Analista Programador Universitario

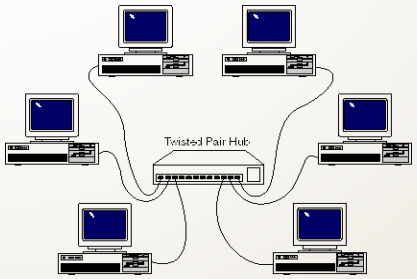
Estructura de Datos

Unidad VII: Grafos

Facultad de Ingeniería
Universidad Nacional de Jujuy

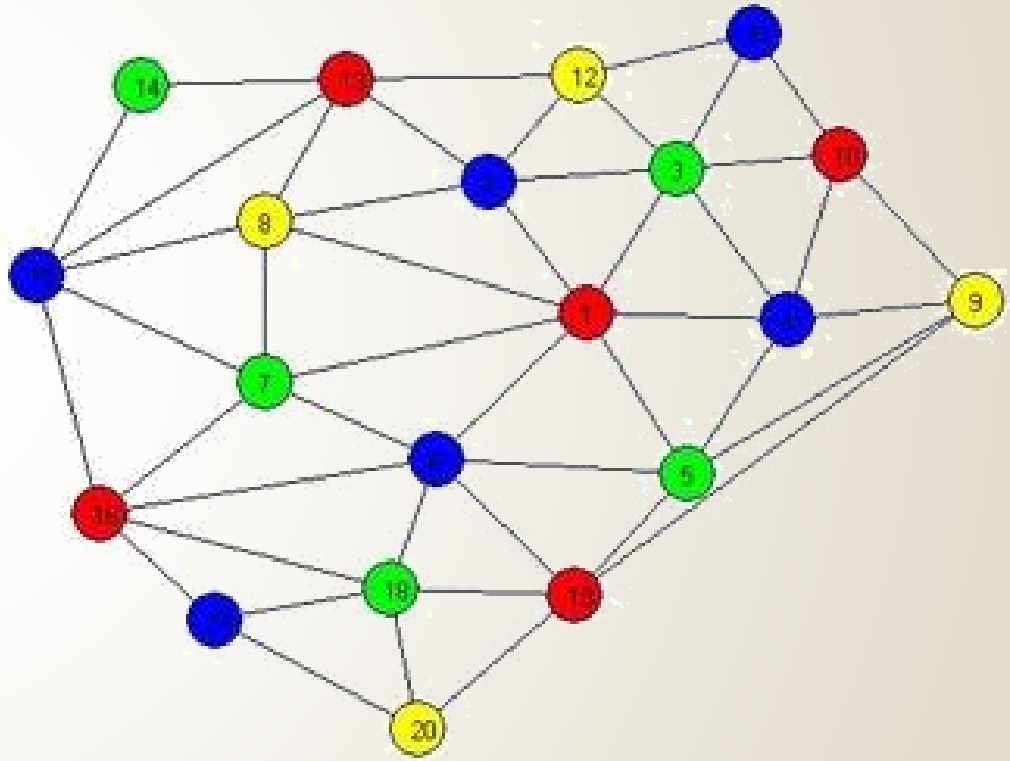
Relaciones Múltiples

- Muchas veces se presentan situaciones en las que es preciso representar relaciones múltiples (muchos a muchos) entre varios elementos.
- Por ejemplo: rutas aéreas entre ciudades, rutas marítimas, rutas de distribución de productos, *networks*, redes sociales, etc.



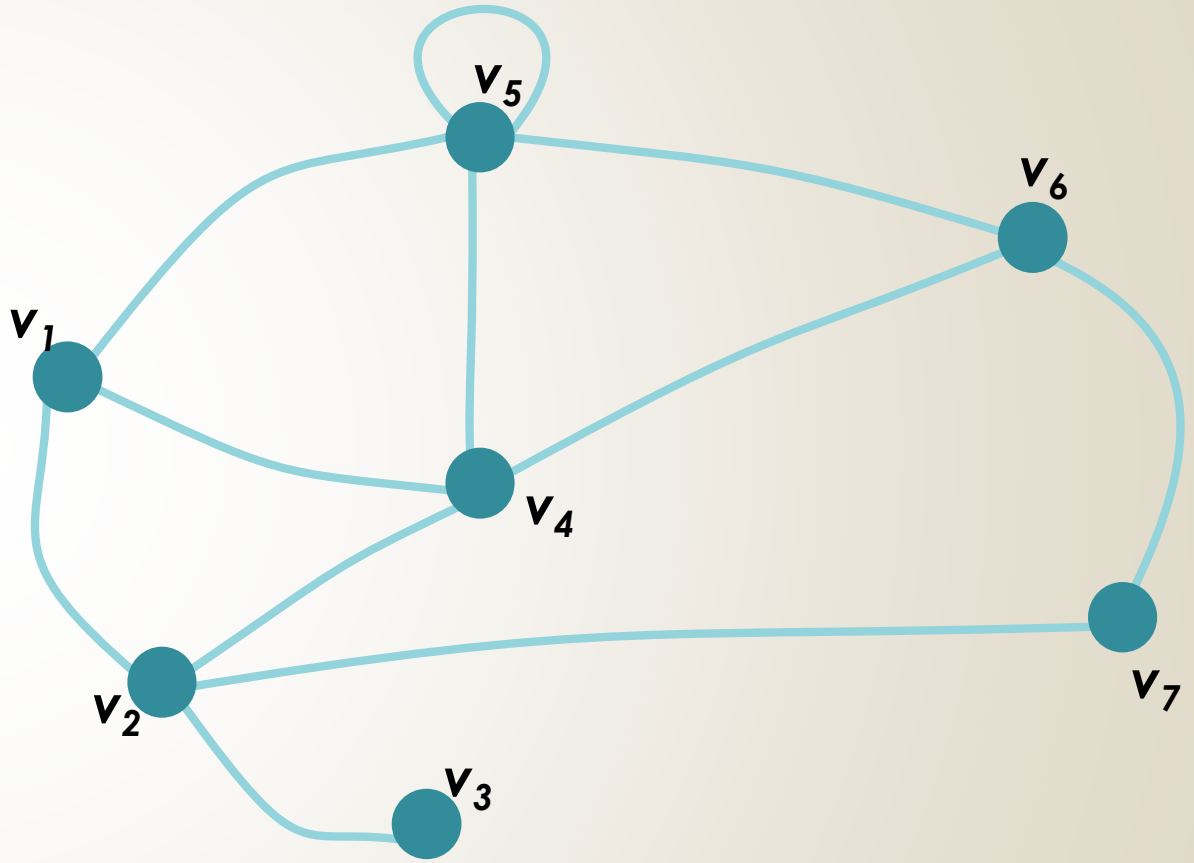
Grafos

- Para tales situaciones es necesario utilizar una estructura que represente los elementos de interés y sus relaciones. La estructura aplicable en estos casos se llama *Grafo*.



Grafos. Definición (1)

Un grafo *G=(V, A)* está formado por un conjunto de vértices (nodos) *V* y un conjunto de arcos (aristas o enlaces) *A*. Cada arco se representa como (v_i , v_j), donde v_i y v_j pertenecen a *V*.

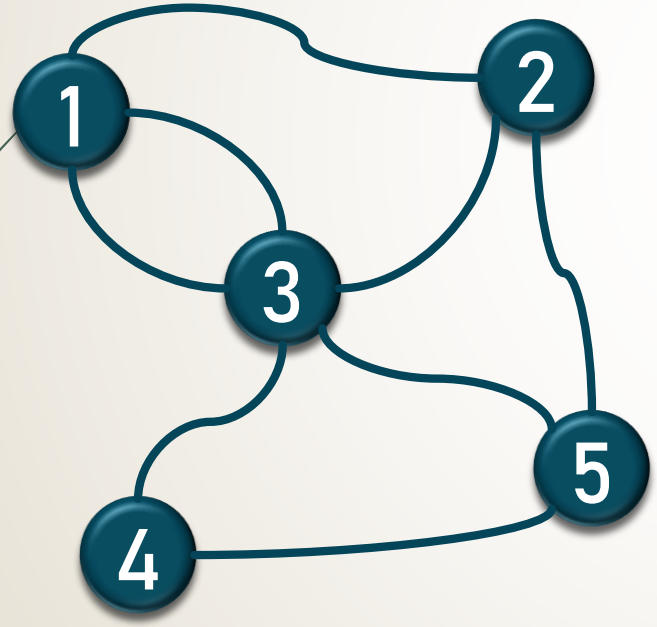


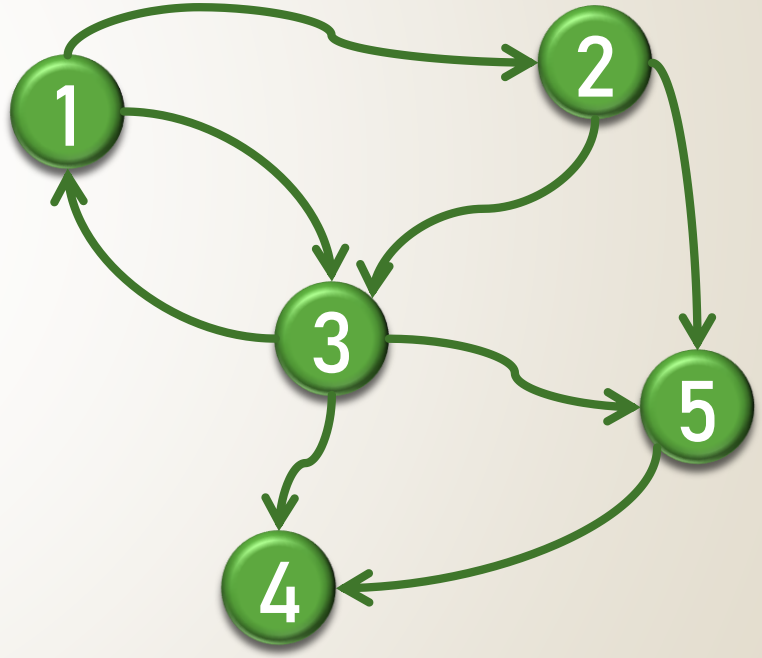
$V = \{v_1, v_2, v_3, v_4, v_5, v_6, v_7, v_8, v_9, v_{10}\}$

$A = \{ (v_1, v_2), (v_1, v_4), (v_1, v_5), (v_2, v_3), (v_2, v_7), (v_4, v_5), (v_4, v_6), (v_5, v_5), (v_5, v_6), (v_6, v_7) \}$

Grafos. Definición (2)

- Los grafos pueden clasificarse en: grafos no dirigidos y grafos dirigidos (digrafos)



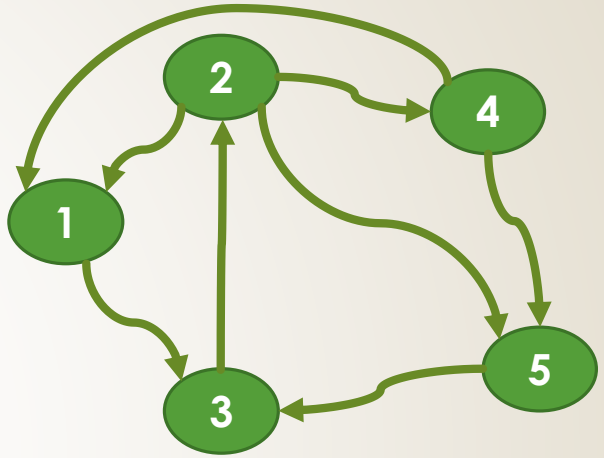


Si el arco tiene dirección, es decir el par (v_i, v_j) está ordenado, se dice que el grafo está dirigido. Los grafos dirigidos se denominan digrafos.

Grafos. Definición (3)

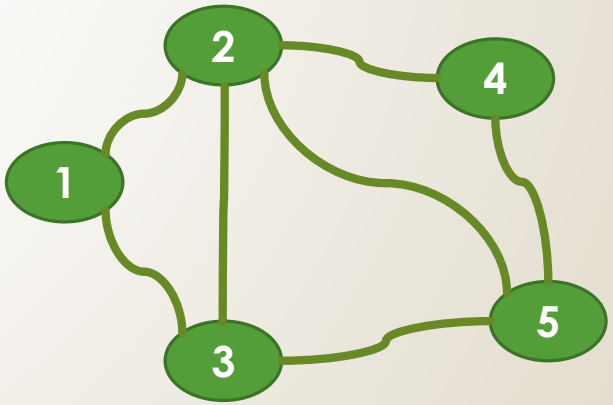
- Se dice que el vértice v_j es adyacente al vértice v_i si y sólo si (v_i, v_j) pertenece al conjunto A.

Ady (3) = 1 y 5



- En grafos no dirigidos, v_i es adyacente al vértice v_j y v_j es adyacente al vértice v_i si existe un arco que une estos nodos.

Ady (3) = 1, 2 y 5



Grafos. Definición (4)

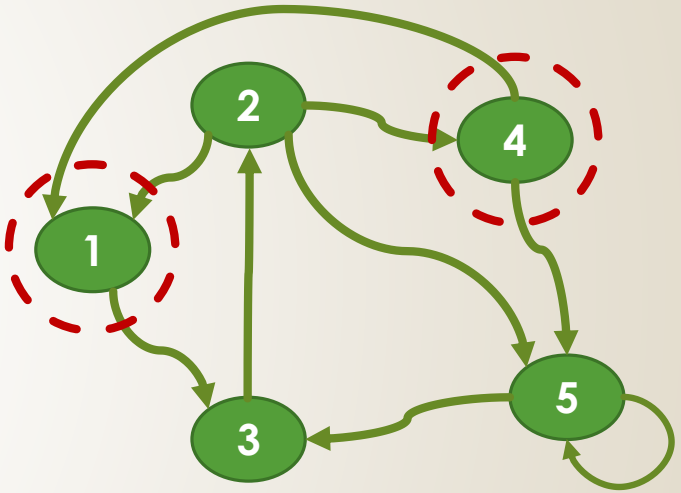
- Un camino es la secuencia de vértices que existe entre los nodos v_i y v_j.

  Ej: Camino de 1 a 4 = 1, 3, 2, 4

- La longitud de un camino es la cantidad de aristas que contiene.

  Ej: Longitud del camino de 1 a 4 = 3

- Un camino simple es aquel donde todos sus vértices son distintos, a excepción del primer y el último que pueden coincidir (ciclo).



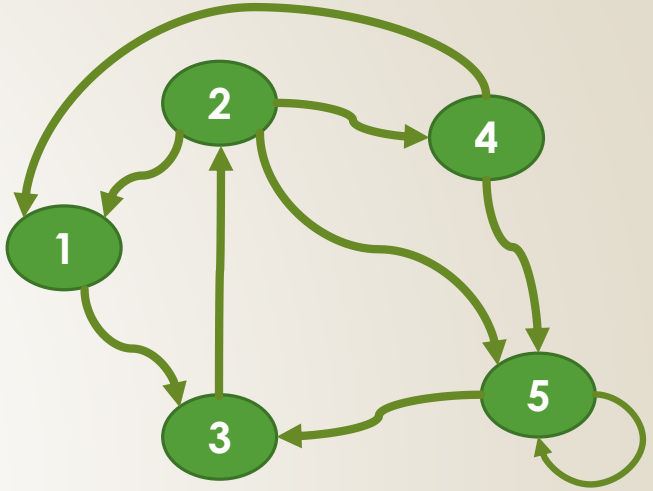
Grafos. Definición (5)

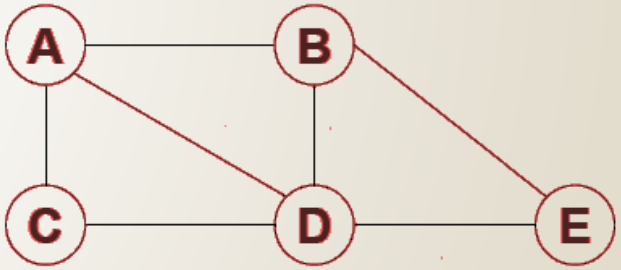
- Un grafo dirigido es <u>cíclico</u> si presenta un camino de longitud >= 1, donde el primer y último vértice son el mismo.

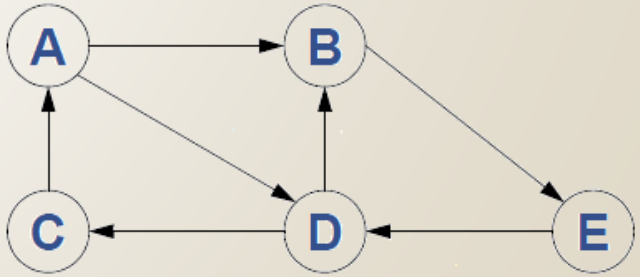
  Ciclo1 = 1, 3, 2, 4, 1

  Ciclo2: 5, 5

- Un grafo es <u>acíclico</u> si no presenta ciclos.
- Un grafo no dirigido está <u>conectado</u> si existe un camino que une cualquier par de nodos.
- Un grafo dirigido con esta propiedad se dice que está <u>fuertemente conectado</u>.

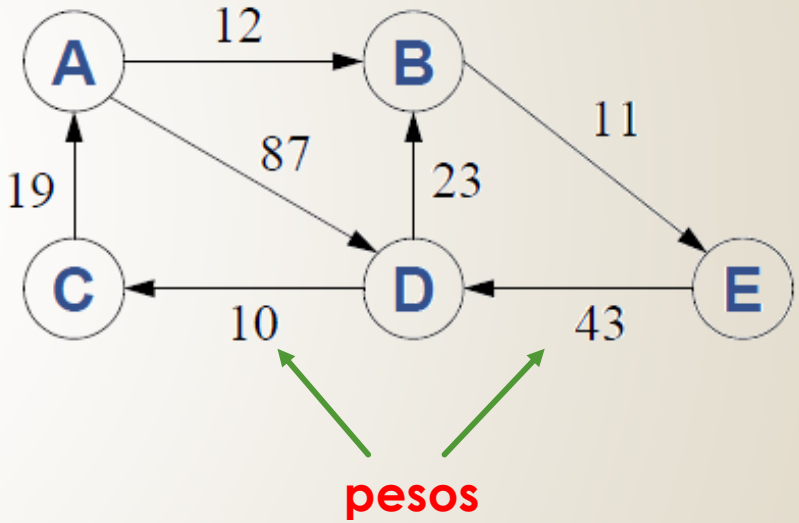






Grafos. Definición (6)

- Orden del grafo: número de nodos o vértices del grafo.
- Grafo nulo: un grafo nulo es aquel de orden 0.
- Peso de los arcos: en algunos situaciones es necesario asociar a los arcos un "peso" que puede representar el costo, distancia, tiempo, etc. que toma pasar de un nodo a otro.



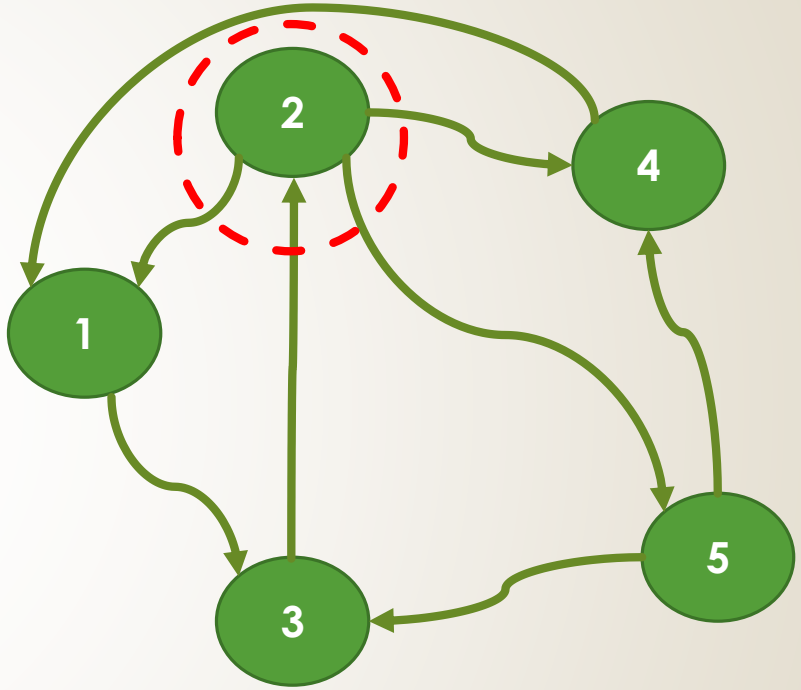
Grafos. Definición (7)

- Grado de entrada de un nodo (grafo dirigido): número de arcos que arriban al nodo (destino).

$$g(2)_e = 1$$

- Grado de salida de un nodo (grafo dirigido): número de arcos que parten del nodo (origen).

$$g(2)_s = 3$$



$$g(2) = g(2)_e + g(2)_s = 4$$

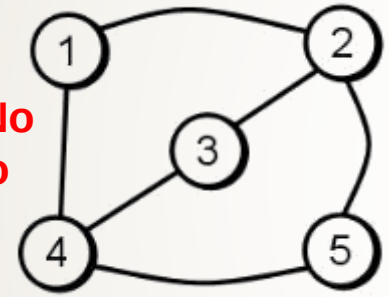
Grafos. Representación (1)

- Existen, básicamente, 2 formas de representación de grafos:
  - Matriz de Adyacencia
  - Lista de Adyacencia
    - ✓ Directorio de nodos
    - ✓ Listas enlazadas

Grafos. Representación (2)

- Matriz de Adyacencia: es una representación de las relaciones entre los nodos de un grafo. Cada celda de la matriz representa un arco que puede unir 2 vértices .

Grafo No dirigido

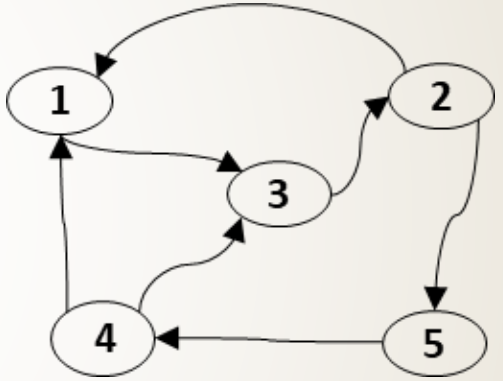


M	1	2	3	4	5
1	0	1	0	1	0
2	1	0	1	0	1
3	0	1	0	1	0
4	1	0	1	0	1
5	0	1	0	1	0

El valor 1 indica que existe un enlace entre el nodo 1 y el nodo 4

La matriz simétrica corresponde a un grafo no dirigido

Grafo Dirigido



M	1	2	3	4	5
1	0	0	1	0	0
2	1	0	0	0	1
3	0	1	0	0	0
4	1	0	1	0	0
5	0	0	0	1	0

El valor 1 indica que existe un enlace que va del nodo 3 al nodo 2

Grafos. Representación (3)

- Implementación de la Matriz de Adyacencia

Alternativa 1

```
typedef bool matady[NODOS][NODOS];
```

M	1	2	3	4	5
1	F	F	F	F	V
2	F	V	V	V	V
3	F	V	F	V	F
4	V	F	F	F	F
5	F	F	V	F	F

Alternativa 2

```
typedef int matady[NODOS][NODOS];
```

M	1	2	3	4	5
1	16	12		14	
2	1	18	1		
3	28				23
4		22	1	9	30
5			8		

Alternativa 3

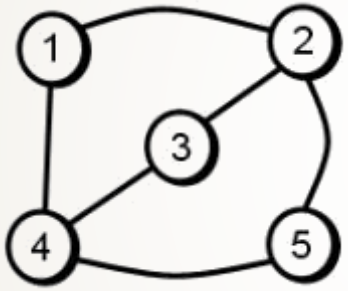
```
typedef struct arco{
        float peso;
        bool existe;
        };
typedef arco matady[NODOS][NODOS];
```

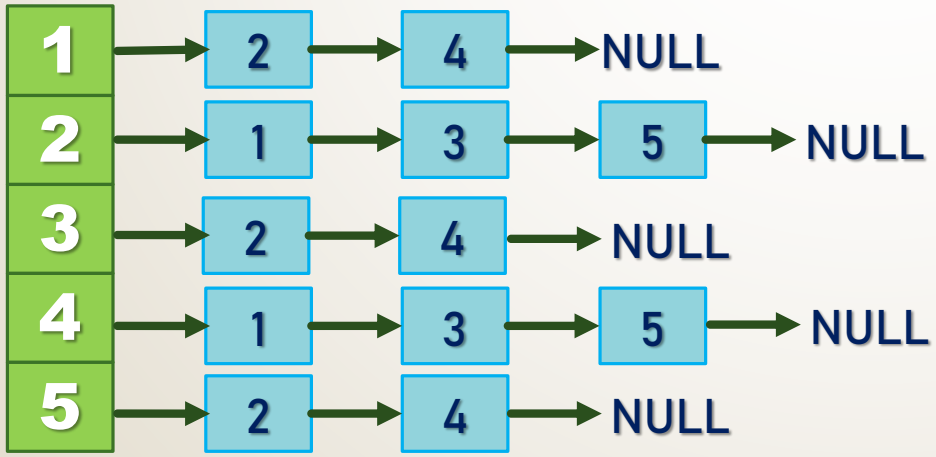
M	1	2	3	4	5
1		0 V		56 V	
2	100 V	22 V			-87 V
3		34 V			
4			-12 V		0 V
5	-56 V				

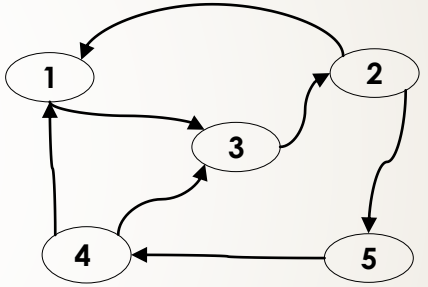
Grafos. Representación (4)

- Directorio de Nodos: representación de las relaciones entre los nodos de un grafo. Por cada nodo se mantiene una lista de los arcos que parten de él (con información acerca del nodo destino).

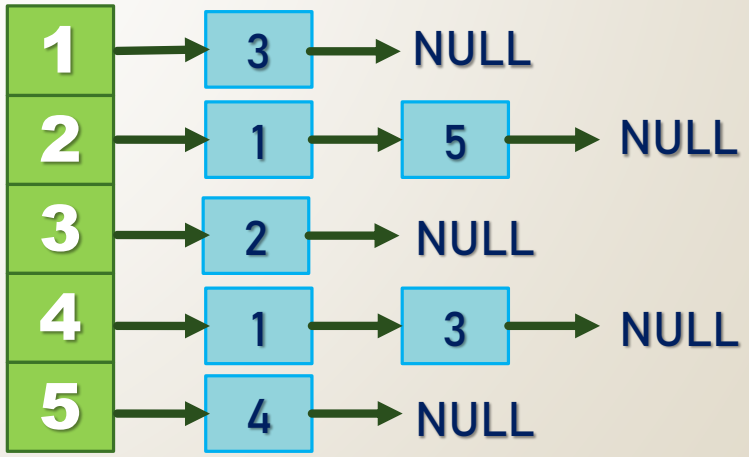
Grafo No dirigido







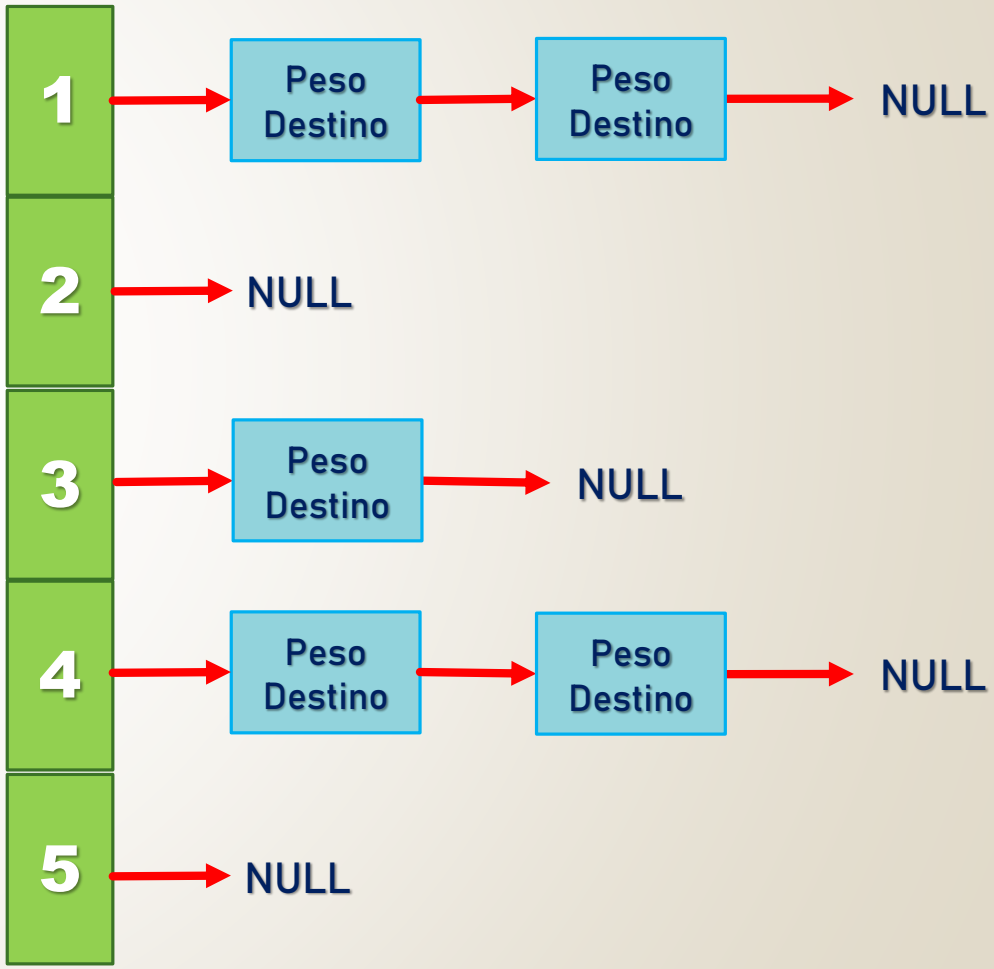
Grafo Dirigido



Grafos. Representación (5)

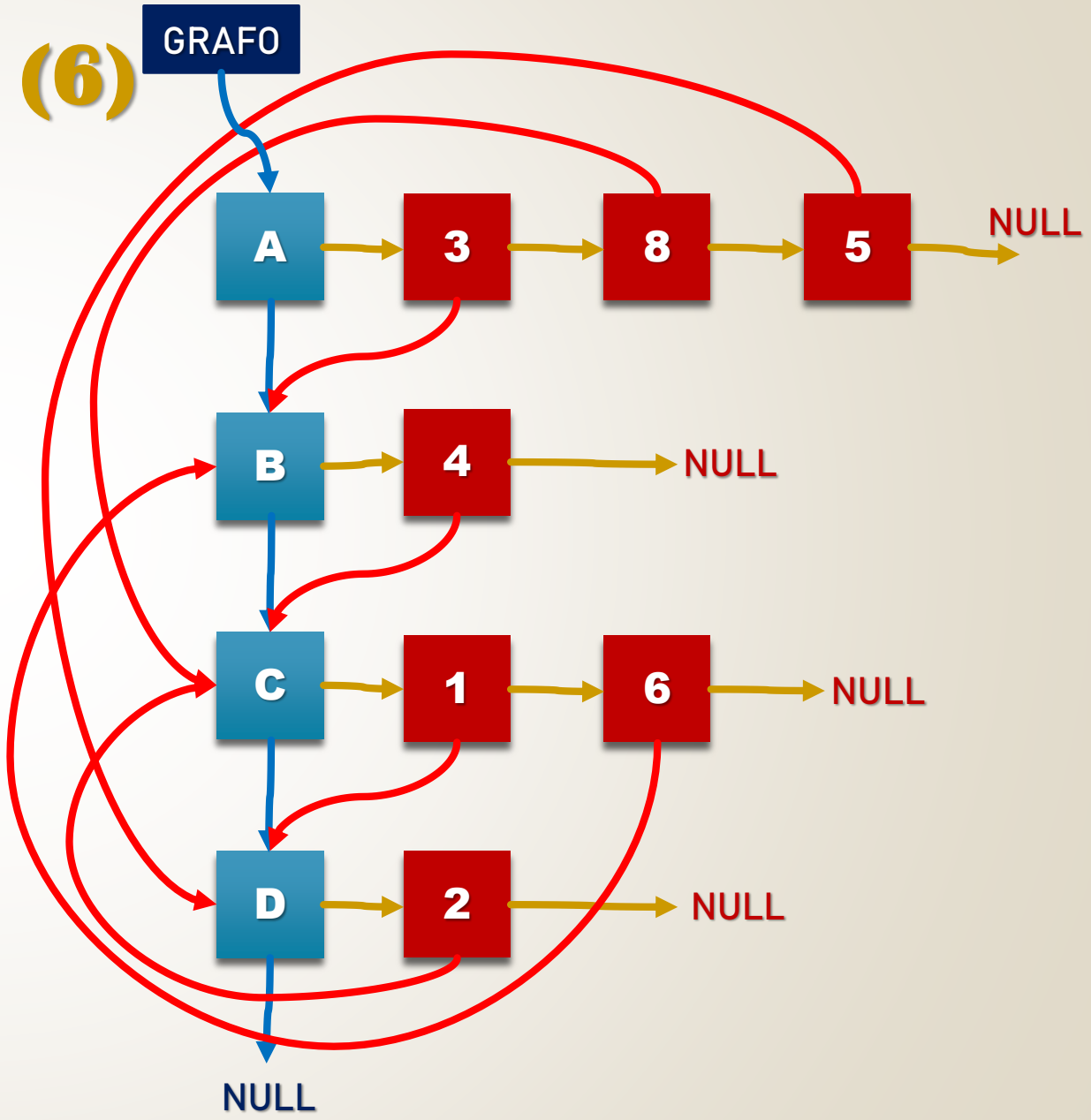
- Implementación del Directorio de Nodos

```
typedef struct arco *parco;
typedef struct arco{
            float peso;
            int destino;
            parco sigarco;
            };
typedef parco dirnodos[NODOS];
```



Grafos. Representación (6)

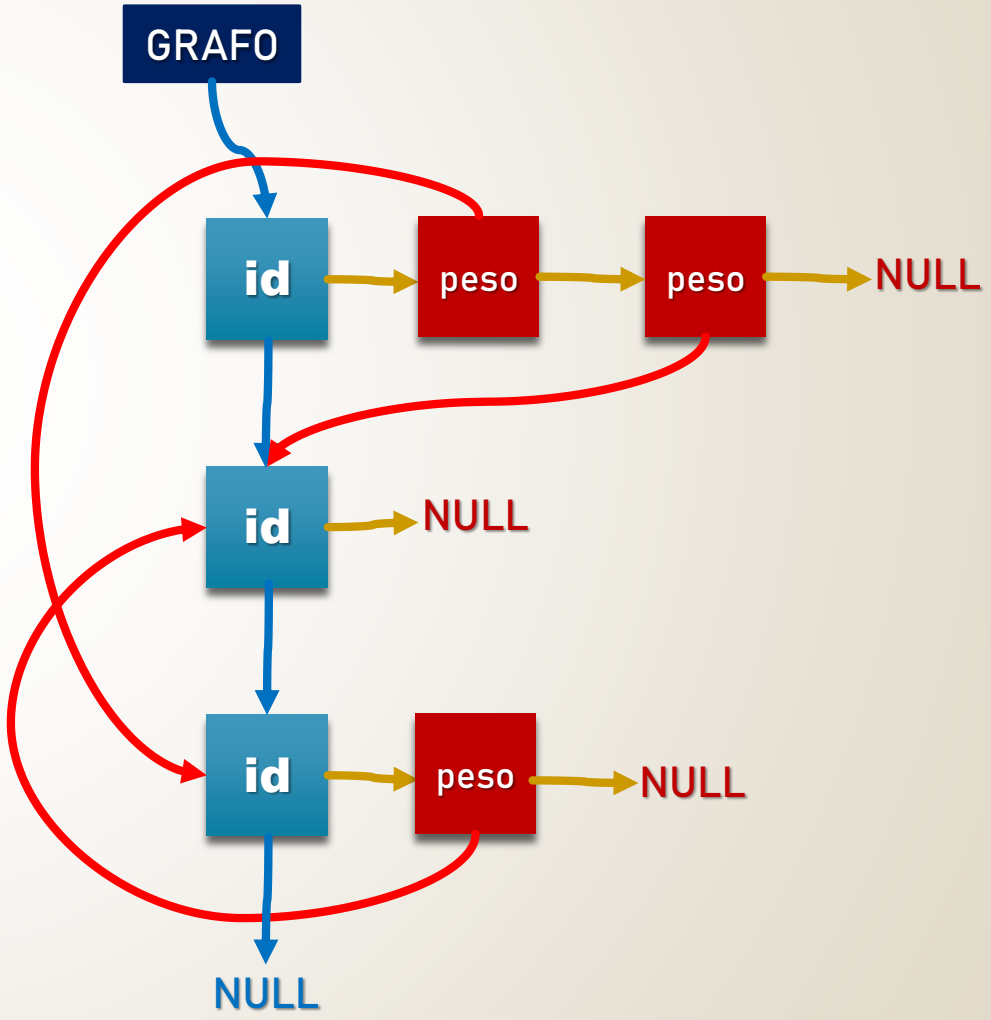
- Representación Enlazada: es una representación de las relaciones entre los nodos de un grafo.
- Se mantiene una lista de vértices, donde cada vértice tiene a su vez una lista de los arcos que parten de cada uno de ellos (con una referencia al nodo destino).



Grafos. Representación (7)

- Implementación de la Representación Enlazada

```
typedef struct arco *parco;
typedef struct vertice *pvertice;
typedef struct arco{
            float peso;
            pvertice destino;
            parco sig_arco;
            };
typedef struct vertice{
            int id;
            pvertice sig_vertice;
            parco lista_arco;
            };
```



Grafos. Operaciones (1)

- Inicialización (matriz de adyacencia)

```
void inicia_grafo (matady &grafo)
{  int i,j;
   for(i=0;i<NODOS;i++)
      for(j=0;j<NODOS;j++)
         grafo[i][j]=false;
}
```

Inicialización de la matriz de adyacencia

```
typedef bool matady[NODOS][NODOS];
```

G	1	2	3	4	5
1	F	F	F	F	F
2	F	F	F	F	F
3	F	F	F	F	F
4	F	F	F	F	F
5	F	F	F	F	F

Grafo

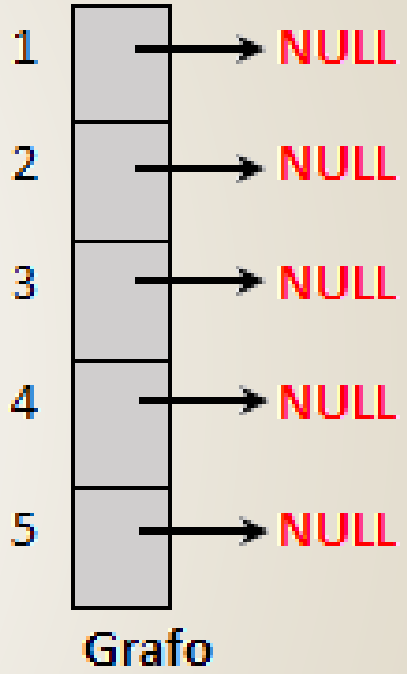
```
grafo[i][j].existe=false;
```

Grafos. Operaciones (2)

- Inicialización (directorio de nodos)

```
void inicia_grafo2 (dirnodos &grafo)
{ int i;
  for(i=0;i<NODOS;i++)
     grafo[i]=NULL
}
```

Inicialización del directorio de nodos




```
grafo[i].lista_arcos=NULL;
```

Grafos. Operaciones (3)

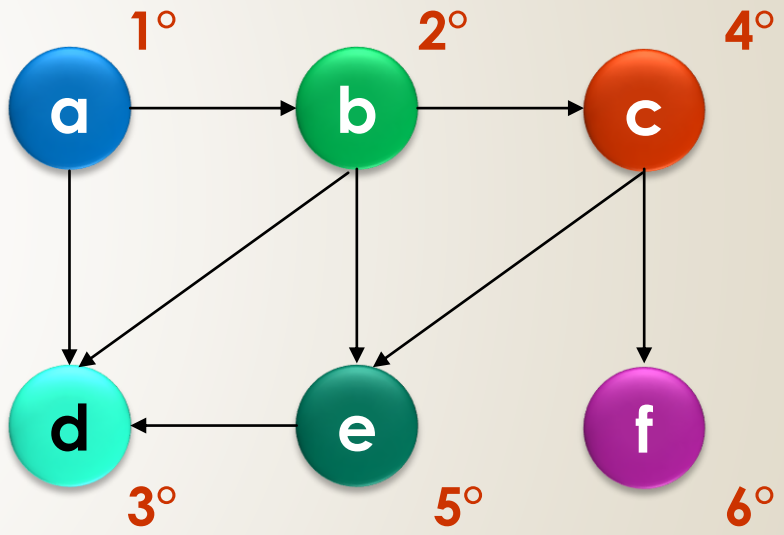
- Inicialización (listas enlazadas)

```
void inicia_grafo3 (pvertice &grafo)
{
    grafo=NULL
}
```

Inicialización de la representación enlazada

Grafos. Operaciones (3)

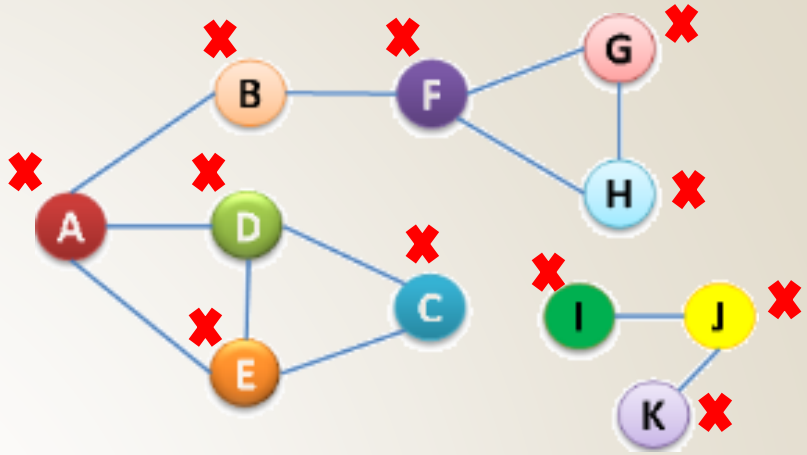
- Recorrido
  - En amplitud: se recorre el grafo a partir de un nodo dado, visitando primero los que están a un arco de distancia y luego los que están a dos arcos de distancia, y así sucesivamente hasta alcanzar todos los nodos accesibles desde el dado.



Grafos. Operaciones (4)

- Recorrido en Amplitud

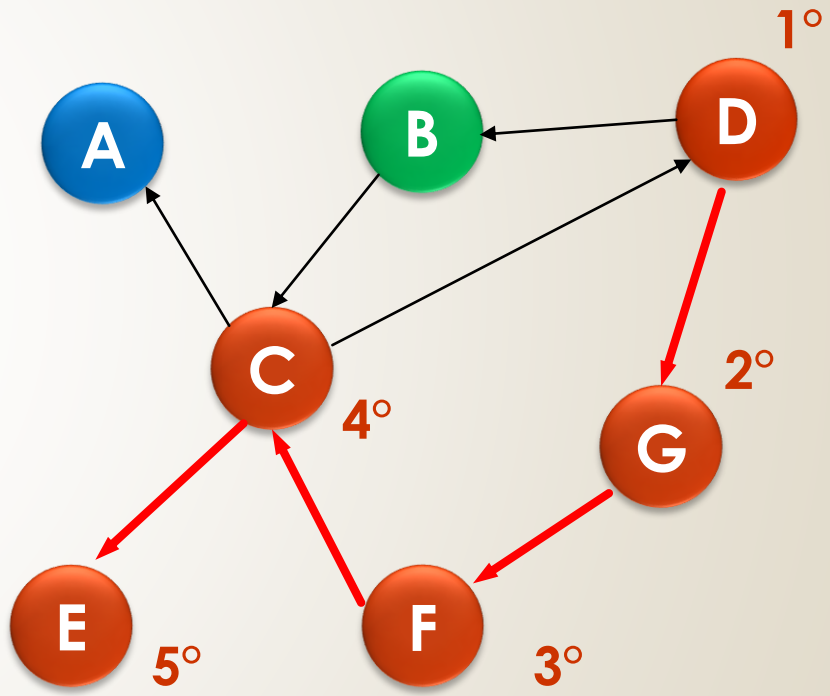
```
PROCEDIMIENTO amplitud(E g:tgrafo)
VARIABLES
   q:tcola
   v,w,z:tvertice
INICIO
   iniciar_cola(q)
   MIENTRAS visitados(g)=F HACER
     v←no_visitado(g)
     marcar(v)
     agregar_cola(q,v)
     MIENTRAS cola_vacia(q)=F HACER
       w←quitar_cola(q)
       procesar(w)
       MIENTRAS ady_no_visitados(w)=V HACER
         z←adyacente_no_visitado(w)
         agregar_cola(q,z)
         marcar(z)
       FIN_MIENTRAS
     FIN_MIENTRAS
   FIN_MIENTRAS
FIN
```



Recorrido del Grafo: A B D E F C G H I J K

Grafos. Operaciones (5)

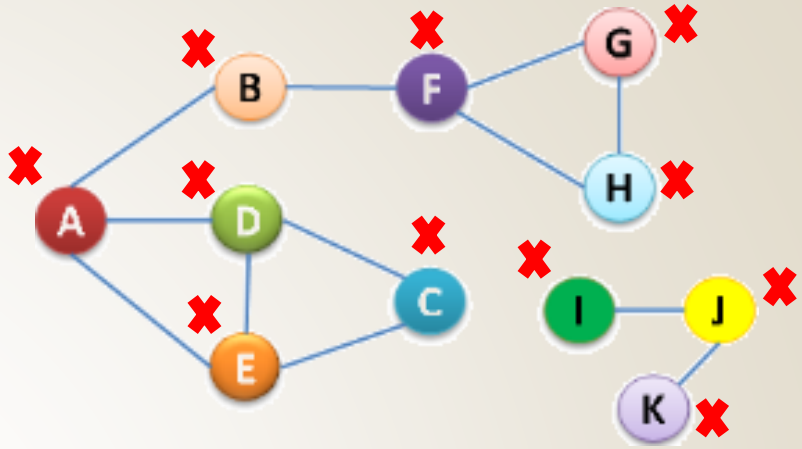
- Recorrido
  - En profundidad: recorre los caminos que parten desde un nodo dado hasta que ya no sea posible avanzar más. Cuando esto ocurre, se vuelve atrás para seguir los caminos alternativos no estudiados previamente

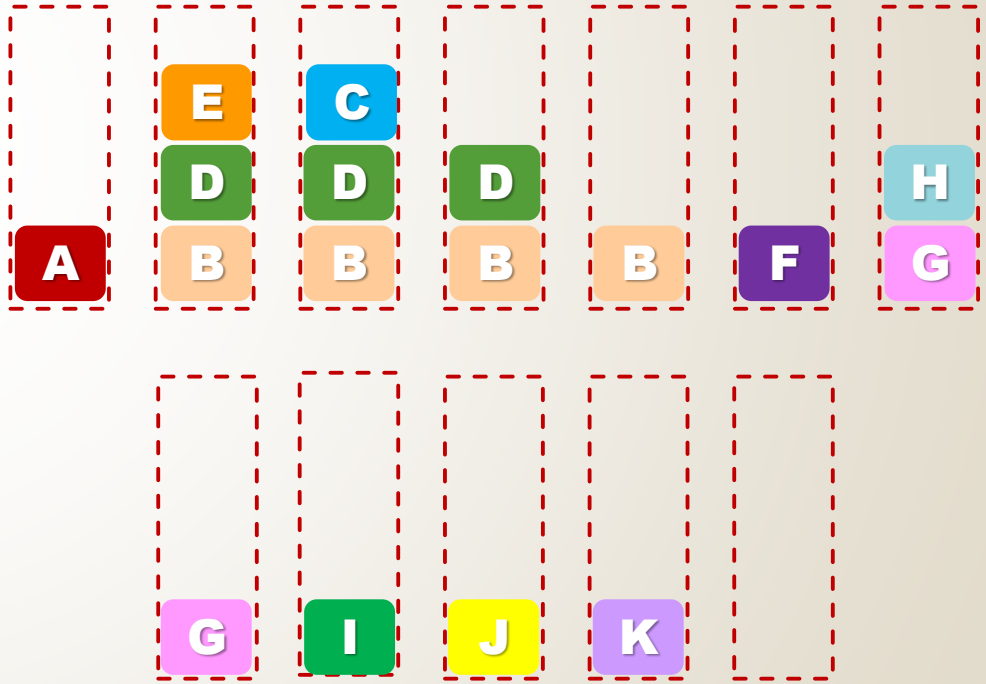


Grafos. Operaciones (6)

- Recorrido en Profundidad

```
PROCEDIMIENTO profundidad (E g:tgrafo)
VARIABLES
   p:tpila
   v,w,z:tvertice
INICIO
   iniciar_pila(p)
   MIENTRAS visitados(g)=F HACER
     v←no_visitado(g)
     marcar(v)
     agregar_pila(p,v)
     MIENTRAS pila_vacia(p)=F HACER
       w←quitar_pila(p)
       procesar(w)
       MIENTRAS ady_no_visitados(w)=V HACER
         z←adyacente_no_visitado(w)
         agregar_pila(p,z)
         marcar(z)
       FIN_MIENTRAS
     FIN_MIENTRAS
   FIN_MIENTRAS
FIN
```





Recorrido del Grafo: A E C D B F H G I J K

Ejemplo (1)

- Una empresa de mudanzas presta servicios entre 20 ciudades de todo el país. Para administrar los destinos y costos de transporte, el gerente del negocio requiere almacenar la siguiente información:
  - Ciudades: nombre ciudad, localidad, provincia, código postal.
  - Rutas de transporte: distancia, tipo de camino.

  De acuerdo a esto: a) defina la estructura de datos que represente la situación planteada y b) desarrolle la operación de inicialización de la estructura.

Ejemplo (2)

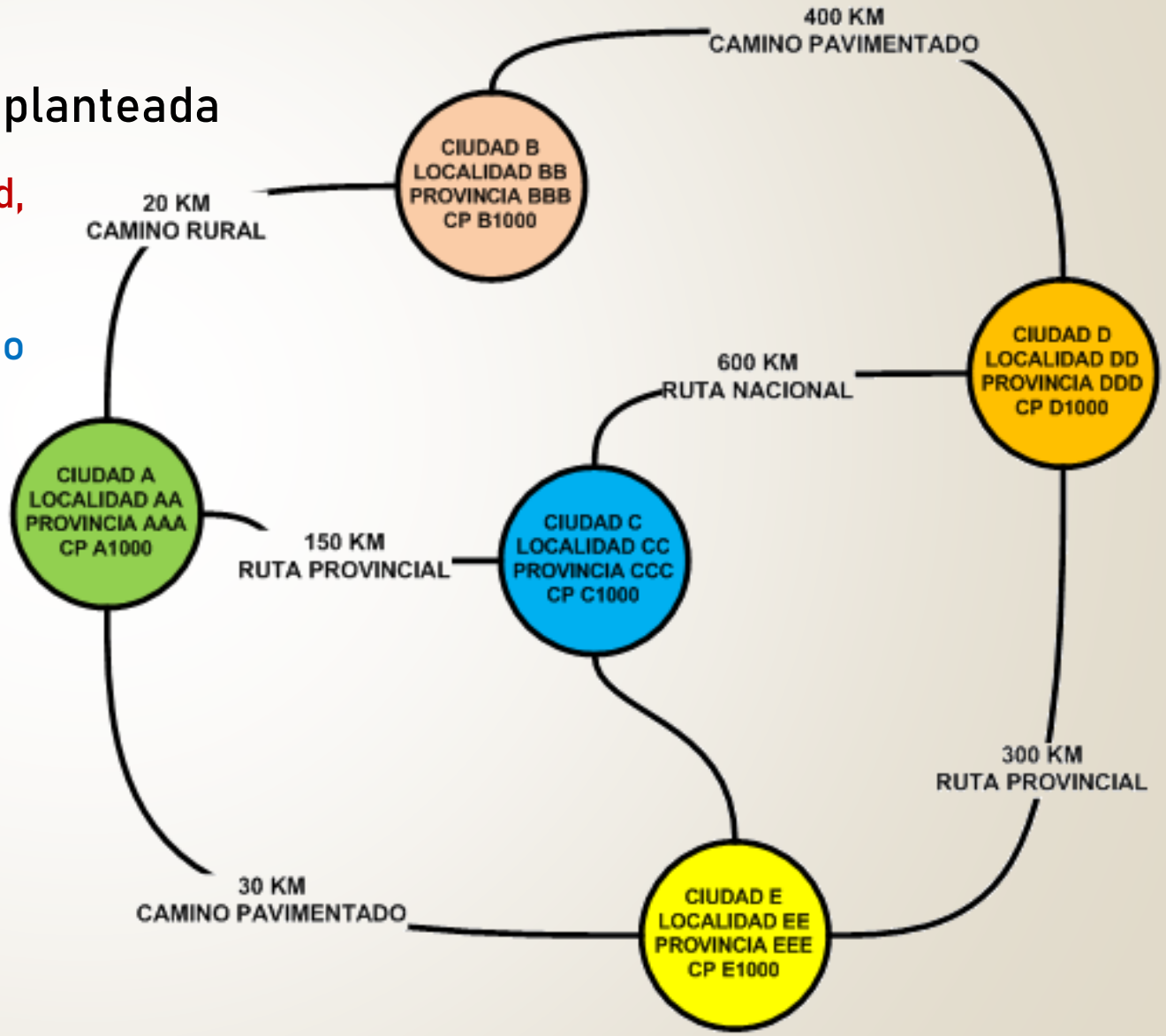
- En el problema se identifican 2 elementos principales
  - Ciudades
  - Rutas de transporte

Las ciudades son los objetos entre los que se establecen relaciones, por tanto, se representarán mediante los vértices del grafo. La información de las ciudades se almacenará en los vértices.

Las rutas de transporte permiten vincular las ciudades y por tanto, se representarán mediante los arcos o aristas del grafo. La información de las rutas se almacenará en los arcos.

Ejemplo (3)

- Grafo que representa la situación planteada
  - Ciudades: nombre ciudad, localidad, provincia, código postal.
  - Rutas de transporte: distancia, tipo de camino.



Ejemplo (4)

ALTERNATIVA 1

- Definición de la estructura usando directorio de nodos

```
typedef struct ruta *pruta;
typedef struct ruta {
  int ciudad_destino;
  float distancia;
  tcadena tipo;
  pruta ruta_sig;
};
typedef struct ciudad {
  tcadena nombre;
  tcadena localidad;
  tcadena provincia;
  pruta lista_rutas;
};
typedef ciudad g_ciudades[20];
```

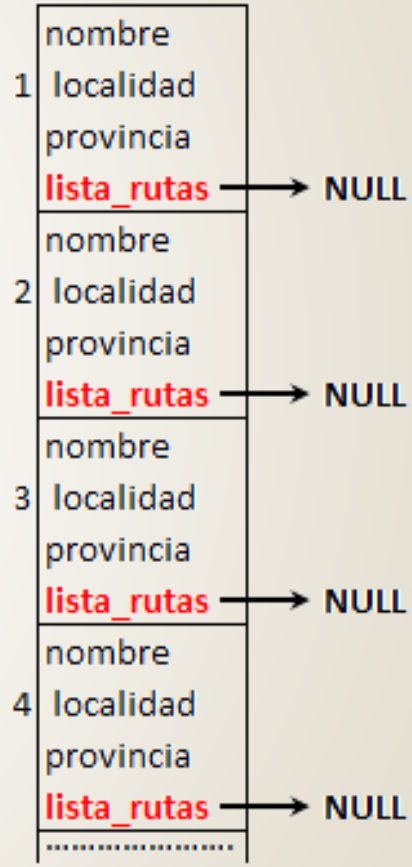
Las rutas se representan mediante las aristas del grafo

Las ciudades se representan mediante los vértices del grafo

Ejemplo (5)

- Operación de Inicialización

```
void inicia_ciudad(g_ciudades &city)
{
   int i;
   for(i=0;i<20;i++)
      city[i].lista_rutas=NULL;
}
```



Ejemplo (6)

ALTERNATIVA 2

- Definición de la estructura usando listas enlazadas

```
typedef struct ruta *pruta;
typedef struct ciudad *pciudad;
typedef struct ruta {
  pciudad ciudad_destino;
  float distancia;
  tcadena tipo;
  pruta ruta_sig; };
typedef struct ciudad {
  tcadena nombre;
  tcadena localidad;
  tcadena provincia;
  pciudad ciudad_sig;
  pruta lista_rutas;};
```

Las rutas se representan mediante las aristas del grafo

Las ciudades se representan mediante los vértices del grafo

Ejemplo (7)

- Operación de Inicialización

```
void inicia_ciudad(pciudad &gcity)
{
    gcity=NULL;
}
```

Ejemplo (8)

- Un sistema de telecomunicaciones móviles gestiona las llamadas entre teléfonos celulares. Por cada teléfono el sistema registra N° de celular, marca, modelo, nombre del usuario, empresa (personal, claro, movistar) y tipo de red (2G, 3G, 4G). Cada vez que se realiza 1 llamada se registra hora de inicio (hh, mm, ss), hora de finalización (hh, mm, ss) y tipo/costo (tipo, costo minuto local, costo minuto larga distancia).

Bibliografía

- Hernández, Roberto *et al*. Estructuras de Datos y Algoritmos. Prentice Hall. 2001.
- Joyanes Aguilar *et al*. Estructuras de Datos en C++. Mc Graw Hill. 2007.
- De Giusti, Armando *et al.* Algoritmos, datos y programas, conceptos básicos. Editorial Exacta. 1998.
- Joyanes Aguilar, Luis. Fundamentos de Programación. Mc Graw Hill. 1996.