20/03/2023

Optimizing Sieve of Eratosthenes
This theorm was used to find the prime numbers b/w 0 to (n).
↳ input

Now we are going to optimize the sieve of eratosthenes as we were trying to run a loop from 2 to n & check whether the number is prime or not which has high time complexity.

Sieve of Eratosthenes
↳ array of numbers & it is a boolean array.

```
vector <bool> Sieve (int n) {
vector <bool> sieve (n+1, true);
sieve [0] = sieve [1] = false;
for (int i = 2; i <= n; i++) {
        if (sieve [i] == true) {
                int j = i * 2;
                while (j <= n) {
                        sieve [j] = false;
                        j = j + i;
                }
        }
}
return sieve;
}
```

The above code we have already discussed but now we can do some modifications to the above code to make it optimal.

1st optimization (Inner loop)

See the inner while loop.

Prime — Non Prime

2 → 4, 6, 8, 10, 12, 14, 16, 18

3 → 6, 9, 12, 15, 18, 21, 24

5 → 10, 15, 20, 25

7 → 14, 21

at

When we are at 2 we have already marked 6 as non-prime & then again at 3, we have marked 6 again as non-prime & this is repeated task. Hence we can do optimization here. Similary many others were marked non-prime which were already non-prime.

So instead of starting from $i * 2$, we will start marking from $i * i$ as they won't be marked

$i = 2$ → Start from 4

$i = 3$ → Start from (9) → not marked by 2

$i = 5$ → Start from (25) → not marked by 2, 3.

Hence the 1st optimization that we can do is changing $j = i * 2$ to $j = i * i$.

2nd optimization (Outer loop)

Can we optimize the outer loop?

Suppose $n = 25$ and $i = 7$ so j runs from 49 & hence this loop won't run as $49 > 25$. Hence we can optimize this.

Modify the outer loop from ($i = 2$ to n) to $i = 2$ to $\sqrt{n}$ but why $\sqrt{n}$.
When $i = \sqrt{n}$, $i * i = \sqrt{n} \times \sqrt{n} = n$ & hence $j <= n$ is satisfied, so inner loop will work here which was not working in the example we took before optimization.

Code

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

    vector <bool> sieve (n+1, true);
    sieve [0] = sieve [1] =          false;
    // Optimization - 2
    for (int i = 2 ; i <= sqrt(n) ; i++) {
            if (sieve [i]) {
                int j = i * i ; //Optimization - 1
                while (j <= n) {
                    sieve [j] = false;
                    j = j + i;
                }
            }
    }
    return sieve;
}
```

Segmented Sieve

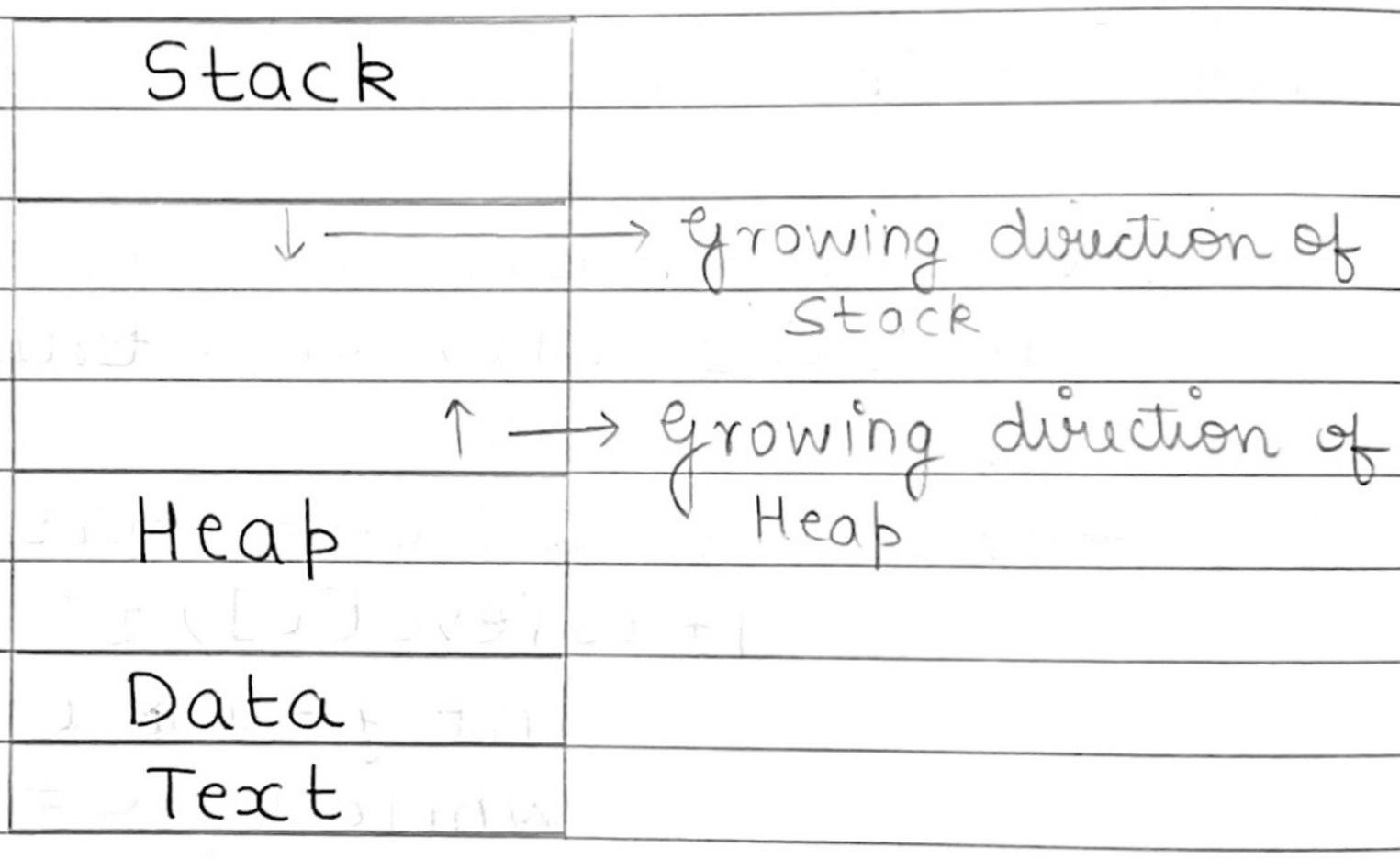
Now suppose that we are given 2 numbers namely L and R & we need to find the prime numbers in this range only, that is within L to R.

Ex→ L = 10, R = 25

Make an array of size 26 & apply the Sieve of eratos theres on this array & then start the traversal from L to R & print whichever entry is marked as true.

But what if $R = 10^9$, then we won't be able to allocate this size array.

There is a limit on stack size



* There is a max size of int, char, double i.e. 10^6
* Boolean array has max size 10^7.
* Global array
  - int, char, double → 10^7
  - bool → 10^8

However the above size will depend on the computer architecture.

So now the question comes to our mind that now how can we solve the segmented sieve.

Ex→ $1 \leq (L, R) \leq 10^9$

$(R - L) \leq 10^6$

So let's say L = 110 & R = 130. This is given to us as the input.
Let's make an array of size R-L+1 = 21.

0th index ⇒ 110
1st index ⇒ 111
:
:
20th index ⇒ 130

Here we are making segmented sieve & not full sieve. Here we didn't make sieve of size 131.

Algorithm

1) Initially in segmented sieve, all are marked as 1, that is prime numbers.
2) Now find out the prime numbers using normal sieve by placing $n = \sqrt{R}$

$\sqrt{130} \rightarrow 11$

11 x 11 ⇒ start from 121 } ✓
13 x 13 ⇒ start from 169 } X

Base Primes = {2, 3, 5, 7, 11}
↳ (end) as 11 x 11 = 121

3) Now find first index to start marking.
index 0 → 110
index 20 → 130

When prime = 2, start marking from 110 as it comes into 2's table.
When prime = 3, start marking from

$\left(\frac{110}{3}\right) \times 3 \Rightarrow 36.6 \times 3 \Rightarrow 36 \times 3 = 108$

Here $108 < L$ and hence add prime to the calculated number

$$108 + 3 = 111$$

Start from 111.

Formulae of 1st multiple $= \left(\frac{L}{Prime}\right) \times Prime$

In normal sieve

$$j = i * i$$

Here

$$j = max\ (first\_multiple, prime * prime);$$

We have to select maximum out of first_multiple and prime * prime as it might be possible that first_multiple is already marked.

Code

```
vector <bool> segSieve (int L, int R){

    vector <bool> sieve = Sieve (sqrt(R));
    vector <int> basePrimes;
     // Step-2
    for(int i=0; i< sieve.size(); i++){
                if (sieve[i])
                        basePrimes.push_back(i);
    }
    // Step-1
    vector <bool> segSieve (R-L+1, true);
```

```
// Base condition
if (L==1 || L==0){
        segSieve[L] = false;
}
// Step-3
for (auto primes : basePrimes){
    int first_mul = (L/prime)* prime;
    if (first_mul < L) {
        first_mul = first_mul + prime;
    }
    int j = max (first_mul, prime * prime);
    while (j<=R){
        segSieve[j-L] = false;
        j = j + prime;
    }
}
return segSieve;
}
```

Condition already done in step-2 ← might be already marked

Same as normal sieve

Note → In segSieve array, the ones which are marked will be prime. So in that case just print i+L.

0th index → 110