

সংখ্যাতত্ত্ব: মৌলিক সংখ্যা- প্রাইম ফ্যাক্টরাইজেশন, SOD এবং NOD

Prime numbers and its algorithms in C++ Bangla tutorial



Sharif Hasan • December 4, 2020 সর্বশেষ আপডেট January 11, 2024 4 2,132 পড়তে 5 মিনিট লাগতে পারে

মৌলিক সংখ্যা

মৌলিক সংখ্যা (prime number) নিয়েই আজকের লিখা, তাই প্রথমেই জেনে নিই যে মৌলিক সংখ্যা কি?

মৌলিক সংখ্যা কাকে বলে?

মৌলিক সংখ্যা হলো এমন একটি প্রকৃত সংখ্যা (২ থেকে শুরু) যা শধু ১ অথবা ওই সংখ্যাদিয়েই নিঃশেষে ভাগ যায়। প্রথম এবং একমাত্র জোড় মৌলিক সংখ্যা হলো ২। পরবর্তি মৌলিক সংখ্যা গুলো হলো ৩, ৫, ৭, ১১, ১৩, ১৭, ১৯, ...। অসীম সংখ্যক মৌলিক সংখ্যা আছে। ০-১০০ এর মধ্যে ২৫ টি, ০-১০০০ এর ১৬৮ টি, ০-১০০০০ এর মধ্যে ১২২৯ টি মৌলিক সংখ্যা আছে। অনেক বড় মৌলিক সংখ্যার (large prime number) উদাহরণ হলো **1299709, 15485863, 179424673, 2147483647, 32416190071, 112272535095293, 48112959837082048697** ইত্যাদি।

*The **largest known prime number** (as of November 2020) is $2^{82,589,933} - 1$, a **number** which has 24,862,048 digits when written in base 10. It was found via a computer volunteered by Patrick Laroche of the Great Internet Mersenne **Prime Search (GIMPS)** in 2018.*

প্রাইম নাম্বার বা মৌলিক সংখ্যা প্রোগ্রামিং প্রতিযোগিতাতে একটি জনপ্রিয় বিষয়। আজকে আমরা প্রাইম নাম্বারের সঙ্গে যুক্ত অ্যালগরিদম নিয়ে আলোচনা করবো।

মৌলিক সংখ্যা কাকে বলে?

গণিতের পরিভাষায় ১ এর চেয়ে বড় সেই সকল সংখ্যাকে মৌলিক সংখ্যা বলা হয় যাদের শুধুমাত্র দুইটি উৎপাদক আছে। একটি হল ১ এবং অন্যটি সে নিজে।

মৌলিক সংখ্যা: প্রাইম নাম্বার টেস্ট করা (

Primality test)

প্রাইম নাম্বার টেস্ট (**primality test**) করার জন্য আমরা যেকোনো সংখ্যা N কে $1, 2, \dots, N - 1$ সংখ্যাগুলো দিয়ে ভাগ দিবো। এই সমাধানটাই আমাদের মাথায় সবার প্রথমে আসে। এই অ্যালগরিদমটির **টাইম কমপ্লেক্সিটি** $O(N)$ । যা আসলে সবচেয়ে ভালো সমাধান নয়। আমরা আরও দ্রুত সংখ্যাটি মৌলিক কিনা টেস্ট করতে পারি।

আরেকটু ভালো অ্যালগরিদম, টাইম কমপ্লেক্সিটি $O(\sqrt{N})$

যদি কোন নাম্বার N প্রাইম বা মৌলিক না হয় তবে আমরা একে দুইটা ফ্যাক্টর বা গুণনীয়ক a এবং b তে ভাগ করতে পারি। সুতরাং আমরা বলতে পারি, $n = a \times b$ এখন এটা দেখানো যাবে যে, a আর b কখনো একসাথে \sqrt{N} এর চেয়ে বড় হবে না এবং এক সাথে ছোটও হবে না। কারণ যদি a আর b দুইটাই \sqrt{N} এর চেয়ে বড় হয় তবে, $a \times b \geq N$ কারন, $\sqrt{N} \times \sqrt{N} = N$ । সুতরাং N এর সকল গুণনীয়ক গুলোর মধ্যে এমন একটি গুণনীয়ক আছে যা \sqrt{N} এর ছোট অথবা সমান থাকবে। তাই আমরা যদি \sqrt{N} এর চেয়ে ছোট কোন ডিভিজর বা গুণনীয়ক না পাই তাহলে বলতে পারি N অবশ্যই একটি মৌলিক সংখ্যা।

আসলে আমরা যদি 2 থেকে \sqrt{N} পর্যন্ত সকল মৌলিক সংখ্যা দিয়ে ভাগ করি কাজ হয়ে যাবে, কিভাবে? একটু ভেবে দেখুন 😊। এর কমপ্লেক্সিটি $O\left(\frac{\sqrt{N}}{\ln(\sqrt{N})}\right)$

আরও পড়ুন [প্রাইম কাহিনি – মৌলিক সংখ্যা এর অ্যালগরিদম গুলো](#)

মৌলিক গুণনীয়ক (**Prime divisors**) বের করার জন্য অপটিমাইজড ডিভিশন অ্যালগরিদম

সংখ্যাতত্ত্ব থেকে জানি, মৌলিক সংখ্যা (**prime number**) N এর শুধু 1 এবং N বাদে অন্য কোনও গুণনীয়ক নেই। অপর দিকে N যদি কোনও যৌগিক সংখ্যা হয় (**Composite number**) তবে একে তার সমস্ত মৌলিক গুণনীয়কের গুণফল হিসেবে প্রকাশ করা যাবে, অর্থাৎ মৌলিক সংখ্যাগুলো হলো যৌগিক সংখ্যার বিল্ডিং ব্লক।

উদাহরণ সরুপ,

$N = 2700 = 2 \times 2 \times 3 \times 3 \times 3 \times 5 \times 5 = 2^2 \times 3^3 \times 5^2$ পরেরটুকুকে মৌলিক সূচকে ফ্যাক্টরাইজেশন বলে।

কোন সংখ্যার (N) মৌলিক গুণনীয়ক বের করার জন্য আমরা প্রথমে একটি মৌলিক

সংখ্যার লিস্ট তৈরি করে পরে দেখতে পারি **N** কে কোন কোন মৌলিক সংখ্যাদিয়ে ভাগ যায়। যেসব দিয়ে ভাগ যায় ওইগুলোই তার মৌলিক গুণনীয়ক।
তবে আমরা চাইলেই একে আরও দ্রুত করতে পারি।

যেকোনো সংখ্যা **N** কে আমরা $N = pf \times N'$ হিসেবে লিখতে পারি। যেখানে **pf** হলো **N** এর একটি মৌলিক গুণনীয়ক, এবং $N' = \frac{N}{pf}$

আমরা যদি **N** কে তার মৌলিক গুণনীয়কদিয়ে ভাগ করার মাধ্যমে কমাতে থাকি তাহলে একসময় $N = 1$ পাওয়া যাবে, তার কারণ সকল সংখ্যাকে ১ এবং তার অন্য মৌলিক গুণনীয়কের গুণফল আকারে লিখা যায়। মানে আমরা **N** কে তার মৌলিক গুণনীয়ক দিয়ে ভাগ করতে থাকবো যতক্ষণ না পর্যন্ত $N = 1$ হয়।

আমরা আমাদের অ্যালগরিদমটিতে একবার চোখ বুলিয়ে নিই।

1. যতক্ষণ **N** কে ২ দ্বারা ভাগ করা যাবে ততক্ষণ আমরা **N** কে ২ দ্বারা ভাগ করে যেতে থাকবো এবং একই সাথে **i** এর মান প্রিন্ট করতে থাকবো। অর্থাৎ
$$N = \frac{N}{2}$$
2. প্রথম কাজের পর **N** অবশ্যই বিজোড় সংখ্যা হবে, কারণ ১ নং এ আমরা সব ২ কে উঠিয়ে দিয়েছি (কেটে দিয়েছি)। এখন একটি লুপ চালাবো, **i = 3** থেকে $i = \sqrt{N}$ পর্যন্ত (কারণ টা ব্যাখ্যা করছি)। যতক্ষণ **i** দ্বারা **N** কে ভাগ যাবে, ততক্ষণ আমরা **N** কে **i** দ্বারা ভাগ করে **N** এর মান আপডেট করতে থাকবো এবং **i** এর মান প্রিন্ট করতে থাকবো। অর্থাৎ
$$N = \frac{N}{i}$$
3. ২ নং ধাপে যদি $N > 2$ হয় তবে আমরা **N** এর মান প্রিন্ট করবো।

অ্যালগরিদমের ব্যাখ্যা

যেহেতু ২ একটি মৌলিক সংখ্যা, তাই প্রথম ধাপে আমাদের সংখ্যা থেকে সকল ২ কে কেটে দিবো। আমাদের সংখ্যাটি যদি $N = 2700 = 2 \times 2 \times 3 \times 3 \times 3 \times 5 \times 5$ হয় তবে, প্রথম ধাপে আমাদের সমস্ত ২ কাটা যাওয়ায় পর $N = 675 = 3 \times 3 \times 3 \times 5 \times 5$ তে গিয়ে দাঁড়াবে। পরের ধাপে আমরা ৩ থেকে ভাগ করা শুরু করবো। এ ধাপে আমাদের সমস্ত ৩ কাটা পরে $N = 25 = 5 \times 5$ বাকি থাকবে। পরে **i = 3** থেকে **i = 5** এ গিয়ে আমরা **N** কে ৫ দিয়েও কাটাকাটি করে ফেলবো। সব শেষে **N** এর মান গিয়ে ১ এ দাঁড়াবে (আহা রে বেচারি **N**, সব খেয়ে দিলাম :')।

এখন আসি, আমরা আমাদের লুপকে কেন $i = \sqrt{N}$ পর্যন্ত নিলাম? আমাদের কোনও প্রাইম ফ্যাক্টর কি \sqrt{N} এর থেকে বড় হতে পারে না? অবশ্যই পারে। তবে যেকোনো একটি। কিছুক্ষন আগে আমরা প্রমাণ দেখেছিলাম, যেকোনো সংখ্যা $N = a \times b$ হলে

a, b একই সাথে \sqrt{N} থেকে বড় বা ছোট হতে পারবে না। এখন আমরা ধরি, N এর যেকোনো ২ টি প্রাইম ফ্যাক্টর/ডিভিজর/গুননীয়ক p_1, p_2 \sqrt{N} থেকে বড়ো। কিন্তু আমরা জানি, $\sqrt{N} \times \sqrt{N} = N$ । তাই, যখন $p_1 > \sqrt{N}$ and $p_2 > \sqrt{N}$ তখন $p_1 \times p_2 > N$ যা সম্ভব না। তাই আমাদের শুধু একটি মৌলিক গুননীয়ক গুননীয়ক সম্ভব যা \sqrt{N} থেকে বড় অথবা সমান। এই মৌলিক গুননীয়কটিকেই আমাদের ৩ নং ধাপ থেকে হেন্ডেল করা হবে।

C++ কোড

```
C++
1 #include <bits/stdc++.h>
2 using namespace std;
3
4 void primeFactors(int n)
5 {
6     // Print the number of 2s that divide n
7     while (n % 2 == 0)
8     {
9         cout << 2 << " ";
10        n = n/2;
11    }
12
13    // n must be odd at this point. So we can skip
14    // one element (Note i = i + 2)
15
16    for (int i = 3; i <= sqrt(n); i = i + 2)
17    {
18        // While i divides n, print i and divide n
19        while (n % i == 0)
20        {
21            cout << i << " ";
22            n = n/i;
23        }
24    }
25
26    // This condition is to handle the case when n
27    // is a prime number greater than 2
28    if (n > 2)
29        cout << n << " ";
30 }
31
32 int main()
33 {
34     int n = 315;
35     primeFactors(n);
36     return 0;
37 }
```

মৌলিক সংখ্যার অ্যালগরিদম: কোনও সংখ্যা (N) এর গুননীয়কের সংখ্যা বের করা।

এরকম সমস্যা, **Given a number N , find the number of divisors (NOD) it has** বা একটি সংখ্যা N দেয়া আছে, এর কতগুলো গুননীয়ক আছে তা বের করুন, প্রোগ্রামিং কনটেক্সটগুলোতে প্রায়ই দেখা যায়। এই সমস্যাটির একটি সমাধান হতে পারে, ১ থেকে শুরু করে N পর্যন্ত সকল সংখ্যা দিয়ে N কে ভাগ করে যাবো। যেসব সংখ্যা দিয়ে ভাগ করার পরে ভাগশেষ শূন্য হবে, ওইগুলোই হবে তার গুননীয়ক।

কোডটি দেখলে নিচের মতো হবে,

```
C++
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6
7      int n,num_divisors=0;
8      cin>>n;
9      for(int i=1;i<=n;++i){
10         if(n%i==0){
11             ++num_divisors;
12         }
13     }
14     cout<<"Number of divisors = "<<num_divisors<<endl;
15     return 0;
16 }
```

এই কোডের টাইম কমপ্লেক্সিটি হলো $O(N)$ । কিন্তু আমরা এই সমস্যাটিকে $O(\log(N))$ এ সমাধান করতে পারি। কিভাবে?

উপরে আমরা একটি সংখ্যাকে মৌলিক গুণনীয়কে বিশ্লেষণ করেছি। আমরা আমাদের ওই মৌলিক গুণনীয়কগুলোকেই ব্যবহার করবো ওই সমস্যা সমাধান করতে।

ধরা যাক, কোন সংখ্যা $N = p_1^{a_1} \times p_2^{a_2} \times p_3^{a_3} \times p_4^{a_4}$, এখানে p_i হলো N এর একটি মৌলিক গুণনীয়ক। এখন চিন্তা করে দেখি, যেকোনো সংখ্যা, d কে যদি N এর গুণনীয়ক হতে হয় তবে d এর মধ্যে কি কি বৈশিষ্ট্য থাকা জরুরি। (১) d এর মধ্যে N এর মৌলিক গুণনীয়ক p_i ছাড়া অন্যকিছু থাকা চলবে না। (২) d এর মধ্যে p_i এর ঘাত N এর মধ্যে p_i এর ঘাত থেকে বেশি হতে পারবে না। অর্থাৎ $d = p_1^{b_1} \times p_2^{b_2} \times p_3^{b_3}$ যেখানে $0 \leq b_i \leq a_i$ । আমাদের NOD এর সূত্র হলো,

$$NOD(N) = (a_1 + 1)(a_2 + 1) \dots (a_k + 1)$$

এই সূত্রটি এমন কেন হলো? কারণ হলো কোন ডিভিজর d এর ভিতরে প্রাইম নাম্বার p_1 এর ঘাত 0 থেকে a_1 যেকোনোটি হতে পারে। অর্থাৎ $a_1 + 1$ রকম। এভাবে প্রতিটি ঘাত কতরকম হতে পারে তা গুন করলেই আমরা মোট কতগুলি উৎপাদক/ডিভিজর আছে তা বের করতে পারি। কোডটি দেখি,

```
C++
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6
7      int n,num_divisors=1;
8      cin>>n;
9      for(int i=2;i*i<=n;++i){
10         int power=0;
11         while(n%i==0){
```

```

12         ++power;
13         n/=i;
14     }
15     num_divisors=num_divisors*(power+1);
16 }
17 //বর্গমূলের থেকে বড় ফ্যাক্টরের জন্য , আমাদের এরকম ফ্যাক্টর একটিই থাকা সম্ভব, এ
18 if(n>=2) num_divisors=num_divisors*(1+1);
19 cout<<"Number of divisors = "<<num_divisors<<endl;
20 return 0;
21 }

```

কোনও সংখ্যা (**N**) এর গুণনীয়কের যোগফল বের করা (Finding sum of divisors of a given number N)

কোন সংখ্যা N এর সবগুলো গুণনীয়কের যোগফল (Sum Of Divisor বা SOD) বের করতে বলা হয়েছে, এই সমস্যাটিকেই আমরা খুব সহজে সমাধান করতে পারি। তার আগে আমরা SOD এর সূত্রটি দেখে নিই।

$$SOD(N) = (p_1^0 + p_1^1 + p_1^2 + \dots + p_1^{a_1})(p_2^0 + p_2^1 + p_2^2 + \dots + p_2^{a_2})$$

সূত্রটি ভালভাবে বুঝার জন্য আমরা $N = 12$ ধরে নিচ্ছি। তাহলে $12 = 2^2 \times 3^1$ ।
অতএব **SOD(12)** হবে,

$(2^0 + 2^1 + 2^2)(3^0 + 3^1)$ বা
 $2^0 \times 3^0 + 2^0 \times 3^1 + 2^1 \times 3^0 + 2^1 \times 3^1 + 2^2 \times 3^0 + 2^2 \times 3^1$ বা
 $1 + 3 + 2 + 6 + 4 + 12$ অতএব আমরা আমাদের গুণনীয়কের যোগফল পেয়ে
 গেলাম। আশাকরি বুঝা গেছে। 1 নং সমীকরণকে আমরা নিচের মতো লিখতে পারি,

$$SOD(N) = \frac{p_1^{a_1+1} - 1}{p_1 - 1} \cdot \frac{p_2^{a_2+1} - 1}{p_2 - 1} \cdot \frac{p_k^{a_k+1} - 1}{p_k - 1}$$

SOD বের করার কোড,

```

C++
1  #include<bits/stdc++.h>
2  using namespace std;
3
4  int main()
5  {
6
7      int n;
8      cin>>n;
9      map <int,int> primes;
10     for(int i=2;i*i<=n;++i){
11         int power=0;
12         while(n%i==0){
13             ++power;
14             n/=i;
15         }
16         if(power>0){
17             primes[i]=power;
18         }
19     }

```

```
20 //বর্গমূলের থেকে বড় ফ্যাক্টরের জন্য , আমাদের এরকম ফ্যাক্টর একটিই থাকা সম্ভব,
21
22 long long sod=1;
23 primes[n]++;
24 for(pair <int,int> k:primes){
25     int p=k.first;
26     int a=k.second;
27
28     sod=sod*(pow(p,a+1)-1)/(p-1);
29 }
30
31 cout<<"Sum of divisors= "<<sod<<endl;
32
33 return 0;
34 }
```

দেখলাম কিভাবে কোন সংখ্যাকে দ্রুত প্রাইম ফ্যাক্টরে ভাগ করতে পারি এবং তা থেকে কিভাবে Number of Divisor এবং Sum of Divisor বের করতে পারি। আগামী লিখায় অন্য কোন টপিকে লিখবো। আজকের মতো যাচ্ছি, ৬ তারিখ ক্লাসের এসাইনমেন্ট জমা দিতে হবে। কারও সমস্যা বা কনফিউশন থাকলে কমেন্ট অথবা মেইলে জানাবেন।

সংখ্যা তত্ত্ব নিয়ে দ্বিতীয় লিখা এখানে [সংখ্যাতত্ত্ব: মডুলার অ্যারিথমেটিক \(Modular arithmetic\) – Big mod](#)

লেখাটি কেমন লেগেছে আপনার?

রেটিং দিতে হার্টের উপর ক্লিক করুন।



গড় রেটিং 4.2 / 5. মোট ভোট: 29

[#C++](#)[#Problem solving](#)[#অ্যালগরিদম](#)[#গণিত](#)[#প্রবলেম সলভিং](#)[#মৌলিক সংখ্যা](#)[#সংখ্যাতত্ত্ব](#)

4 টি মন্তব্য

**Jenny**

December 4, 2020 at 11:53 PM

ধন্যবাদ, আশা করি প্রোগ্রামিং কনটেন্টে কাজ এ লাগবে। ভাইয়া, ম্যাপ শেখার বাংলা কোন সোর্স থাকলে দি়েন।

Reply

**Something**


December 4, 2020 at 11:55 PM

Well writing .

Reply

**Sharif Hasan**

December 5, 2020 at 5:29 PM

Thank you. .

Reply

Pingback: সংখ্যাতত্ত্ব: মডুলার অ্যারিথমেটিক (Modular arithmetic) - Big mod - শরিফ হাসান