Prakhar Newatia

Extended Algorithms

1) Segmented Sieve :-

- Sieve → It is a method to find ~~raw~~ prime numbers within a range from 0 to n (→ max limit)

For eg. range n = 10

take 3 cross all multiples of 3

2	3	~~4~~	5	~~6~~	7	~~8~~	~~9~~	~~10~~

take 2 cross all multiples of 2 → the part which is cut just move forward.

after every operation {2, 3, 5, 7}

- Back to Segmented Sieve → provided a range N – m and find all prime range elements between them → take an array of 20-10+1 length

N – m
10 20 (suppose)

Step 1 → find sieve for ($\sqrt{m}$ i.e. $\sqrt{20}$) → {2, 3} – D

Step 2 → for each D take one by 1.

Eg. a = 2
b = N/a
c = a*b
while c<n
c = c+a

→ b = 10/2 = 5
c = 5*2 = 10
now 10 is in range 10-20

for a = 3
b = 10/3 = 3
c = 3x3 = 9
while 9 < 10 :
c = c+3 = 12 c = 12

Step 3 : make cut for each c in range

Eg.

10	11	12	13	14	15	16	17	18	19	20
~~0~~	1	~~2~~	3	~~4~~	~~5~~	~~6~~	7	~~8~~	9	~~10~~

for c = 10
a = 2

for c = 12
a = 3

remaining non cut values are prime
{11, 13, 17, 19}.

=> Fast Modular Exponential

Our motive a^{b} mod n
(a: base, b: exponent)

Eg. 3^{100} mod 15

1) find binary of 100 (exponent)

$(100)_{10} = (1100100)_2$ → base 2

2) make a table

1	1	0	0	1	0	0
3	12	9	6	3	9	6

first element always base

→ this is the answer

* for each next step check 1 is coming or zero is coming
* for the first case in example 1 is coming :

if 1 →

3^2 mod 15 = 9 mod 15 = 9

now 9 × 3 mod 15 = 27 mod 15 = 12

fill this value

elif 0 →

12^2 mod 15 = 9

fill with this

defaultdict

★ $(a+b) \bmod m = (a\%m + b\%m)\%m$
$(a-b) \bmod m = (a\%m - b\%m)\%m$
$(a*b)\%m = ((a\%m)(b\%m))\%m$
$\left(\frac{a}{b}\right)\%m \neq \left(\frac{a\%m}{b\%m}\right)\%m$ } but for division it is not possible

So to do this
$(a \cdot b^{-1})\%m = ((a\%m)(\underbrace{b^{-1}}_{\text{multiplicative inverse}}\%m))\%m$ ★

Multiplicative inverse →

$a \cdot a^{-1} \equiv 1 \pmod m$

★ Fermat's Theorem →

If m is prime $\Rightarrow a^{m-1} \equiv 1 \pmod m$

Eg $m=5 \quad a=3 \quad$ so $3^4 \Rightarrow 81\%5 = 1$

To find a^{-1} we use this fermat's theorem →

$a \cdot a^{m-2} \equiv 1 \pmod m$
a^{m-2} is mmI

$m=5$
$a=4$ so $a^{m-2} = 4^3 = 64$
now $a * a^{m-2} = 256\%5 = 1$

★ Euler's Totient.

★ if suppose m is not prime but we know a, m are co-prime i.e. $\gcd(a,m)=1$

$a^{\phi(m)} \equiv 1 \pmod m$

$\phi(m) = n \prod (1 - \frac{1}{p})$ $p \leftarrow$ prime $p \leq n$

$616 = 2^3 \times 7^1 \times 11^1$

Eg. $\phi(616) = 616 \times (1-\frac{1}{2})(1-\frac{1}{7})(1-\frac{1}{11}) = 240$

Eg. $5^{240} \equiv 1 \pmod{616}$

So for $a \rightarrow a^{(\phi(m))-1}$ is its mmI

⇒ Extended Euclidean Algorithm

$\gcd(884, 54) = 6$

$884 = 54(16) + 24$
$54 = 24(2) + 6$
$24 = 6(4) + 0$

$6 = 54 - 24(2)$
$= 54 + 24(-2)$
$= 54 + (888 - 54(16))(-2)$
$6 = 54(33) + 888(-2)$

So for eg in inverse modulo
$1 \equiv a(x) + m(y)$
↳ mmI

Eg. $17^{-1} \pmod{43}$
$1 = 2 \times 43 + 17 \times (-5)$ ↳ mmI
$-5 \equiv \underline{38}$ Ans

decimal to binary ⟷ bin(number)
hexadecimal => hex(number)
octal = oct(number)
All these return string

eval()

```
def egcd(a,b):
    x,y,u,v = 0,1,1,0
    while a!=0:
        q,r = b//a, b%a
        m,n = x-u*q, y-v*q
        b,a,x,y,u,v = a,r,u,v,m,n
    gcd = b
    return gcd,x,y
```

✓ Longest Palindromic Substring using Manacher's Algorithm

a	b	a	x	a	b	a	x	a	b	b
1	3	1	7	1	9	≥1	5	1	1	1

. . . .

↓ max

Manacher's Algo - O(n) ⟶ Difficult to implement

Dynamic Programming - $O(n^2)$ → use a tabular approach

Eg. abaabc → length = 6

1) → Create table of 6×6

Table T →

i \ j	0 a	1 b	2 a	3 a	4 b	5 c
0 a	T	F	T	F	F	F
1 b	F	T	F	F	T	F
2 a	F	F	T	T	F	F
3 a	F	F	F	T	F	F
4 b	F	F	F	F	T	F
5 c	F	F	F	F	F	T

0 2
2-0+1=3
1 a b
2 3

Rule 1 →
Mark all T[i][i] = True
as all single letters are palindrome itself

Rule 2 →
if j-i = 1
if a[i] == a[j] (j = i+1)
mark the block True

Rule 3 → The golden rule
if j-i >= 2 Eg T[0][2]
fill the value i.e
(str[i] == str[j] && T[i+1][j-1])

★ For every instance of True you have to store the start index and max length

Manacher's revisited

str → a b a a b c

→ Pre Processing

when pointer comes to # just move forward

mirroring

again

skipping # step

now take ... remove ...

Best Python advantage

eval() → It helps in evaluation

Tutorial

Manacher's revisited

will have default value

str → a b a a b c → L = list(str) k = '#'.join(L) k = '#' + k + '#'

→ Pre Processing one Pointer (iterator).

a # b # a # a # b # c

0 0 0 0 0 0 0 0 0 0 0 0 0

When pointer comes to # just move forward

L ↓ R

a # b # a # a # b # c

0 1 0 0 0 0 0 0 0 0 0 0 0

mirroring i.e find a palindrome at a i.e #a# so taking 2 pointer L & R

L ↓ R

a # b # a # a # b # c

0 1 0 0 0 0 0 0 0 0 0 0 0

again for # move forward

L ↓ R

a # b # a # a # b # c

0 1 0 3 0 0 0 0 0 0 0 0 0

Skipping # Step

L ↓ R

a # b # a # a # b # c

0 1 0 3 0 1 0 0 0 0 0 0 0

min(R - index(a), index(a) - LM) = 1

L ↓ R

a # b # a # a # b # c

0 1 0 3 0 1 4 0 0 0 0 0 0

L ↓ R

a # b # a # a # b # c

0 1 0 3 0 1 4 1 0 0 0 0 0

a # b # a # a # b # c

[0 1 0 3 0 1 4 1 0 1 0 1 0] → final array

now take the max, clip the string remove '#' and that's the answer.

Maximum Non-Divisible Subset

A = [4,5,2,6,1,7,8,12] K=4

So here the question is to find the largest subset of S where the sum of any 2 numbers in S' is not evenly divisible by K.

* To find this we will deal with modulo 4 elements as it will come only 0,1,2,3 for K=4.

new A = [0,1,2,2,1,3,0,0]

now make a new array p = [0]*(k+1) { [0,0,0,0,0] (0 1 2 3 4)

add the count to the p array i.e make it like

p: [3,2,2,1,0] now think that we can only add one 1 element to the list whose K % K == 0

hence we do p[0] = min(p[0], 1)

if K%2==0 then p[k//2] can also be chosen as element atmost 1 so p[k//2] = min(p[k//2], 1) only to when even

* because 4 = 2+2 so if there to be 2 elements

else just pass.

c=0 now apply loop from 0 to length of p//2

c = c + min(p[k], p[len(p)-i] => Fenwic

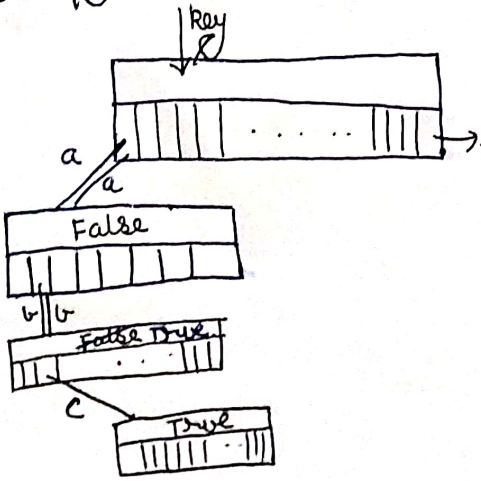
---

* In a 2D Graph if a path is possible from (a,b) to (c,d) then there is a condition that gcd(a,b) = gcd(

Trie implementation in Python

* This data structure is useful in dictionaries question

Basic structure →



inserting 2 strings in order

abc

ab

* find sum in

9+10 =

* Default ... the keys is not set, ... import defaultdict

To check no string is prefix of another we check 2 conditions

i) If while iterating to a string any key value is True then return False

ii) If the whole string has ended and you find there is still atleast 1 child node then return False

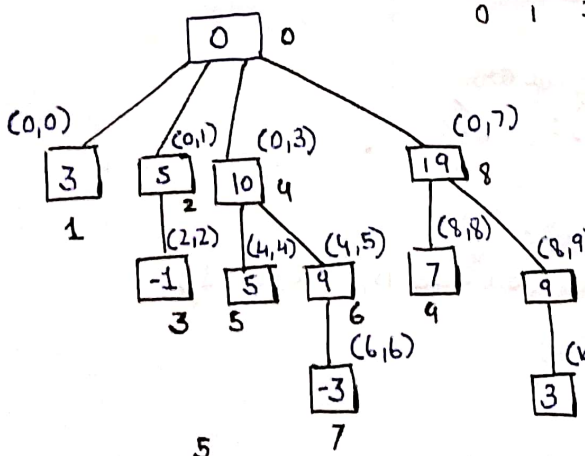
Structure →

```
Class prak
    def __init__(self, initval=None):
        self.value = None
        self.next = None

Class trie:
    def __init__(self, initval=None):
        self.value = initval
        self.check = None → to check if child has any node or not
        self.arr = {}  } for every character make a key value pair where key is the character and value is the prak object
```

=> Fenwick Tree or Binary Indexed Tree

3	2	-1	6	5	4	-3	3	7	2	3
0	1	2	3	4	5	6	7	8	9	10



1 = 0 + 2^0
2 = 0 + 2^1
3 = 2^1 + 2^0
4 = 0 + 2^2
5 = 2^2 + 2^0
6 = 2^2 + 2^1
7 = 2^2 + 2^1 + 2^0

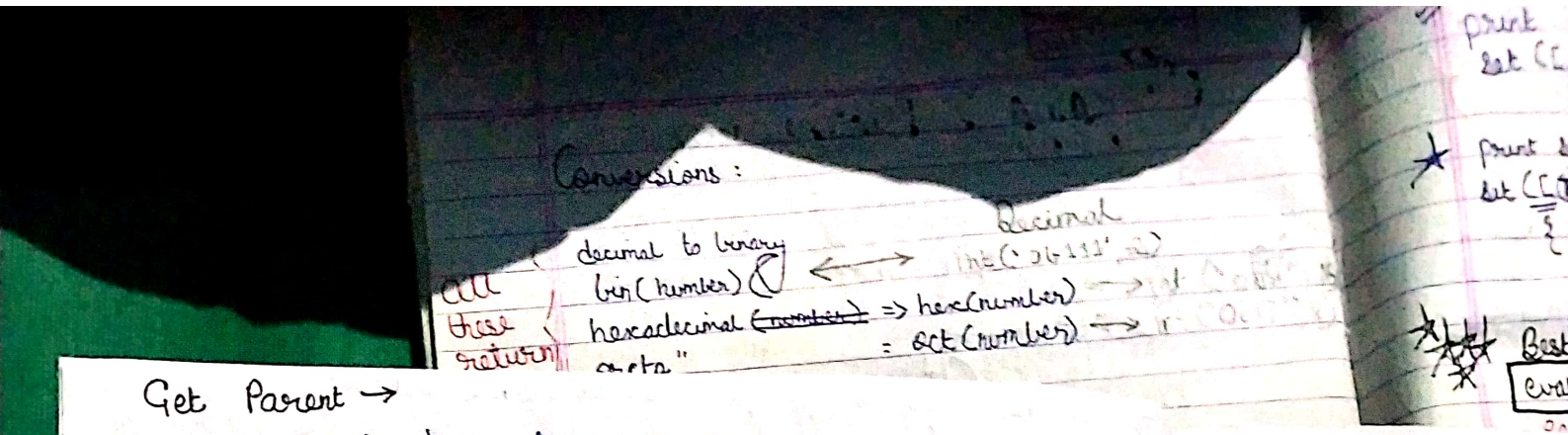
starting from 6 add +1 block to tree node 7

* Find sum in range (0, 5)

9 + 10 = 19

↳ we got to 6 [110]
to 4 ↓ [100]
to 0 ↓ [000]
this is the answer

} removing parity bits (rightmost 1 by 1).



Get Parent →

1) 2's complement
2) AND it with original number
3) Subtract from original number

$x -= x\&(-x)$

O(1)

Get Next →

1) 2's complement
2) AND with original number
3) Add to original number

key to produce the array of BIT Tree

$x += x\&(-x)$

apply this until $x <= n$

O(n log n)

★ how to update any element in the tree? O(log n)

=> suppose we update A[3] = 9 {originally was 6}

get the difference = 9-6 = 3

go next to 3+1 = 4

→ array created by us from next

```
def getsum(BitTree, i):
    b = BitTree
    s = 0
    i = i+1
    while i>0:
        s = s + b[i]
        i -= i&(-i)
    return s
```

```
def updatebit(BitTree, n, i, v):
    b = BitTree
    i += 1
    while i<=n:
        b[i] += v
        i += i&(-i)
```

```
def construct(arr, n):
    b = [0]*(n+1)
    for i in range(n):
        updatebit(BitTree, n, i, arr[i])
    return b
```