



Teoría de Grafos

Juan David Rojas Gacha

2020 - II





Árboles de expansión mínima

Grafo ponderado

Un **grafo ponderado** o **grafo con peso** es un grafo con etiquetas numéricas en las aristas denominadas **pesos de las aristas**.





Árboles de expansión mínima

Grafo ponderado

Un **grafo ponderado** o **grafo con peso** es un grafo con etiquetas numéricas en las aristas denominadas **pesos de las aristas**.

Árbol de expansión mínima

Un **árbol de expansión mínima** en un grafo ponderado conexo es un árbol de expansión que tiene la menor suma de los pesos de sus aristas.



Algoritmo de Kruskal

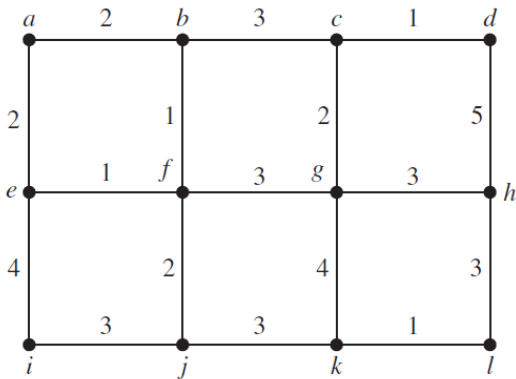
Input: Un grafo ponderado conexo G con n vértices.

Output: Un árbol de expansión mínima T .

Iteración:

1. $V(T) = V(G)$
2. $E(T) = \emptyset$
3. Mientras T sea desconexo
 - a. Seleccione e una arista en G con peso mínimo que une dos componentes de T .
 - b. $T = T \cup \{e\}$





Algoritmo de Prim

Input: Un grafo ponderado conexo G .

Output: Un árbol de expansión mínima T .

Iteración:

1. $V(T) = \{x\}$
2. $E(T) = \emptyset$
3. Para $i = 1$ hasta $n - 1$
 - a. Seleccione $e = uv : u \in V(T), v \notin V(T)$ una arista en G con peso mínimo.
 - b. $V(T) = V(T) \cup \{v\}$
 - c. $E(T) = E(T) \cup \{e\}$



Problema de ruta mínima

Distancia mínima

La **distancia mínima** $d(u, z)$ en un grafo ponderado es la mínima suma de los pesos de las aristas de un u, z -camino. También se denomina la **longitud mínima**.



Problema de ruta mínima

Distancia mínima

La **distancia mínima** $d(u, z)$ en un grafo ponderado es la mínima suma de los pesos de las aristas de un u, z -camino. También se denomina la **longitud mínima**.

Problema de ruta mínima

El **problema de ruta mínima** en un grafo ponderado consiste en encontrar el camino de longitud mínima entre un par de vértices (u, z) .



Algoritmo de Dijkstra

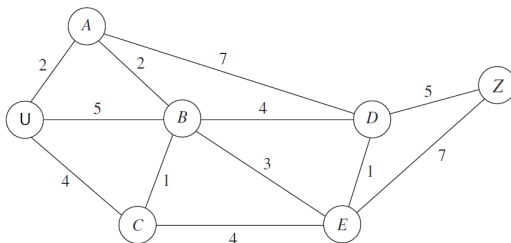
Input: Un grafo o digrafo ponderado G con pesos no negativos. $w(u, v)$ es el peso de la arista (u, v) , sea $w(u, v) = \infty$ si $(u, v) \notin E(G)$.

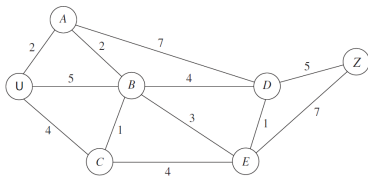
Output: $L(z)$ la distancia mínima de u a z .

Iteración:

1. $L(u) = 0$
2. Para todos los vértices $v \neq u$:
 - a. $L(v) = \infty$
3. $S = \emptyset$
4. Mientras $z \notin S$
 - a. Seleccione un vértice $x \notin S$ con $L(x)$ mínimo.
 - b. $S = S \cup \{x\}$
 - c. Para todo $v \notin S$
 1. $L(v) = \min\{L(v), L(x) + w(x, v)\}$

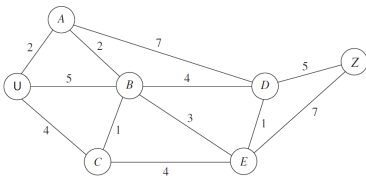






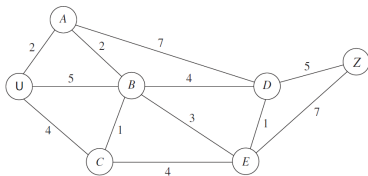
U	A	B	C	D	E	Z





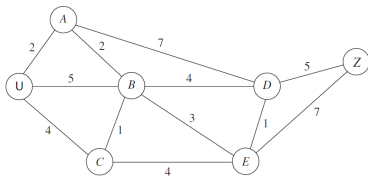
U	A	B	C	D	E	Z
0	∞	∞	∞	∞	∞	∞





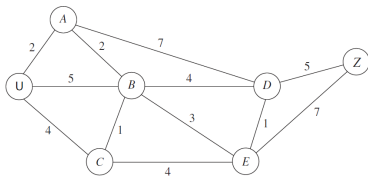
U	A	B	C	D	E	Z
①	∞	∞	∞	∞	∞	∞
—	② ^U	5 ^U	4 ^U	∞	∞	∞





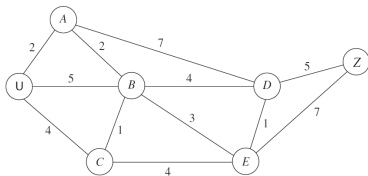
U	A	B	C	D	E	Z
①	∞	∞	∞	∞	∞	∞
—	② ^U	5 ^U	4 ^U	∞	∞	∞
—	—	④ ^A	④ ^U	9 ^A	∞	∞





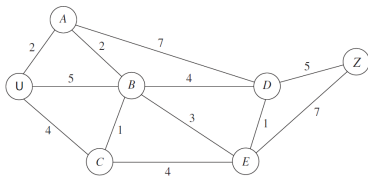
U	A	B	C	D	E	Z
①	∞	∞	∞	∞	∞	∞
—	② ^U	5 ^U	4 ^U	∞	∞	∞
—	—	④ ^A	④ ^U	9 ^A	∞	∞
—	—	—	—	8 ^B	⑦ ^B	∞





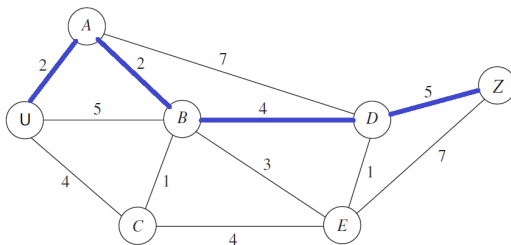
U	A	B	C	D	E	Z
①	∞	∞	∞	∞	∞	∞
—	② ^U	5 ^U	4 ^U	∞	∞	∞
—	—	④ ^A	④ ^U	9 ^A	∞	∞
—	—	—	—	8 ^B	⑦ ^B	∞
—	—	—	—	⑧ ^{B,E}	—	14 ^E





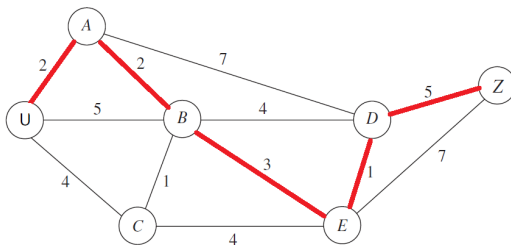
U	A	B	C	D	E	Z
①	∞	∞	∞	∞	∞	∞
—	② ^U	5 ^U	4 ^U	∞	∞	∞
—	—	④ ^A	④ ^U	9 ^A	∞	∞
—	—	—	—	8 ^B	⑦ ^B	∞
—	—	—	—	⑧ ^{B,E}	—	14 ^E
—	—	—	—	—	—	⑬ ^D





$U - A - B - D - Z$





U – A – B – E – D – Z



Algoritmo de Floyd - Warshall

Input: Un grafo o digrafo ponderado G con pesos no negativos.

Output: L_n matriz de distancia mínima cuya entrada ij representa la longitud del camino más corto entre los vértices v_i y v_j .

Iteración:

$$1. W_{ij} := \begin{cases} w(i, j) & \text{si } v_i \text{ y } v_j \text{ son ayacentes} \\ 0 & \text{si } v_i \text{ y } v_j \text{ no son ayacentes} \end{cases}$$

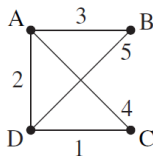
$$2. L_0 := \begin{cases} \infty & \text{si } w_{ij} = 0, i \neq j \\ w_{ij} & \text{en otro caso} \end{cases}$$

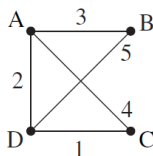
3. Para $k = 1$ hasta $n(G)$

a. $L_k = (l_k(i, j))$ donde

$$l_k(i, j) = \min\{l_{k-1}(i, j), l_{k-1}(i, k) + l_{k-1}(k, j)\}$$

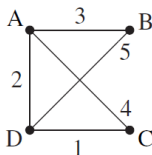






$$W = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 0 & 5 \\ 4 & 0 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$





$$W = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 0 & 5 \\ 4 & 0 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_0 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & \infty & 5 \\ 4 & \infty & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$





$$L_1 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$





$$L_1 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_3 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$



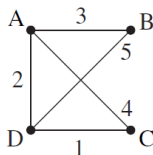
$$L_1 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_2 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_3 = \begin{bmatrix} 0 & 3 & 4 & 2 \\ 3 & 0 & 7 & 5 \\ 4 & 7 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$

$$L_4 = \begin{bmatrix} 0 & 3 & 3 & 2 \\ 3 & 0 & 6 & 5 \\ 3 & 6 & 0 & 1 \\ 2 & 5 & 1 & 0 \end{bmatrix}$$





$$\begin{bmatrix} - & AB & ADC & AD \\ BA & - & BDC & BD \\ CDA & CDB & - & CD \\ DA & DB & DC & - \end{bmatrix}$$





Código Huffman

Código de longitud fija

Un **código de longitud fija** es un código en el que cada carácter es codificado con una cadena de bits de longitud fija.

- El código ASCII (American Standard Code for Information Interchange) representa 128 caracteres usando cadenas de longitud 7.
- El código ASCII extendido representa 256 caracteres usando cadenas de longitud 8.



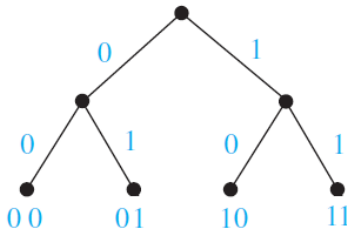


Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char	Byte	Cod.	Char
00000000	0	Null	00100000	32	Spc	01000000	64	@	01100000	96	.
00000001	1	Start of heading	00100001	33	!	01000001	65	A	01100001	97	a
00000010	2	Start of text	00100010	34	"	01000010	66	B	01100010	98	b
00000011	3	End of text	00100011	35	#	01000011	67	C	01100011	99	c
00000100	4	End of transmit	00100100	36	\$	01000100	68	D	01100100	100	d
00000101	5	Enquiry	00100101	37	%	01000101	69	E	01100101	101	e
00000110	6	Acknowledge	00100110	38	&	01000110	70	F	01100110	102	f
00000111	7	Audible bell	00100111	39	'	01000111	71	G	01100111	103	g
00001000	8	Backspace	00101000	40	(01001000	72	H	01101000	104	h
00001001	9	Horizontal tab	00101001	41)	01001001	73	I	01101001	105	i
00001010	10	Line feed	00101010	42	*	01001010	74	J	01101010	106	j
00001011	11	Vertical tab	00101011	43	+	01001011	75	K	01101011	107	k
00001100	12	Form Feed	00101100	44	,	01001100	76	L	01101100	108	l
00001101	13	Carriage return	00101101	45	-	01001101	77	M	01101101	109	m
00001110	14	Shift out	00101110	46	.	01001110	78	N	01101110	110	n
00001111	15	Shift in	00101111	47	/	01001111	79	O	01101111	111	o
00010000	16	Data link escape	00110000	48	0	01010000	80	P	01110000	112	p
00010001	17	Device control 1	00110001	49	1	01010001	81	Q	01110001	113	q
00010010	18	Device control 2	00110010	50	2	01010010	82	R	01110010	114	r
00010011	19	Device control 3	00110011	51	3	01010011	83	S	01110011	115	s
00010100	20	Device control 4	00110100	52	4	01010100	84	T	01110100	116	t
00010101	21	Neg. acknowledg.	00110101	53	5	01010101	85	U	01110101	117	u
00010110	22	Synchronous idle	00110110	54	6	01010110	86	V	01110110	118	v
00010111	23	End trans. block	00110111	55	7	01010111	87	W	01110111	119	w
00011000	24	Cancel	00111000	56	8	01011000	88	X	01111000	120	x
00011001	25	End of medium	00111001	57	9	01011001	89	Y	01111001	121	y
00011010	26	Substitution	00111010	58	:	01011010	90	Z	01111010	122	z
00011011	27	Escape	00111011	59	;	01011011	91	[01111011	123	{
00011100	28	File separator	00111100	60	<	01011100	92	\	01111100	124	
00011101	29	Group separator	00111101	61	=	01011101	93]	01111101	125	}
00011110	30	Record Separator	00111110	62	>	01011110	94	^	01111110	126	~
00011111	31	Unit separator	00111111	63	?	01011111	95	_	01111111	127	Del



Nota

Un código de longitud fija para 2^n caracteres se construye con un árbol binario completo de altura n , cada hoja almacena una cadena de longitud n .



Código Huffman

Un **código Huffman** es un código óptimo de longitud variable que codifica los caracteres en función de su frecuencia.

- Los caracteres de mayor frecuencia son codificados con cadenas de menor longitud.
- Los caracteres de menor frecuencia son codificados con cadenas de mayor longitud.
- La longitud esperada de una cadena es $\sum p_i l_i$ donde el i -ésimo carácter tiene probabilidad p_i y su código tiene longitud l_i .



Entropía

Sea $X = \{x_1, \dots, x_n\}$ un experimento formado por un número finito de eventos x_i con probabilidades p_1, \dots, p_n . X es una variable aleatoria discreta en un alfabeto \mathcal{A} . La **entropía** $H(X)$ de la variable aleatoria X se define como

$$H(X) := - \sum_{x_i \in X} p_i \log_{|\mathcal{A}|} p_i.$$



Entropía

Sea $X = \{x_1, \dots, x_n\}$ un experimento formado por un número finito de eventos x_i con probabilidades p_1, \dots, p_n . X es una variable aleatoria discreta en un alfabeto \mathcal{A} . La **entropía** $H(X)$ de la variable aleatoria X se define como

$$H(X) := - \sum_{x_i \in X} p_i \log_{|\mathcal{A}|} p_i.$$

Teorema (Shannon)

Para todo código binario la longitud esperada es mayor o igual que la entropía: $-\sum p_i \log_2 p_i$. Cuando cada p_i es potencia de $1/2$ se tiene la igualdad.



Algoritmo de Huffman

Input: n símbolos a_i con frecuencias w_i

Output: T un árbol con raíz que define un código de Huffman óptimo.

Iteración:

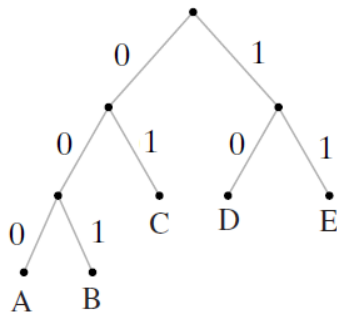
1. F : un bosque de n árboles con raíz, cada uno consiste en un vértice a_i y un peso w_i asignado a la raíz.
2. Mientras que F no sea un árbol
 - a. Reemplace los árboles con raíz T y T' de menor peso en F ($w(T) \leq w(T')$) por un nuevo árbol T^* con subárbol izquierdo T y subárbol derecho T' .
 - b. Asigne $w(T^*) = w(T) + w(T')$.
 - c. Etiquete la nueva arista hacia T con 0 y la nueva arista hacia T' con 1.
3. El código de Huffman para el símbolo a_i es la concatenación de las etiquetas de la aristas en el camino de la raíz al vértice a_i .

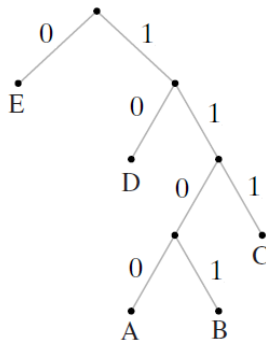


Carácter	Frecuencia
A	2
B	3
C	7
D	8
E	12

- Codificar: ABACEDC
- Decodificar: 1101000100000100









Recorrido de árboles

Algoritmos de recorrido

Los procedimientos para el recorrido sistemático de los vértices de un árbol ordenado con raíz se denominan **algoritmos de recorrido de árboles**.



Recorrido de árboles

Algoritmos de recorrido

Los procedimientos para el recorrido sistemático de los vértices de un árbol ordenado con raíz se denominan **algoritmos de recorrido de árboles**.

- Recorrido preorden.
- Recorrido postorden.
- Recorrido inorden.



Algoritmo de recorrido preorden

Preorden(T)

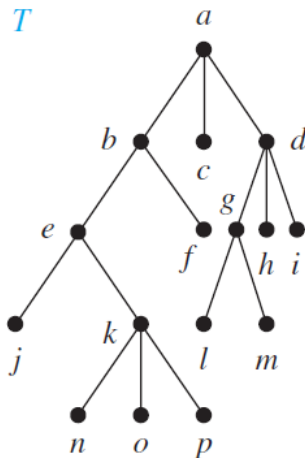
Input: T un árbol ordenado con raíz.

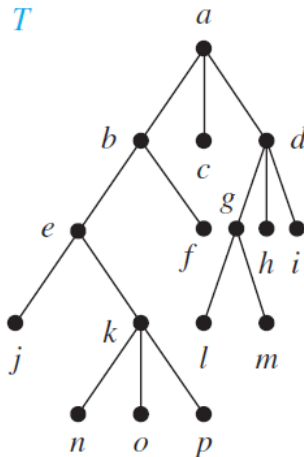
Output: v_1, v_2, \dots, v_n una enumeración de los vértices de T .

Iteración:

1. r : raíz de T .
2. Enumerar r .
3. Para cada hijo c de r de izquierda a derecha
 - a. $T(c)$: subárbol de raíz c .
 - b. *Preorden*($T(c)$).







a *b* *e* *j* *k* *n* *o* *p* *f* *c* *d* *g* *l* *m* *h* *i*
 • • • • • • • • • • • • • • • •

Árboles - Aplicaciones

Algoritmo de recorrido inorden

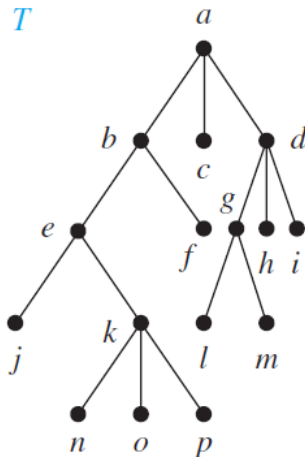
Inorden(T)

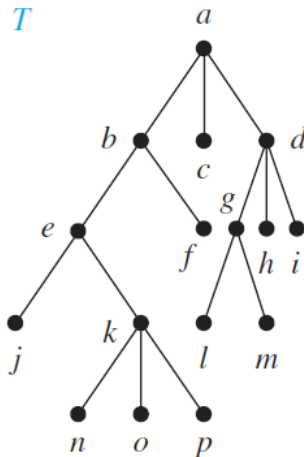
Input: T un árbol ordenado con raíz.

Output: v_1, v_2, \dots, v_n una enumeración de los vértices de T .

Iteración:

1. r : raíz de T .
2. Si r es una hoja
 - a. Enumerar r .
3. Si no
 - a. l : primer hijo de izquierda a derecha
 - b. $T(l)$: subárbol de raíz l .
 - c. *Inorden*($T(l)$).
 - d. Enumerar r .
 - e. Para cada hijo $c \neq l$ de r de izquierda a derecha
 1. $T(l)$: subárbol de raíz c .
 2. *Inorden*($T(c)$).





j e n k o p b f a c l g m d h i

• • • • • • • • • • • • • • • •

Árboles - Aplicaciones

Algoritmo de recorrido postorden

Postorden(T)

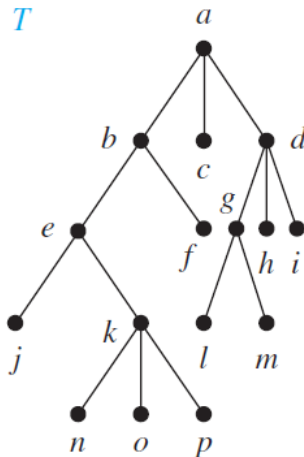
Input: T un árbol ordenado con raíz.

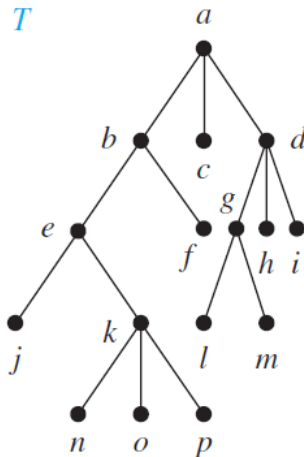
Output: v_1, v_2, \dots, v_n una enumeración de los vértices de T .

Iteración:

1. r : raíz de T .
2. Para cada hijo c de r de izquierda a derecha
 - a. $T(c)$: subárbol de raíz c .
 - b. *Postorden*($T(c)$).
3. Enumerar r .









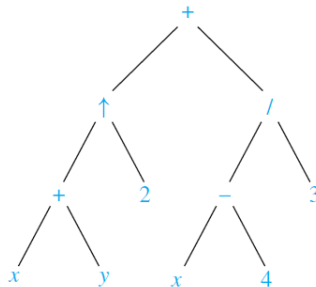
Notación Prefija, Infija y Postfija

- Infija: $((x + y) \uparrow 2) + ((x - 4)/3)$



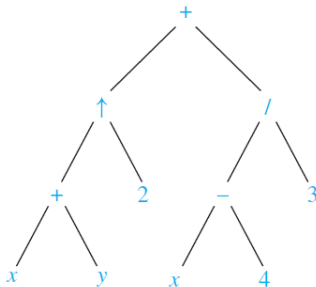
Notación Prefija, Infija y Postfija

- Infija: $((x + y) \uparrow 2) + ((x - 4)/3)$



Notación Prefija, Infija y Postfija

- Infija: $((x + y) \uparrow 2) + ((x - 4)/3)$

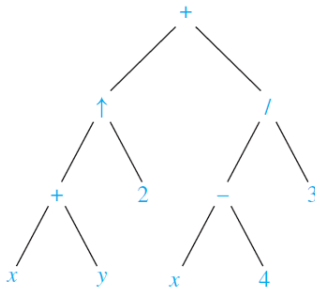


- Prefija: $+ \uparrow + x y 2 / - x 4 3$



Notación Prefija, Infija y Postfija

- Infija: $((x + y) \uparrow 2) + ((x - 4)/3)$



- Prefija: $+ \uparrow + x y 2 / - x 4 3$
- Postfija: $x y + 2 \uparrow x 4 - 3 / +$



Evaluación de una fórmula prefija

$$+ \quad - \quad * \quad 2 \quad 3 \quad 5 \quad / \quad \uparrow \quad 2 \quad 3 \quad 4$$

$$2 \uparrow 3 = 8$$

$$+ \quad - \quad * \quad 2 \quad 3 \quad 5 \quad / \quad 8 \quad 4$$

$$8 / 4 = 2$$

$$+ \quad - \quad * \quad 2 \quad 3 \quad 5 \quad 2$$

$$2 * 3 = 6$$

$$+ \quad - \quad 6 \quad 5 \quad 2$$

$$6 - 5 = 1$$

$$+ \quad 1 \quad 2$$

$$1 + 2 = 3$$



Evaluación de una fórmula postfija

7 2 3 * - 4 ↑ 9 3 / +

└──────────┘

$$2 * 3 = 6$$

7 6 - 4 ↑ 9 3 / +

└──────────┘

$$7 - 6 = 1$$

1 4 ↑ 9 3 / +

└──────────┘

$$1^4 = 1$$

1 9 3 / +

└──────────┘

$$9 / 3 = 3$$

1 3 +

└──────────┘

$$1 + 3 = 4$$





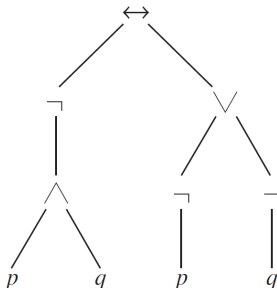
Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$



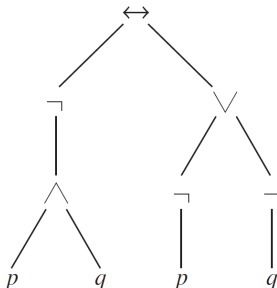
Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$



Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$

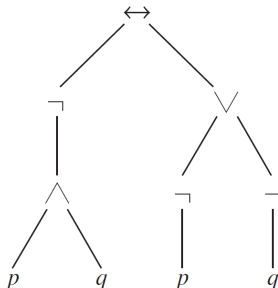



- Fórmula Prefija: $\leftrightarrow \neg \wedge pq \vee \neg p \neg q$



Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$

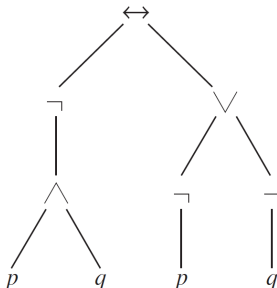


- Fórmula Prefija: $\leftrightarrow \neg \wedge pq \vee \neg p \neg q$ 



Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$

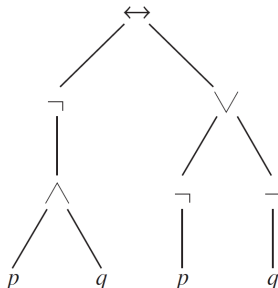



- Fórmula Prefija: $\leftrightarrow \neg \wedge pq \vee \neg p \neg q$  ← Notación Polaca



Notación Polaca

- Fórmula Infija: $(\neg(p \wedge q)) \leftrightarrow ((\neg p) \vee (\neg q))$



- Fórmula Prefija: $\leftrightarrow \neg \wedge pq \vee \neg p \neg q$  ← Notación Polaca
- Fórmula Postfija: $pq \wedge \neg p \neg q \neg \vee \leftrightarrow$



Bibliografía



Douglas B. West

Introduction to graph theory.

Pearson. (2005).



Kenneth Rosen

Discrete Mathematics and its Applications

McGraw Hill. (2012).

