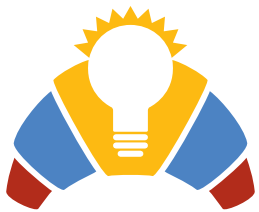
Programación Dinámica
(Introducción a la Programación Dinámica)

Nicolás Ricci

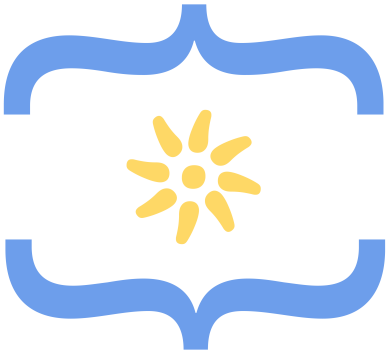
Universidad Nacional de La Plata – Facultad de Informática

Training Camp 2026



Gracias Sponsors!

Organiza



Facultad de INFORMÁTICA

UNIVERSIDAD NACIONAL DE LA PLATA

Diamond Sponsor

Outline

Outline

Empecemos con un problema

Fibonacci

Los números de Fibonacci se definen como:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \quad \text{para } n \geq 2 \end{aligned} \tag{1}$$

Calcular los últimos 6 dígitos de F(2026).

Posible solución (Recursiva)

Fibonacci

Los números de Fibonacci se definen como:

$$\begin{aligned} F(0) &= 0 \\ F(1) &= 1 \\ F(n) &= F(n-1) + F(n-2) \quad \text{para } n \geq 2 \end{aligned} \tag{2}$$

Calcular los últimos 6 dígitos de F(2026).

Posible función

Dado n, quiero calcular F(n)...

- Si n es 1, devolver 1.
- Si n es 2, devolver 2.

Si no, llamar a las dos funciones anteriores, sumarlas, y devolver eso.

Esta solución tiene un problema...

Posible función

Dado n, quiero calcular F(n)...

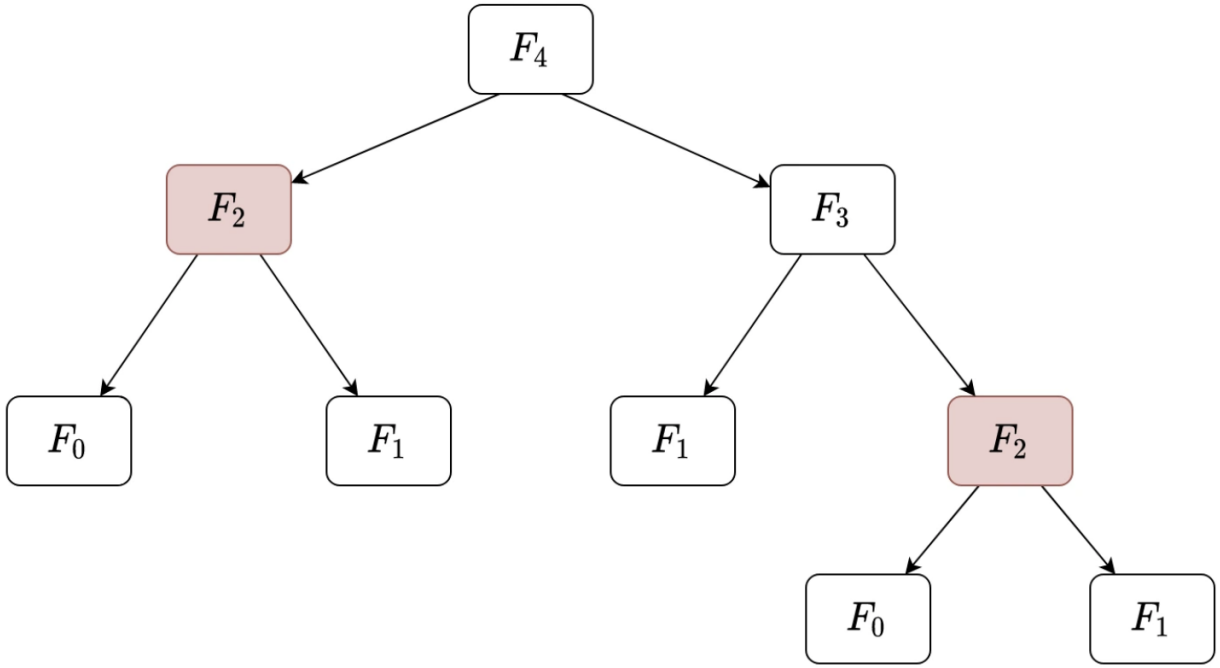
- Si n es 1, devolver 1.
- Si n es 2, devolver 2.

Si no, llamar a las dos funciones anteriores, sumarlas, y devolver eso.

Para calcular F(50) tarda algunos minutos...

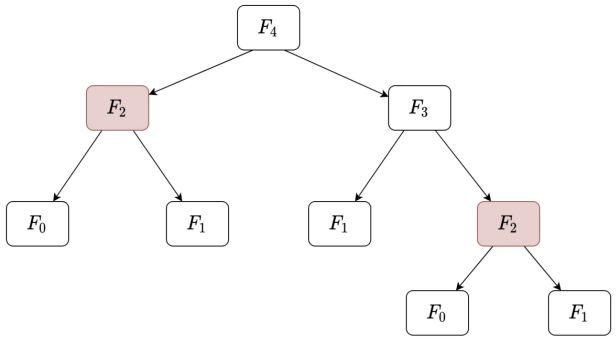
Para calcular F(2026) tarda muchos pero muchos años

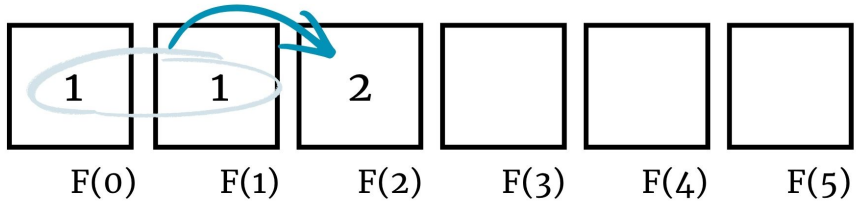
El problema...



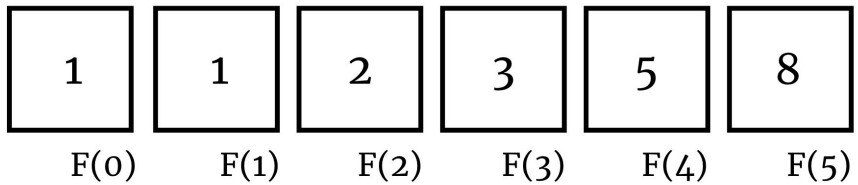
... La solución: Programación Dinámica!

La solución anterior hace (mucho) trabajo innecesario.





En vez de llamar 'desde arriba' a la función, nos podemos ir guardando en una lista los números:



Implementación (Ejemplo de Código)

Código C++

```
#include <bits/stdc++.h>
using namespace std;

int main ()  {

    vector<int> F( 2027, 0 );

    F [ 0 ] = 0;
    F [ 1 ] = 1;

    for ( int i = 2; i <= 2026; i++ ) {
        F [ i ] = F [ i - 1 ] + F [ i - 2 ];

        // nos importan los últimos 6 dígitos:
        F [ i ] %= 1000000;
    }

    cout << "La respuesta es " << F[ 2026 ] << endl;
}
```

Entonces...

La idea es que, a la función F, la convertimos en una fila (vector) donde el i-ésimo elemento es el valor $F(i)$

Entonces, si $F(n)$ depende de los anteriores valores, puedo acceder a estos en $\mathcal{O}(1)$

Esto quiere decir que la solución anterior es $\mathcal{O}(n)$, a diferencia de la primer solución, que es exponencial...

Nota: A este vector F se le llama (informalmente) 'la DP'

Outline

Otro problema

CSES: 'Dice Combinations'

Ema tira dados, los suma, y suma el número n. Contar la cantidad de formas en que puede pasar esto. (En este problema, sí importa el orden de los dados)

Por ejemplo, si $n = 3$, hay 4 formas:

- $1 + 1 + 1$
- $1 + 2$
- $2 + 1$
- 3

Observación recursiva:

Si $f(n)$ es la cantidad de formas de sumar dados para llegar a n...
A partir de $n \geq 7$ se cumple:
$f(n) = f(n-1) + f(n-2) + ... + f(n-6)$

Observación recursiva:

Si $f(n)$ es la cantidad de formas de sumar dados para llegar a n...
A partir de $n \geq 7$ se cumple:
$f(n) = f(n-1) + f(n-2) + ... + f(n-6)$
Además, por comodidad, podemos decir que
$f(0) = 1, f(-1) = 0, f(-2) = 0$... y se cumple la recursión para todos los enteros positivos

Implementación (c++)

Código C++

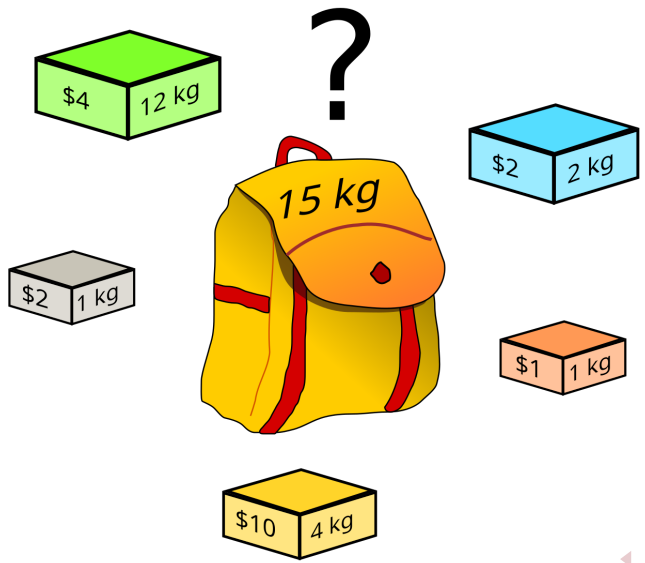
```
#include <bits/stdc++.h>
using namespace std;
using ll = long long; const ll mod = 1e9 + 7;

int main() {

    int n; cin >> n;
    vector<ll> f ( n + 1, 0 );   f[ 0 ] = 1;

    for ( int i = 1; i <= n; i++ ) {
        for ( int v = 1; v <= 6; v++ ) {
            if ( i - v >= 0 ) { f[ i ] += f[ i - v ]; }
        }
        f [ i ] = ( f [ i ] % mod );
    }   cout << f[ n ] << "\n";
}
```

Outline

Problema de la mochila (o Knapsack)

'Knapsack'

En el Louvre hay N objetos, enumerados $1, 2, 3, \ldots, N$.
El ítem i tiene peso w_i, y valor v_i.
Bruno quiere robar algunos de estos objetos, y llevarlos en su mochila, pero su mochila solo puede soportar un peso de W.
Bruno quiere maximizar la suma total de los valores de la mochila.
Encontrar este valor máximo.
Restricciones: $N \leq 400$ $W \leq 400$



Cómo lo resolvemos?

Idea...

Llamamos $DP(i, u)$ al mayor valor que podemos sumar

- Eligiendo sólo entre los primeros i elementos
- En donde sus pesos sumen (exactamente) u

Ejemplo:

Si los objetos son:

- Objeto 1: Peso 2, valor 3000
- Objeto 2: Peso 3, valor 1000
- Objeto 3: Peso 5, valor 5000

Entonces $DP(2, 5) = 4000$ eligiendo los dos primeros, y $DP(3, 5) = 5000$ porque podemos elegir el tercero solamente.

Resolución

Podemos calcular DP(i, u)...

Si $i = 0$, siempre es 0.

Si $i \geq 1$, ahora podemos elegir agregar el objeto i en la mochila o no.

Podemos calcular DP(i, u)...

Si $i = 0$, siempre es 0.

Si $i \geq 1$, ahora podemos elegir agregar el objeto i en la mochila o no.

- Si no lo agregamos, $DP(i, u) = DP(i - 1, u)$.

Resolución

Podemos calcular DP(i, u)...

Si $i = 0$, siempre es 0.

Si $i \geq 1$, ahora podemos elegir agregar el objeto i en la mochila o no.

- Si no lo agregamos, $DP(i, u) = DP(i - 1, u)$.
- Si es que lo agregamos, $DP(i, u)$ se volvería $DP(i - 1, u - w_i) + v_i$, donde w_i es el i-ésimo peso, y v_i es el i-ésimo valor.

Es decir, $DP(i, u)$ va a agarrar el máximo entre estos dos valores

$$(DP(i - 1, u), (DP(i - 1, u - w_i) + v_i)$$

Resolución

Podemos calcular DP(i, u)...

Si $i = 0$, siempre es 0.

Si $i \geq 1$, ahora podemos elegir agregar el objeto i en la mochila o no.

- Si no lo agregamos, $DP(i, u) = DP(i - 1, u)$.
- Si es que lo agregamos, $DP(i, u)$ se volvería $DP(i - 1, u - w_i) + v_i$, donde w_i es el i-ésimo peso, y v_i es el i-ésimo valor.

Es decir, $DP(i, u)$ va a agarrar el máximo entre estos dos valores

$$(DP(i - 1, u), (DP(i - 1, u - w_i) + v_i)$$

Podemos entonces correr un doble for: El primero recorre todos los i del 1 al N; El segundo recorre todos los u del 0 al W.

Outline

Problema: Máxima suma en un camino

Problema

En una grilla de $h \leq 100$ filas y $w \leq 100$ columnas, en cada casilla hay una cantidad de piedras. Cata quiere:

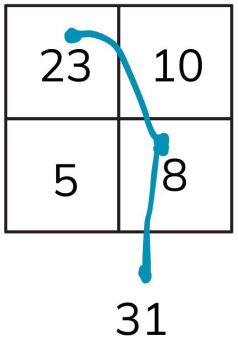
- Empezar en una casilla de la primer fila
- En cada paso, bajar una fila hacia alguna de las tres casillas debajo
- Terminar saliendo de la grilla

En cada paso, agarra las piedras de cada casilla.
Calcular la cantidad de piedras que agarra Cata si es que quiere maximizar su cantidad de piedras

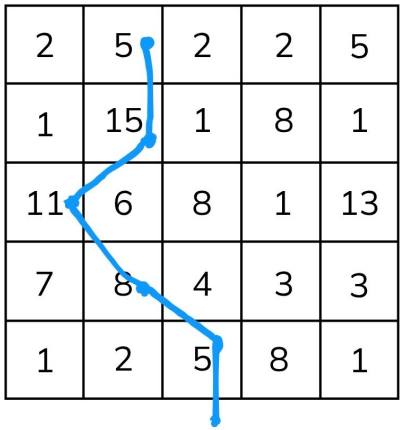
Ejemplo

23	10
5	8

2	5	2	2	5
1	15	1	8	1
11	6	8	1	13
7	8	4	3	3
1	2	5	8	1



31



44

Idea de resolución con DP

Idea:

Para cada casilla, guardar la cantidad máxima de piedras que podemos agarrar en un camino hacia esa casilla.

Pensar en cómo hacer el código

Idea:

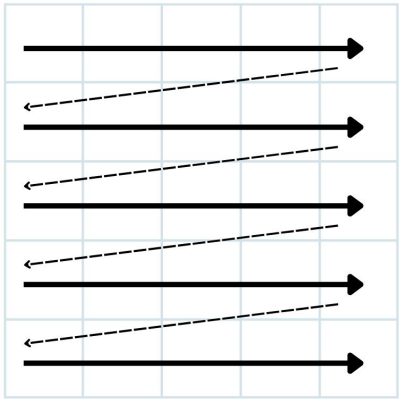
Para cada casilla, guardar la cantidad máxima de piedras que podemos agarrar en un camino hacia esa casilla.

Para completar un valor necesitamos las tres casillas de arriba

Pensar en cómo hacer el código

Para completar un valor necesitamos las tres casillas de arriba...

Entonces podemos ir completando las casillas de esta forma:



Código

```
1  #include<bits/stdc++.h>
2  using namespace std;
3  using vi = vector<int>;
4  #define forn( i, n ) for ( int i = 0; i < n; i++ )
5  const int inf = 1e9;
```

Código

```
7   void cas() {
8
9       int h, w; cin >> h >> w;
10
11      // creamos y leemos la grilla
12      vector<vi> grilla ( h, vi ( w, 0 ) );
13      forn( i, h ) forn( j, w ) { cin >> grilla[ i ][ j ]; }
14
15      // creamos la dp
16      vector<vi> dp ( h, vi ( w, 0 ) );
17
18      // en la primera fila, la dp es el mismo valor que la grilla
19      forn ( j, w ) { dp[ 0 ][ j ] = grilla[ 0 ][ j ]; }
20
```

Código

```
21      //  en cada casilla de  las demás filas
22      // depende de las tres casillas de arriba
23      // ojo con los bordes!
24
25      for ( int i = 1; i < h; i++ )
26      {
27          forn ( j, w ) {
28
29              // estos son los tres valores que puede tener la dp en las
30              // casillas de arriba.
31              // si alguna casilla no existe, entonces va a tener el valor  - inf
32
33              int valor1 = -inf; if ( j > 0 ) { valor1 = dp [ i - 1 ][ j - 1 ]; }
34
35              int valor2 = dp [ i - 1 ][ j ];
36
37              int valor3 = -inf;  if ( j < ( w - 1 )  ) { valor3 = dp[ i - 1 ][ j + 1 ]; }
38
39              dp[ i ][ j ] = grilla[ i ][ j ] + max({ valor1, valor2, valor3 });
40          }
41      }
```

Código

```
43      int rta = -inf;
44      // la respuesta va a ser el máximo entre todos los valores de la última fila
45      forn ( i, w ) { rta = max ( rta, dp[ h - 1 ][ i ] ); }
46      cout << rta << "\n";
47
48  }
```

Outline

Material recomendado

- Problemas discutidos en clase; y aparte, a sus respectivas implementaciones en c++:
  - **Problema suma de dados)**
    Dice Combinations (CSES) |
  - **Knapsack**
    Knapsack (AtCoder) |
  - **Máxima suma en un camino**
    Philosopers Stone (SPOJ) |
- **Lista famosa de problemas de Prog Competitiva**
  Lista 'YouKnowWho' | Lista gigante de problemas y material teórico, recomiendo elegir uno o dos con estrellita.
- **CSES**
  CSES | Colección de problemas clásicos (hay una sección de DP ;)) + 'Competitive Programming Handbook', con algunas de las soluciones.

Consultas

Cualquier duda me pueden enviar (durante la semana) a:

- nicolasricci08@gmail.com

Fin

¡Muchas gracias!