

DETECCIÓN DE CICLOS

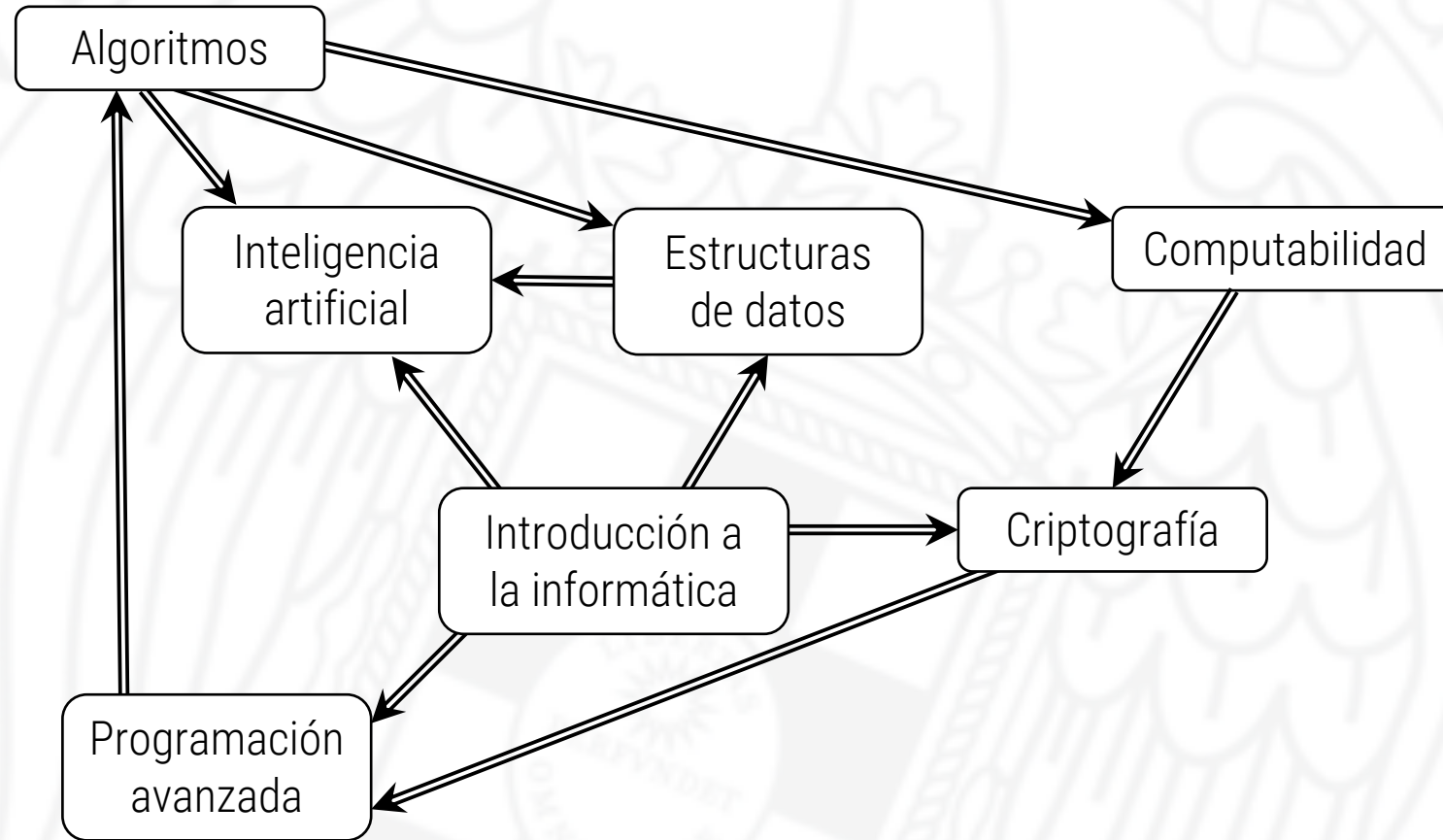


U N I V E R S I D A D
COMPLUTENSE
M A D R I D

ALBERTO VERDEJO

Ciclos dirigidos

- En ocasiones representan problemas.



Ciclos dirigidos

- En ocasiones representan problemas.

```
public class A extends B
{
    ...
}
```

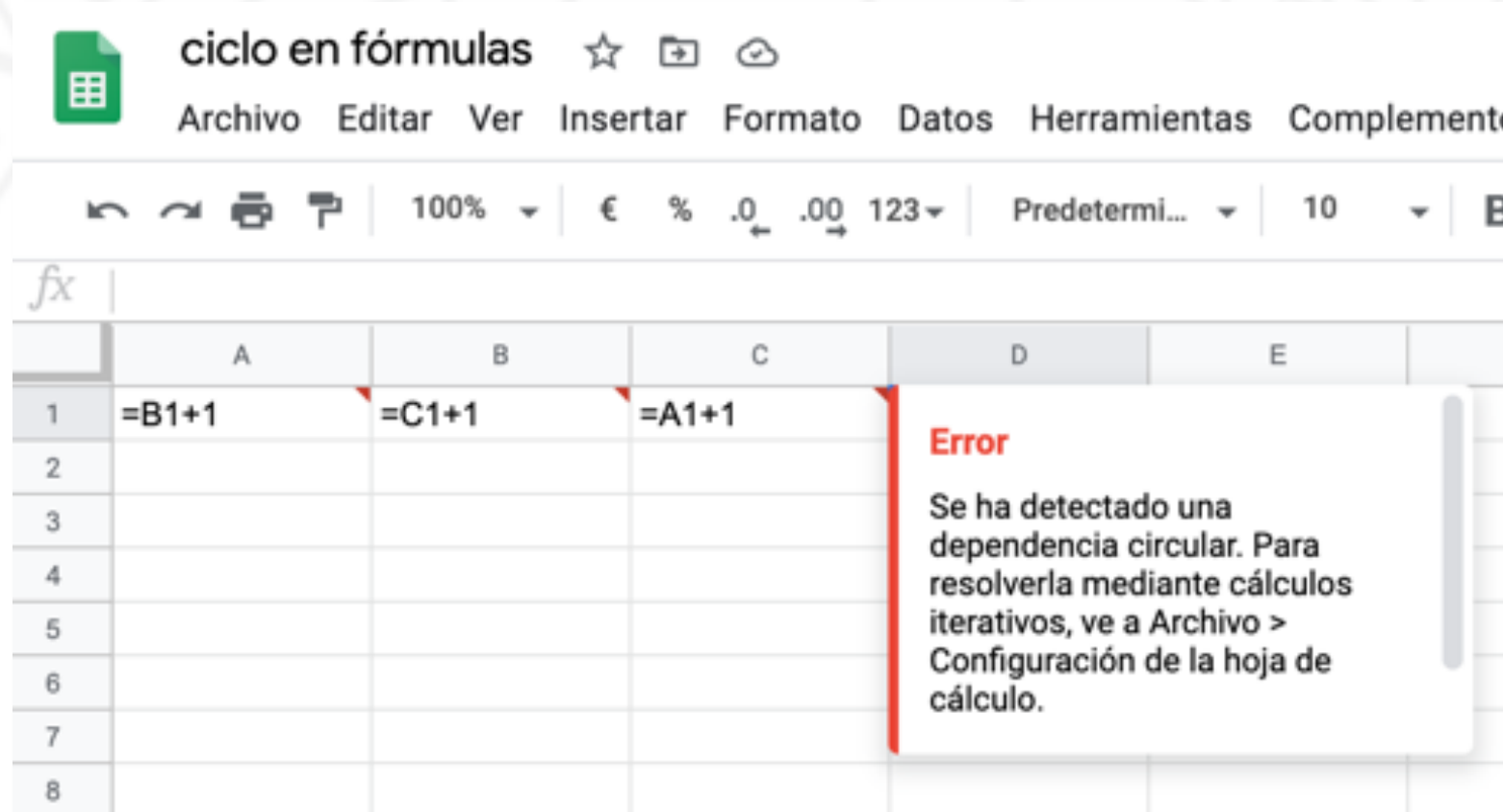
```
public class B extends C
{
    ...
}
```

```
public class C extends A
{
    ...
}
```

```
% javac A.java
A.java:1: cyclic inheritance
involving A
public class A extends B { }
                ^
1 error
```

Ciclos dirigidos

- En ocasiones representan problemas.



The screenshot shows a Google Sheets interface with a spreadsheet titled "ciclo en fórmulas". The spreadsheet has columns A, B, C, D, and E, and rows 1 through 8. The formulas are as follows:

	A	B	C	D	E
1	=B1+1	=C1+1	=A1+1		
2					
3					
4					
5					
6					
7					
8					

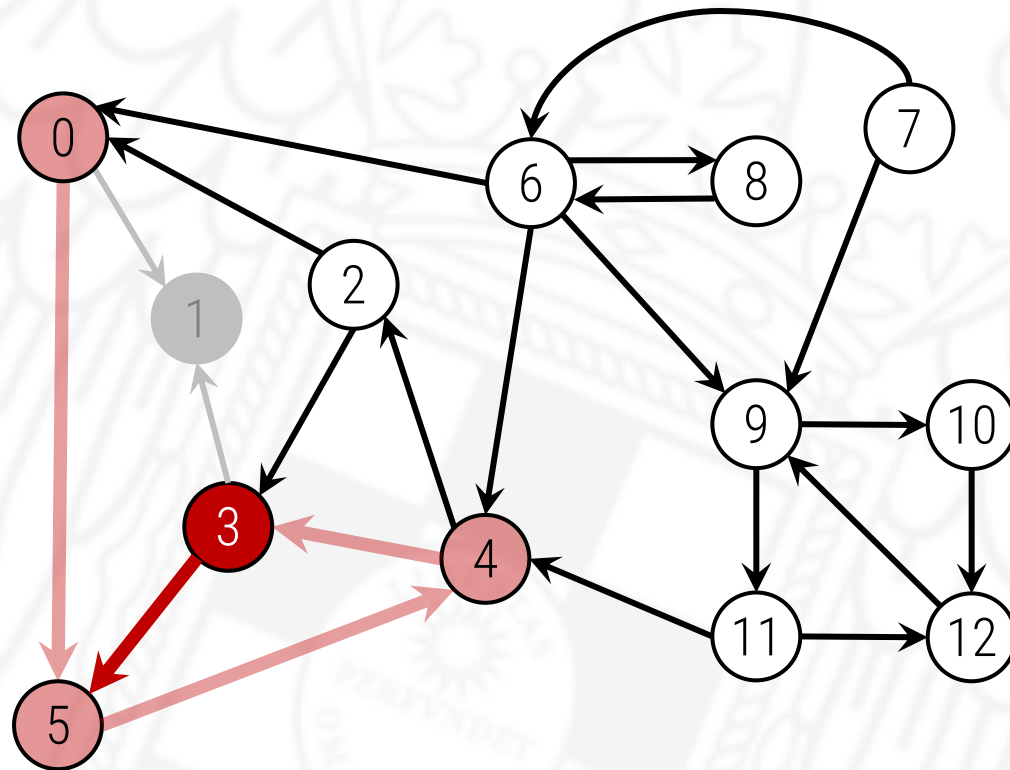
An error message box is displayed on the right side of the spreadsheet, indicating a circular dependency. The message is in Spanish and suggests using iterative calculations to resolve the issue.

Error

Se ha detectado una dependencia circular. Para resolverla mediante cálculos iterativos, ve a Archivo > Configuración de la hoja de cálculo.

Detección de ciclos dirigidos

- Utilizamos un recorrido en profundidad. La pila de la recursión contiene el camino *actual*.



Implementación

```
class CicloDirigido {
public:
    CicloDirigido(Digrafo const& g) : visit(g.V(), false), ant(g.V()),
                                     apilado(g.V(), false), hayciclo(false) {
        for (int v = 0; v < g.V(); ++v)
            if (!visit[v])
                dfs(g, v);
    }

    bool hayCiclo() const { return hayciclo; }

    Camino const& ciclo() const { return _ciclo; }
}
```


Implementación

private:

```
std::vector<bool> visit;    // visit[v] = ¿se ha alcanzado a v en el dfs?
std::vector<int> ant;       // ant[v] = vértice anterior en el camino a v
std::vector<bool> apilado;  // apilado[v] = ¿está el vértice v en la pila?
Camino _ciclo;             // ciclo dirigido (vacío si no existe)
bool hayciclo;
```

Implementación

```
void dfs(Digrafo const& g, int v) {  
    apilado[v] = true;  
    visit[v] = true;  
    for (int w : g.ady(v)) {  
        if (hayciclo) // si hemos encontrado un ciclo terminamos  
            return;  
        if (!visit[w]) { // encontrado un nuevo vértice, seguimos  
            ant[w] = v; dfs(g, w);  
        } else if (apilado[w]) { // hemos detectado un ciclo  
            // se recupera retrocediendo  
            hayciclo = true;  
            for (int x = v; x != w; x = ant[x])  
                _ciclo.push_front(x);  
            _ciclo.push_front(w); _ciclo.push_front(v);  
        }  
    }  
    apilado[v] = false;  
}
```

