NPTEL MOOC

PROGRAMMING, DATA STRUCTURES AND ALGORITHMS IN PYTHON

Week 1, Lecture 2

Madhavan Mukund, Chennai Mathematical Institute
http://www.cmi.ac.in/~madhavan

An algorithm for `gcd(m,n)`

- Use `fm`, `fn` for list of factors of `m`, `n`, respectively
- For each `i` from `1` to `m`, add `i` to `fm` if `i` divides `m`
- For each `j` from `1` to `n`, add `j` to `fn` if `j` divides `n`
- Use `cf` for list of common factors
- For each `f` in `fm`, add `f` to `cf` if `f` also appears in `fn`
- Return largest (rightmost) value in `cf`

Can we do better?

- We scan from `1` to `m` to compute `fm` and again from `1` to `n` to compute `fn`

- Why not a single scan from `1` to `max(m,n)`?

  - For each `i` in `1` to `max(m,n)`, add `i` to `fm` if `i` divides `m` and add `i` to `fn` if `i` divides `n`

Even better?

- Why compute two lists and then compare them to compute common factors `cf`? Do it in one shot.
    - For each `i` in `1` to `max(m,n)`, if `i` divides `m` and `i` also divides `n`, then add `i` to `cf`
- Actually, any common factor must be less than `min(m,n)`
    - For each `i` in `1` to `min(m,n)`, if `i` divides `m` and `i` also divides `n`, then add `i` to `cf`

A shorter Python program

```
def gcd(m,n):

  cf = []
  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      cf.append(i)

  return(cf[-1])
```

Do we need lists at all?

- We only need the largest common factor
- `1` will always be a common factor
- Each time we find a larger common factor, discard the previous one
- Remember the largest common factor seen so far and return it
  - `mrcf` — most recent common factor

No lists!

```
def gcd(m,n):

  for i in range(1,min(m,n)+1):
    if (m%i) == 0 and (n%i) == 0:
      mrcf = i

  return(mrcf)
```

Scan backwards?

- To find the largest common factor, start at the end and work backwards
- Let `i` run from `min(m,n)` to 1
- First common factor that we find will be gcd!

No lists!

```
def gcd(m,n):

  i = min(m,n)

  while i > 0:
    if (m%i) == 0 and (n%i) == 0:
      return(i)
    else:
      i = i-1
```

A new kind of repetition

```
while condition:
    step 1
    step 2
    . . .
    step k
```

- Don’t know in advance how many times we will repeat the steps
- Should be careful to ensure the loop terminates—eventually the condition should become false!

Summary

- With a little thought, we have dramatically simplified our naive algorithm
- Though the newer versions are simpler, they still take time proportional to the values `m` and `n`
- A much more efficient approach is possible