# UNIDAD 1: LOGICA DE PROPOSICIONES

* Sintaxis y semántica de fórmulas proposicionales.
* Interpretaciones.
* Fórmulas satisfacibles.
* Modelos, tautologías y contradicciones.
* Implicación lógica.
* Consecuencia lógica.
* Teorema de la deducción.
* Conectivos adecuados.
* Reglas de prioridad.
* Circuitos lógicos.
* Reglas de inferencia:
  + Modus ponens.
  + Modus tollens.
  + Silogismo hipotético.

**Sintaxis del lenguaje de proposiciones:**

El alfabeto consiste en un conjunto de conjuntos no vacíos y finitos de símbolos los cuales son:

Variables proposicionales:

Conectivos lógicos:

Símbolos auxiliares:

Se escribe:

**Semántica de fórmulas proposicionales:**

Es el significado que adquiere una formula.

Una fórmula puede ser o verdadera o falsa, dependiendo de la verdad o falsedad de las fórmulas más simples que son sus componentes.

Ej.:

La verdad o falsedad de depende de los valores de verdad de los átomos p y q.

Se puede determinar el valor de verdad de esta fórmula recurriendo a los valores dados por alguna interpretación de sus átomos

Una **fórmula bien formada** (fbf), se define como:

1. Una variable proposicional es una fbf.
2. Si es una fbf, es una fbf.
3. Si y son fórmulas bien formadas, entonces también son fbf , , y
4. Una cadena de variables proposicionales, conectivos o símbolos del alfabeto es una fbf si y sólo si puede obtenerse mediante un número finito de aplicaciones de las reglas anteriores.

Ej.:

Si

es fbf

no es fbf

**Interpretación:**

Dada una fórmula proposicional compuesta, una **interpretación** define qué variables proposicionales del lenguaje son falsas (0) y cuales son verdaderas (1).

Definición:

Sea el conjunto de átomos del lenguaje :

Una *interpretación* es una función de , es decir que se asigna a cada elemento de un 0 o un 1

Ej.:

Si ;

Una interpretación se puede dar por:

Esto significa que las imágenes de y son 1, mientras que los restantes elementos del conjunto tendrán imagen 0

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  | 1 | 1 | 0 |

**Valuación**:

* Sea una interpretación.
* Sea el conjunto de fórmulas de L

Una **valuación** bajo una interpretación I, es cualquier función de en {0,1}, que satisface estas reglas:

1. para cada variable proposicional.

Informalmente: Cuando la fórmula es una variable proposicional, su *valuación* coincide con la asignación que le hace su *interpretación*.

1. Si y son fbf arbitrarias, se define según esta tabla:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |
| 0 | 0 | 1 | 0 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 |

Ej.:

Sea

Y la interpretación:

Porque: ,

**Formulas satisfacibles:**

Sea una fórmula e una interpretación.

es verdadera bajo , si su valor de verdad es 1

Entonces:

satisface a ;

Formalmente:

Ej.:

Sea

Y la interpretación:

Por tanto:

Si es falsa bajo , se dice que no satisface a ;

Formalmente:

Ej.:

Sea

Y la interpretación:

Por tanto:

**Modelo:**

Sea una fórmula y sea una interpretación.

es un *modelo* para si satisface a

**Tautología:**

es *tautología* si todas las interpretaciones son modelos para

Las tautologías se denotan con .

Cuando una fórmula es tautológica se escribe:

*Tipos más importantes de tautología:*

**Las equivalencias lógicas**

Se utilizan como esquemas de sustitución en procesos de razonamiento

Ej.:

**Las implicaciones lógicas**

Se utilizan como esquemas de razonamientos válidos

Ej.:

**Contradicción:**

Una fórmula es una contradicción si es falsa para todas las interpretaciones posibles.

Las contradicciones se denotan con

**Contingencia:**

Se trata de una fórmula que no es tautología ni contradicción. Una contingencia es satisfacible, pero no es tautología.

Sea  **un conjunto de fórmulas**, y una valuación

Diremos que una valuación satisface un conjunto de fórmulas, si para toda . Es decir, la valuación es modelo para **si es modelo para todas las fórmulas de** .

Ej.:

Sea buscar una valuación que satisfaga a S

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| p | q | r |  |  |  |
| 0 | 0 | 0 | 1 | 1 |  |
| 0 | 0 | 1 | 1 | 1 |  |
| 0 | 1 | 0 | 1 | 0 |  |
| 0 | 1 | 1 | 1 | 1 |  |
| 1 | 0 | 0 | 0 | 1 |  |
| 1 | 0 | 1 | 0 | 1 |  |
| 1 | 1 | 0 | 1 | 0 |  |
| 1 | 1 | 1 | 1 | 1 |  |

**Consecuencia lógica:**

Sea un conjunto de fórmulas y una fórmula.

es **consecuencia lógica** de , si toda interpretación que es modelo para también lo es para . **(S implica lógicamente a ϕ)**

El conjunto de consecuencias lógicas de se escribe:

Ej.:

,

, , ,

**Teorema de la deducción:**

Sea un conjunto de fórmulas y y fórmulas arbitrarias.

Luego si y sólo si .

Demostración:

1. implica

Sea una interpretación que verifica las fórmulas de

Probaremos que verifica también

Supongamos que , por hipótesis tenemos que:

; luego

Esto es suficiente para probar que verifica

1. implica

Sea una interpretación que verifica las fórmulas de

verifica todas las fórmulas de ;

por hipótesis verifica también

Al ser , necesariamente

Ej.:

Sea demostrar que

Se aplica el teorema de la deducción:

**Equivalencia lógica:**

Dos fórmulas y son equivalentes si para toda interpretación , los valores de verdad coinciden;

Es decir para toda

Ej.:

y se expresa

**Conectivos adecuados:**

Un conjunto de conectivos es adecuado si a partir de sus elementos pueden definirse todos los demás conectivos.

Los conjuntos:

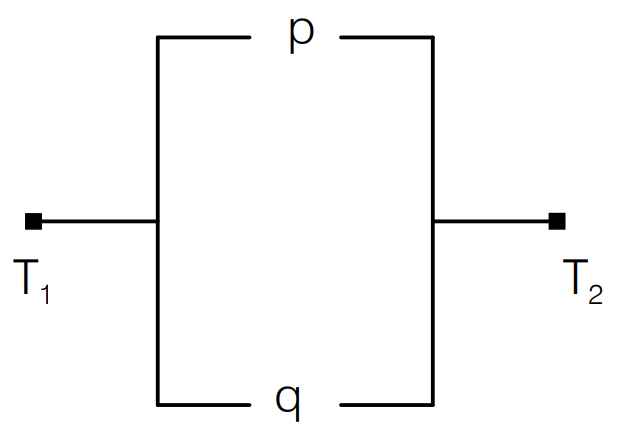
son adecuados

**Reglas de prioridad:**

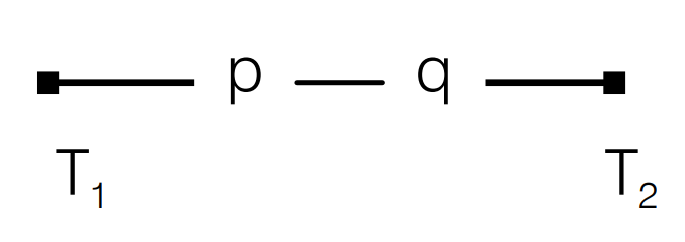
Es posible eliminar los paréntesis de las fórmulas proposicionales, estableciendo una regla de prioridad según el siguiente orden de mayor a menor:

Si en una formula los conectivos tienen el mismo orden de precedencia, la evaluación se realiza de izquierda a derecha

**Circuitos lógicos:**



Aquí los interruptores están en *paralelo* y este esquema se puede representar mediante la proposición



En este circuito es necesario que los interruptores p y q estén cerrados para que la corriente circule de T1 a T2.

Los interruptores están en *serie* y esta red se representa como

**Reglas de inferencia:**

Permiten reconocer razonamientos válidos.

Una inferencia (o razonamiento) es una implicación lógica.

El *antecedente* esta dado por una cadena de conjunciones (llamadas *premisas*) y expresa un conocimiento ya obtenido

El *consecuente* es la conclusión y expresa un conocimiento nuevo obtenido a partir del antecedente.

El razonamiento es válido si la implicación es una tautología.

Si las *premisas* son verdaderas, la *conclusión* debe ser verdadera para que el razonamiento sea considerado válido.

Si las *premisas* son falsas, no importará la *conclusión*; el condicional será verdadero y el razonamiento válido.

**Reglas de inferencia más usuales:**

**Modus Ponens:**

Premisa 1:

Premisa 2:

Conclusión:

Ej.:

P: x es correntino

Q: x es argentino

X es correntino **y**, **si** X es correntino **entonces** X es argentino; **luego** X es argentino

**Modus Tollens:**

Premisa 1:

Premisa 2:

Conclusión:

Ej.:

P: aprobé el examen

Q: te presto los apuntes

**Si** aprobé el examen, **entonces** te presto los apuntes **y** no te presto los apuntes; **luego**, no aprobé el examen

**Silogismo Hipotético (transitividad):**

Premisa 1:

Premisa 2:

Conclusión:

Ej.:

P: Hago mi trabajo más eficiente

Q: Gano mas

R: mejora mi nivel de vida

**Si** Hago mi trabajo más eficiente **entonces** gano más **y** **si** gano más **entonces** mejora mi nivel de vida

**Luego:** **si** hago mi trabajo más eficiente, **entonces** mejora mi nivel de vida

# UNIDAD 2: RELACIONES DE RECURRENCIA

* Sucesiones y sumatorias.
* Definiciones por recurrencia.
* Relaciones de recurrencia.
* Clasificación de las relaciones de recurrencia.
* Resolución de los diferentes tipos de relaciones:
  + lineales – no lineales,
  + homogéneas – no homogéneas
  + coeficientes constantes – coeficientes variables.
* Generación de números (pseudo) aleatorios.

**Progresión Geométrica/sucesión:** Sucesión infinita de números donde el cociente de cualquier término (distinto del primero) entre su predecesor es una constante llamada *razón común*.

**Sucesión Def 1:** Una sucesión es una función que va de Naturales en Reales; si va de Naturales en Enteros se llama *Sucesión entera*

Ej. De sucesión:

son términos de la sucesión, donde es el 1er termino y así hasta que es el termino n-ésimo.

Si se incluye el cero en el dominio de la sucesión, será el primer elemento de la sucesión y será el término *(n+1)-ésimo* de la sucesión.

**Relación de Recurrencia:** Ecuación donde para obtener el valor actual se depende de uno o más valores predecesores inmediatos a él.

Ej. De relación de recurrencia/sucesión:

Sucesión de Fibonacci: 1, 1, 2, 3, 5, 8, 13, 21…

Donde cada termino es la suma de los dos anteriores

No existe fórmula explícita para ; para conocer su valor es necesario conocer los valores de los dos términos anteriores, es decir y

Para calcular los términos de esta sucesión se utiliza la Formula de Binet:

Siendo

**Condiciones Frontera o Iniciales:** Valores iniciales que se necesitan para resolver una relación de recurrencia, estos generan unívocamente una sucesión y se denotan como o

Ej. de *Sucesión, relación de recurrencia y condiciones iniciales*:

Sea la siguiente sucesión tal que S: 2, 10, 50, 250, 1250, …

S:

La ecuación *recurrente* se puede interpretar como:

Y la *no recurrente* como:

**Clasificación de las relaciones de recurrencia**:

***Lineal:*** Cuando cada término con subíndice de la relación de recurrencia aparece elevado a la primera potencia, ej.:

***No Lineal***: Cuando algún término con subíndice de la relación de recurrencia esta elevado a una potencia diferente a la primera, ej.:

***Coeficientes Constantes***: Cuando cada término con subíndice de la relación de recurrencia está multiplicado por una constante, ej.:

***Coeficientes Variables:*** Cuando algún término con subíndice de la relación de recurrencia está multiplicado por un valor variable, por ejemplo:

***Homogénea:*** Cuando f(n) = 0 para todo n ∈ N, por ejemplo:

***No Homogénea***: Cuando f(n) ≠ 0 para todo n ∈ N, por ejemplo:

***De orden k***:

Donde: son constantes Reales, y

Ej.:

1er orden:

2º orden:

3er orden:

**Solución General:** El valor de es una función de *n* que *no depende* de los términos anteriores de la sucesión, una vez definido , que se obtiene a partir de la relación de recurrencia.

**Solución de relaciones de recurrencia homogéneas:**

Relación de recurrencia homogénea:

*Polinomio* característico de grado k asociado:

Donde:

* *r* es una raíz del polinomio
* *c* son constantes reales
* es el termino independiente
* *k* es el grado del polinomio/orden de la ecuación de recurrencia

Ecuación característica:

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

**Relaciones de recurrencia de orden 1, homogéneas, lineales**

Solución general:

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

Ecuación no recursiva:

**Relaciones de recurrencia de orden 2, homogéneas, lineales:**

Raíces reales y distintas S1 y S2

Solución general:

Sistema de ecuaciones:

Si las condiciones iniciales son a0 y a1

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

Sistema de ecuaciones:

Ecuación no recursiva:

Raíces reales e iguales S

Solución general:

Sistema de ecuaciones:

Si las condiciones iniciales son a0 y a1

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

Sistema de ecuaciones:

Ecuación no recursiva:

**Relaciones de recurrencia de orden 3, homogéneas, lineales:**

Raíces reales distintas:

Solución general:

Sistema de ecuaciones:

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

Raíces:

Sistema de ecuaciones:

Ecuación no recursiva:

Una raíz real doble:

Solución general:

Sistema de ecuaciones:

Ej.:

Dada la siguiente relación:

Se hace homogénea:

Polinomio característico:

Ecuación característica:

Raíces:

Sistema de ecuaciones:

Calc. Aux.:

Ecuación no recursiva:

Raíz triple:

Solución general:

Sistema de ecuaciones:

# UNIDAD 3: ESTRUCTURAS ALGEBRAICAS FINITAS

* Leyes de composición interna.
* Propiedades.
* Monoide.
* Semigrupo.
* Semigrupo con unidad.
* Grupo.
* Grupo Abeliano.
* Subgrupo.
* Anillo.
* Anillo con unidad.
* Cuerpo.

Una **ley de composición** es la aplicación de un operador sobre una selección de elementos de un conjunto. El operador toma los elementos originales y los relaciona con otro elemento de un conjunto final el cual puede ser de la misma naturaleza.

Simbólicamente:

Una **ley de composición interna**, definida en un conjunto A no vacío, consiste en una operación que asigna a cada par ordenado de elementos de A un único elemento de A.

**Definición**

Ley de composición interna definida en un conjunto no vacío A, es toda función de A X Á en A.

En símbolos:

Es decir:

Ej.:

La suma en N y la suma en Z.

**Propiedades:**

Asociativa:

Ej.:

Si A = {x / x = 2k, k ∈ Z}; + es el producto ordinario:

Elem. Neutro: es un elemento “e” que al operarlo con cualquier otro elemento “a” resulta el mismo elemento

Ej.:

Si A = {x / x = 2k, k ∈ Z}; + es el producto ordinario:

Para cada 2k debe existir 2t = e, con *t ∈ Z*

entonces

Elem. Inverso: elemento “a´” que al operarlo con “a” resulta el neutro

Ej.:

Si A = {x / x = 2k, k ∈ Z}; + es la adición:

Asumimos que existe *e = 0* (neutro) en A

si

Si

Conmutativa:

**Monoide:**

Un monoide (A, +) es una estructura algebraica en la que A es un conjunto no vacío y + es una función que cumple con la ley de composición interna en A

EJ:

(N, +); +: suma aditiva

(R, +); +: producto ordinario

Propiedades:

Sea A un conjunto no vacío

una función

1. Si existe un neutro en A para +, este es único
2. Sea asociativa y

Si *a* tiene inverso en A, este es único

**Semigrupo:**

(A, +) es semigrupo si:

1. La operación en el monoide dado es una la ley de composición interna
2. La operación es asociativa en A

Ej. de semigrupos:

(N, +); + es suma aditiva

(Z, +); + es suma aditiva

**Semigrupo con unidad:**

(A, +) es semigrupo con unidad si:

1. Cumple con las condiciones de semigrupo
2. Definida una operación binaria +, existe un elemento neutro en el conjunto A, que al ser operado con cualquier elemento “a”, resulta en el mismo elemento.

Ej. de semigrupos con unidad:

(N0, +); + es suma aditiva

Es semigrupo con unidad ya que cuenta con el numero 0 como neutro

(N, +); + es suma aditiva

NO es semigrupo con unidad ya que no existe ningún neutro para la suma dentro del conjunto de los naturales

**Grupo:**

(A, +) es grupo si:

1. Cumple con las condiciones de semigrupo con unidad
2. Definida una operación binaria +, existe para cada elemento de A al menos un elemento inverso ***a´***, que al ser operado con ***a*** da como resultado el neutro ***e***.

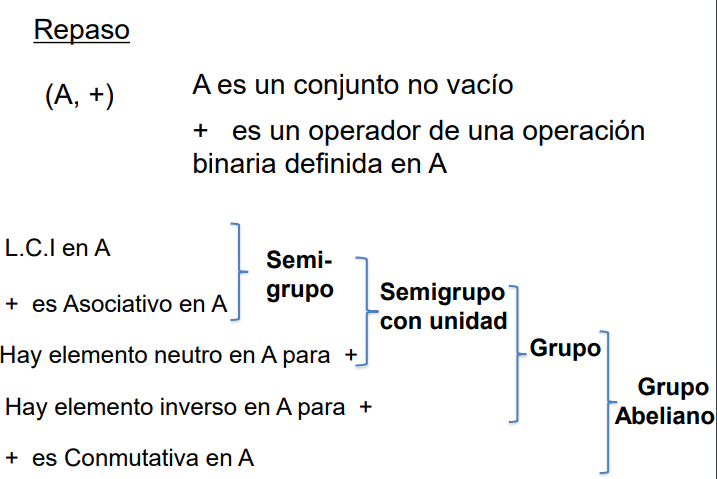
Ej.:

(Z, +); + es suma aditiva

**Grupo Abeliano:**

(A, +) es grupo abeliano si:

1. Cumple con las condiciones de grupo
2. Y además se cumple la propiedad conmutativa



Ej.:

(Z, +); + es suma aditiva

**Grupos Finitos:**

Grupo cuyo conjunto fundamental G tiene un número de elementos finito

Sea (G, +) un grupo finito, G es un conjunto finito.

Orden de G: es el número de elementos de G

G = {e}

|  |  |
| --- | --- |
| + | **e** |
| **e** | *e* |

G = {e, a, b}

|  |  |  |  |
| --- | --- | --- | --- |
| + | **e** | **a** | **b** |
| **e** | *e* | *a* | *b* |
| **a** | *a* | *b* | *e* |
| **b** | *b* | *e* | *a* |

No se deben repetir elementos en la misma línea para no perder la unicidad del neutro

Propiedades:

Sea (G, +) grupo, entonces:

1. El neutro es único. El inverso de cada elemento es único
2. Los elementos de G son regulares
3. Las ecuaciones admiten solución única en G
4. Involución:
5. Complemento:

Ej.:

Grupo de Klein

G = {e, a, b, c}

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| + | **e** | **a** | **b** | **c** |
| **e** | *e* | *a* | *b* | *c* |
| **a** | *a* | *e* | *c* | *b* |
| **b** | *b* | *c* | *e* | *a* |
| **c** | *c* | *b* | *a* | *e* |

**Subgrupos:**

Sea (A, +)

Un conjunto no vacío S es subgrupo de A cuando S es grupo con el operador binario +

Sea (A, +) un Grupo

* S incluido en A ()
* S no vacío ()

El Grupo (S, +) es Sub Grupo de (A, +) si:

S contiene el elemento neutro de A, es decir:

La operación + es cerrada en S

S contiene elementos simétricos

Propiedades de los subgrupos:

1. Todo Grupo A, tiene al menos dos sub grupos

S₁ = {e}

S₂ = A

1. Transitividad de los subgrupos

Sean S₁, S₂ y S3 subgrupos de A

Si *S1 es subgrupo de S₂ y S₂ es subgrupo de S3* entonces: *S₁ es subgrupo de S3*

1. La intersección de dos subgrupos es un subgrupo

Sean S y S’ dos subgrupos de A

Ej.:

Si Σ = {0,1};

Σ3 = {000,001,010, 100,011,101,110,111}

+ Se define: (x1 x2 x3) + (y1 y2 y3) = (x1 + y1, x2 + y2, x3 + y3)

**Anillo**

Sea una estructura algebraica definida en un conjunto G con dos leyes de composición + y •

(A, +, • ) es Anillo si se cumplen las siguientes condiciones:

1. (A, +) es Grupo abeliano
2. (A, •) es semi Grupo
3. La segunda ley de composición (•) es distributivo a izquierda y derecha respecto de la primera (+).

Si la segunda ley de composición es conmutativa, (A, +, • ) es Anillo Conmutativo

Ej. Anillo / anillo conmutativo:

Sea la estructura (Z, +, •)

*• es el producto ordinario*

*+ es la suma aditiva*

Se tiene que:

* (Z, +) es Grupo Abeliano
* (Z, •) es Grupo semigrupo
* • es doblemente distributivo respecto de +
* (Z, +, •) es anillo

(Z, •) además es conmutativo

(Z, +, •) es anillo conmutativo

Si (A, +, • ) es Anillo

Y además posee elemento neutro respecto de •

(A, +, • ) es Anillo con Unidad

Un Anillo con unidad cuyos elementos no nulos son inversibles se llama Anillo con división

1. (A, +) es Grupo Abeliano
2. (A – {0} (Se elimina el neutro del conjunto A), •) es Grupo
3. • es doblemente distributivo respecto de +

Si un Anillo con división es conmutativo, se llama Cuerpo

1. (A, +) es Grupo abeliano
2. (A – {0}, •) es Grupo abeliano
3. • es distributivo respecto de +

Ej.:

(R, +, •) donde + es la adición y • es el producto ordinario

# UNIDAD 4: ALGEBRA DE BOOLE

* Definición.
* Propiedades.
* Principio de dualidad.
* Puertas lógicas y circuitos booleanos.
* ***Minimización de circuitos.***
* Funciones booleanas.
* Diagrama de Karnaugh.

**Definición:**

Un ***algebra booleana B*** consiste en un conjunto *S* que contiene elementos distintos 0 y 1, operadores binarios + y ● en *S*, y un operador unitario ´ en S que satisface las siguientes leyes.

1. Leyes asociativas:

para todo .

1. Leyes conmutativas:

, para todo

1. Leyes distributivas:

para todo

1. Ley de neutro:

, para todo

1. Leyes de complementos:

, para todo

Si B es un álgebra booleana, se escribe *B = (S, +, ●, ', 0, 1).*

**Algebra de Boole en algebra de conjuntos:**

Sea U un conjunto universal y sea S = P (U). Si se definen las siguientes operaciones en S:

, ,

entonces es un álgebra booleana. El conjunto vacío ∅ asume el papel de 0 y el conjunto universal U hace el papel de 1. Si X, Y, y Z son subconjuntos de S, se cumplen estas propiedades:

1. Asociatividad:

para todo

1. Conmutatividad:

,

para todo

1. Distributiva:

para todo

1. Neutro, identidad

, para todo

1. Complementos

, para todo

Ej.:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ∪ | ∅ | {1} | {2} | {1,2} |
| ∅ | ∅ | {1} | {2} | {1,2} |
| {1} | {1} | {1} | {1,2} | {1,2} |
| {2} | {2} | {1,2} | {2} | {1,2} |
| {1,2} | {1,2} | {1,2} | {1,2} | {1,2} |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ∩ | ∅ | {1} | {2} | {1,2} |
| ∅ | ∅ | ∅ | ∅ | ∅ |
| {1} | ∅ | {1} | ∅ | {1} |
| {2} | ∅ | ∅ | {2} | {2} |
| {1,2} | ∅ | {1} | {2} | {1,2} |

**Algebra de Boole en lógica proposicional:**

Si son los operadores, entonces se cumplen las siguientes propiedades:

1. Leyes asociativas:

para todo

1. Leyes conmutativas:

, para todo

1. Leyes distributivas:

para todo

1. Ley de neutro:

, para todo

1. Leyes de complementos:

, para todo

Ej.:

|  |  |  |
| --- | --- | --- |
| ∨ | 0 | 1 |
| 0 | 0 | 1 |
| 1 | 1 | 1 |

|  |  |  |
| --- | --- | --- |
| ∧ | 0 | 1 |
| 0 | 0 | 0 |
| 1 | 0 | 1 |

**Propiedades:**

Sea B = (S, +, ●, ', 0, 1) un álgebra booleana. Las siguientes propiedades se cumplen.

1. Leyes de idempotencia:

, para todo .

1. Leyes de acotación:

, para todo .

1. Leyes de absorción:

, para todo .

1. Leyes de involución:

para todo .

1. Leyes de 0 y 1:

, .

1. Leyes de “De Morgan” para álgebras booleanas:

para todo .

**Principio de dualidad:**

El dual de una afirmación que incluye expresiones booleanas se obtiene intercambiando:

* 0 por 1;
* 1 por 0;
* + por ●;
* ● por +.

Observación:

Si dos expresiones booleanas son iguales, sus duales también lo son.

Ej.:

El dual de

Es

Cada condición en la definición de álgebra booleana incluye su dual. Por lo tanto, se tiene el siguiente resultado.

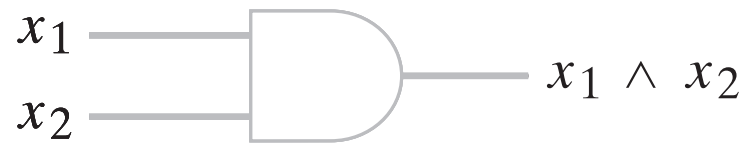
**Teorema:**

El dual de un teorema de álgebras booleanas también es un teorema.

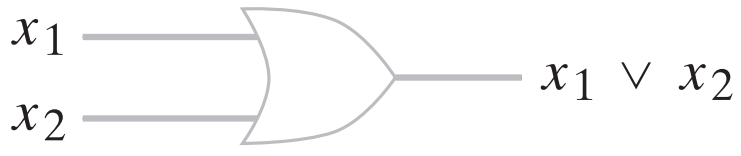
**Demostración**: se supone que *T* es un teorema de álgebras booleanas. Entonces existe una prueba *P* de *T* que involucra sólo la definición de un álgebra booleana. Sea *P’* la secuencia de afirmaciones obtenidas al sustituir cada enunciado en P por su dual. Entonces *P’* es una prueba del dual de *T*.

**Puertas lógicas:**

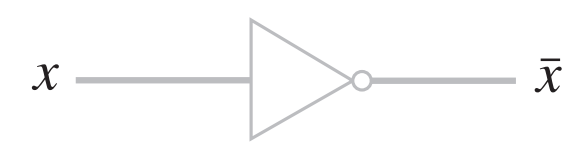
Compuerta **AND:** recibe entradas de bits y , esto produce una salida denotada por , que será 1 si y solo si ambos bits son 1



Compuerta **OR:** recibe entradas de bits y , y produce una salida denotada por , que será aceptada siempre que al menos un bit sea 1



Compuerta **NOT**: recibe solo un bit y produce una salida denotada que será 1 si el bit es 0, y 0 si el bit es 1



**Funciones booleanas**

Las funciones que se pueden representar por expresiones booleanas se llaman funciones booleanas. Son funciones cuyo dominio son las palabras conformadas por los valores binarios 0 o 1 ("falso" o "verdadero", respectivamente), y a imagen son también 0 y 1.

Son funciones de la forma , donde *B* = {0,1} y *n* un entero no negativo correspondiente a la aridad de la función.

Ej.:

Sea una expresión booleana. Una función de la forma

se llama función booleana.

Un **minitérmino** en los símbolos es una expresión booleana de la forma:

donde cada es ya sea

Es decir: el producto lógico de k variables

Ej.:

Una función booleana está definida en **forma normal disyuntiva** si su expresión se da como una suma de minitérminos o disyunciones de conjunciones.

Un **maxitérmino** en los símbolos es una expresión booleana de la forma:

donde cada es ya sea

Es decir: la suma lógica de k variables

Ej.:

Una función booleana está definida en **forma normal conjuntiva** si se expresa como un producto de maxitérminos o conjunciones de disyunciones.

**Mapa de Karnaugh:** diagrama utilizado para la simplificación de funciones algebraicas Booleanas. Consiste en una representación bidimensional de la tabla de verdad de la función a simplificar. Puesto que la tabla de verdad de una función de N variables posee 2n filas, el mapa K correspondiente debe poseer también 2n cuadrados

Ej.:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| *x* | *y* | *z* | *w* | *F* |
| 0 | 0 | 0 | 0 | **1** |
| 0 | 0 | 0 | 1 | **0** |
| 0 | 0 | 1 | 0 | **1** |
| 0 | 0 | 1 | 1 | **0** |
| 0 | 1 | 0 | 0 | **0** |
| 0 | 1 | 0 | 1 | **1** |
| 0 | 1 | 1 | 0 | **0** |
| 0 | 1 | 1 | 1 | **1** |
| 1 | 0 | 0 | 0 | **1** |
| 1 | 0 | 0 | 1 | **0** |
| 1 | 0 | 1 | 0 | **1** |
| 1 | 0 | 1 | 1 | **0** |
| 1 | 1 | 0 | 0 | **0** |
| 1 | 1 | 0 | 1 | **1** |
| 1 | 1 | 1 | 0 | **0** |
| 1 | 1 | 1 | 1 | **1** |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **-** | **x’y’** | **x’y** | **xy** | **xy’** |
| z’w’ | *1* | *0* | *0* | *1* |
| z’w | *0* | *1* | *1* | *0* |
| zw | *0* | *1* | *1* | *0* |
| zw’ | *1* | *0* | *0* | *1* |

# UNIDAD 5: GRAFOS

* Definición.
* Vértices, lados, grados, bucles, caminos.
* Circuitos.
* Circuitos eulerianos.
* Circuitos hamiltonianos.
* Caminos de longitud mínima.
* Matrices de adyacencia y grafos.
* Dígrafos

**Definición:**

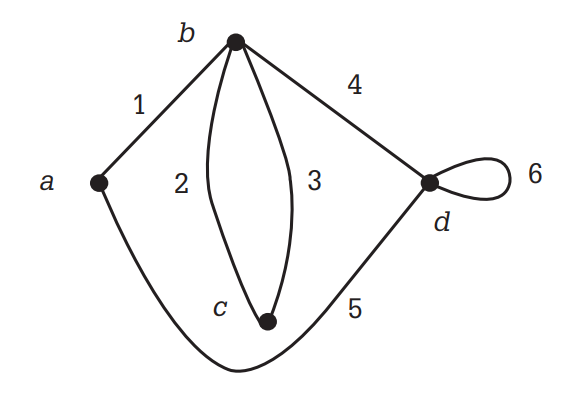
Un **grafo** G consiste en un conjunto V de vértices y un conjunto E de aristas tal que cada arista se asocia con un par no ordenado de vértices. Si existe una arista única asociada con los vértices u y w, se escribe . Aquí, (v, w) es una arista entre v y w en un grafo y no es par ordenado.

Ej.:

Sea V un conjunto finito no vacío de vértices

V: conjunto de vértices

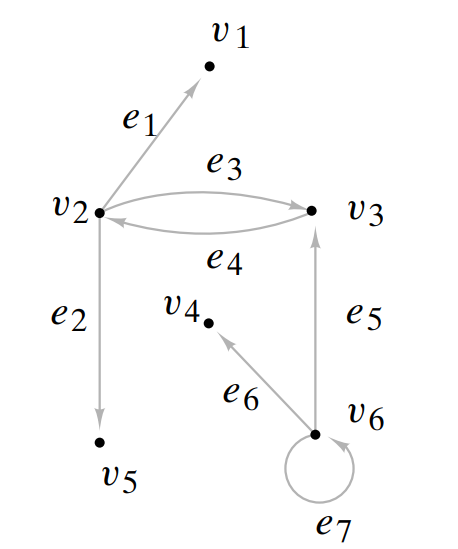
E: conjunto de aristas



Un **dígrafo** G consiste en un conjunto V de vértices y un conjunto E de aristas tales que cada arista está asociada a un par ordenado de vértices. Si hay una arista única asociada con el par ordenado (v, w) de vértices, se escribe , que denota una arista de v a w.

Ej.:

Sea V un conjunto finito no vacío de vértices



Una arista en un grafo (o dígrafo) que se asocia con el par de vértices v y w es **incidente sobre v y w**, y se dice que **v y w son incidentes sobre e** y son **vértices adyacentes**.

Si G es un grafo con vértices V y aristas E, se escribe .

**Vértices:** Se indican por medio de un pequeño círculo y se les asigna un número o letra.

**Aristas:** Son las líneas que unen un vértice con otro y se les asigna una letra, un número o una combinación de ambos.

**Lados paralelos:** Son aquellas aristas que tienen relación con un mismo par de vértices.

**Bucle:** Es aquella arista que sale de un vértice y regresa al mismo vértice.

**Grado de un vértice:** Es el número de lados que salen o entran a un vértice.

**Camino:**

Sean vértices de un grafo no dirigido .

Un **camino** x - y en G es una sucesión alternada finita:

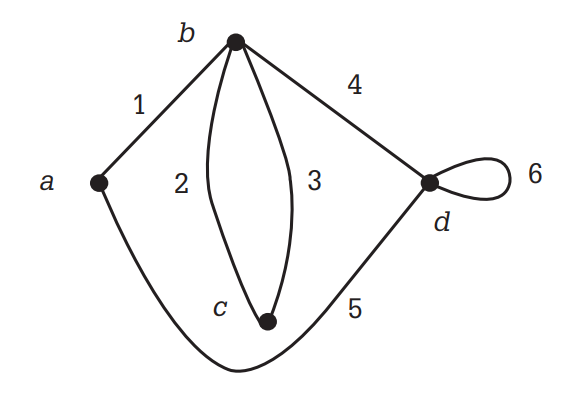
De vértices y aristas de G, y una elección de una arista entre y de forma que ninguna arista se elige más de una vez.

**Longitud del camino**: es el número de **n** aristas que hay en el camino (si n=0 no existen aristas, y el camino se llama trivial)

En un camino se pueden repetir aristas y vértices.

Cualquier camino donde (y ) es un **camino cerrado.** De lo contrario el **camino es abierto.**

Ej.:



Camino abierto / recorrido:

Camino cerrado / ciclo / circuito:

**Recorrido:** se da cuando en un camino x-y no se repite ninguna arista

**Circuito:** es un recorrido cerrado que parte y finaliza en el mismo vértice

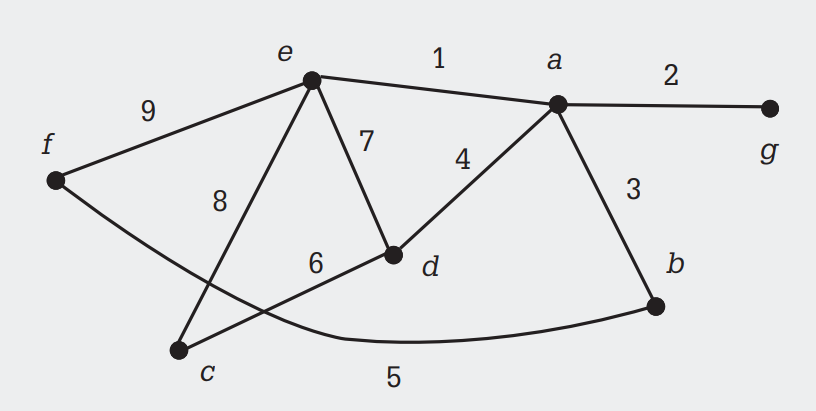
**Camino simple:** ocurre cuando no se atraviesa más de una vez ningún vértice

**Ciclo:** camino simple cerrado que inicia y termina en el mismo vértice

**Tipos de Grafos:**

**Grafo simple**. Es aquel que no tiene lazos ni lados paralelos.

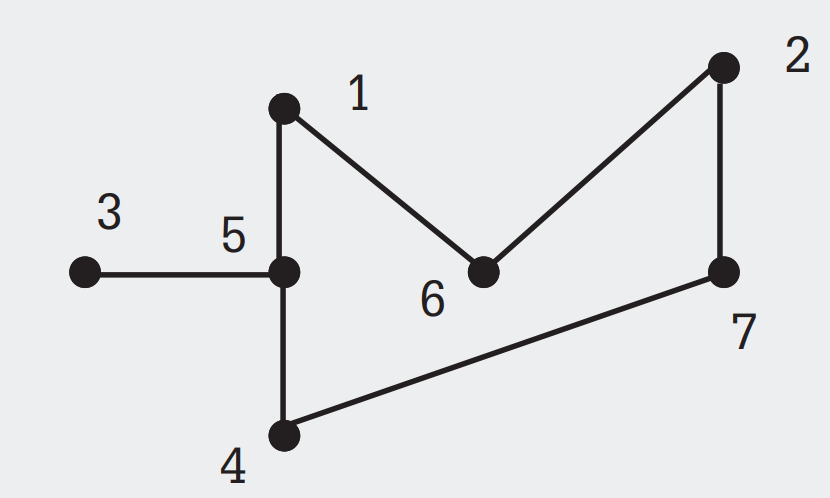
Ej.:



**Grafo bipartido:** Está compuesto por dos conjuntos de vértices A y B en donde los vértices del conjunto A se relacionan con los del B, pero entre los vértices de un mismo conjunto no existe arista que los una.

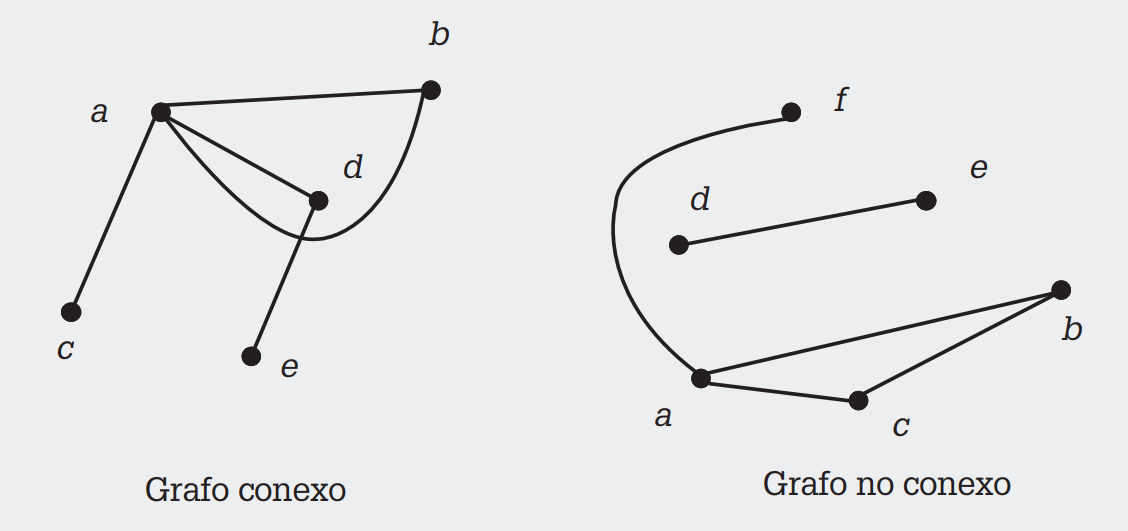
Ej.:

Sean los conjuntos de vértices A = {1, 2, 3, 4} y B = {5, 6, 7}



**Grafo conexo:** Para cualquier par de vértices w, x, distintos entre sí, existe un trayecto para ir de w a x. En el grafo conexo existe un camino para ir de un vértice a otro, en cambio, en el grafo no conexo hay vértices que no pueden ser accedidos.

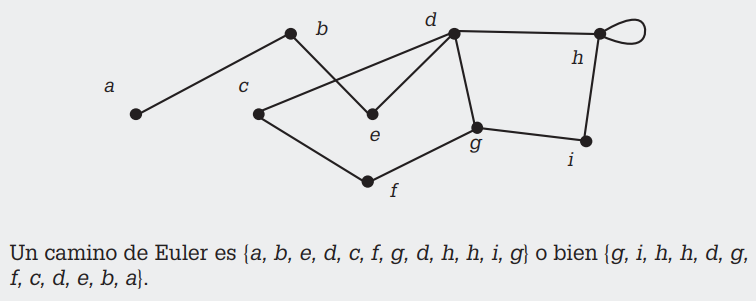
Ej.:



**Camino de Euler:** Camino que recorre todos los vértices pasando por todas las aristas solo una vez. Siempre comienza y termina en vértices con grado impar. Sí un grafo tiene más de dos vértices con valencia impar, entonces no puede tener un camino de Euler.

Aristas: 1 vez. Vértices: varias veces

Ej.:



**Circuito de Euler:** ciclo que recorre todos los vértices pasando por todas las aristas solo una vez.

Un grafo tiene circuito de Euler si y sólo si es conexo y todos sus vértices tienen valencia par.

**Algoritmo de Fleury:** permite determinar un circuito de Euler:

1) Verificar que el grafo sea conexo y que todos los vértices tengan valencia par.

2) Seleccionar un vértice para iniciar el recorrido.

3) Escoger una arista a partir del vértice actual. Esa arista no debe ser “lado puente”, a menos que no exista otra alternativa.

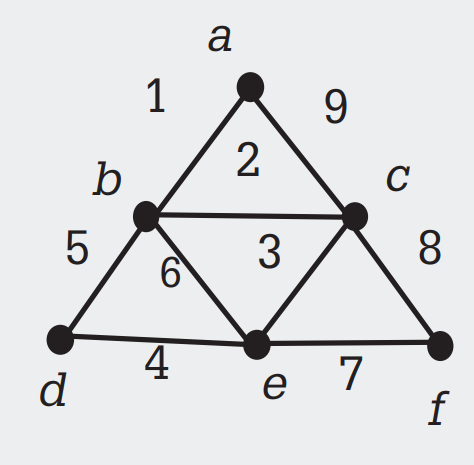
Lado puente es aquella arista que, si se elimina, los grafos pierden la propiedad de ser conexos.

4) Desconectar los vértices unidos por la arista seleccionada.

5) Si todos los vértices del grafo están desconectados, ya se tiene el circuito de Euler y finalizar. De otra manera continuar con el paso 3.

Ej.:

Circuito de Euler: (a, b, c, e, d, b, e, f, c, a)



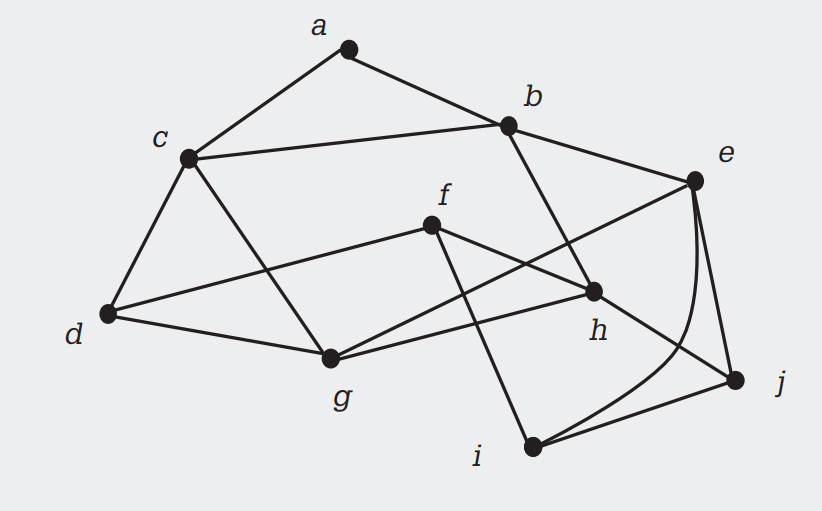
**Trayectoria hamiltoniana**: es una trayectoria que visita cada vértice solo una vez, con excepción del vértice inicial que puede ser también el último.

**Circuito de Hamilton:** circuito que visita cada vértice solo una vez, con excepción del vértice inicial que también es el último.

No hay forma de saber con anticipación si un grafo tiene o no un circuito de Hamilton.

Aristas: Varias veces. Vértices: 1 vez

Ej.:



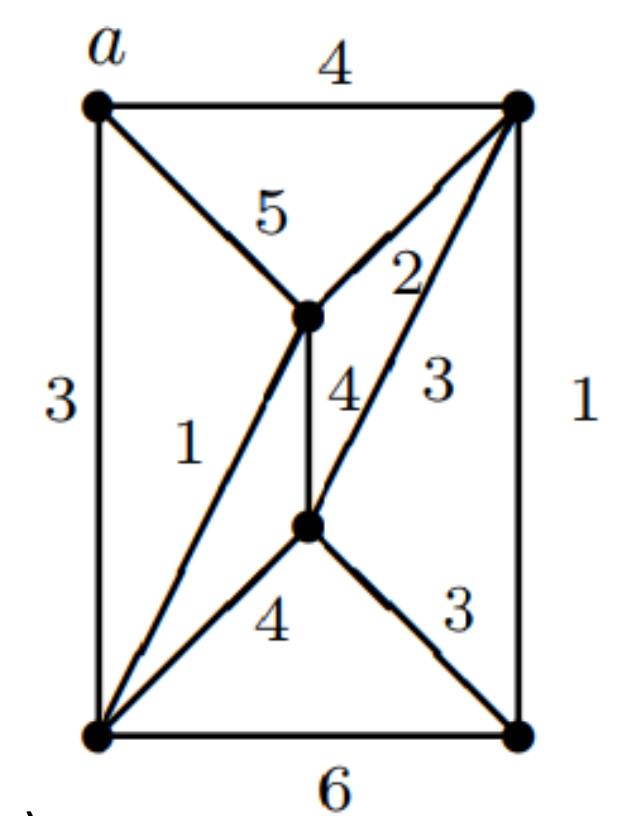
Circuito de Hamilton: {a, b, h, g, e, j, i, f, d, c, a}

**Grafos ponderados:** Son aquellos en donde a las aristas se les da un valor llamado peso y representa la distancia que hay de un vértice a otro.

**Longitud de una ruta:** es la suma de los pesos de las aristas en la ruta.

**Ruta de longitud mínima:** visita todos los vértices exactamente una vez y representa la ruta optima.

A veces es necesario encontrar el camino más corto para ir de un nodo a otro, para hallarlo se hace uso del **algoritmo de Dijkstra**

****

1. Se toma el nodo origen a
2. Usar una matriz que tenga como columnas las etiquetas de los nodos y los pasos para unir todos los vértices existentes, como se muestra en la siguiente tabla:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A |  |  |  |  |  |  |
| B |  |  |  |  |  |  |
| C |  |  |  |  |  |  |
| D |  |  |  |  |  |  |
| E |  |  |  |  |  |  |
| F |  |  |  |  |  |  |

1. Colocar en la columna del primer paso las distancias desde el vértice origen hacia las demás junto con la etiqueta del nodo al que se conectan. Los nodos que no tengan relación con el que se halla seleccionado se les coloca como distancia.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a |  |  |  |  |  |
| B | 4, a |  |  |  |  |  |
| C | 5, a |  |  |  |  |  |
| D | ∞ |  |  |  |  |  |
| E | 3, a |  |  |  |  |  |
| F | ∞ |  |  |  |  |  |

1. Identificar en la columna actual la menor distancia entre todas (la distancia desde el nodo origen hacia si mismo siempre es 0) para así dejar una marca definitiva, generalmente corchetes.

Sobre la fila del nodo seleccionado se colocan marcas en las demás columnas para indicar su descarte

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a |  |  |  |  |  |
| C | 5, a |  |  |  |  |  |
| D | ∞ |  |  |  |  |  |
| E | 3, a |  |  |  |  |  |
| F | ∞ |  |  |  |  |  |

1. Registrar en la columna del paso actual la distancia más corta que resulta de sumar la distancia registrada en el nodo actual + distancia a los vértices adyacentes a él, y seleccionar la distancia más corta cuyo nodo aún no esté seleccionado de esa columna de la matriz, luego marcar en el resto de la fila del nodo seleccionado para no volver a considerarlo, por tanto, la matriz será:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a | ∞ |  |  |  |  |
| C | 5, a | 4, e |  |  |  |  |
| D | ∞ | 7, e |  |  |  |  |
| E | 3, a | [3, a] | - | - | - | - |
| F | ∞ | 9, e |  |  |  |  |

1. Registrar en la columna del paso actual el vértice con la distancia más corta entre todos los nodos que no se han descartado hasta ahora. Además de marcar la fila de dicho nodo para no volverlo a elegir.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a | ∞ | 6, c |  |  |  |
| C | 5, a | 4, e | [4, e] |  |  |  |
| D | ∞ | 7, e | 8, c |  |  |  |
| E | 3, a | [3, a] | - | - | - | - |
| F | ∞ | 9, e | ∞ |  |  |  |

1. Si ya están todos los vértices seleccionados, finalizar. En caso contrario regresar al paso 5.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a | ∞ | 6, c |  |  |  |
| C | 5, a | 4, e | [4, e] | - | - | - |
| D | ∞ | 7, e | 8, c |  |  |  |
| E | 3, a | [3, a] | - | - | - | - |
| F | ∞ | 9, e | ∞ |  |  |  |

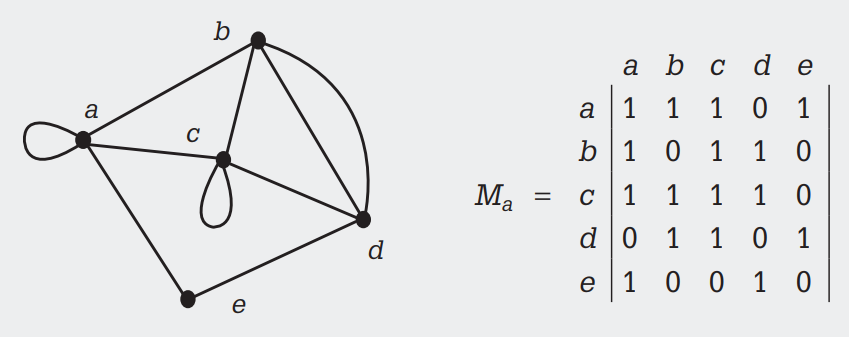
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a | ∞ | 6, c | [6, c] | - | - |
| C | 5, a | 4, e | [4, e] | - | - | - |
| D | ∞ | 7, e | 8, c | 9, b |  |  |
| E | 3, a | [3, a] | - | - | - | - |
| F | ∞ | 9, e | ∞ | 7, b |  |  |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | P1 | P2 | P3 | P4 | P5 | P6 |
| A | 0, a | - | - | - | - | - |
| B | 4, a | ∞ | 6, c | [6, c] | - | - |
| C | 5, a | 4, e | [4, e] | - | - | - |
| D | ∞ | 7, e | 8, c | 9, b | 10, f | [10, f] |
| E | 3, a | [3, a] | - | - | - | - |
| F | ∞ | 9, e | ∞ | 7, b | [7, b] | - |

**Matriz de adyacencia :** Matriz cuadrada en la cual los vértices del grafo se indican comofilas y columnas: el orden de los vértices es el mismo que guardanlas filas y columnas de la matriz. Se coloca un 1 como elemento de lamatriz cuando existe una relación entre uno y otro vértice, o bien un 0.

En esta matriz no se pueden representar aristas paralelas.

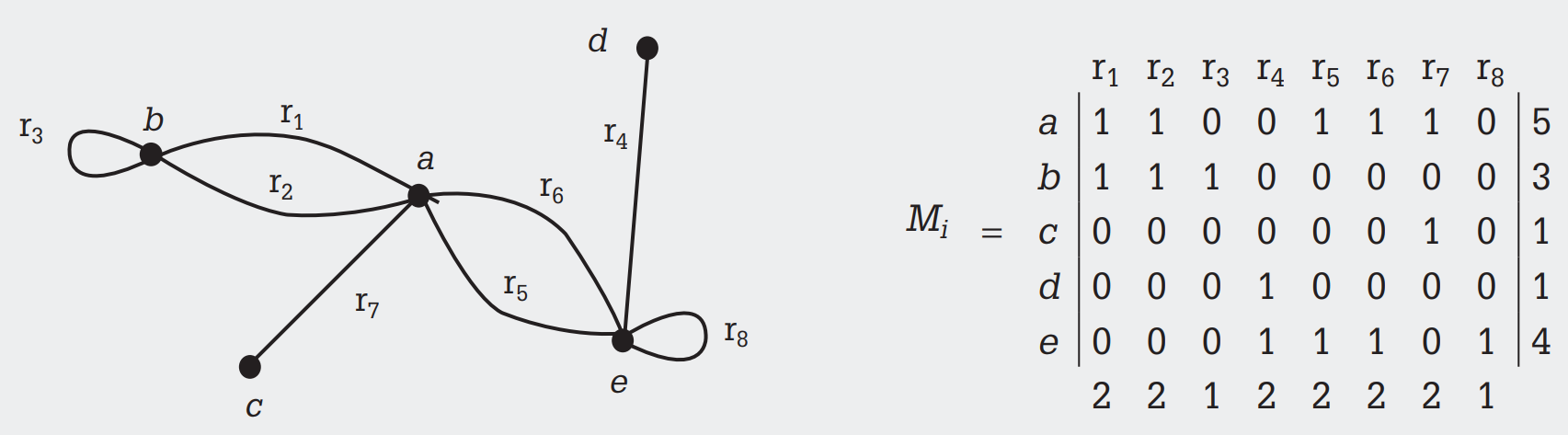
Ej.:

****

**Matriz de incidencia :**

En esta matriz se colocan los vértices del grafo como filas y las aristas como columnas, lo que permite representar aristas paralelas.

Ej.:



# UNIDAD 6: ARBOLES

* Arboles con raíz.
* Arboles etiquetados.
* Arboles binarios.
* Arboles generadores o de expansión.
* Búsquedas en árboles.
* Árbol de expansión mínima.
* Algoritmo de Prim.
* Algoritmo de Kruskal
* Arboles no dirigidos.

**Definición:** Un árbol es un grafo conexo que no tiene ciclos, lazos ni lados paralelos.

Un árbol está compuesto por **niveles** y el vértice más alto de la jerarquía se llama **raíz**. La raíz tiene un nivel 0.

**Árbol con raíz:**

Sea A un conjunto y T una relación en A:

T es un **árbol** si existe un vértice v, en A con la propiedad de que existe una única trayectoria en T de , hacia cualquier otro vértice en A, pero no existe una trayectoria hacia sí mismo.

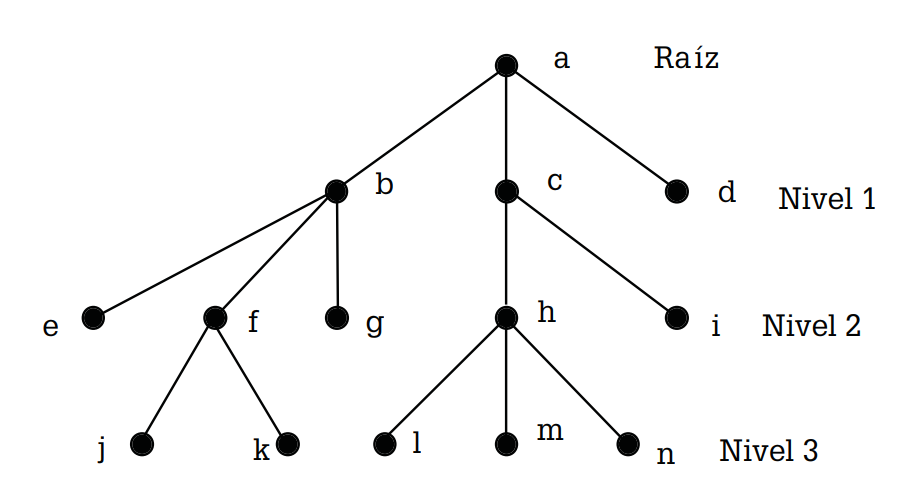
Se conoce a como la **raíz** del árbol T, entonces T es un **árbol con raíz**: Simbólicamente es

Un elemento de A es un **vértice en T**

Todo nodo (excepto la raíz) está vinculado a otro de mayor nivel que se llama **padre**, también cualquier nodo puede tener uno o más elementos relacionados en un nivel más bajo llamados **hijos.**

A los elementos que están en las puntas de las ramas se les llama **hojas**.

Ej.:



Aplicaciones: son esenciales para construir bases de datos y compiladores de lenguajes.

**Teorema 1**. Sea un árbol con raíz. Entonces

1. No existen ciclos en T.
2. es único y se llama *raíz del árbol* en T.
3. Cada vértice en T distinto de tiene grado interno uno, y tiene grado interno cero.

Demo:

1. Si existe un ciclo en T, que comienza y termina en el vértice . Se sabe que , y debe existir una trayectoria de a . Entonces es una trayectoria de a diferente de , lo que contradice la definición de árbol.
2. Si es otra raíz de T, existe una trayectoria de a y una trayectoria de a (ya que es una raíz). Entonces es un ciclo de a , lo que por definición es imposible. Por tanto, es la única raíz.
3. Sea , un vértice en T distinto de . Entonces existe una única trayectoria , de a en T. Esto significa que de modo que , tiene al menos grado interno uno. Si el grado interno de es mayor que uno, deben existir vértices , distintos tales que y se encuentran ambos en T. Si y existen trayectorias de a y de , a , por definición. Entonces y son dos trayectorias diferentes de a , y esto contradice la definición de un árbol con raíz . Por tanto, el grado interno de , es 1.

**Teorema 2**. Sea un árbol con raíz sobre un conjunto A. Entonces

1. T es arreflexivo:

Esto quiere decir que el árbol no puede contener bucles

1. T es asimétrico:

Si existe la relación padre-hijo entre y no puede haber una relación de regreso

Significa que no puede existir una arista que vuelva desde hacia

1. T es atransitivo:

Si es padre de , y es padre de , no puede ser padre de , no puede existir un “atajo” de hacia

**Teorema 3**:

Si es un árbol con raíz y , entonces:

* también es un árbol con raíz .
* es el **subárbol** de T que comienza en v.

Demo:

Existe una trayectoria de a cualquier otro vértice en , por definición de .

Si existe un vértice en tal que:

Existen dos trayectorias distintas y de a y si es la trayectoria en de a .

Entonces:

y serían dos trayectorias distintas en de a .

Esto es imposible porque:

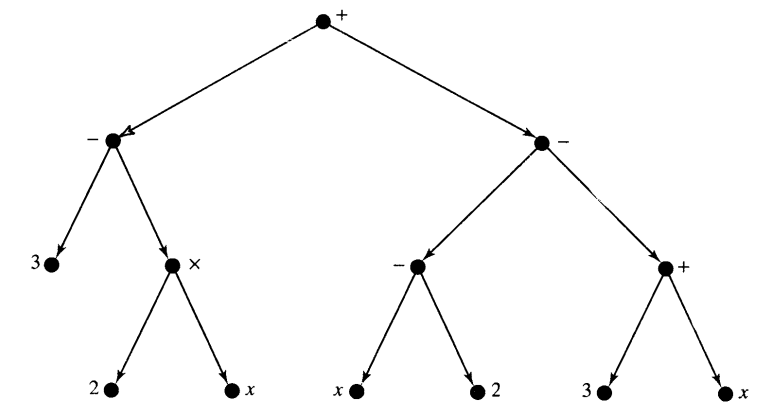
* es un árbol con raíz .
* Cada trayectoria de hacia otro vértice en debe ser único.
* Si es un ciclo en en , es también un ciclo en .
* Esto contradice la definición de árbol, por tanto, no puede existir.

Esto implica que es un árbol con raíz .

**Árbol etiquetado:**

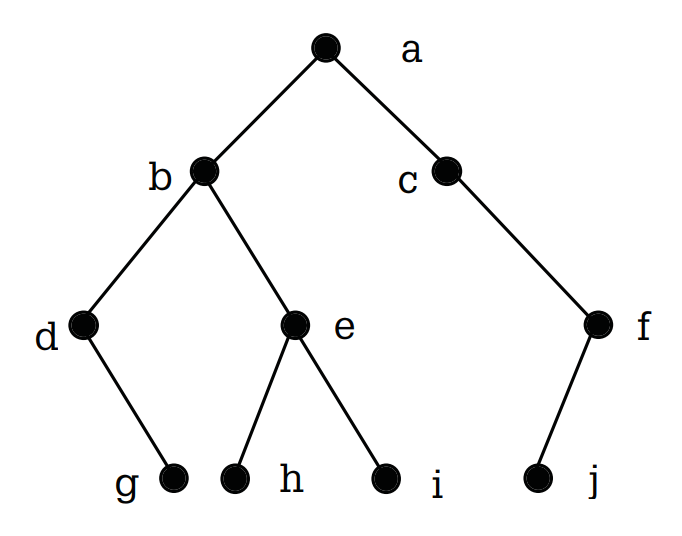
Es útil etiquetar los vértices o aristas de un dígrafo para indicar su uso para un propósito. Los conjuntos de vértices de los árboles no son importantes, sino que la utilidad del árbol se enfatiza mediante las etiquetas de los vértices.

Ej.:



**Árbol binario:** cada nodo tiene como máximo dos hijos, esto es, el nodo puede tener dos ramas, una o ninguna, pero nunca puede tener más de dos.

Ej.:



**Árbol de expansión/generador:** se obtiene a partir de la eliminación de aristas redundantes en un grafo conexo y sin ciclos, manteniendo todos sus vértices unidos.

Para obtener un árbol generador se pueden usar 2 métodos:

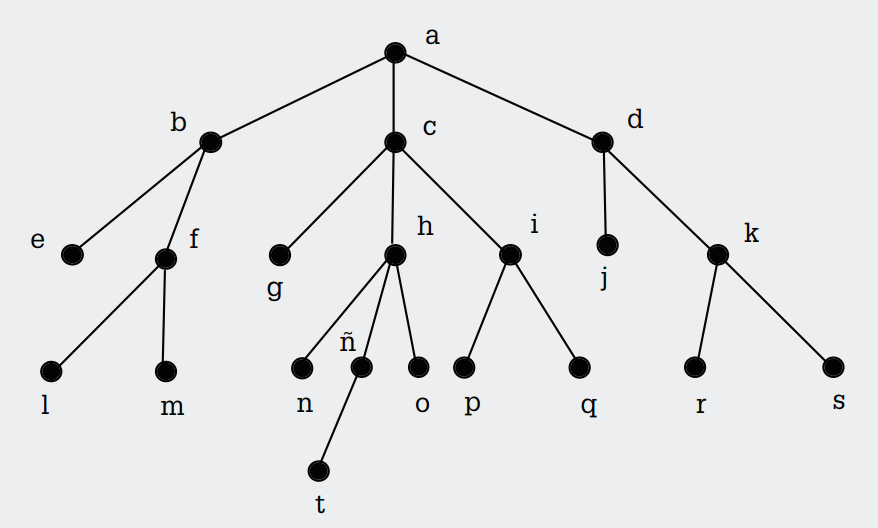
* búsqueda en profundidad
* búsqueda a lo ancho

**Búsqueda a lo ancho:**

Se comienza en la raíz y se examinan todos los hijos de esta de izquierda a derecha. Si la información buscada no está en ese nivel, se busca en el siguiente hasta encontrar la información.

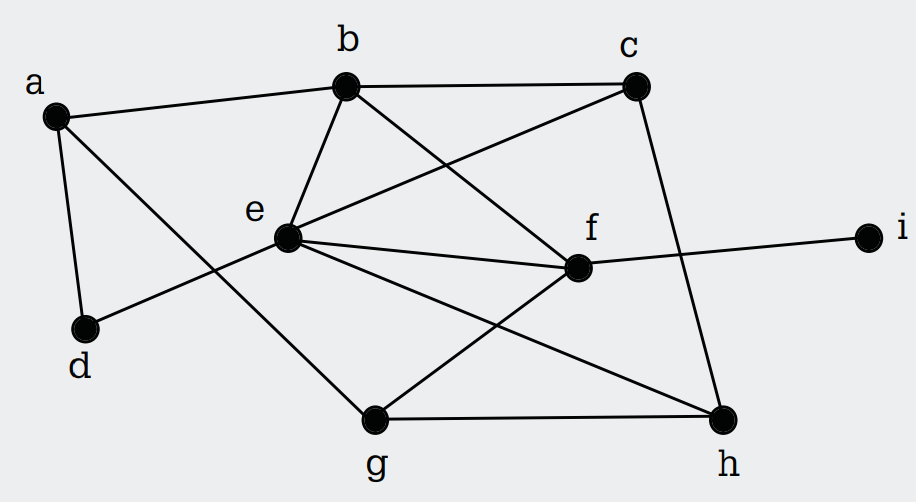
Es útil cuando los árboles están balanceados o tienen pocos niveles en relación a su información.

Ej.:

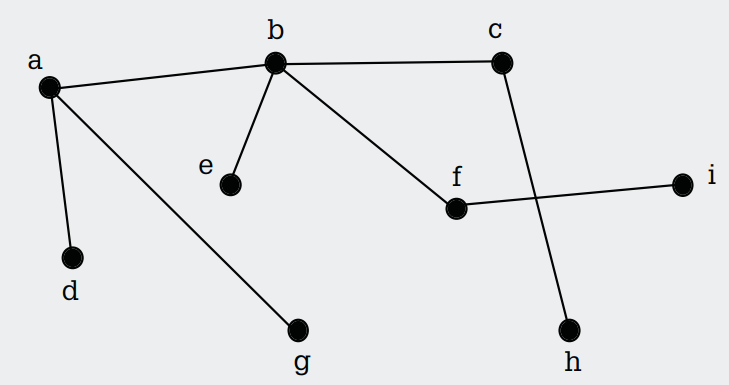


Si se buscara la letra “g”, antes encontrarla, se debería pasar por todos los niveles 0 y 1, además de los nodos “e” y “f” del nivel 2.

Ej. 2: dado el siguiente grafo



Se obtiene el siguiente árbol



**Búsqueda en profundidad:**

Se comienza en el nodo raíz

Se busca en el hijo de la izquierda y si tiene hijos se continúa con el de la izquierda hasta llegar a la parte más baja del árbol.

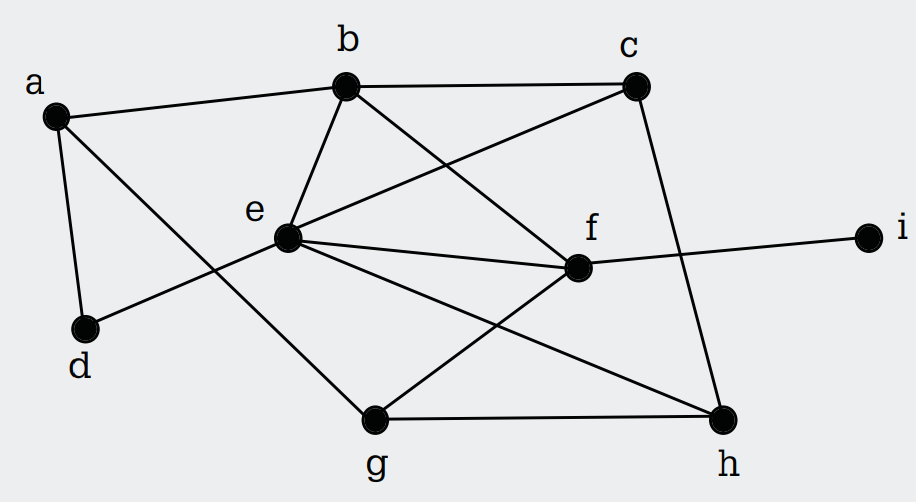
Si este nodo ya no tiene hijo izquierdo, se continúa con el hijo de más a la derecha hasta llegar a la hoja.

Si no se ha encontrado la información, se recorre el camino andado hasta el nodo inmediato anterior que tenga hijos y no hayan sido inspeccionados.

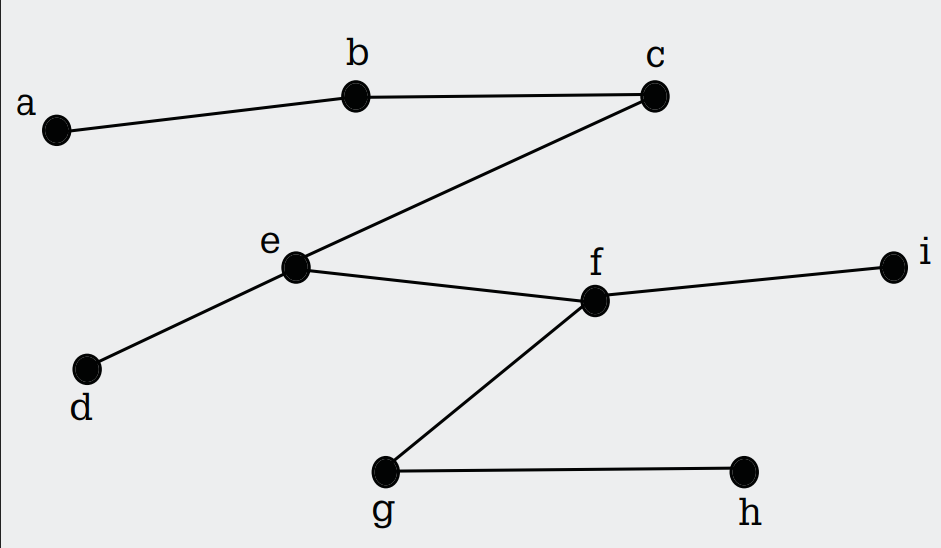
Cuando se llega nuevamente a la hoja, se regresa hasta el nodo inmediato anterior que tenga hijos sin inspeccionar y así sucesivamente.

Si la raíz tiene hijos que no se inspeccionaron se selecciona el nodo más cercano al hijo izquierdo y sus descendientes.

Ej.: dado el siguiente grafo



Se obtiene el siguiente árbol



**Árbol generador mínimo:** árbol generador de un grafo conexo y sin ciclos, que mantiene unidos a todos los vértices de la forma más barata posible.

Para obtener el árbol generador mínimo en un grafo conexo ponderado se puede utilizar el algoritmo de Prim o de Kruskal:

**Algoritmo de Prim:** Se toma una arista de menor costo posible y se construye el árbol con aristas adyacentes con la misma propiedad.

Los vértices se dividen en dos conjuntos:

* Vértices integrados : forman parte del árbol generador mínimo—
* Vértices no integrados .

El conjunto **árbol** tiene todas las aristas que se integran en cada iteración

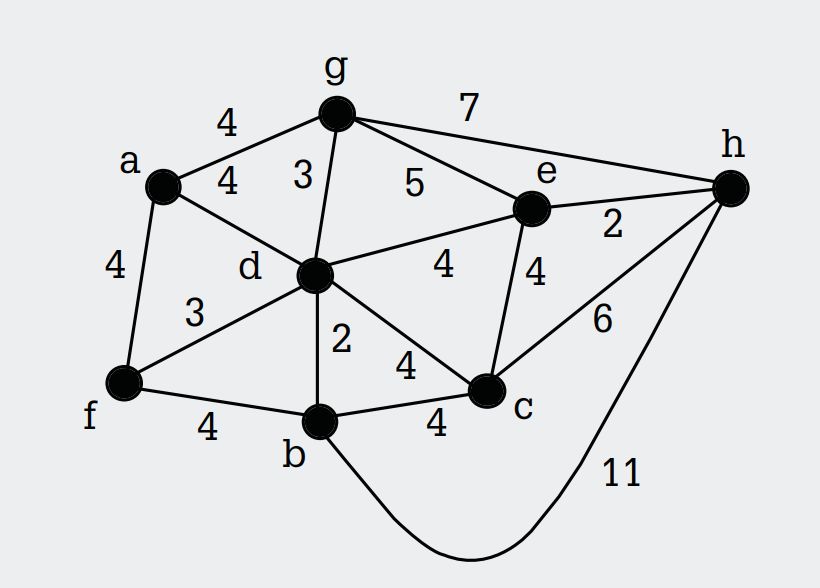
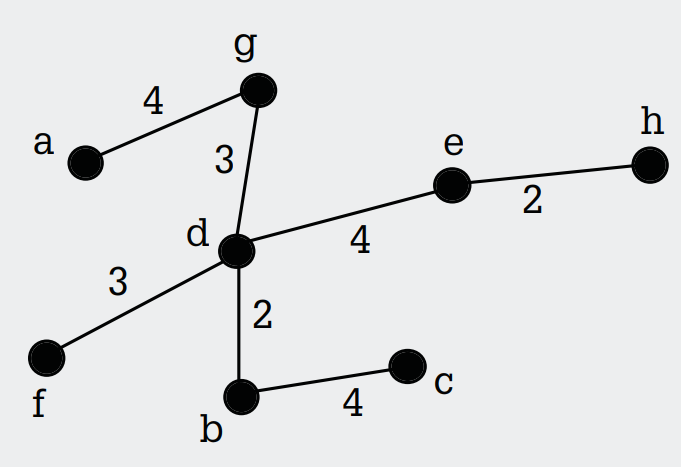
En cada paso se agrega un vértice en el conjunto mientras que el conjunto es disminuido en uno.

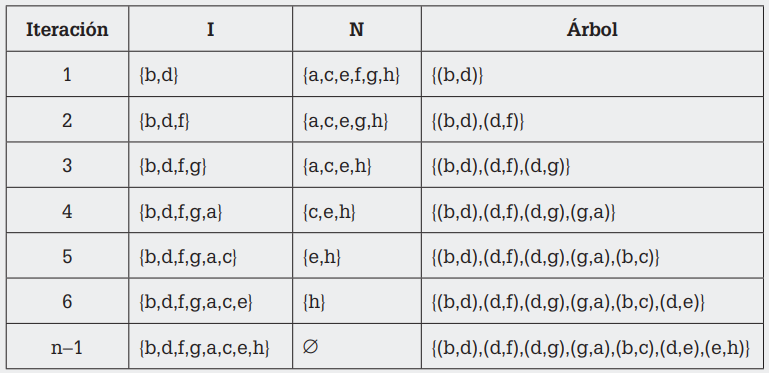
1. Se selecciona una arista de costo mínimo .
2. Se integran los vértices de esa arista al conjunto , serán los primeros nodos que conforman el árbol generador mínimo.
3. Todos los demás nodos pertenecen al conjunto .
4. El Árbol tiene como primera arista , esto es, .
5. Mientras que iterar. En caso contrario terminar.
6. Seleccionar nueva arista con costo mínimo . Una característica es que debe estar contenido en mientras que debe estar en .
7. Se agrega al árbol la arista mínima seleccionada:

.

1. Se agrega al conjunto el nuevo nodo seleccionado:
2. Se elimina el nodo seleccionado del conjunto , esto es, .
3. Regresar al paso 5.

Ej.:

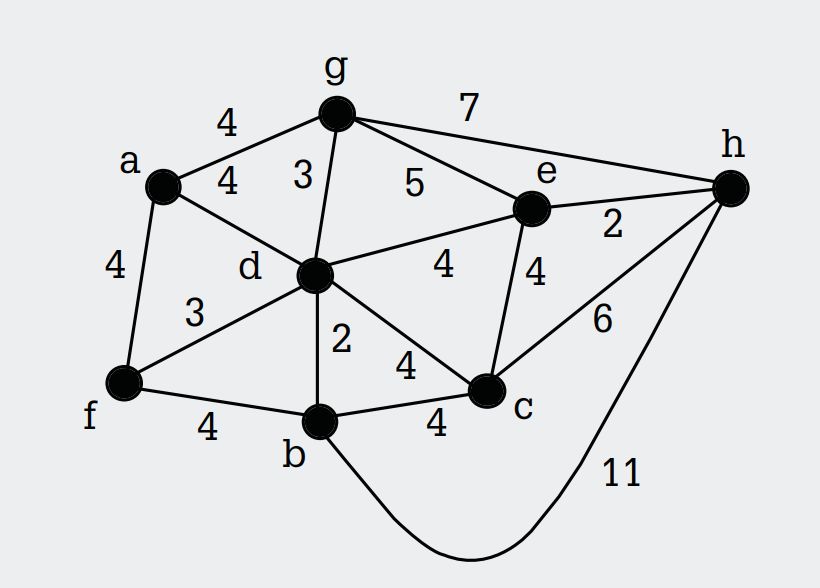
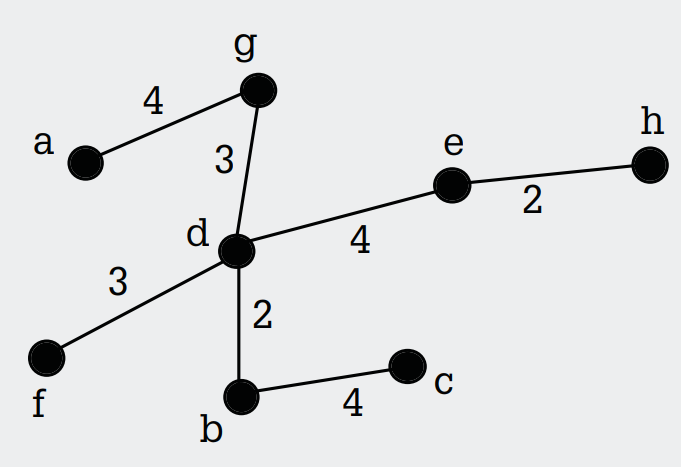
 

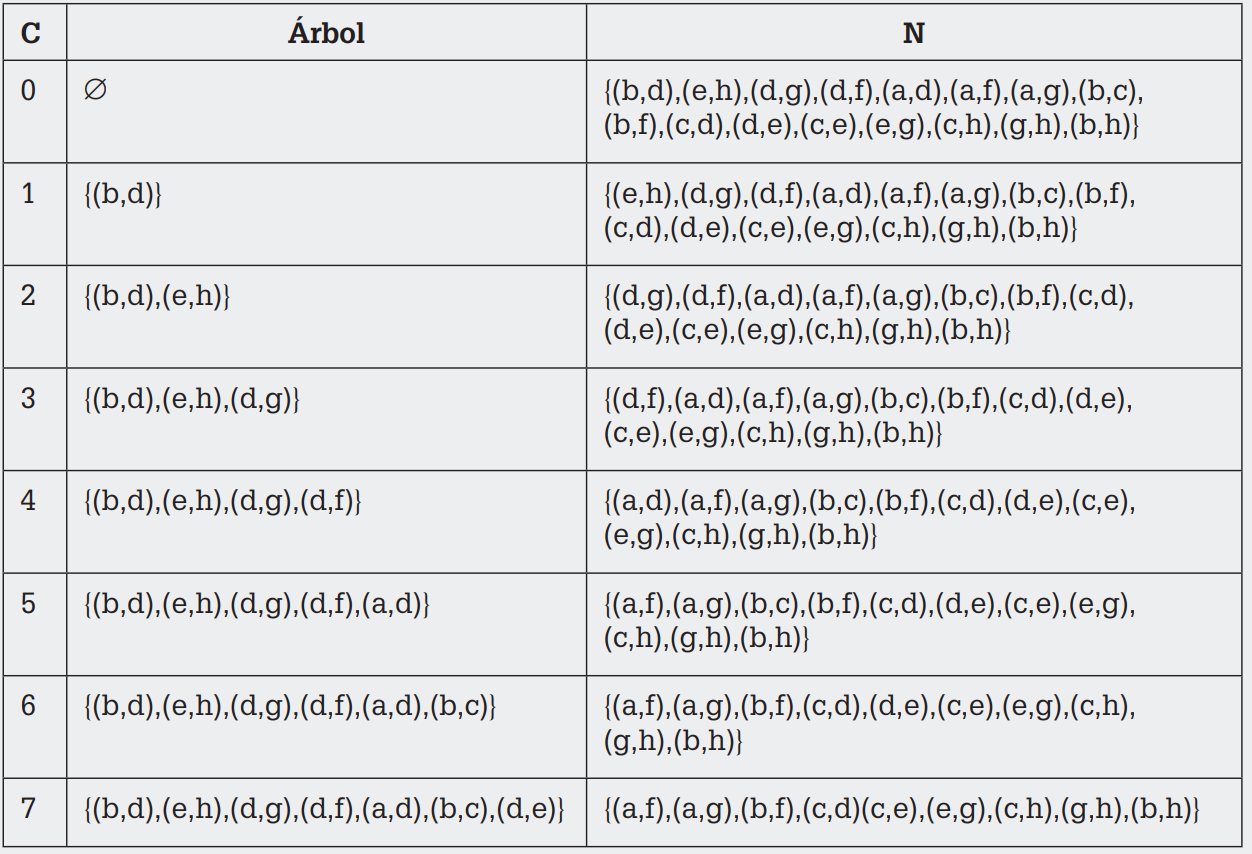


**Algoritmo de Kruskal:** integra al árbol generador mínimo las aristas de menor costo, procurando no formar ciclos.

1. Ordenar los costos de las aristas del grafo en forma ascendente y colocar en el conjunto N las aristas, de acuerdo a este orden.
2. Se incluye la arista con menor costo .
3. Se resta del conjunto la arista seleccionada .
4. Se registra en un contador el número de aristas incluidas, en este caso , ya que sólo se ha incluido una arista.
5. Mientras iterar. En caso contrario finalizar.
6. Si no forma ciclos, entonces:
7. Regresar a paso 4.

Ej.:



**Recorrido de un árbol:**

Pre-orden:

* Visitar raíz
* Buscar en el subárbol/Hijo izquierdo si existe
* Buscar en el subárbol/Hijo derecho si existe

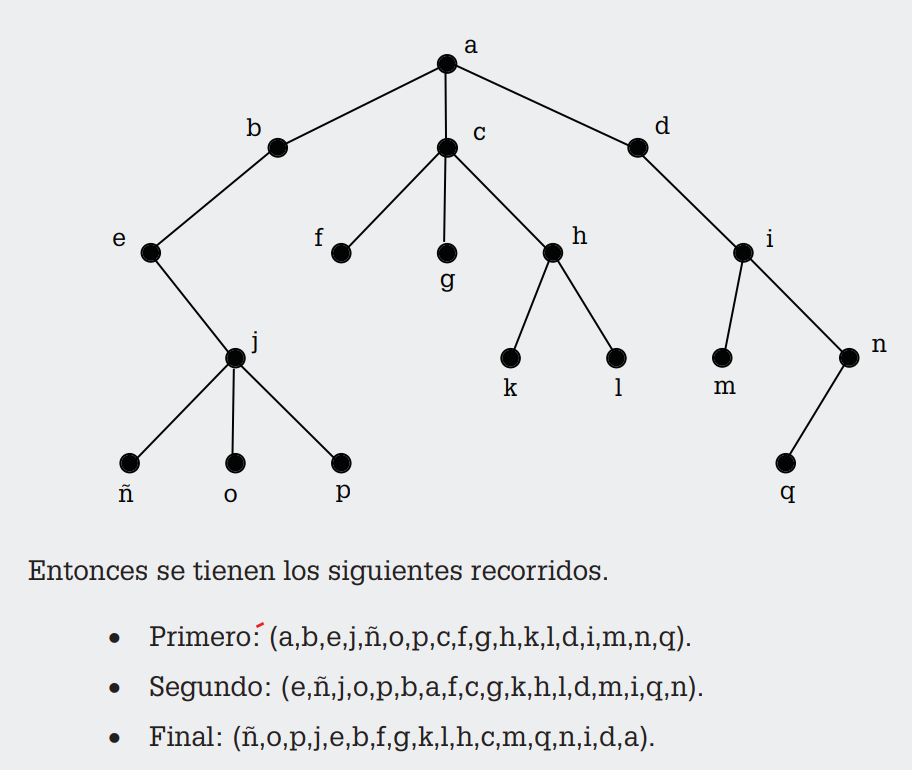
Entre-orden:

* Buscar en el subárbol/Hijo izquierdo si existe
* Visitar raíz
* Buscar en el subárbol/Hijo derecho si existe

Posorden:

* Buscar en el subárbol/Hijo izquierdo si existe
* Buscar en el subárbol/Hijo derecho si existe
* Visitar raíz

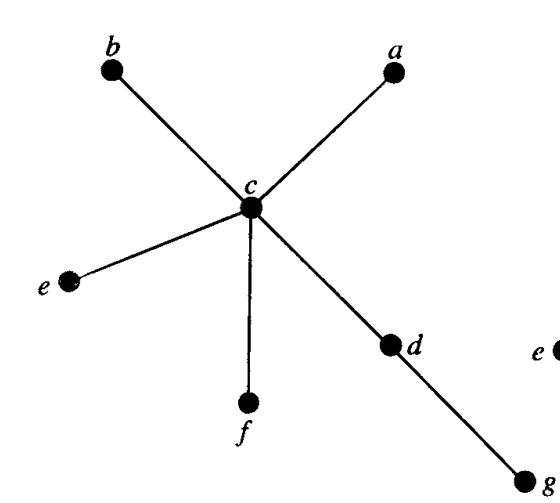
Ej.:



**Árbol no dirigido:** árbol cuyas aristas son todas bidireccionales.

La grafica de un árbol no dirigido tiene una sola línea sin flechas que une los vértices y siempre que . El conjunto es una arista no dirigida de .

Ej.:



Teorema 1: Sea R una relación simétrica en un conjunto A. Entonces las siguientes proposiciones son equivalentes:

1. R es un árbol no dirigido
2. R es conexo y acíclico

Demo:

Supongamos que es un árbol no dirigido.

Observemos primero si , se debe tener o . Esto significa que cada arista no dirigida en la gráfica de R aparece en el dígrafo de .

Por contradicción, se mostrará que no tiene ciclos simples:

Supongamos que R tiene un ciclo simple . Para cada arista en , se elige la pareja o que esté en . Resulta en una figura cerrada con aristas en , donde cada arista debe apuntar en una dirección.

Ahora existen tres posibilidades:

* Todas las flechas apuntan en sentido horario
* Todas apuntan en sentido antihorario
* Algún nodo es apuntado por otros dos nodos distintos

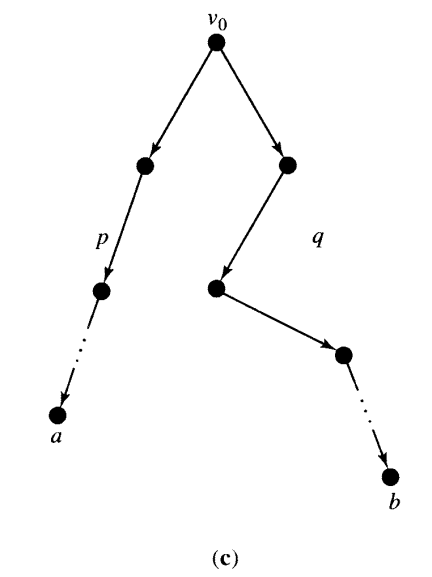
La tercera opción es imposible, ya que cada vértice tiene grado interno 1. Pero cualquiera de los otros dos casos significaría que T tiene un ciclo, esto también es imposible.

*Así, la existencia del ciclo lleva a una contradicción y por tanto es imposible.*

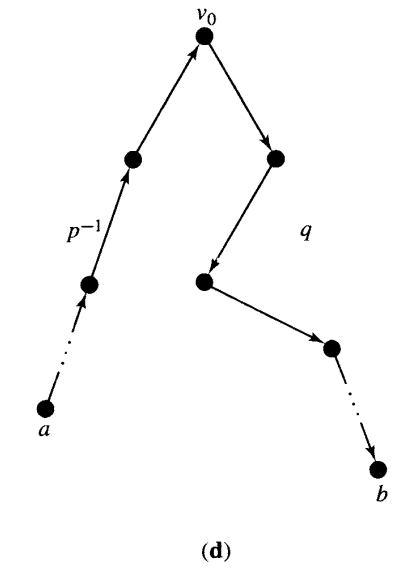
También se debe mostrar que es conexa.

Sea la raíz del árbol .

Entonces, si y son dos vértices en , deben existir trayectorias de a y de a . **(8.25.c)**



Ahora, todas las trayectorias en son reversibles en , de modo que **(8.25.d)**, conecta con en , donde es la trayectoria inversa de .



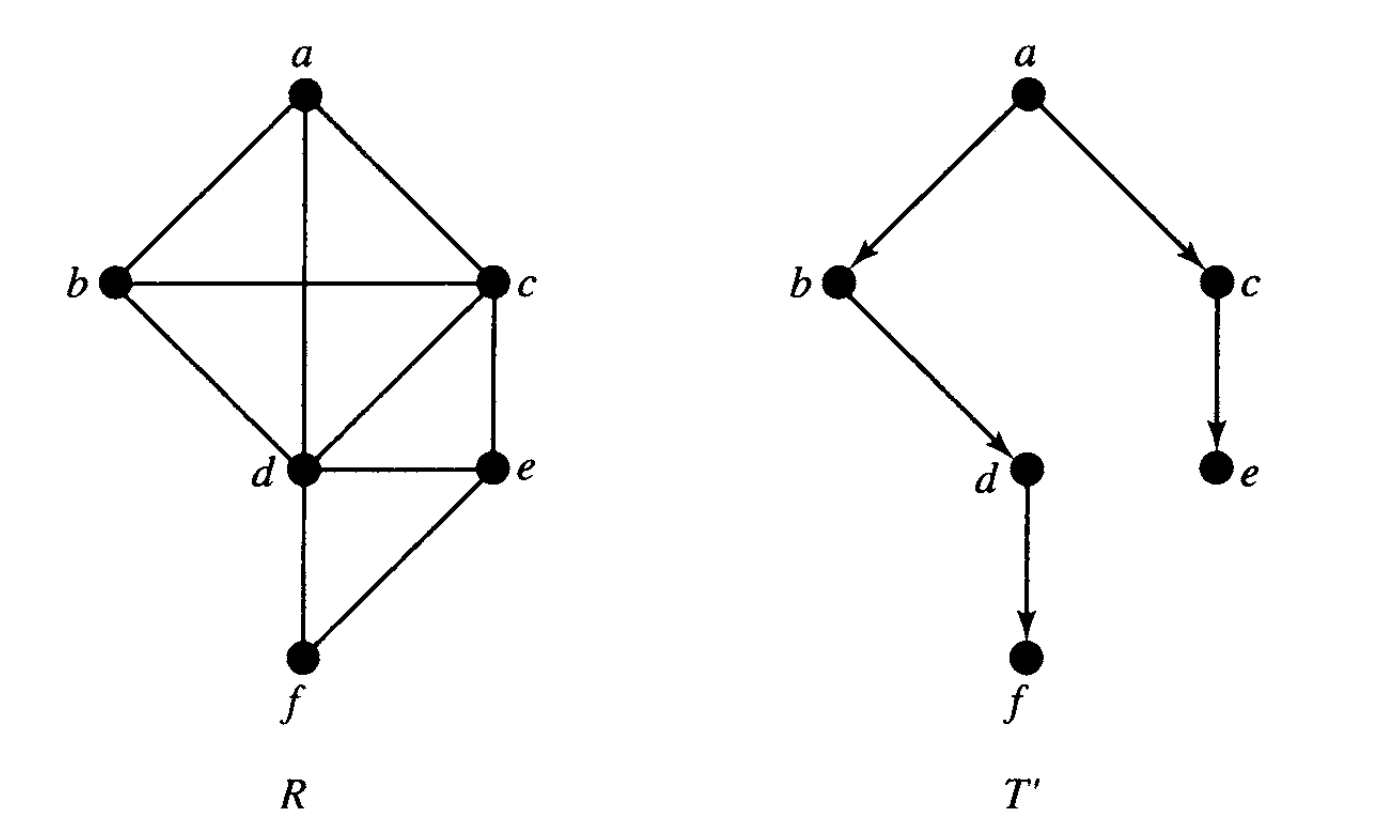
*Como y son arbitrarios, es conexa*.

**Arboles de expansión de relaciones conexas:**

Si es una relación simétrica conexa sobre un conjunto :

Un árbol es un **árbol de expansión** para si es un árbol con los mismos vértices que y se puede obtener eliminando algunas aristas de

Ej.:



# UNIDAD 7.1: LENGUAJES FORMALES

* Definición de lenguajes formales.
* Alfabeto.
* Gramáticas.
* Tipos de gramáticas.
* Forma de Backus-Naur.

Sea A un conjunto finito:

Un **lenguaje formal** sobre es un subconjunto de , es decir el conjunto de todas las cadenas sobre .

Además, sus símbolos son primitivos y las reglas para unir esos símbolos están formalmente especificadas. Este está conformado por cadenas de símbolos y cada cadena debe tener un significado “preciso e inequívoco”.

Un lenguaje posee: Vocabulario/alfabeto y Gramática

**SIMBOLO:** Es la representación abstracta de un objeto. En general, es cualquier carácter que represente algún elemento.

Ej.: 1, a, p.

**ALFABETO O VOCABULARIO:** Es un conjunto finito de símbolos. Debe existir al menos un símbolo en el alfabeto, es decir, el alfabeto no puede ser el conjunto vacío. Notación: Este conjunto se denota con .

Ej.: {a, b, c, d}

Un **lenguaje** está basado en una gramática y reglas para crear palabras propias del lenguaje.

La **gramática** es el sistema que permite establecer las reglas que han de aplicarse a un lenguaje. Aquí se tiene que:

* : conjunto finito de símbolos no terminales o auxiliares para formar cadenas (en letra mayúscula).
* : conjunto finito de símbolos terminales que forman las cadenas del lenguaje (en letra minúscula).
* : subconjunto finito del producto , llamado conjunto de producciones

: lenguaje con todas las cadenas que se pueden formar con el alfabeto

* : símbolo inicial .

Ej.:

**Desarrollo:**

son cadenas del lenguaje L(G)

**Expresión general:**

Este lenguaje presenta estas características:

* Está formado por todas las cadenas que comienzan con un símbolo “a”
* Terminan en un símbolo “c”
* En el medio tienen, o no, un número par de símbolos “b”.

**Jerarquía de Chomsky:**

**Tipo 0 (Gramática sin restricciones):** cuando no existe restricción para las composiciones de G.

Por ser muy generales pierden fuerza descriptiva.

Su importancia se centra en que una producción compuesta por las cadenas posiblemente vacías y a izquierda y derecha del símbolo no terminal , deriva en otra producción que reemplaza la por la cadena vacía , simbolicamente:

Entonces:

La cadena de deducción se contrajo.

Ej.:

**Tipo 1 (Gramática sensible al contexto):** si para cualquier composición de la gramática G, la longitud de símbolos de la izquierda de la composición es menor o igual a la longitud de símbolos de la derecha .

Es decir:

Sus producciones se basan en la derivación de una composición de cadenas posiblemente vacías y , y un símbolo no terminal , hacia otra producción con diferente a la cadena vacía. Simbólicamente:

La longitud de las cadenas no decrece.

*Es muy complicada y difícil de analizar y estudiar debido a la libertad que se tiene para formar palabras de un lenguaje.*

Ej.:

**Tipo 2 (Gramática libre de contexto):** si el lado izquierdo de cada composición es un símbolo no terminal y el lado derecho consta de uno o más símbolos terminales y/o no terminales.

No contiene producciones de tipo “cadena posiblemente vacía - símbolo no terminal - cadena posiblemente vacía” que deriva en una cadena , donde la cadena solo aparecerá si a le precede , y le sucede .

*Este tipo de gramáticas se usa para la creación de lenguajes formales o lenguajes de programación.*

*Tiene relación con los autómatas finitos, autómatas de pila y máquinas de Turing.*

Ej.:

**Tipo 3 (Gramática regular):** si el lado izquierdo de la composición es un símbolo no terminal y el lado derecho tiene uno o más símbolos terminales, incluyendo a lo sumo un símbolo no terminal.

Posee reglas simples de sustitución y generación de palabras de un lenguaje.

*Esta gramática tiene relación con los autómatas finitos.*

Ej.:

**NOTACION BNF (FORMA DE BACKUS-NAUR)**

Una alternativa que poseen las gramáticas de tipo 2 (y tipo 3), para desplegar sus producciones.

1. Se agrupan todas las producciones con un mismo no terminal del lado izquierdo (por ejemplo, ), en la misma línea, colocando el no terminal referido () del lado izquierdo y del lado derecho todos los terminales y no terminales de la producción, en serie.

Cuando un mismo no terminal implica diferentes cadenas de símbolos se puede comprimir la información usando el carácter “|” (se lee “o”)

1. El símbolo “” se reemplaza por el símbolo “” que significa “puede ser”.
2. Cada símbolo no terminal será encerrado entre paréntesis agudos .

Esto permite que los símbolos no terminales tengan espacios dentro de ellos.

Ej.:

Generar -1024:

# UNIDAD 7.2: MAQUINAS Y AUTOMATAS DE ESTADO FINITO

**Definición:**

Una **máquina de estado finito** es un modelo abstracto de una máquina con una memoria primitiva capaz de entregar una salida en función a las señales de entrada y lo que “paso antes”

**Formalmente:**

Una máquina de estado finito consiste en un séxtuple:

1. Un conjunto finito de símbolos de entrada.
2. Un conjunto finito de símbolos de salida.
3. Un conjunto finito de estados.
4. Una función del siguiente estado de *f: I x S → S*.
5. Una función de salida de *g: I x S → O*.
6. Un estado inicial .

Se escribe .

Ej.:

0/1

0/1

0/1

1/0

1/0

1/0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Estado | Función de transición de estados (f) | | Función de salida (g) | |
| Entradas | | | |
|  | **0** | **1** | **0** | **1** |
| **S0** | S2 | S1 | 1 | 0 |
| **S1** | S2 | S1 | 1 | 0 |
| **S2** | S2 | S1 | 1 | 0 |

**MEF como modelo de sistema físico:**

* Redes de conmutación de circuito (acción electromecánica)
* Sistema nervioso humano (acción electroquímica)

**MEF con salida asociada a la transición (Maquina de Mealy)**:

Tipo de máquina de estados finitos que genera una salida basándose en su estado actual y una entrada.

Ej.:

1/0

1/0

1/0

0/1

0/1

0/1

**MEF con salida asociada al estado (Maquina de Moore):**

Es una MEF para la cual, la salida en un momento depende de su estado en ese momento, mientras la transición al siguiente estado depende del estado en que se encuentre y de la entrada introducida.

Ej.:

0

0

0

1

1

1

Ambas maquinas se definen con la séxtupla

**Transformar máquina de Mealy a máquina de Moore**:

Para cada máquina de Mealy (salida asociada a la transición)

Existe una Máquina de Moore (salida asociada al estado) equivalente

Para cada combinación estado/entrada

Con:

Al estado le corresponde la función de salidas:

La transición queda determinada por:

Ej.:

**s0**

**S1**

**S2**

0/a

0/a

0/b

1/b

1/b

1/a

**Tabla de nuevos estados:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S** | **S0** | | **S1** | | **S2** | |
| **I** | **0** | **1** | **0** | **1** | **0** | **1** |
|  | S0 | S2 | S0 | S1 | S0 | S1 |
|  | A | B | A | B | B | A |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **S’** | S0a | S2b | S0a | S1b | S0b | S1a |

**Tabla de transiciones y salidas:**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **S’** | **S0a** | **S0b** | **S1a** | **S1b** | **S2b** |
| **G’** | A | B | A | b | b |
|  | S0a/a | S0b/b | S1a/a | S1b/b | S2b/b |

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| I | **0** | **1** | **0** | **1** | **0** | **1** | **0** | **1** | **0** | **1** |
| **F’** | S0a | S2b | S0a | S2b | S0a | S1b | S0a | S1b | S0b | S1a |

**Resultado Final**:

0

0

0

0

0

1

1

1

1

1

**Autómata de Estado Finito:**

Máquina de estado finito en la que el conjunto de salida es {0, 1} y donde el estado actual determina la última salida.

Aquellos estados para los que la última salida fue 1 se llaman estados de aceptación

Formalmente:

Autómata de Estado Finito es un séxtuple donde:

1. conjunto finito de señales de entradas
2. conjunto de señales de salidas *O =* {0,1}
3. conjunto finito de estados
4. estado inicial
5. función de transición de estados
6. función de salida

Ej.:

1

1

1

0

0

0

|  |  |  |  |
| --- | --- | --- | --- |
| Estados | Entradas | | Salidas |
| 0 | 1 |
| S0 | S1 | S0 | 0 |
| S1 | S1 | S2 | 0 |
| S2 | S1 | S0 | 1 |

Las cadenas aceptadas deben contar al menos con un 1 al final

Cualquier lenguaje reconocido por un Autómata de Estado Finito, es un Lenguaje de Estado Finito.

Los lenguajes generados con Gramática tipo 3 son aceptados por los AEF.

**Tabla de transición:**

Se usa para representar los valores que puede tomar la función .

Mediante esta tabla, el estado inicial y los de aceptación se puede saber si una cadena pertenece o no a un lenguaje.

**Autómatas finitos deterministas:**

Un autómata finito es determinista si por medio de la función de transición se puede determinar cuál es el estado siguiente.

AFD es el siguiente: donde:

1. conjunto finito de señales de entradas
2. conjunto de señales de salidas *O =* {0,1}
3. conjunto finito de estados
4. estado inicial
5. función de transición de estados
6. función de salida

**Autómatas finitos no deterministas:**

La principal diferencia con un autómata finito determinista es que en el AFN la función de estado siguiente no conduce a un estado único.

Formalmente:

1. es un conjunto finito de símbolos de entrada
2. es el conjunto de símbolos de salida
3. es un conjunto finito de estados
4. Un estado inicial .
5. Una función de transición de estados cuya imagen es el conjunto de partes de S.
6. Una función (función de salida)

Ej.:

b

b

a

b

|  |  |  |  |
| --- | --- | --- | --- |
| Estados | Entradas | | Salidas |
| a | b |
| S0 | S1 | S0 | 0 |
| S1 | ∅ | S1, S2 | 0 |
| S2 | ∅ | ∅ | 1 |

**Equivalencia entre AFD y AFND:**

Esta siempre será posible y dado uno de ellos se podrá obtener el autómata equivalente. Por tanto, se puede asegurar que los AFD y los AFND reconocerán el mismo lenguaje.

**Convertir AFND a un AFD:**

* Se puede imaginar que en un instante la máquina se encuentra en un estado dado, donde para un símbolo de entrada, se enfrenta a transiciones alternativas y elige al azar su siguiente estado.

Para saber si una cadena es aceptada se debe suponer que la máquina opera un número de veces suficiente para probar cada ruta.

* La máquina se divide en copias idénticas que, en forma simultánea exploran los caminos alternativos
* En un instante, la máquina se encuentra en una combinación de estados. Esta idea conduce a obtener un AEFD por cada AEFND

Ej.:

Dado el siguiente AFND:

b

b

a

b

El conjunto de estados será ahora P(S) donde:

Los estados de aceptación en el nuevo autómata son todos aquellos que contienen el estado de aceptación del AFND (S2 en este caso).

|  |  |  |  |
| --- | --- | --- | --- |
| **Estados** | **a** | **b** | **Salida** |
| ∅ | ∅ | ∅ | 0 |
| {s0} | {S1} | {s0} | 0 |
| {S1} | ∅ | {S1S2} | 0 |
| {S2} | ∅ | ∅ | 1 |
| {S0S1} | {S1} | {s0S1S2} | 0 |
| {S0 S2} | {S1} | {s0} | 1 |
| {S1S2} | ∅ | {S1S2} | 1 |
| {S0S1S2} | {S1} | {S0S1S2} | 1 |

b

b

a

b

a/b

a/b

a

a

a

b

a

b

b

a

Se pueden eliminar los estados {S2}, {S0, S1}, {SO, S2}, {S0, S1, S2} ya que nunca podrán ser alcanzados, entonces la AFD queda:

b

a

b

a/b

a

a

b

**Relaciones entre gramáticas y autómatas:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Tipo | Lenguaje | Autómata | Normas de producción de gramáticas | Ejemplos |
| 0 | Sin restricciones | Máquina de Turing |  | describe una máquina de Turing} |
| 1 | Sensible a contexto | Autómata linealmente acotado (de cinta) |  |  |
| 2 | Independiente del contexto | Autómata a pila |  |  |
| 3 | Regular | Autómata finito |  |  |
| Significado de los símbolos:   * = terminal * = no terminal * = cadena de terminales y/o no terminales   + = cadena posiblemente vacía   + = cadena no vacía | | | | |