ESTRUCTURAS DE DATOS y ALGORITMOS

Licenciatura en Ciencias de la Computación

GRAFO / DIGRAFO

Objetivos

- Conocer y evaluar los algoritmos tradicionales de manipulación de Grafo/Digrafo

- Evaluar las distintas alternativas de representación de Grafo/Digrafo

- Construir y usar, en problemáticas particulares, los TAD Grafo / TAD Digrafo

GRAFO

Un ***grafo no dirigido*** G=(V,E) se compone de dos conjuntos finitos:

- el conjunto V={v1, v2,...}, que es el conjunto de ***nodos o vértices*** de G y
- el conjunto de ***aristas*** E={e1, e2,...} que es un conjunto de pares **no ordenados** de nodos diferentes de G. $E \subseteq V \times V$

Cada elemento del conjunto E, e=(u,v) se conoce como una ***arista*** y se dice que une los nodos u y v.

Si e=(u,v) es una arista de G, entonces los nodos u y v son ***adyacentes.*** Dado que estamos tratando con pares no ordenados, (u,v) y (v,u) representan la misma arista.

GRAFO

- El ***grado*** de un nodo en un grafo no dirigido está dado por el número de aristas en las que participa dicho nodo.

- Un ***camino P de longitud n***, desde un nodo *u* a un nodo *v*, se define como la secuencia de n+1 nodos : P(u,v)=(v0, v1, ...,vn) donde: u=v0, v=vn, y vi es adyacente a vi-1, 1<=i<=n.

- Un ***camino P es cerrado*** si v0=vn.

- Un ***camino P es simple*** si todos los nodos son distintos, a excepción de *v0* que puede ser igual a *vn*.

- Un ***ciclo*** es un camino simple cerrado de longitud 3 o mas. Un ciclo de longitud k se llama k-ciclo.

GRAFO

$V=\{0,1,2,3,4\}$

$E=\{(0,1), (0,4), (1,3), (1,4), (2,4)\}$

Sec. 11.1 Terminol

Grado(0)= 2
Grado(4)=3
Camino(2,3)= (2,4,1,3)
(2,4,0,1,3)

$G=(V,E)$

GRAFO

•Un ***grafo acíclico***, también llamado ***bosque***, es aquel grafo que no contiene ningún ciclo.

•Si hay un camino desde el nodo *u* al nodo *v*, entonces diremos que *v* es ***accesible*** desde *u*.

•Un ***grafo*** G es ***conexo***, si al menos existe un camino entre cada par de nodos del grafo.

•***Árbol*** es un grafo acíclico conexo.

•Un grafo G está ***etiquetado***, si sus aristas tienen datos asignados.

GRAFO

- Un grafo G tiene peso- ***grafo valorado*** o ***ponderado*** o ***con peso-***, si cada arista *e* de G tiene asignado un valor numérico, *w(e)*, llamado peso o longitud de *e*. $w:\{(u,v)\in E\}\rightarrow R^{+} \cup \{0\}$

- En los grafos ponderados el peso de un camino P es la suma de los pesos de las aristas que unen los nodos que forman el camino. El camino de menor peso, entre dos nodos, se denomina ***camino mínimo***. Si el grafo no es valorado, el camino mínimo entre dos nodos es aquel que contiene el menor número de aristas.

- ***Arbol de recubrimiento*** de un grafo G: subgrafo sin ciclos que contiene a todos los vértices de G. En caso de haber varios árboles de coste mínimo se elige el que posee menos arcos.

GRAFO

Camino(2,3)= (2,4,1,3)
(2,4,0,1,3)

7

5


Sec. 11.1 Termino

3

Camino-Minimo(2,3)= (2,4,0,1,3)

14

9

G=(V,E)

DIGRAFO

Un ***grafo dirigido*** o ***digrafo***, G=(V,E), es tal que cada arista *e* de G tiene una dirección asignada; es decir, cada arista *e* está identificada por un ***par ordenado*** (u,v) de nodos de G.

- ***e*** empieza en ***u*** y termina en ***v***.
- ***u*** es el origen o punto inicial de *e*, y ***v*** es el destino o punto terminal de *e*.
- ***u*** es un predecesor de ***v*** y ***v*** es sucesor de ***u***.

DIGRAFO

•El ***grado de salida o de emisión*** de un nodo *u*, -grad_sal(u)-, es el número de aristas que empiezan en u.

•El ***grado de entrada o de recepción*** de un nodo *u*, - grad_ent(u)-, es el número de aristas que terminan en u.

•Un nodo *u* es ***nodo fuente***, si grad_sal(u)>0 y grad_ent(u)=0.

•Un nodo *u* es ***nodo sumidero***, si grad_ent(u)>0 y grad_sal(u)=0.

DIGRAFO

•***Camino, cadena o trayectoria*** entre los nodos u y v, es una secuencia de nodos P(u,v)=(v0, v1,...,vn), u=v0 y v=vn tal que (v0,v1) $\in$ E, (v1,v2) $\in$ E,..., (vn-1,vn) $\in$ E.

•Un ***digrafo* G *es fuertemente conexo***, si para cada par de nodos *u,v* $\in$V, existe un camino de u a *v* ***y*** un camino de *v* a *u*.

•Un ***dígrafo G es simple conexo***, si para cada par de nodos *u,v* $\in$V, existe un camino de *u* a *v* **o** un camino de *v* a *u*.

DIGRAFO

384

400

```
void tophelp(Graph* G
  G->setMark(v, VISIT
  for (int w=G->first
    if (G->getMark(w)
      tophelp(G, w);
```

GRAFO/DIGRAFO

Aplicaciones frecuentes

Red de comunicaciones -computadoras, aeropuertos, ciudades, terminales, depósitos u otras entidades que pretendan comunicarse- bidireccional o no.

Capacidades conocidas -ancho de banda, distancias, tiempos-, de los canales de comunicación, correspondientes a una red de comunicaciones .

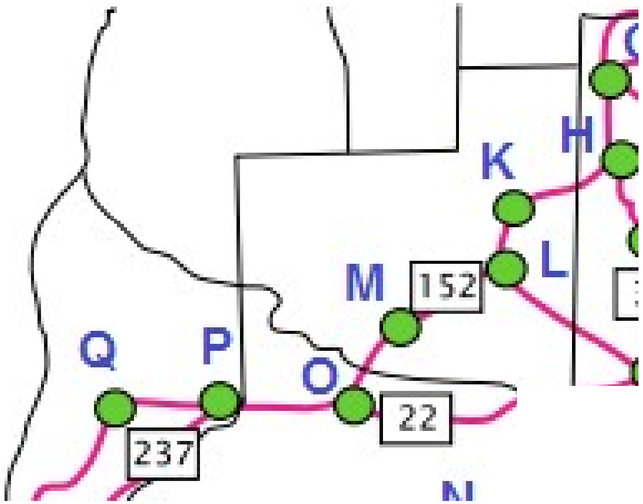
Hipertextos o multimedias.

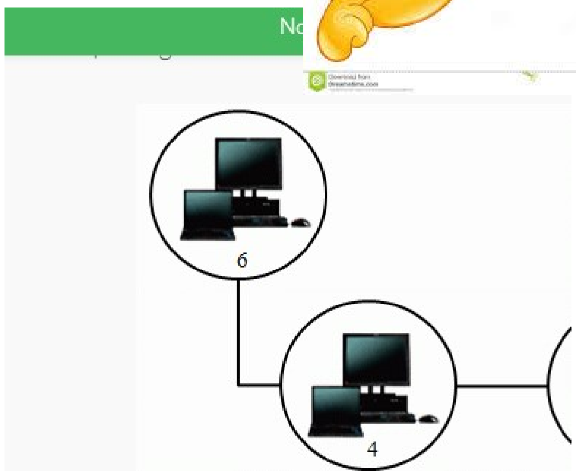
Relaciones de precedencia que pueden existir entre las tareas que deben realizarse para completar un trabajo.

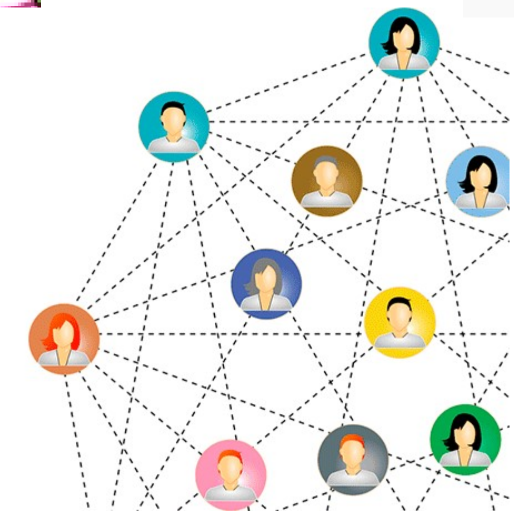
Ver Video Teoria de Grafos-Adrian Paenza.mp4

GRAFO/DIGRAFO

Aplicaciones

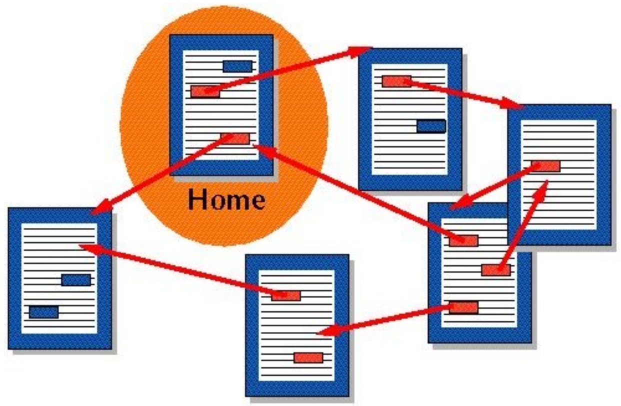


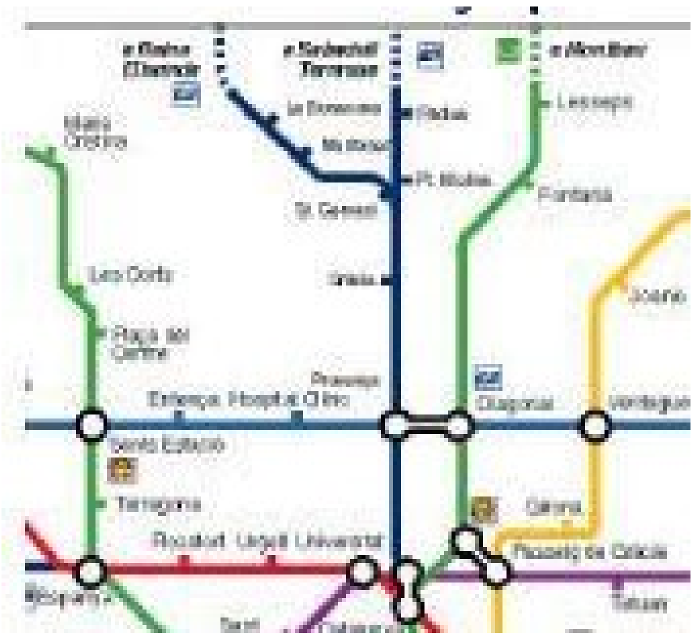


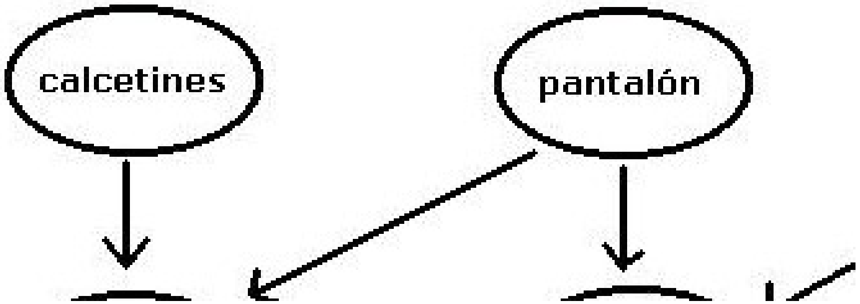


GRAFO/DIGRAFO

Aplicaciones







GRAFO/DIGRAFO

Requerimientos funcionales habituales

¿Que lugares son directamente accesibles desde un sitio particular?

¿En una red vial, están todas las ciudades conectadas?

¿En una red de computadoras, es posible transmitir información desde una computadora a cualquier otra?

Dados dos puntos cualesquiera de una red: ¿Cuál es el camino mas directo (con el menor número de conexiones)? o ¿Cuál es el camino mas corto- distancia, tiempo, costo-?

¿Cuál sería una secuencia posible de realización de tareas?

T.A.D. GRAFO/DIGRAFO – Especificación

Operaciones Abstractas

Sean **G**: Grafo/Digrafo y **u,v**: nodos

NOMBRE	*ENCABEZADO*	*FUNCION*	*ENTRADA*	*SALIDA*
Adyacentes	Adyacentes(G,u)	Determina los nodos adyacentes de u	G y u	Reporta los nodos adyacentes a u.
Camino	Camino(G,u,v)	Determina el camino de u a v	G, u y v	Reporta el camino de u a v, si v es alcanzable desde u; Error en caso contrario
Camino-Mínimo	Camino-Mínimo(G,u,v)	Reporta el camino mínimo de u a v	G, u y v	Reporta el camino mínimo de u a v, si v es alcanzable desde u; Error en caso contrario
Conexo	Conexo(G)	Evalúa si G es conexo	G	V si G es conexo, F en caso contrario
Acíclico	Acíclico(G)	Evalúa si G es acíclico	G	V si G es acíclico, F en caso contrario
Árbol de Recubrimiento	Arbol de Recubrimiento (G)	Reporta el Árbol de recubrimiento (mínimo) de G	G (grafo conexo, ponderado)	Árbol de recubrimiento (mínimo) de G
REA	REA(G)	Procesa todos los elementos de G en anchura	G	Está sujeta al proceso que se realice sobre los elementos de G
REP	REP(G)	Procesa todos los elementos de G en profundidad	G	Está sujeta al proceso que se realice sobre los elementos de G

T.A.D. GRAFO/DIGRAFO – Representación

Representación Secuencial

- Matriz de Adyacencia

$$A[i,j]= \begin{cases} 1 & si\ (i,j) \in E \\ 0 & si\ (i,j) \notin E \end{cases}$$

- Matriz de Pesos

$$W[i,j]= \begin{cases} w & si\ (i,j) \in E \\ 0 & si\ (i,j) \notin E \end{cases}$$

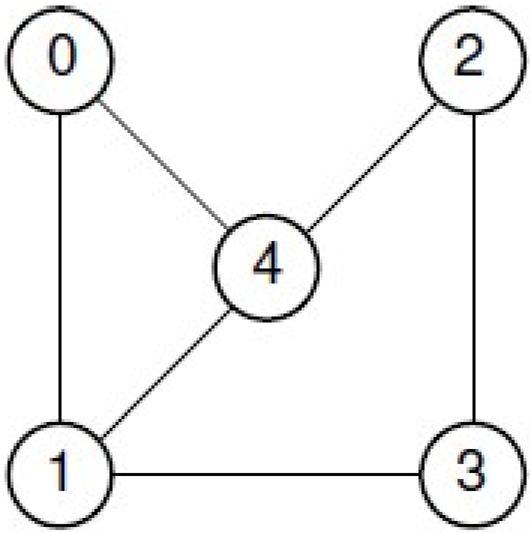
Representación Encadenada

- Listas de Adyacencia

T.A.D. GRAFO

Representación

Sec. 11.1 Terminology and Representations



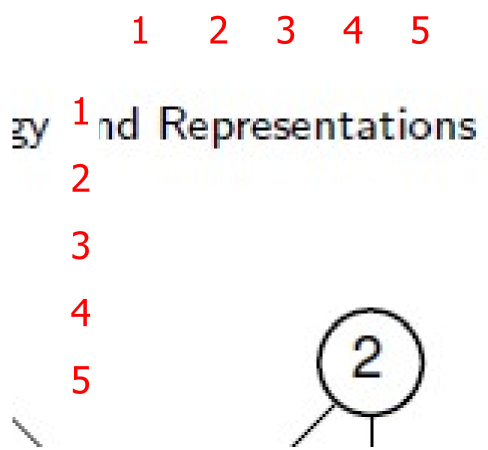
(a)

T.A.D. GRAFO

Representación Secuencial

A

A[1,1]	A[1,2]	A[1,3]	A[1,4]	A[1,5]
A[2,1]	A[2,2]	A[2,3]	A[2,4]	A[2,5]
A[3,1]	A[3,2]	A[3,3]	A[3,4]	A[3,5]
A[4,1]	A[4,2]	A[4,3]	A[4,4]	A[4,5]
A[5,1]	A[5,2]	A[5,3]	A[5,4]	A[5,5]



Cuantas componentes se requieren para almacenar la matriz simétrica?

$|V| = n$

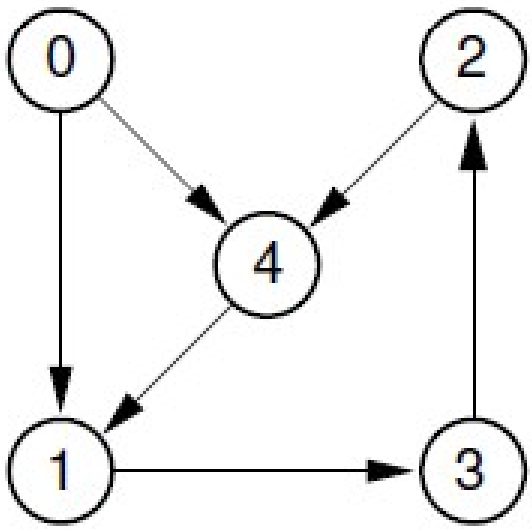
$$\frac{n(n+1)}{2}$$ componentes

Como se localiza un elemento particular?

$$Loc(A[i,j]) = \frac{i(i-1)}{2} + j$$

T.A.D. DIGRAFO – Representación

384



(a)

T.A.D. GRAFO/DIGRAFO – Representación

Cuanta memoria requiere cada representación?

T.A.D. GRAFO/DIGRAFO – Representación

TAD / REPRESENTACION	GRAFO	DIGRAFO
SECUENCIAL		
ENLAZADA		

T.A.D. GRAFO/DIGRAFO –

Construcción de las Operaciones Abstractas

Recorrido en Anchura -REA

Recorrido o **Búsqueda en anchura** - *BFS* - (Breadth First Search) es un algoritmo utilizado para recorrer o buscar elementos en un grafo. Intuitivamente, se comienza algún nodo origen y se exploran todos los vecinos de este nodo. A continuación para cada uno de los vecinos se exploran sus respectivos vecinos adyacentes, y así hasta que se recorra todo el grafo.

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

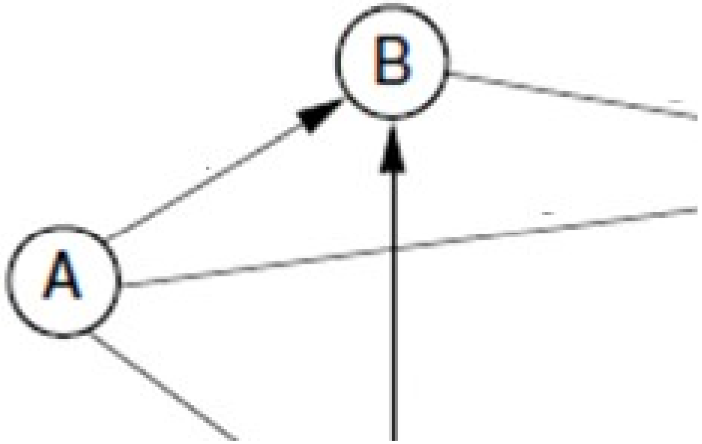
Recorrido (Búsqueda) en Anchura

```
REA (G, s)          s es el origen de G- Grafo/Digrafo
    Para cada v ∈ V hacer
        d[ v ] ← ∞                  todos los nodos están no marcados
    FinPara
    d[s] ← 0                        marcar el origen
    Insertar ( Cola , s)
    Mientras no Vacía ( Cola)  hacer
        Suprimir (Cola , v)
        Para Cada u Adyacente a v  hacer
            Si  d [ u ] = ∞
                entonces    d [ u ] ← d [ v ] + 1      marcar  u
                            Insertar (Cola, u)
            FinSi
        FinPara
    FinMientras
```

T.A.D. GRAFO/DIGRAFO – Construcción de las Operaciones Abstractas

Algoritmo REA

```
REA (G, s)
    Para cada v ∈ V hacer
          d[ v ] ← ∞
    FinPara
    d[s] ← 0
    Insertar ( Cola , s)
    Mientras no Vacía ( Cola)  hacer
           Suprimir (Cola , v)
           Para Cada u Adyacente a v  hacer
                 Si  d [ u ] = ∞
                     entonces     d [ u ] ← d [ v ] + 1
                                  Insertar (Cola, u)
                 FinSi
           FinPara
     FinMientras
```



G

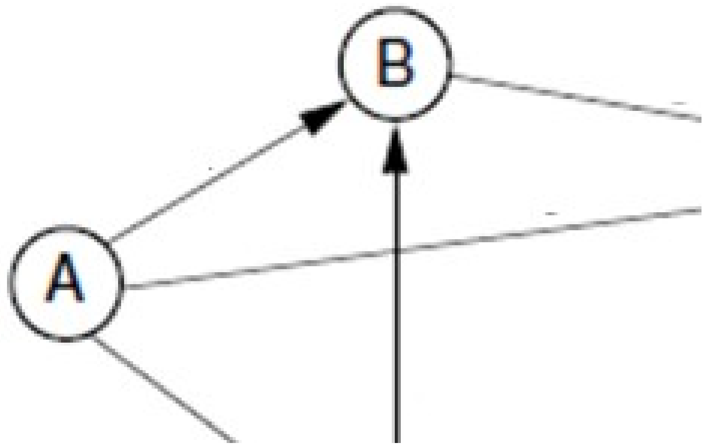
S : A

	A	B	C	D	E
d					

Cola

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

```
REA (G, s)
    Para cada v ∈ V hacer
          d[ v ] ← ∞
    FinPara
    d[s] ← 0
    Insertar ( Cola , s)
    Mientras no Vacía ( Cola)  hacer
           Suprimir (Cola , v)
           Para Cada u Adyacente a v  hacer
                      Si  d [ u ] = ∞
                          entonces     d [ u ] ← d [ v ] + 1
                                       Insertar (Cola, u)
                      FinSi
            FinPara
     FinMientras
```



G

d

0	**1**	**1**	**1**	**2**
A	B	C	D	E

Cola

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

Algoritmo de Dijkstra

El **algoritmo de Dijkstra**, también llamado **algoritmo de caminos mínimos**, es un algoritmo para la determinación del camino más corto desde un vértice origen al resto de vértices, en un grafo ponderado.

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

Dijkstra(T: Tabla);

```
Para i desde 1 hasta |V| hacer
       v ← vertice con la distancia mas corta y desconocido
       T[v].conocido ← True
        Para   cada w adyacente a v  hacer
                 Si T[w].conocido = False
                     entonces
                          Si  T[v].distancia + w(v,w) < T[w].distancia
                                entonces
                                Reducir ( T[w].distancia  a  T[v].distancia + w(v,w))
                                T[w].camino ← v
                           FinSi
                     FinSi
         FinPara
  FinPara
```

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

lucionarlo, sino que además
iente, tenemos 7 ciudades, y
e rápidamente cuál es, pero

Vértices	Conocido	Distancia	Camino
v_1	F	0	
v_2	F	∝	
v_3	F	∝	
v_4	F	∝	
v_5	F	∝	
v_6	F	∝	
v_7	F	∝	

T

T.A.D. GRAFO/DIGRAFO –

Construcción de las Operaciones Abstractas

```
Dijkstra(T: Tabla);

  Para i desde 1 hasta |V| hacer
   v ← vertice con la distancia mas corta y desconocido
   T[v].conocido ← True
    Para   cada w adyacente a v  hacer
       Si T[w].conocido = False
           entonces
              Si  T[v].distancia + w(v,w) < T[w].distancia
                   entonces
                      Reducir ( T[w].distancia  a  T[v].distancia + w(v,w))
                      T[w].camino ← v
               FinSi
        FinSi
    FinPara
FinPara
```

T

Vértices	Conocido	Distancia	Camino
v_1	T	0	
v_2	T	3	v_1
v_3	T	4	v_2
v_4	T	3	v_1
v_5	T	5	v_2
v_6	T	4	v_4
v_7	T	6	v_4

lucionarlo, sino que además
iente, tenemos 7 ciudades, y
e rápidamente cuál es, pero

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

Algoritmo de Prim

El **algoritmo de Prim** encuentra un árbol de recubrimiemto mínimo en un grafo conexo, **no** dirigido y cuyas aristas están ponderadas. En otras palabras, el algoritmo encuentra un subconjunto de aristas que forman un árbol con todos los vértices, donde el peso total de todas las aristas en el árbol es el mínimo posible. Si el grafo no es conexo, entonces el algoritmo encontrará el árbol de recubrimiento mínimo para uno de los componentes conexos que forman dicho grafo no conexo.

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

Prim(T: Tabla);

```
Para i desde 1 hasta |V| hacer
     v ← vertice con la distancia mas corta y desconocido
     T[v].conocido ← True
      Para   cada w adyacente a v  hacer
             Si T[w].conocido = False
                entonces
                    Si  w(v,w) < T[w].distancia
                        entonces
                        T[w].distancia ← w(v,w)
                        T[w].camino ← v
                     FinSi
                FinSi
      FinPara
 FinPara
```

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

T

Vértices	Conocido	Distancia	Camino
v_1	F	0	
v_2	F	∝	
v_3	F	∝	
v_4	F	∝	
v_5	F	∝	
v_6	F	∝	
v_7	F	∝	

http://cybertesis.uni.edu.pe/bitstream/uni/3416/1/tocto_ip.pdf

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

Algoritmos de Warshall y de Floyd

El algoritmo de **Warshall**, es para caminos en grafos dirigidos y el algoritmo de **Floyd** es para caminos mínimos en grafos dirigidos ponderados, representados en matrices de adyacencia o de peso .

T.A.D. GRAFO/DIGRAFO –
Construcción de las Operaciones Abstractas

A: Matriz de Adyacencia

P: Matriz de Camino

```
WARSHALL ( A, P)
P=A
Para   1<= k <=n hacer
        Para   1<= i <=n hacer
              Para   1<= j <=n hacer
                 P [ i , j] = P [ i, j ]  ∨
                          (P [ i , k ] ∧ P  [ k , j ])
              FinPara
        FinPara
FinPara
```

T.A.D. GRAFO/DIGRAFO – Construcción de las Operaciones Abstractas

W: Matriz de Peso   Q: Matriz de Camino Mínimo

```
Floyd ( W, Q)
    Q = W
     Para   1<= k <=n hacer
            Para   1<= i <=n hacer
                   Para   1<= j <=n hacer
                          Q[ i , j] = Mínimo (Q [ i , j] , Q [ i , k] + Q [ k , j] )
                   FinPara
            FinPara
     FinPara
```

T.A.D. GRAFO/DIGRAFO – Construcción de las Operaciones Abstractas

Recorrido (Búsqueda) en Profundidad

```
REP (G)
    Para cada v ∈ V hacer
            d[v] ← 0
    FinPara
    tiempo ← 0
    Para cada s ∈ V hacer
            Si d[s] = 0
                entonces
                    REP-Visita (G,s)
            FinSi
    FinPara
```

```
REP- Visita (G, s)
        tiempo ← tiempo + 1
        d[ s] ← tiempo
        Para Cada u Adyacente a s  hacer
            Si d[ u ] = 0
                entonces
                    REP-Visita (G,u)
        FinPara
        tiempo ← tiempo + 1
        f [ s ] ← tiempo
```

Ordenación - Topológica (G)

Ejecutar REP (G) insertando nodos a la cabeza de la lista L conforme son terminados

Retornar L