I, ME AND MYSELF !!!

MONDAY, SEPTEMBER 14, 2009

Segmented Sieve

Memory and time efficient :)

Problem Statement:

Your are given two integers **a** and **b**. You have to find all the primes within range **a** and **b**. Here, $1 \le a \le b \le 2^{31}-1$ and $b - a \le 10^5$.

Note: You have to handle 1, 2 and even numbers for appropriate case of your own.

Solution:

```
#include <string.h>

#define MAX 46656
#define LMT 216
#define LEN 4830
#define RNG 100032

unsigned base[MAX/64], segment[RNG/64], primes[LEN];

#define sq(x) ((x)*(x))
#define mset(x,v) memset(x,v,sizeof(x))
#define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))
#define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))

/* Generates all the necessary prime numbers and marks them in base[]*/
void sieve()
{
    unsigned i, j, k;
    for(i=3; i<LMT; i+=2)
        if(!chkC(base, i))
            for(j=i*i, k=i<<1; j<MAX; j+=k)
                setC(base, j);
    for(i=3, j=0; i<MAX; i+=2)
        if(!chkC(base, i))
            primes[j++] = i;
}

/* Returns the prime-count within range [a,b] and marks them in segment[]
*/
int segmented_sieve(int a, int b)
{
    unsigned i, j, k, cnt = (a<=2 && 2<=b)? 1 : 0;
    if(b<2) return 0;
    if(a<3) a = 3;
    if(a%2==0) a++;
    mset(segment,0);
    for(i=0; sq(primes[i])<=b; i++)
```

Zobayer Hasan
Dhaka, Bangladesh
Undergraduate Student
School: CSE-DU
View my complete profile

SUBSCRIBE
Posts
Comments

BLOG HITS
39207

BLOG ARCHIVE
► 2011 (15)
► 2010 (33)
▼ 2009 (27)
► December (11)
► November (4)
► October (1)
▼ September (2)
SPOJ Solve list comparison tool
Segmented Sieve
► August (1)
► July (8)

CATAGORIES
academic study (15) access modifiers (1) algorithm (28) bash (1) beginner (17) bfs (1) bigint (1) binary tree (1) bitwise (4) blogger (5) bpm (2) brainfuck (1) bst (1) c (1) c++ (36) changes (1) character device driver (1) combinatorics (2) comparator (1) compression (1) computational geometry (2)

```
    {
        j = primes[i] * ( (a+primes[i]-1) / primes[i] );
        if(j%2==0) j += primes[i];
        for(k=primes[i]<<1; j<=b; j+=k)
            if(j!=primes[i])
                setC(segment, (j-a));
    }
    for(i=0; i<=b-a; i+=2)
        if(!chkC(segment, i))
            cnt++;
    return cnt;
}
```

This is a sample program which demonstrates segmented sieve. Very fast and memory efficient version. 'base' is the array which holds the flags for all the primes upto $\sqrt{(2^{31}-1)}$, i.e. the square-root of the max limit, and all the primes are stored in the 'primes' array. Later, these primes are used to determine whether a number is a composite or not within a certain range in the segmented sieve. To avoid overflow and sign bit problems, unsigned type is used.

A little explanation:

First of what what these macros mean?
#define MAX 46656
#define LMT 216
#define LEN 4830
#define RNG 100032

MAX is the sqrt of maximum possible input, in case of here, the maximum is integer range sqrt of which is almost MAX used here. So, MAX is not maximum allowed input, it is just sqrt of maximum input which is pretty big as 2147483647 i.e. 32 bit signed integer maximum.
LMT is sqrt of MAX. We all know, we run sieve upto sqrt MAX
LEN is the maximum possible different primes that can be stored using this algorithm with specific range defined as **RNG**, on which the segmented sieve will run and collect the primes out of it.

Now the next two vital macros:
#define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))
#define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))
And why we divide by 64:

Ok, yes, it is clearly bit shifting. But you must know what we do in bitwise sieve first in order to get this. Instead of using a whole array position to store just one flag, we can use its each 32 bits to store one flag, which saves memory by a factor 1/32. So its very logical to capture memory upto MAX/32, but why MAX/64 and RNG/64 here?
Because, we really have no point of handling the even number as 2 is the only even prime and we can handle it manually, without any stress. So, if we do not consider the even numbers at all, the total numbers are again reduced by a factor 1/2, isn't it? So what we get total is MAX/32/2 = MAX/64, same for RNG.

Now, the two macros chkC and setC is pretty straight forward. chkC checks if a specific bitflag is 1 or 0, and setC sets a specific bitflag 1 to mark it as a composite. They work similarly, so I will explain only the chkC part.

In bitwise sieve, where is a specific value n located? Obviously $(n/32)^{th}$ index, and on that index, $(n\%32)^{th}$ bit from LSB (right hand side). But we just said before, we are interested with only odd numbers, so we map n with n/2 as follows:

```
Actual numbers
 1   2   3   4   5   6   7   8   9 ...........   n     ;[n is odd]
 |   |   |   |   |   |   |   |   | ...........   |
 0   x   1   x   2   x   3   x   4 ........... (n/2)
Position on which they are represented in the bitstrings.
```

So, n is actually n/2, which implies the previous statement: n's (actually n/2 's) position is in index [n/2/32] = [n/64] = [n>>6] and on the bit position (n/2)%32 = (n>>1)&31 ;[We know, modding with a power of 2 is same as ANDing with (same power of 2)-1]. The rest is, how we check / set this specific bit. I have another post explaining these operations: Bitwise operations in C: Part 3, and obviously you can google it :)

<3 <3 th!2 1st.

Posted by Zobayer Hasan at 1:59 AM

10 comments:

fushar said...

Thanks! After reading this article I managed to get AC on SPOJ's PRIME1.

September 27, 2009 10:42 AM

Arun Chauhan said...

great code dude...
but i have some problems understanding it....
if you don't mind can you explain me.... some of the following things....
i want help in understanding the
#define chkC(x,n) (x[n>>6]&(1<<((n>>1)&31)))
#define setC(x,n) (x[n>>6]|=(1<<((n>>1)&31)))

i know they are using bitshift and operators..... but what exactly they are trying to do can
somebody make me understand.... why is it needed to divide n by 64 and why are they ANDing (n/2) with 31..... and why MAX=46656 and RNG=100032 and LEN=4830....
LMT is ok b'cos it's sqrt(MAX)

thanx in advance....

January 8, 2010 12:48 AM

Zobayer Hasan said...

@Arun, I have added the description for the macros. I think you already got it as 1 and a half months passed and I was really busy. :(

March 10, 2010 12:44 AM

Anonymous said...

For anyone who is wondering what the macros are doing, shifting to right (c>>n) divides c by 2^n. So 32>>2 divides 32 by 2^2(=4), which results in 8. So, 32>>2 = 8. the bit shifting in those macros divides by 64 to make the address aligned with the word size of the machine(64-bits). The AND operator is a fast modulus operation for power of two numbers. As long as Y is some power of two(2^n, 2,4,8,16,32,64,128, etc...) X%Y can be written as X&Y-1. So, for example N%32 can be written as N&31. Wikipedia is a golden source of information, and google helps too.

September 13, 2010 9:48 PM

Anonymous said...

How do you print out the primes in the segment? Can you also post the main function of the program pls?

November 20, 2010 5:54 AM

Zobayer Hasan said...

@Anonymous, you see the last loop in segmented sieve function? That is counting number of primes in the interval [a, b] mapping a -> 0 and b -> b-a.

So, when the if is true, (you were counting i) just print a + i.
That should do, and play around it a bit more. As you can do it in a more compact way.

November 20, 2010 12:58 PM

Anonymous said...

Yes, I saw (print a+i), but as you said I needed to put some extra checks. Thank you, after some more work I made it on PRIME1.

November 20, 2010 6:08 PM

Kumar said...

can u please explain the for loop in the segmented_sieve function??

July 11, 2011 3:08 PM

Anonymous said...

Can u please explain the first for loop in the segmented_sieve function??

July 11, 2011 3:10 PM

Zobayer Hasan said...

@Kumar, how do we check if a number is prime or not? isn't it sufficient to try to divide it primes below its square root?

July 11, 2011 6:51 PM

Post a Comment

Comment as:

Post Comment Preview

Subscribe to: Post Comments (Atom)

I am only one, but still I am one.
I cannot do everything, but still I can do something.
And because I cannot do everything I will not refuse to do the something that I can do.
{Helen Keller}