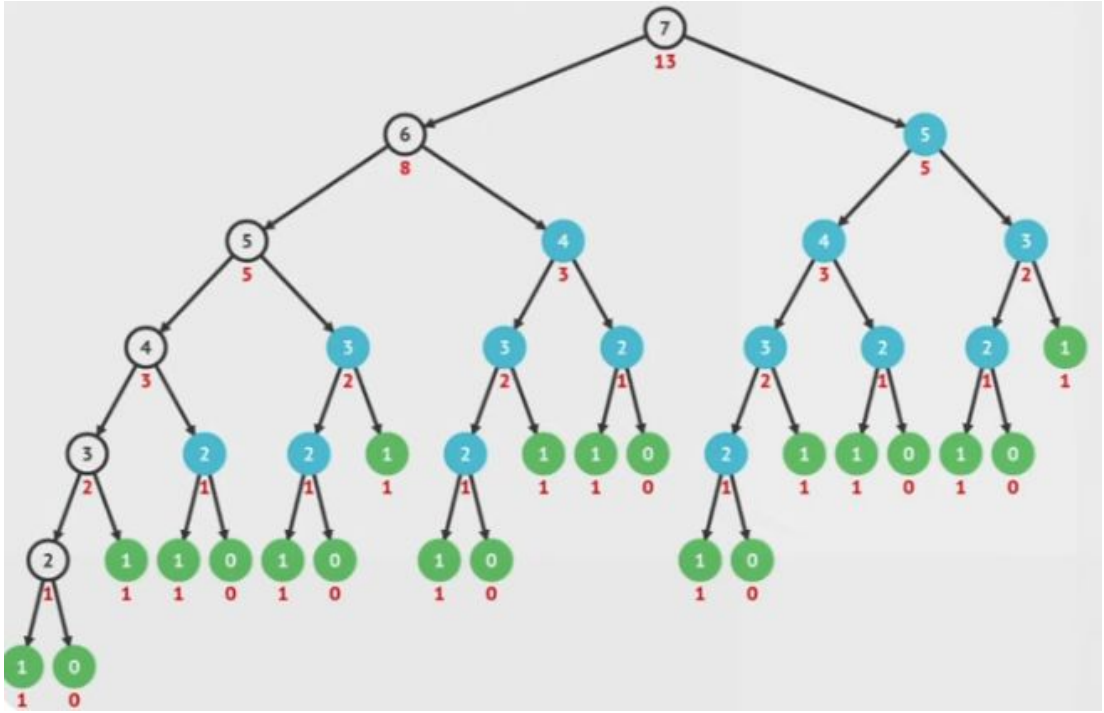
Programación Dinámica

```
def fibo(n):
    if n <= 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```

```
def fibo(n):
    if n <= 1:
        return n
    else:
        return fibo(n-1) + fibo(n-2)
```



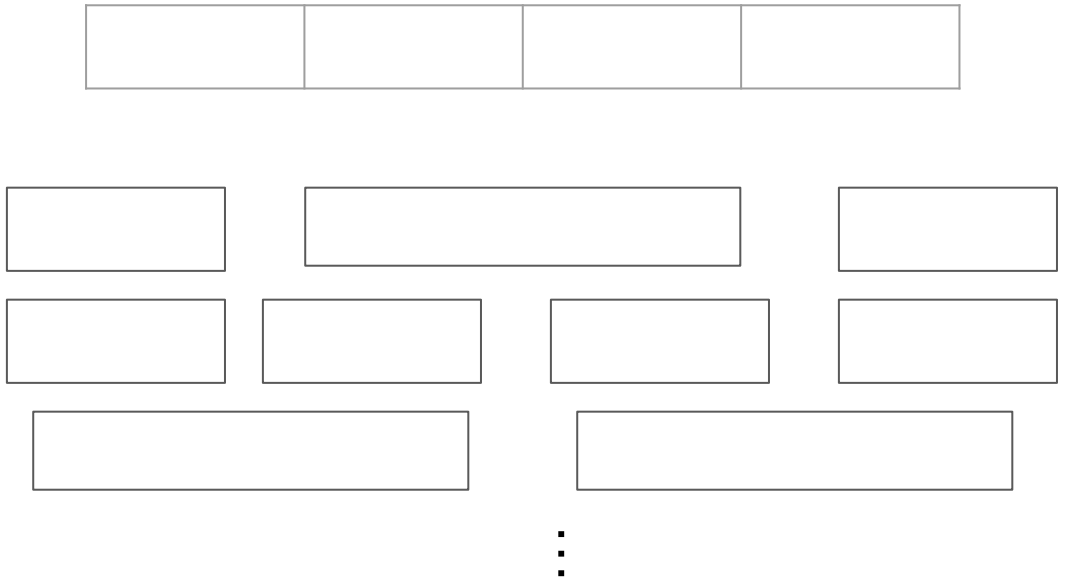
Programacion Dinamica

- Divide y venceras
  - Separar problemas en subproblemas
  - Resolver sub-problemas
  - Juntar solución de sub-problemas para resolver el problema inicial
- La programación dinámica se aplica solo cuando los sub-problemas están superpuestos
  - Calculamos cada sub-problema unico solamente una vez

Ejemplo

Cómo cortar una varilla para maximizar el precio de venta?
Diferentes tamanhos de varillas resultantes tienen diferentes precios

Tamanho	Costo
1	1
2	5
3	8
4	9
5	18



Pasos a seguir

1. Describir la estructura de la solución óptima
2. Definir (recursivamente) el valor de esa solución
3. Calcular el valor óptimo utilizando Programación Dinámica (DP: Dynamic Programming). Reutilizando sub-soluciones previas ya calculadas
4. Obtener la solución óptima

$$r(n) = \max(p_n,\ r(1)+r(n-1),\ \ldots,\ r(n-1)+r(1))$$

$$r(n) = \max_{1 \le i \le n}(p_i + r(n-i))$$

$$r(n) = \max_{1 \le i \le n}(p_i + r(n-i))$$

```python
def cortar_var(tam_var, lista_p):
    if tam_var == 0:
        return 0
    mejor_valor = 0
    for i in range(1, tam_var+1):
        valor = lista_p[i] + cortar_var(tam_var-i, lista_p)
        if valor > mejor_valor:
            mejor_valor = valor

    return mejor_valor
```

Ahora si DP - Utilizando memoización (Top down approach)

```python
def cortar_var(tam_var, lista_p, memory):
    if tam_var == 0:
        return 0
    if memory[tam_var] != -1:
        return memory[tam_var]

    mejor_valor = 0
    for i in range(1, tam_var+1):
        valor = lista_p[i] + cortar_var(tam_var-i, lista_p, memory)
        if valor > mejor_valor:
            mejor_valor = valor

    memory[tam_var] = mejor_valor
    return mejor_valor


if __name__ == "__main__":
    lista_p = [0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30]
    n = int(input())
    memo = [-1] * (n+1)
    print(cortar_var(n, lista_p, memo))
```

Ahora si DP - Utilizando tabulación (Bottom up approach)

```python
def cortar_var(tam_var, lista_p):
    tabla = [0] * (tam_var+1)

    for varilla_actual in range(1, tam_var+1):
        mejor_valor = 0
        for i in range(1, varilla_actual+1):
            valor = lista_p[i] + tabla[varilla_actual-i]
            if valor > mejor_valor:
                mejor_valor = valor

        tabla[varilla_actual] = mejor_valor

    return tabla[tam_var]


if __name__ == "__main__":
    lista_p = [0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30]
    n = int(input())
    print(cortar_var(n, lista_p))
```

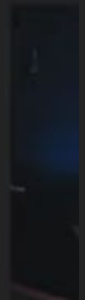
Recursivo Normal:
$O(2^n)$

Recursivo DP:
$O(n^2)$

```python
def cortar_var(tam_var, lista_p, memory, camino):
    if tam_var == 0:
        return 0
    if memory[tam_var] != -1:
        return memory[tam_var]

    mejor_valor = 0
    mejor_corte = 0
    for i in range(1, tam_var+1):
        precio = 0 if len(lista_p) <= i else lista_p[i]
        valor = precio + cortar_var(tam_var-i, lista_p, memory, camino)
        if valor > mejor_valor:
            mejor_valor = valor
            mejor_corte = i

    memory[tam_var] = mejor_valor
    camino[tam_var] = mejor_corte
    return mejor_valor
```



```python
def print_solution(tam_var, camino):
    print("Tamaño de varillas:")
    while tam_var > 0:
        print(camino[tam_var])
        tam_var = tam_var - camino[tam_var]


if __name__ == "__main__":
    lista_p = [0, 1, 5, 8, 9, 10, 17, 17, 20, 24, 30]
    n = int(input())
    memo = [-1] * (n+1)
    camino = [0] * (n+1)
    print(cortar_var(n, lista_p, memo, camino))
    print_solution(n, camino)
```