# 1 Problem A: The $3n+1$ problem

## Background

Problems in Computer Science are often classified as belonging to a certain class of problems (e.g., NP, Unsolvable, Recursive). In this problem you will be analyzing a property of an algorithm whose classification is not known for all possible inputs.

## The Problem

Consider the following algorithm:

1. input $n$
2. print $n$
3. if $n = 1$ then STOP
4. if $n$ is odd then $n \longleftarrow 3n+1$
5. else $n \longleftarrow n/2$
6. GOTO 2

Given the input 22, the following sequence of numbers will be printed

22 11 34 17 52 26 13 40 20 10 5 16 8 4 2 1

It is conjectured that the algorithm above will terminate (when a 1 is printed) for any integral input value. Despite the simplicity of the algorithm, it is unknown whether this conjecture is true. It has been verified, however, for all integers $n$ such that $0 < n < 1,000,000$ (and, in fact, for many more numbers than this.)

Given an input $n$, it is possible to determine the number of numbers printed before and including the 1 is printed. For a given $n$ this is called the *cycle-length* of $n$. In the example above, the cycle length of 22 is 16.

For any two numbers $i$ and $j$ you are to determine the maximum cycle length over all numbers between and including both $i$ and $j$.

## The Input

The input will consist of a series of pairs of integers $i$ and $j$, one pair of integers per line. All integers will be less than 10,000 and greater than 0.

You should process all pairs of integers and for each pair determine the maximum cycle length over all integers between and including $i$ and $j$.

### The Output

For each pair of input integers $i$ and $j$ you should output $i$, $j$, and the maximum cycle length for integers between and including $i$ and $j$. These three numbers should be separated by one space with all three numbers on one line and with one line of output for each line of input. The integers $i$ and $j$ must appear in the output in the same order in which they appeared in the input and should be followed by the maximum cycle length (on the same line).

### Sample Input

```
1 10
100 200
201 210
900 1000
```

### Sample Output

```
1 10 20
100 200 125
201 210 89
900 1000 174
```

# 2 Problem B: The Blocks Problem

## Background

Many areas of Computer Science use simple, abstract domains for both analytical and empirical studies. For example, an early AI study of planning and robotics (STRIPS) used a block world in which a robot arm performed tasks involving the manipulation of blocks.

In this problem you will model a simple block world under certain rules and constraints. Rather than determine how to achieve a specified state, you will "program" a robotic arm to respond to a limited set of commands.

## The Problem

The problem is to parse a series of commands that instruct a robot arm in how to manipulate blocks that lie on a flat table. Initially there are $n$ blocks on the table (numbered from 0 to $n-1$) with block $b_i$ adjacent to block $b_{i+1}$ for all $0 \leq i < n-1$ as shown in the diagram below:

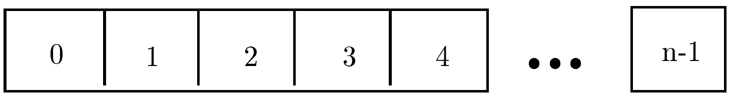



Figure 1: Initial Blocks World

The valid commands for the robot arm that manipulates blocks are:

- move $a$ onto $b$

  where $a$ and $b$ are block numbers, puts block $a$ onto block $b$ after returning any blocks that are stacked on top of blocks $a$ and $b$ to their initial positions.

- move $a$ over $b$

  where $a$ and $b$ are block numbers, puts block $a$ onto the top of the stack containing block $b$, after returning any blocks that are stacked on top of block $a$ to their initial positions.

- pile $a$ onto $b$

  where $a$ and $b$ are block numbers, moves the pile of blocks consisting of block $a$, and any blocks that are stacked above block $a$, onto block $b$. All blocks on top of block $b$ are moved to their initial positions prior to the pile taking place. The blocks stacked above block $a$ retain their order when moved.

- pile $a$ over $b$

  where $a$ and $b$ are block numbers, puts the pile of blocks consisting of block $a$, and any blocks that are stacked above block $a$, onto the top of the stack containing block $b$. The blocks stacked above block $a$ retain their original order when moved.

- quit

  terminates manipulations in the block world.

Any command in which $a = b$ or in which $a$ and $b$ are in the same stack of blocks is an illegal command. All illegal commands should be ignored and should have no affect on the configuration of blocks.

### The Input

The input begins with an integer $n$ on a line by itself representing the number of blocks in the block world. You may assume that $0 < n < 25$.

The number of blocks is followed by a sequence of block commands, one command per line. Your program should process all commands until the `quit` command is encountered.

You may assume that all commands will be of the form specified above. There will be no syntactically incorrect commands.

### The Output

The output should consist of the final state of the blocks world. Each original block position numbered $i$ ($0 \leq i < n$ where $n$ is the number of blocks) should appear followed immediately by a colon, followed by one space, followed by a list of blocks that appear stacked in that position with each block number separated from other block numbers by a space. There should be one line of output for each block position (i.e., $n$ lines of output where $n$ is the integer on the first line of input).

### Sample Input

```
10
move 9 onto 1
move 8 over 1
move 7 over 1
move 6 over 1
pile 8 over 6
pile 8 over 5
move 2 over 1
move 4 over 9
quit
```

### Sample Output

```
0: 0
1: 1 9 2 4
2:
3: 3
4:
5: 5 8 7 6
6:
7:
8:
9:
```

# 3 Problem C: Ecological Bin Packing

### Background

Bin packing, or the placement of objects of certain weights into different bins subject to certain constraints, is an historically interesting problem. Some bin packing problems are NP-complete but are amenable to dynamic programming solutions or to approximately optimal heuristic solutions.

In this problem you will be solving a bin packing problem that deals with recycling glass.

### The Problem

Recycling glass requires that the glass be separated by color into one of three categories: brown glass, green glass, and clear glass. In this problem you will be given three recycling bins, each containing a specified number of brown, green and clear bottles. In order to be recycled, the bottles will need to be moved so that each bin contains bottles of only one color.

The problem is to minimize the number of bottles that are moved. You may assume that the only problem is to minimize the number of movements between boxes.

For the purposes of this problem, each bin has infinite capacity and the only constraint is moving the bottles so that each bin contains bottles of a single color.

### The Input

The input consists of a series of lines with each line containing 9 integers. The first three integers on a line represent the number of brown, green, and clear bottles (respectively) in bin number 1, the second three represent the number of brown, green and clear bottles (respectively) in bin number 2, and the last three integers represent the number of brown, green, and clear bottles (respectively) in bin number 3. For example, the line

10 15 20 30 12 8 15 8 31

indicates that there are 20 clear bottles in bin 1, 12 green bottles in bin 2, and 15 brown bottles in bin 3.

Integers on a line will be separated by one or more spaces. Your program should process all lines in the input file.

## The Output

For each line of input there will be one line of output indicating what color bottles go in what bin to minimize the number of bottle movements. You should also print the minimum number of bottle movements.

The output should consist of a string of the three upper case characters 'G', 'B', 'C' (representing the colors green, brown, and clear) representing the color associated with each bin.

The first character of the string represents the color associated with the first bin, the second character of the string represents the color associated with the second bin, and the third character represents the color associated with the third bin.

The integer indicating the minimum number of bottle movements should follow the string, with one space separating it from the string.

If more than one order of brown, green, and clear bins yields the minimum number of movements then the alphabetically first string representing a minimal configuration should be printed.

## Sample Input

```
1 2 3 4 5 6 7 8 9
5 10 5 20 10 5 10 20 10
```

## Sample Output

```
BCG 30
CBG 50
```

# 4 Problem D: Stacking Boxes

## Background

Some concepts in Mathematics and Computer Science are simple in one or two dimensions but become more complex when extended to arbitrary dimensions. Consider solving differential equations in several dimensions and analyzing the topology of an $n$-dimensional hypercube. The former is much more complicated than its one dimensional relative while the latter bears a remarkable resemblance to its "lower-class" cousin.

## The Problem

Consider an $n$-dimensional "box" given by its dimensions. In two dimensions the box (2,3) might represent a box with length 2 units and width 3 units. In three dimensions the box (4,8,9) can represent a box $4 \times 8 \times 9$ (length, width, and height). In 6 dimensions it is, perhaps, unclear what the box (4,5,6,7,8,9) represents; but we can analyze properties of the box such as the sum of its dimensions.

In this problem you will analyze a property of a group of $n$-dimensional boxes. You are to determine the longest *nesting string* of boxes, that is a sequence of boxes $b_1, b_2, \ldots, b_k$ such that each box $b_i$ nests in box $b_{i+1}$ $(1 \leq i < k)$.

A box D = $(d_1, d_2, \ldots, d_n)$ nests in a box E = $(e_1, e_2, \ldots, e_n)$ if there is some rearrangement of the $d_i$ such that when rearranged each dimension is less than the corresponding dimension in box E. This loosely corresponds to turning box D to see if it will fit in box E. However, since any rearrangement suffices, box D can be contorted, not just turned (see examples below).

For example, the box D = (2,6) nests in the box E = (7,3) since D can be rearranged as (6,2) so that each dimension is less than the corresponding dimension in E. The box D = (9,5,7,3) does NOT nest in the box E = (2,10,6,8) since no rearrangement of D results in a box that satisfies the nesting property, but F = (9,5,7,1) does nest in box E since F can be rearranged as (1,9,5,7) which nests in E.

Formally, we define nesting as follows: box D = $(d_1, d_2, \ldots, d_n)$ *nests* in box E = $(e_1, e_2, \ldots, e_n)$ if there is a permutation $\pi$ of $1 \ldots n$ such that $(d_{\pi(1)}, d_{\pi(2)}, \ldots, d_{\pi(n)})$ "fits" in $(e_1, e_2, \ldots, e_n)$ i.e., if $d_{\pi(i)} < e_i$ for all $1 \leq i \leq n$.

## The Input

The input consists of a series of box sequences. Each box sequence begins with a line consisting of the the number of boxes $k$ in the sequence followed by the dimensionality of the boxes, $n$ (on the same line.)

This line is followed by $k$ lines, one line per box with the $n$ measurements of each box on one line separated by one or more spaces. The $i^{th}$ line in the sequence $(1 \leq i \leq k)$ gives the measurements for the $i^{th}$ box.

There may be several box sequences in the input file. Your program should process all of them and determine, for each sequence, which of the $k$ boxes determine the longest nesting string and the length of that nesting string (the number of boxes in the string).

In this problem the maximum dimensionality is 10 and the minimum dimensionality is 1. The maximum number of boxes in a sequence is 30.

### The Output

For each box sequence in the input file, output the length of the longest nesting string on one line followed on the next line by a list of the boxes that comprise this string in order. The "smallest" or "innermost" box of the nesting string should be listed first, the next box (if there is one) should be listed second, etc.

The boxes should be numbered according to the order in which they appeared in the input file (first box is box 1, etc.).

If there is more than one longest nesting string then any one of them can be output.

### Sample Input

```
5 2
3 7
8 10
5 2
9 11
21 18
8 6
5 2 20 1 30 10
23 15 7 9 11 3
40 50 34 24 14 4
9 10 11 12 13 14
31 4 18 8 27 17
44 32 13 19 41 19
1 2 3 4 5 6
80 37 47 18 21 9
```

### Sample Output

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

# 5 Problem E: Arbitrage

## Background

The use of computers in the finance industry has been marked with controversy lately as programmed trading — designed to take advantage of extremely small fluctuations in prices — has been outlawed at many Wall Street firms. The ethics of computer programming is a fledgling field with many thorny issues.

## The Problem

*Arbitrage* is the trading of one currency for another with the hopes of taking advantage of small differences in conversion rates among several currencies in order to achieve a profit. For example, if \$1.00 in U.S. currency buys 0.7 British pounds currency, £1 in British currency buys 9.5 French francs, and 1 French franc buys 0.16 in U.S. dollars, then an arbitrage trader can start with \$1.00 and earn $1 \times 0.7 \times 9.5 \times 0.16 = 1.064$ dollars thus earning a profit of 6.4 percent.

You will write a program that determines whether a sequence of currency exchanges can yield a profit as described above.

To result in successful arbitrage, a sequence of exchanges must begin and end with the same currency, but any starting currency may be considered.

## The Input

The input file consists of one or more conversion tables. You must solve the arbitrage problem for each of the tables in the input file.

Each table is preceded by an integer $n$ on a line by itself giving the dimensions of the table. The maximum dimension is 20; the minimum dimension is 2.

The table then follows in row major order but with the diagonal elements of the table missing (these are assumed to have value 1.0). Thus the first row of the table represents the conversion rates between country 1 and $n-1$ other countries, i.e., the amount of currency of country $i$ ($2 \leq i \leq n$) that can be purchased with one unit of the currency of country 1.

Thus each table consists of $n+1$ lines in the input file: 1 line containing $n$ and $n$ lines representing the conversion table.

## The Output

For each table in the input file you must determine whether a sequence of exchanges exists that results in a profit of more than 1 percent (0.01). If a sequence exists you must print the sequence of exchanges that results in a profit. If there is more than one sequence that results in a profit of more than 1 percent you must print a sequence of minimal length, i.e., one of the sequences that uses the fewest exchanges of currencies to yield a profit.

(continued)

Because the IRS (United States Internal Revenue Service) notices lengthy transaction sequences, all profiting sequences must consist of $n$ or fewer transactions where $n$ is the dimension of the table giving conversion rates. The sequence 1 2 1 represents two conversions.

If a profiting sequence exists you must print the sequence of exchanges that results in a profit. The sequence is printed as a sequence of integers with the integer $i$ representing the $i^{th}$ line of the conversion table (country $i$). The first integer in the sequence is the country from which the profiting sequence starts. This integer also ends the sequence.

If no profiting sequence of $n$ or fewer transactions exists, then the line

```
no arbitrage sequence exists
```

should be printed.

### Sample Input

```
3
1.2 .89
.88 5.1
1.1 0.15
4
3.1    0.0023    0.35
0.21   0.00353   8.13
200    180.559   10.339
2.11   0.089     0.06111
2
2.0
0.45
```

### Sample Output

```
1 2 1
1 4 3 1
no arbitrage sequence exists
```

## 6 Problem F: The Skyline Problem

### Background

With the advent of high speed graphics workstations, CAD (computer-aided design) and other areas (CAM, VLSI design) have made increasingly effective use of computers. One of the problems with drawing images is the elimination of hidden lines — lines obscured by other parts of a drawing.

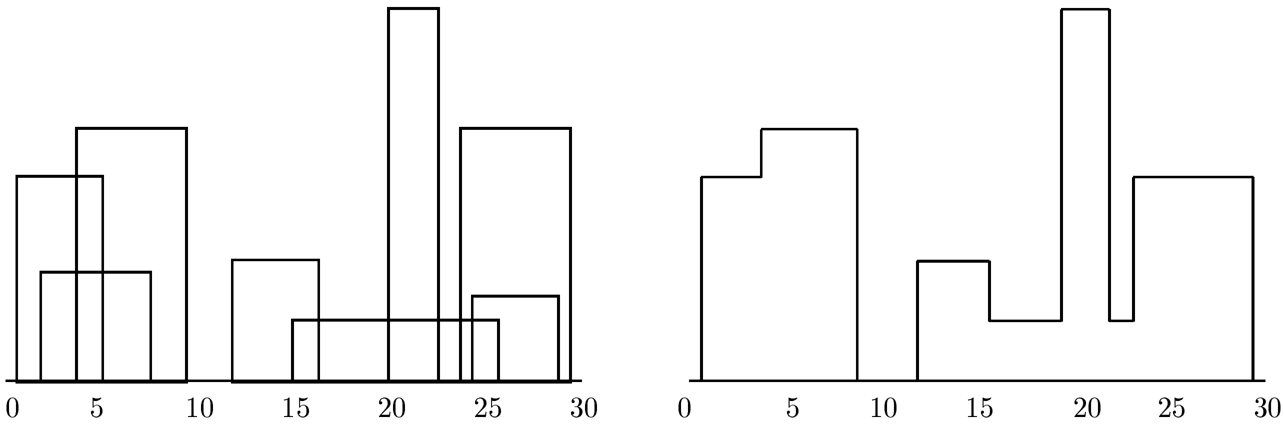
### The Problem

You are to design a program to assist an architect in drawing the skyline of a city given the locations of the buildings in the city. To make the problem tractable, all buildings are rectangular in shape and they share a common bottom (the city they are built in is very flat). The city is also viewed as two-dimensional. A building is specified by an ordered triple $(L_i, H_i, R_i)$ where $L_i$ and $R_i$ are left and right coordinates, respectively, of building $i$ and $H_i$ is the height of the building. In the diagram below buildings are shown on the left with triples

$$(1, 11, 5), (2, 6, 7), (3, 13, 9), (12, 7, 16), (14, 3, 25), (19, 18, 22), (23, 13, 29), (24, 4, 28)$$

the skyline, shown on the right, is represented by the sequence:

$$(1,\ 11,\ 3,\ 13,\ 9,\ 0,\ 12,\ 7,\ 16,\ 3,\ 19,\ 18,\ 22,\ 3,\ 23,\ 13,\ 29,\ 0)$$




### The Input

The input is a sequence of building triples. All coordinates of buildings are integers less than 10,000 and there will be at least one and at most 50 buildings in the input file. Each building triple is on a line by itself in the input file. All integers in a triple are separated by one or more spaces. The triples will be sorted by $L_i$, the left $x$-coordinate of the building, so the building with the smallest left $x$-coordinate is first in the input file.

### The Output

The output should consist of the vector that describes the skyline as shown in the example above. In the skyline vector $(v_1, v_2, v_3, \ldots, v_{n-2}, v_{n-1}, v_n)$, the $v_i$ such that $i$ is an even number represent a horizontal line (height). The $v_i$ such that $i$ is an odd number represent a vertical line ($x$-coordinate). The skyline vector should represent the "path" taken, for example, by a bug starting at the minimum $x$-coordinate and traveling horizontally and vertically over all the lines that define the skyline. Thus the last entry in all skyline vectors will be a 0.

### Sample Input

```
1 11 5
2 6 7
3 13 9
12 7 16
14 3 25
19 18 22
23 13 29
24 4 28
```

### Sample Output

```
1 11 3 13 9 0 12 7 16 3 19 18 22 3 23 13 29 0
```