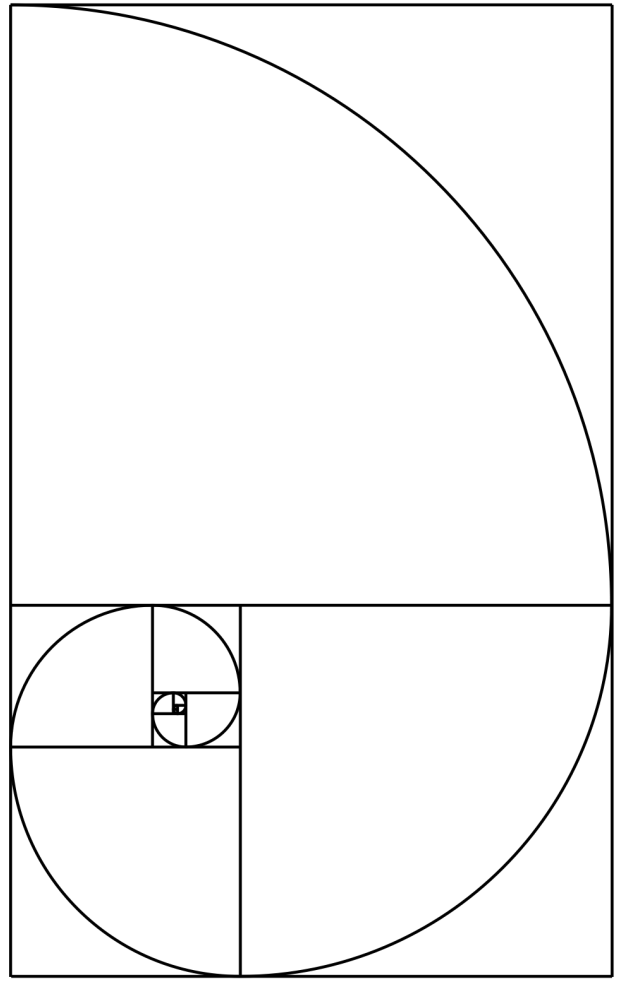
Ayudantía 12

Programación Dinámica

Fibonacci

- Secuencia de números dada por la suma de los dos anteriores
- Presente en la naturaleza :O
- 0, 1, 1, 2, 3, 5, 8, 13…



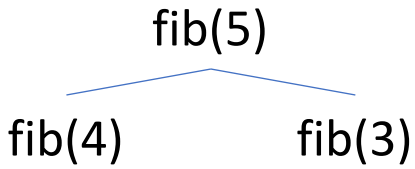
Intro a la progra

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

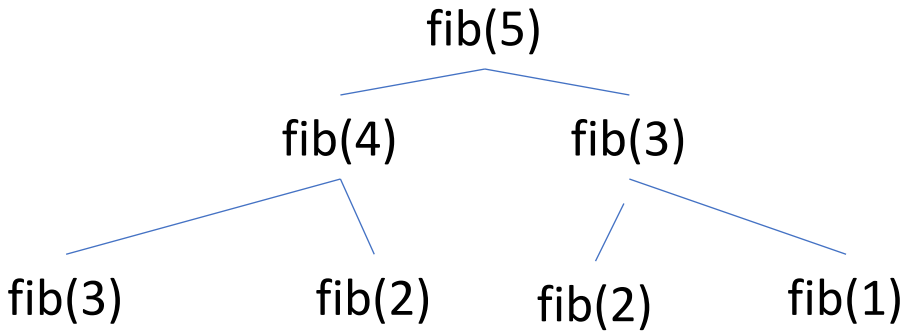
Complejidad :S

fib(5)

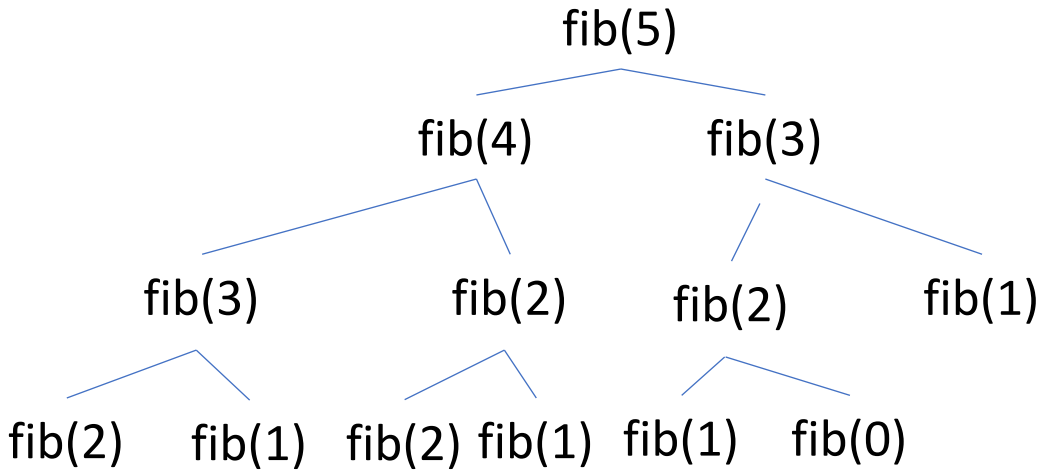
Complejidad :S



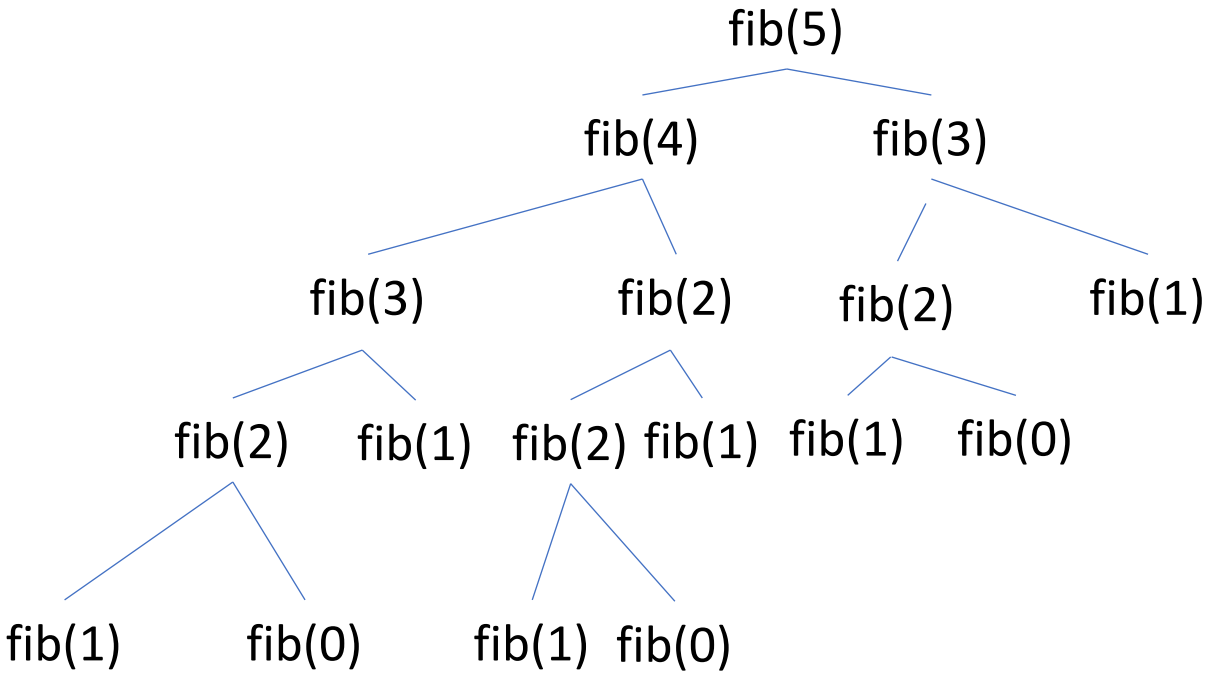
Complejidad :S



Complejidad :S



Complejidad :S



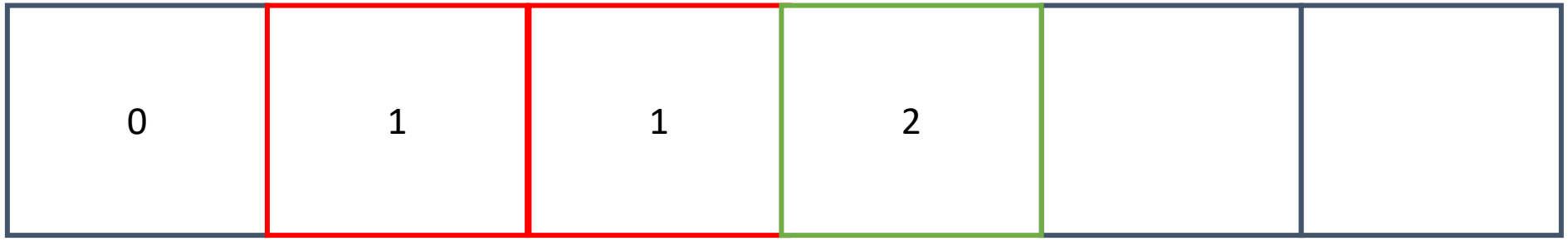
$O(2^n)!!!$

Fibonacci dinámico

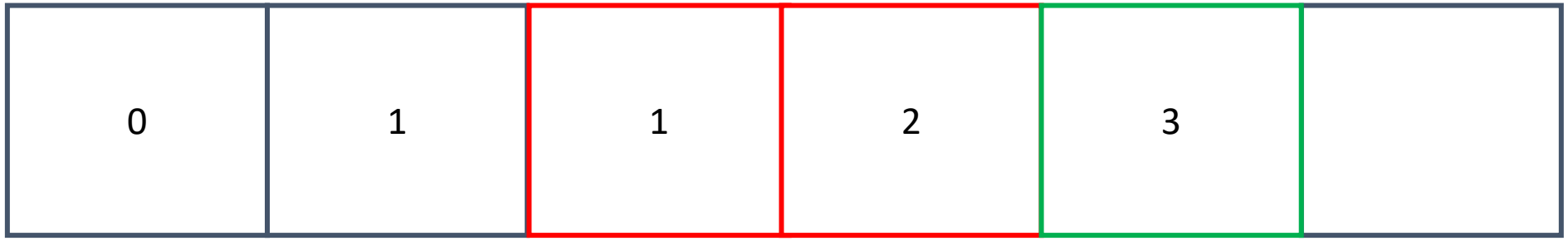
Fibonacci dinámico

Fibonacci dinámico

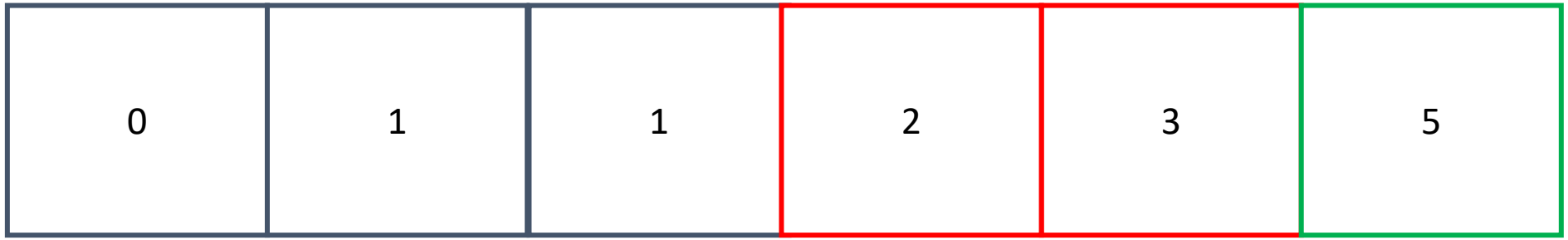
Fibonacci dinámico



Fibonacci dinámico



Fibonacci dinámico



Complejidad?

Fibonacci dinámico

O(n)!!!

Código

```
def fib_dp(n):
        if n < 2:
                return n
        fibonaccis = [0, 1]
        for i in range(2, n):
                fibonaccis.append(fibonaccis[i – 1] + fibonaccis[i – 2])
```

Sub-matrices de 1s

- Supongamos que tenemos una lista de listas
- Cada lista esta compuesta de 1s o 0s
- Queremos retornar cuantos cuadrados de 1s hay
- Ej:

[[0, 1, 1, 1],
[1, 0, 1, 1],
[0, 0, 1, 0]]

Sub-matrices de 1s

0	1	1	1
1	1	2	2
0	1	2	3

Código

```
def countSquares(matrix):
    counter = 0
    for i in matrix[0]:
        if i == 1:
            counter += 1
    for i in range(1, len(matrix)):
        if matrix[i][0] == 1:
            counter += 1
    for i in range(1, len(matrix)):
        for j in range(1, len(matrix[0])):
            if matrix[i][j] == 1:
                matrix[i][j] += min(
                    matrix[i - 1][j - 1],
                    matrix[i - 1][j],
                    matrix[i][j - 1],
                )
            counter += matrix[i][j]
    return counter
```

Problema 2

Dado un nodo fuente s, escribe un algoritmo tal que se marque

$$v.distance = -\infty$$

para todo vértice v si es que se encuentra un ciclo negativo en algún camino desde s hasta v

Bellman-Ford

- Calcula los caminos más cortos desde un nodo a todo el resto
- Es más lento que Dijkstra y Floyd Warshall
- Ventaja → Se puede usar aun que hayan pesos negativos

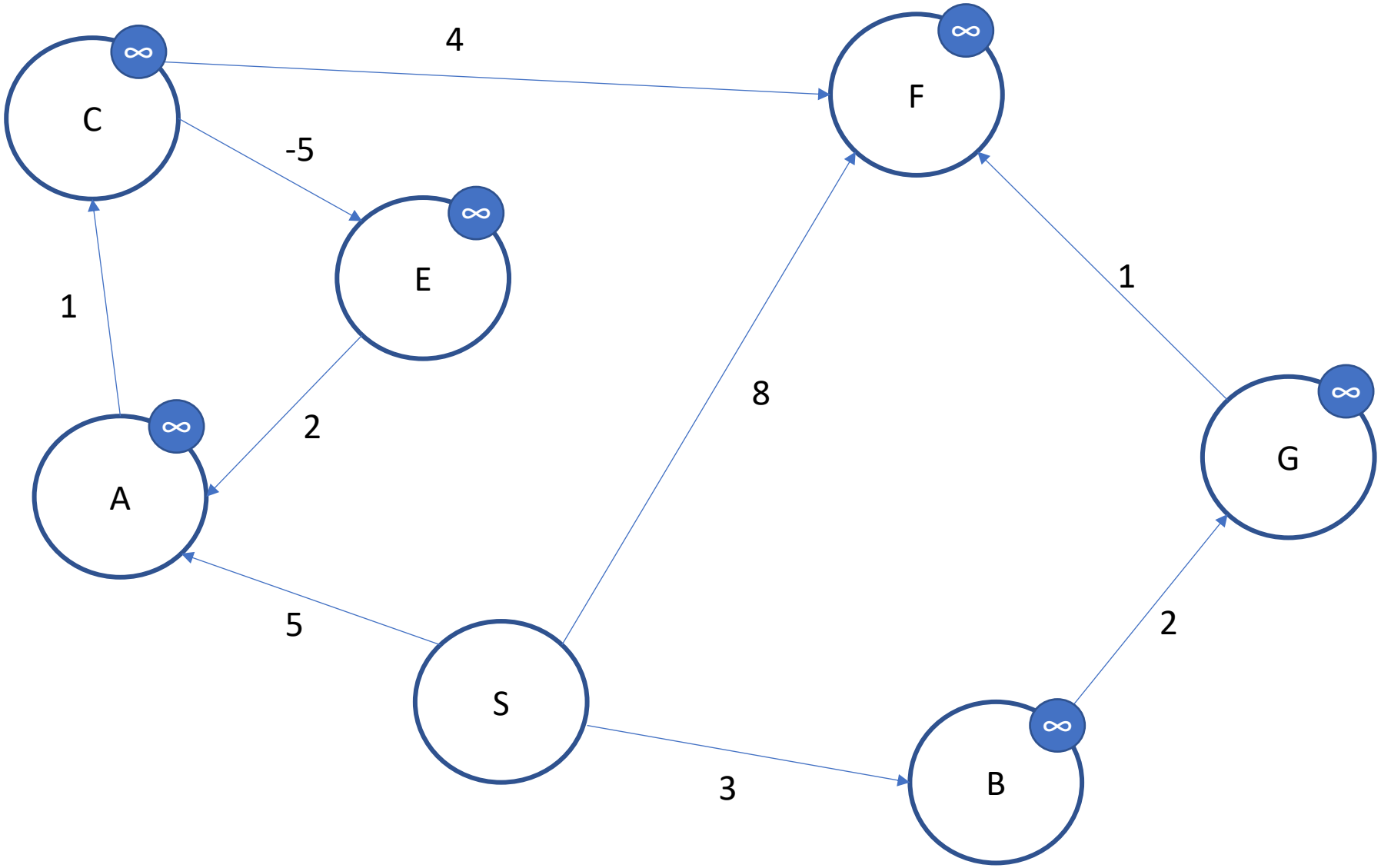
Bellman-Ford

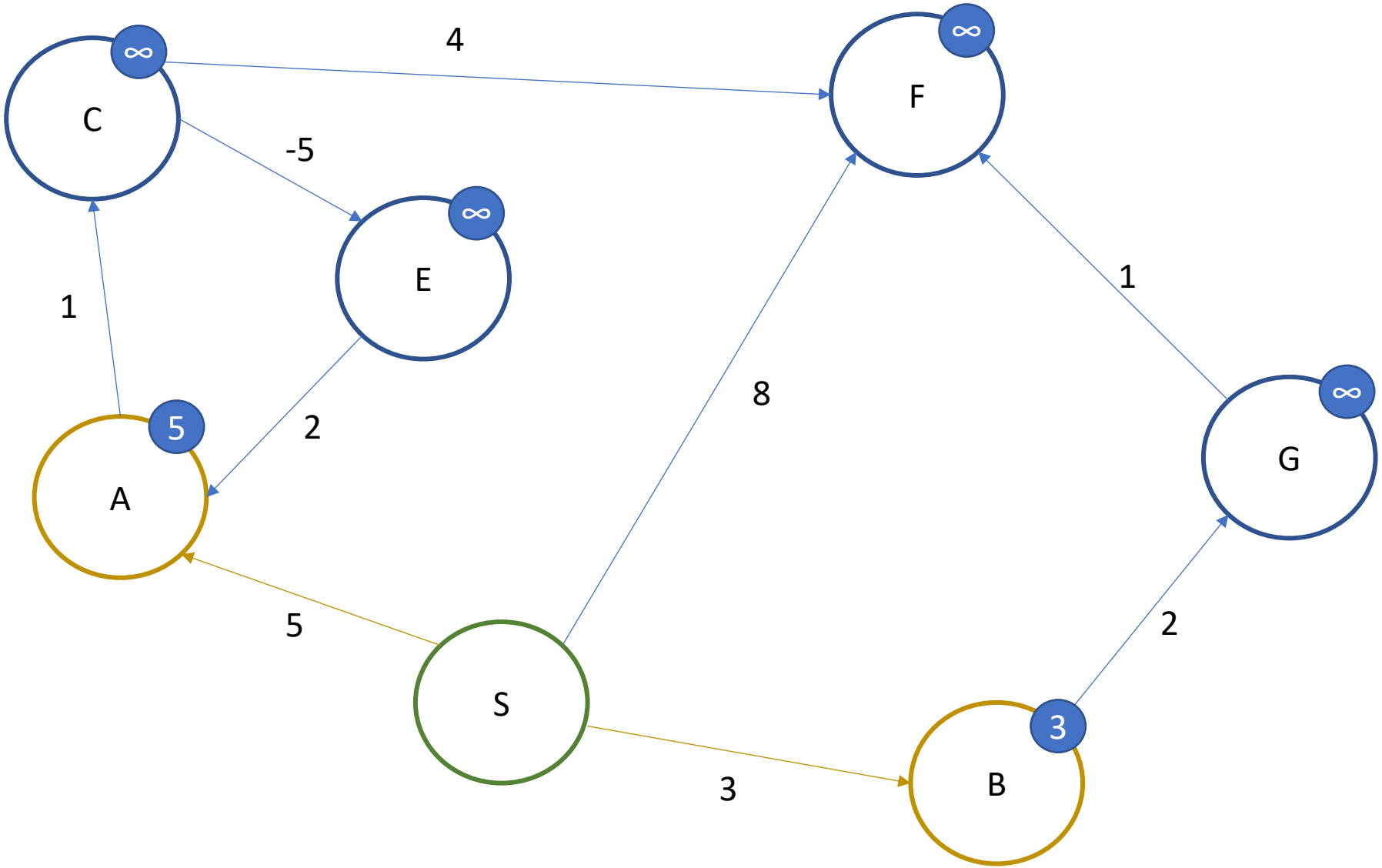
```
Bellman-Ford(s): —s es el vértice de partida
    for each u in V:
        d[u] ← ∞; π[u] ← null
    d[s] ← 0
    for k = 1 … |V|-1:
        for each (u,v) in E:
            if d[v] > d[u] + costo(u,v):
                d[v] ← d[u] + costo(u,v); π[v] ← u
```

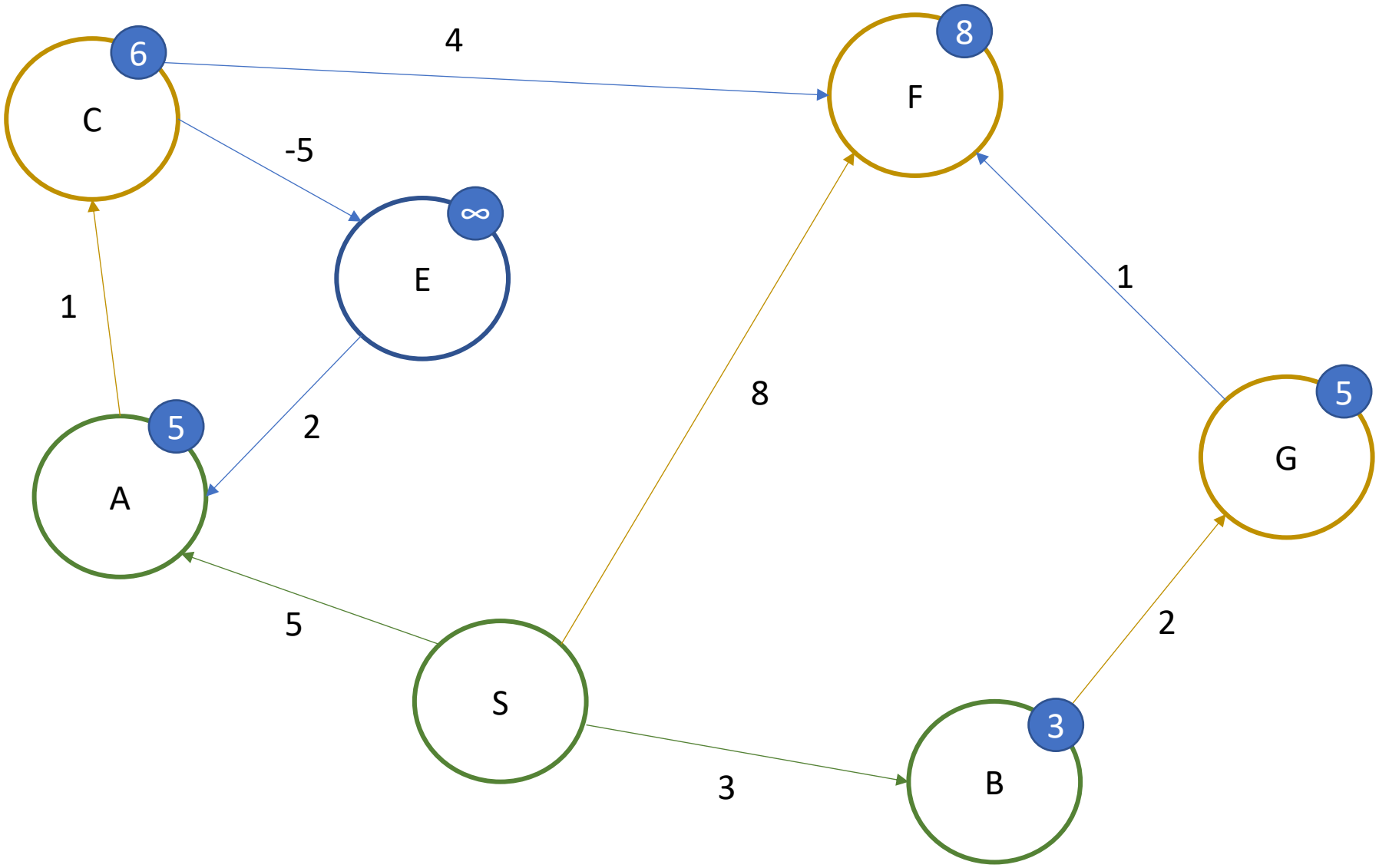
Bellman-Ford enchulado

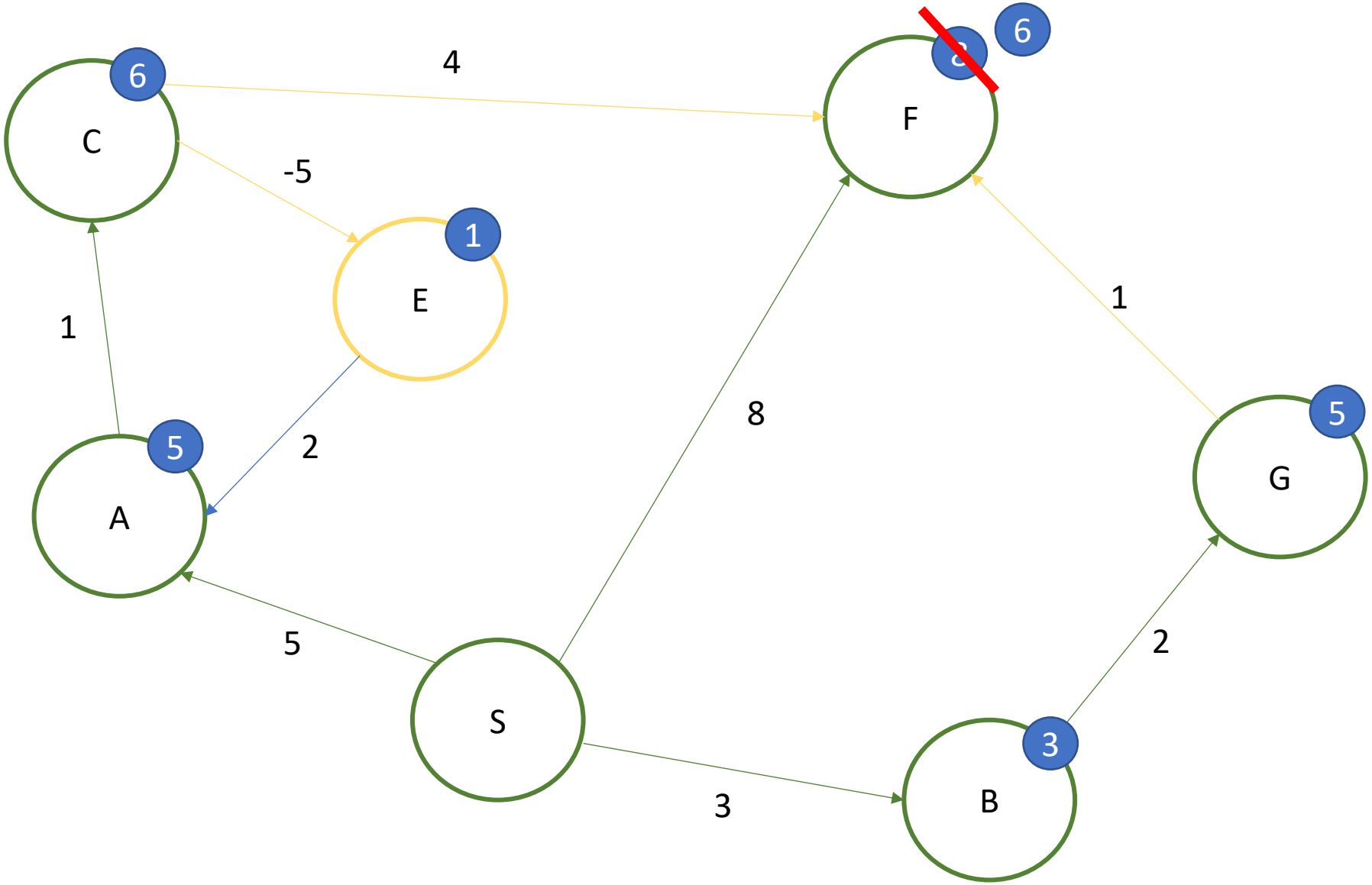
- Queremos marcar como -∞ la distancia a un vértice si en su camino hay un ciclo negativo

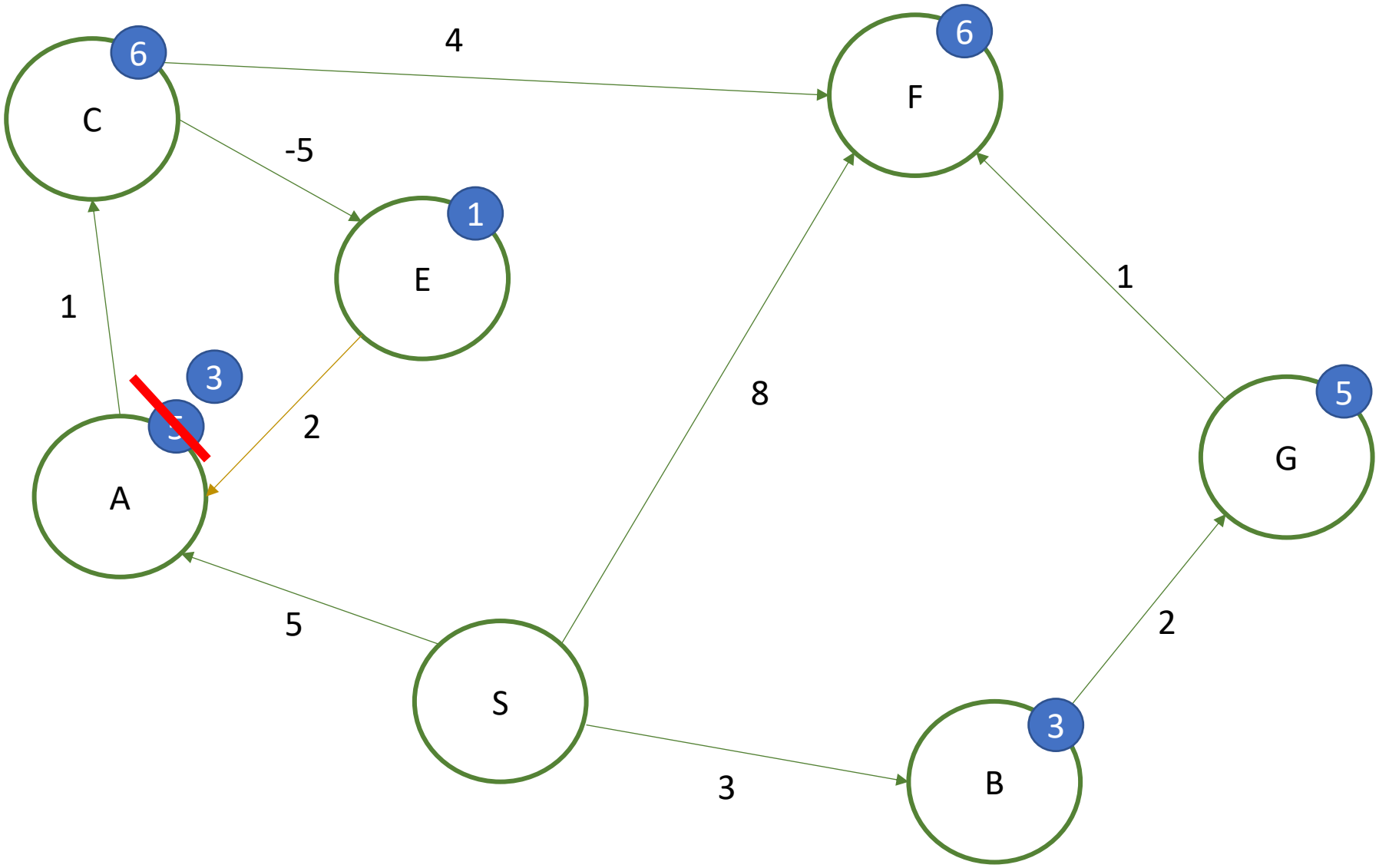
- Un ciclo negativo es un ciclo (dah) que la suma de los pesos de sus vértices es menor a 0

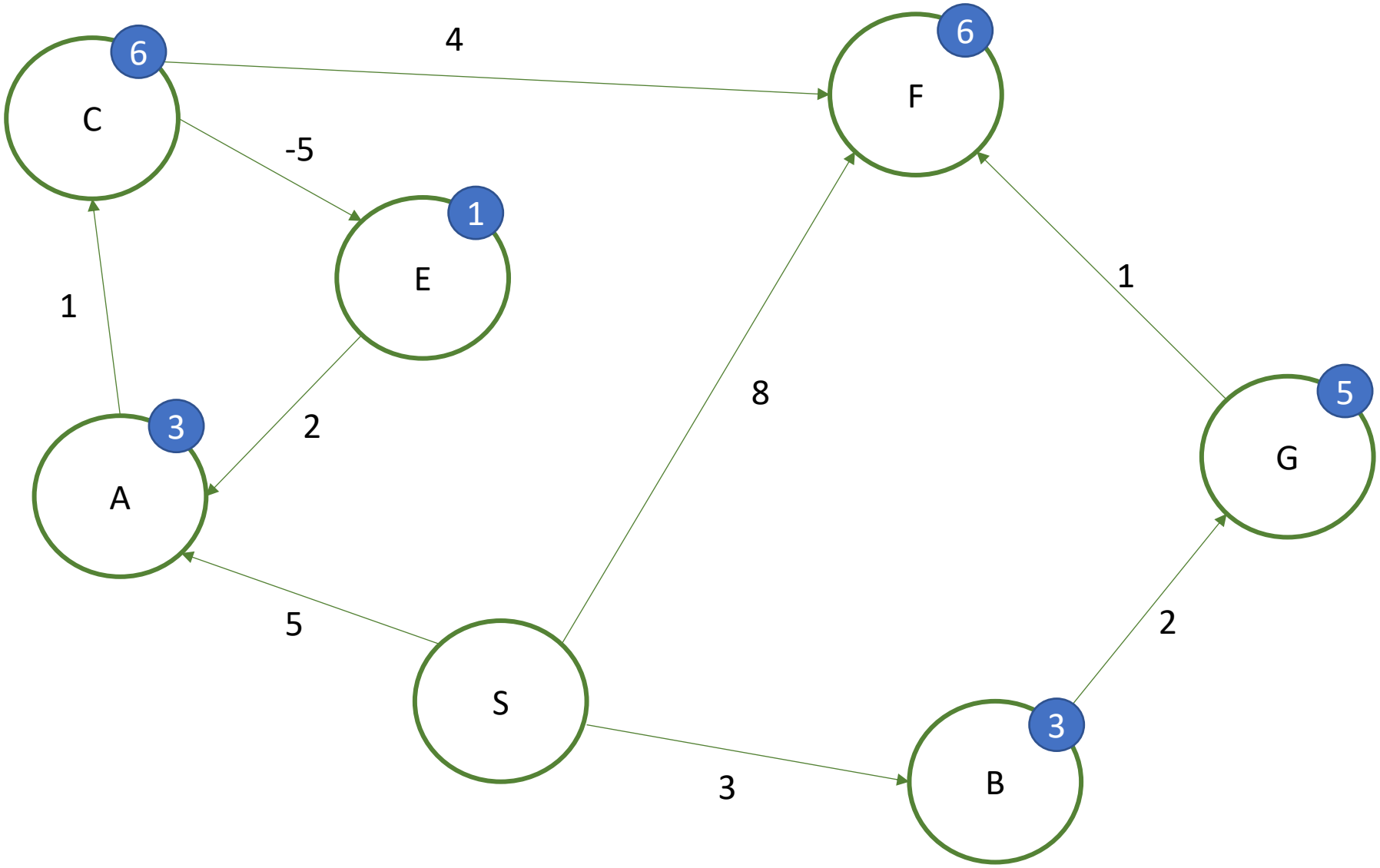

C
∞
4
F
∞
-5
E
∞
1
1
2
A
∞
8
G
∞
5
2
S
3
B
∞

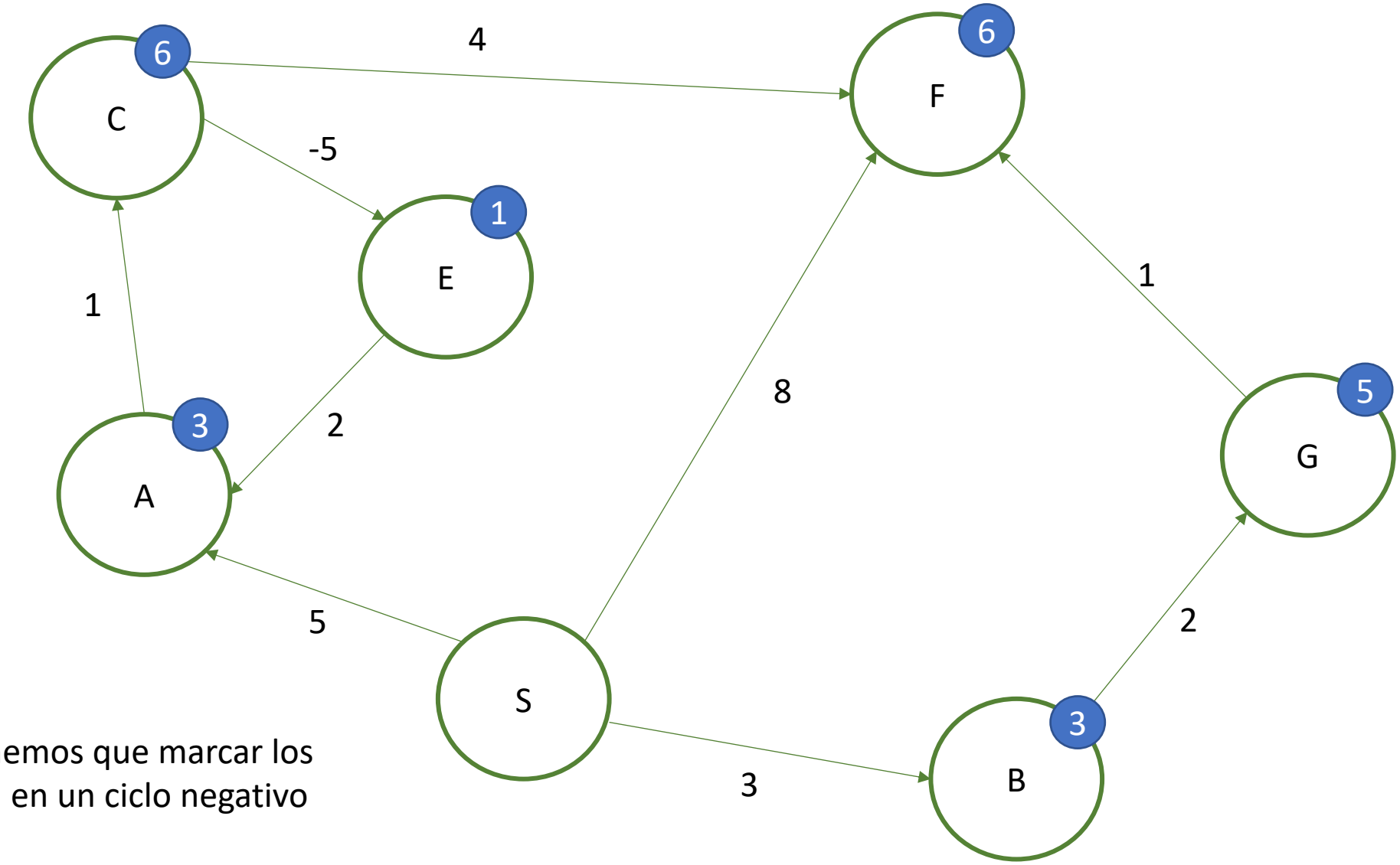

∞
C
4
∞
F
-5
∞
E
1
1
8
5
A
2
∞
G
5
S
2
3
B
3


6
C
4
8
F
-5
∞
E
1
1
5
A
2
8
5
G
5
2
S
3
3
B


6
C
4
6
F
-5
1
E
1
1
5
A
2
8
5
G
5
S
2
3
3
B

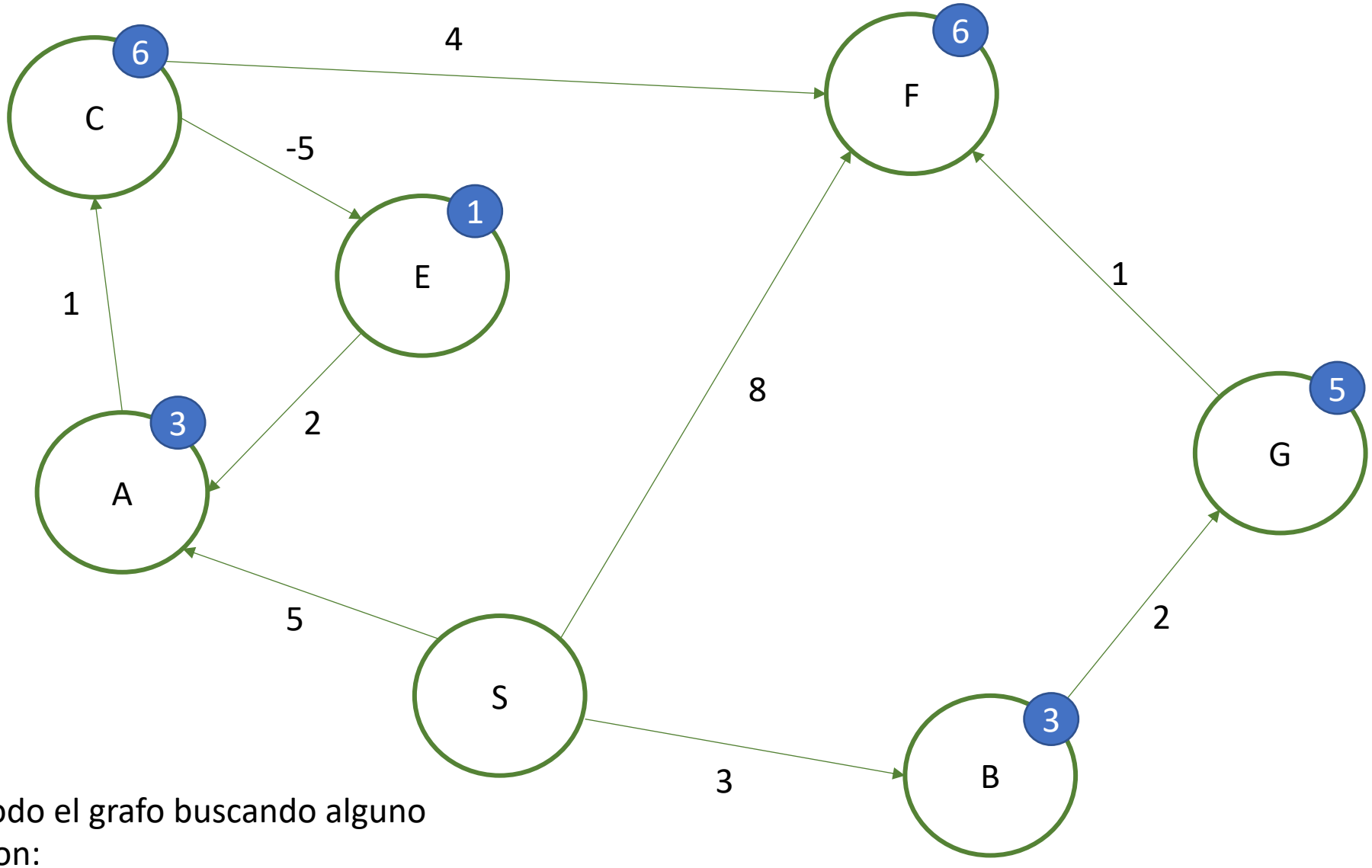

6
C
4
6
F
-5
1
E
1
1
3
A
2
8
5
G
5
2
S
3
B
3


6
C
4
6
F
-5
1
E
1
1
8
5
G
3
A
2
5
S
2
3
B
3



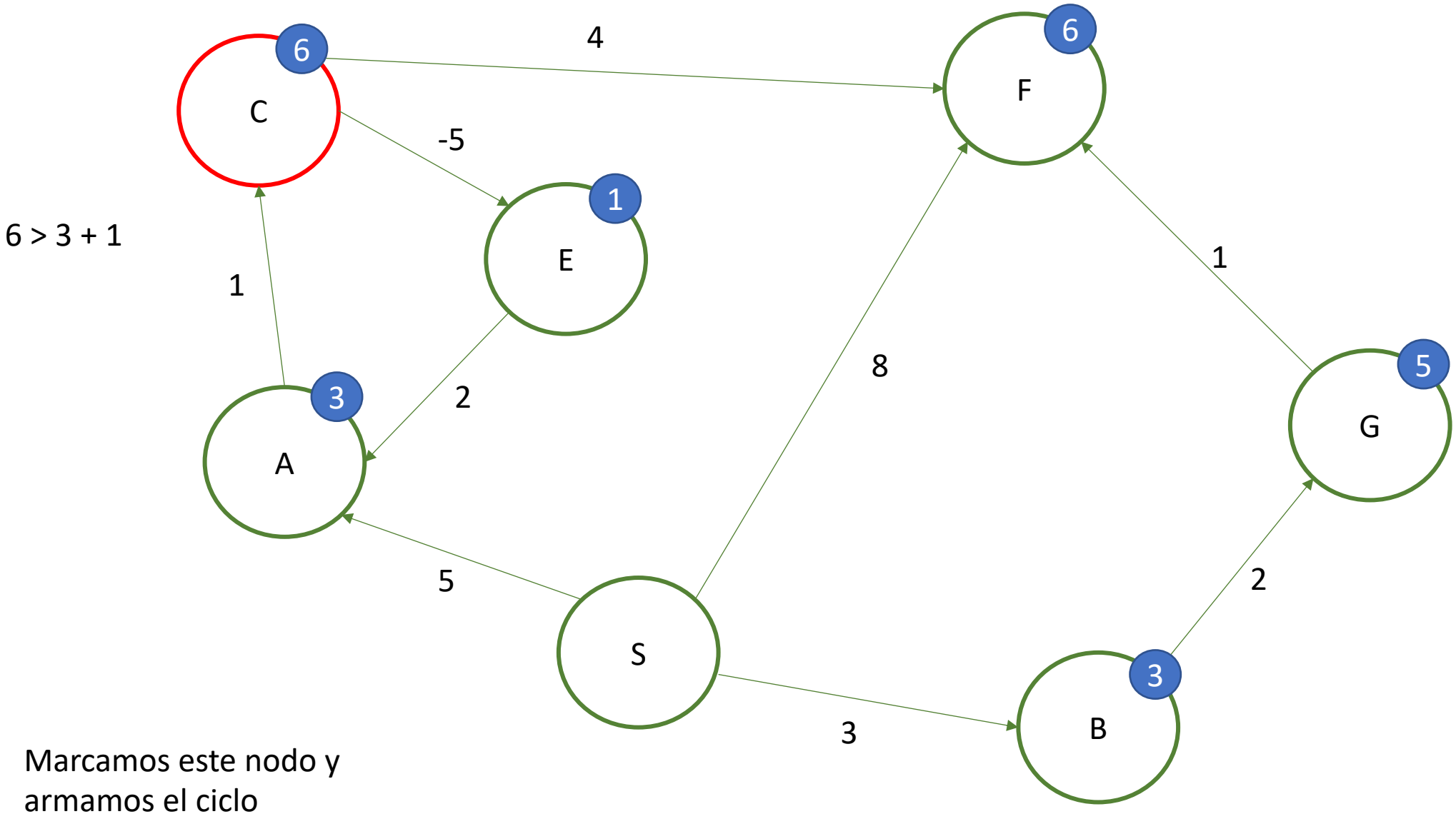
Ahora tenemos que marcar los que estén en un ciclo negativo

¿Cómo los identificamos?



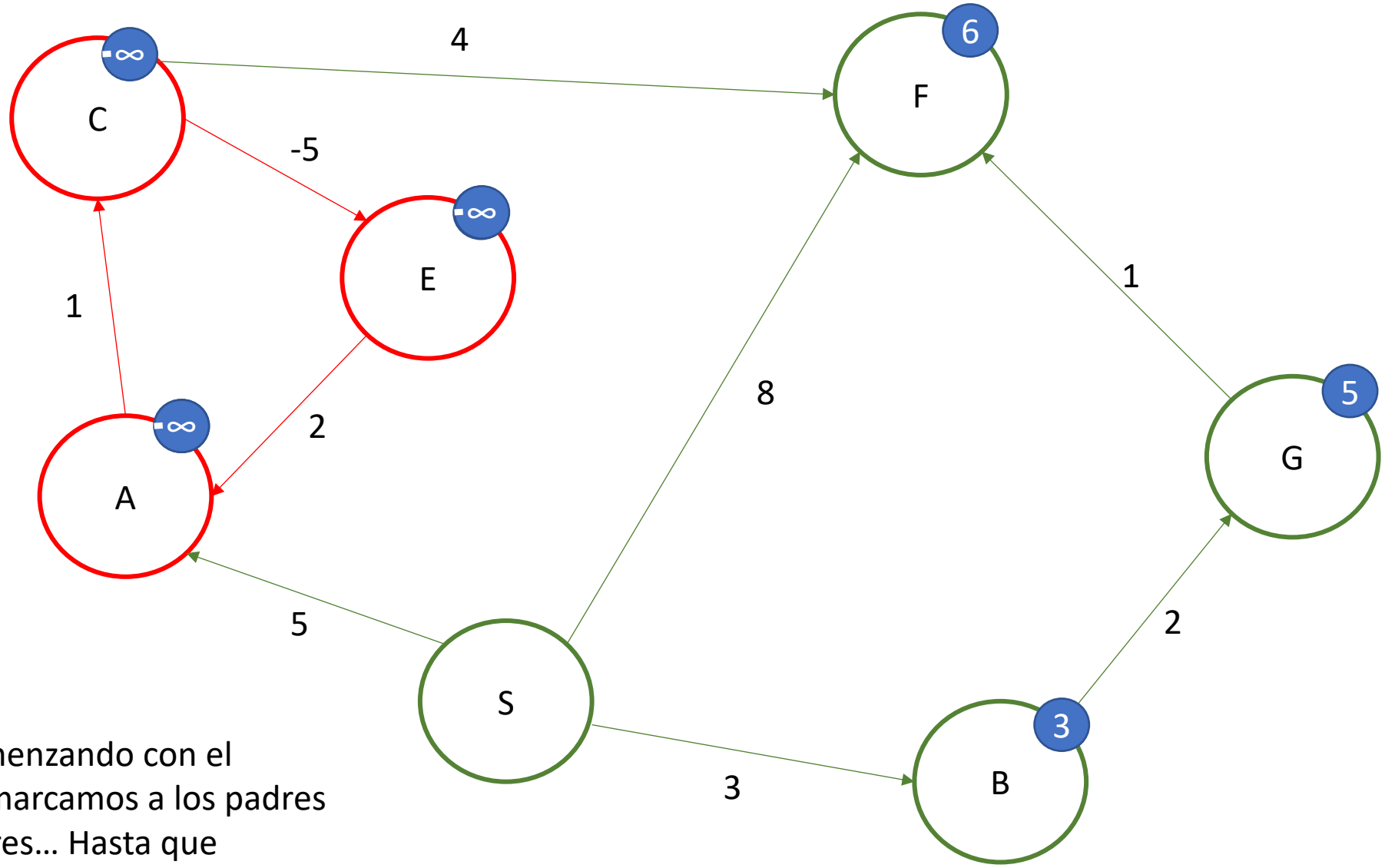
Exploramos todo el grafo buscando alguno que cumpla con:

nodo.distancia > padre.distancia + vértice

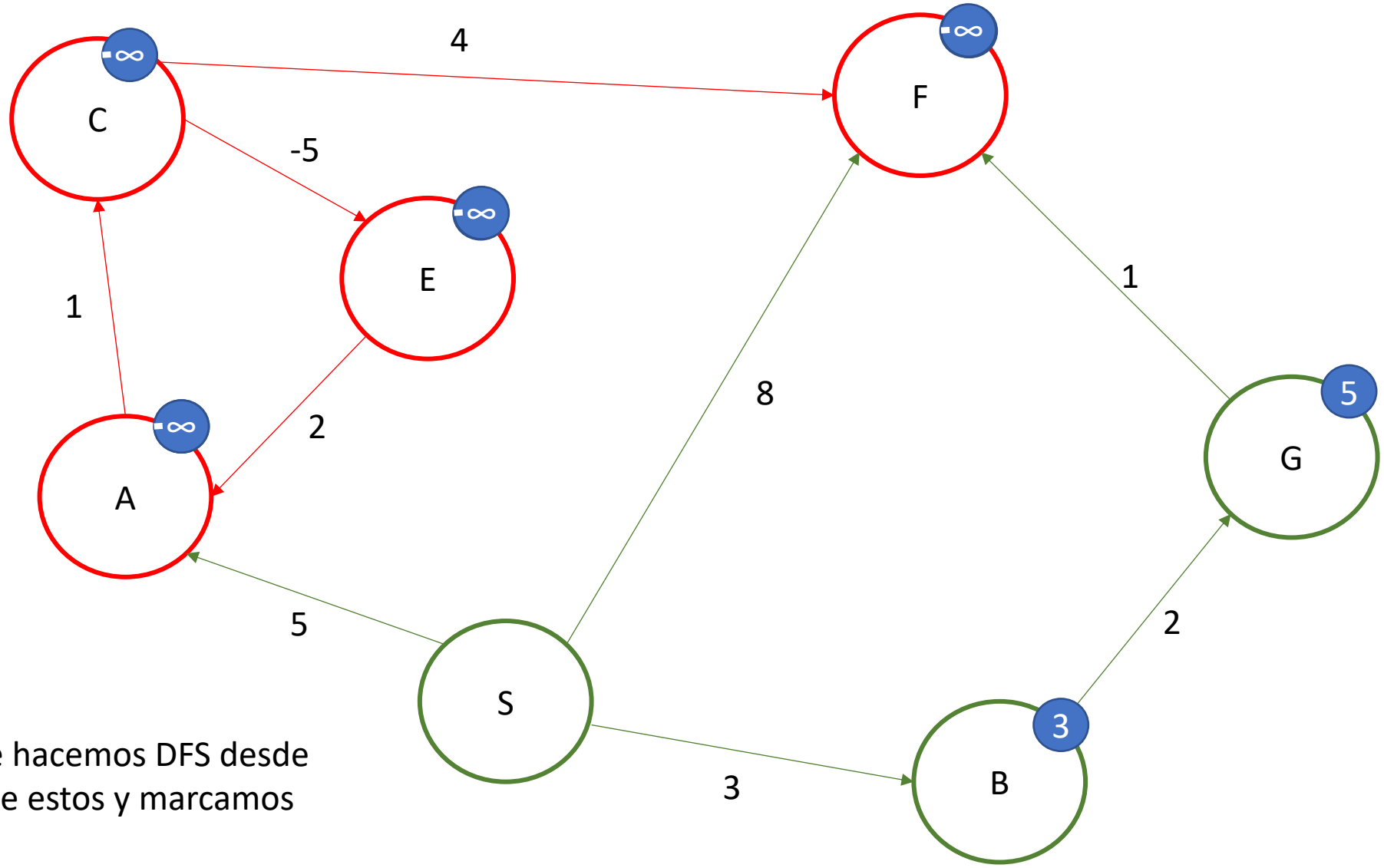


6 > 3 + 1

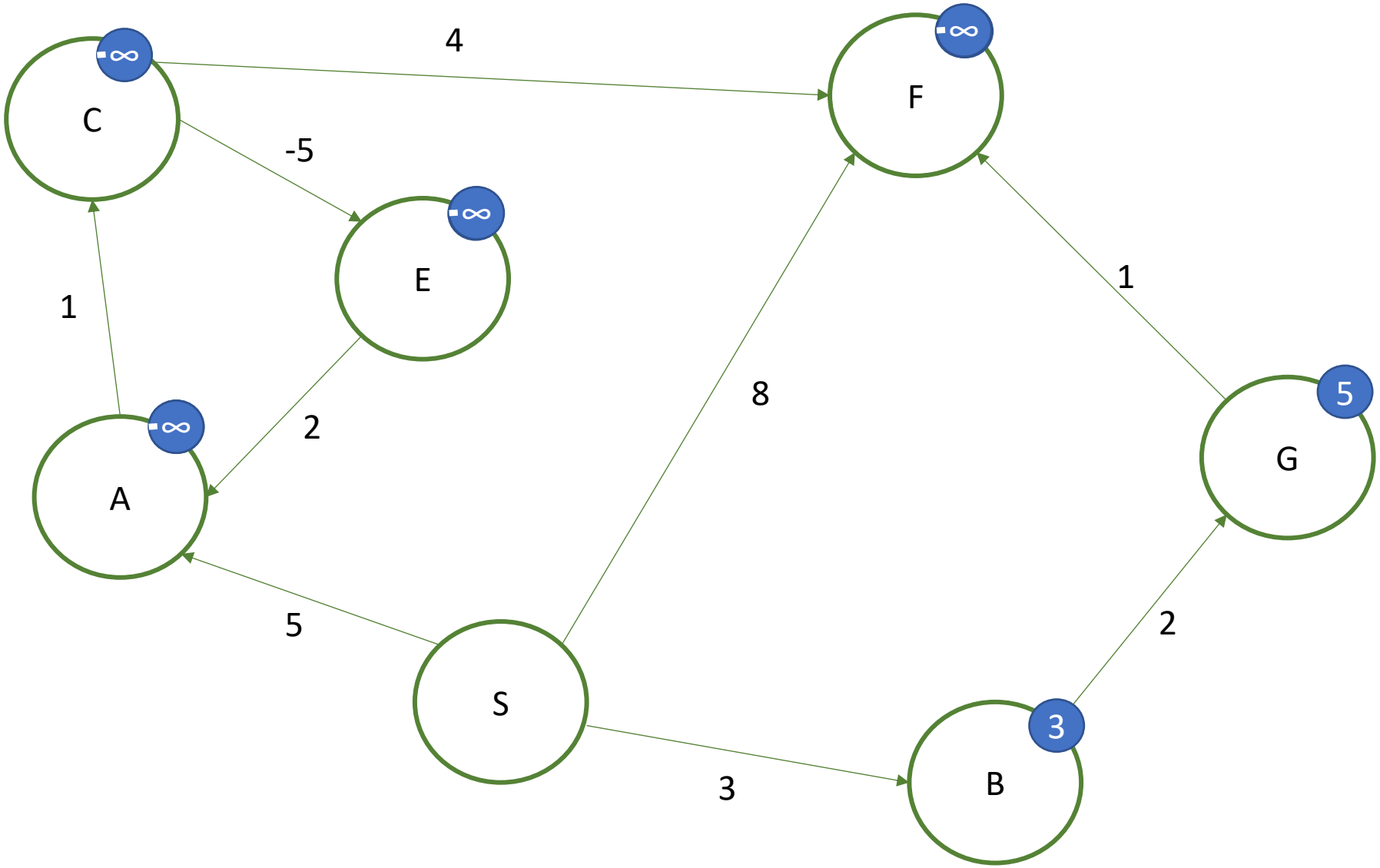
Marcamos este nodo y armamos el ciclo



Luego, comenzando con el marcado, marcamos a los padres de los padres… Hasta que cerramos el ciclo



Finalmente hacemos DFS desde cada uno de estos y marcamos con -∞


C
F
E
A
S
B
G
-∞
-∞
-∞
-∞
5
3
4
-5
1
2
8
1
5
3
2

Pseudocódigo

```
1  BELLMAN-FORD'(G, w, s)
2      INITIALIZE-SINGLE-SOURCE(G, s)
3      for i = 1 to |G.V| - 1
4          for each edge (u, v) ∈ G.E
5              RELAX(u, v, w)
6      for each edge(u, v) ∈ G.E
7          if v.d > u.d + w(u, v)
8              mark v
9      for each vertex v ∈ marked vertices
10         FOLLOW-AND-MARK-PRED(v)
```

```
1  FOLLOW-AND-MARK-PRED(v)
2      if v != NIL and v.d != -∞
3          v.d = -∞
4          FOLLOW-AND-MARK-PRED(v.π)
5      else
6          return
```