CP Gym

by Kevin Foong

1 Maths

1.1 Check number parity fast

$$x\&0b1 == 0?even : odd$$

1.2 Quick multiply/divide by powers of 2

$$x >>= n$$

to divide by 2^n,

$$x <<= n$$

to multiply by 2^n

1.3 Fast exponentiation by squaring

To find a^b in $O(logb)$, let b be represented as powers of 2, simplest way to express in binary. e.g. $10 = 1010 = 2^3 * 2^1$

1 when pow is odd, convert to even pow by: $a^{pow} = a^{pow-1} * a$

2 when pow is even, $a^{pow} = a^{pow/2} * a^{pow/2} = (a^{pow/2})^2 = (a^2)^{pow/2}$

Since the depth of the recursion pow / 2 for each call, the max number of calls is $O(logn)$.

Read more: Fast power algo explanation

1.4 Gaussian Addition

Sum of 1 to n, $f(n) = (n+1) * n/2$
Can be applied to compute multiples of k, $g(k) = k + 2k + 3k... = k(1 + 2 + ... + n)$
Intuition:

1 + 2 + ... + n = n + n-1 + ... + 1
hence 2 * f(n) = (n + 1) + (n + 1) ... n times
f(n) = n * (n-1) / 2

1.5 GCD and Euclidean Algorithm

Brute force is to iterate through the space and search for i s.t $a\%i == 0\&\&b\%i == 0$ where $i \leq a$ and $i \leq b$. Otherwise, use euclidean algo,

$$gcd(a, 0) = gcd(0, a) = a \quad \text{(base case)}$$
$$gcd(a, b) = gcd(a, a\%b) \quad \text{(recursive case)}$$

Intuition:

for (2), lets assume $a > b$ and express a in remainder-quotient form where $a = x * b + r$
gcd(a,b) is s.t $k * gcd(a, b) = a$ and $y * gcd(a, b) = b$ exists.
hence, $r = k * gcd(a, b) - x * y * gcd(a, b)$ and hence $r|gcd(a, b)$

next, we know gcd(a, b) divides b and c.
hence, $gcd(b, c) \geq gcd(a, b)$

next, prove that gcd(b,c) also divides a. (same proof as gcd(a,b) divides c)
then, $gcd(a, b) \geq gcd(b, c)$

hence, $gcd(a, b) = gcd(b, a - b)$
$r = a\%b = a - b - b... - b$
hence, $gcd(a, b) = gcd(b, a\%b)$

C++ STL

```
#include <algorithm>
  __gcd(int, int)
```

Read more: Euclidean Algo Khan Acad

1.6 GCD and LCM

$$gcd(a, b) * lcm(a, b) = a * b$$

Intuition:

suppose a = 6 and b = 15, represent in unique prime factorisation by fundamental theorem of arithmetic

$$a = 2 * 3 \tag{1}$$
$$b = 2 * 3 * 5 \tag{2}$$

gcd(a,b) is the common prime factors which is $2 * 3$
lcm(a,b) is the max power of each prime factor of a and b which is $2^1 * 3^1 * 5^1$.

$$a * b = 2^2 * 3^2 * 5 \tag{3}$$
$$= gcd(a, b) * 2 * 3 * 5 \qquad = \tag{4}$$

1.7 Find primes with Sieve of Eratosthenes

There are no known formulas to find prime numbers without some kind of search in the number space. Sieve is a complicated sounding but actually simple and brute-force like algorithm to finding primes.

Sieve video

0 we know that 0 and 1 are not prime numbers (for fundamental theorem of arithmetic to hold)

1 store a prime number tracker array from 0 -¿ n (hence size is n + 1), let all elems be initially marked as prime.

2 for each integer from 2 till n, if prime, then update all multiples of i to non-prime starting from i * j where j = i until $j * i > n$. else, skip elem. (this is the 'sieve'/'filter' part)

3 the tracker array stores all primes up to n.

To find prime factors of n: finding prime factorization with sieve of Eratosthenes Construct the sieve and for each composite number c, store the smallest prime number p s.t $k * p = c$
then, we can reconstruct each composite number out of its prime factors by doing n / p = new n and repeating till new n = 1.
The prime numbers used is the unique prime factorization of n.

Reusing sieve for multiple test cases Check question constraints, possible to generate reusable prime numbers list.

1.8 Finding factors by reducing search space to $\sqrt{n}$

find all a and b such that $a * b = n$
$min(a, b) \leq \sqrt{n}$ (proof is intuitive and can be found here: Proof)
hence, only check for divisibility from 1 till $\sqrt{n}$ and the corresponding factor to find all factors.

2 Sorting Algorithms

2.1 Counting sort

O(n+k) runtime

If range k of values is known, we can apply counting sort to sort fast!

1 iterate through the elems and count the occurrence of each elem.

2 store the cumulative count of elems ¡ i and i itself.

3 iterate through each list elem, lookup the count from (2), position (one-based) = stored count.

4 place list elem into position and decrement the stored count from (2) by 1.

The intuition for steps 3 and 4 are that step 2 stores the position of the last elem with the value, hence once we place list elem, we decrement the position for the next one.

3 Tutorials

3.1 CodeChef Problem Code:CHEFSQRS

Find min x which is a square which when added to n also results in a square.

Solving technique for math definition problems - convert question by expressing as an equation to solve
want to find m and x s.t

$$n + m * m = x * x \tag{5}$$
$$n = (x * x) - (m * m) \tag{6}$$
$$n = (x - m) * (x + m) \tag{7}$$

want the min square, so want min m. find factors of n = a * b such that a - b is min. hence, start closest to the $\sqrt{n}$ since the diff between factors will be the min.

3.2 Question 1(Watermelon 800):

Concept: Parity of addition of odd and even numbers

Let n be even if $n = 2k$. Let n be off if $n = 2k + 1$.

Hence,

- for x,y are odd, $x + y = 2w + 1 + 2z + 1 = 2(w + z + 1)$, hence x + y is even (by definition of even numbers)
- for x is even and y is odd, $x + y = 2w + 2z + 1 = 2(w + z) + 1$, hence x + y is odd (by definition of odd numbers)
- for x,y are even, $x + y = 2w + 2z = 2(w + z)$, hence x + y is even (by definition of even numbers)

Concept: LSB of odd numbers is 1.

Let x be an odd number and y be an even number. Use this property to check parity fast.

$$x\&0b1 = 1 \tag{8}$$
$$y\&0b1 = 0 \tag{9}$$

3.3 Question 2(Weights Assignment for Tree Edges 1500):

n-node trees have n-1 edges. There is a unique path between root to all nodes.

We can augment the tree with a corresponding array ie. index i stores weight of node i.

The question wants an increasing path weight for each node in the given order. hence, we can pre-allocate the path weights.

Since a unique path exists to the root to all nodes, hence, dist[root, node i] = dist[root, node i's parent] + weight[i, i's parent]

Hence, to determine each edge weight, just take the pre-allocated distances and minus to get the weight of the edge.

To check if ordering is invalid, check if the $dist[root, nodei'sparent] > dist[root, nodei]$ which is impossible since i's parent to root is a subpath of i to root.

3.4 Question 3(Escape the maze hard 1900):

Simply perform BFS on all friends and vlad.
If vlad's new frontier is empty, then no path can be found to a leaf.
Else, if vlad's new frontier has a new node that has only 1 neighbour i.e. a leaf, then the path has been found.

To count the minimum number of friends needed, simply count the number of LCA. We know that Vlad will **only ever encounter LCA during his traversal.** Hence, number of LCA = number of times vlad visits a node visited by a friend = min friends.

3.5 Question 4(ATM and Students 1800)

Initially, i attempted this question using a modified Zidane's algorithm - but this didn't consider that the solution must be locally optimal at all times. e.g. ATM bal = 0, -5, 5 is invalid despite the total sum >= which seems valid.