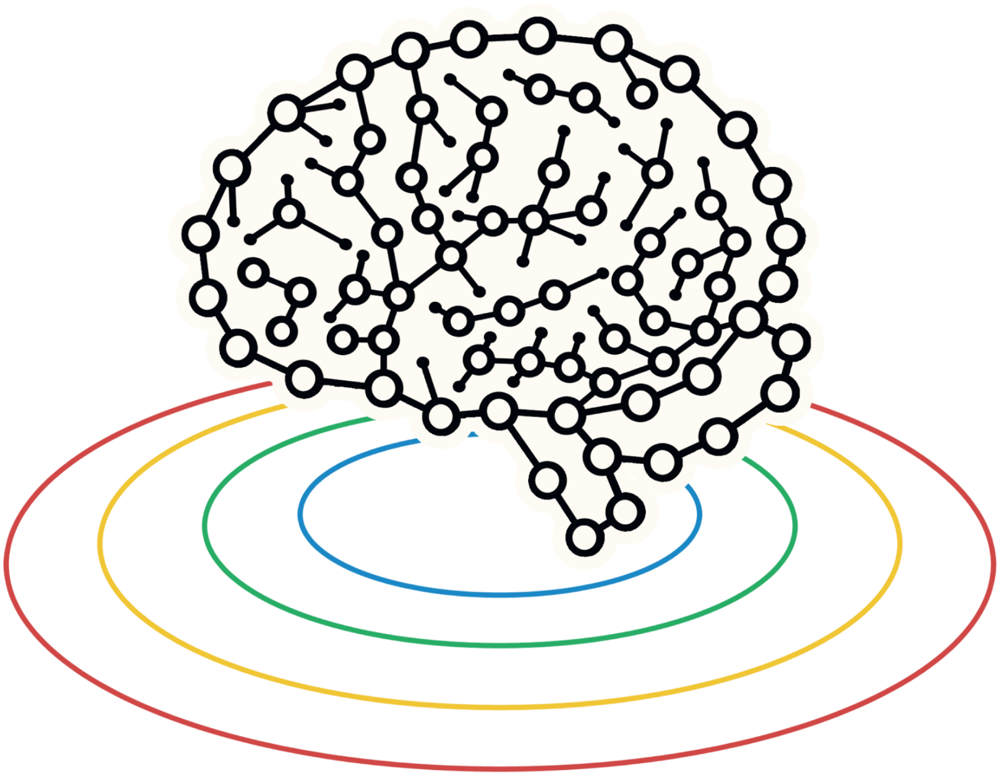
Association for Computing Machinery
University of the Philippines Diliman Student Chapter, Inc.



ALGOLYMPICS 2021

UP ACM PROGRAMMING COMPETITION

ELIMINATION ROUND EDITORIALS

Introduction

We hope you enjoyed the problems from the UP ACM Algolympics Online Elimination Round!

The problems in this round served to provide a friendly introduction to competitive programming newcomers, while still giving some interesting problems to returning participants. If you didn't qualify for the Online Final Round, you can still try the Finals problem set as it will also be available on Codeforces! And train harder for next year's Algolympics!

The problems will still be available on the Codeforces contest page for practice. They will be available forever.[1]

If you wish to discuss the problems (hints, solutions, etc.) with fellow participants and/or the Algolympics scientific committee members, a Discord server has been set up just for that. Please email algolympics@upacm.net with your name, team name and school to get an invite to the Algolympics Discord server.

Many members of the Algolympics scientific committee are also part of the National Olympiad in Informatics - Philippines (NOI.PH), an algorithmic competition aimed at high school students. We invite you to say hello in our Discord! Please email ask@noi.ph with your name and school if you wish to have access to the NOI.PH Discord server.

The NOI.ph Elimination Round usually runs at around the same time as the Algolympics Elimination Round. If you need more practice, you can try out the problem set used for the elimination round by joining the NOI.PH Codeforces group: Go to `https://codeforces.com/group/Sw3sdIlMPV/contests` and click on the "Join" button at the bottom-right panel of this page.

All past problems of official NOI.PH contests can be accessed through this link: `https://noi.ph/past-problems`.

Please regularly check your email, the Algolympics Discord and UP ACM's Facebook page for news regarding invitations to the On-Site Final Round and other potentially important announcements.

[1]where "forever" means "for as long as Codeforces exists" :P

Contents

Problem A: The Rice of Skywalker

Setters: Kevin Atienza

Testers: JD Dantes, Serge Rivera

Statement Authors: Payton Yao

Test Data Authors: Patrick Celon

Editorialists: JD Dantes

A.1 Solution

In this first problem, we are given two integers which we must compare to print the appropriate output depending on their ratio.

In Python, we can simply do:

```
r = int(input())
w = int(input())

ratio = r / w

if ratio < 1/3:
   print('WE NEED MORE RICE.')
elif ratio == 1/3:
   print('THE RICE IS RIGHT!')
else:
   print('WE NEED MORE WATER.')
```

A.2 Data Types

Note that Python gives us some conveniences, being a dynamically typed high-level programming language.

In statically typed languages like C++, we may end up writing code like:

```
int r, w;
cin >> r >> w;

double ratio = (double) r / (double) w;

if (ratio < 1 / 3) {
   printf("WE NEED MORE RICE.\n");
} else if (ratio == 1 / 3) {
   printf("THE RICE IS RIGHT!\n");
} else {
   printf("WE NEED MORE WATER.\n");
}
```

Can you spot any pitfalls from that snippet?

It's about the data types! For integers, division will still result to integers; remainders are ignored. This means that 1/3 will be *floored* to 0, leading to incorrect comparison.

A quick fix would be to typecast the number(s) first so that the division will treat the operands as floating points. Revising the earlier snippet, we have:

```
double ratio = (double) r / (double) w;

if (ratio < (double) 1 / 3) {
   printf("WE NEED MORE RICE.");
} else if (ratio == (double) 1 / 3) {
   printf("THE RICE IS RIGHT!");
} else {
   printf("WE NEED MORE WATER.");
}
```

A.3 Floating Points

Note that using floating points may not be advisable in some situations. This is because such data types have intrinsic limitations in how accurately they can represent numbers.

For example, try this in Python:

```
>>> 0.2 + 0.1
0.30000000000000004
>>> 0.2 + 0.1 == 0.3
False
```

As you can see, we expect $0.2 + 0.1$ to be exactly equal to 0.3, but due to the limitations, there's a small portion which causes it to be inaccurate, making the comparison fail.

For this problem, the requirements are not very sensitive to floating point precision, so we may be able to get away with division. For other problems or purposes, we may want to avoid division altogether, for example by only using integers and multiplication.

Tweaking our code from before, an implementaion could be:

```
if r * 3 < w:
   print('WE NEED MORE RICE.')
elif r *3 == w:
   print('THE RICE IS RIGHT!')
else:
   print('WE NEED MORE WATER.')
```

A.4 I/O

This problem also exposes the participant to the I/O format of programming contests. As mentioned in several places, input data are guaranteed to be within the specified constraints.

This means that your code will not receive input outside of those bounds, and thus, you do not need to do checks like:

```
if r < 1 or r > 10000:
   print('Please enter another number')
```

Also, generally speaking, all lines should end with a newline character ('\n'), even if there is only one line of output.

So even if there are no succeeding lines, do:

```
   printf("WE NEED MORE RICE.\n"); // (C++) Note the '\n' at the end.
```

Note that for Python, `print()` already appends the newline character by default.

As there were still some teams who seemed to have missed these, we reiterate the following recommendations here:

- Read the first few pages of the PDF version of the statements, labeled as "Important!".
- Check the announcements on the contest platform. The recommendation to read the PDF was mentioned there as well.
- Read the problem statement carefully.
- Be careful about unnecessary characters. *One* incorrect or missing punctuation, different capitalization, and leading or trailing whitespace may cause your submission to be judged as Wrong Answer.
- Sometimes, faster I/O routines may be needed to avoid running over the time limit. For C/C++, `printf()` and `scanf()` are recommended. For Java, there's `BufferedReader` and `PrintWriter`.
- All problems are solvable in C++ and Java. There are no such guarantees for Python as it may be significantly slower.

Problem B: The Pork Awakens

Setters: Kevin Atienza

Testers: JD Dantes

Statement Authors: Payton Yao

Test Data Authors: Patrick Celon

Editorialists: JD Dantes

B.1 Solution

For each serving, all of the ingredients must be used. Following this setup, we can loop through the servings and continuously count a complete meal as long as we still have budget for it.

Sample implementation:

```
n = int(input())
b = int(input())

grams = list(map(int, input().split()))
prices = list(map(int, input().split()))

made = 0

while True:
   serving_cost = 0
   for i in range(n):
      serving_cost += grams[i] * prices[i]

   if serving_cost <= b:
      b -= serving_cost
      made += 1
   else:
      break

print(made)
```

Note that the serving cost does not change every time we make a meal, so we can move it outside the `while` loop to save on time.

Furthermore, we can just divide the total budget with the calculated cost per serving. So the `while` loop can be disregarded altogether, and instead we can just calculate:

```
made = b // serving_cost
```

Note that in Python, two forward slashes indicate that we're treating the numbers as integers only, flooring the result. Using only one slash will yield fractional values, which we do not want.

Finally, if you're using Python, there are list comprehensions and built-in functions like `sum()` which we can use for convenience. So to compute the cost per serving, we can shorten the previous code to just:

```
serving_cost = sum([grams[i] * prices[i] for i in range(n)])
```

Of course, Python is slower than C++/Java, so you may need to switch to those for problems where your solution can't run within the time limit.

Problem C: Revenge of the C2

Setters: Kevin Atienza

Testers: JD Dantes

Statement Authors: Payton Yao

Test Data Authors: Patrick Celon

Editorialists: JD Dantes

C.1 Solution

As described in the statement, one size is better than another if it costs less than the other to buy some common reference amount of Cola.

In other words, we want to find the most cost-efficient size, comparing them in cost per unit of Cola.

So we can just take the ratio of cost to volume for each size, then pick the one with the minimum.

Here's a sample implementation:

```
cases = int(input())

for t in range(cases):
   volumes = list(map(int, input().split()))
   costs = list(map(int, input().split()))

   ratios = [costs[i]/volumes[i] for i in range(3)]

   for i in range(3):
      if ratios[i] == min(ratios):
         print(i+1)
         break
```

Problem D: Space Bar

Setters: JD Dantes

Testers: JD Dantes

Statement Authors: JD Dantes

Test Data Authors: Josh Quinto

Editorialists: JD Dantes

D.1 Solution

For this problem, we just need some way to easily check if characters are lowercase or uppercase. Programming languages usually have built-ins for such purposes. For example, Python has `isupper()` and `islower()`.

We can then use such functions in our implementation:

```
cases = int(input())

for t in range(cases):
   chars = input()
   flags = input()

   good = True

   for i in range(8):
      if chars[i].isupper():
         if i - 1 >= 0:
            if flags[i - 1] == '1':
               good = False
         if i + 1 < 8:
            if flags[i + 1] == '1':
               good = False

   if good:
      print('made some spaCe, welcome to the Bar!')
   else:
      print('not enough spaCe, please Bounce!')
```

D.2 ASCII Codes

Built-in functions are convenient and keep us from having to hardcode numerous `if-else` statements just to check if a character is capitalized, a letter from the alphabet, or even a digit.

But what if we're not aware of the above functions, or simply forgot what they're named? Fortunately, there are other ways to check.

For example, in Python, you can import the `string` module which has constants like `string.ascii_uppercase`.

You can then use the `in` operator to check:

```
>>> string.ascii_uppercase
'ABCDEFGHIJKLMNOPQRSTUVWXYZ'
>>> 'A' in string.ascii_uppercase
True
```

You can check the documentation for others like digits, punctuation, and even whitespace.

Furthermore, common characters actually map to some numbers. For example, search for "ASCII Table" or "ASCII Codes", or check sites like `http://www.asciitable.com/`. You'll find that 'A' corresponds to 65, 'a' maps to 97, and '0' maps to 48.

You don't need to memorize these either. You can use tools like Python's `ord()` and `chr()` to print out those ASCII mappings:

```
>>> ord('A')
65
>>> chr(65)
'A'
```

The nice thing about this is that since we have these mappings as numbers, we can use numerical operators to compare them! And if you noticed, the ASCII codes are also sensibly ordered, in the sense that 'B' maps to 66, 'C' maps to 67, and so on. Thus, we can do something like:

```
>>> def is_upper(c):
... return ord('A') <= ord(c) and ord(c) <= ord('Z')
...
>>> is_upper('S')
True
>>> is_upper('s')
False
```

This holds for other languages too, like C++. Check the following snippet:

```
int i = 65;
int ii = 'A';
char c = 65;
char cc = 'A';

printf("%d %d %d %d\n", i, ii, c, cc);
printf("%c %c %c %c\n", i, ii, c, cc);
```

That leads to the following output:

```
65 65 65 65
A A A A
```

As before, we can do:

```
bool is_upper(char c) {
    return 'A' <= c && c <= 'Z';
}
```

Finally, sometimes it may be useful to make use of arithmetic operators. For example, to get the 5th uppercase letter, you could do:

```
printf("%c\n", 'A' + 4); // Prints 'E'.
```

D.3 Editorial

Wondered why certain two characters were emphasized in the sample I/O, and even subtly referenced by the artist in the image header of the statement?

The problem was inspired by some kind of riddle or wordplay: if you looked at your keyboard and visualized the letters as persons, which letters would be nearest to the space bar, and are also present in "space bar"? Just picking out and naming those two letters won't really make for a good competitive programming problem though, so it was left as a sort of Easter egg instead, haha.

Problem E: Space Bar Space

Setters: Kevin Atienza, JD Dantes

Testers: JD Dantes, Hans Olano

Statement Authors: JD Dantes

Test Data Authors: Josh Quinto

Editorialists: JD Dantes

E.1 Solution

For this problem, we have to once more find the uppercase character(s), then propagate the capitalization to the left and right until whitespace is encountered.

For options on how to check for letters, capitalization, or spaces, you may want to check the editorial for the problem *Space Bar*.

We can scan each line to look for the letters, then propagate as described while flagging the characters which we already passed through so we don't duplicate the operations.

Just be careful that you're scanning each line as a whole, and not reading input using routines that disregard whitespace. As an example, for C++ you may want to use `fgets()` or `getline()`. Also make sure that you don't print out any unnecessary spaces either.

Problem F: Good Boy? Or Big Hands?

Setters: JD Dantes

Testers: JD Dantes

Statement Authors: JD Dantes

Test Data Authors: Kevin Atienza, Josh Quinto

Editorialists: JD Dantes

F.1 Solution

Similar to the previous problems, we want to filter out and normalize the words to letters only. Here's one way in which you may do it, in Python:

```
>>> def process_word(word):
    word = ''.join([c.lower() for c in word if c.isalpha()])
    return word
...
>>> process_word("Can't")
'cant'
```

For more discussion on filtering based on capitalization and other convenient utilities, you may want to check the editorial of *Space Bar*.

Now, we just have to build a vocabulary of the normallized words, and we can store the unique words using standard data structures like sets.

What remains is to just go over the new notes and then compare the count of old words to new words.

Here's what that logic could look like:

```
l = int(input())
vocab = set()
for _ in range(l):
    line = input()
    for token in line.split():
        vocab.add(process_word(token))

n = int(input())
for _ in range(n):
    m = int(input())
    num_old = 0
    num_new = 0
    for _ in range(m):
        line = input()
        for token in line.split():
            token = process_word(token)
            if token in vocab:
```

```
            num_old += 1
        else:
            num_new += 1

if num_old >= num_new:
    print('GOOD BOY!')
else:
    print('BIG HANDS!')
```

F.2 Editorial

The story in this problem goes together with the story in *Space Bar*. The references are about a relatively recent show which you might like if you're into AI and startups.

Problem G: Jammonds

Setters: Kevin Atienza

Testers: Patrick Celon

Statement Authors: Payton Yao, JD Dantes

Test Data Authors: Kevin Atienza

Editorialists: JD Dantes

G.1 Solution

For this problem, the formula was already given, and the solution is straightforward.

Rearranging the formula, we simply need to print out $a \cdot d^2$.

The input constraints are also small and should not cause any arithmetic overflows, though of course you can still use languages with big integer implementations like Python.

G.2 Editorial

This problem references the inverse-square relationship between a star's absolute luminosity, apparent brighness, and distance that is observable in real life.

Problem H: We Found Love (TBA)

Setters: Kevin Atienza

Testers: Patrick Celon

Statement Authors: Payton Yao, JD Dantes

Test Data Authors: Kevin Atienza, Josh Quinto

Editorialists: Patrick Celon

H.1 Solution

Coming soon!

Problem I: We Found Love (TBA)

Setters: Jared Asuncion

Testers: Josh Quinto

Statement Authors: Tim Dumol

Test Data Authors: Patrick Celon

Editorialists: Josh Quinto

I.1 Solution

Coming soon!

Problem J: Auld Lang Syne (TBA)

Setters: Payton Yao

Testers: Kevin Atienza

Statement Authors: Payton Yao

Test Data Authors: Patrick Celon

Editorialists: Josh Quinto

J.1 Solution

Coming soon!

Problem K: Astronomic Plagiarism (TBA)

Setters: Patrick Celon

Testers: Josh Quinto

Statement Authors: Patrick Celon

Test Data Authors: Patrick Celon, Kevin Atienza

Editorialists: Patrick Celon

K.1 Solution

Coming soon!

Problem L: Starry Knight (Hint)

Setters: JD Dantes

Testers: Kevin Atienza

Statement Authors: JD Dantes

Test Data Authors: JD Dantes

Editorialists: Kevin Atienza

L.1 Solution

Hint: BFS.

More details coming soon!

Problem M: Raggy Orpov (Hint)

Setters: JD Dantes

Testers: Kevin Atienza

Statement Authors: JD Dantes

Test Data Authors: JD Dantes

Editorialists: Kevin Atienza

M.1 Solution

Hint: Let $s(u,w)$ denote a shortest path from a node u to the node w. Let v be an intermediary node in a shortest path from u to w. What can you say about such a shortest path $s(u,w)$, in relation to $s(u,v)$ and $s(v,w)$?

More details coming soon!

Problem N: Hot Sus (Hint)

Setters: JD Dantes

Testers: Josh Quinto

Statement Authors: Payton Yao

Test Data Authors: JD Dantes

Editorialists: Josh Quinto

N.1 Solution

Hint: Treat it more of a sets problem, rather than of graphs. Consider a case-by-case approach. Use bitsets.

More details coming soon!

Problem O: Hot Susej (Hint)

Setters: JD Dantes, Kevin Atienza

Testers: Josh Quinto

Statement Authors: Payton Yao

Test Data Authors: JD Dantes

Editorialists: Josh Quinto

O.1 Solution

Hint: Treat it more of a sets problem, rather than of graphs. Consider a case-by-case approach. Use bitsets.

More details coming soon!

Problem P: Divisor Counting (Hint)

Setters: Kevin Atienza

Testers: Patrick Celon

Statement Authors: Patrick Celon

Test Data Authors: Kevin Atienza

Editorialists: Patrick Celon

P.1 Solution

Hint: Sieve of Eratosthenes/Atkin

More details coming soon!

Problem Q: Divisor Counting 2: Exactic Boogaloo (Hint)

Setters: Kevin Atienza

Testers: Patrick Celon

Statement Authors: Patrick Celon

Test Data Authors: Kevin Atienza

Editorialists: Kevin Atienza

Q.1 Solution

Hint: Big integer

More details coming soon!

Problem R: The Picks of Destiny (TBA)

Setters: Kevin Atienza

Testers: Josh Quinto

Statement Authors: Payton Yao

Test Data Authors: Kevin Atienza

Editorialists: Kevin Atienza

R.1 Solution

Coming soon!