ALGO-101

Week 3 - Math

Hacer Akıncı

ITU ACM

October 2022

Topics

Topics covered at week 3:

- Prime Numbers & Sieve
- Prime Factorization
- Permutation & Combination
- Fast Exponentiation
- Bits & Bitset
- Game Theory

Primality Test

A number is prime if it is only evenly divisible by one and itself. How to find if a number is prime or not?

Naive Approach:
We should check the numbers from 2 to n-1 to see if any number evenly divides n. The divisor of a number must be equal to or less than itself, so we just need to look at numbers less than n.

How to optimize the naive approach?
To iterate the numbers to $\sqrt{n}$ instead of n. If the number (n) has a divisor bigger than $\sqrt{n}$, then the other multiplier must be smaller than $\sqrt{n}$. And, we already checked the numbers to $\sqrt{n}$, so we don't need to look at the rest of it.

Optimized Primality Test

```
1
2  bool isPrime(int n){
3
4      if(n == 0 || n == 1) return false;
5
6      for(int i = 2; i*i <= n; i++){
7          // If there is a number divide n evenly, n is not a prime
8          if(n % i == 0) return false;
9      }
10
11     return true;
12 }
```

Finding Primes Up to N

We learned how to find if a number is prime or not. Now, we will learn how to find primes up to N.

Naive Approach:
We can use isPrime function for every number from 1 to N. It's time complexity is $O(N \times \sqrt{N})$.

Sieve of Eratosthenes Approach:
It is one of the most efficient ways to find primes up to N. It is based on marking multiples of prime numbers as not prime. Let's look at the example.

Sieve Algorithm

For $N = 50$

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sieve Algorithm

We start from 2 and marked multiples of 2.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sieve Algorithm

Next unmarked number would be our next prime. Mark multiples of 3. Skip 4 because it is marked.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sieve Algorithm

Unmarked numbers are prime numbers.Like as optimized naive approach, we check to $\sqrt{N}$.

1	2	3	4	5	6	7	8	9	10
11	12	13	14	15	16	17	18	19	20
21	22	23	24	25	26	27	28	29	30
31	32	33	34	35	36	37	38	39	40
41	42	43	44	45	46	47	48	49	50

Sieve Algorithm Code

```
vector<bool> Sieve_of_Erotosthenes(int n){

    vector<bool> primes (n+1, true);
    primes[0] = false;
    primes[1] = false;

    for(int i = 2; i*i < n; i++){
        if(primes[i]){
            for(int j = i*2; j < n; j+=i){
                primes[j] = false;
            }
        }
    }

    return primes;
}
```

Time Complexity of Sieve Algorithm

Inside for loop does n/p_i step when p_i is prime. Total processes is equal to sum of $1/p_i$ from 1 up to $\sqrt{n}$.

$$\frac{n}{2} + \frac{n}{3} + \frac{n}{5} + \ldots + \frac{n}{p_i} = n \times \left\{ \frac{1}{2} + \frac{1}{3} + \frac{1}{5} + \ldots + \frac{1}{p_i} \right\}$$

$$\sum_{i=2}^{n} \frac{1}{p_i} = \log \log n.$$

According to prime harmonic series, the sum $(1/i)$ where i is prime is $\log \log n$. Time complexity of sieve approach is O(n$\times$log *logn*)

Factorization Algorithm

How to find all factors of given number n?

Naive Approach:
Check the numbers from 1 to n if divides n, then it is a factor of n.

Optimized naive approach:
Instead of checking up to n, check the numbers up to $\sqrt{n}$, then add i and n/i to the array.
The logic of this approach is the same as for optimized primality test.

Prime Factorization

How to find prime factors of a number in high school?

To find prime factors of a number (n), we can use a similar method to sieve approach. Create an array that stores the smallest prime factor of every number from 2 to n.

We add arr[n], the first prime factor of n, to the array of prime factors. Then we divide n by its smallest prime factor. The result will be n/arr[n]. We repeat this operation until the result is 1. Let's look at the example.

Prime Factorization

For N = 18
Create an array size of $n + 1$, then assign every index to index value.

index	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
value	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18

Starting from two, mark all multiples of it to i.

index	1	**2**	3	**4**	5	**6**	7	**8**	9	**10**	11	**12**	13	**14**	15	**16**	17	**18**
value	1	**2**	3	**2**	5	**2**	7	**2**	9	**2**	11	**2**	13	**2**	15	**2**	17	**2**

Prime Factorization

For the next index if arr[i] equals i, i is a prime number, mark all multiples. If it is not, that means i is not a prime number, skip it.
While marking the multiples. If it is already marked with another value, skip it. Because every index should store the smallest prime factor of itself. In this example, skip 6, 12, 18.

index	1	**2**	**3**	**4**	5	**6**	7	**8**	**9**	**10**	11	**12**	13	**14**	**15**	**16**	17	**18**
value	1	**2**	**3**	**2**	5	**2**	7	**2**	**3**	**2**	11	**2**	13	**2**	**3**	**2**	17	**2**

It is redundant to look at any bigger than $\sqrt{n}$, because the rest unsigned are prime numbers.

Prime Factorization

After creating the array, to find prime factors;

$n = 18$ arr[18] = **2**;
$n = 18 \ / \ 2$;

$n = 9$ arr[9] = **3**;
$n = 9 \ / \ 3$;

$n = 3$ arr[3] = **3**;
$n = 3 \ / \ 3$;

$n = 1$ end condition

Prime factors = {2, 3, 3}

18	2
9	3
3	3
1	

Permutation

Permutation is the number of possible orders of a given set. Formula;

$$P = \frac{n!}{(n-k)!}$$

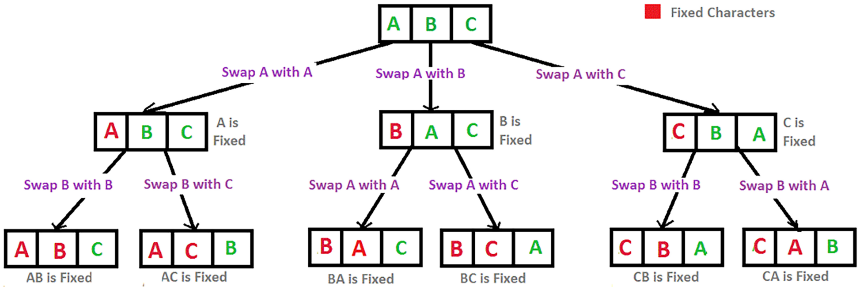
We can find the p with a for loop and its time complexity is k. For bigger numbers it will be find with fast exponentiation. But how can we find the all permutation of given set?

There are other ways to find all permutations. However, all of them time complexity is $O(n \times n!)$. In this approach the problem is solved by swapping or more known name backtracking.

Generating Permutation

First, swap with 0 index element to other indexes starting from 0.
First element was swapped. Skip to second element, and swap with other elements starting from second element.
It goes like until no element left to swap. For example;

$$s = \{A, B, C\};$$



Recursion Tree for Permutations of String "ABC"

Permutation Code

```
1  class Solution {
2      // store all permutations
3      vector<vector<int>> permutations;
4
5      public:
6      vector<vector<int>> permute(vector<int>& set) {
7          // swap starting 0 index
8          swap_function(set, 0);
9          return permutations;
10     }
```

Permutation Code

```
1     void swap_function(vector<int>& subset, int index){
2         // return when there is no element left to swap
3         if(index == subset.size()){
4             permutations.push_back(subset);
5             return;
6         }
7         for(int j = index; j < subset.size(); j++){
8             // create a vector to not change original set for other usage
9             vector<int> temp;
10            temp = subset;
11            swap(temp[index], temp[j]);
12
13            // 0 to index value are swapped and fixed
14            // so call the swap_function for the rest of it
15            swap_function(temp, index + 1);
16        }
17    }
18 };
```

Combination

Combination is to find subsets length k of n length set. Unlike permutation, order does not matter for combination. Formula for combination;

$$C = \frac{n!}{(n-k)! \times k!}$$

It can be solved by a factorial function and its time complexity is $O(n)$.
In optimized version, time complexity would be $O(r)$, where r is minimum of $n - k$ and k.

While finding combination and permutation, overflow may happen if the result is bigger than limit of "int" in any step . To avoid this, the modulo operation is used.

Modular Inverse

It is known that the product of a number by its multiplicative is 1. The inverse of a number n is 1/n except 0.

$$n \times \frac{1}{n} = 1$$

Modular multiplicative inverse of n under modula m is n^{-1};

$$n \times n^{-1}(mod\ m) = 1$$

Example;

n = 3 m = 11 $n^{-1} = ?$

$(3 \times 4)\%11 = 1$

Note: m must be coprime to n, and the modular inverse of n should be in the range 1 to m-1.
How to find modular inverse of n under modulo m?

Naive Approach:

Look in the range [1,m-1], and check a number (k) $(n \times k)(mod m) is 1$.

Fermat's Little Theorem

Fermat's little theorem states that if p is a prime number, then for any integer a, the number $a^p - a$ is multiple of p.

$$\begin{aligned} a^p - a &= k \times p \\ a \times (a^{p-1} - 1) &= k \times p \\ a^p &\equiv a (mod\ p). \end{aligned}$$

Special Case: If a is not divisible by p, we can multiply both sides by a^{-1}, and we get;

$$a^{p-2} = a^{-1} (mod\ p)$$

So we can find modular inverse of a under modulo p as a^{p-2}.

Exponentiation

First way that come to mind for to find n^k is multiply n to itself k times in a for loop. It is naive approach of exponentiation.

However there is a way this problem can be solved by another approach with $O(\log k)$ time complexity called **fast exponentiation**. That base on this rule

if k is **even** $n^k = n^{k/2} \times n^{k/2}$

if k is **odd** $n^k = n^{k-1/2} \times n^{k-1/2} \times n$

Fast Exponentiation Code

```
1
2  long long power(long long n, long long m, long long mod){
3
4      if(n == 1 || m == 0) return 1;
5      if(n == 0) return 0;
6
7      long long x = power(n, m/2) % mod;
8
9      x = (x*x) % mod;
10
11     if(m%2 == 0) return (x);
12     else return (x*n) % mod;
13
14 }
```

Bits & Bitset

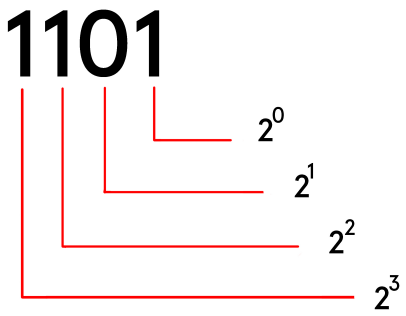
What is bit?
We use the decimal digit system more in our daily life, but bit (binary digit) are used in computers and programming. A bit is the smallest unit of data that a computer can process and store.
1 and 0, yes/no, on/off or true/false

Bitset is an array of bits. Why do we use bitset in competitive programming?

- Faster (same information but in compress manner)
- Space-optimize (bool array use 1 byte for each value while bitset use one bit for each value)
- Bitsets have the feature of being able to be constructed from and converted to both integer values and binary strings

Bits & Bitset

It can be used for integer operations. How to change decimal to binary?



$(2^3).1 + (2^2).1 + (2^1).0 + (2^0).1 = 13$

std::bitset

```cpp
// default constructor initializes with all bits 0
    bitset<4> a4; //0000
    bitset<8> a8; //00000000

// initialize with an integer
    bitset<4> b(11); //1101

// initialize with a binary string
    bitset<4> c(string("1101"));

// assign value to a index
// indexes stars from right
    b[2] = 0 //1001
//               ^
//               | = 0th index

// count() returns number of set bits(1) in bitset
    int number_of_ones = b.count(); // equals 2 for b = 1001
```

std::bitset

```
1
2  // size() returns the total bits in bitset
3      int total = b.size(); //equals 4 for b = 1001
4
5  // any() returns true, if atleast 1 bit is set***
6      bool is_set = b.any(); // equals true for b = 1001
7      bool is_set = a4.any(); // equals falsee for a1 = 0000
8
9  // set() set all bits
10     b.set(); // b = 1111
11 // set(position, value) for given index
12     b.set(3, 0); // b = 0111
13
14 // reset(position) unset the given index
15     b.reset(0); // b = 0110
16
17 // reset() unsets all bits
18     b.reset(); // b = 0000
```

std::bitset

```
// flip() flips all bits (1 <-> 0)
    b.flip(); // b = 1111

// flip() flips the given index
    b.flip(2); // b = 1011

// OPERATIONS

bitset<4> x(5); // 0101
bitset<4> y(7); // 0111

// & (AND) compare two bits and return 1 if both are 1
bitset<4> z = x & y //0101
int and = x & y; // and = 5

// | (OR) compare two bits and return 1 if any of the two is 1
bitset<4> z = x | y //0111
int or = x | y; // or = 7
```

std::bitset

```cpp
// ^ (XOR) compare two bits and return 1 if two bits are different
bitset<4> z = x ^ y //0010
int xor = x ^ y; // xor = 2

// bitwise operation and assignment
x &= y; // x = x & y

// ~ (NOT) inverts all the bits
bitset<4> z = ~x // 1010
int not = ~x // not = 10

// << (left shift) shift to left by the amount of second variable
x << 1 ; // 1010

// >> (right shift) shift to right by the amount of second variable
x >> 2 ; // 0010
```

Bits & Bitset

Where do we use bitset?

Check if a number is even or odd
(n & 1) for even numbers result is 0, for odd numbers result is 1
Example;
n = 50
100010 & 000001 = 000000
n = 17
010001 & 000001 = 000001

Bits & Bitset

To multiply by 2

$12 \times 2 =$
$001100 = 12$
$12 << 1$
$011000 = 24$

To divide by 2

$17/2 =$
$010001 = 17$
$17 >> 1$
$001000 = 8$

int 32 bit store but since first bit is used for sign, it can be used 31 bit. Integer bigger than $2^{31} - 1$ cause overflow.

Note: unsigned int used 32 bit all. So the max limit is $2^{32} - 1$

Game Theory

Game theory is a comprehensive topic that includes methods for solving different types of games. Therefore a few games will be explained in this lesson, since it will take a long time to mention all of them. You can learn more by looking at this **PDF**.

Who is winner?

Game of Replacing Elements

This game is played with two players. An integer array is given and the first player chooses two different numbers from the array (x, y). All the y's in the array are replaced by x. Then the second player chooses two different numbers and does the same. In the end, when all the elements in the array are the same, the player with the turn of the move loses because two different numbers cannot be selected.

Example

arr[ ] = {1, 3, 3, 2, 2, 1}

First plays always loses irrespective of the numbers chosen by him.

First player picks 1 & 3 replace all 3 by 1.
Now array become arr[ ] = {1, 1, 1, 2, 2, 1}

Then second player picks 1 & 2 either he replace 1 by 2 or 2 by 1
Array become arr[ ] = {1, 1, 1, 1, 1, 1}

Now first player is not able to choose. Second player win.

Solution

It is about whether the number of distinct elements is even or not.

Let's say we have n distinct element in the array. First player chooses 2 of them and replaces one to another, and now we have n-1 distinct elements. Then the second player replaces distinct element numbers decreased by one (n-2). It goes like that until (n-i) will be one. Therefore, if n is even, the first player always wins.

Game is valid or not

Check if the game is valid or not

A game is played with two players, but there are 3 players (P1, P2, P3). Let's say this game is table tennis. One player is waiting for other players to play. The spectator plays with winner of the match. This time defeated player wait and game continues like this.

Note: There is no draw in the game and first game is played by P1 and P2.
The number of game and array of winners each game are given, and asked this game is valid or not in the question.

Example

Input :
Number of Games : 4
Winner of the Game G_i : 1 1 2 3
Output : YES

Explanation :
Game1 : P1 vs P2 : P1 wins
Game2 : P1 vs P3 : P1 wins
Game3 : P1 vs P2 : P2 wins
Game4 : P3 vs P2 : P3 wins
None of the winners were invalid

Input :
Number of Games : 2
Winner of the Game G_i : 2 1
Output : NO

Explanation :
Game1 : P1 vs P2 : P2 wins
Game2 : P2 vs P3 : P1 wins (Invalid winner)
In Game2 P1 is spectator

Solution

We know that winner shouldn't be spectator. First spectator is player 3. Then after every match we can find next spectator by subtracting the current spectator and winner from total sum. Then we should check in other step if winner is spectator. If it is, this game is invalid.

```
bool is_game_valid (int arr[]){
    int size = arr.size();
    int spectator = 3;

    for (int i = 0; i < size; i++){
        if(arr[i] = spectator) {
            return false;
        }
        spectator = 6 - arr[i] - spectator;
    }

    return true;
}
```

Problem Session

Count Primes - Prime Numbers
Permutations - Permutation
ZSUM - Fast Exponentiation
Power of Two - Bitset
Sum vs XOR - Bitset
Tower Breakers - Game Theory
Nim Game - Game Theory

References

https://www.geeksforgeeks.org/
https://www.geeksforgeeks.org/write-a-c-program-to-print-all-permutations-of-a-given-string/
https://www.geeksforgeeks.org/compute-ncr-p-set-3-using-fermat-little-theorem/
https://www.geeksforgeeks.org/multiplicative-inverse-under-modulo-m/
https://www.geeksforgeeks.org/c-bitset-and-its-application/
https://github.com/inzva/Algorithm-Program/blob/master/bundles/03-math-1/03$_{m}ath1.pdf$
Sieve of Eratosthenes (n.d.). In Codility. Retrieved October 21, 2018, from
https://codility.com/media/train/9-Sieve.pdf