CodeChef Discussion

Computing Factorials of a huge number in C/C++: A tutorial

50

11

Hello @all,

As in MARCH13 contest we needed to use primary school arithmetics once again, and as it is a topic that comes up quite frequentely here in the forums thanks to this problem, and also, as I don't see a complete and detailed tutorial on this topic here, I decided I'd write one before my Lab class at university :P (spending free time with Codechef is always a joy!!)

- **Some preliminary experiences with C**

If we want to implement a factorial calculator efficientely, we need to know what we are dealing with in the first place...

Some experiences in computing factorials iteratively with the following code:

```
#include <stdio.h>

long long int factorial(int N)
{
    long long ans = 1;
    int i;
    for(i=1; i <= N; i++)
    ans *= i;
    return ans;
}
```

Follow this question

By Email:

Once you sign in you will be able to subscribe for any updates here

By RSS:

Answers

Answers and Comments

Tags:

factorial ×33

bignum ×11

tutorial ×9

Asked: **13 Mar '13, 21:12**

Seen: **18,290 times**

Last updated: **13 Mar, 18:37**

Related questions

Factorials for large numbers

Small factorials in C#

```
int main()
{
    int t;
    for(t=1; t <= 21; t++)
    {
        printf("%lld\n", factorial(t));
    }
    return 0;
}
```

will produce the following output on Ideone:

```
1
2
6
24
120
720
5040
40320
362880
3628800
39916800
479001600
6227020800
87178291200
1307674368000
20922789888000
355687428096000
6402373705728000
121645100408832000
2432902008176640000
-4249290049419214848
```

So, we can now see that even when using the long long data type, the maximum factorial we can expect to compute correctly, is only 20!

When seen by this point of view, suddenly, 100! seems as an impossible limit for someone using C/C++, and this is actually **only partially true**, such that we can say:

It is impossible to compute factorials larger than 20 when using built-in data types.

However, the beauty of algorithms arises on such situations... After all, if long long data type is the largest built-in type available, how can people get AC solutions in C/C++? (And, as a side note, how the hell are those "magic" BigNum

Factorisation in c++

A tutorial on Suffix Arrays

Please help me with this code for Small Factorial

FCTRL - TLE

An article on Hashing - opinions required

FCTRL: Getting compile error

factorial(n) /factorial(r1) factorial(r2)...factorial(rk)

A tutorial on the Extended Euclid's Algorithm

and variable precision arithmetic libraries implemented?).

The answer to these questions is surprisingly and annoyingly "basic" and "elementar", in fact, we shall travel back to our primary school years and apply what was taught to most of us when we were 6/7/8 years old.

I am talking about doing all operations by hand!!

- **The underlying idea behind long operations and how to map it into a programming language**

12*11 = 132

Any programming language will tell you that. But, so will any 8 year old kid that's good with numbers. But, the kid's way of telling you such result is what we are interested in:

Here's how he would do it:

```
      12
    x 11
  ---------
      12
    +12
 ----------
     132
```

But, why is this idea way more interesting than simply doing it straighforwardly? It even looks harder and more error-prone... But, it has a fundamental property that we will exploit to its fullest:

The intermediate numbers involved on the intermediate calculations never exceed 81

This is because it is the largest product possible of two 1-digit numbers (9*9 = 81), and these numbers,well, we can deal with them easily!!

The main idea now is to find a suitable data structure to store all the intermediate results and for that we can use an array:

Say **int a[200]** is array where we can store 200 1-digit numbers. (In fact, at each position we can store an integer, but we will only store 1 digit for each position.)

So, we are making good progress!! We managed to understand two important things:

- Primary school arithmetic can prove very useful on big number problems;
- We can use all built-in structures and data-types to perform calculations;

Now, comes up a new question:

How can we know the length of such a huge number? We can store an array and 200 positions, but, our big number may have only 100 digits for example.

The trick is to use a variable that will save us, at each moment, the number of digits that is contained in the array. Let's call it **m**.

Also, since we want only one digit to be stored in every position of array, we need to find a way to "propagate" the carry of larger products to higher digits and sum it afterwards. Let's call the variable to hold the carry, **temp**.

1. **m** -> Variable that contains the number of digits in the array in any given moment;
2. **temp** -> Variable to hold the "carry" value resultant of multiplying digits whose product will be larger than 9. (8*9 = 72, we would store 2 on one array position, and 7 would be the "carry" and it would be stored on a different position.)

So, now that we have an idea on how to deal with the multiplications, let's work on mapping it into a programming language.

- **Coding our idea and one final detail**

Now, we are ready to code our solution for the FCTRL2 problem.

However, one last remark needs to be done:

How do we store a number in the array, and why do we store it the way we do?

If after reading this tutorial you look at some of the accepted solutions in C/C++ for this problem, you will see that contestants actually stored the numbers "backwards", for example:

123 would be saved in an array, say a, as:

a = [3,2,1];

This is done such that when the digit by digit calculations are being performed, the "carry" can be placed on the positions of the array with higher index. This way, we are sure that carry is computed and placed correctly on the array.

Also, computing the products this way and maintaining the variable, **m**, allows us to print the result directly, by looping from a[m-1] until a[0].

As an example, I can leave here an implementation made by @upendra1234, that I took the liberty to comment for a better understanding:

```
#include<stdio.h>
int main()
{
    int t;
    int a[200]; //array will have the capacity to store 200 digits.
    int n,i,j,temp,m,x;

    scanf("%d",&t);
    while(t--)
    {
       scanf("%d",&n);
```

```
        a[0]=1;  //initializes array with only 1 digit, the digit 1.
        m=1;    // initializes digit counter

        temp = 0; //Initializes carry variable to 0.
        for(i=1;i<=n;i++)
        {
             for(j=0;j<m;j++)
             {
                x = a[j]*i+temp; //x contains the digit by digit product
                a[j]=x%10; //Contains the digit to store in position j
                temp = x/10; //Contains the carry value that will be stored on
later indexes
             }
              while(temp>0) //while loop that will store the carry value on array.
              {
                a[m]=temp%10;
                temp = temp/10;
                m++; // increments digit counter
              }
       }
               for(i=m-1;i>=0;i--) //printing answer
               printf("%d",a[i]);
               printf("\n");
    }
    return 0;
}
```

I hope this tutorial can help someone to gain a better understanding of this subject and that can help some people as it is why we are here for :D

Best Regards,

Bruno Oliveira

EDIT: As per @betlista comment, it's also worth pointing out that, since we keep only a single digit at each position on the array, we could have used the data-type char instead of int. This is because internally, a char is actually an integer that only goes in the range 0 - 255 (values used to represent the ASCII codes for all the characters we are used to see). The gains would be only memory-wise.

factorial bignum tutorial

This question is marked "community wiki".

wikified 29 Mar '13, 13:51

balajiganapath ▯▯
565●3●14●21

asked 13 Mar '13, 21:12

kuruma
13.3k●57●127●186
accept rate: 7%

When digits are stored in the form of ascii code then digits 0-9 can't be treated as 0-9. Digits will be treated as 48-57. To use digit 1 as number 1 if it is stored as char then 48 should be subtracted from ascii value of 1 i.e 49. 49-48=1

ritujain1971 (01 Aug '13, 12:23)

---

5 Answers:

oldest newest **most voted**

12

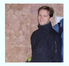
Just a small tip (a got it too, I'm not the author), you do not need to have digits in your a array ;-) If you want to use digits, using char array is more space efficient...

Let say you want to find the result of 98*76

```
      9 8
x     7 6
---------
    54 48
63  56
=========
63 110 48
~~~~~~~~~ (mod 10)
        8 48 % 10
     4    (110+4)%10
74        63+11
=========
7 4  4  8
```

link

answered **13 Mar '13, 22:04**



betlista □□
11.5k●41●95●184
accept rate: 9%

1 Yes, using a char array instead of an int array, since we are only storing digits and not numbers, would make more sense when talking about a memory efficient code.

On this case, I chose clarity over efficiency, as I believe that for a newbie that reads this tutorial, introducing the idea that a char is actually a very small int could be unnecessary complicated :)

kuruma (14 Mar '13, 00:17)

1

And I took the liberty to fork the above code to find out very large powers of n. :D

//Code to store very large powers of 2.

```
#include<stdio.h>
int main()
```

```
{
    int t;
    int a[1000]; //array will have the capacity to store 1000 digits.
    int n,i,j,temp,m,x;

    scanf("%d",&t);
    while(t--)
    {
       scanf("%d",&n);// n is the power.
       a[0]=1;  //initializes array with only 1 digit, the digit 1.
       m=1;    // initializes digit counter
       i=2;// i is base 2
       int k=1;//k is a counter that goes from 1 to  n.
       temp = 0; //Initializes carry variable to 0.
       while(k<=n)
       {
            for(j=0;j<m;j++)
            {
               x = a[j]*i+temp; //x contains the digit by digit product
               a[j]=x%10; //Contains the digit to store in position j
               temp = x/10; //Contains the carry value that will be stored on
later indexes
            }
             while(temp>0) //while loop that will store the carry value on array.
             {
               a[m]=temp%10;
               temp = temp/10;
               m++; // increments digit counter
             }
             k++;
      }
              for(i=m-1;i>=0;i--) //printing answer
              printf("%d",a[i]);
              printf("\n");

    }
    return 0;
}
```

link

answered **14 Nov '13, 11:38**

saikatkumardey
16
accept rate: 0%

0

n! - simply you can find factorial for any number........ mine on C#

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace BigMultiplier
{
    class Program
    {

        static void Main(string[] args)
        {
            int[] s1 = new int[1];
            int[] s2 = new int[1];

            s1[0]=1;
            Program p = new Program();

            int[] s3 = p.doit(s1, s2);
            Console.WriteLine("Enter the Number for Which Factorial to be Found
(below 1000)");
            int limit = Convert.ToInt32(Console.ReadLine());



            for (int i = 1; i <= limit; i++)
            {
                s3 = p.return_array(i);
                s1 = p.doit(s1, s3);
            }
            int sum = 0;
            for (int j = s1.Length-1; j >=0; j--)
            {
                Console.Write(s1[j]);
                sum = sum + s1[j];
            }
            Console.Write("sum = "+sum);
            Console.WriteLine();
            Console.ReadLine();
        }

        int[] return_array(int num)
        {
            int[] num_arr = new int[1]; ;

            if (num <= 9)
            {
                num_arr = new int[1];
                num_arr[0] = num;
            }
            else if (num <= 99)
            {
                num_arr = new int[2];
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num;
            }
            else if (num <= 999)
            {
                num_arr = new int[3];
                num_arr[2] = num % 10;
                num = num / 10;
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num % 10;
            }
            else if (num <= 9999)
            {
                num_arr = new int[4];
                num_arr[3] = num % 10;
                num = num / 10;
                num_arr[2] = num % 10;
                num = num / 10;
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num % 10;
            }
            else if (num <= 99999)
            {
                num_arr = new int[5];
                num_arr[4] = num % 10;
                num = num / 10;
                num_arr[3] = num % 10;
                num = num / 10;
                num_arr[2] = num % 10;
                num = num / 10;
                num_arr[1] = num % 10;
                num = num / 10;
                num_arr[0] = num % 10;
            }
            return num_arr;
        }



        int[] doit(int[] num1_int, int[] num2_int)
        {
//              String num1_string, num2_string;
//             Console.WriteLine("Enter the Number 1");
//             num1_string = Console.ReadLine();
//             Console.WriteLine("Enter the Number 2");
//             num2_string = Console.ReadLine();

//             int[] num1_int = new int[num1_string.Length];
//             int[] num2_int = new int[num2_string.Length];

//             for (int j = 0; j < num1_string.Length; j++)
//             {
//                 num1_int[j] = num1_string[j]-48;
//                 Console.Write(" " + num1_int[j]);
//             }

//             for(int j=0;j<num2_string.Length;j++)
//             {
//                 num2_int[j] = num2_string[j]-48;
//                 Console.Write(" " + num2_int[j]);
//             }

            int[,] num3_int = new int[num2_int.Length, (num1_int.Length + 1)];
            int i,k,temp=0;

            //Multiplication on Individual Digits Done and the Values are there in
the Individual Cells of the Array
            for(i=0;i<num2_int.Length;i++)
            {
                for (k = 0; k < num1_int.Length; k++)
                {
                    int mul = (num1_int[k] * num2_int[i])+temp;
                    num3_int[i, k] = mul % 10;
                    temp = mul / 10;
                    if (k == (num1_int.Length - 1))
                    {
                        num3_int[i, k+1] = temp;
                    }
//                      Console.Write(" " + num3_int[i, k]);
                }
//                  Console.Write(" " + num3_int[i, k]);
                    temp=0;
//                      Console.WriteLine();
            }

//              temp = 0;

//              Console.ReadLine();
            int[] result_int = new int[num1_int.Length + num2_int.Length];

//              int[,] num3_int = new int[num2_string.Length, (num1_string.Length +
1)];

            double result=0;

            for(i=0;i<num2_int.Length;i++)
                for (k = 0; k < (num1_int.Length + 1); k++)
                {
//                      Console.Write(" i = " + i + " k=" + k+"   ");
//                      Console.Write(num3_int[i, k]);
                    result=result+(num3_int[i,k]*Math.Pow(10,k)*Math.Pow(10,i));
//                      Console.WriteLine("
"+(num3_int[i,k]*Math.Pow(10,k)*Math.Pow(10,i)));
//                      Console.WriteLine("10^k  " + Math.Pow(10,k));



                }

            int[,] num4_int= new int[num2_int.Length,num1_int.Length +
num2_int.Length];
```

```
            for (i = 0; i < num2_int.Length; i++)
            {
                for (k = 0; k < (num1_int.Length + 1); k++)
                {
                    //for (int l = 0; l < 0; l++)
                    {
                        num4_int[i,k + i] = num3_int[i,k];
                    }
                }
            }

            for (i = 0; i < num2_int.Length; i++)
            {
                for (k = 0; k < (num1_int.Length + num2_int.Length); k++)
                {
//                    Console.Write(" " + num4_int[i, k]);
                }
//                Console.WriteLine();
            }

            int[] re_int = new int[num1_int.Length + num2_int.Length];
            temp=0;
            for (i = 0; i < re_int.Length; i++)
            {
                re_int[i] = 0;
                for (k = 0; k < num2_int.Length; k++)
                {
                    re_int[i] = num4_int[k, i] + re_int[i];
                }
                int t = re_int[i] + temp;
                re_int[i]=t%10;
                temp=t/10;
                if(i==(re_int.Length-1))
                {
                    //Need to check
                }
            }

//            Console.WriteLine("Final Result - Reversed Order ");
//            for(i=0;i<re_int.Length;i++)
 //               Console.Write(" "+re_int[i]);
```

```
            //          re_int[i+k]=;

//            Console.WriteLine("Final Result - Correct Order ");
 //           for(i=re_int.Length-1;i>=0;i--)
 //               Console.Write(" " + re_int[i]);

//            Console.WriteLine(result);
//            Console.ReadLine();
            return re_int;
        }
    }
}
```

link

edited 13 Mar, 18:36
betlista
11.5k●41●95●184

answered 29 Mar '13, 11:01
gopaltirupur
16●1
accept rate: 0%

that `return_array` method is really ugly!!!

betlista (13 Mar, 18:37)

0

What about last ten non zero digit of 10^14 factorial ??

link

answered 17 Aug '13, 17:46
may3
28●2●6
accept rate: 0%

-1

THANK YOU SO MUCH TO DESCRIBE ME

link

answered 13 Dec '13, 01:26
guptanitin
0
accept rate: 0%

?

[hide preview]

☐ community wiki

Post Your Answer