CAMINOS MÍNIMOS ENTRE TODO PAR DE VÉRTICES

ALBERTO VERDEJO

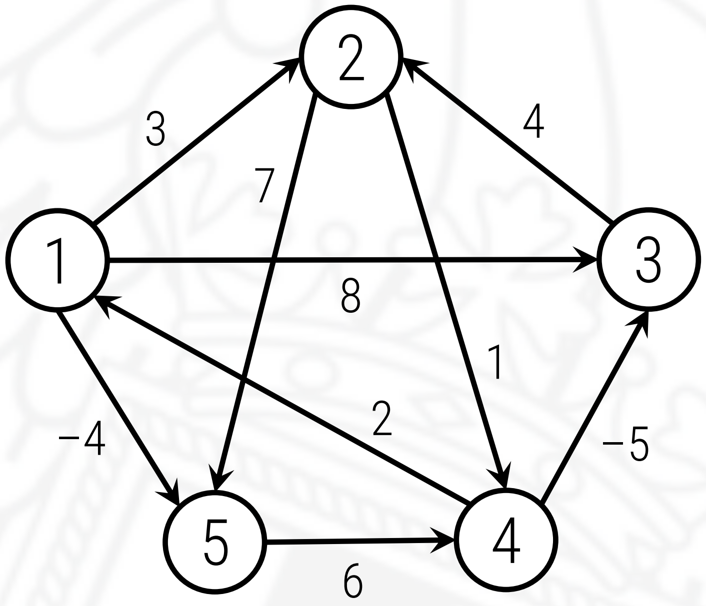
UNIVERSIDAD COMPLUTENSE MADRID

Caminos mínimos entre todo par de vértices

- Dado un digrafo valorado, calcular el camino de coste mínimo entre cada par de vértices.
- Si los pesos son positivos y el grafo es disperso, utilizando el algoritmo de Dijkstra V veces, obtenemos un algoritmo con coste en $O(V\,A \log V)$.
- El algoritmo de Floyd resuelve el caso general, con pesos posiblemente negativos, con coste en $O(V^3)$.

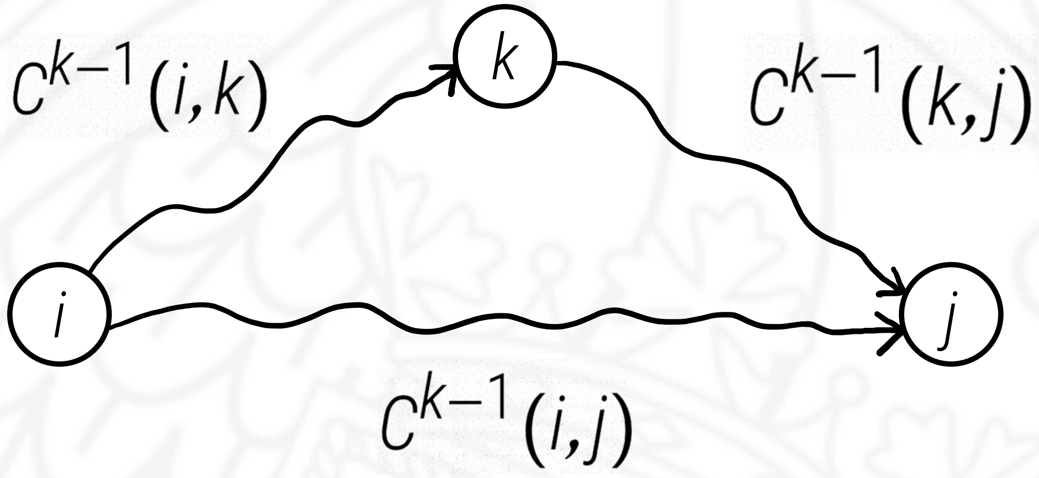
$$G[i][j] = \begin{cases} 0 & \text{si } i = j \\ \mathit{coste} & \text{si hay arista de } i \text{ a } j \\ +\infty & \text{si no hay arista de } i \text{ a } j \end{cases}$$

Caminos mínimos entre todo par de vértices



$C^k(i,j) \quad = \quad$ coste *mínimo* para ir de *i* a *j* pudiendo utilizar como vértices intermedios aquellos entre 1 y *k*

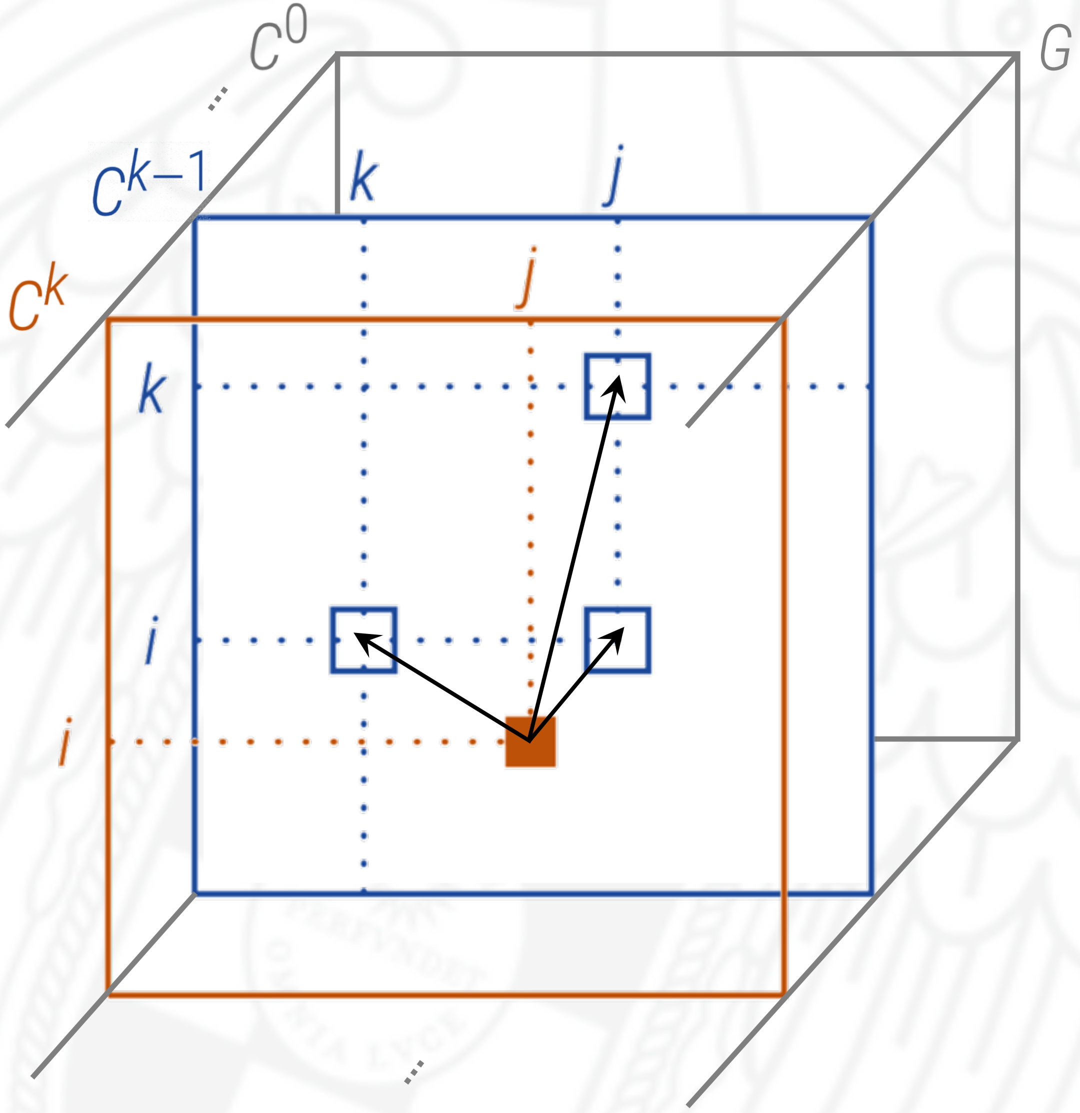
Recurrencia de Floyd



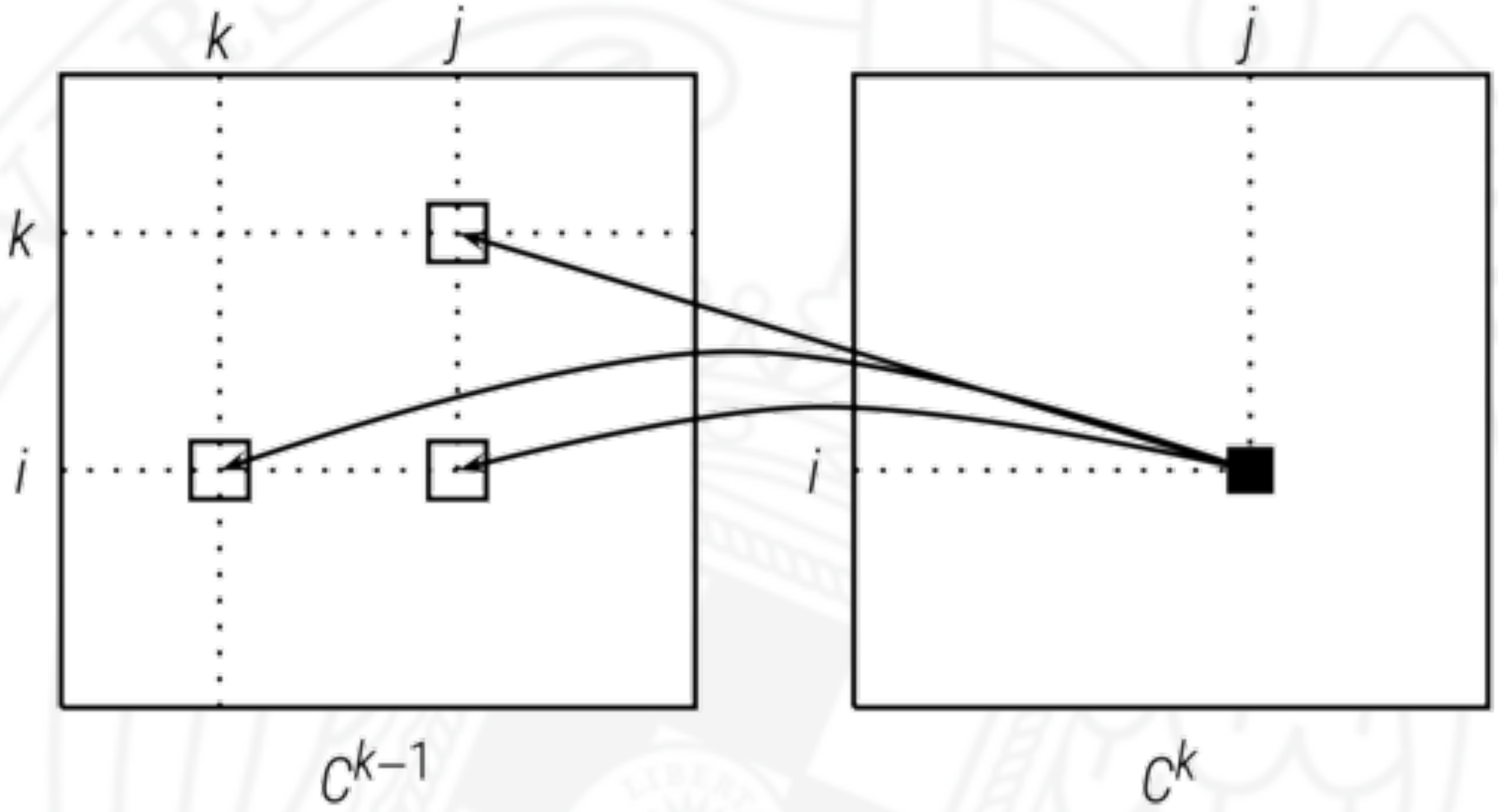
$$C^k(i,j) = \text{mín}(\, C^{k-1}(i,j),\, C^{k-1}(i,k) + C^{k-1}(k,j)\,)$$

$$C^0 = G$$

Tabla



Tabla



$$C^k(k,j) = \text{mín}(\,C^{k-1}(k,j),\, C^{k-1}(k,k) + C^{k-1}(k,j)\,) = C^{k-1}(k,j)$$

Reconstrucción de los caminos mínimos

$A^k(i,j) \quad = \quad$ vértice anterior a j en el camino mínimo de i a j pudiendo utilizar como vértices intermedios aquellos entre 1 y k

$$A^0(i,j) = \begin{cases} -1 & \text{si } i = j \vee G[i][j] = +\infty \\ i & \text{si } i \neq j \wedge G[i][j] < +\infty \end{cases}$$

$$A^k(i,j) = \begin{cases} A^{k-1}(i,j) & \text{si } C^{k-1}(i,j) \leq C^{k-1}(i,k) + C^{k-1}(k,j) \\ A^{k-1}(k,j) & \text{si } C^{k-1}(i,j) > C^{k-1}(i,k) + C^{k-1}(k,j) \end{cases}$$

Implementación

```
void Floyd(Matriz<EntInf> const& G, Matriz<EntInf> & C, Matriz<int> & A) {
   int V = G.numfils(); // número de vértices de G
   // inicialización
   C = G;
   A = Matriz<int>(V, V, -1);
   for (int i = 0; i < V; ++i) {
      for (int j = 0; j < V; ++j) {
         if (i != j && G[i][j] != Infinito)
            A[i][j] = i;
      }
   }
```

Implementación

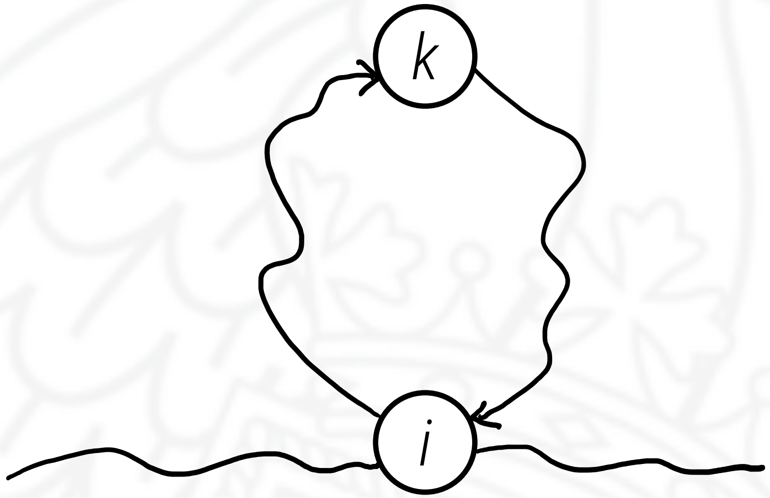
```
    // actualizaciones de las matrices
    for (int k = 0; k < V; ++k) {
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                auto temp = C[i][k] + C[k][j];
                if (temp < C[i][j]) { // es mejor pasar por k
                    C[i][j] = temp;
                    A[i][j] = A[k][j];
                }
            }
        }
    }
}
```

Implementación

```cpp
using Camino = std::deque<int>;

Camino ir_de(int i, int j, Matriz<int> const& A) {
    Camino cam;
    while (j != i) {
        cam.push_front(j);
        j = A[i][j];
    }
    cam.push_front(i);
    return cam;
}
```

Detección de ciclos de coste negativo



$$c^k(i,i) = \min(\, c^{k-1}(i,i),\, c^{k-1}(i,k) + c^{k-1}(k,i)\,)$$

$$c^k(i,i) < 0$$

Implementación

```
bool Floyd(Matriz<EntInf> const& G, Matriz<EntInf> & C, Matriz<int> & A) {
    ...
    for (int k = 0; k < V; ++k) {
        for (int i = 0; i < V; ++i) {
            for (int j = 0; j < V; ++j) {
                ...
            }
            if (C[i][i] < 0) return false;
        }
    }
    return true;
}
```