Program for Fibonacci numbers

The Fibonacci numbers are the numbers in the following integer sequence.

0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, ……..

In mathematical terms, the sequence Fn of Fibonacci numbers is defined by the recurrence relation

$$F_n = F_{n-1} + F_{n-2}$$

with seed values

$$F_0 = 0 \text{ and } F_1 = 1.$$

Recommended: Please solve it on "*PRACTICE* " first, before moving on to the solution.

Write a function *int fib(int n)* that returns F_n. For example, if *n* = 0, then *fib()* should return 0. If n = 1, then it should return 1. For n > 1, it should return $F_{n-1} + F_{n-2}$

```
For n = 9
Output:34
```

Following are different methods to get the nth Fibonacci number.

Method 1 (Use recursion)

A simple method that is a direct recursive implementation mathematical recurrence relation given above.

C

```
//Fibonacci Series using Recursion
```

```
#include<stdio.h>
int fib(int n)
{
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Run on IDE

Java

```
//Fibonacci Series using Recursion
class fibonacci
{
    static int fib(int n)
    {
    if (n <= 1)
       return n;
    return fib(n-1) + fib(n-2);
    }

    public static void main (String args[])
    {
    int n = 9;
    System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

Run on IDE

Python

```
# Function for nth Fibonacci number

def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 0
    elif n==1:
        return 0
    # Second Fibonacci number is 1
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

# Driver Program

print(Fibonacci(9))

#This code is contributed by Saket Modi
```

Run on IDE

Output

```
34
```

Time Complexity: T(n) = T(n-1) + T(n-2) which is exponential.

We can observe that this implementation does a lot of repeated work (see the following recursion tree). So this is a bad implementation for nth Fibonacci number.

```
                         fib(5)
                     /
              fib(4)                fib(3)
            /                      /
        fib(3)      fib(2)         fib(2)    fib(1)
       /             /           /
  fib(2)   fib(1)  fib(1) fib(0) fib(1) fib(0)
  /
fib(1) fib(0)
```

Extra Space: O(n) if we consider the function call stack size, otherwise O(1).

Method 2 (Use Dynamic Programming)

We can avoid the repeated work done is the method 1 by storing the Fibonacci numbers calculated so far.

C

```c
//Fibonacci Series using Dynamic Programming
#include<stdio.h>

int fib(int n)
{
  /* Declare an array to store Fibonacci numbers. */
  int f[n+1];
  int i;

  /* 0th and 1st number of the series are 0 and 1*/
  f[0] = 0;
  f[1] = 1;

  for (i = 2; i <= n; i++)
  {
      /* Add the previous 2 numbers in the series
         and store it */
      f[i] = f[i-1] + f[i-2];
  }

  return f[n];
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Run on IDE

Java

```
// Fibonacci Series using Dynamic Programming
class fibonacci
{
   static int fib(int n)
    {
        /* Declare an array to store Fibonacci numbers. */
    int f[] = new int[n+1];
    int i;

    /* 0th and 1st number of the series are 0 and 1*/
    f[0] = 0;
    f[1] = 1;

    for (i = 2; i <= n; i++)
    {
       /* Add the previous 2 numbers in the series
         and store it */
        f[i] = f[i-1] + f[i-2];
    }

    return f[n];
    }

    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

Run on IDE

Python

```
# Function for nth fibonacci number - Dynamic Programing
# Taking 1st two fibonacci nubers as 0 and 1

FibArray = [0,1]

def fibonacci(n):
    if n<0:
        print("Incorrect input")
    elif n<=len(FibArray):
        return FibArray[n-1]
    else:
        temp_fib = fibonacci(n-1)+fibonacci(n-2)
        FibArray.append(temp_fib)
        return temp_fib

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi
```

Run on IDE

Output:

```
34
```

Time Complexity: O(n)

Extra Space: O(n)

Method 3 (Space Optimized Method 2)

We can optimize the space used in method 2 by storing the previous two numbers only because that is all we need to get the next Fibonacci number in series.

C/C++

```
// Fibonacci Series using Space Optimized Method
#include<stdio.h>
int fib(int n)
{
  int a = 0, b = 1, c, i;
  if( n == 0)
    return a;
  for (i = 2; i <= n; i++)
  {
     c = a + b;
     a = b;
     b = c;
  }
  return b;
}

int main ()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Run on IDE

Java

```
// Java program for Fibonacci Series using Space
// Optimized Method
class fibonacci
{
    static int fib(int n)
    {
        int a = 0, b = 1, c;
        if (n == 0)
            return a;
        for (int i = 2; i <= n; i++)
        {
            c = a + b;
            a = b;
            b = c;
        }
        return b;
    }
```

```
    public static void main (String args[])
    {
        int n = 9;
        System.out.println(fib(n));
    }
}

// This code is contributed by Mihir Joshi
```

Run on IDE

Python

```python
# Function for nth fibonacci number - Space Optimisataion
# Taking 1st two fibonacci numbers as 0 and 1

def fibonacci(n):
    a = 0
    b = 1
    if n < 0:
        print("Incorrect input")
    elif n == 0:
        return a
    elif n == 1:
        return b
    else:
        for i in range(2,n):
            c = a + b
            a = b
            b = c
        return b

# Driver Program

print(fibonacci(9))

#This code is contributed by Saket Modi
```

Run on IDE

Time Complexity: O(n)
Extra Space: O(1)

Method 4 (Using power of the matrix {{1,1},{1,0}})

This another O(n) which relies on the fact that if we n times multiply the matrix M = {{1,1},{1,0}} to itself (in other words calculate power(M, n)), then we get the (n+1)th Fibonacci number as the element at row and column (0, 0) in the resultant matrix.

The matrix representation gives the following closed expression for the Fibonacci numbers:

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

C

```c
#include <stdio.h>

/* Helper function that multiplies 2 matrices F and M of size 2*2, and
  puts the multiplication result back to F[][] */
void multiply(int F[2][2], int M[2][2]);

/* Helper function that calculates F[][] raise to the power n and puts the
  result in F[][]
  Note that this function is designed only for fib() and won't work as general
  power function */
void power(int F[2][2], int n);

int fib(int n)
{
  int F[2][2] = {{1,1},{1,0}};
  if (n == 0)
      return 0;
  power(F, n-1);

  return F[0][0];
}

void multiply(int F[2][2], int M[2][2])
{
  int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
  int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
  int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
  int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

  F[0][0] = x;
  F[0][1] = y;
  F[1][0] = z;
  F[1][1] = w;
}

void power(int F[2][2], int n)
{
  int i;
  int M[2][2] = {{1,1},{1,0}};

  // n - 1 times multiply the matrix to {{1,0},{0,1}}
  for (i = 2; i <= n; i++)
      multiply(F, M);
}

/* Driver program to test above function */
int main()
{
  int n = 9;
  printf("%d", fib(n));
  getchar();
  return 0;
}
```

Run on IDE

Java

```java
class fibonacci
{

    static int fib(int n)
    {
    int F[][] = new int[][]{{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);
```

```
        return F[0][0];
    }

     /* Helper function that multiplies 2 matrices F and M of size 2*2, and
     puts the multiplication result back to F[][] */
    static void multiply(int F[][], int M[][])
    {
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
    }

    /* Helper function that calculates F[][] raise to the power n and puts the
    result in F[][]
    Note that this function is designed only for fib() and won't work as general
    power function */
    static void power(int F[][], int n)
    {
    int i;
    int M[][] = new int[][]{{1,1},{1,0}};

    // n - 1 times multiply the matrix to {{1,0},{0,1}}
    for (i = 2; i <= n; i++)
        multiply(F, M);
    }

    /* Driver program to test above function */
    public static void main (String args[])
    {
    int n = 9;
    System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

Run on IDE

Time Complexity: O(n)
Extra Space: O(1)

Method 5 (Optimized Method 4)

The method 4 can be optimized to work in O(Logn) time complexity. We can do recursive multiplication to get power(M, n) in the prevous method (Similar to the optimization done in this post)

C

```
#include <stdio.h>

void multiply(int F[2][2], int M[2][2]);

void power(int F[2][2], int n);
```

```
/* function that returns nth Fibonacci number */
int fib(int n)
{
  int F[2][2] = {{1,1},{1,0}};
  if (n == 0)
    return 0;
  power(F, n-1);
  return F[0][0];
}

/* Optimized version of power() in method 4 */
void power(int F[2][2], int n)
{
  if( n == 0 || n == 1)
      return;
  int M[2][2] = {{1,1},{1,0}};

  power(F, n/2);
  multiply(F, F);

  if (n%2 != 0)
     multiply(F, M);
}

void multiply(int F[2][2], int M[2][2])
{
  int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
  int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
  int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
  int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

  F[0][0] = x;
  F[0][1] = y;
  F[1][0] = z;
  F[1][1] = w;
}

/* Driver program to test above function */
int main()
{
  int n = 9;
  printf("%d", fib(9));
  getchar();
  return 0;
}
```

Run on IDE

Java

```
//Fibonacci Series using  Optimized Method
class fibonacci
{
    /* function that returns nth Fibonacci number */
    static int fib(int n)
    {
    int F[][] = new int[][]{{1,1},{1,0}};
    if (n == 0)
        return 0;
    power(F, n-1);

    return F[0][0];
    }

    static void multiply(int F[][], int M[][])
    {
```

```
    int x =  F[0][0]*M[0][0] + F[0][1]*M[1][0];
    int y =  F[0][0]*M[0][1] + F[0][1]*M[1][1];
    int z =  F[1][0]*M[0][0] + F[1][1]*M[1][0];
    int w =  F[1][0]*M[0][1] + F[1][1]*M[1][1];

    F[0][0] = x;
    F[0][1] = y;
    F[1][0] = z;
    F[1][1] = w;
    }

    /* Optimized version of power() in method 4 */
    static void power(int F[][], int n)
    {
    if( n == 0 || n == 1)
      return;
    int M[][] = new int[][]{{1,1},{1,0}};

    power(F, n/2);
    multiply(F, F);

    if (n%2 != 0)
       multiply(F, M);
    }

    /* Driver program to test above function */
    public static void main (String args[])
    {
         int n = 9;
     System.out.println(fib(n));
    }
}
/* This code is contributed by Rajat Mishra */
```

Run on IDE

Time Complexity: **O(Logn)**

Extra Space: O(Logn) if we consider the function call stack size, otherwise O(1).

Method 6 (O(Log n) Time)

Below is one more interesting recurrence formula that can be used to find n'th Fibonacci Number in O(Log n) time.

```
If n is even then k = n/2:
F(n) = [2*F(k-1) + F(k)]*F(k)

If n is odd then k = (n + 1)/2
F(n) = F(k)*F(k) + F(k-1)*F(k-1)
```

How does this formula work?

The formula can be derived from above matrix equation.

$$\begin{pmatrix} 1 & 1 \\ 1 & 0 \end{pmatrix}^n = \begin{pmatrix} F_{n+1} & F_n \\ F_n & F_{n-1} \end{pmatrix}.$$

Taking determinant on both sides, we get

$(-1)^n = F_{n+1}F_{n-1} - F_n^2$

Moreover, since $A^nA^m = A^{n+m}$ for any square matrix A, the following identities can be derived (they are obtained form two different coefficients of the matrix product)

$F_mF_n + F_{m-1}F_{n-1} = F_{m+n-1}$

By putting n = n+1,

$F_mF_{n+1} + F_{m-1}F_n = F_{m+n}$

Putting m = n

$F_{2n-1} = F_n^2 + F_{n-1}^2$

$F_{2n} = (F_{n-1} + F_{n+1})F_n = (2F_{n-1} + F_n)F_n$ (Source: Wiki)

To get the formula to be proved, we simply need to do following

If n is even, we can put k = n/2

If n is odd, we can put k = (n+1)/2

Below is the implementation of above idea.

C++

```
// C++ Program to find n'th fibonacci Number in
// with O(Log n) arithmatic operations
#include <bits/stdc++.h>
using namespace std;

const int MAX = 1000;

// Create an array for memoization
int f[MAX] = {0};

// Returns n'th fuibonacci number using table f[]
int fib(int n)
{
    // Base cases
    if (n == 0)
        return 0;
    if (n == 1 || n == 2)
        return (f[n] = 1);

    // If fib(n) is already computed
    if (f[n])
        return f[n];

    int k = (n & 1)? (n+1)/2 : n/2;

    // Applyting above formula [Note value n&1 is 1
    // if n is odd, else 0.
    f[n] = (n & 1)? (fib(k)*fib(k) + fib(k-1)*fib(k-1))
           : (2*fib(k-1) + fib(k))*fib(k);

    return f[n];
}

/* Driver program to test above function */
```

```
int main()
{
    int n = 9;
    printf("%d ", fib(n));
    return 0;
}
```

Run on IDE

Java

```
// Java Program to find n'th fibonacci
// Number with O(Log n) arithmetic operations
import java.util.*;

class GFG {

    static int MAX = 1000;
    static int f[];

    // Returns n'th fibonacci number using
    // table f[]
    public static int fib(int n)
    {
        // Base cases
        if (n == 0)
            return 0;

        if (n == 1 || n == 2)
            return (f[n] = 1);

        // If fib(n) is already computed
        if (f[n] != 0)
            return f[n];

        int k = (n & 1) == 1? (n + 1) / 2
                            : n / 2;

        // Applying above formula [Note value
        // n&1 is 1 if n is odd, else 0.
        f[n] = (n & 1) == 1? (fib(k) * fib(k) +
                        fib(k - 1) * fib(k - 1))
                       : (2 * fib(k - 1) + fib(k))
                       * fib(k);

        return f[n];
    }

    /* Driver program to test above function */
    public static void main(String[] args)
    {
        int n = 9;
        f= new int[MAX];
        System.out.println(fib(n));
    }
}

// This code is contributed by Arnav Kr. Mandal.
```

Run on IDE

Output :

```
34
```

Time complexity of this solution is O(Log n) as we divide the problem to half in every recursive call. This method is contributed by Chirag Agarwal.

Related Articles:

Large Fibonacci Numbers in Java

Please write comments if you find the above codes/algorithms incorrect, or find other ways to solve the same problem.

References:

http://en.wikipedia.org/wiki/Fibonacci_number

http://www.ics.uci.edu/~eppstein/161/960109.html

GATE CS Corner   Company Wise Coding Practice

Dynamic Programming Mathematical Fibonacci series

Please write to us at contribute@geeksforgeeks.org to report any issue with the above content.

Recommended Posts:

Tribonacci Numbers
Dynamic Programming | Set 9 (Binomial Coefficient)
Minimum number of jumps to reach end
Dynamic Programming | Set 4 (Longest Common Subsequence)
Dynamic Programming | Set 1 (Overlapping Subproblems Property)
Large Fibonacci Numbers in Java
Minimum sum submatrix in a given 2D array
Maximum sum subsequence with at-least k distant elements
Number of NGEs to the right
Length of Longest Balanced Subsequence

(Login to Rate)

3.1 Average Difficulty : **3.1/5.0**
Based on **174** vote(s)

Basic Easy Medium Hard Expert

Add to TODO List
Mark as DONE

Writing code in comment? Please use ide.geeksforgeeks.org, generate link and share the link here.

Load Comments

Share this post!

 Contact Us! About Us! Careers! Privacy Policy

▲