Algoritmo DFS, Depth-First Search (búsqueda en profundidad).
Búsqueda de caminos en profundidad.

Algorithm 2.3: Graph depth-first search with a stack.

```
StackDFS(G, node) → visited
    Input: G = (V, E), a graph
           node, the starting vertex in G
    Output: visited, an array of size |V| such that visited[i] is TRUE if we
            have visited node i, FALSE otherwise
1   S ← CreateStack()
2   visited ← CreateArray(|V|)
3   for i ← 0 to |V| do
4       visited[i] ← FALSE
5   Push(S, node)
6   while not IsStackEmpty(S) do
7       c ← Pop(s)
8       visited[c] ← TRUE
9       foreach v in AdjacencyList(G, c) do
10          if not visited[v] then
11              Push(S, v)
12  return visited
```

Algoritmo BFS, Breadth-First Search (búsqueda en amplitud).
Camino más cortos en grafos con aristas sin peso

Algorithm 2.5: Graph breadth-first search.

```
BFS(G, node) → visited
    Input: G = (V, E), a graph
           node, the starting vertex in G
    Output: visited, an array of size |V| such that visited[i] is TRUE if we
            have visited node i, FALSE otherwise
1   Q ← CreateQueue()
2   visited ← CreateArray(|V|)
3   inqueue ← CreateArray(|V|)
4   for i ← 0 to |V| do
5       visited[i] ← FALSE
6       inqueue[i] ← FALSE
7   Enqueue(Q, node)
8   inqueue[node] ← TRUE
9   while not IsQueueEmpty(Q) do
10      c ← Dequeue(Q)
11      inqueue[c] ← FALSE
12      visited[c] ← TRUE
13      foreach v in AdjacencyList(G, c) do
14          if not visited[v] and not inqueue[v] then
15              Enqueue(Q, v)
16              inqueue[v] ← TRUE
17  return visited
```

Algoritmo de Dijkstra.
Camino más cortos en grafo con aristas con pesos positivos.

Algorithm 7.1: Dijkstra's algorithm.

```
Dijkstra(G, s) → (pred, dist)
    Input: G = (V, E), a graph
           s, the starting node
    Output: pred, an array of size |V| such that pred[i] is the predecessor
            of node i in the shortest path from s
            dist, an array of size |V| such that dist[i] is the length of the
            shortest path calculated from node s to i
1   pred ← CreateArray(|V|)
2   dist ← CreateArray(|V|)
3   pq ← CreatePQ()
4   foreach v in V do
5       pred[v] ← −1
6       if v ≠ s then
7           dist[v] ← ∞
8       else
9           dist[v] ← 0
10      InsertInPQ(pq, v, dist[v])
11  while SizePQ(pq) ≠ 0 do
12      u ← ExtractMinFromPQ(pq)
13      foreach v in AdjacencyList(G, u) do
14          if dist[v] > dist[u] + Weight(G, u, v) then
15              dist[v] ← dist[u] + Weight(G, u, v)
16              pred[v] ← u
17              UpdatePQ(pq, v, dist[v])
18  return (pred, dist)
```

Decimal - Binary - Octal - Hex – ASCII Conversion Chart

Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII	Decimal	Binary	Octal	Hex	ASCII
0	00000000	000	00	NUL	32	00100000	040	20	SP	64	01000000	100	40	@	96	01100000	140	60	`
1	00000001	001	01	SOH	33	00100001	041	21	!	65	01000001	101	41	A	97	01100001	141	61	a
2	00000010	002	02	STX	34	00100010	042	22	"	66	01000010	102	42	B	98	01100010	142	62	b
3	00000011	003	03	ETX	35	00100011	043	23	#	67	01000011	103	43	C	99	01100011	143	63	c
4	00000100	004	04	EOT	36	00100100	044	24	$	68	01000100	104	44	D	100	01100100	144	64	d
5	00000101	005	05	ENQ	37	00100101	045	25	%	69	01000101	105	45	E	101	01100101	145	65	e
6	00000110	006	06	ACK	38	00100110	046	26	&	70	01000110	106	46	F	102	01100110	146	66	f
7	00000111	007	07	BEL	39	00100111	047	27	'	71	01000111	107	47	G	103	01100111	147	67	g
8	00001000	010	08	BS	40	00101000	050	28	(	72	01001000	110	48	H	104	01101000	150	68	h
9	00001001	011	09	HT	41	00101001	051	29	)	73	01001001	111	49	I	105	01101001	151	69	i
10	00001010	012	0A	LF	42	00101010	052	2A	*	74	01001010	112	4A	J	106	01101010	152	6A	j
11	00001011	013	0B	VT	43	00101011	053	2B	+	75	01001011	113	4B	K	107	01101011	153	6B	k
12	00001100	014	0C	FF	44	00101100	054	2C	,	76	01001100	114	4C	L	108	01101100	154	6C	l
13	00001101	015	0D	CR	45	00101101	055	2D	-	77	01001101	115	4D	M	109	01101101	155	6D	m
14	00001110	016	0E	SO	46	00101110	056	2E	.	78	01001110	116	4E	N	110	01101110	156	6E	n
15	00001111	017	0F	SI	47	00101111	057	2F	/	79	01001111	117	4F	O	111	01101111	157	6F	o
16	00010000	020	10	DLE	48	00110000	060	30	0	80	01010000	120	50	P	112	01110000	160	70	p
17	00010001	021	11	DC1	49	00110001	061	31	1	81	01010001	121	51	Q	113	01110001	161	71	q
18	00010010	022	12	DC2	50	00110010	062	32	2	82	01010010	122	52	R	114	01110010	162	72	r
19	00010011	023	13	DC3	51	00110011	063	33	3	83	01010011	123	53	S	115	01110011	163	73	s
20	00010100	024	14	DC4	52	00110100	064	34	4	84	01010100	124	54	T	116	01110100	164	74	t
21	00010101	025	15	NAK	53	00110101	065	35	5	85	01010101	125	55	U	117	01110101	165	75	u
22	00010110	026	16	SYN	54	00110110	066	36	6	86	01010110	126	56	V	118	01110110	166	76	v
23	00010111	027	17	ETB	55	00110111	067	37	7	87	01010111	127	57	W	119	01110111	167	77	w
24	00011000	030	18	CAN	56	00111000	070	38	8	88	01011000	130	58	X	120	01111000	170	78	x
25	00011001	031	19	EM	57	00111001	071	39	9	89	01011001	131	59	Y	121	01111001	171	79	y
26	00011010	032	1A	SUB	58	00111010	072	3A	:	90	01011010	132	5A	Z	122	01111010	172	7A	z
27	00011011	033	1B	ESC	59	00111011	073	3B	;	91	01011011	133	5B	[	123	01111011	173	7B	{
28	00011100	034	1C	FS	60	00111100	074	3C	<	92	01011100	134	5C	\	124	01111100	174	7C	|
29	00011101	035	1D	GS	61	00111101	075	3D	=	93	01011101	135	5D	]	125	01111101	175	7D	}
30	00011110	036	1E	RS	62	00111110	076	3E	>	94	01011110	136	5E	^	126	01111110	176	7E	~
31	00011111	037	1F	US	63	00111111	077	3F	?	95	01011111	137	5F	_	127	01111111	177	7F	DEL

This work is licensed under the Creative Commons Attribution-ShareAlike License. To view a copy of this license, visit http://creativecommons.org/licenses/by-sa/3.0/