Teoría de grafos.

Modelos matemáticos : El objetivo es obtener información que sea útil para la toma de decisión. Si una información llega tarde, **no sirve.** Para eso se necesita la velocidad que ofrece el computador. Entre el modelo de la realidad y el modelo computacional, existe una interfaz que debe ser solucionada por medio de un modelo matemático.

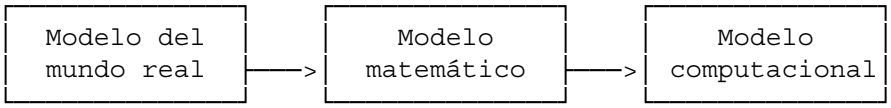


Figura 2.4.

Grafo : Dado un conjunto de puntos P y un conjunto de relaciones E, la representación gráfica es lo que comunmente denominamos **grafo G(P, E).**

Ejemplo :Sea P = { x, y, z }
E = { (x, y) ; (x, z) ; (z, z) }

El grafo **G(P, E)** es

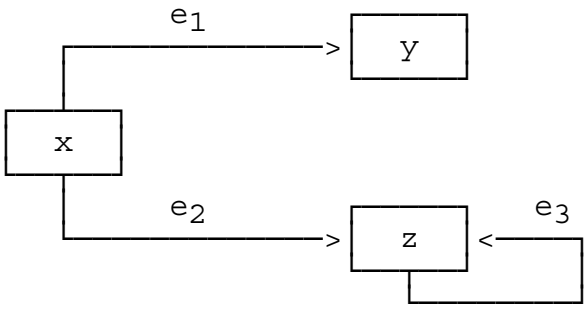


Figura 2.5.

Donde x, y, z son los **nodos** del grafo y e_1, e_2 y e_3 son los **arcos** del grafo

Los grafos sirven para modelizar matemáticamente una estructura de datos. De este modelo, se pueden derivar resultados matemáticos teóricos.

Subgrafo parcial : El grafo G'(P', E') es un subgrafo parcial de G(P, E) si P' = P .and. E' está incluido en E

Subgrafo : El grafo G'(P', E') es un subgrafo G(P, E)

si P' esta incluido en P .and. E' está incluido en E

Implementación de un grafo en memoria

En el siguiente ejemplo se puede ver como se implementaría un grafo en memoria volátil, por medio de celdas y links de memoria o en memoria permanente de un computador por medio de archivos relativos o indexados.

El objetivo de implementaciones en memoria volátil (RAM) es didáctico y permite aprender el manejo de punteros en memoria. A tal efecto, es importante el significado del valor **null** o **nil** simbolizado con el caracter **(^)**, utilizado en Pascal o C. Un campo que posee este valor, es equivalente a decir que ese campo es de dirección **(tipo pointer)** y que identifica la ausencia de dirección de memoria.

Ejemplo : Sea el grafo de la Figura 2.5., un grafo que representa un organigrama. De esta manera podemos deducir que **x supervisa a z** y también a **y** . Derivado del modelo, z se supervisa a si mismo, lo cual no tiene sentido y por lo tanto se descarta esta relación. Si se desea implementar este grafo, por medio de archivos relativos, entonces el conjunto P sería el archivo maestro de legajos y el conjunto E, el archivo estructura de empleados.

Maestro de empleados

NRR	Nombre	Legajo
001	x	A00001
002	y	B23514
003	z	B25663

Estructura de empleados

NRR	From	To
001	001	002
002	001	003

Si el maestro de empleados fuera indexado con el campo **Legajo** como clave, el archivo de estructura de empleados quedaría de la siguiente manera :

Estructura de empleados

From	To
A00001	B23514
A00001	B25663

Si la implementación se hiciera en memoria RAM, debería diseñarse un registro con un campo para guardar el identificador del nodo y tantos campos para direcciones de memoria como arcos salgan del nodo.

Complejidad espacial y computacional.

A lo largo del desarrollo de la materia, se estudiarán distintos tipos de algoritmos, muchas veces destinados a resolver un mismo problema. Al aplicar un algoritmo para la resolución de un problema, se busca que el mismo sea óptimo en términos de recursos utilizados.

Un ejemplo es la persona que utiliza un cajero automático. El objetivo del cajero automático es minimizar el tiempo que el usuario pierde en las colas del banco. Suponiendo que el programa que permite la extracción de dinero, o pagar una cuenta lo hiciera sin ningún problema pero que la operación delante de la pantalla se extendiera por 30 minutos, el cajero automático no tendría ningún sentido

pues perdería de vista el principal objetivo que es la agilidad del servicio.

En los sistemas de datos se tiende a medir los algoritmos en términos de tiempo de procesador utilizado ***(complejidad computacional)*** y en espacio *(en disco o memoria)* utilizado ***(complejidad espacial)***. De esta manera un algoritmo será mejor que otro si utiliza menos recursos ya sea en tiempo de procesador utilizado o en memoria.

Conceptos de Teoría de grafos : Dado un grafo G(P, E) donde

P = { x, y, z, d }

E = { (x, x) ; (x, y) ; (x, z) ; (y; d) ; (d, d)}

El grafo **G(P, E)** es

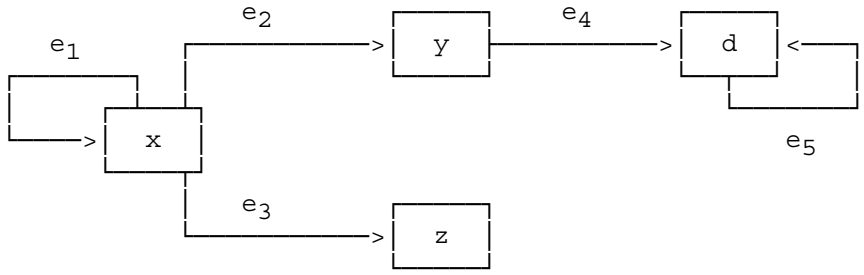


Figura 2.6.

Loops : Son las relaciones de la forma (x, x)

Minimal : **a** pertenece al conjunto minimal de **G** si se cumple

a ∈ P .and. b ∈ P .and. (b, a) ∈ E => a = b

En el ejemplo de la Figura 2.6. :

(x , y) ∈ E .and. x <> y => y no pertenece al conjunto minimal

(y, d) ∈ E .and. y <> d => d no pertenece al conjunto minimal

(x, z) ∈ E .and. x <> z => z no pertenece al conjunto minimal

(x, x) ∈ E .and. x = x => x puede pertenecer al conjunto minimal.

No existe ninguna otra relación en donde x sea 2da. componente, por tanto **x es un minimal** del grafo G.

Maximal : **a** pertenece al conjunto maximal de **G** si se cumple

a ∈ P .and. b ∈ P .and. (a, b) ∈ E => a = b

En el ejemplo :(x, y) ∈ E .and. x <> y => x no pertenece al conjunto maximal

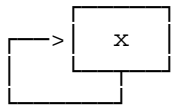
(y, d) ∈ E .and. y <> d => y no pertenece al conjunto maximal

(d, d) ∈ E .and. d = d => d puede pertenecer al conjunto minimal

No existe ninguna otra relación en donde **d** sea 1er. componente, por tanto **d es un maximal** del grafo G.

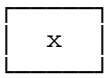
No existe ninguna relación en donde **z** sea 1er. componente, por tanto **z es un maximal** del grafo G. Otros ejemplos de minimales y maximales :

1)



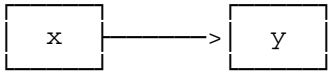
x es minimal y maximal

2)



x es minimal y maximal

3)



x es minimal e **y** es maximal

Left : El left de un nodo **a** es el conjunto de nodos que son primera componente de una relación donde a es la segunda componente. La definición algebraica es : **L(a) = { b ∈ P / (b, a) ∈ E }. Ejemplo** :

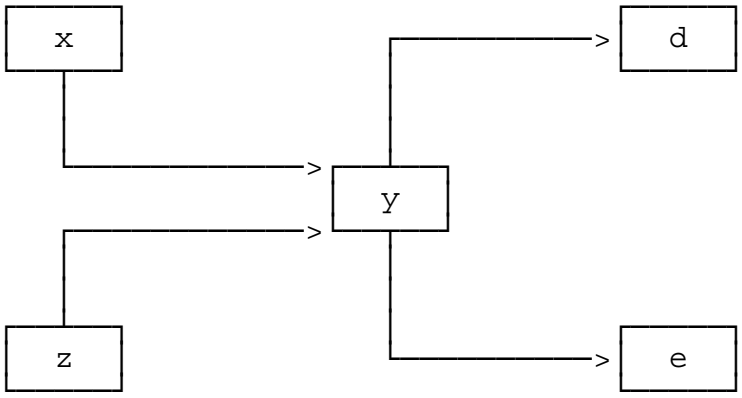


Figura 2.7.

El left del nodo y es : **L(y) = { x, z }**

Right : El right de un nodo **a** es el conjunto de nodos que son segunda componente de una relación donde a es primera componente. La definición algebraica es : **R(a) = { b ∈ P / (a, b) ∈ E }**

En la Figura 2.7. el right del nodo y es : **R(y) = { d, e }**

In-Degree : El in-degree de un nodo, es el grado de entrada del nodo. Es la cantidad de nodos que se relacionan con un nodo dado. La notación algebraica para el in-degree es **°L(x)°**

En el ejemplo de la Figura 2.7.

°L(x)° = 0 °L(z)° = 0 °L(y)° = 2 °L(d)° = 1 °L(e)° = 1

Out-Degree : El out-degree de un nodo, es el grado de salida del nodo. Es la cantidad de nodos con los que el nodo se relaciona. La notación algebraica para el out-degree es **°R(x)°**

En el ejemplo de la Figura 2.7.

°R(x)° = 1 °R(z)° = 1 °R(y)° = 2 °R(e)° = 0 °R(d)° = 0

Paso : Se dice que existe **paso** entre el nodo **a** y el nodo **b**, $\delta_{(a, b)}$ si existe una secuencia de nodos $< y_0, y_1, \ldots, y_n >$ con $n \cdot 0$ si se cumple

1) $a = y_0$; $b = y_n$

2) $y_{i-1} <> y_i$

3) $(y_{i-1}, y_i) \in E$ para $1 \mu i \mu n$

Ejemplo : En el grafo de la Figura 2.7. existen los siguientes pasos

$\delta_{(x, x)}$, $\delta_{(x, y)}$, $\delta_{(x, d)}$, $\delta_{(x, e)}$

$\delta_{(z, z)}$, $\delta_{(z, y)}$, $\delta_{(z, d)}$, $\delta_{(z, e)}$

$\delta_{(y, y)}$, $\delta_{(y, d)}$, $\delta_{(y, e)}$, $\delta_{(d, d)}$

$\delta_{(e, e)}$

De los ejemplos vistos, intuitivamente puede observarse que un paso es la secuencia de arcos que debo recorrer para llegar hasta un nodo partiendo de otro nodo. Por lo tanto, se deduce que :

1) para todo $x \in P$ existe $\delta_{(x, x)}$ donde $|\delta_{(x, x)}| = 0$, pues no hay ningún arco

2) si hubiere un loop, la longitud del paso es 1.

Paso simple : Se dice que un paso es simple si en la secuencia de nodos

$$< y_0, y_1, \ldots, y_{i-1}, y_i, \ldots, y_n >$$

para todo i los y_i son distintos entre sí.

Longitud del paso : Es el número de arcos que se registra en el paso desde un nodo a otro. **Ejemplo :**

Sea **G(P, E)** donde P = { x, y, v, z } y E = { (x, y) ; (y, v) ; (v, z) }

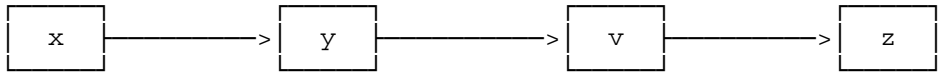


Figura 2.8.

La longitud de paso de **x** a **z** es igual a : $|\delta_{(x, z)}| = 3$

Ciclo : Es un paso simple de longitud · 2 donde el primer y el último nodo de la secuencia de nodos coinciden.

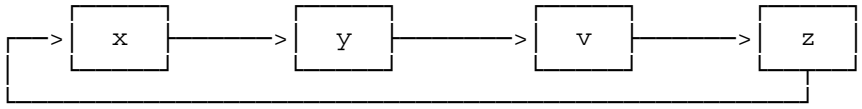


Figura 2.9.

existe un ciclo en x pues existe $\delta_{(x, x)}$ de $|\delta_{(x, x)}| = 4$

Relación de paso : Si se aplica el concepto de paso a un nodo cualquiera, se podrán definir todas las relaciones donde el nodo considerado es primera componente y la segunda componente es otro nodo con el cuál el nodo en cuestión tiene paso. En la Figura 2.8.

1) por propiedad de paso, existe $\delta_{(x, x)}$ para todo x ∈ P

2) por transitividad, se verifica que :

si existe $\delta_{(x, y)}$ .and. existe $\delta_{(y, z)}$ => existe $\delta_{(x, z)}$

aplicando las propiedades al grafo considerado, el resultado es el grafo derivado de la relación de paso de la Figura 2.10. :

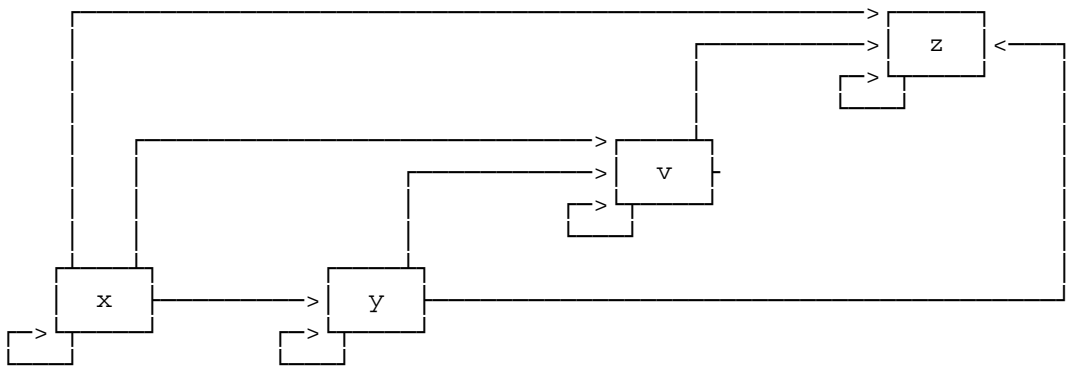


Figura 2.10.

Entonces, la relación de paso es $\delta_D = \delta_{(P,\_)}$. Por extensión:

δ_D = { (x, x); (x, y); (x, v); (x, z); (y, y); (y, v); (y, z); (v, v); (v, z); (z, z) }

Sabemos que este grafo es consecuencia de haber aplicado al grafo original las propiedades reflexiva y transitiva. Estas son dos de las tres propiedades que deben cumplirse en las relaciones de orden parcial, y la relación de equivalencia.

Si cumple además con la propiedad **débilmente antisimétrica**, la relación es de orden parcial y el grafo es **acíclico** es decir, **sin ciclos**. El grafo de la Figura 2.8. es acíclico. Esto se puede deducir a simple vista pero cuando el grafo es sumamente complejo, habrá que recurrir al análisis algebraico.

Si además el grafo derivado cumple con la propiedad **simétrica**, la relación es de equivalencia y si la partición induce una única clase, se dice que el grafo es **fuertemente conectado**.

Como se puede observar, el concepto de paso es más amplio que los otros conceptos vistos. Es posible derivar una relación que sirve para obtener otro grafo, sobre el cual, al analizar las propiedades algebraicas que cumple, pueden deducirse ciertas características del grafo original.

Grafo fuertemente conectado : Un grafo es **fuertemente conectado** si su relación de paso inducida es una relación de equivalencia consistente en una única clase. Este tipo de grafos se caracteriza por el hecho de que para cualquier par de nodos **x**,**z** ∈ P, existe $\delta_{(x,\ z)}$. **Ejemplo :**

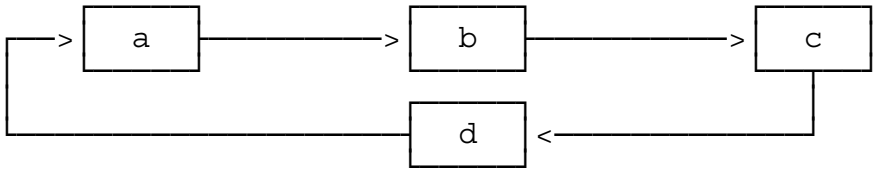


Figura 2.11.

P = { a, b, c, d }

E = { (a, b); (b, c); (c, d); (d, a) }

D = { (a, a); (a, b); (a, c); (a, d); (b, b); (b, c); (b, d); (b, a); (c, c); (c, d); (c, a); (c, b); (d, d); (d, a); (d, b); (d, c)}

Ejemplo

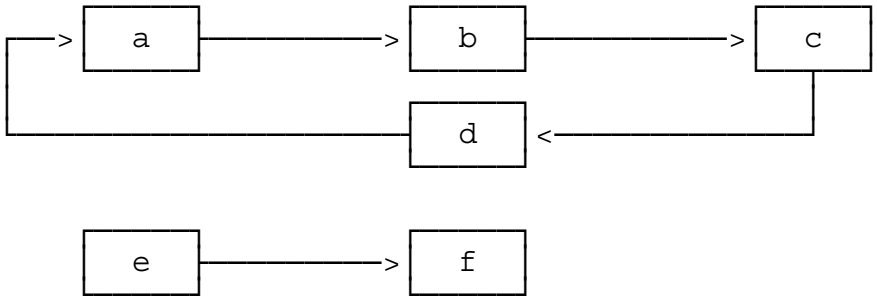


Figura 2.12.

En este caso la relación de paso no induce a una relación de equivalencia pues o bien no existe $\delta_{(a,\ e)}$, o bien no se cumple la simetría entre los nodos **e** y **f**.

Teorema : Un grafo G(P, E) es acíclico <=> la relación de paso inducida es una relación de **orden parcial**.

En el ejemplo de la Figura 2.8. el grafo inducido de la relación de paso es el grafo de la Figura 2.10. En este grafo, aplicando las propiedades se observa :

1) que se cumple la propiedad **reflexiva** pues :

para todo x ∈ P existe (x, x) ∈ E

2) que se cumple la propiedad **transitiva** pues :

si (x, y) ∈ E .and. (y, z) ∈ E => (x, z) ∈ E

3) que se cumple la propiedad débilmente antisimétrica pues :

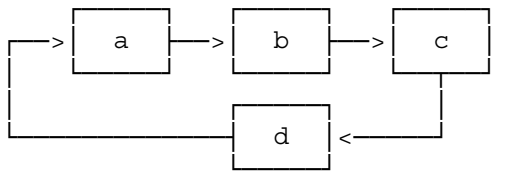
para todo (x,y) ∈ E se cumple que si (x,y) .and (y,x) => x = y

Componente conexa : Una componente conexa está formada por todos los nodos que están relacionados de forma tal que partiendo de un nodo cualquiera, puedo recorrer toda la estructura y volver al mismo nodo. Esto implica una partición en P producida por la relación de walk. Cada componente conexa es una clase de equivalencia al ser la relación, una relación de equivalencia. De esta afirmación puede deducirse que :

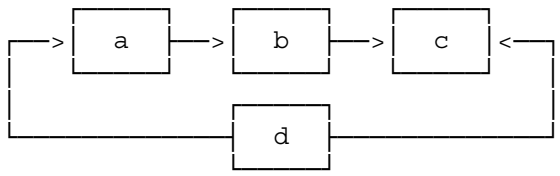
a) En una componente conexa, no hay nodos aislados
b) La unión de todas las componentes conexas devuelve el conjunto original
c) La intersección de dos componentes conexas cualquiera es vacía

Tipos de grafos. Otros ejemplos.

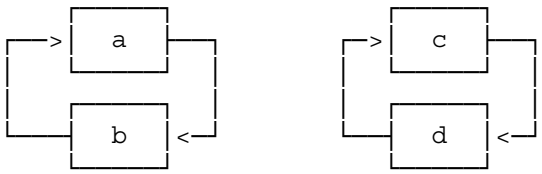
Los siguientes grafos son facilmente clasificados a traves del dibujo. En el caso que de grafos complejos o que se quisiera implementar el grafo a traves de un computador es necesario verificar que tipo de grafo es por medio de las propiedades algebraicas.



Grafo Fuertemente conectado



Grafo Acíclico



Componentes Conexas

Ideal izquierdo: El ideal izquierdo de un nodo a,

simbolizado $\overline{L}(a)$, es el conjunto de los nodos que son primera componente de una relación que pertenece a la relación de paso y donde la segunda componente es el nodo considerado. En notación algebraica :

$$L_{(x)} = \{ y \in P \ / \text{ existe } \delta_{(y,\ x)} \}$$

Ejemplo :

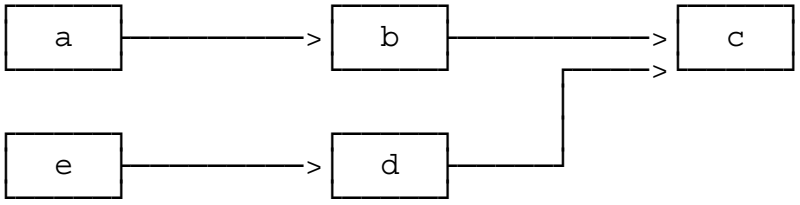


Figura 2.13.

$\overline{\mathbf{L}}_{\mathbf{(a)}} = \{ a \}$ $\overline{\mathbf{L}}_{\mathbf{(b)}} = \{ a, b \}$ $\overline{\mathbf{L}}_{\mathbf{(c)}} = \{ a, b, c, d, e \}$

$\overline{\mathbf{L}}_{\mathbf{(e)}} = \{ e \}$ $\overline{\mathbf{L}}_{\mathbf{(d)}} = \{ e, d \}$

Ideal derecho : El ideal derecho de un nodo **a**, simbolizado

$\overline{\mathbf{R}}_{\mathbf{(a)}}$, es el conjunto de los nodos que son segunda componente de una relación que pertenece a la relación de paso y donde la primera componente es el nodo considerado.

En notación algebraica :

$$\overline{R}_{(x)} = \{ y \in P \ / \text{ existe } \delta_{(x,\ y)} \}$$

En el ejemplo de la Figura 2.13.

$\overline{R}_{(a)} = \{ a, b, c \}$ $\overline{R}_{(b)} = \{ b, c \}$ $\overline{R}_{(c)} = \{ c \}$

$\overline{R}_{(e)} = \{ e, d, c \}$ $\overline{R}_{(d)} = \{ d, c \}$

Ideal Principal : Si el ideal esta compuesto por un único elemento, entonces el ideal se denomina **ideal principal** (derecho o izquierdo). En el caso de que el grafo esté compuesto por un único elemento, coinciden los ideales.

Grafo básico : Cuando se representa graficamente o se dibuja una relación de orden parcial sobre un conjunto, generalmente se omiten arcos en el dibujo para una mayor claridad del mismo. Así por ejemplo es posible quitar un arco (x, y) cuya presencia sea inferida por la existencia del paso $\delta_{(x, y)}$. Si al grafo de la Figura 2.10., le son quitados los arcos redundantes, quedará el grafo de la Figura 2.8. al cuál llamaremos ***grafo básico***. El concepto de usar un grafo básico en una representación gráfica, puede llevarse al campo computacional aprovechando la eliminación de arcos redundantes, disminuyendo de esta manera el espacio ocupado, (pues no se guardarán relaciones redundantes) y aumentando el tiempo de búsqueda pues, si bien existe la relación entre **x** e **y**, la cantidad de arcos que deben recorrerse para establecer la relación será mayor.

Teorema : Un grafo acíclico G(P, E) es básico <=>

1) No tiene loops
2) para todo x, z ∈ P; si existe $|\delta_{(x, z)}| \cdot 2$ => (x, z) no ∈ E

Clausura transitiva de un grafo : Así como por definición, en el grafo básico se omiten los arcos, la clausura transitiva de un grafo surge como consecuencia de agregar al grafo original los arcos de las relaciones resultantes al obtener la relación de paso, mostrando en el grafo todas las relaciones posibles. Así por ejemplo, si no existe el arco (x, y) pero existe $\delta_{(x, y)}$ entonces se agregará en el grafo de la clausura transitiva el arco (x, y). El concepto de usar la clausura transitiva de un grafo en una representación gráfica, puede llevarse al campo computacional aprovechando el agregado de todos los arcos redundantes, disminuyendo de

esta manera el tiempo de búsqueda pues si bien **x** está relacionado con **y** por una cierta cantidad de arcos, en el grafo de la clausura transitiva, deberá recorrerse un sólo arco para ir de x a y. **Ejemplo :** el grafo de la Figura 2.10. es el grafo de la clausura transitiva del grafo de la Figura 2.8.

Si se quisiera comparar la eficiencia del modelo de clausura transitiva respecto del grafo básico por medio de un par de ejes cartesianos, cuyas coordenadas fueran tiempo de procesador utilizado y espacio ocupado el gráfico sería el siguiente.

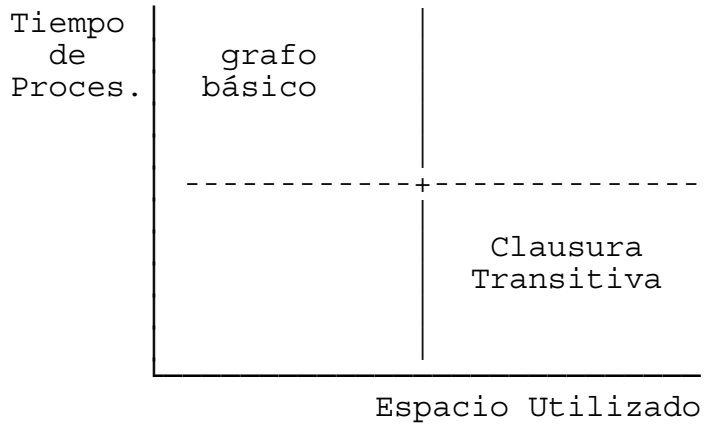


Figura 2.14.

Clasificación de grafos : Una primera clasificación de grafos permite agruparlos como :

a) Lineales
b) Acíclicos
c) Irrestrictos
d) Arboles

Grafo lineal : Un grafo es lineal cuando la relación de paso es de orden total y además es básico. **Ejemplo :** El grafo de la Figura 2.10. es de orden total pero no es básico. El grafo de la Figura 2.8. es de orden total y además es básico por lo tanto es un grafo lineal.

Los grafos acíclicos fueron definidos previamente y los grafos irrestrictos y los árboles serán estudiados posteriormente.

2. Programa para Grafos implementados en memoria.

```
{ * Arma una Matriz de 5 x 5 * }
Const Cantnod = 5;

Type  Nodos  = array [1..CantNod] of char;
      Matriz = array [1..CantNod,1..CantNod] of integer;
var   nodo : char;
      c    : integer;
      Nod  : Nodos;
      Mat  : Matriz;

Procedure CargoNodos (var WorkNodo : Nodos);
var   i  : integer;
begin
   For i := 1 to CantNod do
   begin
      clrscr;
      Write(lst,' Ingrese Nodo   : ');
      Readln(nodo);
      WorkNodo[i] := nodo;
   end;
   For i := 1 to CantNod do
   begin
      writeln(lst,' El Nodo (',i,') es : ',WorkNodo[i]);
   end;
end;

Procedure ArmoMatriz (var WorkMat : Matriz);
var   i : integer; j : integer;
begin
   For i := 1 to CantNod do
   begin
      For j := 1 to CantNod do
          Mat[i,j] := 0;
   end;
end;

Procedure CargoMatriz (var Mat : Matriz);
var DatoX : char; DatoY : char; k : integer; t : integer;
senal : boolean;
begin
  t := 0;
  write(' ingrese nodo desde donde parte el arco [ "0"
finaliza ] : ');
  readln(DatoX);
  write(' ingrese nodo a donde llega el arco [ "0" finaliza ]
: ');
  readln(DatoY);
  Repeat
  begin
    k := 1;
    t := t+1;
    Senal := False;
```

```
    Repeat
    begin
      if Nod[k] = DatoX
         then begin
                Mat[t,k] := 1;
                senal := true;
              end;
      k:=k+1;
    end;
    until (k > CantNod) OR (Senal = true);
    k := 1;
    Senal := false;
    Repeat
    begin
      if Nod[k] = DatoY
         then begin
                Mat[t,k] := Mat[t,k]+2;
                Senal := true;
              end;
      k:=k+1;
    end;
    until (k > CantNod) OR (Senal = true);
    write(' ingrese nodo desde donde sale el arco [ "0"
finaliza ] : ');
    readln(DatoX);
    write(' ingrese nodo a donde llega el arco [ "0" finaliza
] : ');
    readln (DatoY);
  end;
  until (DatoX = '0') AND (DatoY = '0');
end;

Procedure ListoMatriz (var WorkMat : Matriz);
var   i : integer; j : integer;
begin
  writeln('              Listado de la Matriz de adyacencia
');
  for i:= 1 to CantNod do
  begin
    for j:= 1 to CantNod do
        Writeln(' Posicion ',i,',',j,' valor : ',Mat[i,j]);
  end;
end;

Procedure Minimal (var Mat : Matriz;Nod : Nodos;CantNod :
Integer);
var   i : integer; j : integer; Contador : boolean; Marca :
boolean;
begin
  i := 0;
  j := 0;
  Marca := True;
  Writeln('                      Listado de Minimales');
```

```
  Repeat
  begin
    j := j + 1;
    i := 0;
    Contador := True;
    Repeat
    begin
      i := i+1;
      if Mat[i,j] = 2
         then begin
                Contador := False;
                Marca := False;
              end;
    end;
    until (i = CantNod) OR (Contador = False);
    if Contador
       then writeln(' El Nodo ',Nod[j],' es minimal');
  end;
  until j = CantNod;
  if Marca
     then writeln(' No existen Nodos que sean Minimales ');
end;

Procedure Maximal (var Mat : Matriz;Nod : Nodos;CantNod :
integer);
var i : integer; j : integer; Contador : boolean; Marca :
boolean;
begin
   Marca := True;
   i     := 0;
   j     := 0;
   Writeln('                      Listado de Maximales');
   Repeat
      j := j + 1;
      i := 0;
      Contador := true;
      Repeat
         i := i+1;
         if Mat[i,j] = 1
            then begin
                   Contador := False;
                   Marca    := False;
                 end;
      until (i = CantNod) OR (Contador = False);
      if Contador
         then writeln(' El Nodo ',Nod[j],' es Maximal');
   until j = CantNod;
   if Marca
      then writeln(' No existen Nodos que sean Maximales');
end;

Procedure Left (var Mat : Matriz; Nod : Nodos; CantNod :
integer);
var   k,j,h,r,i,H1 : integer; DatoX : char;
```

```
      Izquierdo,Derecho : Nodos;  Eureka : boolean;
begin
   k := 0;
   r := 0;
   j := 0;
   Eureka := False;
   write(' Ingrese el Nodo del cual desea el conjunto Left  :
');
   readln(DatoX);
   Repeat
      k := k + 1;
      if (Nod[k] = DatoX)
         then begin
                 j := k;
                 Eureka := true;
              end;
   until (Eureka) or (k >CantNod);
   For i := 1 to CantNod do
   begin
      if Mat[i,j] = 2
         then begin
                 For h := 1 to CantNod do
                 begin
                    if Mat[i,h] = 1
                       then begin
                               r := r + 1;
                               Izquierdo[r] := Nod[h];
                            end;
                 end;
              end;
   end;

   if r <> 0
      then begin
              For h := 1 to r do
                  writeln(Izquierdo[h],'_ al Left de
',DatoX);
           end
      else writeln(' el Nodo ',DatoX,'no tiene Left');
    {* Analizo el Right *}
    k := 0;
    r := 0;
    j := 0;
    Eureka := False;
    write(' Ingrese el Nodo del Cual desea el Conjunto Right
: ');
    readln(DatoX);
    Repeat
       k := k + 1;
       if Nod[k] = DatoX
          then begin
                  j := k;
                  Eureka := True;
               end;
```

```
     until (Eureka) or (k > CantNod);
     if eureka
        then begin
               For i := 1 to CantNod do
               begin
                  if Mat[i,j] = 1
                     then begin
                             For h := 1 to CantNod do
                             begin
                                if Mat[i,h] = 2
                                   then begin
                                          r := r + 1;
                                          Derecho[r] :=
Nod[h];
                                end;
                           end;
               end;
             end;
             if r <> 0
                then begin
                        For h := 1 to r do
                        begin
                           writeln(Derecho[h],' _ al Right de
',DatoX);
                            end;
                        end;
                     end
                 else writeln(' el Nodo ',DatoX,'no tiene
right');
end;

Begin        {* Program Body *}
 CargoNodos (Nod);
 ArmoMatriz (Mat);
 CargoMatriz(Mat);
 ListoMatriz(Mat);
 Minimal(Mat,Nod,CantNod);
 Maximal(Mat,Nod,CantNod);
 Left(Mat,Nod,CantNod);
end.
```

3. Matriz de la Claurura Transitiva

Dado un grafo **G** supongamos que se representa a través de su matriz de adyacencia booleana. Sea el grafo G

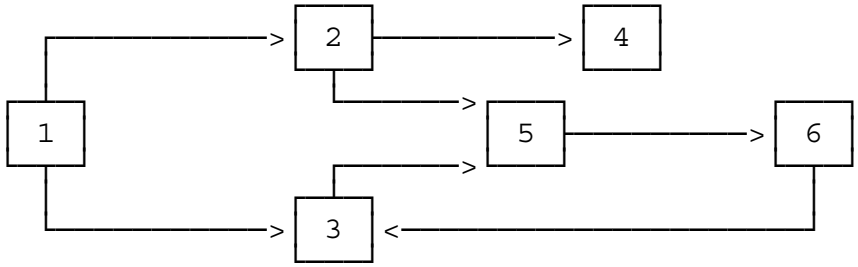


Figura 2.15.

La matriz de adyacencia será : $A_G^{(0)}$

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	1	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	0
5	0	0	0	0	0	1
6	0	0	1	0	0	0

	1	2	3	4	5	6
1	1	1	1	0	0	0
2	0	1	0	1	1	0
3	0	0	1	0	1	0
4	0	0	0	1	0	0
5	0	0	0	0	1	1
6	0	0	1	0	0	1

$A_G^{(1)}$

	1	2	3	4	5	6
1	1	1	1	1	1	0
2	0	1	0	1	1	1
3	0	0	1	0	1	1
4	0	0	0	1	0	0
5	0	0	1	0	1	1
6	0	0	1	0	1	1

$A_G^{(2)}$

	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	0	0	1	0	1	1
4	0	0	0	1	0	0
5	0	0	1	0	1	1
6	0	0	1	0	1	1

$A_G^{(3)}$

	1	2	3	4	5	6
1	1	1	1	1	1	1
2	0	1	1	1	1	1
3	0	0	1	0	1	1
4	0	0	0	1	0	0
5	0	0	1	0	1	1
6	0	0	1	0	1	1

Al armar la matriz de adyacencia; se marcan los pasos de longitud 1. Al agregar los 1 en la diagonal principal, queda registrado en la matriz $A_G^{(0)}$ la propiedad reflexiva, donde cada nodo esta relacionado consigo mismo por definición de paso, es decir aquellos pasos de longitud 0.

La matriz $A_G^{(1)}$ se obtiene como resultado de la multiplicación booleana de la matriz $A_G^{(0)}$ por sí misma y representa la existencia de todos los pasos de longitud 0, 1 y 2.

La matriz $A_G^{(2)}$ se obtiene como resultado de la multiplicación booleana de la matriz $A_G^{(1)}$ por $A_G^{(0)}$, es decir, $A_G^{(0)} * A_G^{(0)} * A_G^{(0)}$, y representa la existencia de todos los pasos de longitud 0, 1 y 2 y 3.

La matriz $A_G^{(3)}$ es idéntica a la matriz $A_G^{(2)}$, por lo tanto $A_G^{(3)}$ es la matriz de la clausura transitiva y la que permite conocer si entre dos nodos cualquiera existe paso. Esto puede verificarse facilmente buscando en la celda determinada por la fila y la columna de los nodos considerados de la matriz $A_G^{(3)}$ si hay un 1, en cuyo caso hay paso.

Todo este desarrollo nos devuelve como resultado una matriz que solamente nos indica si entre dos nodos dados hay paso. El algoritmo podría sintetizarse de la siguiente manera :

```
for j = 1 to n
```

```
for i = 1 to n
    for k = 1 to n
        a(i,k) := a(i, k) .or. ( a(i, j) .and. a(j, k) )
```

4. Algoritmo de Warshall

Uso del algoritmo :

1) Para conocer la cantidad de pasos distintos de igual longitud hay entre dos nodos.
2) Para saber la longitud de paso mínima entre dos nodos.

El ***algoritmo de Warshall*** nos permite obtener más información que la matriz de la clausura transitiva, utilizando una técnica similar. En lugar de usar una multiplicación booleana se utiliza la multiplicación aritmética de matrices.

```
for i = 1 to n
    for k = 1 to n
        for j = 1 to n
            a(i,k) := a(i, k) + ( a(i, j) * a(j, k) )
```

Esta última expresión podemos reescribirla como :

$$a_{(i,k)} = \sum_{j=1..n} (a_{(i,j)} * a_{(j,k)})$$

$A_G^{(1)}$: Pasos de longitud 1

	1	2	3	4	5	6
1	0	1	1	0	0	0
2	0	0	0	1	1	0
3	0	0	0	0	1	0
4	0	0	0	0	0	0
5	0	0	0	0	0	1
6	0	0	1	0	0	0

$A_G^{(2)}$: Pasos de longitud 2

	1	2	3	4	5	6
1	0	0	0	1	2	0
2	0	0	0	0	0	1
3	0	0	0	0	0	1
4	0	0	0	0	0	0
5	0	0	1	0	0	0
6	0	0	0	0	1	0

$A_G^{(3)}$: Pasos de longitud 3

	1	2	3	4	5	6
1	0	0	0	0	0	2
2	0	0	1	0	0	0
3	0	0	1	0	0	0
4	0	0	0	0	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

D : Matriz de Warshall

	1	2	3	4	5	6
1	^	1	1	2	2	3
2	^	^	3	1	1	2
3	^	^	3	^	1	2
4	^	^	^	^	^	^
5	^	^	2	^	3	1
6	^	^	1	^	2	3

Sea $A_G^{(1)}$ la matriz de adyacencia donde se indican las relaciones originales, es decir *los pasos de longitud 1*. Sea D la matriz de Warshall, donde se guardarán los valores de paso mínimo entre dos nodos. Esta matriz se inicializa con el valor nil en todos sus elementos, que indica la ausencia de paso. En cada iteración de la matriz $A_G^{(n)}$ se obtienen **los pasos de longitud n** por tanto se actualizarán con este valor **(n)** los elementos de la matriz D en donde se encontró paso. Este valor es el menor paso que puede obtenerse pues la siguiente iteración indica el estudio de un paso de longitud mayor.

Siguiendo con el proceso, al multiplicar algebraicamente la matriz $A_G^{(1)}$ por sí misma, se obtiene la matriz $A_G^{(2)}$ que indica los pasos de longitud 2. En forma simultánea se actualiza la matriz D con los valores obtenidos en la segunda iteración. Este proceso se repite hasta |P| veces o hasta que todos los valores de la matriz $A_G^{(n)}$ sean cero, lo cual indica la ausencia de paso.

$$A_G^{(2)} := A_G^{(1)} * A_G^{(1)}$$

$$A_G^{(3)} := A_G^{(2)} * A_G^{(1)} \Rightarrow A_G^{(3)} := A_G^{(1)} * A_G^{(1)} * A_G^{(1)}$$

$$A_G^{(n)} := A_G^{(n-1)} * A_G^{(1)} \Rightarrow A_G^{(n)} := \underbrace{A_G^{(1)} * .... * A_G^{(1)} * A_G^{(1)}}_{\text{n veces}}$$

De esta forma si analizamos el significado de un elemento de cada tabla es posible extraer la siguiente información.

Si $a_{(i,k)}^{(L)} = m$ <=> existen **m** pasos de longitud **L** de **i** a **k** .

Si $d_{(i,k)} = \hat{}$ => no existe paso de **i** a **k** .

Si $d_{(i,k)} = m$ => la longitud de paso mínima de **i** a **k** es **m**.