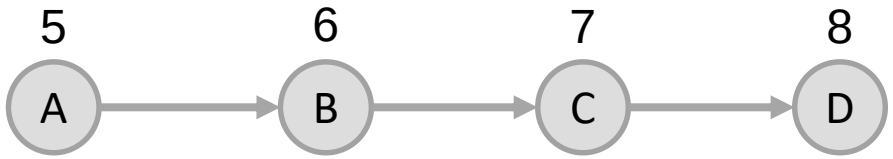
Máximo conjunto independiente

Máximo conjunto independiente

Entrada: Una secuencia de *N* elementos con pesos no negativos *w*

Salida: Subconjunto de elementos no adyacentes (conjunto independiente) que maximice la suma de pesos

Ejemplo:



Solución: {B,D}

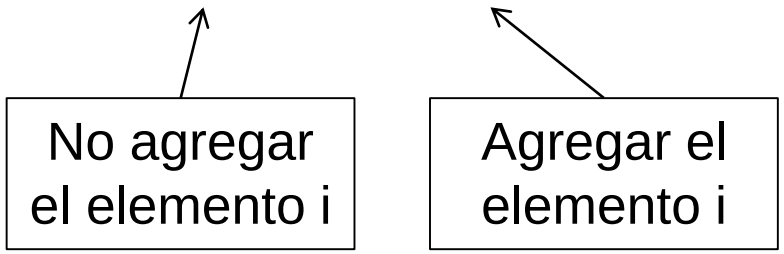
Máximo conjunto independiente

Para un valor de *N*, ¿Cuántas posibles soluciones (incluyendo las inválidas, es decir cuando contiene nodos adyacentes) tiene este problema? 2^N

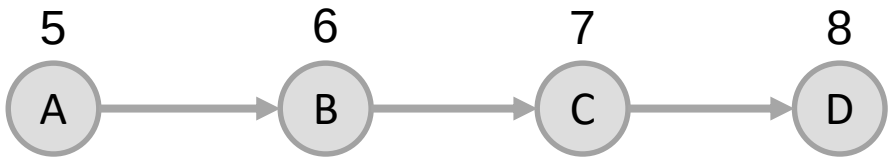
Solución por búsqueda exhaustiva: Generar un arreglo de *N* valores binarios y evaluar en *O(N)* cada una de las 2^N posibilidades

Solución top-down (recursiva):

```
function MIS(i):
   if i <= 1:
      return w_i // con w_0=0
   else
      return MAX(MIS(i-1), w_i + MIS(i-2))
```

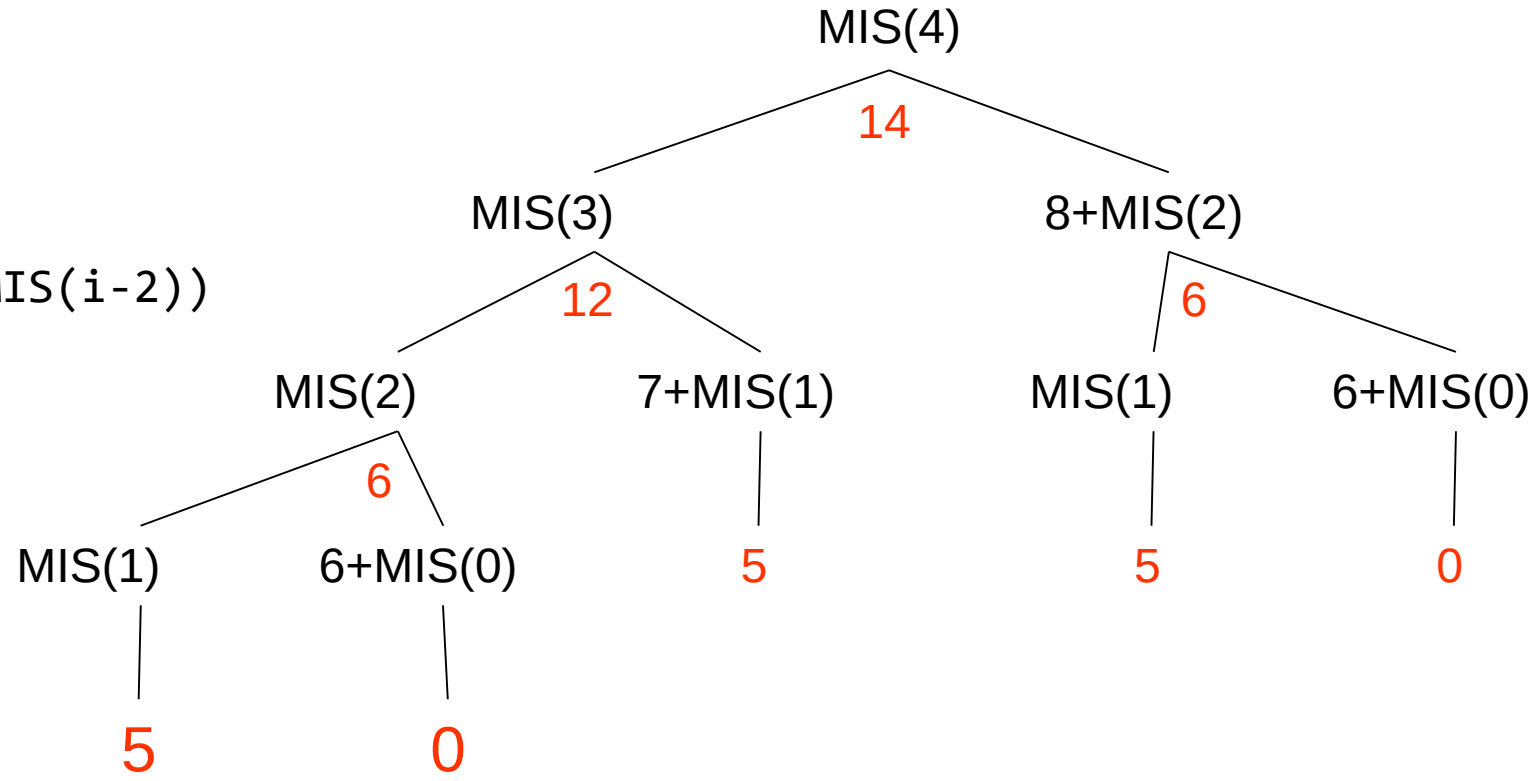


Ejemplo:



w = [5, 6, 7, 8]

```
function MIS(i):
   if i <= 1:
      return w_i
   else
      return MAX(MIS(i-1), w_i + MIS(i-2))
```

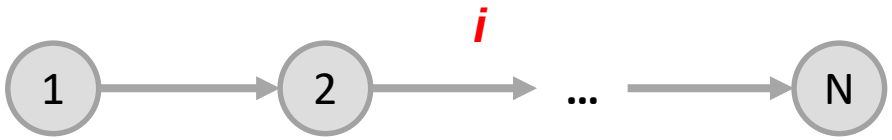


¿Cuál es la eficiencia de este algoritmo?

$O(2^N)$, es decir, este algoritmo es prácticamente igual de eficiente que el de fuerza bruta.

Solución mediante programación dinámica

El primer paso a la hora de definir una solución mediante esta técnica es no pensar en el problema original y en cambio preguntarse ¿Qué forma debería tener la solución óptima de un subproblema?



Al analizar un nodo *i*, habría dos alternativas:

1. Agregar *i* a la solución, de esta forma la máxima suma de pesos sería w_i mas la sumatoria de pesos de la solución óptima para los nodos restantes *1* a *i-2*
2. No agregar *i* a la solución, de esta forma la máxima suma de pesos sería la sumatoria de pesos de la solución óptima para los nodos restantes *1* a *i-1*

Solución mediante programación dinámica

```
S_0 = 0
S_1 = w_1
for i=2 to N:
    S_i = MAX(w_i + S_i-2, S_i-1)
print(S_N)
```

Ejemplo: w = [5, 6, 7, 8]

i	0	1	2	3	4
S	0	5	6	12	14

¿Cuál es la eficiencia de este algoritmo?

O(*N*), sin duda mucho mejor que O($\mathbf{2}^{\boldsymbol{N}}$)