Diseño de Algoritmos

Sesión 12

Profesores:

Tomás Lara Valdovinos – sir.thomas.lara@gmail.com

Jessica Meza-Jaque – jessicamezajaque.uchile@gmail.com

OBJETIVOS DE LA SESIÓN

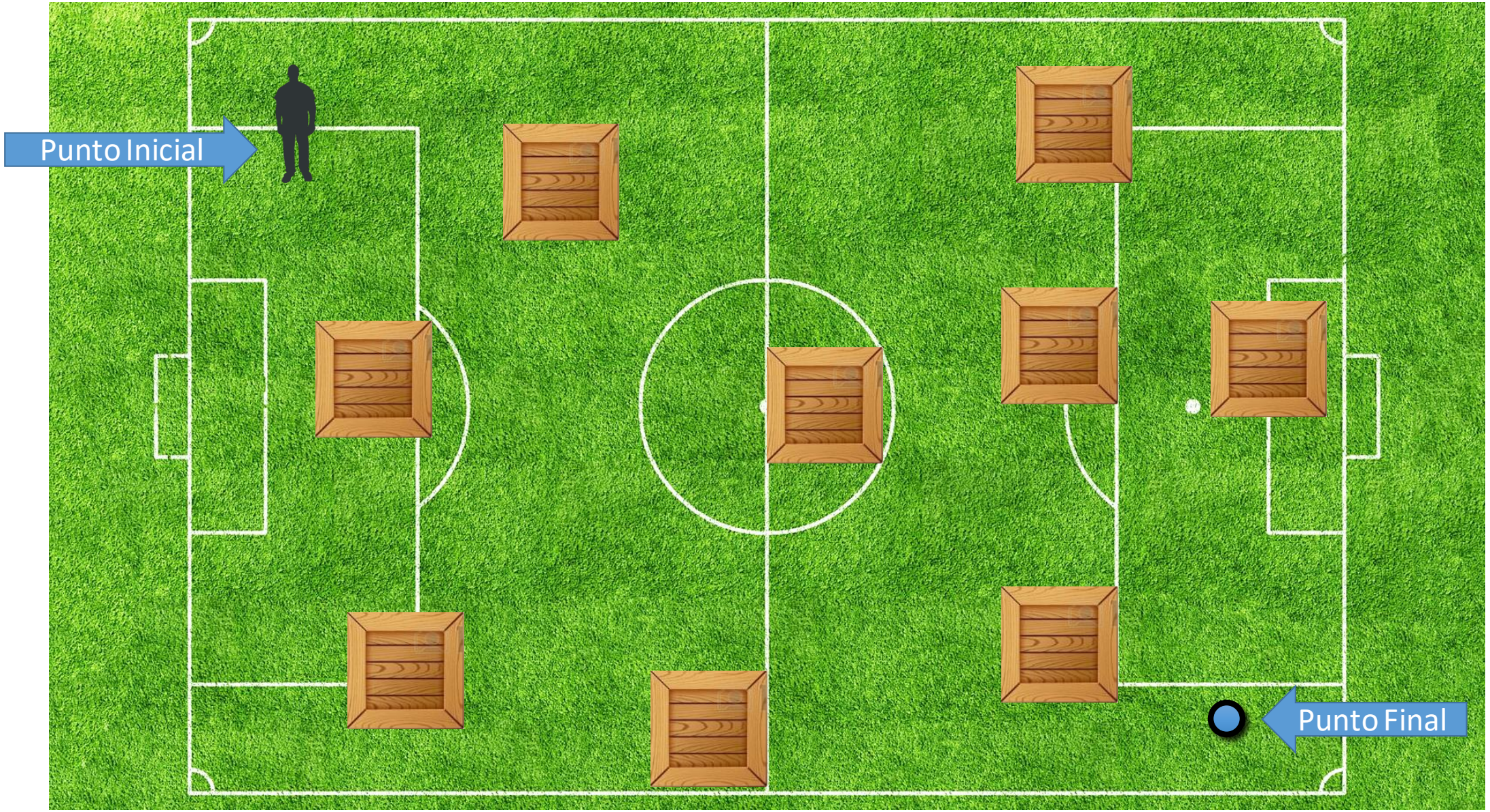
- Analizar el Problema de los caminos mínimos entre todos los pares de vértices en un grafo.
- Analizar el Algoritmo de Floyd-Warshall

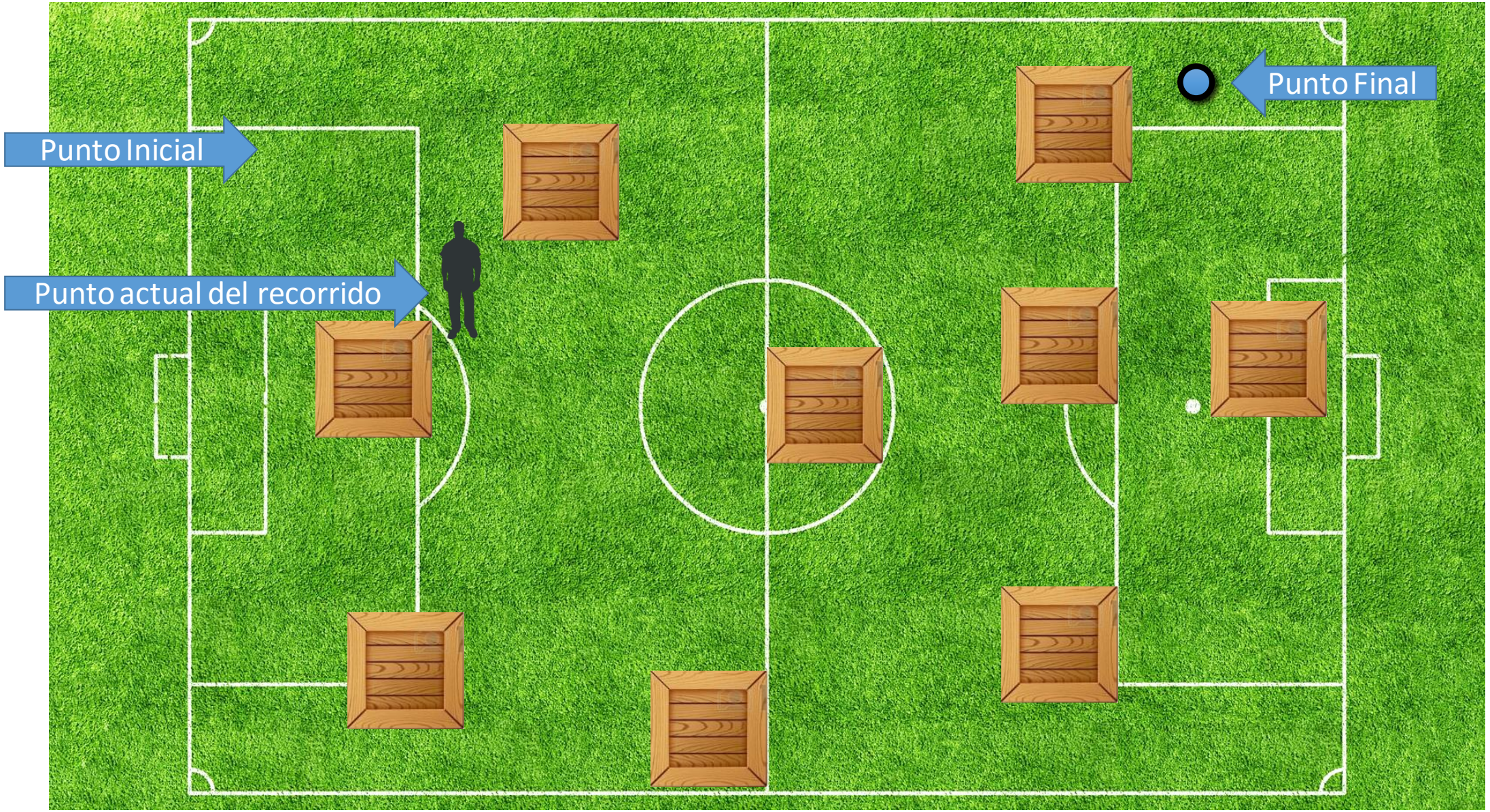
CONTENIDOS DE LA SESIÓN

- Algoritmo de Floyd-Warshall
- Ejemplo

Problema de los caminos mínimos

- Siguiendo con el ejemplo visto en la sesión anterior
- ¿Qué pasaría si el punto final cambiara de posición a medida que pasa el tiempo?, o sea
- ¿Qué pasaría si el punto final es un objetivo móvil?


Punto Inicial
Punto Final


Punto Final
Punto Inicial
Punto actual del recorrido

Problema de los caminos mínimos

- Esto genera un **nuevo problema** de …
- ***definición de ruta más corta***

¿Podemos resolver esto con el algoritmo de Dijsktra aún?

Respuesta a la interrogante

- Sí, podemos seguir utilizando Dijsktra para recalcular la ruta desde el **punto actual** en que se encuentra la persona hasta el **nuevo punto final**.

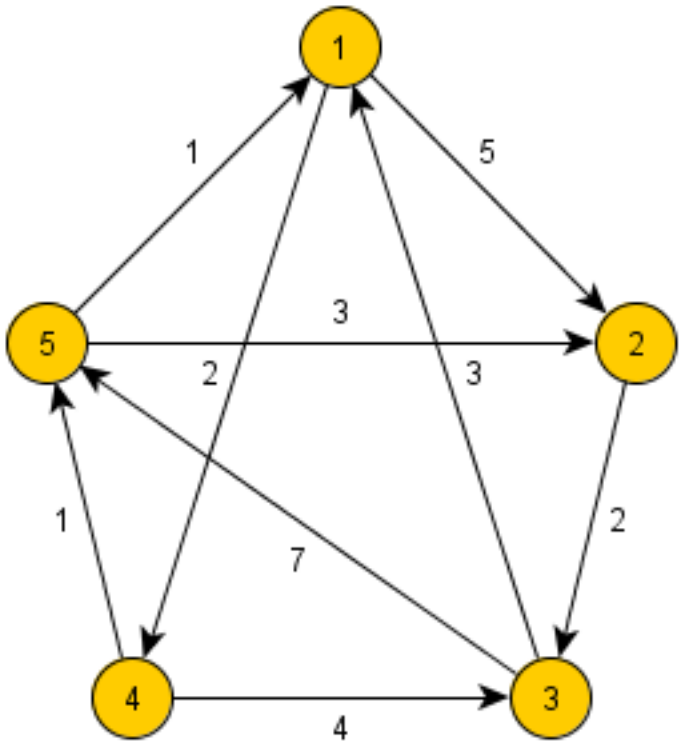
¿Es esta manera de resolverlo **EFICIENTE**?

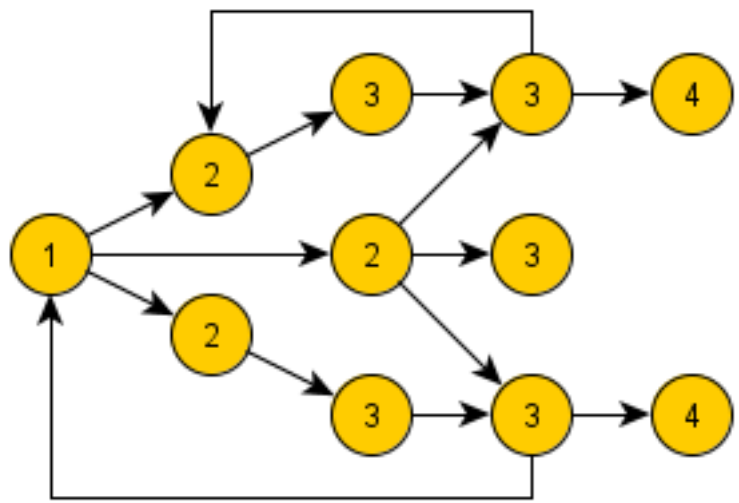
Analizando la situación, tenemos

- Es necesaria una gran cantidad de **cómputo**.
- Podría ser necesario calcular una nueva ruta **poco tiempo después** de ya haber calculado una anterior.

- **¿Existe un algoritmo que resuelva de manera más eficiente este problema?**

Algoritmo de Floyd-Warshall





Algoritmo de Floyd-Warshall

- Utiliza la **programación dinámica**
- Encuentra la ruta más corta entre todos los pares de vértices en el grafo.

Algoritmo en Pseudocódigo

```
allPairsShortestPath (G)
  foreach u in V do
    dist[u][v] = ∞
    pred[u][v] = NULL

  dist[u][u] = 0

  foreach neighbour v of u do
    dist[u][v] = weigth(u,v)
    pred[u][v] = u
```

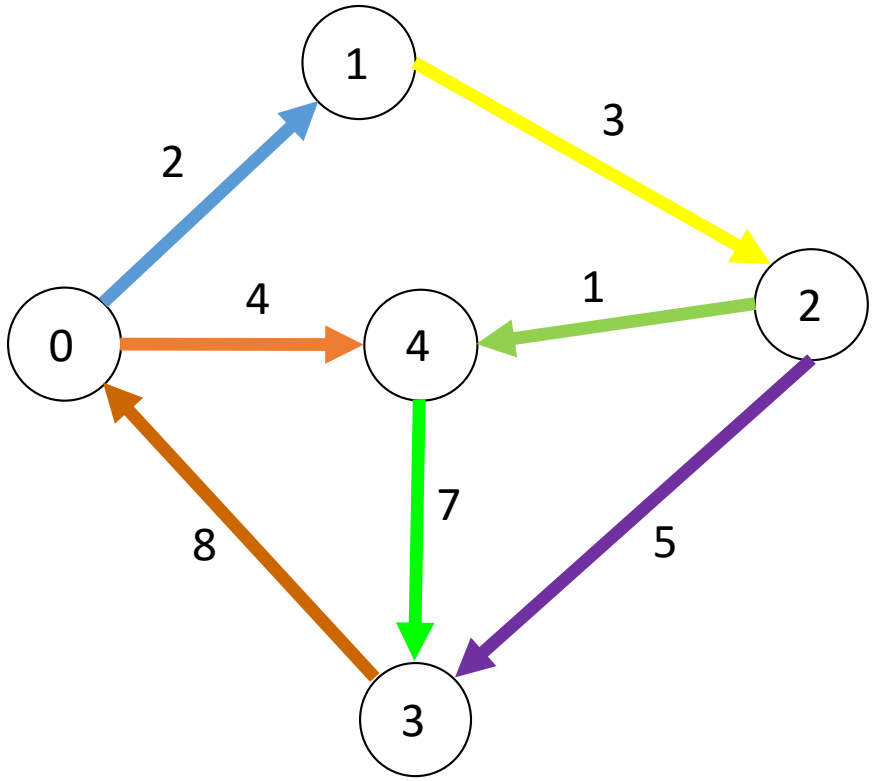
```
foreach t in V do
  foreach u in V
    foreach v in V

      newLen = dist[u][t] + dist[t][v]

      if (newLen < dist[u][v]) then
        dist[u][v] = newLen
        pred[u][v] = pred[t][v]

end
```

Explicación del algoritmo

- Floyd-Warshall utiliza 2 matrices:
  - dist[][]: Indica la distancia total del recorrido que existe entre 2 vértices Vi a Vj
  - pred[][]: indica el nodo por el cual debe pasarse para llegar de Vi a Vj

- Da la primicia de que los nodos directamente conectados son la ruta más corta para llegar entre ellos.

```
1  allPairsShortestPath (G)
2    foreach u in V do
3      dist[u][v] = ∞
4      pred[u][v] = NULL
5
6    dist[u][u] = 0
7
8    foreach neighbour v of u do
9      dist[u][v] = weigth(u,v)
10     pred[u][v] = u
```



dist[u][v]

	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	∞	∞	0	∞
4	∞	∞	∞	7	0

pred[u][v]

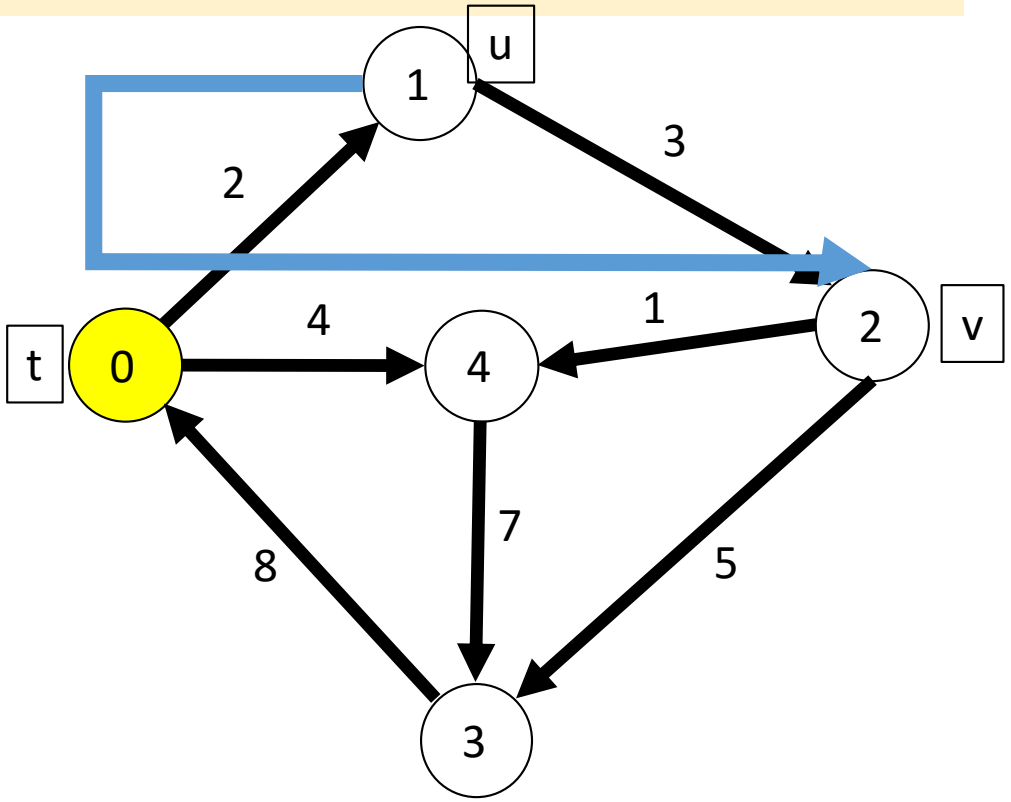
	0	1	2	3	4
0		0			0
1			1		
2				2	2
3	3				
4				4	

Explicación del algoritmo

- En la segunda parte del algoritmo Floyd-Warshall va mejorando las rutas evaluando la distancia entre cada par de vértice a través de cada vértice.

- Por ejemplo: Evalúa la distancia de un vértice **v** a un vértice **u** pasando a través de un vértice **t**

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 0
u = 1
v = 2

dist[1][0] = ∞
dist[0][2] = ∞

newLen = ∞

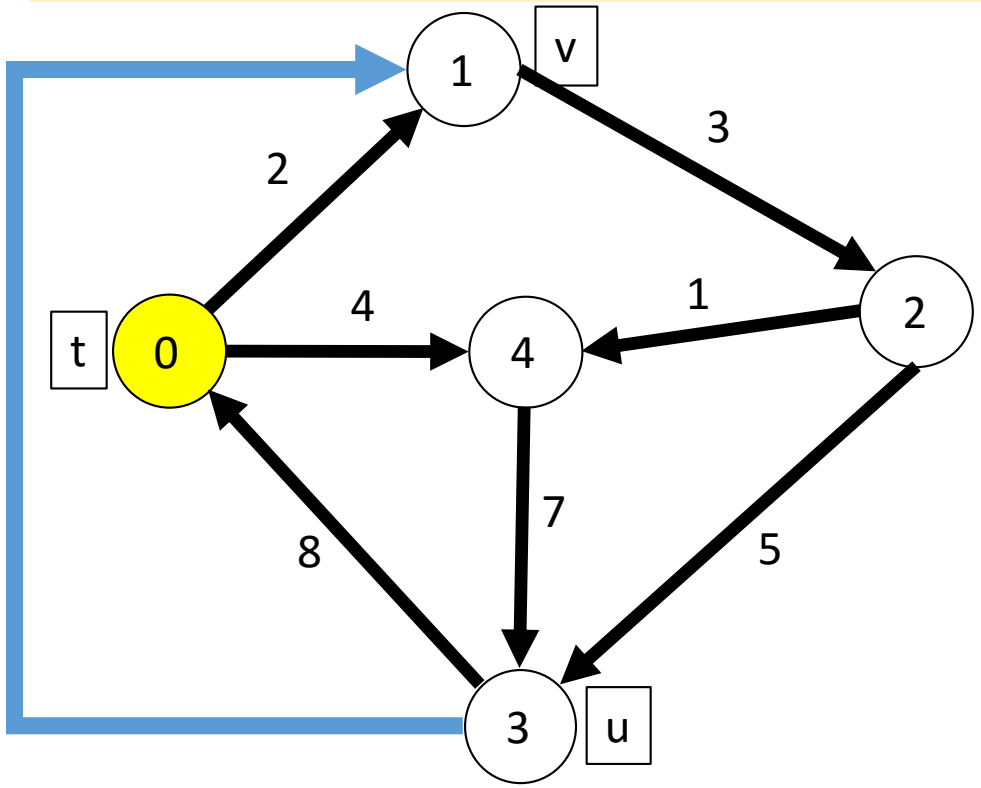
dist[u][v]

	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	∞	∞	0	∞
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0			0
1			1		
2				2	2
3	3				
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 0
u = 3
v = 1

dist[3][0] = 8
dist[0][1] = 2

newLen = 10

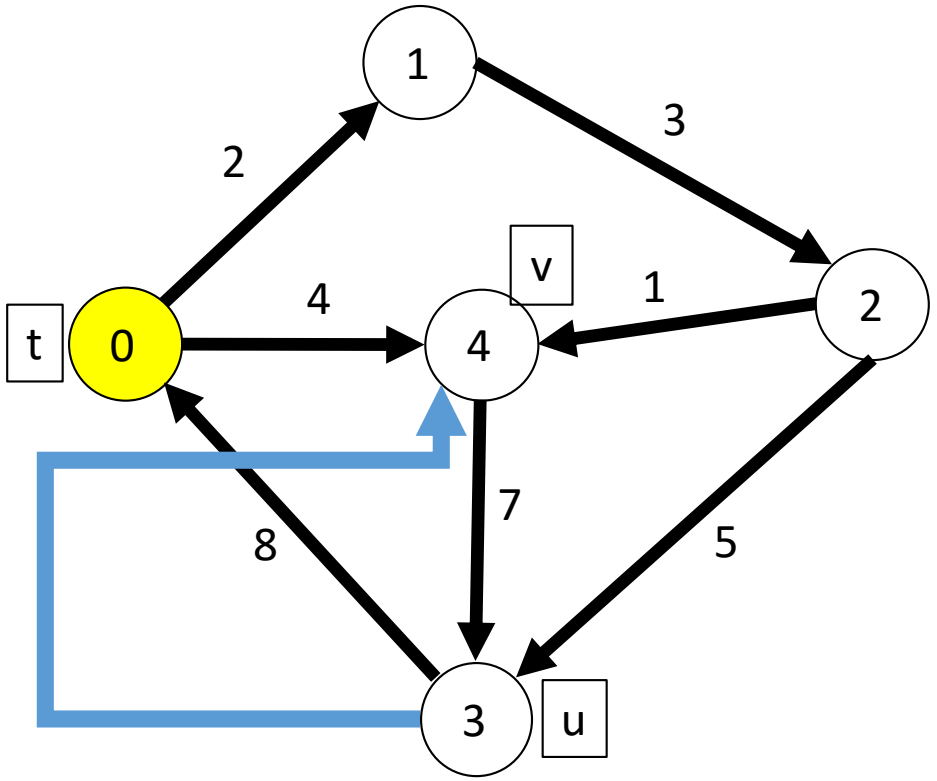
dist[u][v]

	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	∞	0	∞
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0			0
1			1		
2				2	2
3	3	0			
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 0
u = 3
v = 4

dist[3][0] = 8
dist[0][4] = 4

newLen = 12

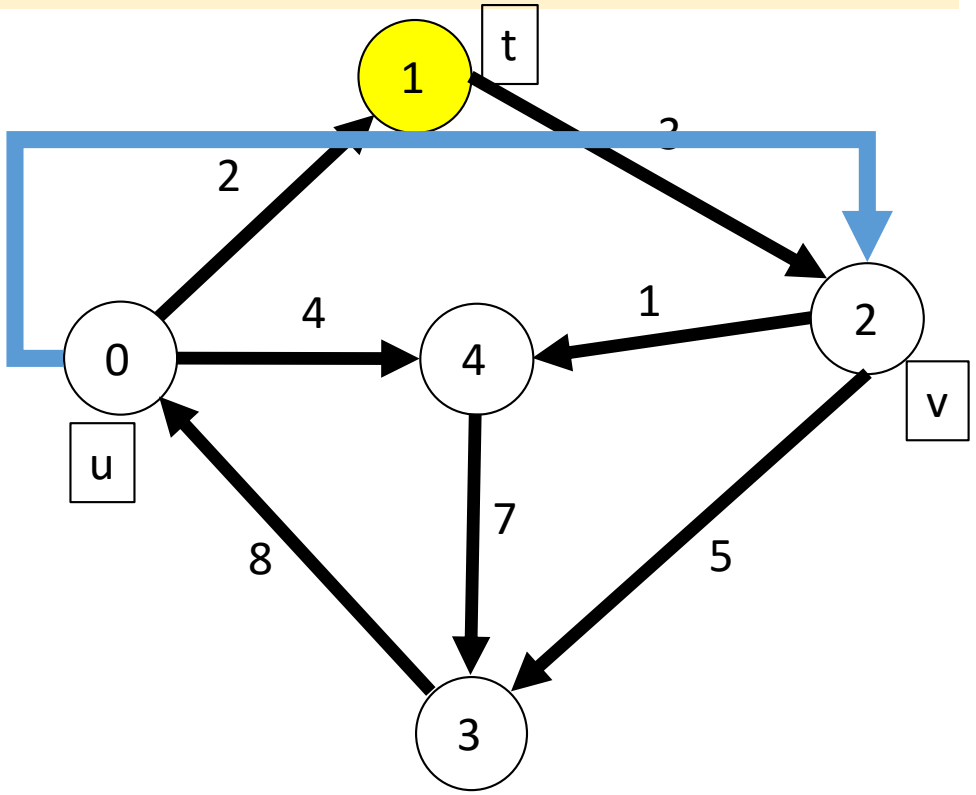
dist[u][v]

	0	1	2	3	4
0	0	2	∞	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	∞	0	12
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0			0
1			1		
2				2	2
3	3	0			0
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 1
u = 0
v = 2

dist[0][1] = 2
dist[1][2] = 3

newLen = 5

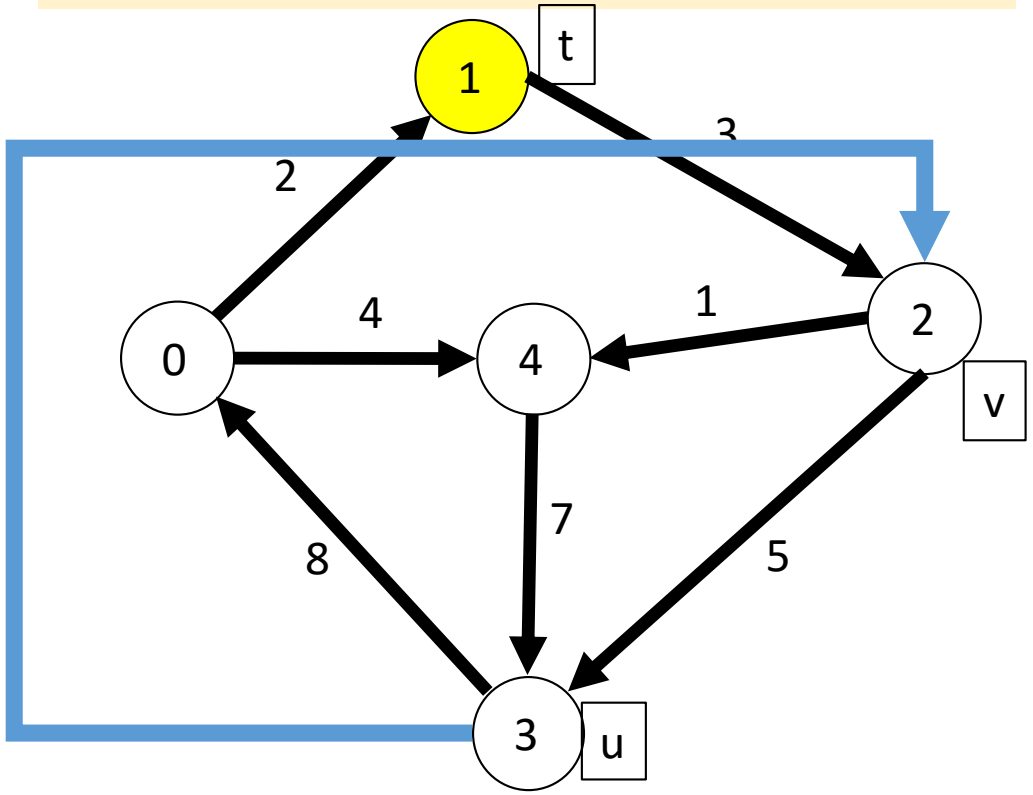
dist[u][v]

	0	1	2	3	4
0	0	2	5	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	∞	0	12
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0	1		0
1			1		
2				2	2
3	3	0			0
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 1
u = 3
v = 2

dist[3][1] = 10
dist[1][2] = 3

newLen = 13

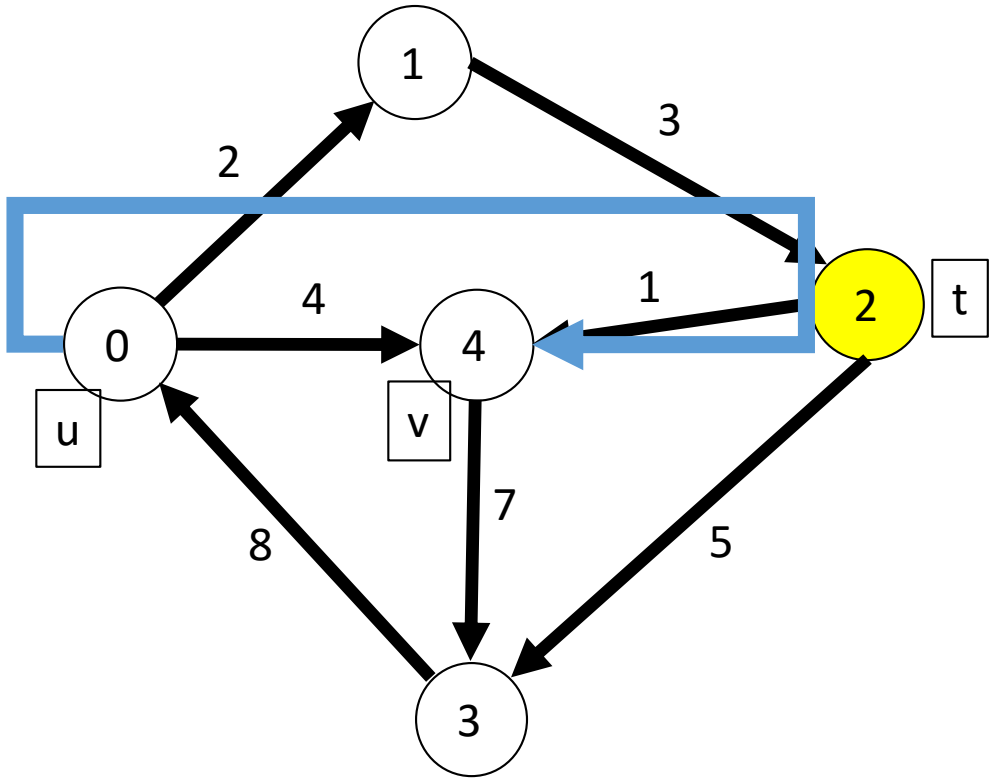
dist[u][v]

	0	1	2	3	4
0	0	2	**5**	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	**13**	0	12
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0	1		0
1			1		
2				2	2
3	3	0	1		0
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 2
u = 0
v = 4

dist[0][2] = 5
dist[2][4] = 1

newLen = 6

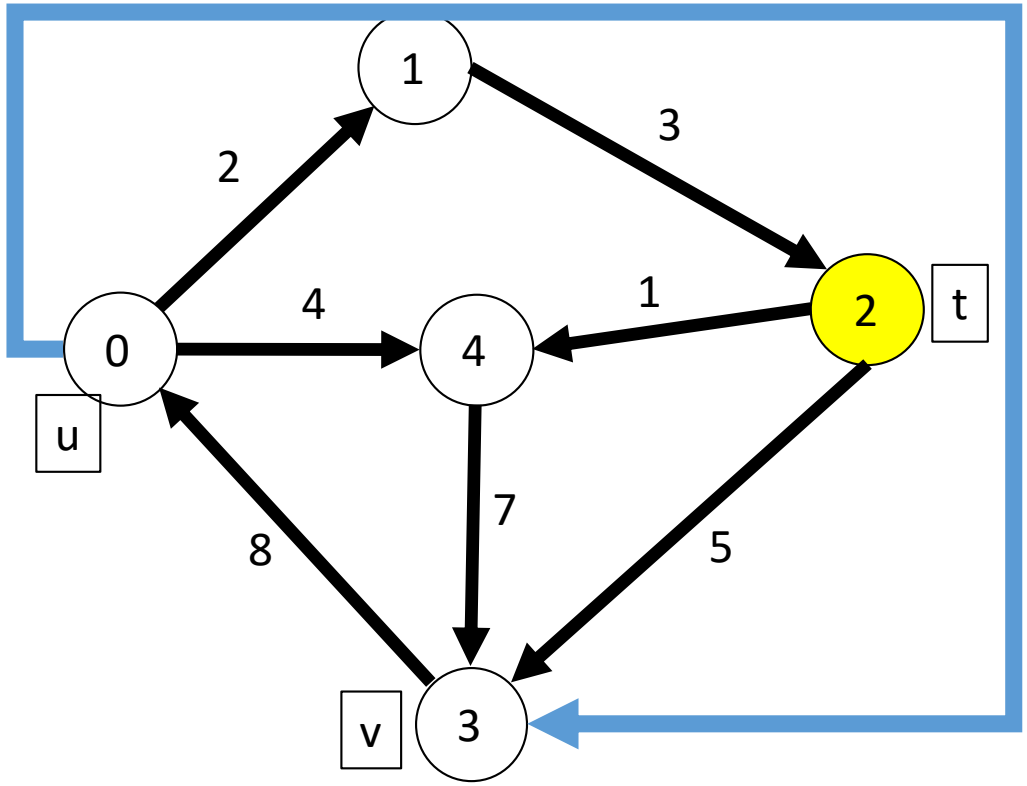
dist[u][v]

	0	1	2	3	4
0	0	2	5	∞	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	13	0	12
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0	1		0
1			1		
2				2	2
3	3	0	1		0
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



t = 2
u = 0
v = 3

dist[0][2] = 5
dist[2][3] = 5

newLen = 10

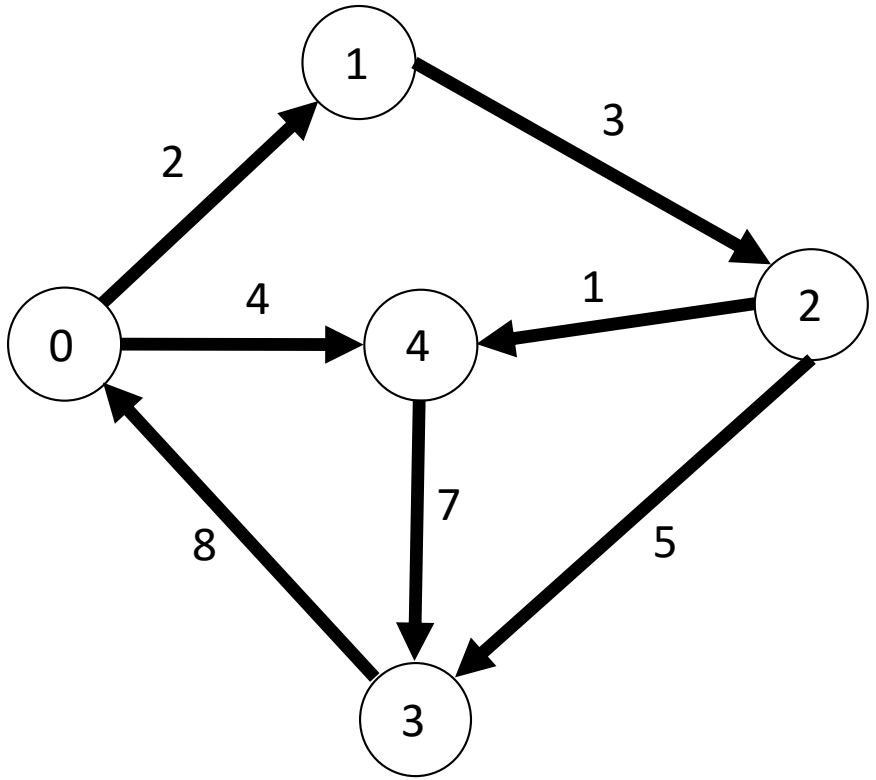
dist[u][v]

	0	1	2	3	4
0	0	2	5	**10**	4
1	∞	0	3	∞	∞
2	∞	∞	0	5	1
3	8	10	13	0	12
4	∞	∞	∞	7	0

pred[u][v]

	0	1	2	3	4
0		0	1	2	0
1			1		
2				2	2
3	3	0	1		0
4				4	

```
1  foreach t in V do
2    foreach u in V
3      foreach v in V
4
5        newLen = dist[u][t] + dist[t][v]
6
7        if (newLen < dist[u][v]) then
8          dist[u][v] = newLen
9          pred[u][v] = pred[t][v]
10 end
```



dist[u][v]

	0	1	2	3	4
0	0	2	5	10	4
1	16	0	3	8	4
2	13	15	0	5	1
3	8	10	13	0	12
4	15	17	20	7	0

pred[u][v]

	0	1	2	3	4
0		0	1	2	0
1	3		1	2	2
2	3	3		2	2
3	3	0	1		0
4	3	3	3	4	

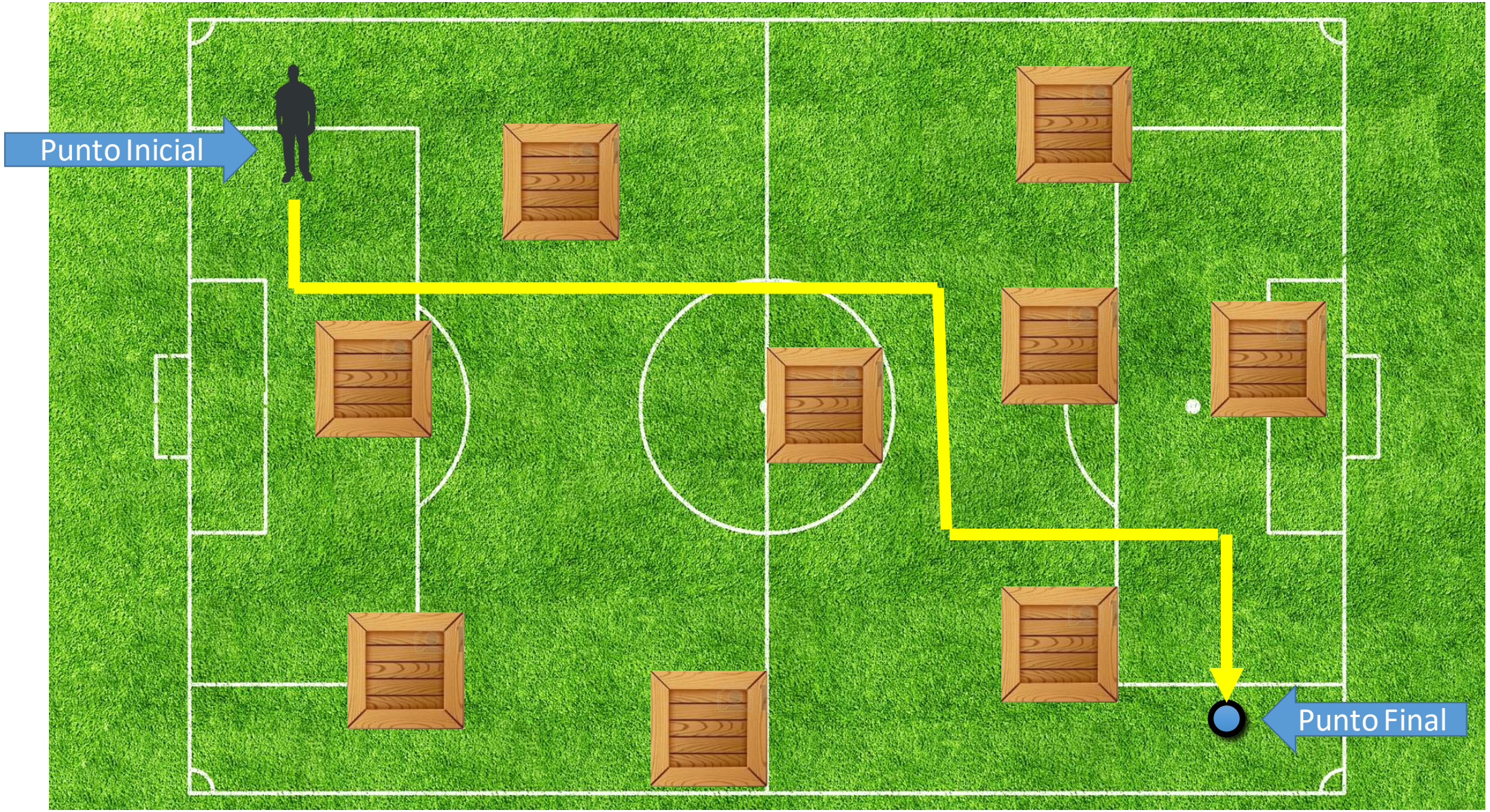
Encontrar la ruta más corta

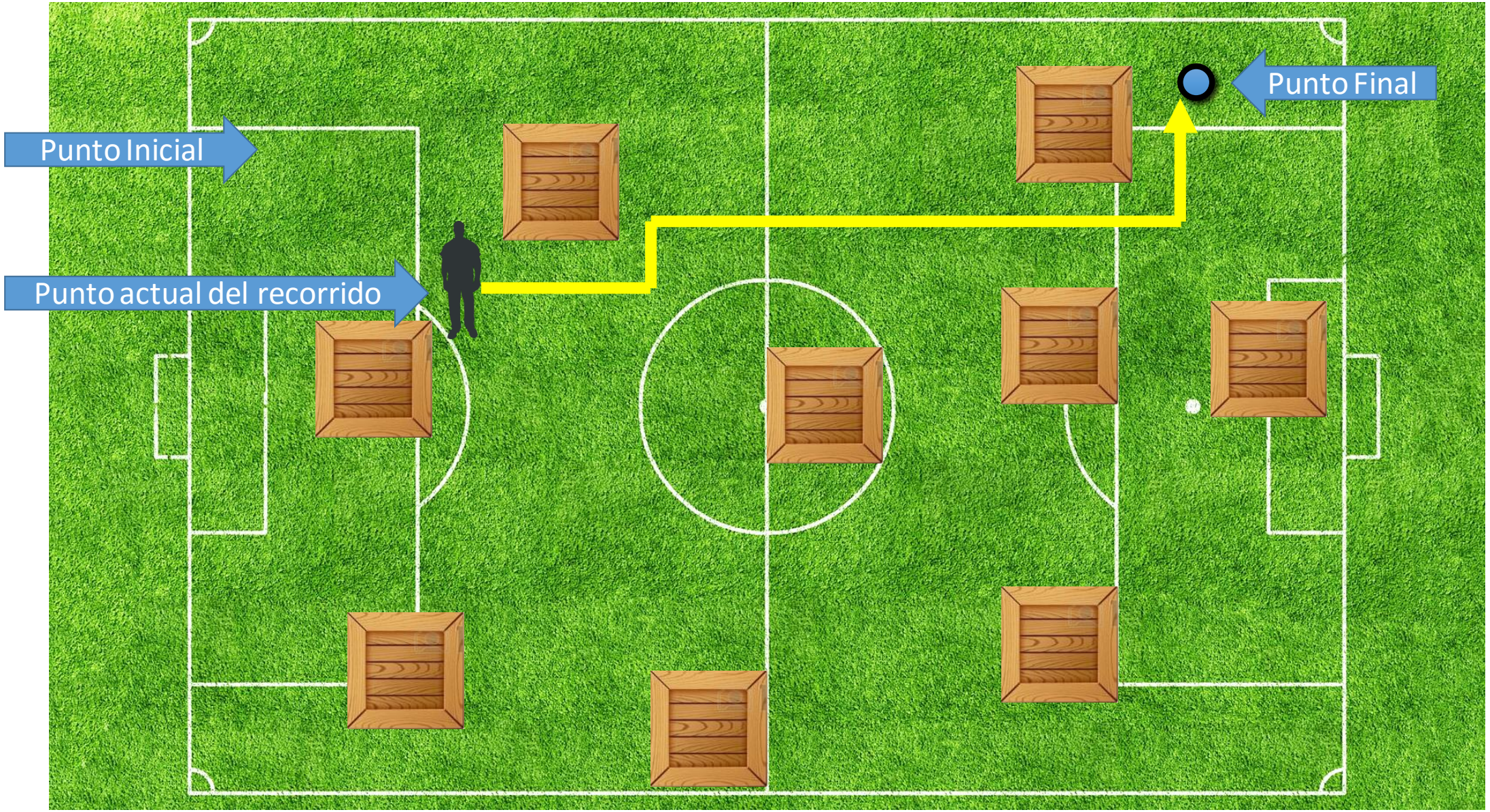
- Por lo tanto, gracias a la matriz pred[][] podemos encontrar la ruta más corta de cualquier vértice **u** a otro **v**

- **El siguiente algoritmo en C encuentra la ruta entre 2 vértices usando la matriz pred[][]**

Para ejercitar y comprobar

```
/**
* Output path as vector of vertices from s to t given the pred results
* from an allPairsShortest execution. Note that s and t must be valid
* integer vertex identifiers. If no path is found between s and t, then an
* empty path is returned.
*/

void constructShortestPath(
   int s, int t, /* in */
   vector< vector<int> > const &pred, /* in */
   list<int> &path)  /* out */
   {
      path.clear( );
      if (t < 0 || t >= (int) pred.size( ) || s < 0 || s >= (int) pred.size( )) {
         return;
      }

   // construct path until we hit source 's' or −1 (NULL) if there is no path.
   path.push_front(t);
   while (t != s) {
      t = pred[s][t];
      if (t == -1) { //if t is NULL
         path.clear( );
         return;
      }
      path.push_front(t);
   }
}
```


Punto Inicial
Punto Final


Punto Inicial
Punto actual del recorrido
Punto Final

CHECK - OBJETIVOS DE LA SESIÓN

- Analizar el Problema de los caminos mínimos entre todos los pares de vértices en un grafo.
- Analizar el Algoritmo de Floyd-Warshall

CHECK

Diseño de Algoritmos

Sesión 12

Profesores:

Tomás Lara Valdovinos – sir.thomas.lara@gmail.com

Jessica Meza-Jaque – jessicamezajaque.uchile@gmail.com