Grafos

CAMINOS MINIMOS

Vamos a ver…

Grafos:

- Definición
- Ejemplo
- Tipos

Caminos mínimos:

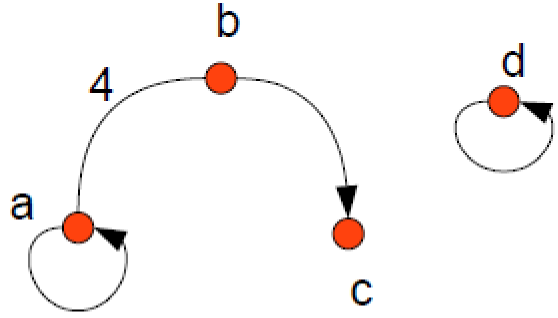
- Dijkstra
- Floyd-Warshall

UNAHUR

Grafos: Definición

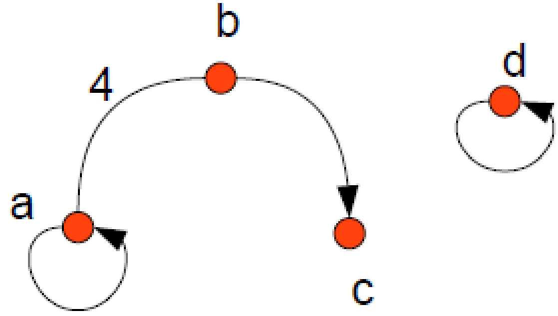
- Dupla compuesta por un conjunto no vacío de vértices y; un conjunto de aristas que vinculan pares de esos vértices.
- Las aristas pueden ser *dirigidas* o *no dirigidas*.

- G = (V ; A)   V = { a ; b ; c ; d}   A = { {a ; b} ; (b ; c) ; (a ; a) ; (d ; d) }



Grafos: Otras definiciones

- **Vértice aislado**: sin relación (mediante aristas) con otro vértice.
- **Lazo**: arista cuyo vértice origen y destino coincide.
- **Arista ponderada**: a las aristas se les puede asociar un valor representativo de la relación que representan, en este caso podría representar la cantidad de Km. entre la ciudad a y b.
- **Peso de una arista**: es el valor asociado a una arista ponderada.



Grafos: Otras definiciones

- **Vértices adyacentes** (v,w): v y w son adyacentes si están relacionados.
- **Incidencia** (v): conjunto de aristas finalizadas / comenzadas en v.
- **Incidencia de entrada** (v): conjunto de aristas dirigidas finalizadas en v.
- **Incidencia de salida** (v): conjunto de aristas dirigidas comenzadas en v.
- **Adyacencia** (v): conjunto de vértices relacionados con v.
- **Adyacencia de entrada** (v): vértices iniciales de la incidencia de entrada (v).
- **Adyacencia de salida** (v): vértices finales de la incidencia de salida (v).

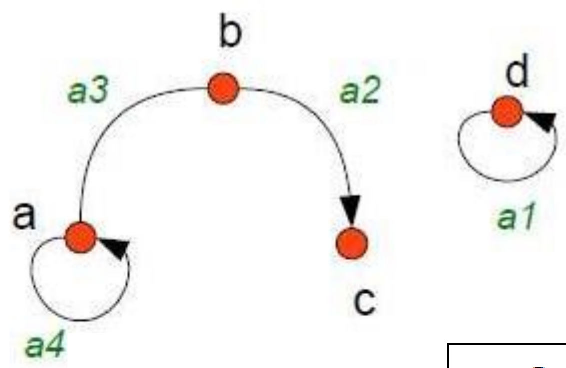
UNAHUR

Grafos: Otras definiciones

- **Grado** (v): cantidad de ocurrencias de v en el conjunto de aristas.
- **Grado de entrada** (v): cantidad de aristas dirigidas finalizadas en v.
- **Grado de salida** (v): cantidad de aristas dirigidas comenzadas en v.
- **Fuente**: vértice cuyo grado de salida es 0 y no es aislado.
- **Sumidero**: vértice cuyo grado de entrada es 0 y no es aislado.

UNAHUR

Grafos: Otras definiciones

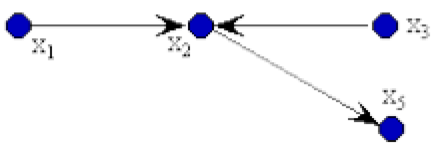


A = { {a ; b} ; (b ; c) ; (a ; a) ; (d ; d) }

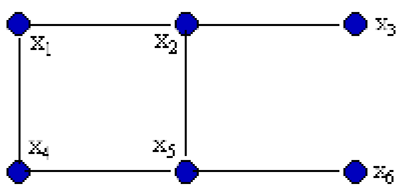
	a	b	c	d
Adyacencia	*{a,b}*	*{a,c}*	*{b}*	*{d}*
Adyacencia de Entrada	*{a}*	*{}*	*{b}*	*{d}*
Adyacencia de Salida	*{a}*	*{c}*	*{}*	*{d}*
Incidencia	*{a3,a4}*	*{a3,a2}*	*{a2}*	*{a1}*
Incidencia de Entrada	*{a4}*	*{}*	*{a2}*	*{a1}*
Incidencia de Salida	*{a4}*	*{a2}*	*{}*	*{a1}*
Grado	*3*	*2*	*1*	*2*
Grado de Entrada	*1*	*0*	*1*	*1*
Grado de Salida	*1*	*1*	*0*	*1*

UNAHUR

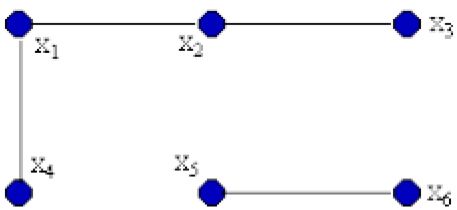
Grafos: Tipos



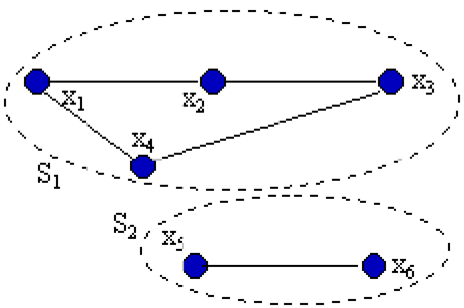
Conexo dirigido



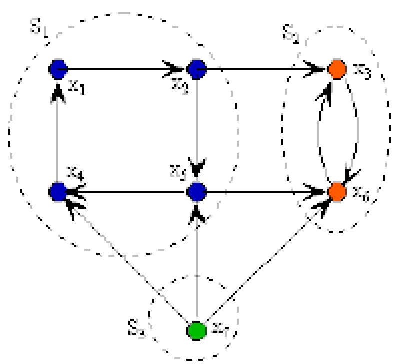
Conexo



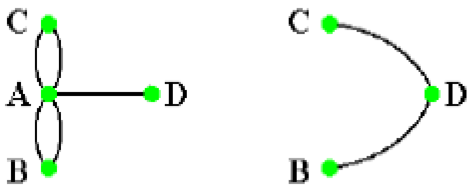
Disconexo



Componentes Conexas



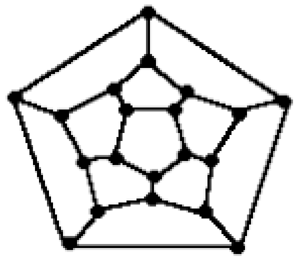
Componentes fuertemente conexas



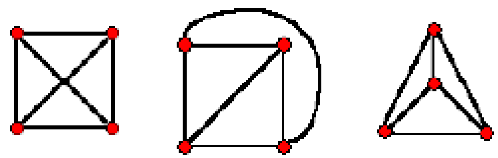
Subgrafo

UNAHUR

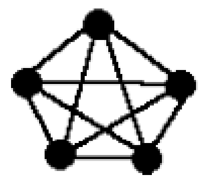
Grafos: Tipos



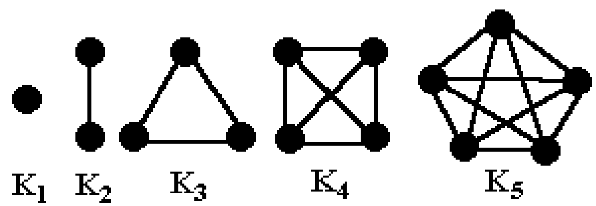
Grafo regular



Grafos planos



Grafo no plano



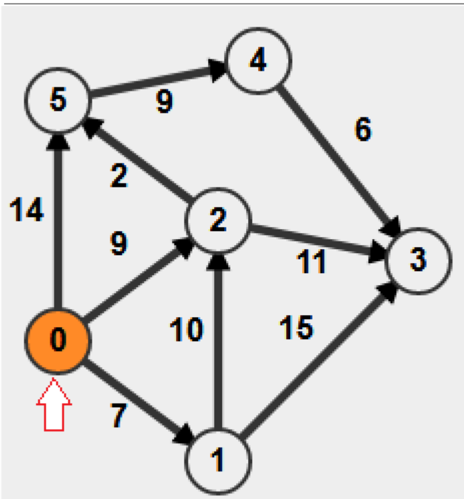
Grafos completos

Dijkstra

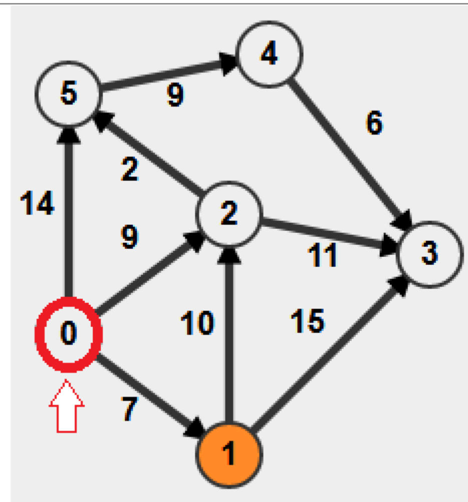
- Es un algoritmo para la determinación del *camino mínimo*, dado un **vértice origen**, hacia el resto de los vértices en un grafo que tiene pesos en cada arista.
- Consiste en ir *explorando todos los caminos más cortos* que parten del vértice origen y que llevan a todos los demás vértices.
- Cuando se obtiene el camino más corto desde el vértice origen hasta el resto de los vértices que componen el grafo, el algoritmo se detiene.

UNAHUR

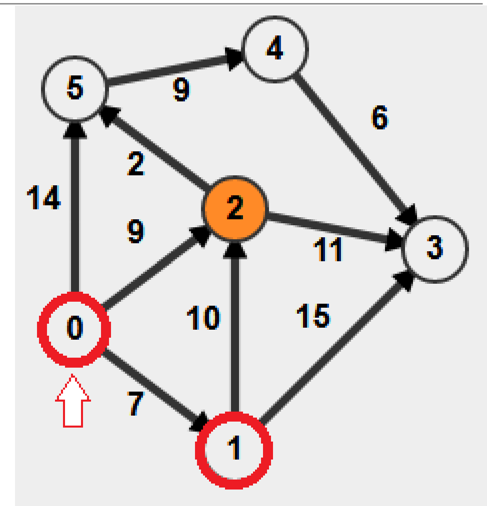
Dijkstra



distancia:	0	7	9	∞	∞	14
	0	1	2	3	4	5
visto:	t	f	f	f	f	f
	0	1	2	3	4	5



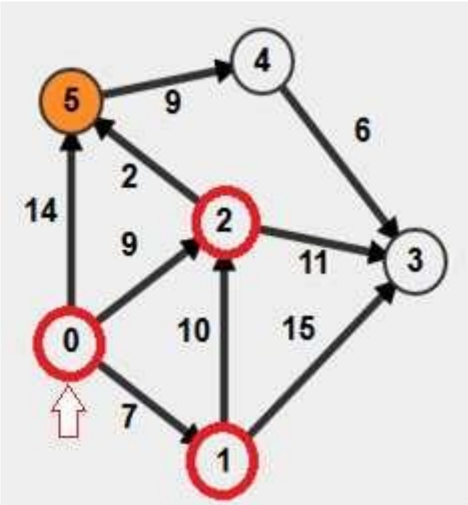
distancia:	0	7	9	22	∞	14
	0	1	2	3	4	5
visto:	t	t	f	f	f	f
	0	1	2	3	4	5



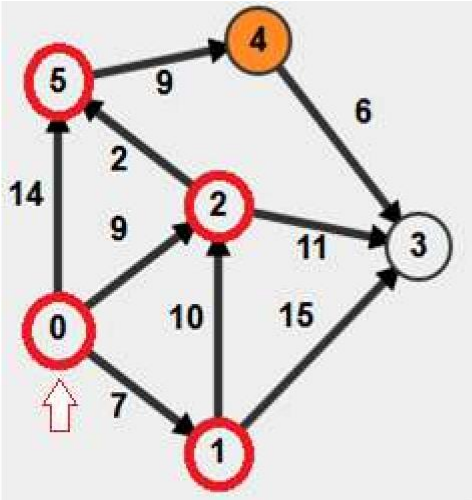
distancia:	0	7	9	20	∞	11
	0	1	2	3	4	5
visto:	t	t	t	f	f	f
	0	1	2	3	4	5

UNAHUR

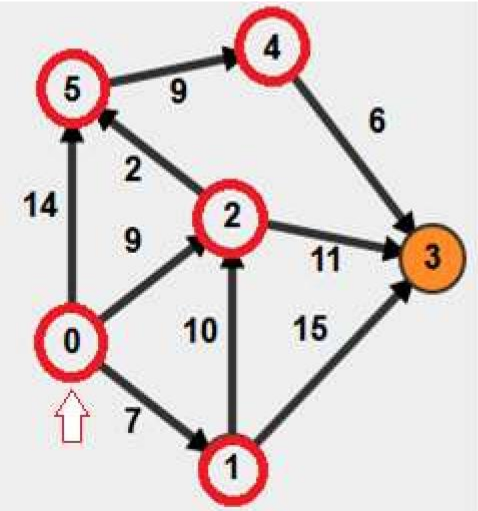
Dijkstra



distancia:	0	7	9	20	20	11
	0	1	2	3	4	5
visto:	t	t	t	f	f	t
	0	1	2	3	4	5



distancia:	0	7	9	20	20	11
	0	1	2	3	4	5
visto:	t	t	t	f	t	t
	0	1	2	3	4	5



distancia:	0	7	9	20	20	11
	0	1	2	3	4	5
visto:	t	t	t	t	t	t
	0	1	2	3	4	5

UNAHUR

Dijkstra

```
función Dijkstra (Grafo G, nodo_inicial s)
entero distancia[n]
booleano visto[n]
  para cada w ∈ V[G] hacer
     Si (no existe arista entre s y w) entonces
         distancia[w] = Infinito
     Si_no
         distancia[w] = peso (s, w)
     fin si
  fin para
  distancia[s] = 0
  visto[s] = true
```

Dijkstra

```
mientras que (no_estén_vistos_todos) hacer
      vértice = tomar_el_mínimo_del_vector distancia y que no esté
visto;
      visto[vértice] = true;
      para cada w ∈ sucesores (G, vértice) hacer
            si distancia[w]>distancia[vértice]+peso (vértice, w)
entonces
               distancia[w] = distancia[vértice]+peso (vértice, w)
            fin si
      fin para
fin mientras
fin función.
```

Dijkstra (con cola de prioridad)

```
DIJKSTRA (Grafo G, nodo_fuente s)
      para u ∈V[G] hacer
          distancia[u] = INFINITO
          padre[u] = NULL
          visto[u] = false
      distancia[s] = 0
      adicionar (cola, (s, distancia[s]))
      mientras que cola no es vacía hacer
          u = extraer_mínimo(cola)
          visto[u] = true
          para todos v ∈adyacencia[u] hacer
              si ¬ visto[v]
                  si distancia[v] > distancia[u] + peso (u, v) hacer
                      distancia[v] = distancia[u] + peso (u, v)
                      padre[v] = u
                      adicionar(cola,(v, distancia[v]))
```

UNAHUR

Floyd-Warshall

- Es otro algoritmo para encontrar el camino mínimo en grafos dirigidos ponderados.
- Encuentra el camino entre todos los pares de vértices en una única ejecución.
- Compara todos los posibles caminos a través del grafo entre cada par de vértices.
- Para que haya coherencia numérica, supone que no hay ciclos negativos.

UNAHUR

Floyd-Warshall

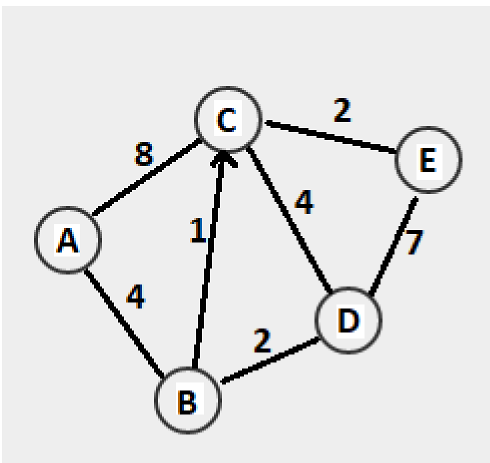
- Formar las matrices iniciales D y R, donde D es la matriz de adyacencia o *distancias,* y R es una matriz de *recorridos* del mismo tamaño vacía.
- Se toma k=1.
- Se selecciona la fila y la columna k de la matriz D y entonces, para i y j, con i≠k, j≠k e i≠j, hacemos:

  Si (Dik + Dkj) < Dij → Dij = Dik + Dkj y Rij = Rkj

  En caso contrario, dejamos las matrices como están.
- Si k ≤ n, aumentamos k en una unidad y repetimos el paso anterior, en caso contrario páramos las interacciones.
- La matriz final D contiene los costos óptimos para ir de un vértice a otro, mientras que la matriz R contiene los penúltimos vértices de los caminos óptimos que unen dos vértices.

UNAHUR

Floyd-Warshall



iteracion0

distancias:

	A	B	C	D	E
A	0	4	8	∞	∞
B	4	0	1	2	∞
C	8	∞	0	4	2
D	∞	2	4	0	7
E	∞	∞	2	7	0

recorridos:

	A	B	C	D	E
A	-	B	C	D	E
B	A	-	C	D	E
C	A	B	-	D	E
D	A	B	C	-	E
E	A	B	C	D	-

iteracion1

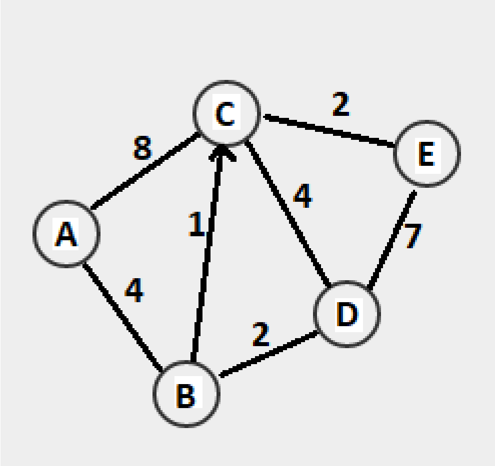
distancias:

	A	B	C	D	E
A	0	4	8	∞	∞
B	4	0	1	2	∞
C	8	12	0	4	2
D	∞	2	4	0	7
E	∞	∞	2	7	0

recorridos:

	A	B	C	D	E
A	-	B	C	D	E
B	A	-	C	D	E
C	A	A	-	D	E
D	A	B	C	-	E
E	A	B	C	D	-

Floyd-Warshall



iteracion1

distancias:

	A	B	C	D	E
A	0	4	8	∞	∞
B	4	0	1	2	∞
C	8	12	0	4	2
D	∞	2	4	0	7
E	∞	∞	2	7	0

recorridos:

	A	B	C	D	E
A	-	B	C	D	E
B	A	-	C	D	E
C	A	A	-	D	E
D	A	B	C	-	E
E	A	B	C	D	-

iteracion2

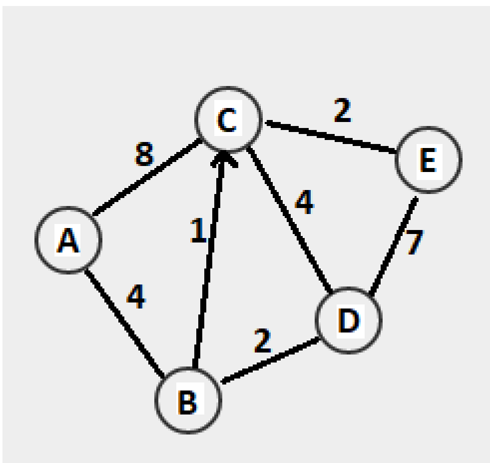
distancias:

	A	B	C	D	E
A	0	4	5	6	∞
B	4	0	1	2	∞
C	8	12	0	4	2
D	6	2	3	0	7
E	∞	∞	2	7	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	E
B	A	-	C	D	E
C	A	A	-	D	E
D	B	B	B	-	E
E	A	B	C	D	-

Floyd-Warshall



iteracion2

distancias:

	A	B	C	D	E
A	0	4	5	6	∞
B	4	0	1	2	∞
C	8	12	0	4	2
D	6	2	3	0	7
E	∞	∞	2	7	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	E
B	A	-	C	D	E
C	A	A	-	D	E
D	B	B	B	-	E
E	A	B	C	D	-

iteracion3

distancias:

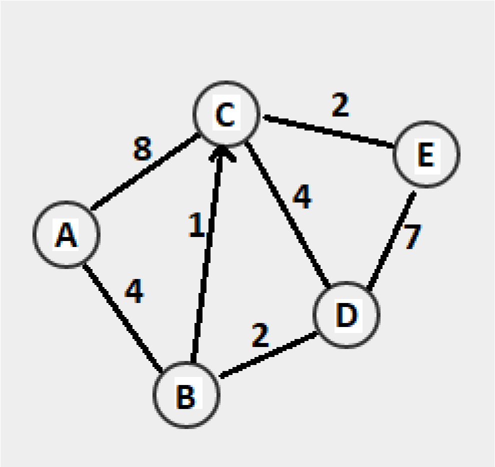
	A	B	C	D	E
A	0	4	5	6	7
B	4	0	1	2	3
C	8	12	0	4	2
D	6	2	3	0	5
E	10	14	2	6	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	C
B	A	-	C	D	C
C	A	A	-	D	E
D	B	B	B	-	C
E	C	C	C	C	-

UNAHUR

Floyd-Warshall



iteracion3

distancias:

	A	B	C	D	E
A	0	4	5	6	7
B	4	0	1	2	3
C	8	12	0	4	2
D	6	2	3	0	5
E	10	14	2	6	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	C
B	A	-	C	D	C
C	A	A	-	D	E
D	B	B	B	-	C
E	C	C	C	C	-

iteracion4

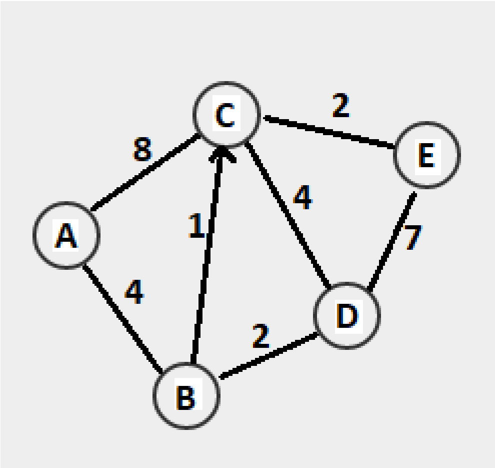
distancias:

	A	B	C	D	E
A	0	4	5	6	7
B	4	0	1	2	3
C	8	6	0	4	2
D	6	2	3	0	5
E	10	8	2	6	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	C
B	A	-	C	D	C
C	A	D	-	D	E
D	B	B	B	-	C
E	C	D	C	C	-

Floyd-Warshall



iteracion4

distancias:

	A	B	C	D	E
A	0	4	5	6	7
B	4	0	1	2	3
C	8	6	0	4	2
D	6	2	3	0	5
E	10	8	2	6	0

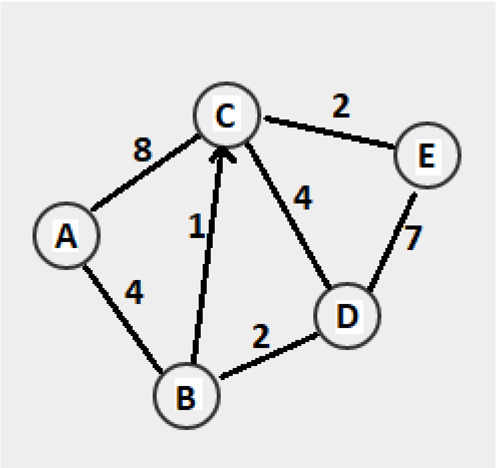
recorridos:

	A	B	C	D	E
A	-	B	B	B	C
B	A	-	C	D	C
C	A	D	-	D	E
D	B	B	B	-	C
E	C	D	C	C	-

UNAHUR

Floyd-Warshall

- Para leer el recorrido de A a E:



iteracion4

distancias:

	A	B	C	D	E
A	0	4	5	6	7
B	4	0	1	2	3
C	8	6	0	4	2
D	6	2	3	0	5
E	10	8	2	6	0

recorridos:

	A	B	C	D	E
A	-	B	B	B	C
B	A	-	C	D	C
C	A	D	-	D	E
D	B	B	B	-	C
E	C	D	C	C	-

Floyd-Warshall

```
vector< vector<int> > ady;

vector< vector<int> > Grafo :: floydWarshall(){

    vector< vector<int> > distancias = this->ady;

    for(int i = 0; i < n; i++) //n: cantidad de nodos

        distancias[i][i] = 0;

    for(int k = 0; k < n; k++)

        for(int i = 0; i < n; i++)

            for(int j = 0; j < n; j++){

                int dt = distancias[i][k] + distancias[k][j];

                if(dt < distancias[i][j])

                    distancias[i][j] = dt;    }

    return distancias;}
```

Fin

UNAHUR