Loop invariant and binary search

Monash ICPC workshop

Shizhe Zhao

Introduction

- Pre-condition
- Invariant, e.g.:
  - “is prime”
  - $\sum i$
  - …
- Post-condition

```cpp
// prime sieve
for (int i=2; i<=n; i++) {
  bool flag = false;
  for (int j=2; j<i && flag == false; j++)
    if (i % j == 0) flag = true;
  if (flag == false)
    primes.push_back(i);
}

// prefix sum
sum = 0
for (int i=0; i<n; i++) sum += i;
```

Facts about "*invariant*":

- backbone of program, help you:
  - write code elegantly
  - understand code fast
- 90% of my bugs are caused by this

Example 1: Prime Sieve

Naive way:

```
p = []                      # empty prime list at beginning
for i in range(2,n+1):      # is i a prime?
  f = False
  for j in range(2, i):     # any divisor in [2, i-1]?
    if i % j == 0:
      f = True
      break
  if not f:                 # no divisor
    p.append(i)             # yeah, it's a prime!
```

Example 1: Prime Sieve

Naive way:

```
p = []                      # empty prime list at beginning
for i in range(2,n+1):      # is i a prime?
  f = False
  for j in range(2, i):     # any divisor in [2, i-1]?
    if i % j == 0:
      f = True
      break
  if not f:                 # no divisor
    p.append(i)             # yeah, it's a prime!
```

$2+3+2+5+\ldots \approx O(\frac{n^2}{logn})$ (https://oeis.org/A088821)

Example 1: Prime Sieve

More efficient way:

```
f = [0] * (n+1)             # set f[0]=0, f[1]=0, ... f[n]=0
p = []                      # empty prime list at beginning
for i in range(2, n+1):     # is i a prime?
  if not f[i]:              # not sieved by any value
    p.append(i)             # yeah, it's a prime!

  j = 2                     # sieve:
  while j*i <= n:           # 2*i,
    f[j*i] = True           # 3*i,
    j += 1                  # ...
```

Example 1: Prime Sieve

More efficient way:

```
f = [0] * (n+1)              # set f[0]=0, f[1]=0, ... f[n]=0
p = []                       # empty prime list at beginning
for i in range(2, n+1):      # is i a prime?
  if not f[i]:               # not sieved by any value
    p.append(i)              # yeah, it's a prime!

  j = 2                      # sieve:
  while j*i <= n:            # 2*i,
    f[j*i] = True            # 3*i,
    j += 1                   # ...
```

$\frac{n}{2} + \frac{n}{3} + \ldots + 1 \approx O(nlogn)$ (Harmonic sequence)

Example 1: Prime Sieve

Linear prime sieve:

```
f = [0] * (n+1)            # set f[0]=0, f[1]=0, ... f[n]=0
p = []                     # empty prime list at beginning
for i in range(2, n+1):    # is i a prime?
  if not f[i]:             # not sieved by any value
    p.append(i)            # yeah, it's a prime!

  for j in p:              # let j be a known prime
    if j * i > n: break    # reach the upper bound
    f[j * i] = True        # sieve j * i
    if i % j == 0: break   # guarantee j is the minimum divisor
```

Example 1: Prime Sieve

Linear prime sieve:

```
f = [0] * (n+1)                # set f[0]=0, f[1]=0, ... f[n]=0
p = []                         # empty prime list at beginning
for i in range(2, n+1):        # is i a prime?
  if not f[i]:                 # not sieved by any value
    p.append(i)                # yeah, it's a prime!

  for j in p:                  # let j be a known prime
    if j * i > n: break        # reach the upper bound
    f[j * i] = True            # sieve j * i
    if i % j == 0: break       # guarantee j is the minimum divisor
```

Each number is only sieved by it's minimum divisor once.

Example 1: Prime Sieve

Linear prime sieve:

```
f = [0] * (n+1)            # set f[0]=0, f[1]=0, ... f[n]=0
p = []                     # empty prime list at beginning
for i in range(2, n+1):    # is i a prime?
  if not f[i]:             # not sieved by any value
    p.append(i)            # yeah, it's a prime!

  for j in p:              # let j be a known prime
    if j * i > n: break    # reach the upper bound
    f[j * i] = True        # sieve j * i
    if i % j == 0: break   # guarantee j is the minimum divisor
```

Each number is only sieved by it's minimum divisor once.
It's linear!

Example 2: Pairs

Given an array a, integer k, count the number of pair that $a_i - a_j = k$.

Example 2: Pairs

Given an array a, integer k, count the number of pair that $a_i - a_j = k$.

```
a.sort()
tot, i, j, n = 0, 0, 0, len(a)

while i < n:

    while j < n and a[j] - a[i] < k: j+=1

    cnt = 0
    while j < n and a[j] - a[i] == k:
        cnt += 1
        j += 1

    tot += cnt
    while i+1 < n and a[i+1] == a[i]:
        i += 1
        tot += cnt
    i += 1
```

Binary Search

- Most popular topic in tech-interview
- Most common algorithm in your programming life
- More complicated than people expect
  - Not in standard library
  - People hardly ever do it correct!
  - So many variants

How complicated?

Alice and Bob are playing number guessing game,

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

Bob guesses:

- Bob: what's that

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

Bob guesses:

- Bob: what's that
- Bob: lower-bound/upper-bound

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

Bob guesses:

- Bob: what's that
- Bob: lower-bound/upper-bound
- Bob: maximum false less than lower-bound

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

Bob guesses:

- Bob: what's that
- Bob: lower-bound/upper-bound
- Bob: maximum false less than lower-bound
- Bob: minimum true larger than upper-bound
- ...

How complicated?

Alice and Bob are playing number guessing game,
Alice chooses:

- Alice: an integer in $[1, 100000]$
- Alice: an integer from a sorted list
- Alice: a continuous range of integers
- ...

Bob guesses:

- Bob: what's that
- Bob: lower-bound/upper-bound
- Bob: maximum false less than lower-bound
- Bob: minimum true larger than upper-bound
- ...

All combinations!

Keep in mind

- What's your search space at beginning? (precondition)
- How to deal with each branch?(invariant)
- Proof the correctness? (postcondition)
- How to guarantee there is a termination?

My favoriate pattern

Assuming the game rule: Alice choose an integer, tell Bob true ($\geq$), or false ($<$).

```
best = None
while l <= r:           # search space is [l, r]
  m = (l + r) // 2      # choose middle
  if check(m) :         # is m >=
                        # invariant: all v in [l, r] >=
    l = m + 1           # shrink search space
    best = m            # best so far
  else:
    r = m - 1           # shrink search space
return best
```

Why *Binary*?

The math model of cost function $C(n)$:

- branch: $C(i)$ or $C(n-i)$, depends on "feedback"
- choice: $i \in [1 \ldots n]$
- worst case: $max(C(i), C(n-1))$
- minimize total cost: $min(\texttt{worst case}_i)$

In total:

$$C(n) = min(\; max(\; C(i),\; C(n-i)\;)\;)_{i \in [1 \ldots n-1]}$$

Thus we choose $i = \frac{n}{2}$.

Coding time

Firstly, make sure you know how to deal with IO

- `https://vjudge.net/contest/361612` (support: `py, cpp`)
- `https://vjudge.net/contest/361790` (cpp only)

Practice problems (support: `py, cpp`):

- `https://vjudge.net/contest/361685`