Computing Laboratory

Recap

Malay Bhattacharyya

Assistant Professor

Machine Intelligence Unit
Indian Statistical Institute, Kolkata

January, 2021

1 Problems

Problem 2-2

Suppose the version number of a software is denoted as $x_1.x_2.\cdots.x_n$ where x_1, x_2, ..., and x_n denote the version number, subversion number, and so on, respectively. E.g., GCC 9.1 is a version of GNU C Compiler. Consider that $x_i \in [0, 9]$ for all i. Given the lower and upper bound of version numbers of a software and n (the number of components of a version number) as user inputs, list up all the possible versions that might exist within the given range (inclusive).

Some random thoughts!!!

- Given the bounds in the form $x_1.x_2.\cdots.x_n$, we need to have controls over those n components.

Some random thoughts!!!

- Given the bounds in the form $x_1.x_2.\cdots.x_n$, we need to have controls over those n components.
- The controls that we require over the said components are iterative in nature.

Some random thoughts!!!

- Given the bounds in the form $x_1.x_2.\cdots.x_n$, we need to have controls over those n components.
- The controls that we require over the said components are iterative in nature.
- Do we really need n loops?

Understanding the problem – Simple case

Lower bound: 1.2, **Upper bound:** 2.1

	x_1	.	x_2	
Lower bound	1	.	2	12
...	1	.	3	13
...	...	.	...	...
...	2	.	0	20
Upper bound	2	.	1	21

Understanding the problem – Simple case

Lower bound: 1.2, **Upper bound:** 2.1

	x_1	.	x_2	
Lower bound	1	.	2	12
...	1	.	3	13
...	...	.	...	...
...	2	.	0	20
Upper bound	2	.	1	21

Note: The constraint $x_i \in [0, 9]$ for all i is essentially a favorable scenario here.

Understanding the problem – Special case I

Lower bound: 1.2.1, **Upper bound:** 2.1

	x_1	.	x_2	.	x_3	
Lower bound	1	.	2	.	1	121
...	1	.	2	.	2	122
...	...	.	...	.	...	...
...	2	.	0	.	9	209
Upper bound	2	.	1			210 (A mismatch)

Understanding the problem – Special case II

Lower bound: 1.2, **Upper bound:** 2.1.1

	x_1	.	x_2	.	x_3	
Lower bound	1	.	2			120 (A mismatch)
...	1	.	2	.	1	121
...	...	.	...	.	...	...
...	2	.	1	.	0	210
Upper bound	2	.	1	.	1	211

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?
  - Use a `for` loop

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?
  - Use a `for` loop
- Can we put the dots ('.') back to the version numbers?

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?
  - Use a `for` loop
- Can we put the dots ('.') back to the version numbers?
  - Use the `print()` function with an appropriate end delimiter

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?
  - Use a `for` loop
- Can we put the dots ('.') back to the version numbers?
  - Use the `print()` function with an appropriate end delimiter
- How to deal with the bordering mismatches?

Brainstorming on the problem

- Can we remove dots ('.') from the input bounds of version numbers and get them as integers?
  - Use the `replace()` and `int()` function
- Can we loop through the integers?
  - Use a `for` loop
- Can we put the dots ('.') back to the version numbers?
  - Use the `print()` function with an appropriate end delimiter
- How to deal with the bordering mismatches?
  - Print the input bounds in exact and loop through the intermediate values (exclusive)

The approach

1. Print the lower bound of version number.
2. Convert the lower and upper bounds to integers.
   i) Remove dots.
   ii) Match the lengths by padding trailing zeros.
   iii) Convert to integers.
3. Loop through the intermediate values (exclusive).
4. Print the intermediate values as version numbers.
   i) Convert to strings.
   ii) Insert dots.
5. Print the upper bound of version number.

Writing the code

```
def RangeVer(lb, ub):
    for v in range(lb, ub):
        for i in range(len(str(v))-1):
            print(str(v)[i], end = '.')
        print(str(v)[i+1])
lb, ub = input("Enter lower & upper bound: ").split()
print(lb)                  # Lower bound as given by user
lbn, ubn = lb.replace('.', ''), ub.replace('.', '')
if len(lbn) > len(ubn):
    ubn = ubn.ljust(len(lbn), '0')   # Trailing zeros
lbn = lbn.ljust(len(ubn), '0')       # Trailing zeros
RangeVer(int(lbn)+1, int(ubn))      # Exclusive range
print(ub)                  # Upper bound as given by user
```