as illustrated in this diagram:



The two sensors, represented by black dots on the left car, report the angles between straight ahead (the dashed lines going to the right) and some point on the right car. The angles are measured clockwise from straight, so that in this example, the angle for the right sensor would be negative. By taking these readings at intervals of a second, it is possible to establish the position of the point being tracked and its velocity relative to the sensors.

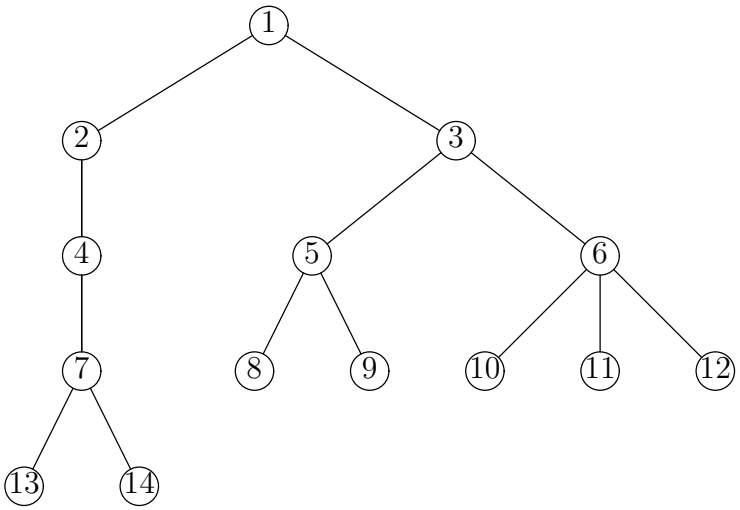
Write a program that, given two sets of sensor readings, reports where and when the point being tracked will intersect the line containing the two sensors (shown as a vertical dashed double arrow in the diagram). The input will consist of one or more sets of data. Each set will contain five floating-point numbers. The first, w, will be the distance between the two sensors. The next two will be θ_1 and θ_2 at time 0, and the final two will be θ_1 and θ_2 at time 1. Angles are in degrees clockwise (i.e., to the right) of straight ahead. You may assume that for each reading, $-90 < \theta_2 < \theta_1 < 90$.

For each input set, output the (extrapolated) time and position at which the point being tracked would intersect the line between the sensors, taking the left sensor's position as 0, and the right sensor's position as w. If there is no future intersection, output "Not approaching". Use the format shown in the examples.

Example:

Input	Output
`2 10 -10 20 -20`	`Intersects at x=1.00, t=1.94`
`2 10 -10 5 -20`	`Intersects at x=-1.80, t=4.57`
`2 10 -10 5 -5`	`Not approaching`

9. There are any number of programs or libraries that will figure out how to display the nodes of a tree on a screen or page in some relatively tidy fashion. Here, you are to contribute still one more. Given a tree, you are to choose the horizontal coordinates of its nodes (the vertical coordinates will simply be determined by the depths of the nodes, and you'll never need to consider them). The sort of layout we're looking for is illustrated by the example below:



Specifically, we lay out the nodes horizontally so that:

- Any inner node is positioned half way between its leftmost and rightmost (immediate) child.
- For any node, n, its descendants are at least one unit to the left of the descendants of any right sibling of n and at least one unit to the right of the descendants of any left sibling of n.
- Subject to the above constraints, all nodes are as close together horizontally as possible.
- The leftmost node has horizontal position 0.

For our purposes, the *descendants* of n include n itself.

The input to your program will consist of one or more trees described in a prefix notation (in free format):

$$n \quad L \quad C_1 \quad C_2 \quad \cdots \quad C_n$$

where $n \geq 0$ is an integer, L is a unique numeric label and the C_i represent the children of the node in the same notation. For example, the tree in the example could be written like this (with indentation added to show tree structure):

```
2 1
  1 2
    1 4
      2 7
        0 13 0 14
  2 3
    2 5
      0 8 0 9
    3 6
      0 10 0 11 0 12
```

For each input tree, the output of your program will give a sequence number followed by a listing of the node labels in the same order they appear in the input, each followed by its horizontal position, rounded to four decimal places. Use the format shown in the example.

Example:

Input	Output

`2 1  1 2  1 4  2 7 0 13 0 14`
`  2 3 2 5 0 8 0 9`
`    3 6 0 10 0 11 0 12`
`0 5`
	`Tree 1:`
	`  1 2.1250`
	`  2 0.5000`
	`  4 0.5000`
	`  7 0.5000`
	`  13 0.0000`
	`  14 1.0000`
	`  3 3.7500`
	`  5 2.5000`
	`  8 2.0000`
	`  9 3.0000`
	`  6 5.0000`
	`  10 4.0000`
	`  11 5.0000`
	`  12 6.0000`
	`Tree 2:`
	`  5 0.0000`