বিটওয়াইজ সিভ প্রাইম সংখ্যা বের করার জন্য প্রচলিত অ্যালগোরিদম Sieve of Eratosthene এ মেমরির ব্যবহার অনেক কমিয়ে আনা যায়! সাধারণ সিভে N পর্যন্ত প্রাইম জেনারেট করলে N সাইজের একটি অ্যারে ডিক্লেয়ার করতে হয়। অ্যরের প্রতিটি এলিমেন্ট একটি করে ফ্ল্যাগ হিসাবে কাজ করে যেটা দেখে আমরা বুঝি একটি সংখ্যা প্রাইম নাকি কম্পোজিট। বিটওয়াইজ্ সিভে আমরা ফ্ল্যাগ হিসাবে ইন্টিজার বা বুলিয়ান এর বদলে সরাসরি বিট ব্যবহার করবো।
এ টিউটোরিয়াল পড়ার আগে দুটি বিষয় তোমাকে জেনে আসতে হবে
১. Sieve of Eratosthene এর সাধারণ ভার্সণ,তুমি এটা আমার এই পোস্টটি পড়ে শিখতে পারবে সহজেই।
২. সি/সি++ এ বিটওয়াইজ্ অপারেটরের ব্যবহার। এটাও খুব সহজে শিখতে পারবে এখান থেকে। টিউটোরিয়ালটির ১ম ২টি অংশ খুব ভালো করে পড়ে ফেলো,বিটমাস্ক ডিপি,বিএফএস যখন শিখবে তখন অনেক কাজে লাগবে।
আশা করি এখন তুমি বিটওয়াইজ্ অপারেটর সম্পর্কে অনেক কিছু জানো,সাধারণ সিভ লিখতে কোনো সমস্যা হয়না তোমার। এবার আমরা শিখবো বিটওয়াইজ সিভ।

সাধারণ সিভে status বা flag অ্যারেটার কাজ কি? এই অ্যারের ইনডেক্সের মান দেখে আমরা বলতে পারি একটি সংখ্যা প্রাইম কিনা। ধরলাম তোমার status অ্যারেটা ইন্টিজার টাইপের। প্রতিটি ইন্টিজারের মধ্য আছে ৩২টি বিট। আমরা কেনো এতগুলো বিট ব্যবহার করবো খালি ০ বা ১ নির্দেশ করতে? আমরা অ্যারের যেকোনো ইনডেক্সের প্রতিটি বিট দিয়ে একটি সংখ্যা নির্দেশ করতে পারি।

তুমি যখন ইন্টিজার অ্যারেতে ১-৭ পর্যন্ত প্রাইম জেনারেট কর, তোমার অ্যারের অবস্থা বাইনারিতে থাকে এরকম:

```
status[১]=০০০........০০(৩২টি শূন্য)
status[২]=০০০........০১(৩১টি শূন্য,১টি ১)
status[৩]=০০০........০১(৩১টি শূন্য,১টি ১)
status[৪]=০০০........০০(৩২টি শূন্য)
status[৫]=০০০........০১(৩১টি শূন্য,১টি ১)
status[৬]=০০০........০০(৩২টি শূন্য)
status[৭]=০০০........০১(৩১টি শূন্য,১টি ১)
```

প্রতিটি ইনডেক্সে ৩১টি বিট কোনো কাজে লাগছেনা অথচ এই বিশাল সংখ্যক অব্যবহৃত বিট আমরা সহজেই কাজে লাগাতে পারি। আমরা ধরে নিবো:

```
status[০] এর
    >>> শূন্যতম বিট ০ এর প্রাইমালিটি নির্দেশ করে (সবথেকে ডানের বিট==০ তম বিট)
    >>> ১ম বিট ১ এর প্রাইমালিটি নির্দেশ করে (সবথেকে ডানের বিট==০ তম বিট)
```

```
   >>> ২য় বিট ২ এর প্রাইমালিটি নির্দেশ করে
   >>> ৩য় বিট ৩ এর প্রাইমালিটি নির্দেশ করে
  .........................
   >>> ৩১তম বিট ৩১ এর প্রাইমালিটি নির্দেশ করে

status[১] এর
   >>> শূন্যতম বিট ৩২ এর প্রাইমালিটি নির্দেশ করে
   >>> ১ম বিট ৩৩ এর প্রাইমালিটি নির্দেশ করে
  ...............

status[২] এর
     >>> শূন্যতম বিট ৬৪ এর প্রাইমালিটি নির্দেশ করে
    >>> ১ম বিট ৬৫ এর প্রাইমালিটি নির্দেশ করে
```

তাহলে ১-৭ পর্যন্ত প্রাইম জেনারেট করলে তোমার অ্যারের অবস্থা দাড়াবে:

status[১]= ০০০০…….১০১০১১০০(মোট ৩২টি বিট) (সবথেকে ডানের বিট==০ তম বিট)

৭টি সংখ্যার কাজ একটি ইনডেক্সেই শেষ!!। শুধু ৭টি নয়,আসলে ৩১টি সংখ্যার কাজ শেষ হবে ১টি ইনডেক্স(কারণ প্রতি ইনডেক্সের ৩১টি বিট ব্যবহার করছি আমরা)। এখন প্রশ্ন হলো কোনো সংখ্যার প্রাইমালিটি কত নম্বর ইনডেক্সের কত নম্বর বিট দিয়ে নির্দেশ করা হবে? খুব সহজ, সংখ্যাটি i হলে i/৩২ নম্বর ইনডেক্সের i%৩২ নম্বর বিট আমাদের চেক করতে হবে। তাহলে i=১ হলে চেক করবো ০ নম্বর ইনডেক্সের ১ নম্বর বিট, i=৩৩ হলে চেক করবো ১ নম্বর ইনডেক্সের ১ নম্বর বিট ইত্যাদি। (শুন্য বেসড ইনডেক্সিং)

কোডিং অংশ একদম সহজ। তুমি যেহেতু বিটওয়াজ অপারেটরের ব্যবহার জানো,কোনো সংখ্যার pos তম বিটে ১ বা ০ আছে নাকি সহজেই চেক করতে পারবে। pos তম বিটে নিজের ইচ্ছামত ১ বা ০ বসাতেই পারবে। আমাদের এখানে ০ বসানো দরকার নেই,১ বসাতে পারলেই চলবে। দুটি ফাংশন লিখে ফেলি:

```
bool Check(int N,int pos){return (bool)(N & (1<<pos));}
int Set(int N,int pos){            return N=N | (1<<pos);}
```

এবার সিভ লিখে ফেলি:

```
int N =100,prime[100];
int status[100/32];
void sieve()
```

```
{
    int i, j, sqrtN;
    sqrtN = int( sqrt( N ) );
    for( i = 3; i <= sqrtN; i += 2 )
    {
                        if( check(status[i/32],i%32)==0)
                        {
                                    for( j = i*i; j <= N; j += 2*i )
                                    {
                                                status[j/32]=Set(status[j/32],j % 32)   ;
                                    }
                        }
            }
            puts("2");
            for(i=3;i<=N;i+=2)
                        if( check(status[i/32],i%32)==0)
                        printf("%d\n",i);

}
```

লক্ষ্য করো,আমরা সাধারণ সিভের মত করেই সব লিখেছি,তবে status[i] এর মান চেক করার বদলে status[i/32] এর i%32 নম্বর বিটের মান চেক করেছি।

বিটওয়াইজ সিভ ব্যবহার করে ১০^৮ পর্যন্ত প্রাইম তুমি জেনারেট করে ফেলতে পারবে। সাধারণ সিভের থেকে সময় + মেমরি কম লাগবে। সাধারণ গুণ,ভাগ অপারেশনের থেকে বিটের অপারেশনগুলো দ্রুত কাজ করে। আমরা আরো কিছু অপটিমাইজেশন করতে পারি। যেমন তুমি উপরে দেয়া টিউটোরিয়াল পড়ে থাকলে জানো যে কাওকে ২ দিয়ে গুণ করা আর সংখ্যাটির বাইনারিকে ১ ঘর বামে শিফট করা একই কথা। আবার ২ দিয়ে ভাগ করা আর ১ ঘর ডানে শিফট করা একই কথা,৩২ দিয়ে mod করা আর ৩১ দিয়ে AND করা একই কথা। তাহলে আমরা নিচের মত করে কোডটি লিখতে পারি:

```
int status[(mx/32)+2];
void sieve()
{
            int i, j, sqrtN;
    sqrtN = int( sqrt( N ) );
    for( i = 3; i <= sqrtN; i += 2 )
    {
                        if( Check(status[i>>5],i&31)==0)
                        {
                                    for( j = i*i; j <= N; j += (i<<1) )
                                    {
                                                status[j>>5]=Set(status[j>>5],j & 31)   ;
                                    }
                        }
            }

            puts("2");
            for(i=3;i<=N;i+=2)
                        if( Check(status[i>>5],i&31)==0)
                        printf("%d\n",i);
}
```

ফাংশন ব্যবহার না করে ম্যাক্রো ব্যবহার করলে আরো কম সময় লাগবে। প্রোগ্রামিং কনটেস্টে খুব কমই বিটওয়াইজ সিভ ব্যবহার করা দরকার হয়,সাধারণ সিভেই কাজ চলে। তারপরেও এটা শিখলে বিটের কনসেপ্ট গুলো কিছুটা পরিস্কার হবে,অন্য কোনো প্রবলেমে হয়তো মেমরি কমিয়ে ফেলতে পারবে। ডাইনামিক প্রোগ্রামিং এ আমরা প্রায়ই বিটমাস্কের ব্যবহার করি মূলত মেমরি কমানোর জন্য।

তোমার ইম্প্লিমেন্টেশন সঠিক নাকি চেক করতে নিচের সমস্যাটি সমাধান করে ফেলো:

http://www.spoj.pl/problems/TDPRIMES/

(নোট: অনেকের ভূল ধারণা আছে যে bool টাইপের ভ্যারিয়েবলের আকার ১বিট। আসলে bool এর আকার ৮বিট বা এক বাইট,char এর সমান। এর কারণ হলো কম্পিউটার ১ বাইটের ছোটো মেমরি সেগমেন্টকে অ্যাড্রেস করতে পারেনা, তাই ভ্যারিয়েবলের নূন্যতম আকার ১ বাইট)