GRAFOS Y SUS REPRESENTACIONES

Definición:

a) Es una estructura de datos formada por dos conjuntos: 1° Conjunto finito de datos llamados ***vértices*** y 2°) Conjunto finito de *pares ordenados* llamados **aristas/relaciones/arcos** de la forma (f, d). El atributo de *ordenado* especifica que el par (f, d) ***no*** equivale al par (d, f) → ***Grafo dirigido o dígrafo***.

b) Conjunto de datos(*vértices*) multi relacionados (*aristas/relaciones/arcos*).

Aplicaciones

Los grafos son usados generalmente para representar muchas situaciones de la vida real:

Por ejemplo:

Rutas de transporte (Terrestre, aérea, marítima, etc.)
Líneas de distribución (agua, electricidad, gas, drenaje, etc.)
Redes (Telefonía, tv, etc.)
Redes de streaming (audio, video, tv, etc.)
Redes de computadoras

Representaciones:

a) Formal / matemática

G = (V, A)

Donde:

$V = \{v_1, v_2, v_3, \ldots, v_n\}$

y

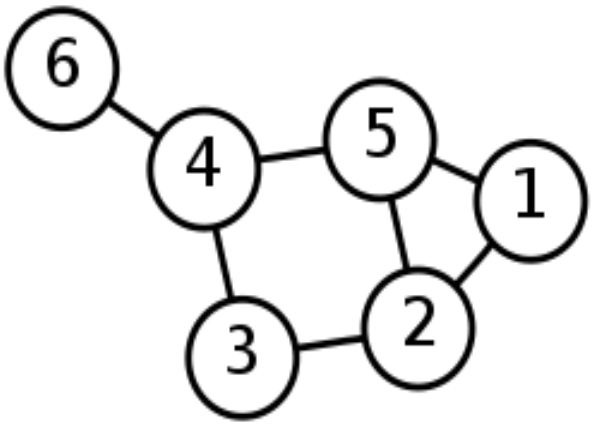
$A = \{a_{1,1}, a_{1,2}, a_{1,3}, \ldots, a_{1,n}, a_{2,1}, a_{2,2}, a_{2,3}, \ldots, a_{2,n},$
$a_{3,1}, a_{3,2}, a_{3,3}, \ldots, a_{3,n}, \ldots a_{n,1}, a_{n,2}, a_{n,3}, \ldots, a_{n,n}\}$

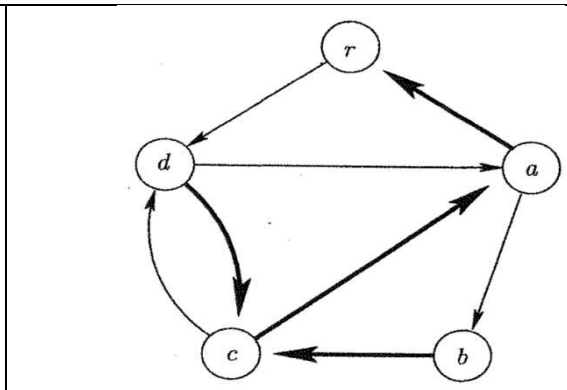
Generalizando:

Costo/etiqueta/Color

$a_{i,j} = (v_i, v_j [, c_{i,j}])$ y no necesariamente: v_i v_j

b) Gráfica





c) Matriz de relación/adyacencia

VERTICES
nmax=5
cv = 5

0	A
1	B
2	C
3	D
4	E

ARISTAS/ARCOS/RELACIONES

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

**Estructuras

```
#define NMAX 20

typedef struct
{
  int nmax;          //15
  int cv;            //3
  int vecVer[NMAX];
  int matRel[NMAX][NMAX];
}  GRAFO_MR;
```

**Operaciones básicas

*** Inicialización ➔ Estado de vacío, “Limpiarla”

GRAFO_MR grafo;

```
int iniGrafoMR(GRAFO_MR *g, int nm)
{
  int res=0;

  if(nm <= NMAX)
  {
    g→nmax = nm;
    g→cv = 0;
    res=1;
  }
   return(res);
}
```

*** inserción de vértices

```
int insVerMR(GRAFO_MR *g, int v)
{
  int res=0, i;

  if(g→cv < g→nmax)
 {
    for(i=0; i < g→cv && v != g→vecVer[i]; i++);
    if(i == g→cv)   //Fuera del ciclo:  ¿No está?
    {
       g→vecVer[g→cv] = v;
       for(i=0; i<=g→cv; i++)
             g→matRel[g→cv][i] = g→matRel[i][g→cv] = 0;
       g→cv++;
       res=1;
     }
 }
 return(res);
}
```

*** Capturar vértices

```
void capturaVerMR(GRAFO_MR *g)
{
  int  res, dato;
  char resp;

  do {
        printf("Dame el vértice: ");
        scanf("%d", &dato);
        res = insVerMR(g, dato);
        if(res == 1)
        {
          printf("Dato ingresado, ¿Otro(s/n)? ");
          scanf("%c", &resp);
         }
         else
              printf("Grafo lleno\n");
      } while(res == 1 && resp == 's');
}
```

** Insertar relación MR

```
int insRelMR(GRAFO_MR g, int vf, int vd)
{
   int res=0, ren, col;

  for(ren=0; ren<g.cv && vf != g.vecVer[ren]; ren++);
  if( ren < g.cv)   // ¿Está?
  {
    for(col=0; col<g.cv && vd != g.vecVer[col]; col++);
    if(col < g.cv)
        res = g.matRel[ren][col] = 1;
   }
   return(res);
}
```

** Capturar relaciones

```
int capturaRelsMR(GRAFO_MR g)
{
   int res,  vo, vd;
   char resp;

   do {
         printf("Dime el vértice origen: ");
         scanf("%d", &vo);
         printf("Dime el vértice destino: ");
         scanf(""%d", &vd);
         res = insRelMR(g, vo, vd);
         if(res == 1)
            printf("Relación formalizada\n");
         else
            printf("No formalizada\n");

        printf("Otra ?");
        scanf("%c", &resp);
        } while( resp=='s');
    return(res);
}
```

**Recorrido de un grafo MR ➔ Acceso a los datos*

*** Escribir una función para mostrar la información del grafo.

A: B D G
B: A C Z T
C: A B D F
. . .

```
void muestraGrafoMR(GRAFO_MR g)
{
 int i, j;

 for(i=0; i<g.cv; i++)      // Recorrido de vértices
 {
   printf("%d: ", g.vecVer[i]);  //Recorrido de relaciones
   for(j=0; j<g.cv; j++)
     if(g.matRel[i][j] == 1)
         printf("%d  ", g.vecVer[j]);
   printf("\n");
  } // Siguiente vértice
 }
```

*** Escribir una función para mostrar los vértices de un grafo con su No. de relaciones.

```
void muestraVerNoRels(GRAFO_MR g)
{
 int i, j, cont;

 for(i=0; i<g.cv; i++)
 {
  cont=0;
  for(j=0; j<g.cv; j++)
     if(g.matRel[i][j] == 1)      // Estás dos líneas se pueden simplificar en una
        cont++;                   // cont += g.matRel[i][j];

  printf("%d tiene %d relaciones\n", g.vecVer[i], cont);
  } // Siguiente vértice
 }
```

*** Escribir una función para encontrar/entregar el vértice más "cool" (más relacionado) ➔ contar relaciones + Algoritmo del mayor.

```
int encontrarVerMasRel(GRAFO_MR g)
{
 int i, j, cont, sMayor=0, masRel;

 for(i=0; i<g.cv; i++)
 {
    cont=0;
    for(j=0; j<g.cv; j++)
       cont += g.matRel[i][j];

    if(cont > sMayor)  //Algoritmo del mayor
    {
      sMayor = cont;
      masRel = g.vecVer[i];
     }
  } // Siguiente vértice
  return(masRel);
 }
```

*** Determinar si hay relación(unidireccional, digrafo) entre 2 vértices. (determinar => 1, 0)

```
int hayRelMR(GRAFO_MR g, int vf, int vd)
{
  int res=0, ren, col;

  for(ren=0; ren<g.cv && vf != g.vecVer[ren]; ren++);
  if(ren < g.cv)
  {
    for(col=0; col<g.cv && vd != g.vecVer[col]; col++);
    if(col<g.cv && g.matRel[ren][col] == 1)
         res=1;
   }
   return(res);
  }
```

*** Determinar si hay relación(bidireccional) entre 2 vértices. (determinar => 1, 0)

```
int hayRelBidirMR(GRAFO_MR g, int vf, int vd)
{
  int res=0, ren, col;

  for(ren=0; ren<g.cv && vf != g.vecVer[ren]; ren++);
  if(ren < g.cv)
  {
    for(col=0; col<g.cv && vd != g.vecVer[col]; col++);
    if(col<g.cv && g.matRel[ren][col] == 1 && g.matRel[col][ren] == 1)
         res=1;
   }
   return(res);
  }
```

*** Determinar si 3 vértices forman un camino, es decir *v1→ v2 → v3*

```
int hayCamino(GRAFO_MR g, int v1, int v2, int v3)
{
  int res=0, pos1, pos2, pos3;

  for(pos1=0; pos1<g.cv && v1 != g.vecVer[pos1]; pos1++);
  if(pos1 < g.cv)
  {
    for(pos2=0; pos2<g.cv && v2 != g.vecVer[pos2]; pos2++);
    if(pos2<g.cv  && g.matRel[pos1][pos2] == 1)
    {
       for(pos3==; pos3<g.cv && v3 != g.vecVer[pos3]; pos3++);
       if(pos3<g.cv && g.matRel[pos2][pos3] == 1)
          res=1;
     }
    }
    return(res);
  }
```

*** *Eliminación*

** Elimina relación

```
int elimRelMR(GRAFO_MR g, int vf, int vd)
{
  int res=0, ren, col;

  for(ren=0; ren<g.cv && vf != g.vecVer[ren]; ren++);
  if(ren<g.cv)
  {
    for(col=0; col<g.cv && vd != g.vecVer[col]; col++);
    if(col<g.cv)
   {
     g.matRel[ren][col] = 0;
     res=1;
   }
  }
  return(res);
 }
```

**Eliminación de vértice