Erick Vargas

Redes III

Redes III

Contents

1 Programación dinámica

Partimos de una solución recursiva bruta y podemos hacer uso de una función de memorización. El ejemplo más clásico es el de Fibonacci

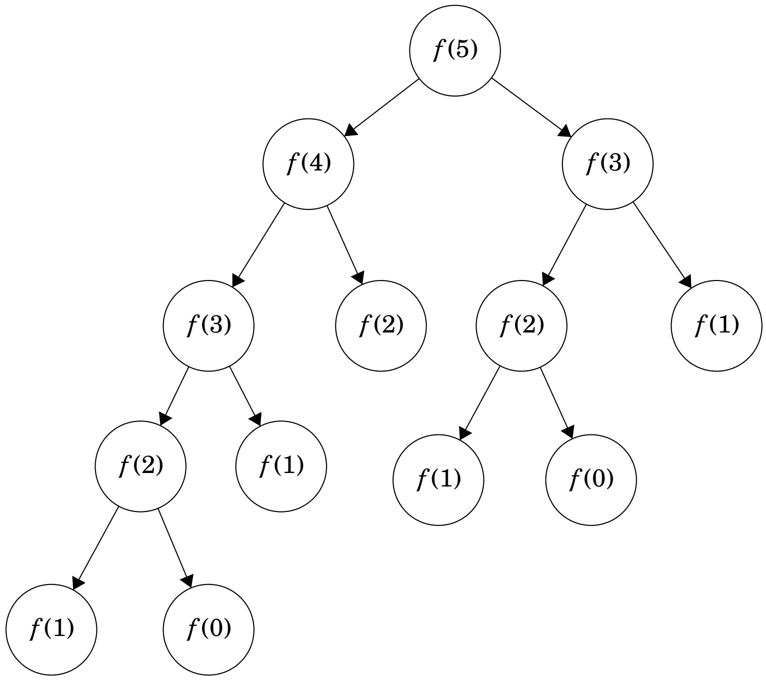
1.1 Fibonacci

$$fibonacci(n) = \begin{cases} 0 & , n = 0 \\ 1 & , n = 1 \\ fibonacci(n-1) + fibonacci(n-2) & , n \geq 2 \end{cases}$$

En código

```
int fibonacci( int n ){
    if( n == 0 )
        return 0;
    if( n == 1 )
        return 1;
    return fibonacci( n - 1 ) * fibonacci( n - 2 );
}
```

Si dibujamos las llamadas recursivas como un árbol tenemos lo siguiente



Como podemos observar hay llamadas recursivas que se repiten varias veces como ejemplo tenemos la llamada recursiva con un valor de 3, o 2. ¿Podemos evitar esto?

```
int memoria[ 100000 ];
.
.
.
int fibonacci( int n ){
    if( n == 0 )
        return 0;
    if( n == 1 )
        return 1;
    if( memoria[ n ] != -1 )
        return memoria[ n ];
    return memoria[ n ] = ( fibonacci( n - 1 ) * fibonacci(
        n - 2 ) );
}

int main(){
    memset( memoria, -1, sizeof(memoria) );
    .
    .
    .
}
```

La complejidad la podemos calcular viendo los estados, en nuestro caso es el tamaño de la memoria, por ejemplo si llamamos a fibonacci de 10 siempre tenemos el mismo resultado, por tanto multiplicamos los estados por la complejidad de la función en nuestro caso es $10^5 * constante$

1.2 Coeficiente binomial

$$Coef\_Bin(n,k) = \begin{cases} 1 & , n = k \\ 1 & , k = 0 \\ Coef\_Bin(n-1,k-1) + Coef\_Bin(n-1,k) & , k \leq n \end{cases}$$

Si programamos esta función tenemos lo siguiente:

```
int coef_bin( int n, int k ){
    if( n == k || k == 0 )
        return 1;
    return ( coef_bin( n - 1, k - 1 ) + coef_bin( n - 1, k )
         );
}
```

Si aplicamos DP

```
memoria[ 1000 ][ 1000 ];
int coef_bin( int n, int k ){
    if( n == k || k == 0 )
        return 1;
    if( mem[ n ][ k ] != -1 )
        return memoria[ n ][ k ];
    return memoria[ n ][ k ] = ( coef_bin( n - 1, k - 1 ) +
        coef_bin( n - 1, k ) );
}
```

En complejidad tenemos $O(n*k)$ y sin la función de memorización tenemos $O(2^n)$