Bisección y recorridos en amplitud y en profundidad

Programación: encontrar la función, tope de pasos y ciclos

Carlos A Delgado S

Pontificia Universidad Javeriana Cali
Sesiones de refuerzo de algoritmia

Contenido

1 Bisección: encontrar la función

2 Recorrido en amplitud con tope de pasos

3 Recorrido en profundidad y ciclos

4 Problemas para practicar

Contenido

1 Bisección: encontrar la función
- Concepto
- Ejemplo
- Ejercicio

2 Recorrido en amplitud con tope de pasos

3 Recorrido en profundidad y ciclos

4 Problemas para practicar

Antenas en una calle

El problema

En una calle hay N casas, cada una en el número que es su distancia desde el comienzo. Se instalan M antenas, en cualquier punto. Se pide la menor distancia máxima posible entre una casa y su antena más cercana.

Lo primero que se intenta

Probar dónde poner las antenas. Las posiciones son números reales, así que no hay nada que recorrer: el espacio de respuestas es infinito.

Cambiar la pregunta

De pedir el óptimo a comprobar un candidato

Encontrar el mejor valor es difícil. Comprobar si un valor dado sirve es mucho más fácil, y con eso basta:

$$\varphi(d) : \text{¿alcanzan } M \text{ antenas si cada una cubre hasta } d \text{ metros?}$$

Por qué basta

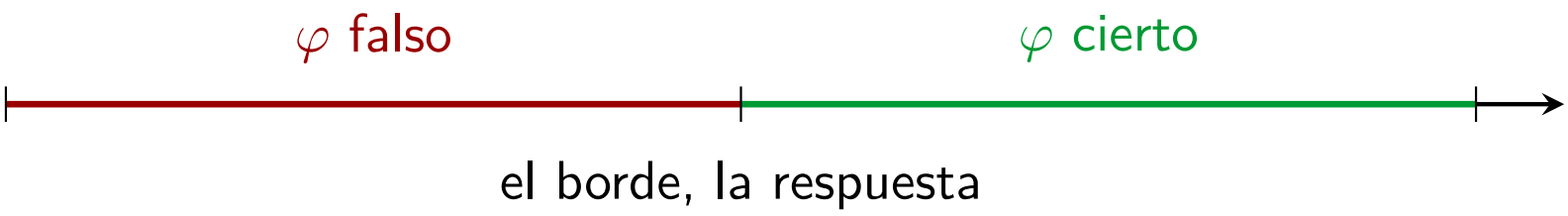
Si se sabe contestar φ, la respuesta del problema es el menor d donde φ pasa de falso a cierto. Buscar ese borde es lo que hace la bisección.

Cinco pistas para dar con la función

Qué mirar en el enunciado

1. La respuesta es un número, y piden el menor o el mayor que cumple algo.
2. Ese algo se comprueba para un candidato fijo en una pasada.
3. Al comprobarlo, el candidato pasa de incógnita a dato.
4. Si un candidato sirve, los de un lado también.
5. La respuesta está acotada por el peor y el mejor caso.

La monotonía es lo que hay que verificar



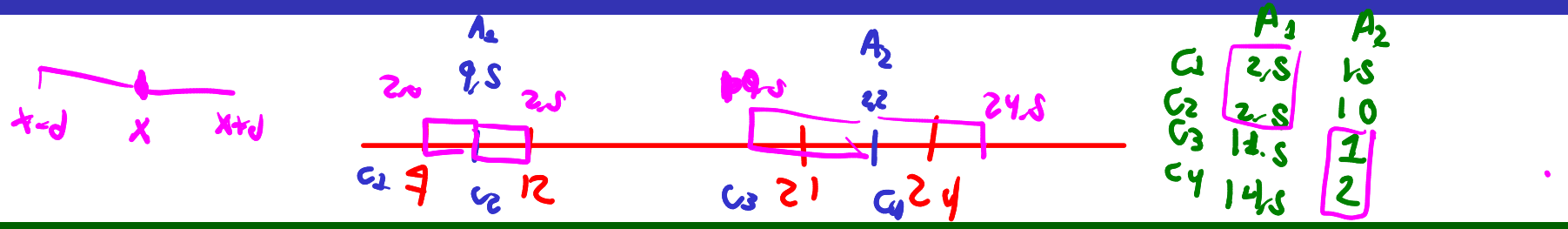
Qué se está diciendo

Si con d alcanzan M antenas, con cualquier $d' > d$ también, porque cada antena cubre más. Los candidatos quedan partidos en dos tramos y hay un solo borde.

Sin monotonía no hay bisección

Mirar el punto medio solo sirve si de ahí se deduce en qué mitad está la respuesta. Esa deducción es la monotonía, y por eso se comprueba antes de escribir el ciclo.

Un caso pequeño

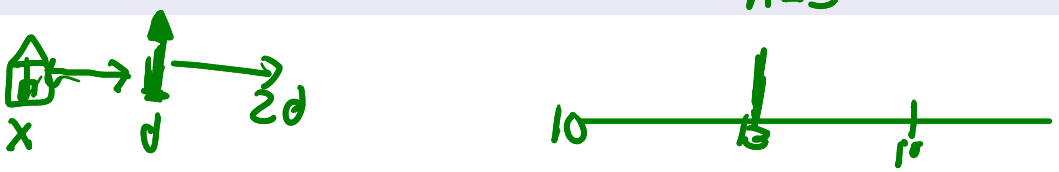


Cuatro casas y dos antenas

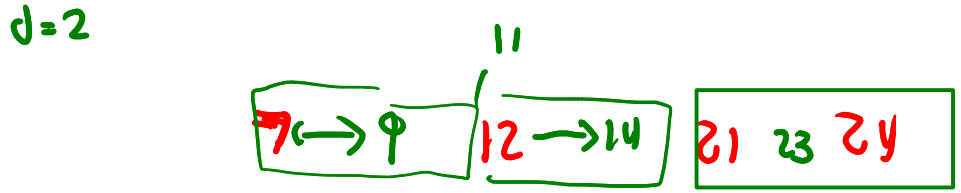
Casas en 12, 21, 7 y 24, con $M = 2$. La respuesta es 2,5, con las antenas en 9,5 y en 22.

Cómo se comprueba un candidato

Con las casas ordenadas, se toma la primera sin cubrir y se pone su antena lo más a la derecha posible, en casa $+ d$, que cubre hasta casa $+ 2d$. Se repite y se cuentan las antenas.



Dos candidatos, paso a paso



d	primera casa libre	antena en	cubre hasta
2	7	9	11
	12	14	16
	21	23	25

Tres antenas

Con $d = 2$ hacen falta tres y solo hay dos: $\varphi(2)$ es falso.

Dos candidatos, paso a paso

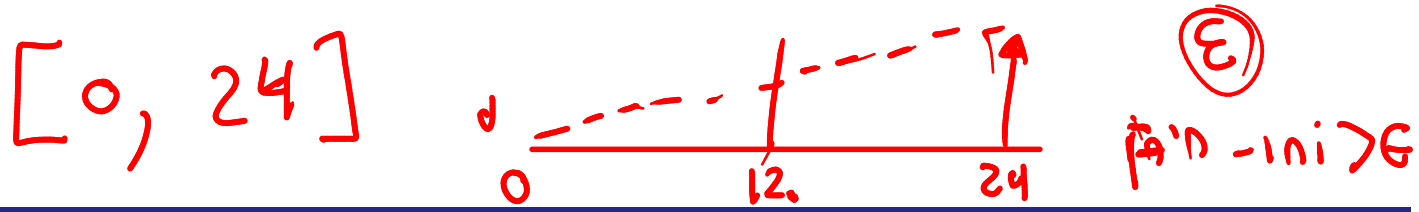


d	primera casa libre	antena en	cubre hasta
2,5	7	9,5	12
	21	23,5	26

Dos antenas

Con $d = 2{,}5$ bastan dos: $\varphi(2{,}5)$ es cierto. El borde está entre 2 y 2,5, y la bisección lo encuentra.

Los límites y las vueltas



Dónde buscar

Por debajo, $d = 0$: cada casa necesitaría su antena. Por arriba, la longitud de la calle: una sola antena cubre todo. La respuesta está en ese intervalo.

Cuándo parar

La respuesta es real, así que no hay un entero al que llegar. Se parte el intervalo a la mitad cien veces, que lo dejan en 10^{-30} del tamaño original. Cada vuelta recorre las casas una vez.

7 12 21 24

d=0 ——— 24

d=12 ←——— 15 ———→ $\phi(12) = T \leq 2$

3 28

[0, 12]

d=6 ←——— 13 ———→ ←——— 27 ———→ $\phi(6) = T$

7 19 21 33

d=3 ←——— 10 ———→ ←——— 24 ———→ $\phi(3) = T$

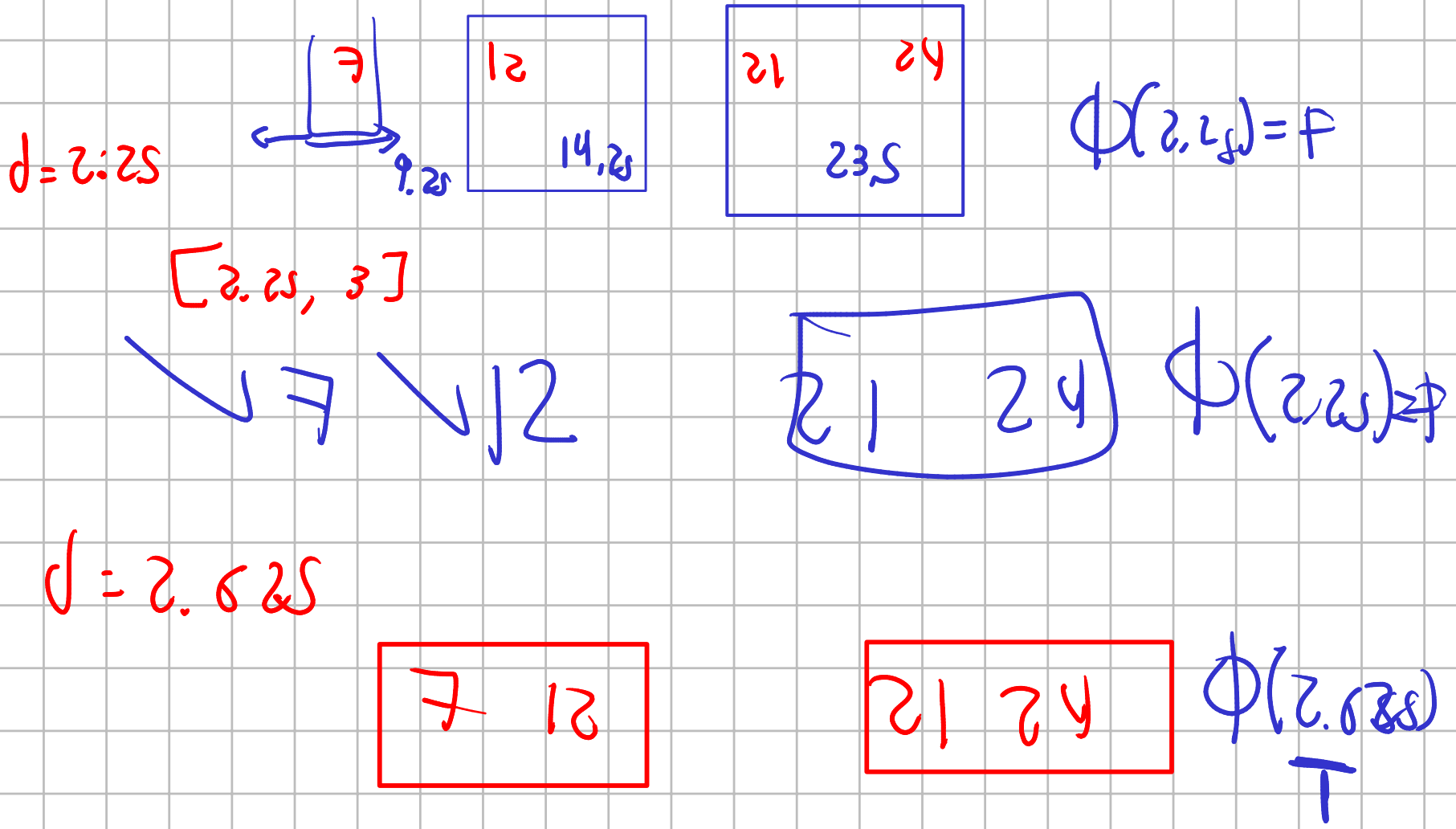
7 13 21 27

7 12 12 24

d=1.5 ←— 8.5 —→ ←— 13.5 —→ ←— 22.5 —→ $\phi(1.5) = F$

7 10 12 15 21 24

[1.5, 3]


d = 2.25
7
9.25
12
14,25
21
24
23,5
Φ(2,25) = F
[2.25, 3]
7
12
21
24
Φ(2,5) = F
d = 2.625
7
12
21
24
Φ(2.625) = T
[2,25, 2.625]

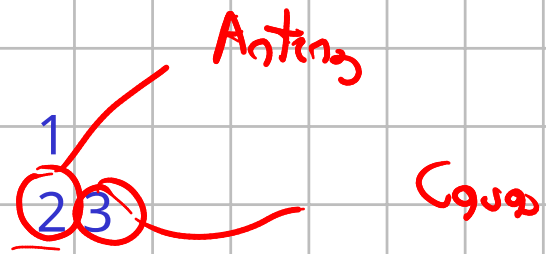
$d = 2.4375$ $A = 3$ $\phi(2.4375) = F$

$d = 2.5375$ $A = 2$ $\phi = T$

$d = 2.4875$ $A = 3$ $\phi = F$

$\varepsilon = 10^{-9}$

$d = 2.5$



1.0

1
2 3
1
3
10

1 3 10

[0, 10]

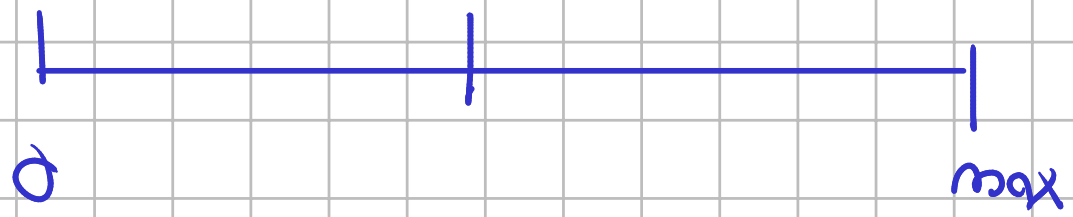
d=5

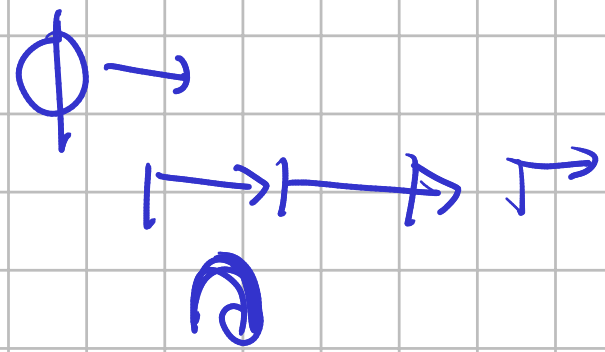
[0, 5]
[0, 2.5]
[0, 1.25]

[0.625, 1.25]
[0.9375, 1.25]
[0.9375, 1.0937]

[0.9375, 1,09375]

0.9375, 1.015625d





Finite ✓

10^{-3} arbitrary

$\frac{1}{2^n} < 10^{-3}$

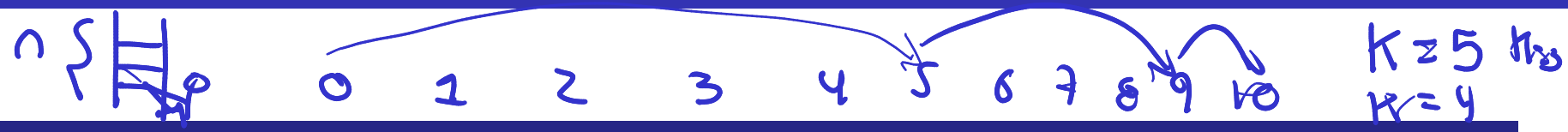
$O(\log(n) \times n)$

$\frac{1}{2} \to \frac{1}{4} \to \frac{1}{8} \to \frac{1}{2^n}$

$2^n < 10^3 \quad n \log(2) \leq 3 \log(10)$

$n = 3 \log(\epsilon)$

UVa 12032, The Monkey and the Oiled Bamboo



El problema

Una escalera tiene n peldaños a alturas $r_1 < \cdots < r_n$. Se sube de uno en uno, sin saltarse ninguno. Se arranca con fuerza k: ningún salto puede medir más de k, y el salto que mide exactamente k baja la fuerza en uno. Se pide el menor k con el que se llega arriba.

Antes de programar, conteste

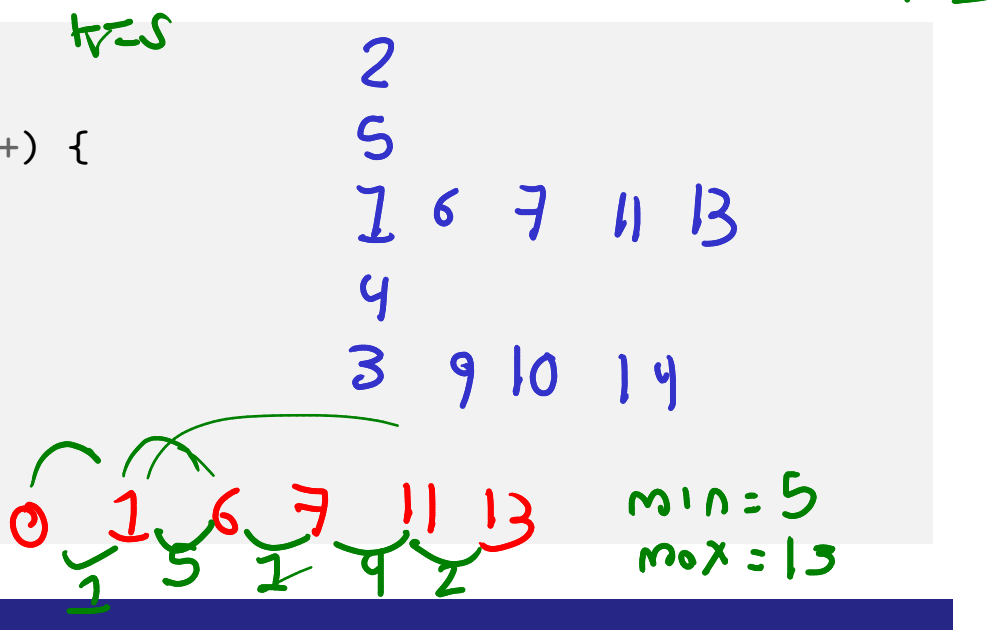
¿Cuál es $\varphi(k)$? ¿Por qué es monótona? ¿Entre qué dos valores está la respuesta?

Enunciado y envío: `https://vjudge.net/problem/UVA-12032`
La página pide cuenta para ver el enunciado.

La lectura de la entrada

https://vjudge.net/problem/UVA-12032

```
int casos;
scanf("%d", &casos);
for (int caso = 1; caso <= casos; caso++) {
  int n;
  scanf("%d", &n);
  vector<int> alturas(n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &alturas[i]);
  }
  /* aquí va la bisección sobre k */
```



Por qué sirve así

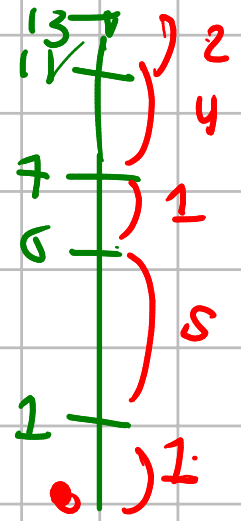
Las alturas vienen en una sola línea, pero `scanf` con %d se salta espacios y saltos de línea por igual, de modo que el mismo ciclo las lee sin importar cómo estén repartidas.

```python
import sys

datos = sys.stdin.read().split()
pos = 0
casos = int(datos[pos])
pos = pos + 1
caso = 1
while caso <= casos:
    n = int(datos[pos])
    pos = pos + 1
    alturas = []
    i = 0
    while i < n:
        alturas.append(int(datos[pos]))
        pos = pos + 1
        i = i + 1
    # el primer salto va desde el suelo, que está en 0
    saltos = [alturas[0]]
    i = 1
    while i < n:
        saltos.append(alturas[i] - alturas[i - 1])
        i = i + 1
    print('caso ' + str(caso) + ': ' + str(n) + ' peldaños')
    print('    alturas: ' + ' '.join(str(a) for a in alturas))
    print('    saltos:  ' + ' '.join(str(s) for s in saltos))
    caso = caso + 1
```

```cpp
int casos;
scanf("%d", &casos);
for (int caso = 1; caso <= casos; caso++) {
  int n;
  scanf("%d", &n);
  vector<int> alturas(n);
  for (int i = 0; i < n; i++) {
    scanf("%d", &alturas[i]);
  }
```

2
5
1 6 7 11 13
4
3 9 10 14



$\phi(k) \in$ Si es posible recorrer la lista

$[5, 13]$

5

0 — 1 — 6 — 7 — 11 — 13

1 5 1 4 2

K 5 4 4 3 3

$\Theta(n)$

Contenido

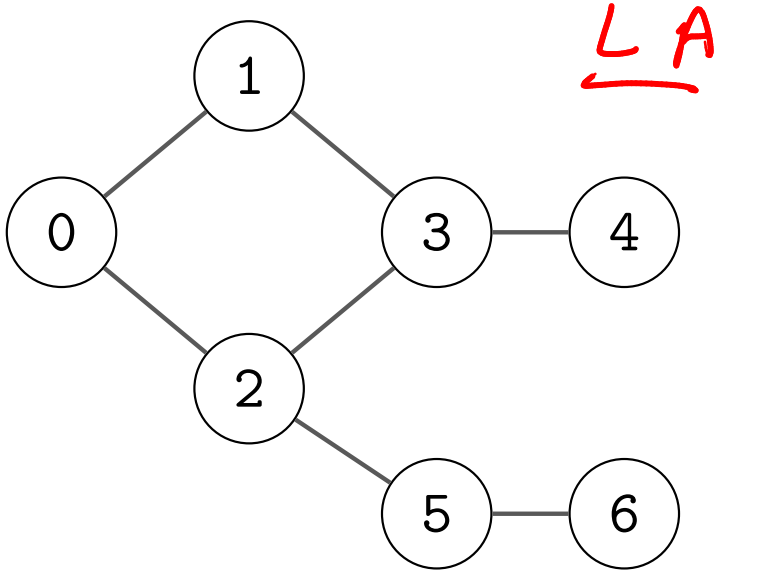
1 Bisección: encontrar la función

2 Recorrido en amplitud con tope de pasos
- Concepto
- Ejemplo
- Ejercicio

3 Recorrido en profundidad y ciclos

4 Problemas para practicar

La lista de adyacencia



Mat. L A

Nodo	adj[u]
0	1, 2
1	0, 3
2	0, 3, 5
3	1, 2, 4
4	3
5	2, 6
6	5

Cada nodo guarda sus vecinos

En un grafo no dirigido, cada arista aparece en dos listas. Recorrer los vecinos de u cuesta lo que mida su lista.

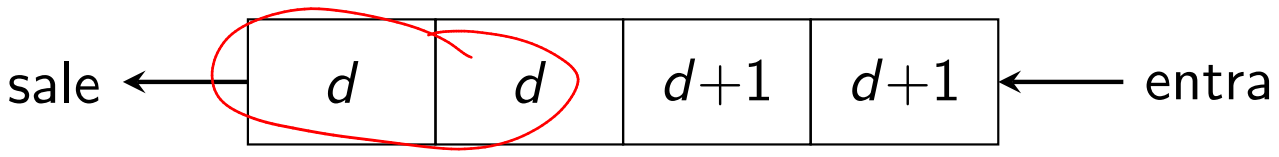
El recorrido en amplitud

Una vuelta del ciclo

1. Sale el primer nodo u de la cola.
2. Se miran sus vecinos en `adj[u]`.
3. El vecino sin tocar recibe $dist[u] + 1$ y entra al final de la cola.

El ciclo termina cuando la cola queda vacía.

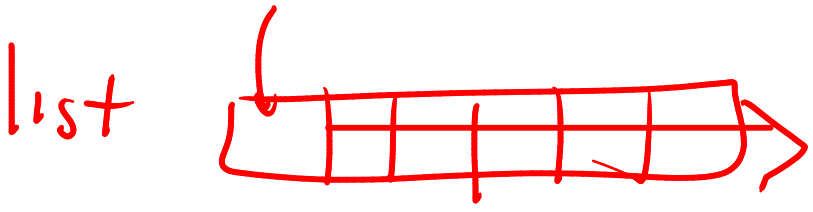
La distancia sale del primer toque



La cola queda en orden de distancia

Primero salen todos los de distancia d, y los que estos tocan entran detrás con $d + 1$. Así, el primero que toca un nodo es uno de los más cercanos, y esa distancia ya es la menor. La distancia aquí cuenta aristas.

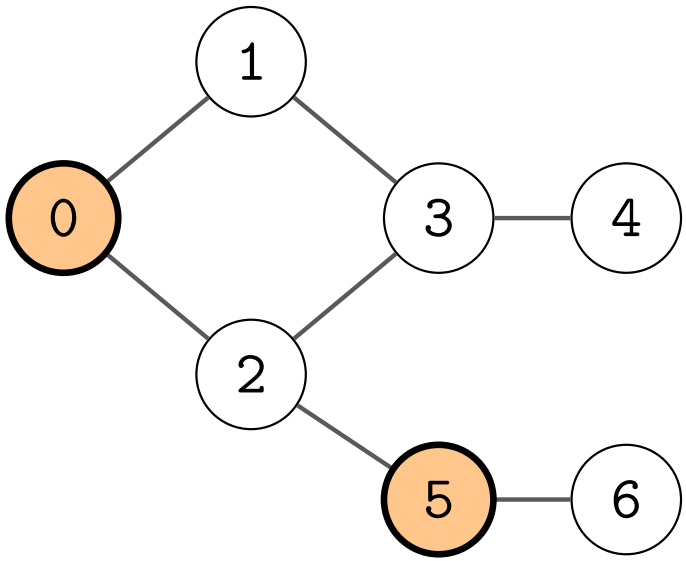
Lo que cuesta y dónde aplica



$O(V + E)$

Cada nodo entra una vez a la cola y su lista se recorre una vez. Sacar de la cola cuesta $O(1)$ con deque en Python y con queue en C++. Se reconoce porque las aristas no tienen peso y la pregunta habla de pasos, saltos o días: la menor cantidad, o quién queda a K o menos.

Un contagio con dos focos y $K = 2$

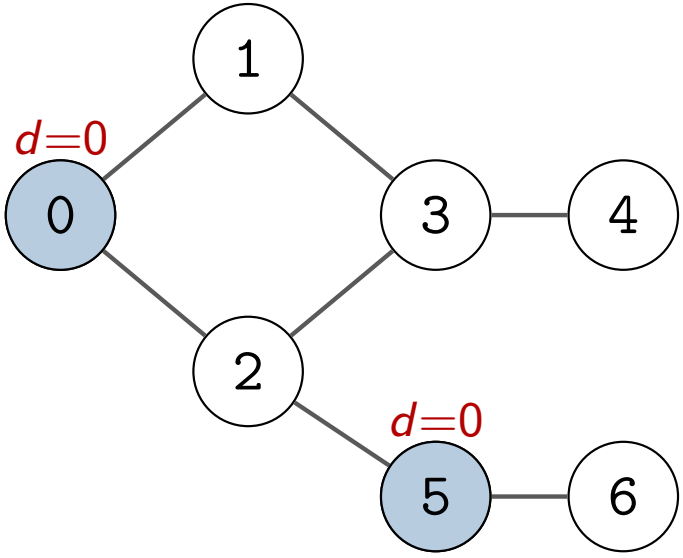


Sale	d	Toca	Cola

El problema

Cada nodo es una persona y cada arista un contacto. Los focos son 0 y 5. Cada día la enfermedad pasa de un enfermo a sus vecinos. ¿A quién ha llegado al cabo de $K = 2$ días?

Un contagio con dos focos y $K = 2$

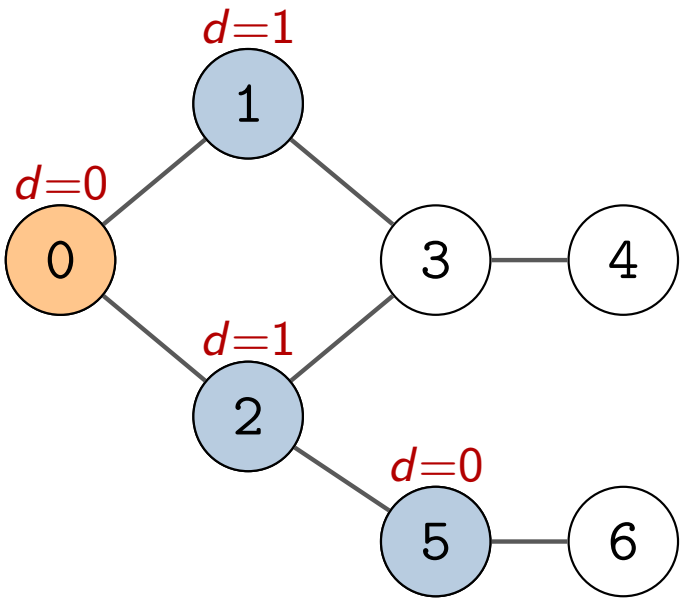


Sale	d	Toca	Cola
inicio		0, 5	0 5

Las dos fuentes

Los dos focos entran a la cola con distancia 0, antes de que empiece el ciclo.

Un contagio con dos focos y $K = 2$

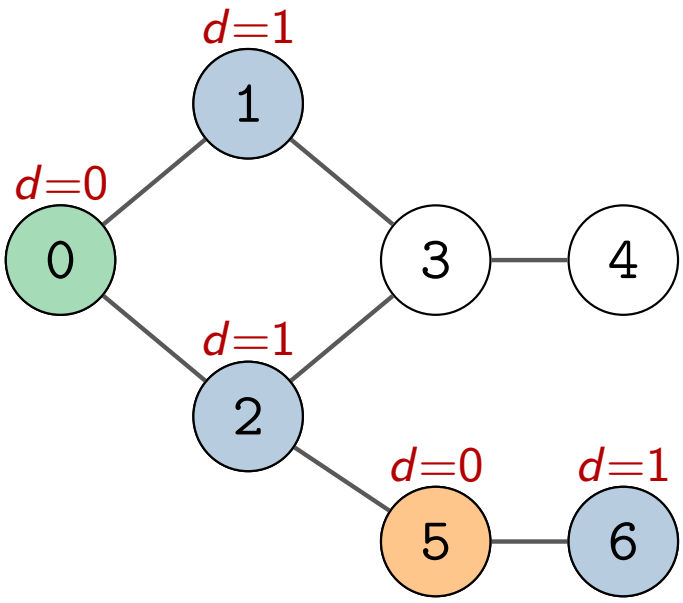


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2

Sale el nodo 0

Como $d = 0 < K$, se expande: 1 y 2 reciben $d = 1$ y van al final de la cola.

Un contagio con dos focos y $K = 2$

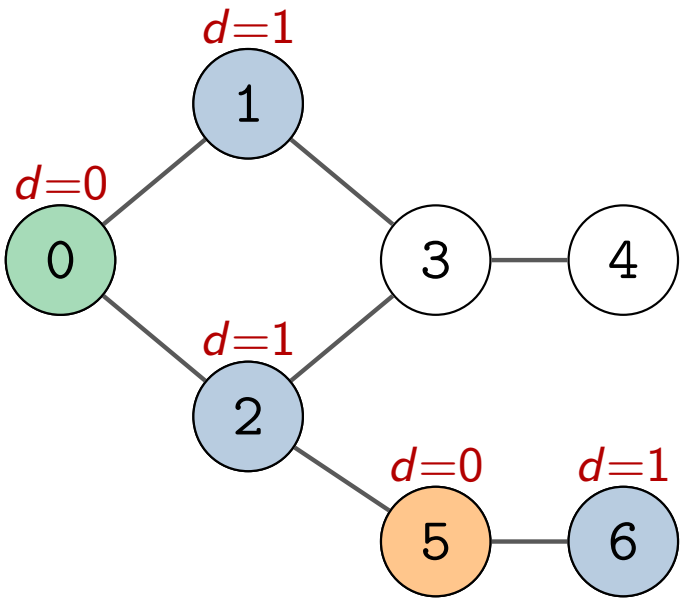


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6

Sale el nodo 5

También tiene $d = 0$. Su vecino 6 recibe $d = 1$. Su vecino 2 ya tiene distancia y se deja como está.

Un contagio con dos focos y $K = 2$

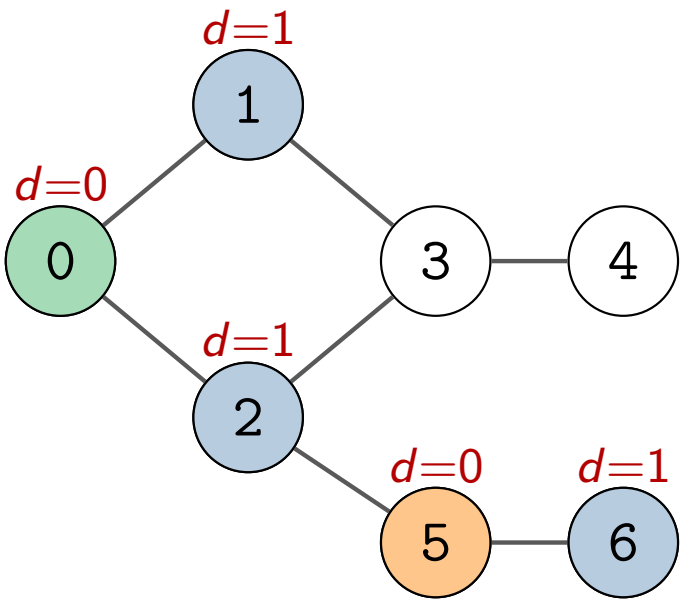


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6

Pregunta

¿Por qué el nodo 2 no se vuelve a poner en la cola?

Un contagio con dos focos y $K = 2$

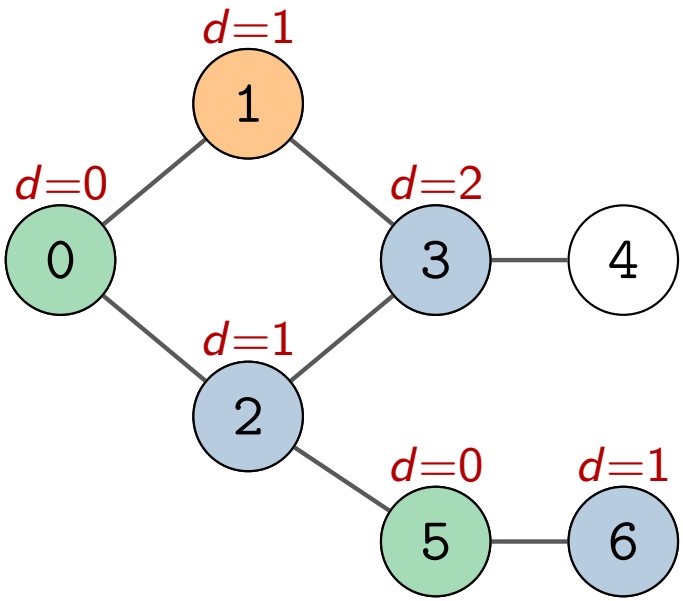


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6

Respuesta

La cola sale en orden de distancia, de modo que el primer toque ya dio la menor. Cada nodo entra una sola vez y se expande una sola vez, y de ahí sale el costo $O(V + E)$.

Un contagio con dos focos y $K = 2$

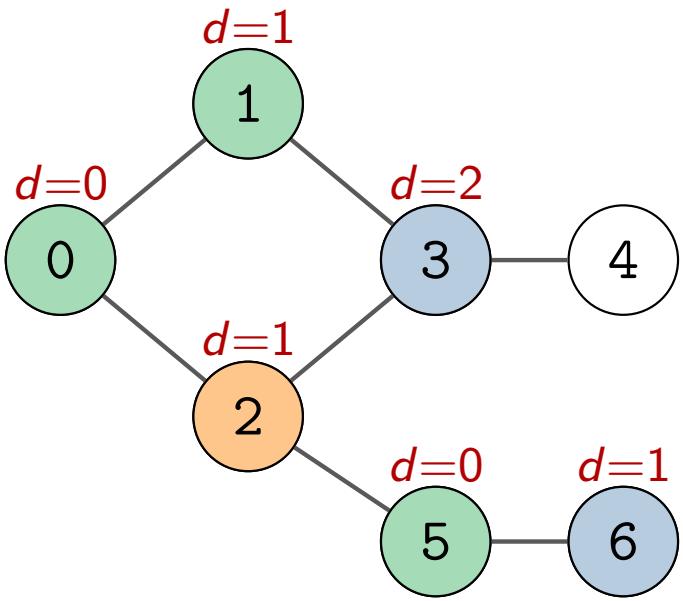


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6
1	1	3	2 6 3

Sale el nodo 1

Con $d = 1 < K$ se expande: 3 recibe $d = 2$. El vecino 0 ya tenía distancia.

Un contagio con dos focos y $K = 2$

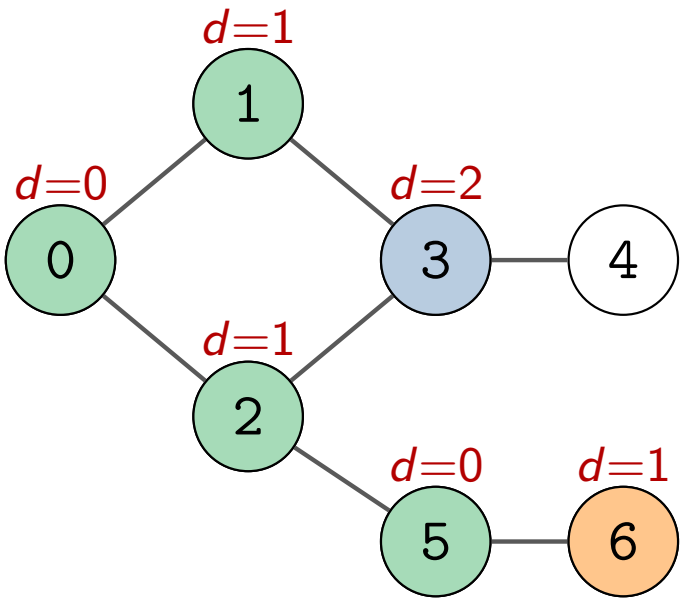


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6
1	1	3	2 6 3
2	1		6 3

Sale el nodo 2

Sus vecinos 0, 3 y 5 ya tienen distancia, así que no entra nadie a la cola.

Un contagio con dos focos y $K = 2$

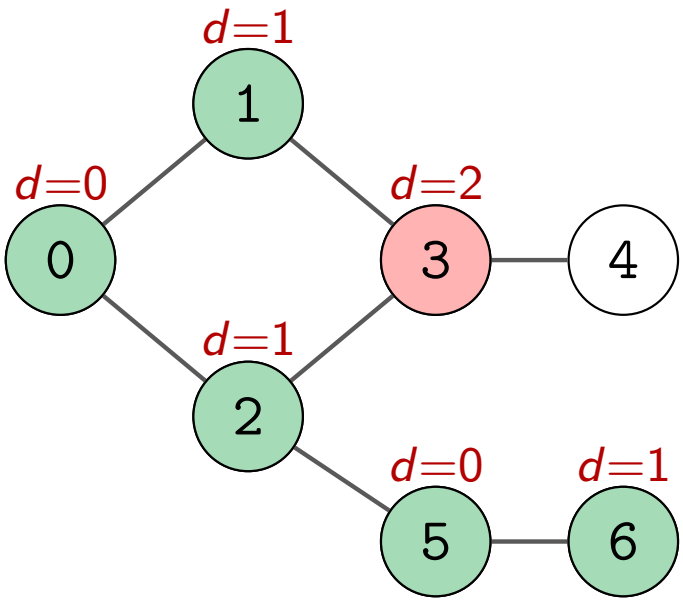


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6
1	1	3	2 6 3
2	1		6 3
6	1		3

Sale el nodo 6

Su único vecino, 5, ya tiene distancia.

Un contagio con dos focos y $K = 2$

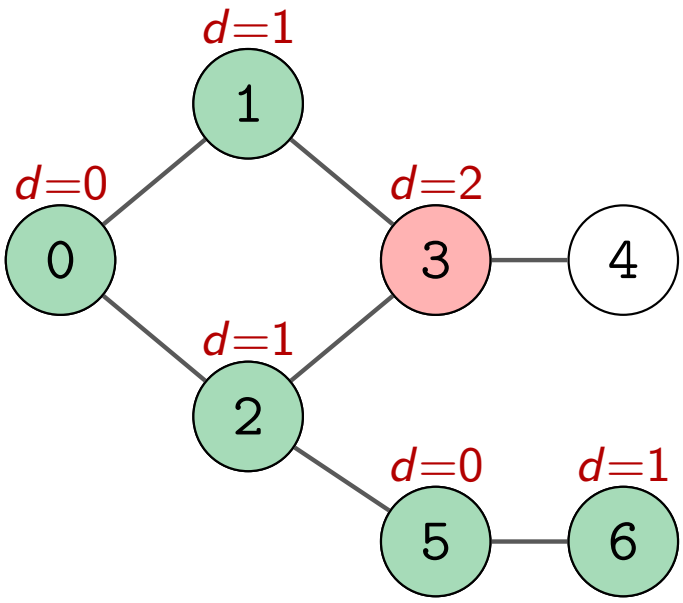


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6
1	1	3	2 6 3
2	1		6 3
6	1		3
3	2		

Sale el nodo 3

Aquí $d = 2 = K$ y no se expande: su vecino 4 se queda sin tocar. La cola queda vacía y el ciclo termina.

Un contagio con dos focos y $K = 2$

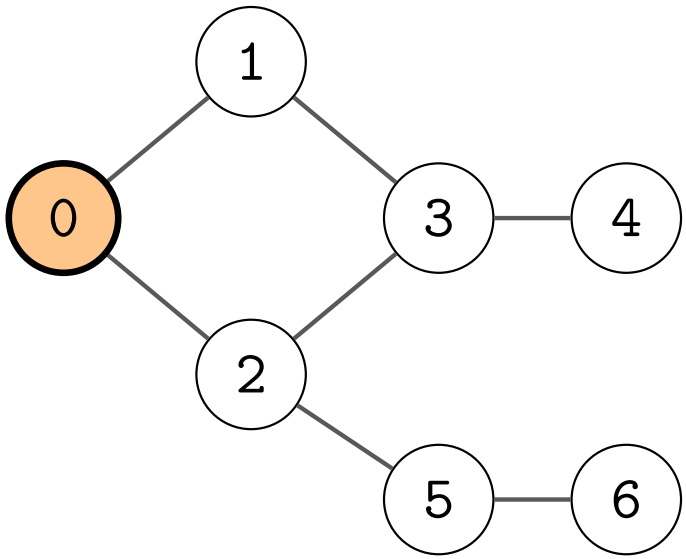


Sale	d	Toca	Cola
inicio		0, 5	0 5
0	0	1, 2	5 1 2
5	0	6	1 2 6
1	1	3	2 6 3
2	1		6 3
6	1		3
3	2		

El resultado

La enfermedad llegó a 0, 1, 2, 3, 5 y 6. El nodo 4 está a distancia 3 de los dos focos.

Y con un solo foco

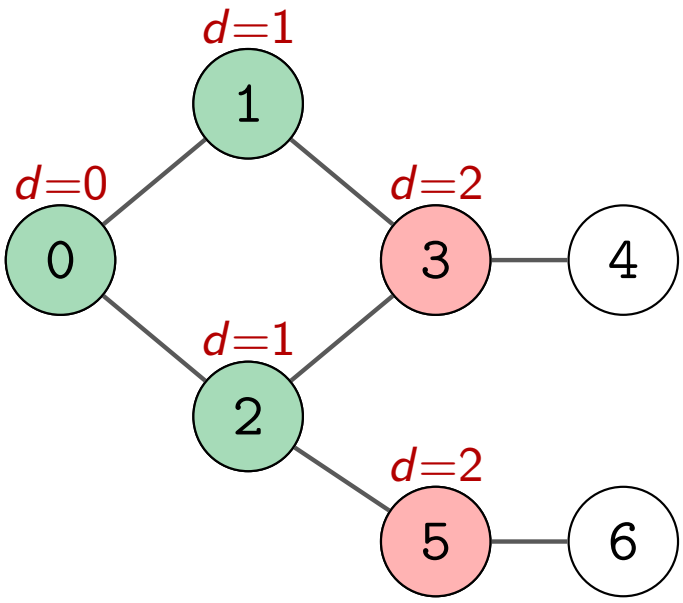


Sale	d	Toca	Cola

Pregunta

Si el único foco es el nodo 0, con el mismo $K = 2$, ¿qué cambia en el recorrido?

Y con un solo foco



Sale	d	Toca	Cola
inicio		0	0
0	0	1, 2	1 2
1	1	3	2 3
2	1	5	3 5
3	2		5
5	2		

Respuesta

Solo el arranque: entra un nodo y no dos, y el ciclo es el mismo. Ahora 5 recibe $d = 2$ y no se expande, así que 6 se queda sin tocar, igual que 4.

El recorrido con tope, en Python

```
def bfs(adj, inicio, k):
    # Cuántos nodos quedan a distancia menor o igual a k desde inicio.
    dist = [-1] * len(adj)
    cola = deque()
    dist[inicio] = 0
    cola.append(inicio)
    alcanzados = 1
    while len(cola) > 0:
        u = cola.popleft()
        if dist[u] < k:
            for v in adj[u]:
                if dist[v] == -1:
                    dist[v] = dist[u] + 1
                    alcanzados = alcanzados + 1
                    cola.append(v)
    return alcanzados
```

Limite

El tope

La condición `dist[u] < k` deja el nodo a distancia k contado y sin expandir.

El recorrido con tope, en C++

```cpp
int bfs(const vector<vector<int> > &adj, int inicio, int k) {
  // Cuantos nodos quedan a distancia menor o igual a k desde inicio.
  vector<int> dist(adj.size(), -1);
  queue<int> cola;
  dist[inicio] = 0;
  cola.push(inicio);
  int alcanzados = 1;
  while (!cola.empty()) {
    int u = cola.front();
    cola.pop();
    if (dist[u] < k) {
      for (size_t i = 0; i < adj[u].size(); i++) {
        int v = adj[u][i];
        if (dist[v] == -1) {
          dist[v] = dist[u] + 1;
          alcanzados = alcanzados + 1;
          cola.push(v);
        }
      }
    }
  }
  return alcanzados;
}
```

No ha sido descubierto

UVa 336, A Node Too Far

El problema

Cada mensaje de una red lleva un TTL. La estación que lo recibe le resta uno y lo reenvía solo si el resultado no es cero, de modo que llega a los nodos a distancia menor o igual al TTL. Por cada consulta, se pide cuántos nodos de la red no alcanza.

Enunciado y envío: `https://vjudge.net/problem/UVA-336`
La página pide cuenta para ver el enunciado.

La entrada

Varias redes

Cada red empieza con su número de conexiones, seguido de esos pares de nodos. Los nodos son enteros positivos, con un máximo de 30 por red. Después vienen consultas de la forma `nodo TTL`, que terminan en `0 0`. Un número de conexiones igual a cero termina la entrada.

La salida

Una línea por consulta

`Case 1: 5 nodes not reachable from node 35 with TTL = 2.`

El número del caso sigue contando de una red a la siguiente. La frase no tiene singular: la muestra del enunciado escribe `1 nodes`.

La lectura de la entrada

```
int conexiones;
int caso = 0;
while (scanf("%d", &conexiones) == 1 && conexiones != 0) {
  for (int i = 0; i < conexiones; i++) {
    int a, b;
    scanf("%d %d", &a, &b);
    /* aquí va el diccionario de número a índice */
  }
  int nodo, ttl;
  while (scanf("%d %d", &nodo, &ttl) == 2 && (nodo != 0 || ttl != 0)) {
    caso = caso + 1;
    /* aquí va el recorrido en amplitud */
  }
}
```

Dónde se detiene cada ciclo

Las consultas paran cuando los dos valores son cero, porque 35 0 es una consulta válida.

Cómo atacarlo

El parámetro y la trampa

El parámetro es el TTL, un tope de pasos. Como el recorrido visita por distancia creciente, basta con no expandir los nodos a distancia TTL. La trampa está en los números de los nodos: pueden ser cualquier entero positivo, y hace falta un diccionario de número a índice. Una consulta puede nombrar un nodo que no esté en la red.

A resolver en el computador

UVa 336

1. Arme la lista de adyacencia con el diccionario.
2. Corra el recorrido con tope una vez por consulta.
3. Responda con el total de nodos menos los alcanzados.

La muestra del enunciado da 5, 1, 8, 5, 3 y 1.

Contenido

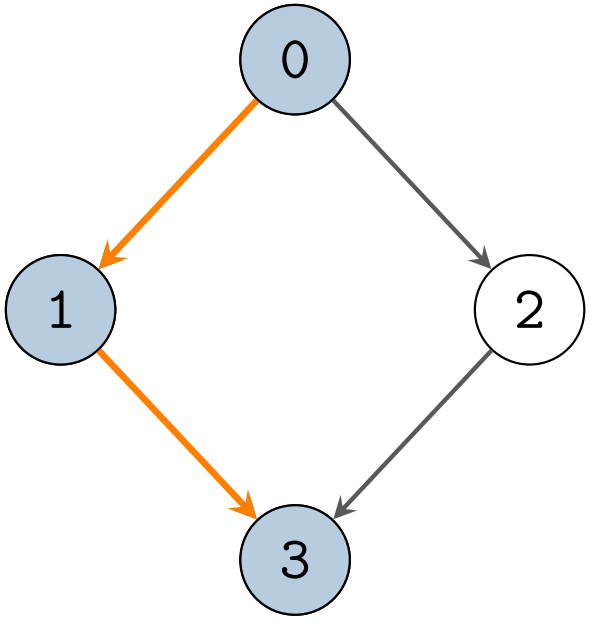
1 Bisección: encontrar la función

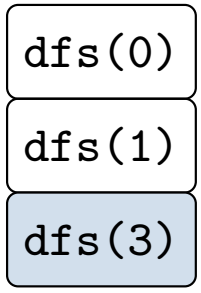
2 Recorrido en amplitud con tope de pasos

3 Recorrido en profundidad y ciclos
- Concepto
- Ejemplo
- Ejercicio

4 Problemas para practicar

La llamada recursiva es el camino actual





la que corre

Un nodo por llamada abierta

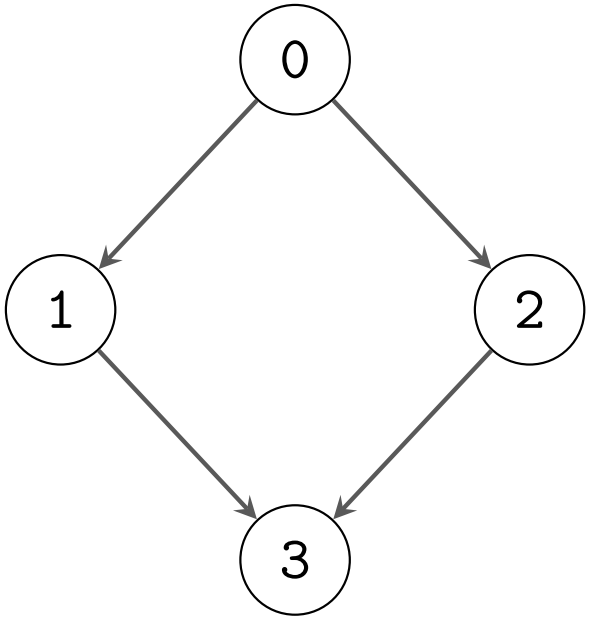
`dfs(u)` marca u y se llama sobre cada vecino sin visitar. Las llamadas abiertas forman el camino del inicio al nodo actual, y al devolver, el nodo sale del camino.

Lo que cuesta y dónde aplica

$O(V + E)$

En un grafo dirigido, la arista $a \to b$ va solo en la lista de a. Cada nodo tiene una llamada y cada arista se mira una vez, como en amplitud. Este recorrido no da distancias: sigue un camino hasta el fondo antes de volver. Se reconoce cuando la pregunta depende del camino, como los ciclos y los órdenes con dependencias.

¿Basta con visitado?

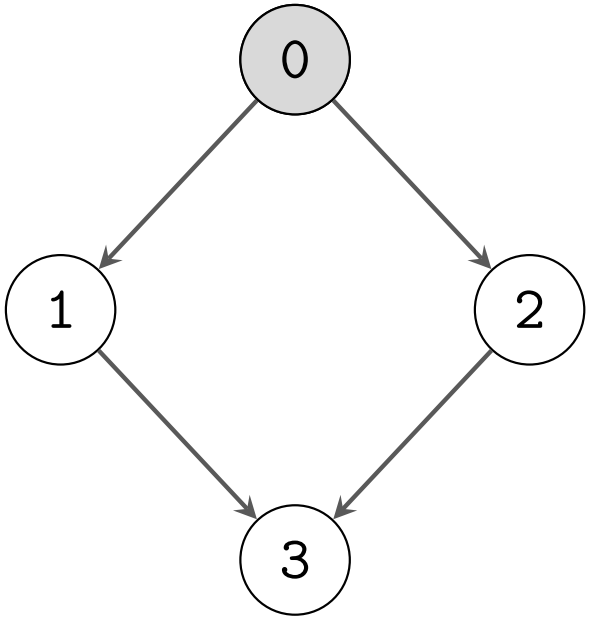


Paso	Destino	Visitados

La primera idea

Para saber si un grafo dirigido tiene un ciclo, se marca cada nodo como visitado al llegar, y se declara ciclo cuando una arista llega a un nodo ya visitado. ¿Basta con eso? Se prueba sobre este grafo.

Solo con visitado

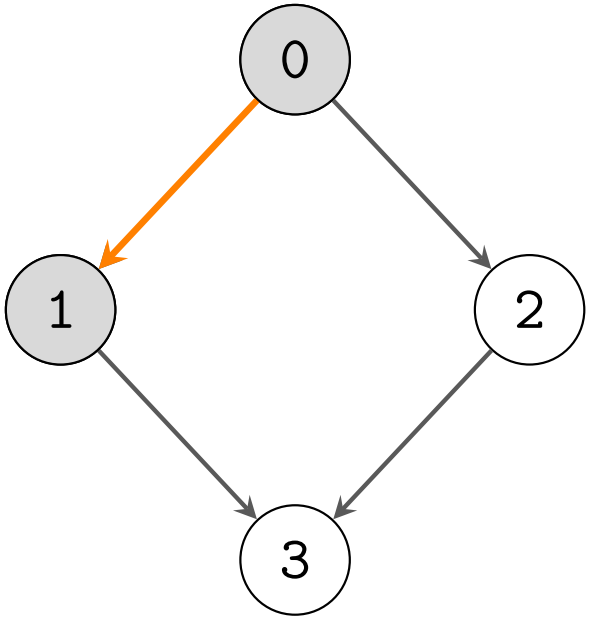


Paso	Destino	Visitados
entra 0		0

Se empieza en 0

Se llama a `dfs` desde 0 y se marca como visitado.

Solo con visitado

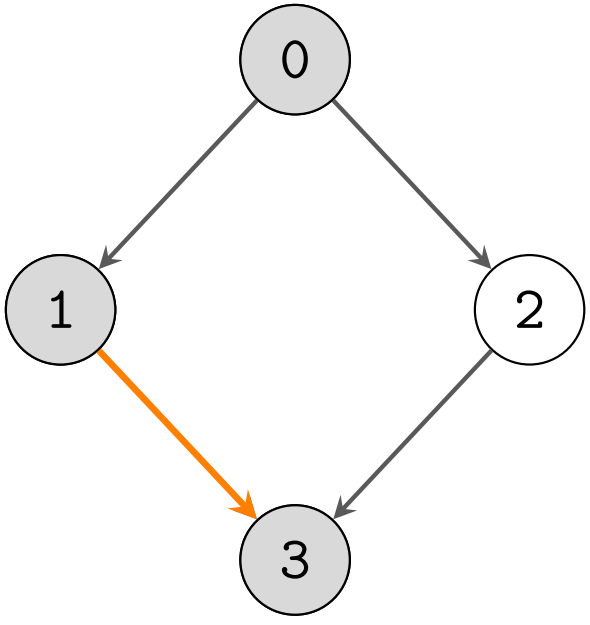


Paso	Destino	Visitados
entra 0		0
$0 \rightarrow 1$	sin visitar	0 1

De 0 a 1

El nodo 1 no estaba visitado: se marca y se baja.

Solo con visitado

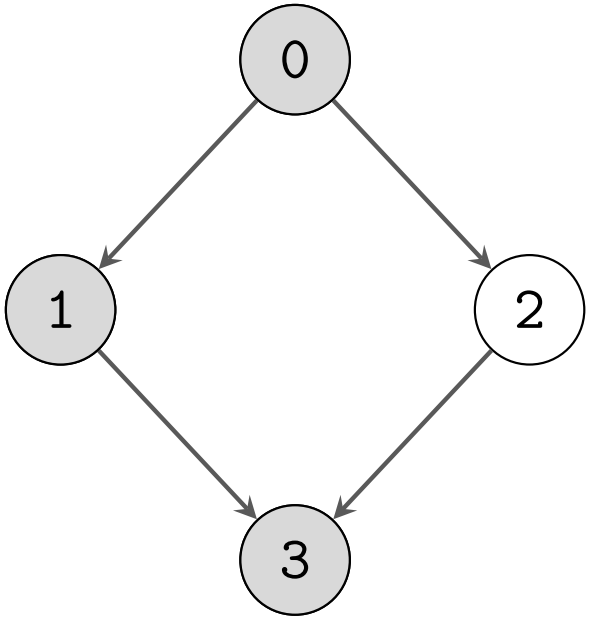


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3

De 1 a 3

El nodo 3 tampoco: se marca y se baja.

Solo con visitado

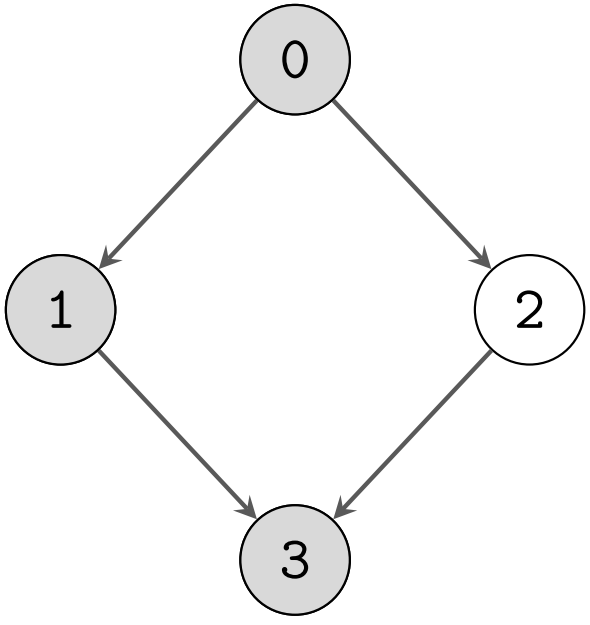


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
vuelve de 3		0 1 3

Vuelve de 3

El nodo 3 no tiene aristas de salida y su llamada devuelve.

Solo con visitado

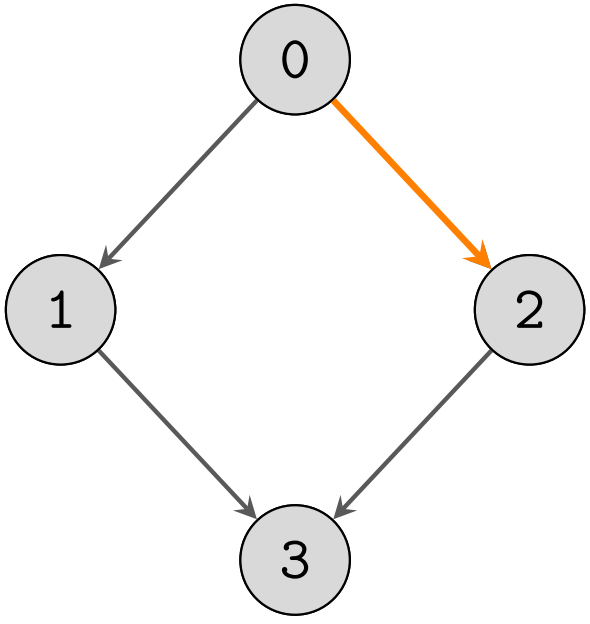


Paso	Destino	Visitados
entra 0		0
$0 \rightarrow 1$	sin visitar	0 1
$1 \rightarrow 3$	sin visitar	0 1 3
vuelve de 3		0 1 3
vuelve de 1		0 1 3

Vuelve de 1

Al nodo 1 no le quedan aristas. Se regresa a 0.

Solo con visitado

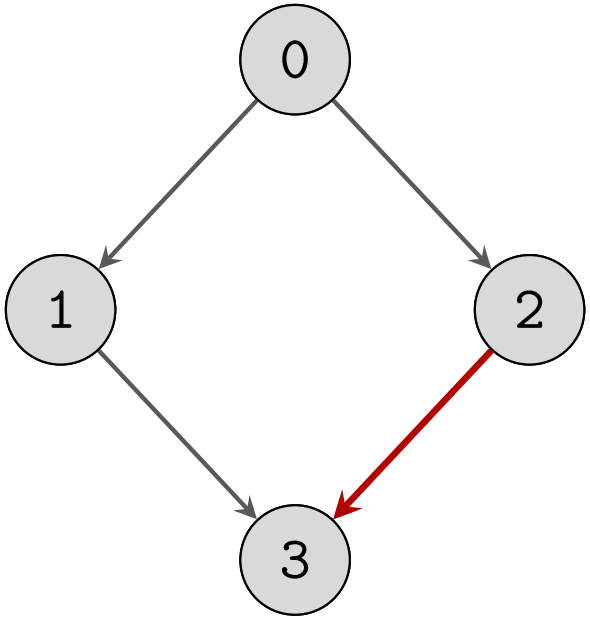


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
vuelve de 3		0 1 3
vuelve de 1		0 1 3
$0 \to 2$	sin visitar	0 1 2 3

De 0 a 2

El nodo 2 no estaba visitado: se marca y se baja.

Solo con visitado

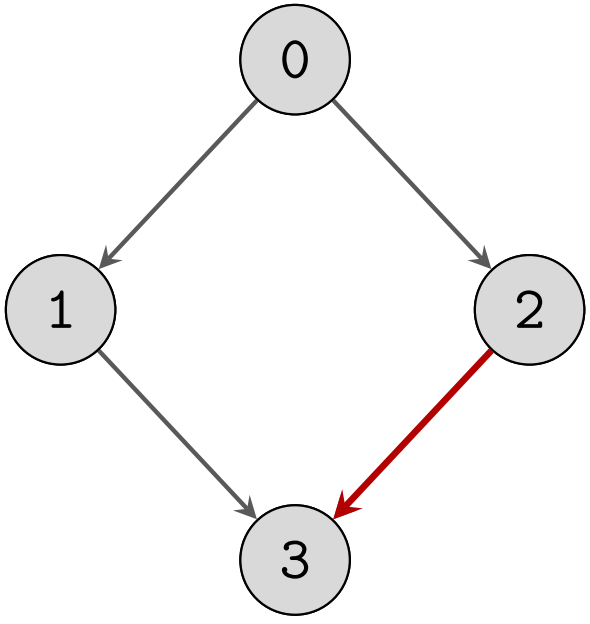


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
vuelve de 3		0 1 3
vuelve de 1		0 1 3
$0 \to 2$	sin visitar	0 1 2 3
$2 \to 3$	visitado	0 1 2 3

De 2 a 3

El nodo 3 ya está visitado, y con esta regla eso se declara ciclo.

Solo con visitado

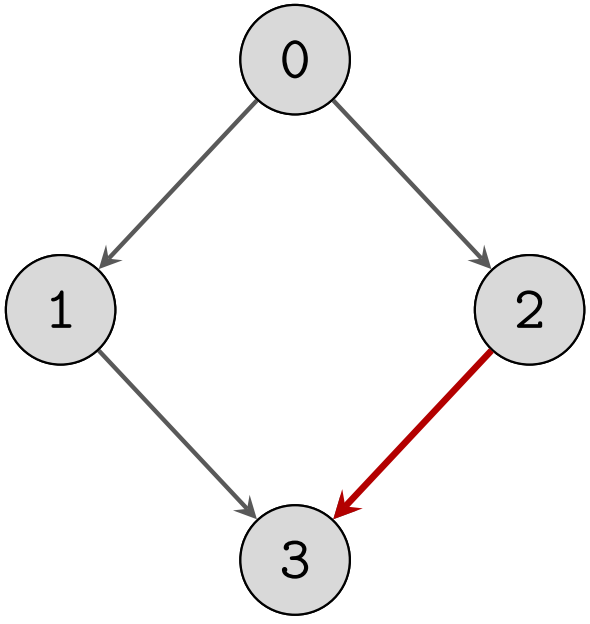


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
vuelve de 3		0 1 3
vuelve de 1		0 1 3
$0 \to 2$	sin visitar	0 1 2 3
$2 \to 3$	visitado	0 1 2 3

Pregunta

¿Hay un ciclo en este grafo?

Solo con visitado

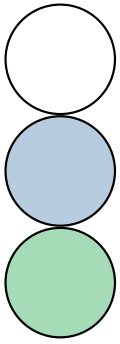


Paso	Destino	Visitados
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
vuelve de 3		0 1 3
vuelve de 1		0 1 3
$0 \to 2$	sin visitar	0 1 2 3
$2 \to 3$	visitado	0 1 2 3

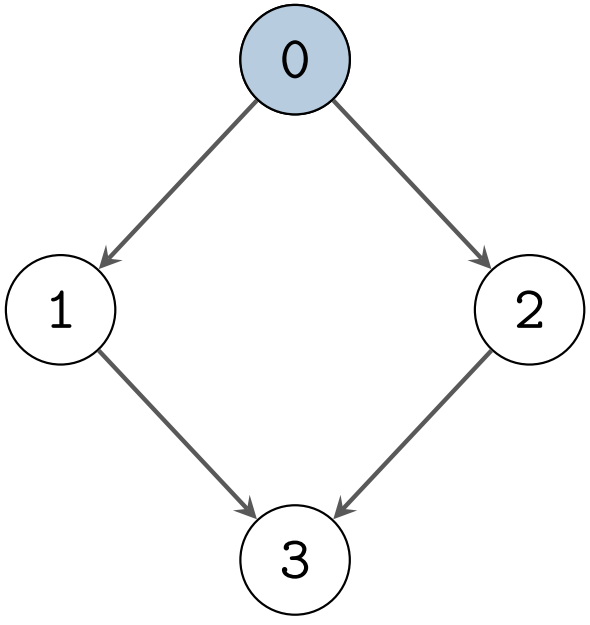
Respuesta

No: todas las aristas bajan y ningún camino sube de vuelta. El nodo 3 se marcó por otra rama, cuya llamada ya terminó. El camino actual es 0 2, y 3 no está en él.

Tres estados

	Estado de v	Su llamada	La arista $u \to v$
	sin visitar	no ha empezado	baja a v
	en camino	sigue abierta	cierra un ciclo
	terminado	ya devolvió	no hace nada

El diamante con tres estados

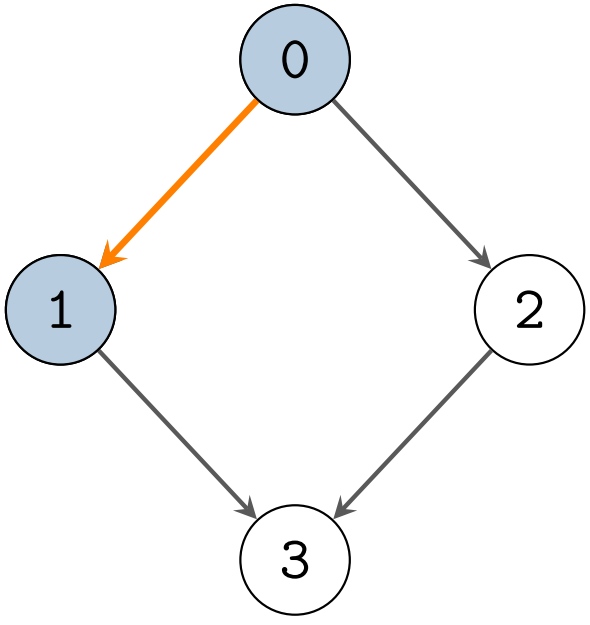


Paso	Destino	Camino
entra 0		0

Se empieza en 0

El nodo 0 pasa a en camino, y el camino actual es solo 0.

El diamante con tres estados

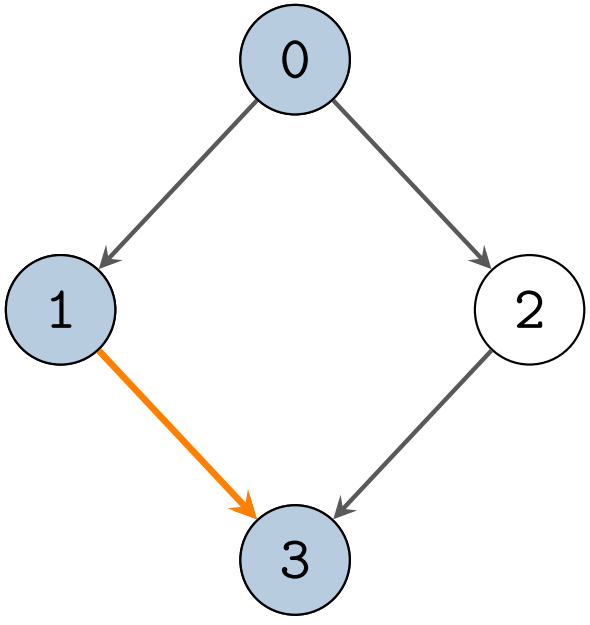


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1

De 0 a 1

El nodo 1 está sin visitar: se baja, y queda en camino.

El diamante con tres estados

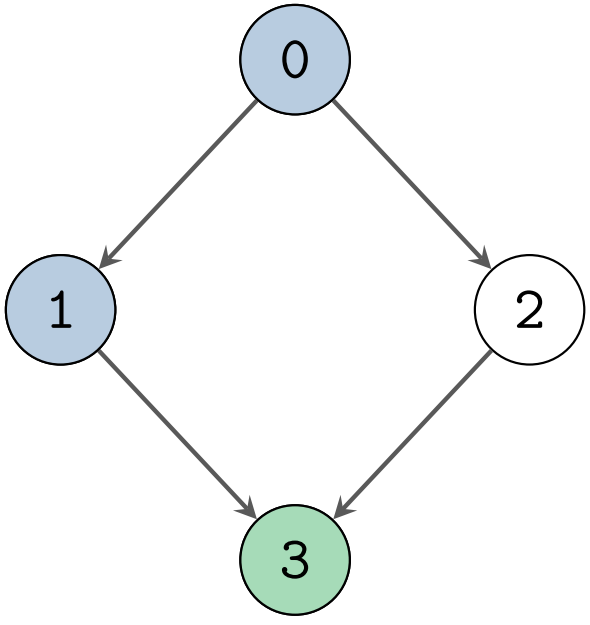


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3

De 1 a 3

El nodo 3 está sin visitar: se baja.

El diamante con tres estados

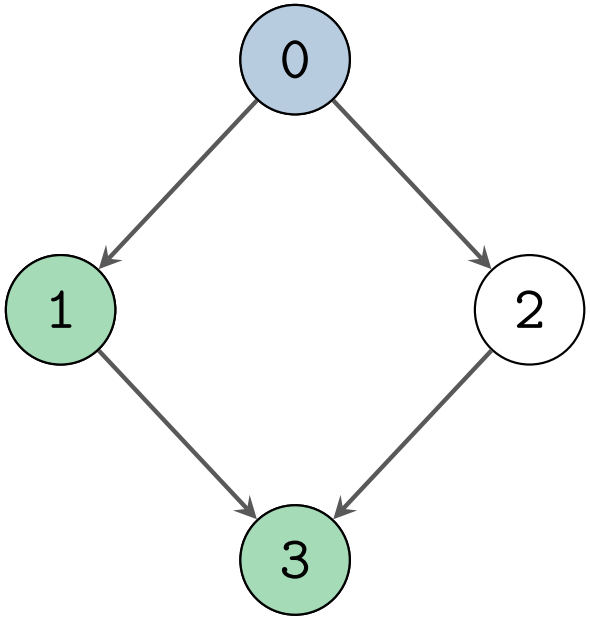


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1

Termina 3

No tiene aristas de salida: termina y sale del camino.

El diamante con tres estados

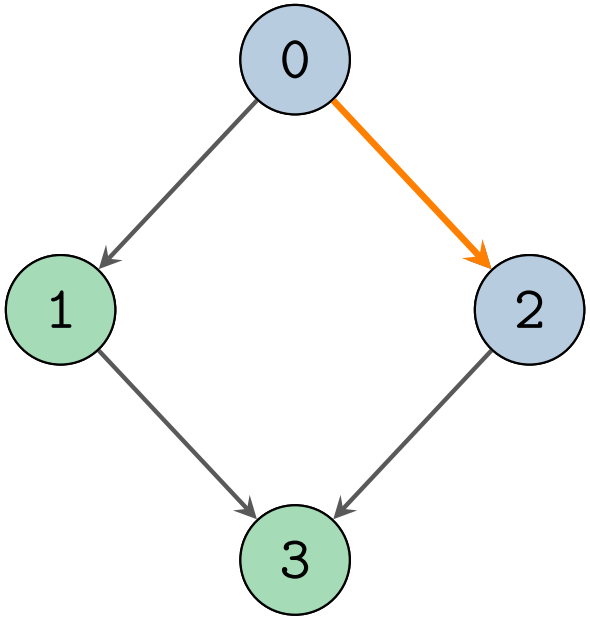


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1
termina 1		0

Termina 1

No le quedan aristas: termina y sale del camino.

El diamante con tres estados

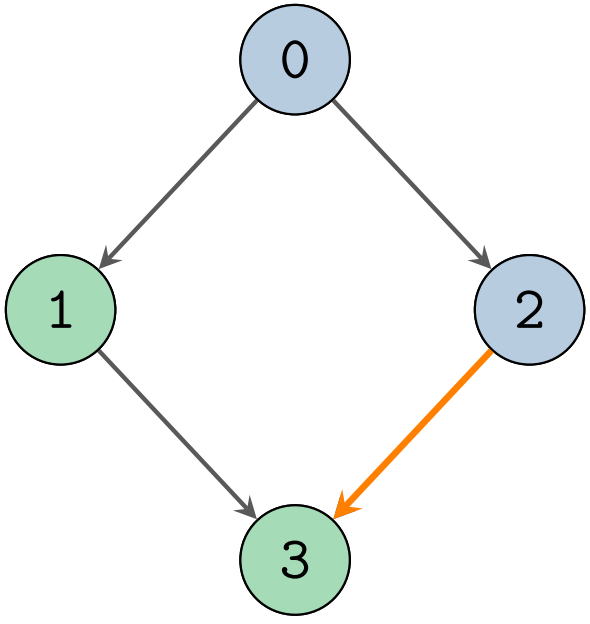


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2

De 0 a 2

El nodo 2 está sin visitar: se baja.

El diamante con tres estados

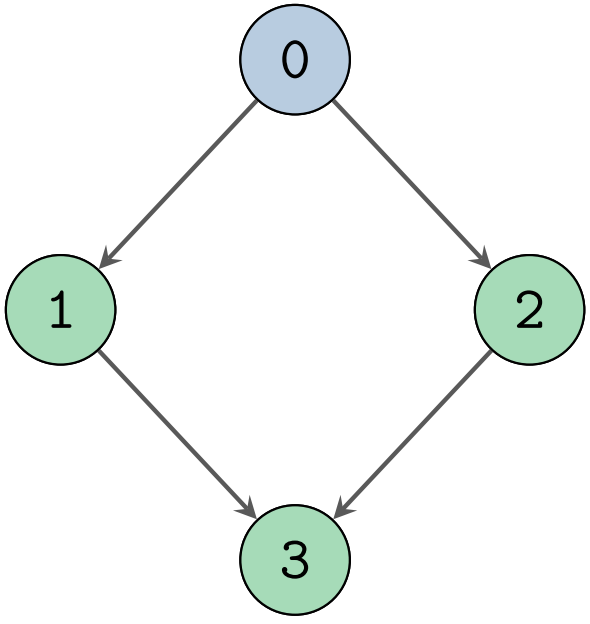


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2

De 2 a 3

El nodo 3 está terminado y ya no pertenece al camino. Esta arista no cierra nada.

El diamante con tres estados

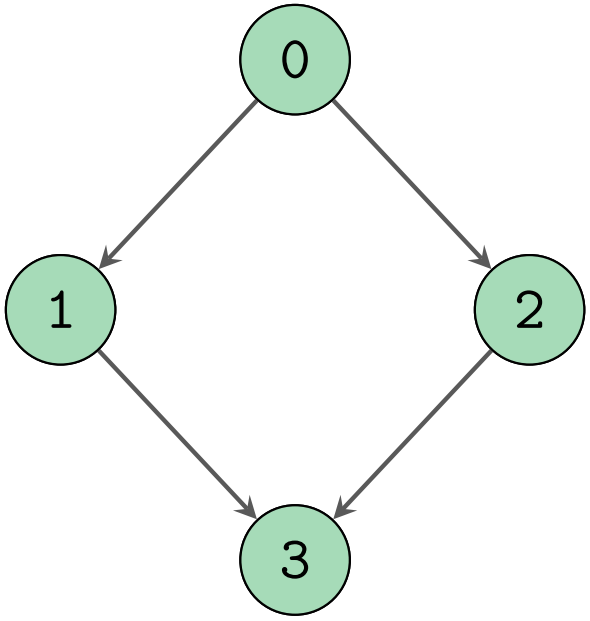


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2
termina 2		0

Termina 2

No le quedan aristas: termina y sale del camino.

El diamante con tres estados

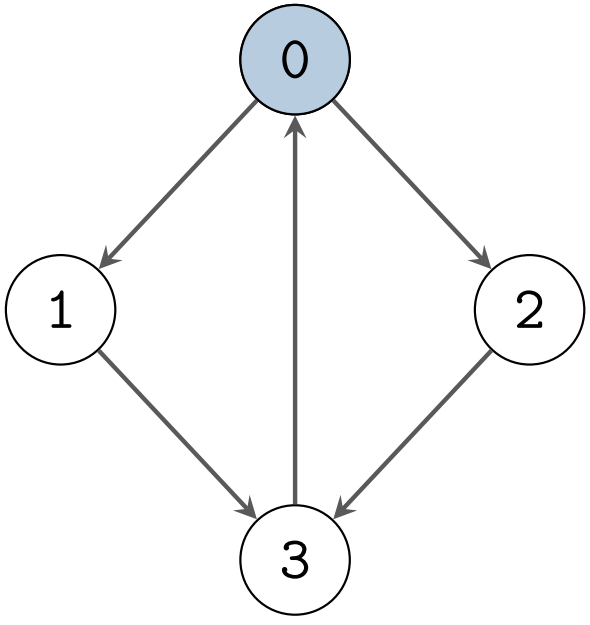


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2
termina 2		0
termina 0		

Termina 0

El camino queda vacío y ninguna arista llegó a un nodo en camino: no hay ciclo.

Una arista más: de 3 a 0

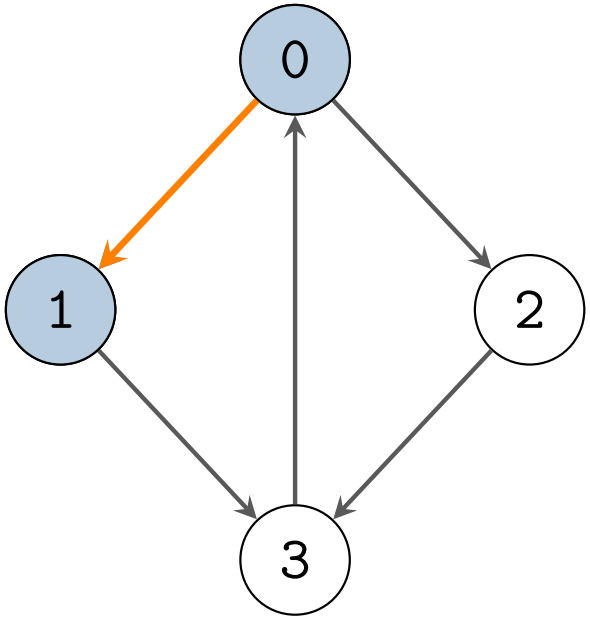


Paso	Destino	Camino
entra 0		0

Se empieza en 0

El mismo diamante, con una arista más: de 3 a 0.

Una arista más: de 3 a 0

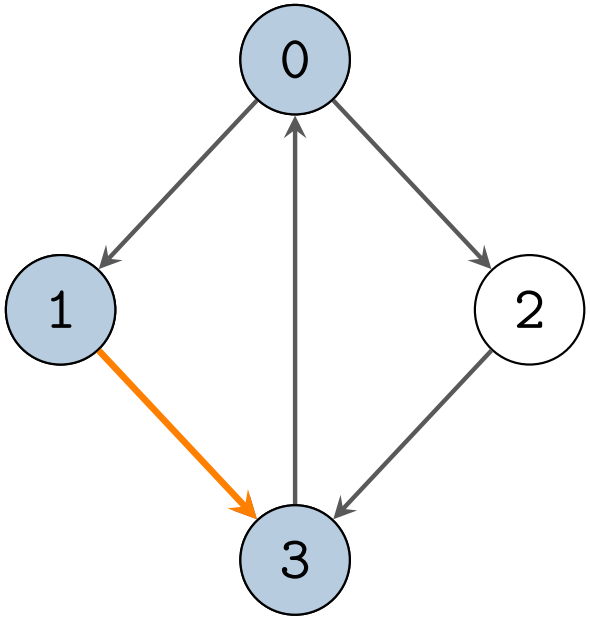


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1

De 0 a 1

El nodo 1 está sin visitar: se baja.

Una arista más: de 3 a 0

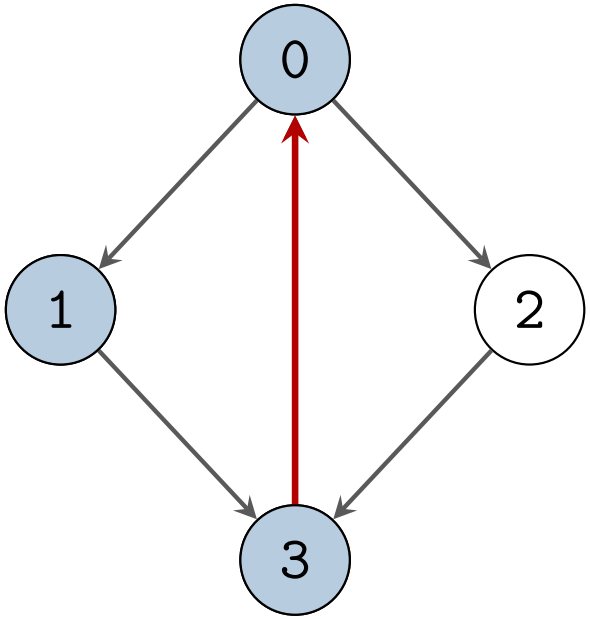


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3

De 1 a 3

El nodo 3 está sin visitar: se baja.

Una arista más: de 3 a 0

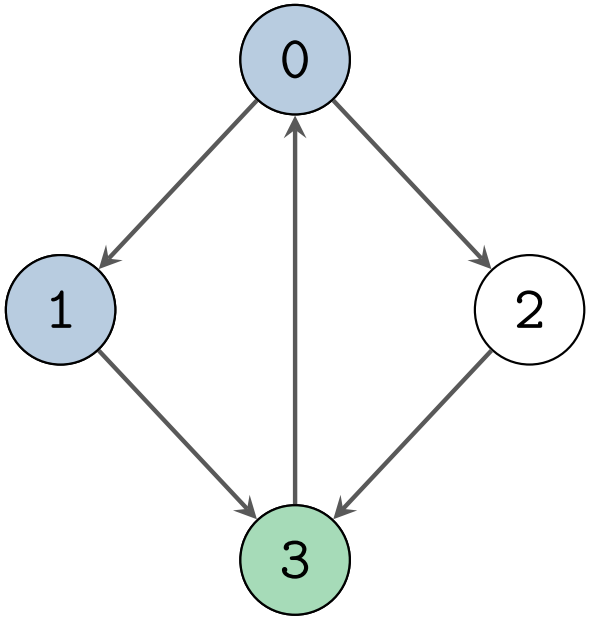


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3

De 3 a 0

El nodo 0 sigue en camino: de 0 se bajó hasta 3, y esta arista vuelve a 0. Hay un ciclo.

Una arista más: de 3 a 0

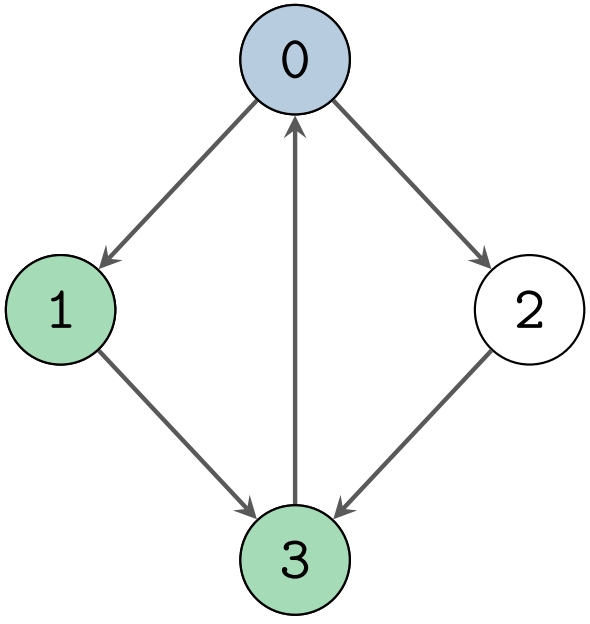


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1

Termina 3

El ciclo queda anotado y el recorrido sigue hasta el final.

Una arista más: de 3 a 0

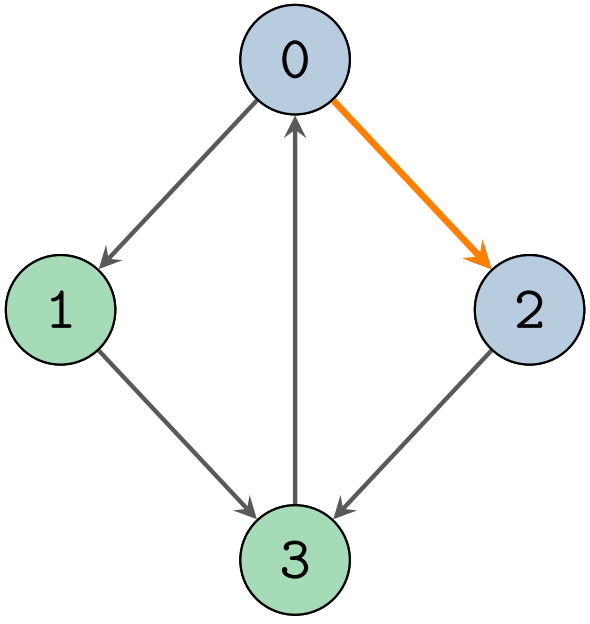


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1
termina 1		0

Termina 1

No le quedan aristas: termina y sale del camino.

Una arista más: de 3 a 0

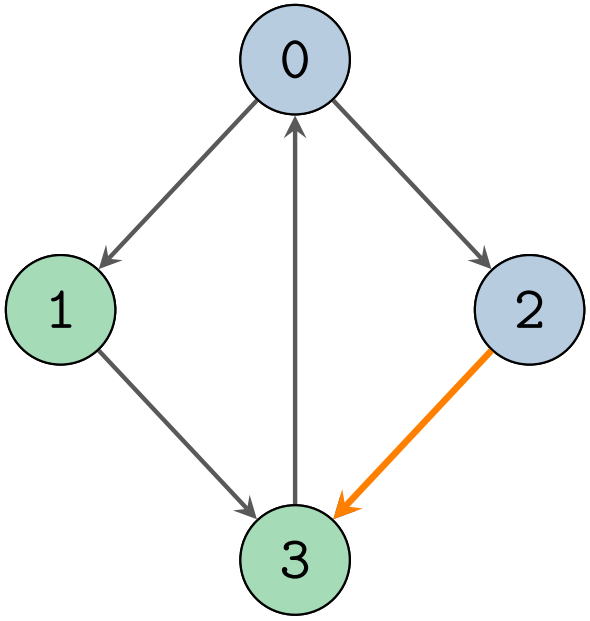


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2

De 0 a 2

El nodo 2 está sin visitar: se baja.

Una arista más: de 3 a 0

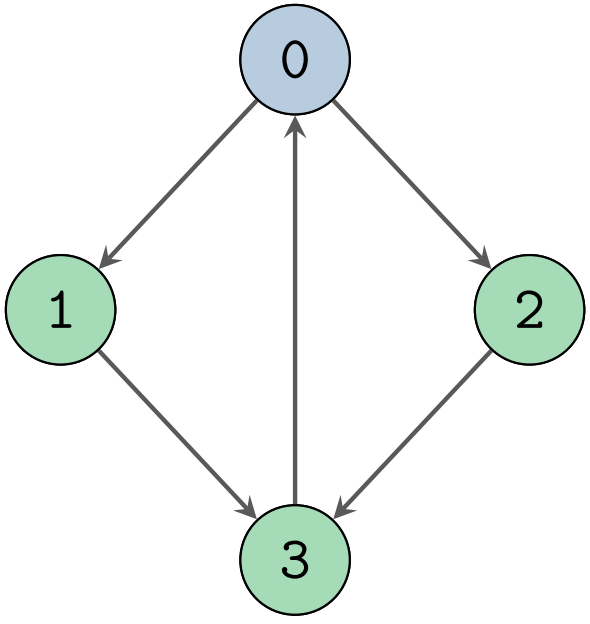


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2

De 2 a 3

El nodo 3 está terminado, y esta arista no cierra nada aunque el grafo tenga un ciclo.

Una arista más: de 3 a 0

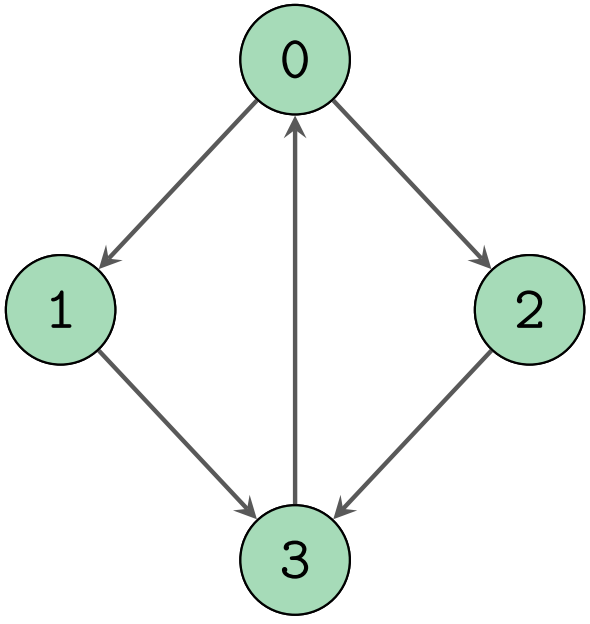


Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2
termina 2		0

Termina 2

No le quedan aristas: termina y sale del camino.

Una arista más: de 3 a 0

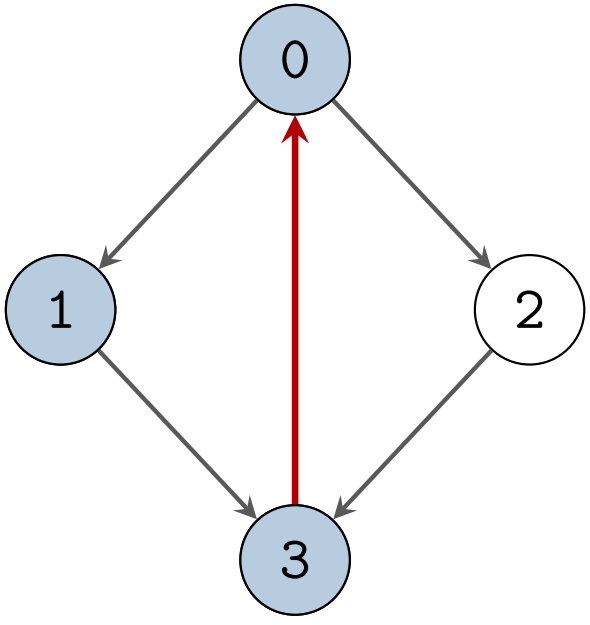


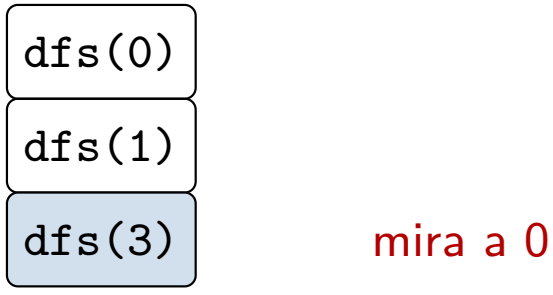
Paso	Destino	Camino
entra 0		0
$0 \to 1$	sin visitar	0 1
$1 \to 3$	sin visitar	0 1 3
$3 \to 0$	en camino	0 1 3
termina 3		0 1
termina 1		0
$0 \to 2$	sin visitar	0 2
$2 \to 3$	terminado	0 2
termina 2		0
termina 0		

Termina 0

El camino queda vacío y la respuesta es la que se anotó en $3 \to 0$: hay ciclo.

Qué es un ciclo





Una arista hacia el camino actual

Un ciclo es una arista a un nodo que sigue en el camino actual. Con el camino 0 1 3, la arista $3 \to 0$ cierra la vuelta: de 0 se llega a 3, y de 3 a 0.

La función con colores, en Python

```
def dfs(adj, color, orden, u):
    # Recorre lo que sale de u y devuelve si encontró un ciclo.
    hayCiclo = False
    color[u] = EN_CAMINO
    for v in adj[u]:
        if color[v] == EN_CAMINO:
            hayCiclo = True
        elif color[v] == SIN_VISITAR:
            if dfs(adj, color, orden, v):
                hayCiclo = True
    color[u] = TERMINADO
    orden.append(u)
    return hayCiclo
```

La función con colores, en C++

```cpp
void dfs(const vector<vector<int> > &adj, vector<int> &color,
         vector<int> &orden, bool &hayCiclo, int u) {
  // Recorre lo que sale de u; una arista a un nodo en camino cierra un ciclo.
  color[u] = EN_CAMINO;
  for (size_t i = 0; i < adj[u].size(); i++) {
    int v = adj[u][i];
    if (color[v] == EN_CAMINO) {
      hayCiclo = true;
    } else if (color[v] == SIN_VISITAR) {
      dfs(adj, color, orden, hayCiclo, v);
    }
  }
  color[u] = TERMINADO;
  orden.push_back(u);
}
```

UVa 11686, Pick up sticks

El problema

Hay n palitos numerados de 1 a n y m pares `a b`, cada uno con el significado de que a está encima de b en algún punto. Un palito se recoge solo cuando ya se recogieron todos los que tiene encima. Se pide un orden de recogida, si existe.

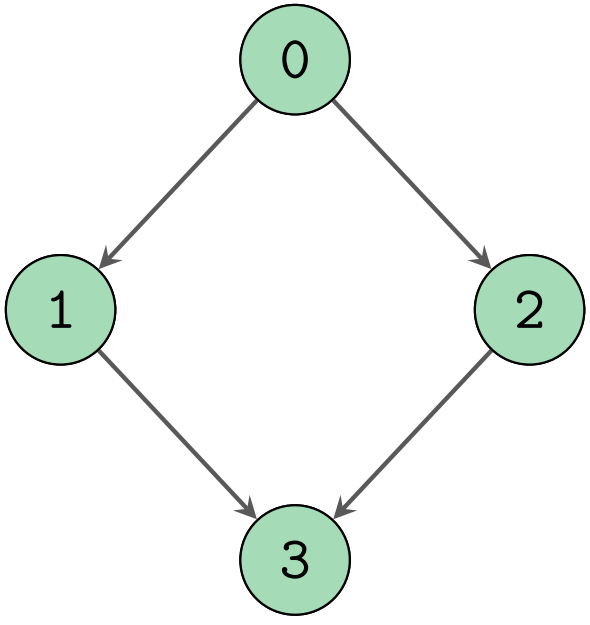
Enunciado y envío: `https://vjudge.net/problem/UVA-11686`
La página pide cuenta para ver el enunciado.

La entrada y la salida

Varios casos

Cada caso trae `n m`, con $1 \leq n, m \leq 1\,000\,000$, y después los m pares. La entrada termina en `0 0`. Por caso se imprime un palito por línea, o `IMPOSSIBLE` si no existe ningún orden. El enunciado acepta cualquier orden correcto.

De las llamadas al orden

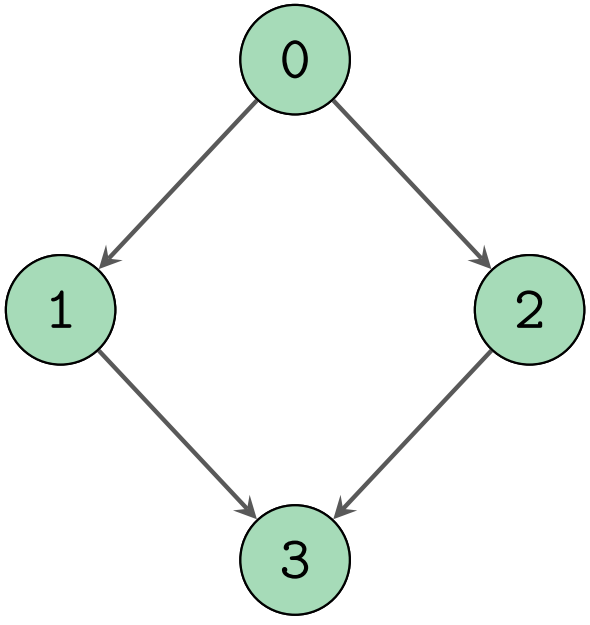


Terminaron	3 1 2 0

Pregunta

En el diamante, las llamadas terminaron en el orden 3, 1, 2, 0. ¿Cómo sale de ahí un orden en que cada arista vaya de izquierda a derecha?

De las llamadas al orden



Terminaron	3 1 2 0
Al revés	0 2 1 3

Respuesta

Al revés: 0, 2, 1, 3. Cuando un nodo termina, todo lo que sale de él ya terminó, así que en el orden invertido cada nodo queda antes de sus vecinos.

La lectura de la entrada

```
int n, m;
while (scanf("%d %d", &n, &m) == 2 && (n != 0 || m != 0)) {
  vector<vector<int> > adj(n + 1);
  for (int i = 0; i < m; i++) {
    int a, b;
    scanf("%d %d", &a, &b);
    adj[a].push_back(b);
  }
  /* aquí va el recorrido en profundidad */
```

Dos detalles

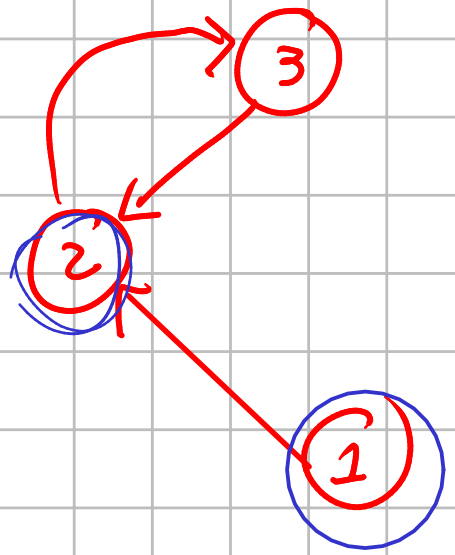
Las aristas se leen todas antes de recorrer, aunque ya se vea un ciclo: las que queden sin leer se tomarían como el caso siguiente. Los palitos se numeran desde 1, y por eso `adj` tiene $n + 1$ listas.

Sample Input

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

Sample Output

```
1
2
3
```





La profundidad de la pila

Lo que se midió

En una cadena de palitos, cada uno es una llamada abierta encima de la anterior. En C++ con `-O2` y una pila de 8 MB, la recursiva pasó con 86 851 niveles y reventó con 87 826. El enunciado admite un millón.

Cómo atacarlo

El parámetro y la trampa

El parámetro es $n + m$, y la profundidad de la recursión puede llegar a n. La propiedad es la del camino actual: hay ciclo exactamente cuando una arista llega a un nodo en camino. La trampa está en los nodos terminados, que con solo dos estados parecen ciclos, y en los palitos sueltos: se llama desde todo nodo sin visitar, no solo desde el 1.

A resolver en el computador

UVa 11686

1. Lea el caso y arme la lista de adyacencia.
2. Llame a `dfs` desde todo nodo sin visitar.
3. Si apareció un ciclo, imprima `IMPOSSIBLE`.
4. Si no, imprima los nodos en orden de terminación invertido.

La muestra del enunciado, 3 2, 1 2, 2 3, da 1, 2, 3.

Contenido

1 Bisección: encontrar la función

2 Recorrido en amplitud con tope de pasos

3 Recorrido en profundidad y ciclos

4 Problemas para practicar

Para seguir

Bisección con otro disfraz

UVa 714, repartir libros entre escribas para que el que más copie copie lo menos posible. La pregunta es *¿alcanza con t?*

Recursión y estructuras

UVa 839, un móvil de barras colgadas de barras, donde el árbol de llamadas es el móvil. **UVa 10935** y **UVa 12100**, dos colas. **UVa 156**, agrupar palabras por sus letras ordenadas.