ICC4101 - Algorithms and Competitive Programing

Javier Correa

Remember!

- Submit your code with the correct filename:
  - `01_CORREA_JAVIER.py` is correct.
  - `1_CORREA_JAVIER.py` is also correct.
  - `1_JAVIER_CORREA.py` is also correct.
  - `P1.py` is **incorrect**.
  - `Problema1_my_name.py` is **incorrect**.

- Comment your `open` statement!

```python
from sys import stdin
#stdin = open(...) # <- Comment this line before submitting

import io
#stdin = io.StringIO("...") # <- Comment this line before
submitting
```

- Do not prompt for data with the `input` statement

```python
input("Type something") # <- Output will be different from the
expected!
input()# <- Ok!
```

- Similarly, if not explicitly stated, do not print extra stuff!

```python
print(f"The result is {result}") # <- Output maybe different from
the expected
```

Previous problems

1. Formula 1

Straight forward solution, just follow the instructions!

```
def main():
    ## Read G and P and keep executing while G != 0
    while True:
        G, P = map(int, infile.readline().split())
        if G == 0:
            break

## Read the results
    results = []
    for _ in range(G):
        results.append(list(map(int, infile.readline().split())))

## Read scores
    S = int(infile.readline())
    for _ in range(S):
        numbers = list(map(int, infile.readline().split()))
        scoring.append(numbers[1:])
```

2. Climbing worm

Simulate the process of the worm. The only difficulty was to remember to check if you reach the top of the container before simulating the resting of the worm.

```
from sys import stdin

while True:
    n, u, d = map(int, stdin.readline.split())
    if n == 0:
        break
    steps = 0
    while n > 0:
        n += (-u) if steps % 2 == 0 else d
        step += 1
    print(steps)
```

3. Ecological bin packaging

Identify 6 configurations of interest and calculate their values.

```
from sys import stdin

names = ("BCG", "BGC", "GBC", "GCB", "CBG", "CGB")
for line in stdin.readlines():  # for line in stdin:
    b1, g1, c1, b2, g2, c2, b3, g3, c3 = map(int,
stdin.readline.split())
    BCG = b2 + b3 + c1 + c3 + g1 + g2
    BGC = b2 + b3 + g1 + g3 + c1 + c2
    GBC = g2 + g3 + b1 + b3 + c1 + c2
    GCB = g2 + g3 + c1 + c3 + b1 + b2
    CBG = c2 + c3 + b1 + b3 + g1 + g2
    CGB = c2 + c3 + g1 + g3 + b1 + b2
    values = (BCG, BGC, GBC, GCB, CBG, CGB)
    c, v = min(zip(names, values), key=lambda x[1])
    print(f"{c} {v}")
```

4. No Bariner

Check which number is greater.

```
from sys import stdin

N = int(stdin.readline())
for line in stdin:  # for line in stdin.readlines():
    a, b = map(int, stdin.readline.split())
    if a >= b:
        print("MMM BRAINS")
    else:
        print("NO BRAINS")
```

More Python

This week we will practice more python, specifucally, the usage of the built-in data structures.

Lists

The full documentation can be found at https://docs.python.org/3/library/stdtypes.html#sequence-types-list-tuple-range.

List in Python are linked lists.

Accessing elements in the list is $O(1)$ (Python implements them as an array of pointers to the actual object).

Accesing elements is achived by the `[]` operator:

- `A[4]` access the 5th element
- `A[-1]` access the last element
- `A[-2]` access the second to last element

- `A[:3]` is a sublist with the elements from the first until the third element (elements at indices 0, 1, 2)
- `A[1:3]` is a sublist with the second and third elements (elements at index 1 and 2)
- `A[:-1]` is a sublist with all the elements of A except the last element.
- `A[1:6:2]` is a sublist with elements starting from index 1 to indices less than 6 in steps of 2 (elements 1, 3, 5)

Other operators

- `len(A)` number of elements of A.
- `A.append(x)` appends element `x` to the list.
- `A.insert(x, n)` inserts element `x` at index `n`.
- (`A` and `B` are lists) `A + B` is a new list of concatenated elements of `A` and `B`.
- (`n` a number and `x` an element) `n*[x]` a list with the element `x` repeated `n` times.

- `x in A` / `x not in A` check if element `x` is in/is not in `A`.
- `min(A)`, `max(A)` finds the minimum/maximum element in A
- `A.count(x)` count the ocurrences of element `x` in `A`
- `sorted(A)` a list with the elements of A sorted incrementally.

You can pass a `key` parameter to `min/max/sorted` with a function which calculates a value by which compare element in the list. For example:

In [13]:
```
A = [(1,1), (1, 2), (2, 0), (3, 1)]
min(A, key=lambda x: x[1])
```

Out[13]:
```
(2, 0)
```

In [14]:
```
sorted(A, key=lambda x: sum(x))
```

Out[14]:
```
[(1, 1), (2, 0), (1, 2), (3, 1)]
```

Note!

- Checking if an item is in a list has a time complexity of $O(n)$. If your program relies on this check and it's executed several times, consider using `set` or `dict`.
- Elements of a list are references to the real object. This could cause problem in scenarios like this:

```
A = [[]]*3
A[0].append(1)
print(A)
```

```
[[1], [1], [1]]
```

If you want to create a list with three independent lists as elements you should do:

```
A = [[] for _ in range(3)]
```

Set

Full documentation at https://docs.python.org/3/library/stdtypes.html#set-types-set-frozenset.

Sets in Python are unordered containers with unique elements (equivalent to mathematical sets). Since sets contains no repeated elements, only inmutable objects are storable within a set.

Seta can be initialized with the constructor `set`:

```
A = set([1, 2, 3])
```
or with the `{}` syntax:

```
A = {1, 2, 3}
```

Sets cannot contain mutable elements. If we try to create a set with lists `[1]` and `[2]` as elements, it fails:

In [18]:
```
set([[1], [2]])
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Cell In[18], line 1
----> 1 set([[1], [2]])

TypeError: unhashable type: 'list'
```

Operations with sets:

- `x in A` check if element `x` is in set `A` (with time complexity $O(1)$)
- `A.add(x)` add element `x` to set `A` (inplace)
- `A < B` check if every element of A is in B and A is not equal to B.

- `A | B` union of sets A and B
- `A & B` intersection of sets A and B
- `A - B` difference of sets A and B
- `A ^ B` proper differencet of sets A and B (elements in A or B but not in the intersection)

There are also update operators `|=`, `&=` and `^=`.

Mappings (dict)

Full documentation at https://docs.python.org/3/library/stdtypes.html#mapping-types-dict

A `dict` is a map between keys and values. The only requirements is for keys to be inmutable objects.

Operations with mappings:

- `A[key]` return the element associated with the `key`
- `key in A` checks if the `key` is present in the dict object
- `A.get(key, default)` similar to `A[key]` but if the key is not found, the `default` value is returned.

A useful variation of a dict are `defaultdict`, found in the collections module.

```
from collections import defaultdict

A = defaultdict(list)
A[1].append(1)
```

`defaultdict` when the key is not present, it will call the given function to create a new element and associated with the key.

Roughly equivalent to:

```
A = dict()

if key not in A:
    A[key] = list()

A[1].append(1)
```

`dict` can be used to simulate/implement matrices or multidimensional arrays (specially if they are sparse):

```
maze = defaultdict(lambda: '#')

maze[1,1] = "."
maze[1,2] = "."
...
```

In this case the keys are tuples.

Problem 1. Map maker

`https://www.udebug.com/UVa/394`

The Cybersoft Computer Company (a leader in programming languages) has hired you to work on a new programming language named A--. Your task is to work on the array mapping tasks of the language. You will take an array reference such as x[5,6] and map it to an actual physical address. In preparation for doing this, you will write a program that will read in several array declarations and references and give the physical address of each reference. The physical address output by the program should be an integer number in base 10. The physical address of an array reference A[i[1] , i[2] , ... , i[D]] is calculated from the formula C[0] + C[1]i[1] + C[2]i[2] + ⋯ + C[D]i[D], where the constants C[0] ... C[D] are calculated as specified below.

```
B = Base address of the array
D = Number of dimensions in the array
Ld = Lower bound of dimension d
Ud = Upper bound of dimension d
CD = Array element size in bytes
Cd = C[d + 1](U[d+1] −L[d+1] +1) for 1 ≤ d < D
C0 = B−C[1]L[1]−C[2]L[2]−···−C[D]L[D]
```

Input

The first line of the input file contains two positive integers. The first integer specifies N, the number of arrays defined in the data file, and the second integer specifies R, the number of array references for which addresses should be calculated. The next N lines each define an array, one per line, and the following R lines contain one array reference per line for which an address should be calculated.

Each line which defines an array contains, in the following order, the name of the array (which is limited to 10 characters), a positive integer which specifies the base address of the array, a positive integer which specifies the size in bytes of each array element, and D, the number of dimensions in the array (no array will have fewer than 1 or more than 10 dimensions). This is followed on the same line by D pairs of integers which represent the lower and upper bounds, respectively, of dimensions 1 . . . D of the array.

Each line which specifies an array reference contains the name of the array followed by the integer indexes i1, i2, ... , iD where D is the dimension of the array.

Output

The output file should contain the array references and the physical addresses. There should be one array reference and physical address per line. The formatting guidelines below must be adhered to.

For each line of output:

1. Output the name of the array
2. Output a left square bracket
3. Output each index value (each pair of indexes should have a single comma and space between them)
4. Output a right square bracket, a space, an equal sign, and another space
5. Output the physical address

Sample Input

```
3 4
ONE 1500 2 2 0 3 1 5
TWO 2000 4 3 1 4 0 5 5 10
THREE 3000 1 1 1 9
ONE 24
THREE 7
TWO 206
TWO 339
```

Sample Output

```
ONE[2, 4] = 1526
THREE[7] = 3006
TWO[2, 0, 6] = 2148
TWO[3, 3, 9] = 2376
```

Problem 2. Machined surface

`https://www.udebug.com/UVa/414`

An imaging device furnishes digital images of two machined surfaces that eventually will be assembled in contact with each other. The roughness of this final contact is to be estimated.

A digital image is composed of the two characters, 'X' and '' (space). There are always 25 columns to an image, but the number of rows, N, is variable. Column one (1) will always have an 'X' in it and will be part of the left surface. The left surface can extend to the right from column one (1) as contiguous 'X's.

Similarly, column 25 will always have an 'X' in it and will be part of the right surface. The right surface can extend to the left from column 25 as contiguous 'X's.

Digital-Image View of Surfaces

```
Left                Right
XXXX                XXXXX ← 1
XXX               XXXXXXX
XXXXX                XXXX
XX                 XXXXXX
           ⋮ ⋮
XXXX                 XXXX
XXX                XXXXXX ← N
↑                       ↑
1                      25
```

In each row of the image, there can be zero or more space characters separating the left surface from the right surface. There will never be more than a single blank region in any row.

For each image given, you are to determine the total "void" that will exist after the left surface has been brought into contact with the right surface. The "void" is the total count of the spaces that remains between the left and right surfaces after theyhave been brought into contact.

The two surfaces are brought into contact by displacing them strictly horizontally towards each other until a rightmost 'X' of the left surface of some row is immediately to

the left of the leftmost ‘X’ of the right surface of that row. There is no rotation or twisting of these two surfaces as they are brought into contact; they remain rigid, and only move horizontally.

Note: The original image may show the two surfaces already in contact, in which case no displacement enters into the contact roughness estimation.

Input

The input consists of a series of digital images. Each image data set has the following format:

First line – A single unsigned integer, N, with value greater than zero (0) and less than 13. The first digit of N will be the first character on a line.

Next N lines – Each line has exactly 25 characters; one or more ‘X’s, then zero or more spaces, then one or more ‘X’s.

The end of data is signaled by a null data set having a zero on the first line of an image data set and no further data.

Output

For each image you receive as a data set, you are to reply with the total void (count of spaces remaining after the surfaces are brought into contact). Use the default output for a single integer on a line.

Note: In the example input file below, the spaces have been replaced with the character ‘B’ for ease of reading. **The actual input file will use the ASCII-space character, not ‘B’.**

Sample Input

```
4
XXXXBBBBBBBBBBBBBBBXXXXX
XXXBBBBBBBBBBBBBBBXXXXXXX
XXXXXBBBBBBBBBBBBBBBXXXX
XXBBBBBBBBBBBBBBBBXXXXXX
2
XXXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXXX
1
XXXXXXXXXBBBBBBBBBBBBBXX
0
```

Sample Output

4
0
0

Problem 3. Permutation Arrays

In many computer problems, it is necessary to permute data arrays. That is, the data in an array must be re-arranged in some specified order. One way to permute arbitrary data arrays is to specify the permutations with an index array to point out the position of the elements in the new array. Let x be an array that is to be permuted and let x′ be the permuted array. Then, we have the relationship between x and x′ that x′[p[i]] = x[i].

Input

The input begins with a single positive integer on a line by itself indicating the number of the cases following, each of them as described below. This line is followed by a blank line, and there is also a blank line between two consecutive inputs.

Each input set will contain two lines of numbers. The first line will be an index array p containing the integers 1 ... n, where n is the number of integers in the list. The numbers in the first line will have been permuted in some fashion. The second line will contain a list numbers in floating point format.

Output

For each test case, the output must follow the description below. The outputs of two consecutive cases will be separated by a blank line.

The output for this program will be the list of floating point numbers from the input set, ordered according to the permutation array from the input file. The output numbers must be printed one per line in the same format in which they each appeared in the input file.

Sample Input

```
1

3 1 2
32.0 54.7 -2
```

Sample Output

```
54.7
-2
32.0
```

Problem 4. Brothers

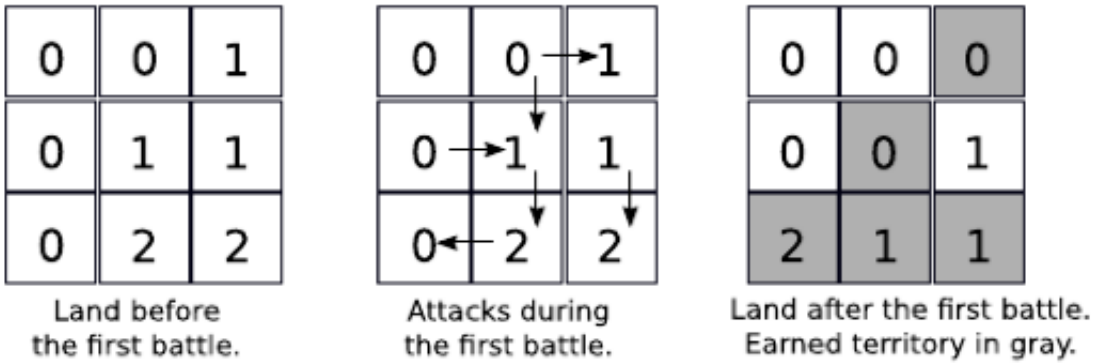
`https://www.udebug.com/UVa/12187`

In the land of ACM ruled a great King who became obsessed with order. The kingdom had a rectangular form, and the King divided the territory into a grid of small rectangular counties. Before dying, the King distributed the counties among his sons.

However, he was unaware that his children had developed a strange rivalry: the first heir hated the second heir, but not the rest; the second heir hated the third heir, but not the rest, and so on . . . Finally, the last heir hated the first heir, but not the other heirs.

As soon as the King died, the strange rivalry among the King's sons sparked off a generalized war in the kingdom. Attacks only took place between pairs of adjacent counties (adjacent counties are those that share one vertical or horizontal border). A county X attacked an adjacent county Y whenever the owner of X hated the owner of Y . The attacked county was always conquered by the attacking brother. By a rule of honor all the attacks were carried out simultaneously, and a set of simultaneous attacks was called a battle. After a certain number of battles, the surviving sons made a truce and never battled again.

For example, if the King had three sons, named 0, 1 and 2, the figure below shows what happens in the first battle for a given initial land distribution:



You were hired to help an ACM historian determining, given the number of heirs, the initial land distribution and the number of battles, what was the land distribution after all battles.

Input

The input contains several test cases. The first line of a test case contains four integers N, R, C and K, separated by single spaces. N is the number of heirs ($2 \le N \le 100$), R and C are the dimensions of the kingdom ($2 \le R, C \le 100$), and K is the number of battles ($1 \le K \le 100$). Heirs are identified by sequential integers starting from zero (0 is the first heir, 1 is the second heir, . . ., N − 1 is the last heir). Each of the next R lines contains C integers Hr,c separated by single spaces, representing initial land distribution: Hr,c is the

initial owner of the county in row r and column c ($0 \leq Hr,c \leq N - 1$).

The last test case is followed by a line containing four zeroes separated by single spaces.

Output

For each test case, your program must print R lines with C integers each, separated by single spaces in the same format as the input, representing the land distribution after all battles.

Sample Input

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

Sample Output

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

Problem 5. Football sort

Write a program, that given the fixtures of a football championship, outputs the corresponding classi- fication following the format specified below. Win, draw and loss earn respectively three, one and zero points.

The criteria of classification are the number of points scored, followed by goal difference (goals scored minus goals suffered), and then scored goals. When more than one team have exactly the same number of points, goal difference, and scored goals, these are considered as having the same position in the classification.

Input

The input will consist of a series of tests. Each test starts with a line containing two positive integers 28 ≥ T ≥ 1 and G ≥ 0. T is the number of teams and G is the number of games played. Follow then T lines, each containing the name of a squad. Squad names have up to 15 characters, and may only contain letters and dash characters ('-').

Finally the following G lines contain the score of each game. The scores are output with the following format: name of home team, number of goals scored by home team, a dash, number of goals scored by away team, and name of away team.

The input ends with a test case where T = G = 0 and should not be processed.

Output

The program shall output the classification tables corresponding to each input test separated by blank lines. In each table, the teams appear in order of classification, or alphabetically when they have the same position. The statistics of each team are displayed on a single line: team position, team name, number of points, number of games played, number of scored goals, number of suffered goals, goal difference, and percentage of earned points, when available. Note that if several teams have are in a draw, only the position of the first is printed. Fields shall be formatted and aligned as shown in the sample output.

Sample Input

```
11 54
a
b
c
d
e
f
g
h
i
j
k
a 1 - 0 b
a 1 - 0 c
a 1 - 0 d
```

a 1 – 0 e
a 1 – 0 f
a 1 – 0 g
a 1 – 0 h
a 1 – 0 i
a 1 – 0 j
a 1 – 0 k
b 1 – 0 c
b 1 – 0 d
b 1 – 0 e
b 1 – 0 f
b 1 – 0 g
b 1 – 0 h
b 1 – 0 i
b 1 – 0 j
b 1 – 0 k
c 1 – 0 d
c 1 – 0 e
c 1 – 0 f
c 1 – 0 g
c 1 – 0 h
c 1 – 0 i
c 1 – 0 j
c 1 – 0 k
d 1 – 0 e
d 1 – 0 f
d 1 – 0 g
d 1 – 0 h
d 1 – 0 i
d 1 – 0 j
d 1 – 0 k
e 1 – 0 f
e 1 – 0 g
e 1 – 0 h
e 1 – 0 i
e 1 – 0 j
e 1 – 0 k
f 1 – 0 g
f 1 – 0 h
f 1 – 0 i
f 1 – 0 j
f 1 – 0 k
g 1 – 0 h
g 1 – 0 i
g 1 – 0 j
g 1 – 0 k
h 1 – 0 i
h 1 – 0 j
h 1 – 0 k
i 1 – 0 j

```
i 1 – 0 k
0 0
```

Sample Output

```
 1.              a  30  10  10   0  10 100.00
 2.              b  27  10   9   1   8  90.00
 3.              c  24  10   8   2   6  80.00
 4.              d  21  10   7   3   4  70.00
 5.              e  18  10   6   4   2  60.00
 6.              f  15  10   5   5   0  50.00
 7.              g  12  10   4   6  -2  40.00
 8.              h   9  10   3   7  -4  30.00
 9.              i   6  10   2   8  -6  20.00
10.              j   0   9   0   9  -9   0.00
                 k   0   9   0   9  -9   0.00
```

Problem 6. Open Source

At an open-source fair held at a major univer- sity, leaders of open-source projects put sign-up sheets on the wall, with the project name at the top in capital letters for identification.

Students then signed up for projects using their userids. A userid is a string of lower-case letters and numbers starting with a letter.

The organizer then took all the sheets off the wall and typed in the information. Your job is to summarize the number of students who have signed up for each project.

Some students were overly enthusiastic and put their name down several times for the same project. That's okay, but they should only count once. Students were asked to commit to a single project, so any student who has signed up for more than one project should not count for any project.

There are at most 10,000 students at the university, and at most 100 projects were advertised.

Input

The input contains several test cases, each one ending with a line that starts with the digit 1. The last test case is followed by a line starting with the digit 0.

Each test case consists of one or more project sheets. A project sheet consists of a line

containing the project name in capital letters, followed by the userids of students, one per line.

Output

For each test case, output a summary of each project sheet. The summary is one line with the name of the project followed by the number of students who signed up. These lines should be printed in decreasing order of number of signups. If two or more projects have the same number of signups, they should be listed in alphabetical order.

Sample Input

```
UBQTS TXT
tthumb
LIVESPACE BLOGJAM
philton
aeinstein
YOUBOOK
j97lee
sswxyzy
j97lee
aeinstein
SKINUX
1
0
```

Sample Output

```
YOUBOOK 2
LIVESPACE BLOGJAM 1
UBQTS TXT 1
SKINUX 0
```

In [ ]: