Universidade Aberta

Aluno 1002599 Octávio Augusto da Silva Oliveira
21/02/2017

Substituição

Codificar

Dados

Palavra: ataque

A chave tem um deslocamento de 3 letras.

Determinar a chave

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	z
Chave K:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Z	A	B	C

	a	t	a	q	u	e
Palavra:						
C=K(ataque)	**d**	**w**	**d**	**t**	**x**	**h**

Descodificar

Dados

Cifra: dwdsxh

Chave:Deslocamento de 3 letras

	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	z
Chave K:	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Z	A	B	C

	d	w	d	t	x	h
Cifra:						
Decifra:	**a**	**t**	**a**	**q**	**u**	**e**

Transposição

Codificar

Dados
Palavra: Universidade aberta
Chave K: Octavio

O	c	t	a	v	i	o
4	2	6	1	7	3	5
U	n	i	v	e	r	s
i	d	a	d	e	_	a
b	e	r	t	a	*	*

A leitura faz-se em coluna começando em 1 e terminando em 7

Cifra: **v d t n d e r U i b s a i a r e e a**

Descodificar

Dados
Cifra: vdtnder Uibsaiareea
Chave:Octavio

Cifra:	v	d	t	n	d	e	r	U	i	b	s	a	i	a	r	e	e	a
	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
		1			**2**			**3**			**4**			**5**			**6**			**7**

Cifra tem 19 letras
Chave tem: 7 letras => 7 colunas
Linhas? **19/7 = 2,7 => 3; 3x7=21 e 21-19 = 2 celulas de sobra (as ultimas)**

	O	c	t	a	v	i	o
	4	**2**	**6**	**1**	**7**	**3**	**5**
0	U	n	i	v	e	r	s
1	i	d	a	d	e		a
2	b	e	r	t	a	*	*

Decifra: **Universidade aberta**

Nota:
Tamanho da chave dá o nº colunas
Total de letras = nº celulas
Nº linhas = nº celulas/nº colunas
Celulas de sobra = nº colunas x nº linhas – total de letras
As sobras ficam no fim.

Chave única
Codificar

Binário									
1	**1**	**1**	**1**	1	1	1	1		
128	**64**	**32**	**16**	8	4	2	1		Dec.
0	**1**	**0**	**0**	0	0	0	1	=	**65**
0	**1**	**0**	**0**	1	1	0	0	=	**76**

Usar por exemplo um XOR

0	**0**	**0**	**1**	0	0	1	1	=	**20**
0	**0**	**0**	**1**	1	1	1	0	=	**30**

Dados: AL e chave: 101 0010

Chars	A	L
ASCII (dec.)	65	76
Binário	0100 0001	0100 1100
K	0101 0010	0101 0010
XOR	0001 0011	0001 1110
ASCII (dec.)	20	30
Cifra	**device ctrl4**	**record sep.**

Descodificar

Chars	device ctrl4	record sepa.
ASCII (dec.)	20	30
Binário	0001 0011	0001 1110
K	0101 0010	0101 0010
XOR	0100 0001	0100 1100
ASCII (dec.)	65	76
Decifra	**A**	**L**

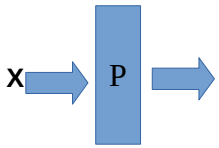
Caixa permutação P

Codificar

Dados

Palavra binária: 0010 1100

Chave K=3607 1245



X=0010 1100

Entrada Palavra	Ordem	K	Resultado	**Saída** Inverter
0	**7**	**3**	1	1
0	**6**	**6**	0	0
1	**5**	**0**	0	1
0	**4**	**7**	0	0
1	**3**	**1**	0	0
1	**2**	**2**	1	0
0	**1**	**4**	0	0
0	**0**	**5**	1	1

Y=1010 0001

1º

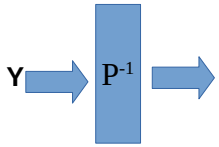
K	Inv
3	0
6	1
0	2
7	3
1	4
2	5
4	6
5	7

Descodificar

Dados

Cifra binária: 1010 0001

Chave K=36071245



Y=1010 0001

Entrada Invertido	Inverter	K	Ordem	**Saída** Resultado
1	1	**3**	**7**	0
0	0	**6**	**6**	0
1	0	**0**	**5**	1
0	0	**7**	**4**	0
0	0	**1**	**3**	1
0	1	**2**	**2**	1
0	0	**4**	**1**	0
1	1	**5**	**0**	0

X=0010 1100

Chave inversa

Ord.	Kinv
7	3
6	1
5	7
4	6
3	0
2	5
1	4
0	2

2º

Kinv = 3176 0542

Nota: Cifra em bloco tem caixas S e P

C = E (P,Ke)
P - texto a ciferar Ke – chave da cifra C – texto cifrado E – função matemática que implementa o método da cifra dependente da chave Ke

P = D(C, Kd)
C- texto cifrado Kd – chava da decifra P – texto cifrado D - função matemática que implementa o método da decifra dependente da chave Kd

Intrusos:
Passivo – escuta; não altera
Ativo – modifica e cria mensagens

Redundância e tempo– usados para validar mensagens;

Criptógrafo – Faz cifras; usa métodos conhecidos pelo emissor e recetor;

Criptoanalista – decifra a cifra (texto codificado);

Aumento do comprimento da chave => aumento exponencial da decifra;

Cifra transposição => alteração da posição dos símbolos;

Cifra blocos – Texto simples dividido em blocos dá cifra em blocos com mesmo nº bits;

Chave simétrica – usa a mesma chave para codificar e descodificar;

Caixas P (Permutação) - faz transposição de bits;
Caixas S (Substituição) – substituiu um texto n bits por outro n bits;

Cifra de produto – utilização em cascata de caixas P e S;

Cifra – garante a confidencialidade da mensagem através da sua codificação

Assinatura digital – garante a autenticidade e integridade da mensagem (exemplo DSS)

Controlo de não-repudiação – Autor da mensagem não pode negar que assinou a mensagem;

Evitar ataque de reenvio de mensagens genuínas – tática do intruso ativo, evitar com tempo de validade da mensagem;

Chave pública – esquema assimétrico
com chave de encriptação (chave pública)
com chave de desencriptação (chave privada correspondente)
as duas chaves são relacionadas entre si
Qualquer um pode ter a chave pública correspondente à minha chave privada e encriptar texto, mas apenas eu, que tenho a chave privada, posso descodificar o texto.

Ds (Es(P)) = P
Ds – chave secreta (privada) do recetor s
Es – chave publica para enviar para o recetor s

Algoritmo RSA – lento 1024 bits

Sumário de mensagens – Mensagens tem autenticação (assinatura) e sigilo (cifra), cifrar é lento e pode não ser necessário. O sumário de mensagens é um esquema de autenticação que não exige a criptografia do texto inteiro. O sumário de mensagens é uma função de hash que utiliza apenas uma parte qualquer do texto, é a função **MD(P)**, que só é determinada sabendo P. (a função deve ter 128 bits)

Caixa substituição S

Descodificar

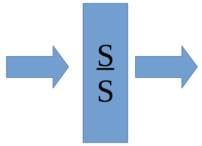
Dados

Palavra binária:1010 0001

Chave K=36071245BE8F9ACD

1º Converter 0100 para decimal

1	**1**	**1**	**1**		
8	4	2	1		
1	0	1	0	=	**10**
0	0	0	1	=	**1**
0	1	1	0	=	**6**
1	0	0	1	=	**8**



Z

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
		X									X					
Chave INV K	**3**	**6**	**0**	**7**	**1**	**2**	**4**	**5**	**B**	**E**	**8**	**F**	**9**	**A**	**C**	**D**
Res.		0110									1000					

Cifra: 1000 0110

Descodificar

Dados

Cifra binária:1000 0110

Chave K=36071245BE8F9ACD

Z=0110 1000 0110 = 6 e 1000 = 8 na chave

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
K	3	6	0	7	1	2	4	5	B	E	8	F	9	A	C	D
		X									X					
Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Res.		0001									1010					

Chave INV

Hex	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
Kinv	2	4	5	0	6	7	1	3	A	C	D	8	E	F	9	B

Decifra: 1010 0001 = Y

Compactação
Codificar

Dados
Palavra:ABBEEEEACDDDFFDDDEEEE

	A	B	C	D	E	F
Quantidade	1	2	1	6	8	2
Razão	0,05	0,1	0,05	0,3	0,4	0,1

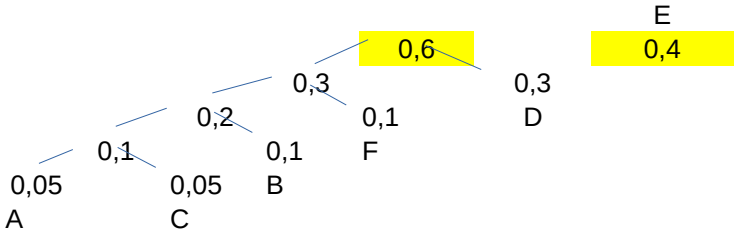
A C	B	F	D	E	
0,05	0,1	0,1	0,3	0,4	

	B	F	D	E
0,1	0,1	0,1	0,3	0,4
0,05 A	0,05 B			

		F	D	E
0,2		0,1	0,3	0,4
0,1	0,1			
0,05 A	0,05 C	B		

			D	E
0,3			0,3	0,4
0,2		0,1		
0,1	0,1	F		
0,05 A	0,05 C	B		

Regras
Ordenar em crescente;
Juntar sempre os menores;
O menor fica mais à esquerda



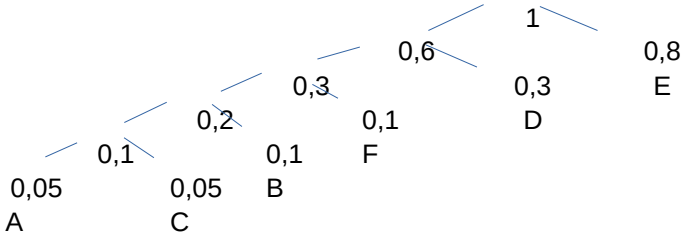


Tabela de mapeamento de símbolos

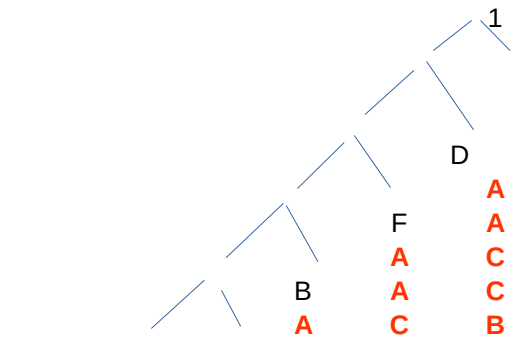
Símbolo	Código
A	00000
C	00001
B	0001
F	001
D	01
E	1

Resultado: 00000 0001 0001 1 1 1 1 00000 00001 01 01 01 001 001 01 01 01 1 1 1 1

Compactação
Descodificar

Dados
Código binário: 000000001000111110000000001010101001001010101111
Árvore binariA ou tabela de mapeamento de símbolos

Regras
Segue sempre até uma folha (1)
A letra com menor frequência tem a maior sequência de zeros e por isso tem a folha mais à esquerda



																Inicio: 000000001000111110000000001010101001001010101111	out:00000
															A	**Seguinte: 000100011111000000000101010100100101010111111**	out:00001
														E	**A**	**C**	**Seguinte: 000111110000000000101010100100101010111111**	out:0001
													A	**C**	**B**	**Seguinte: 111100000000001010101001001010101111**	out:1
												D	**A**	**C**	**B**	**E**	**Seguinte: 111000000000101010100100101010111111**	out:1
											A	**C**	**B**	**E**	**E**	**Seguinte: 110000000001010101001001010111111**	out:1
										F	**A**	**C**	**B**	**E**	**E**	**E**	**Seguinte: 100000000101010100100101011111**	out:1
										A	**C**	**B**	**E**	**E**	**E**	**E**	**Seguinte: 000000001010101001001010111111**	out:00000
									B	**A**	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**Seguinte: 000010101010010010101011111**	out:00001
									A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**Seguinte: 010101001001010111111**	out:01
								A	**C**	**A**	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**Seguinte: 01010010010101011111**	out:01
							A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**Seguinte: 0100100101011111**	out:01
						A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**Seguinte: 001001010111111**	out:001
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**Seguinte: 0010101011111**	out:001
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**F**	**Seguinte: 0101011111**	out:01
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**F**	**D**	**Seguinte: 01011111**	out:01
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**F**	**DD**	**Seguinte: 011111**	out:01
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**F**	**DDD**	**Seguinte: 1111**	out:1
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**FF**	**DDD**	**E**	**Seguinte: 111**	out:1
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**FF**	**DDD**	**EE**	**Seguinte: 11**	out:01
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**FF**	**DDD**	**EEE**	**Seguinte: 1**	out:01
					A	**C**	**B**	**E**	**E**	**E**	**E**	**A**	**C**	**D**	**D**	**D**	**F**	**F**	**DDD**	**EEEE**	

L(ave): Numero médio de bits para representar dada informação.

$$L(ave) = P(mi) \times L(P(mi)) + ... + P(mn) \times L(P(mn)) \qquad \log_b(a) = \frac{\log_c(a)}{\log_c(b)}$$

$$L(P(mi)) = -\log_2(P(mi)) = \log_2(P(mi)^{-1}) = \log_2(\frac{1}{P(mi)})$$

$$L(Huff) = P(mi) \times num_{bits}$$

Exemplo para representar a Informação: ABCDEF

L(ave)

P(m1)	P(m2)	P(m3)	P(m4)	P(m5)	P(m6)	Soma
0,05	0,05	0,1	0,1	0,3	0,4	**1**
L(P(m1))	L(P(m2))	L(P(m3))	L(P(m4))	L(P(m5))	L(P(m6))	
4,3219	4,3219	3,3219	3,3219	1,737	1,3219	18,347
						bits
						19

L(Huff)

P(A)	P(C)	P(B)	P(F)	P(D)	P(E)	Soma
0,05	0,05	0,1	0,1	0,3	0,4	**1**
L(P(m1))	L(P(m2))	L(P(m3))	L(P(m4))	L(P(m5))	L(P(m6))	
5	5	4	3	2	1	20
						bits
						20

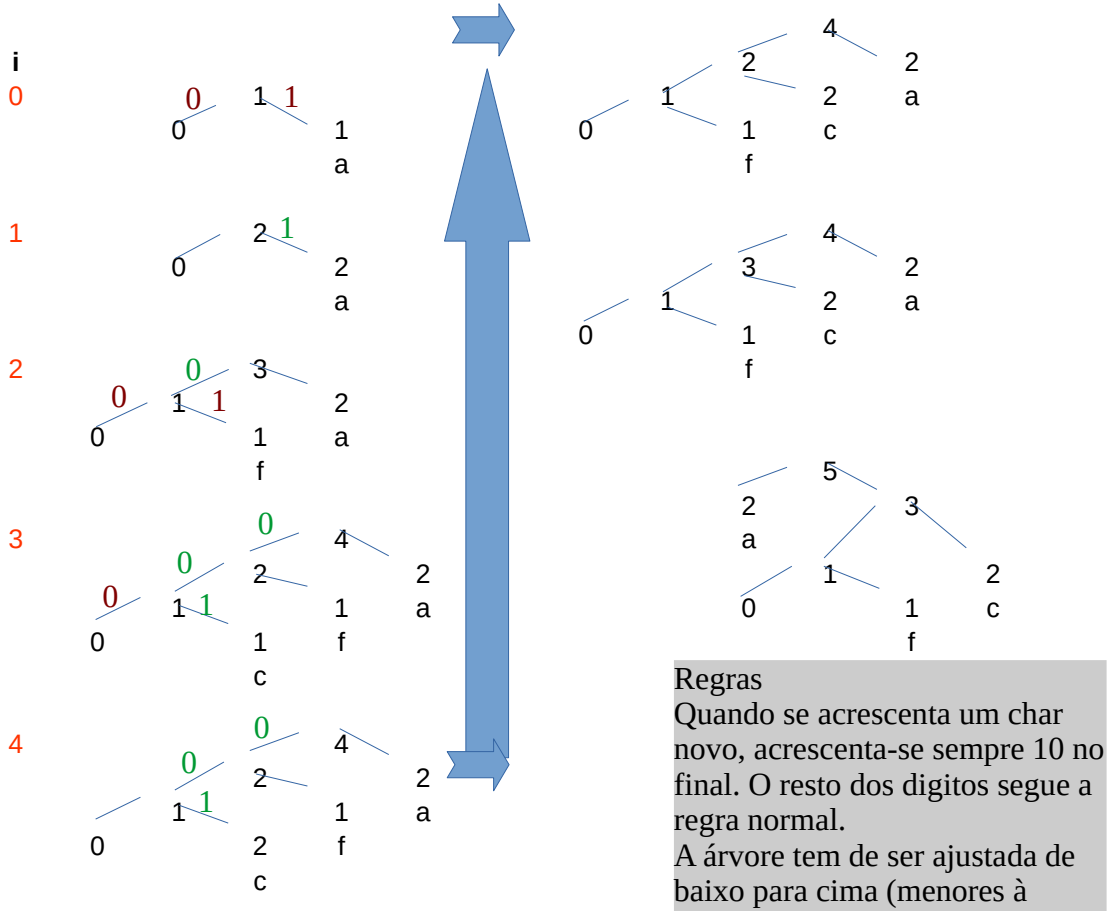
Símbolo	Código
A	00000
C	00001
B	0001
F	001
D	01
E	1

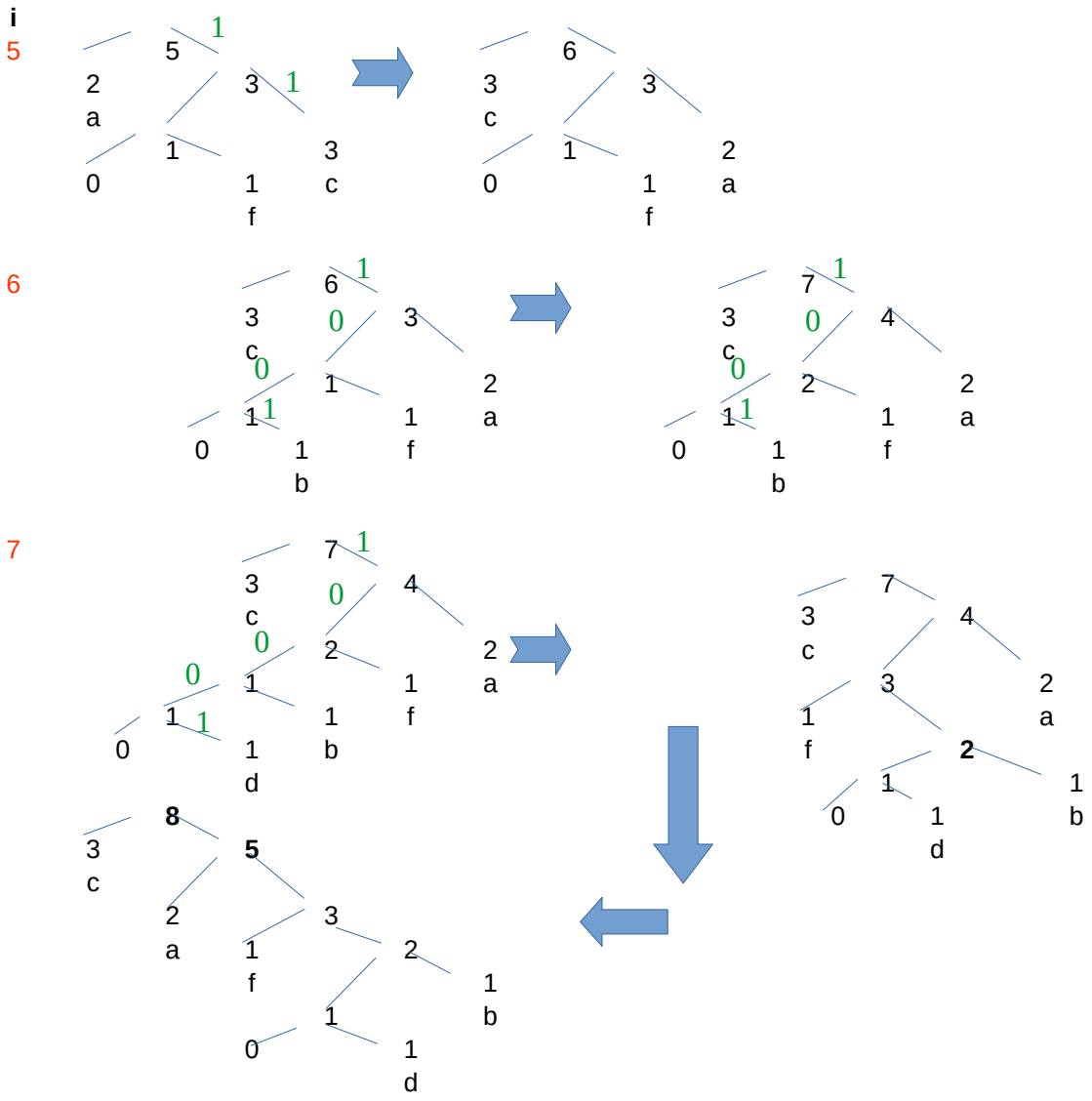
Diferença de apenas 1 bit

Compactação
Codificação adaptativa

Dados
Palavra: aafcccbd

i	in	out
0	a	**10**
1	a	1
2	f	0**10**
3	c	00**110**
4	c	001
5	c	11
6	b	100**110**
7	d	1000**110**




i
5
6
7
8

Codificação LZW S="DADABACDADBACDEEEBACEAABCDDDBCDABBABBABAC"

Palavra dada			Entrada			Saída			Tabela de códigos			Entrada	Saída
i	CH		i	Si	CH	Código	Tabela	Sf	idx	Palavra	i	D	***
0	D	X	0	***	D	***	***	D	1	A	0	A	4
1	A		1	D	A	4	DA	A	2	B	0	DA	1
2	D		2	A	D	1	AD	D	3	C	0	B	6
3	A	X	3	D	A	***		DA	4	D	0	A	2
4	B		4	DA	B	6	DAB	B	5	E	0	C	1
5	A		5	B	A	2	BA	A	6	D A	1	DA	3
6	C		6	A	C	1	AC	C	7	A D	2	D	6
7	D		7	C	D	3	CD	D	8	D A B	4	BA	4
8	A	X	8	D	A	***		DA	9	B A	5	CD	9
9	D		9	DA	D	6	DAD	D	10	A C	6	E	11
10	B		10	D	B	4	DB	B	11	C D	7	EE	5
11	A	X	11	B	A	***		BA	12	D A D	9	BAC	16
12	C		12	BA	C	9	BAC	C	13	D B	10	E	14
13	D	X	13	C	D	***		CD	14	B A C	12	A	5
14	E		14	CD	E	11	CDE	E	15	C D E	14	A	1
15	E		15	E	E	5	EE	E	16	E E	15	B	1
16	E	X	16	E	E	***		EE	17	E E B	17	CD	2
17	B		17	EE	B	16	EEB	B	18	C E	20	D	11
18	A	X	18	B	A	***		BA	19	EA	21	DB	4
19	C	X	19	BA	C	***		BAC	20	AA	22	CD	13
20	E		20	BAC	E	14	CE	E	21	AB	23	AB	11
21	A		21	E	A	5	EA	E	22	BC	24	BA	21
22	A		22	A	A	1	AA	A	23	CDD	26	B	9
23	B		23	A	B	1	AB	B	24	DD	27	BAB	2
24	C		24	B	C	2	BC	C	25	DBC	29	AC	28
25	D	X	25	C	D	***	***	CD				***	10
26	D		26	CD	D	11	CDD	D					
27	D		27	D	D	4	DD	D					
28	B	X	28	D	B	***	***	DB					
29	C		29	DB	C	13	DBC	C					

Descodificação LZW

Nota: Nova entrada é Palavra(i-1) + 1ºCHAR Palavra(i)

Dados: Tabela de código e códigos de entrada.

	Entrada	Saída	Nova Entrada			**Tabela de códigos**	
i	Código	Palavra	idx	Palavra	Obs.	**Código**	**Palavra**
0	4	D	***	***		1	A
1	1	A	6	DA		2	B
2	6	DA	7	AD		3	C
3	2	B	8	DAB		4	D
4	1	A	9	BA		5	E
5	3	C	10	AC			
6	6	DA	11	CD			
7	4	D	12	DAD			
8	9	BA	13	DB			
9	11	CD	14	BAC			
10	5	E	15	CDE			
11	16	EE	16	EE	**Especial => ultimo ch+ultimo ch**		
12	14	BAC	17	EEB			
13	5	E	18	**BACE**	**Resulta de 2 linhas que já existem (BA+CE)**		
14	1	A	19	EA			
15	1	A	20	AA			
16	2	B	21	AB			
17	11	CD	22	BC			
18	4	D	23	CDD			
19	13	DB	24	DD			
20	11	CD	25	DBC			
21	21	AB	26	CDA			
22	9	BA	27	ABB			
23	2	B	28	BAB			
24	28	BAB	29	BB			
25	10	AC	30	BABA			

Palavra dada			Entrada			Saída			Tabela de códigos		
i	CH		i	Si	CH	Código	Tabela	Sf	idx	Palavra	i
30	D	X	30	C	D	***	***	CD	26	CDA	31
31	A		31	CD	A	11	CDA	A	27	ABB	33
32	B	X	32	A	B	***	***	AB	28	BAB	35
33	B		33	AB	B	21	ABB	B	29	BB	36
34	A	X	34	B	A	***	***	BA	30	BABA	39
35	B		35	BA	B	9	BAB	B			
36	B		36	B	B	2	BB	B			
37	A	X	37	B	A	***	***	BA			
38	B	X	38	BA	B	***	***	BAB			
39	A		39	BAB	A	28	BABA	A			
40	C		40	A	C	***	***	AC			
41			41	AC	***	10	***				
			42	***							

Resultado: Código = 4,1,6,2,1,3,6,4,9,11,5,16,9,3,… para a tabela de códigos A=1; B=2; C=3; D=4; E=5

Codificar LZ77 **S="AAB ABA CBA CBA CBC ACB ACB CAA". (os espaços não contam)**

Palavra dada i	CH		i	L1=9 Buffer Procura	L2=9 Look-ahead	Saída Triplas	Nota
0	A			8 ← 0	buffer		
1	A		0		AABABACBA	(0,0,A)	
2	B		1		**A** **A***B*ABACBAC	(0,1,B)	
3	A		2		A**AB** **ABA***C*BACBA	(1,3,C)	AB e volta a A
4	B	0	3		AABA**BAC** **BACBACB***C*A	(2,7,C)	BAC volta a BAC e volta a B
5	A		4		CB**ACBACBC** **ACBACBC**A*A*	(6,8,A)	ACBACBC e volta a A
6	C						
7	B						
8	A						
9	C	1					
10	B						
11	A	2					
12	C						
13	B						
14	C						
15	A	3					
16	C						
17	B						
18	A						
19	C						
20	B	4					
21	C						
22	A						
23	A						

Tripla = (Posição, Tamanho, CH) = (P,T,CH);
Posição = no buffer de procura, neste caso da direita para a esquerda ←;
Exemplo P da iteração 4:

CB**A C B A C B C**
6 5 4 3 2 1 0

Tamanho = numero de caracteres a contar da posição para o lado contrario, neste caso →;
Nota: pode repetir sequências começadas em P até ao tamanho T.

O resultado são as triplas.

Taxa de Compressão:

A = Total de letras x bits cada letra
B =Total de triplas x (bits de algarismos x 2 + bits de letra)
Taxa (%)= (A-B)/Ax100

Descodificar LZ77

Triplas dadas		L1=9 **Buffer Procura**	L2=9 **Look-ahead**	**Triplas**	**Nota**	**Saída**
(0,0,A)	i	8 ← 0	buffer			
(0,1,B)	0		A	(0,0,A)		
(1,3,C)	1		A A*B*	(0,1,B)	A+B	
(2,7,C)	2		AAB ABA*C*	(1,3,C)	AB+A+C	
(6,8,A)	3		AABABAC BACBACB*C*	(2,7,C)	BAC+BAC+B+C	
	4		CBACBACBC ACBACBCA*A*	(6,8,A)	ACBACBC+A+A	**AABABA**
	5		ACBACBCAA ***			**AABABACBACBACBC**

Resultado: AAB ABA CBA CBA CBC ACB ACB CAA

S="ABCBBCBAABBABABABABCCABACBCCACCABCABC"

	Entrada		
L1=8	L2=8	**Saída**	
Buffer Procura	**Look-ahead**	**Triplas**	
	A…	(0,0,A)	
A	B…	(0,0,B)	
AB	C…	(0,0,C)	
ABC	BB…	(1,1,B)	
ABCBB	CBA…	(2,2,A)	
ABCBBCBA	ABB..	(7,2,B)	
BBCBAABB	ABA	(2,2,A)	
BAABBABA	BABABC	(3,5,C)	BABA e volta a B
BABABABC	CA	(0,1,A)	
BABABCCA	BAC	(5,2,C)	
ABCCABAC	BCCAC	(6,4,C)	
BACBCCAC	CAB	(2,2,B)	
BCCACCAB	CABC	(2,3,C)	

Total letras = 37x8 bits = 296
Tripla = 2x3 bits + 8 bits = 14 bits
Total triplas = 13x14=182 bits
Taxa =(296-182)/296x100 = 38,5%

Descodificar LZ77

Dado:

Triplas		L1=8 **Buffer Procura**	L2=8 **Look-ahead**	**Triplas**	**Nota**	**Saída**
	i	7 ← 0	buffer			
	0		A	(0,0,A)		
	1	A	B	(0,0,B)		
	2	AB	C	(0,0,C)		
	3	ABC	BB	(1,1,B)		
	4	ABCBB	CBA	(2,2,A)		
	5	ABCBBCBA	ABB	(7,2,B)		
		BBCBAABB	ABA	(2,2,A)		ABC
		BAABBABA	BABABC	(3,5,C)		ABCBBC
		BABABABC	CA	(0,1,A)		…
		BABABCCA	BAC	(5,2,C)		
		ABCCABAC	BCCAC	(6,4,C)		
		BACBCCAC	CAB	(2,2,B)		
		BCCACCAB	CABC	(2,3,C)		
		CCABCABC				

Compressão RLE

O método de compressão RLE (Run Length Encoding,ou RLC - Run Length Coding) é utilizado por numerosos formatos de imagens BMP, PCX, TIFF). Baseia-se na repetição de elementos consecutivos.

A cadeia “AAAAAHHHHHHHHHHHHHH” comprimida dá “5A14H”

Ou outra forma de representar a cadeia seguinte:

AAACCDDEEEEE
(ch, n) => (A,3) (C,2) (D,2) (E,5)
Ou (Cm,ch,n) sendo cm uma marca de sequência.

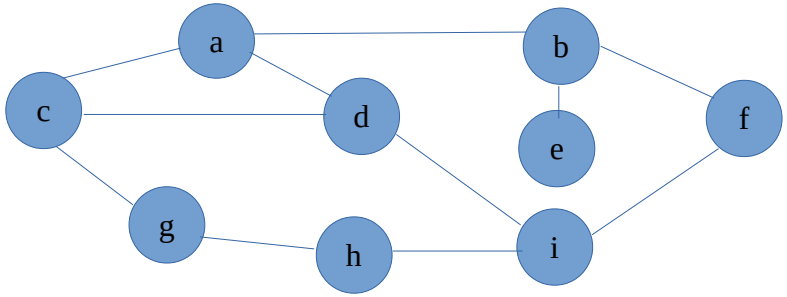


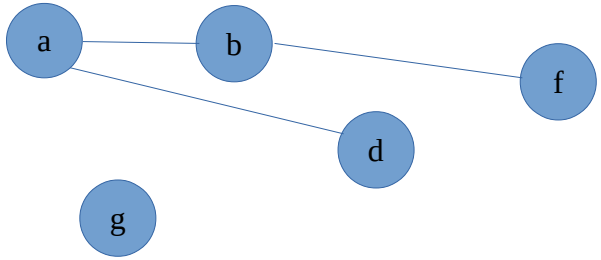
Tabela de adjacências

Vértice	Vértices adjacentes			
a	b	c	d	
b	a	d	e	f
c	a	b	g	
d	a	c	i	
e	b			
f	b	i		
g	c	h		
h	g	i		
i	d	f	h	

Matriz de incidências

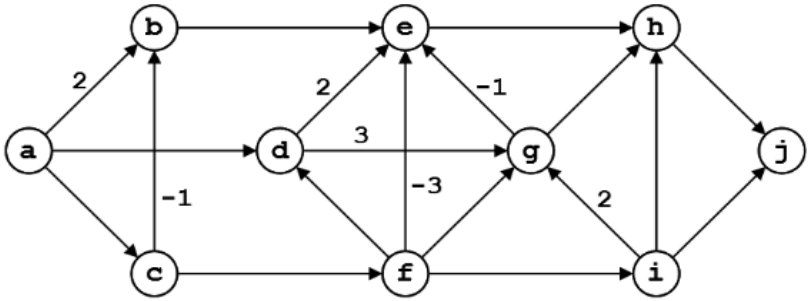
Vértice	ab	ac	ad	be	bf	cd	cg	di	fi	gh	hi
a	1	1	1	0	0	0	0	0	0	0	0
b	1	0	0	1	1	0	0	0	0	0	0
c	0	1	0	0	0	1	1	0	0	0	0
d	0	0	1	0	0	1	0	1	0	0	0
e	0	0	0	1	0	0	0	0	0	0	0
f	0	0	0	0	1	0	0	0	1	0	0
g	0	0	0	0	0	0	1	0	0	1	0
h	0	0	0	0	0	0	0	0	0	1	1
i	0	0	0	0	0	0	0	1	1	0	1

Representação gráfica do grafo induzido pelos vértices:{a,b,d,f,g}



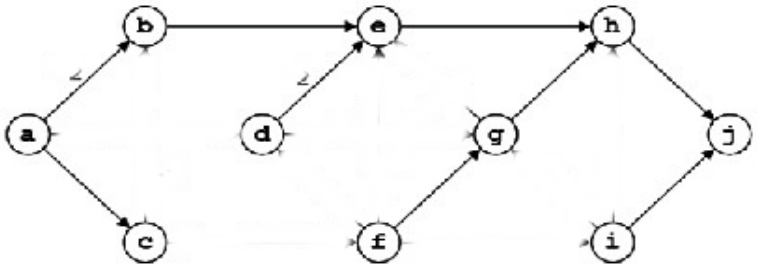
Matriz de adjacência

Faz-se em função dos vértices adjacentes

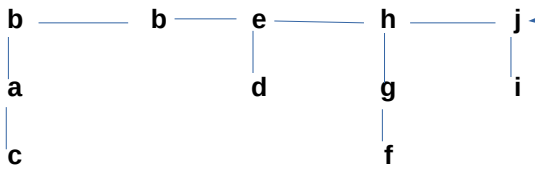


	a	b	c	d	e	f	g	h	i	j
a	0	1	1	1	0	0	0	0	0	0
b	0	0	0	0	1	0	0	0	0	0
c	0	1	0	0	0	1	0	0	0	0
d	0	0	0	0	1	0	1	0	0	0
e	0	0	0	0	0	0	0	1	0	0
f	0	0	0	1	1	0	1	0	0	0
g	0	0	0	0	1	0	0	1	0	0
h	0	0	0	0	0	0	0	0	0	1
i	0	0	0	0	0	0	1	1	0	1
j	0	0	0	0	0	0	0	0	0	0

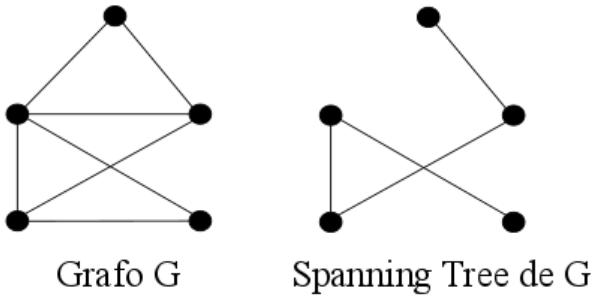
Nota: **a - a** é 0 porque não tem um laço. O resto é igual.



Spannig tree (árvore de expansão)



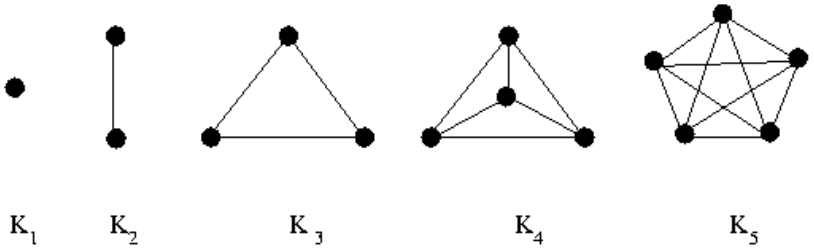
Exemplo:



Tipos de Grafos

Completos

Todos vértices estão ligados entre si



K_1 K_2 K_3 K_4 K_5

$$= \binom{n}{2} = \frac{n!}{(n-2)!\ 2!} = \frac{n(n-1)(n-2)!}{(n-2)!\ 2} = \frac{n(n-1)}{2}$$

Nulo

Não tem vértices

Vazio

Não tem arestas

Regular

Todos os vértices têm mesmo grau

Multigrafo

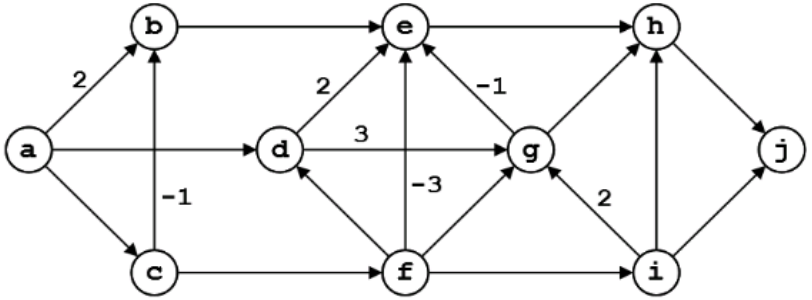
Várias arestas no mesmo vertice

Pseudografo

Contém arestas paralelas e laços.

Simples

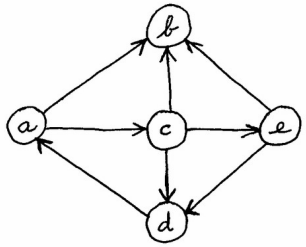
Não direcionado?
Sem laços
Existe no máximo uma aresta entre quaisquer dois vértices



Este é simples, orientado(digrafo), ponderado.
Digrafo = grafo orientado

Arestas (ou arcos) **incidentes** com o vértice **d** = **ad, fd**
Vértices adjacentes ao vértice **d** = **e, g**

Ciclo (ou circuito) = caminho que começa e acaba com o mesmo vértice
Este não tem ciclos.

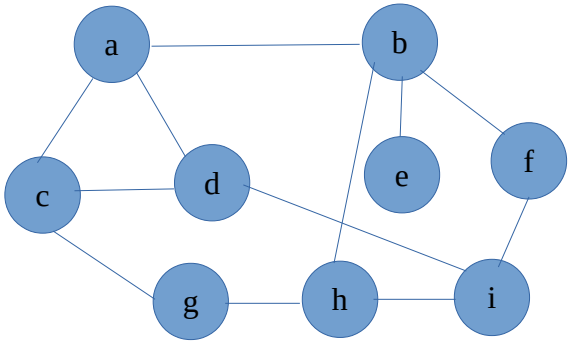


Este já tem 2 ciclos: **[a,c,d, a] e [a,c,e,d, a]**

Busca em Largura

Inicio: Fila = a; Todos Numero = 0

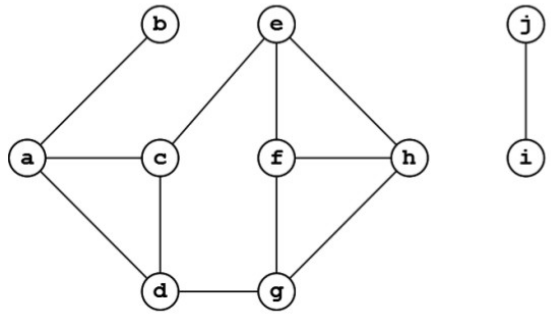
Vértice	Numero	i	Retirado	adjacentes	Fila
a	1	0	***	***	a
b	2	1	a	**b, c, d**	b, c, d
c	3	2	b	a, **e, f, h**	c, d, e, f, h
d	4	3	c	a, d, **g**	d, e, f, h, g
e	5	4	d	a, c, **i**	e, f, h, g, i
f	6	5	e	b	f, h, g, i
g	8	6	f	b, i	h, g, i
h	7	7	h	b, g, i	g, i
i	9	8	g	c, h	i
		9	i	d, h, f	***
		10			




```
busca_largura()
    Fila = V inicial;
    num_Vinicial = 0;
    i = 0;
    Para todos vertices V
        num_V = 0;
     Enquanto existe num_V == 0
        Enquanto existe V na Fila
            Retira V da Fila;
            Para todos adjacentes U de V
                Se num_U == 0
                    Insere U na Fila;
                    num_U = i++;
                Fim
            Fim
        Fim
        Se existe num_V == 0
            Insere V na Fila;
            num_V = i++;
        Fim
    Fim
Fim
```

Inicio: Fila = c; Todos Numero = 0

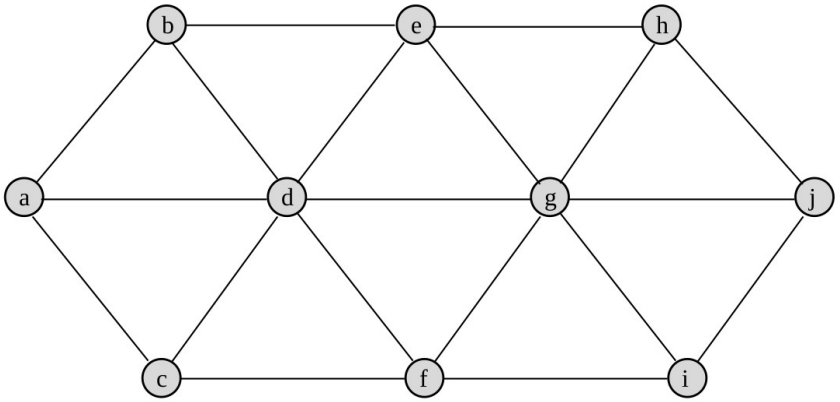
Vértice	Numero	i	Retirado	adjacentes	Fila
a	2	0	***	***	c
b	5	1	c	**a, d, e**	a, d, e
c	1	2	a	**b**, c, d	d, e, b
d	3	3	d	a, c, **g**	e, b, g
e	4	4	e	c, **f, h**	b, g, f, h
f	7	5	b	a	g, f, h
g	6	6	g	d, f, h	f, h
h	8	7	f	e, g, h	h
i	9	8	h	e, f, g	***
j	10	9	***	***	i
		10	i	j	i
		11	j	i	***

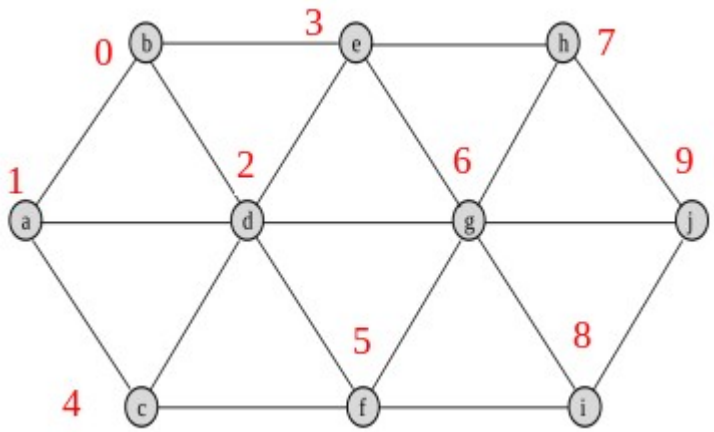


Inicio vértice b

Inicio: Fila = b; Todos Numero = 0

Fila	adjacentes						Vértice	Número
b	a	d	e				b	0
a, d, e	b	d	c				a	1
d, e, c	a	b	c	e	f	g	d	2
e, c, f, g	b	d	g	h			e	3
c,f, g, h	a	d	f				c	4
f, g, h, i	d	c	g	i			f	5
g, h, i, j	d	e	f	h	i	j	g	6
h, i, j	e	g	j				h	7
i, j	f	g	j				i	8
j	g	h	i				j	9

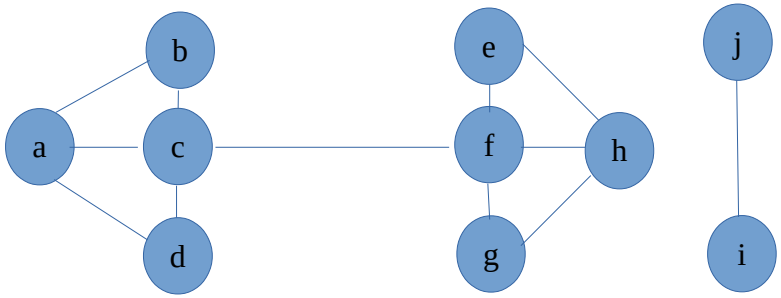




Busca em profundidade

Pesquisa em profundidade começando em **b**

	Visitado	i
DFS(b)	**b**	**1**
b→a	a	0
b → c	c	0
DFS(a)	**a**	**2**
a → c	c	0
a → d	d	0
DFS(c)	**c**	**3**
c→d	d	0
c→ f	g	0
DFS(d)	**d**	**4**
DFS(f)	**f**	**5**
f→e	e	0
f→ g	g	0
f→ h	h	0
DFS(e)	**e**	**6**
e → h	h	0
DFS(h)	**h**	**7**
h → g	g	0
DFS(g)	**g**	**8**
DFS(i)	**i**	**9**
i → j	j	0
DFS(j)	**i**	**9**
	j	10



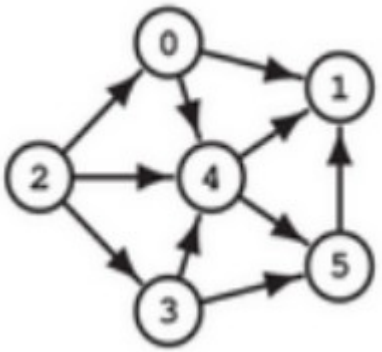

```
procura_profundidade()
    para todos vertices V
        visitado(v) = 0;
    i=0;
    DFS(V0);
    Enquanto existe V com visitado(v)=0;
        DFS(V);
fim
```

```
DFS(V)
    visitado(v)=i;
    i = i+1;
    para todos vertices U adjacentes de V
        se visitado(U) == 0
            DFS(U);
fim
```

Resultado
Ordem de visita: b,a,c,d,f,e,h,g,i,j

Vértice inicial 0

Vértice	Adjacentes			Visitado		
0	1	4		0		
1				1		
2	0	4	3		4	
3	4	5			5	
4	1	5				2
5	1					3



Ordem de visita: f, d, a, b, c, e, h, g, i

DFS (profundidade)

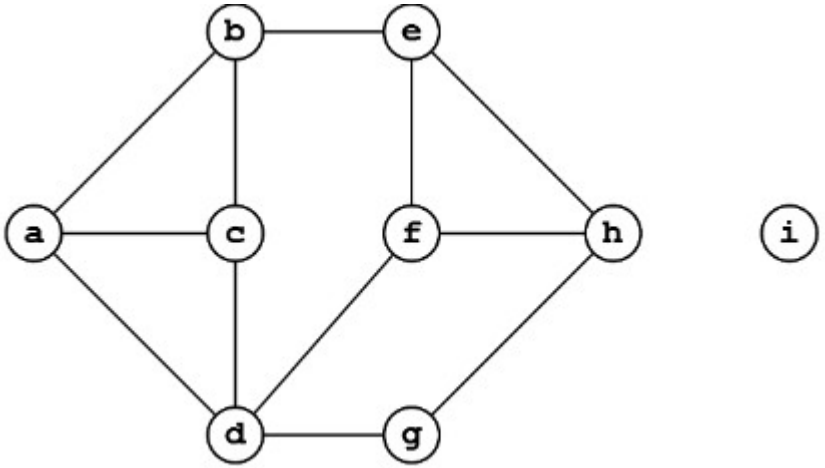
Vértice inicial f

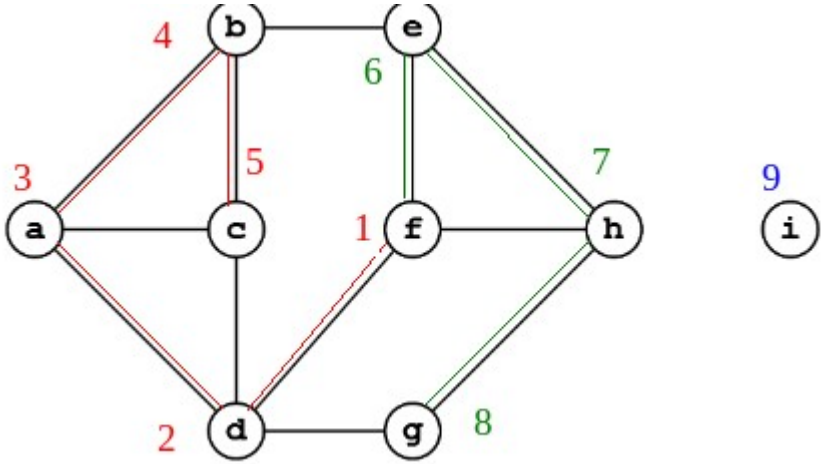
Vértice	Vertices adjacentes	Visitado
f	f → d; f → e; f → h	1
d	d → a; d → c; d → g	2
a	a → b; a → c	3
b	b → c; b → e	4
c		5
e	e → h	6
h	h → g	7
g		8
i		9

Ordem de visita: f, d, a, b, c, e, h, g, i

```
DFS(V, I)
    V.visitado = I;
    I++;
    Para todos U adjacentes de V
        Se U.visitado == 0
            DFS(U,I);
Fim

procura_profundidade()
    Para todos vertices V
        V.visitado = 0;
    I = 0;
    Enquanto existir V.visitado == 0
        DFS(V,I);
Fim
```

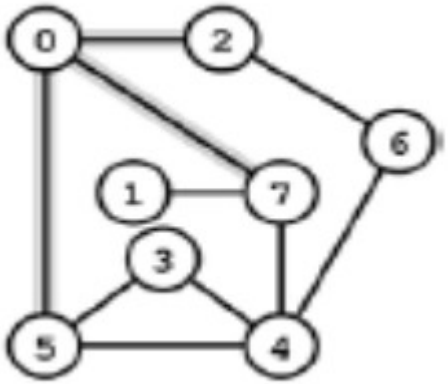


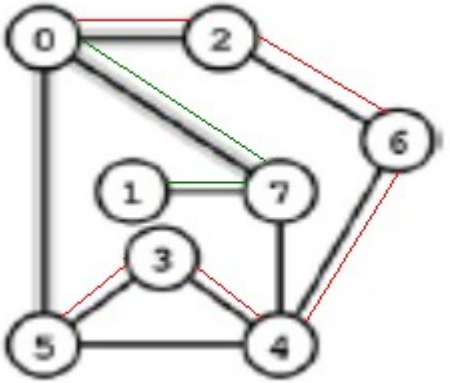


Vértice inicial 0

Vértice	Adjacentes	DFS()
0	2,5,7	0
1	7	2
2	0,6	6
3	4,5	4
4	3,5,6,7	3
5	0,3,4	5
6	2,4	7
7	0,1,4	1

Ordem de visita: 0,2,6,4,3,5,7,1

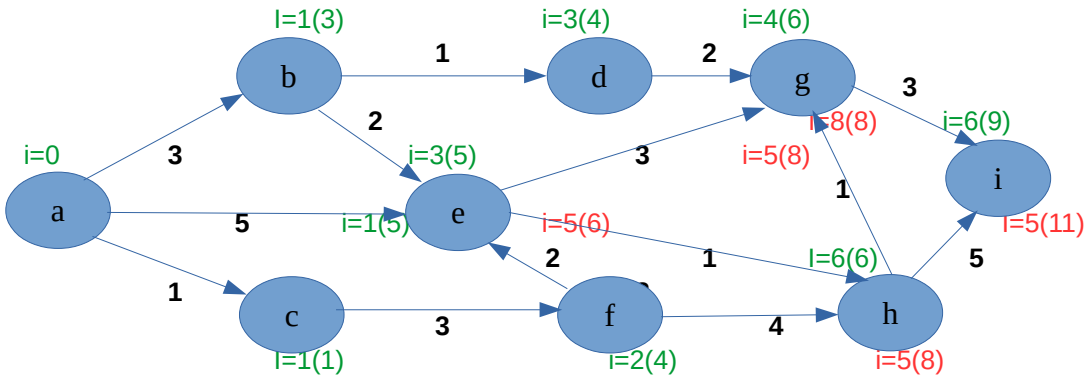




Algoritmo Dijkstra

iteração	0	1	2	3	4	5	6	7	8	
Vértice ativo	***	**a**	**c**	**b**	**d**	**f**	**e**	**g**	**h**	**(x)**
Vértce										
a	0									
b	∞	3	3							
c	∞	1								
d	∞	∞	∞	4						
e	∞	5	5	5	5	**5**				
f	∞	∞	4	4	4					
g	∞	∞	∞	∞	6	6	**6**			
h	∞	∞	∞	∞	∞	8	**6**			
i	∞	∞	∞	∞	∞	∞	∞	9	**9**	
Menor		c	b	d	f	e	g	h		

```
Dijkstra()
    Vinicial.distancia = 0;
    Para todos V diferentes Vinicial
        V.distancia = inf;
    I = 0;
    Enquanto i < totalVertices
        Seleciona Vertice V com menor distancia;
        Para todos vertices U adjacentes de V
            D = distancia(V, U);
            Se D < U.distancia
                U.distancia = D;
        Fim
        I++;
    Fim
Fim
```



Distancia mais curta: 9

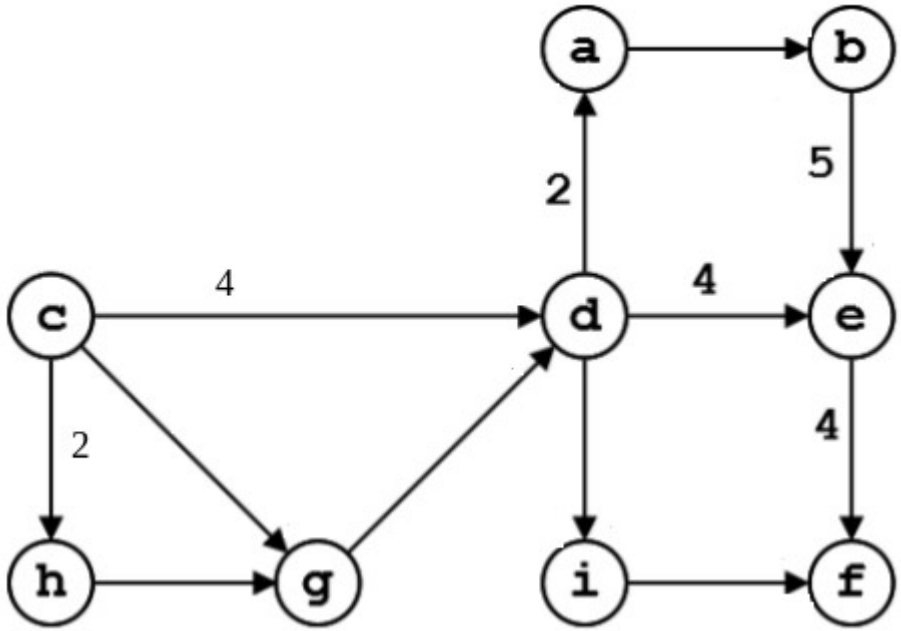
Caminho: a, b, d, g, i

(x) O vértice ativo surge à medida que avança a iteração.

Inicio vértice c

iteração										
iteração	0	1	2	3	4	5	6	7	8	
Vértice ativo	***	**c**	**g**	**d**	**h**	**i**	**a**	**b**	**e**	**(x)**
a	∞	∞	∞	4	4	4	***			
b	∞	∞	∞	∞	∞	∞	5	***		
c	0	***								
d	∞	4	2							
e	∞	∞	∞	6	6	6	6	6	***	
f	∞	∞	∞	∞	∞	4	4	4	4	
g	∞	1	***							
h	∞	2	2	2	***					
i	∞	∞	∞	3	3	***				
Menor		g	d	h	i	a	b	e		

Distancia mais curta: 4
Caminho: c, g, d, i, f



1º Selecionar vértice inicial, neste caso (a) e distâncias para os adjacentes (b, c, e);

2º Selecionar o de menor distância, no caso (c) e coloca-lo no vértice ativo, fazer o mesmo de 1º;

3º Continuar 2º e 1º, somando sempre sempre o caminho. Os valores das colunas anteriores passam para a seguinte quando não são atualizados;

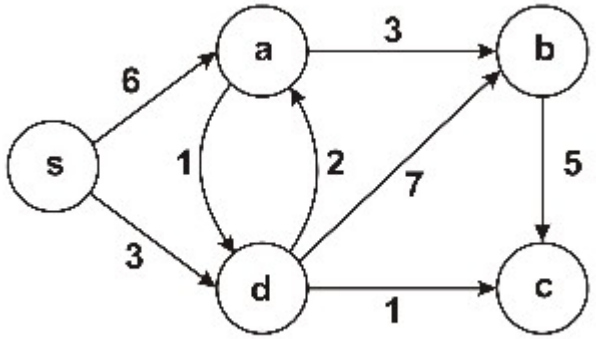
4º Quando chega a nó fica sempre a iteração de menor caminho;

5º Quando o menor é igual segue a ordem alfabética.

Inicio vértice S

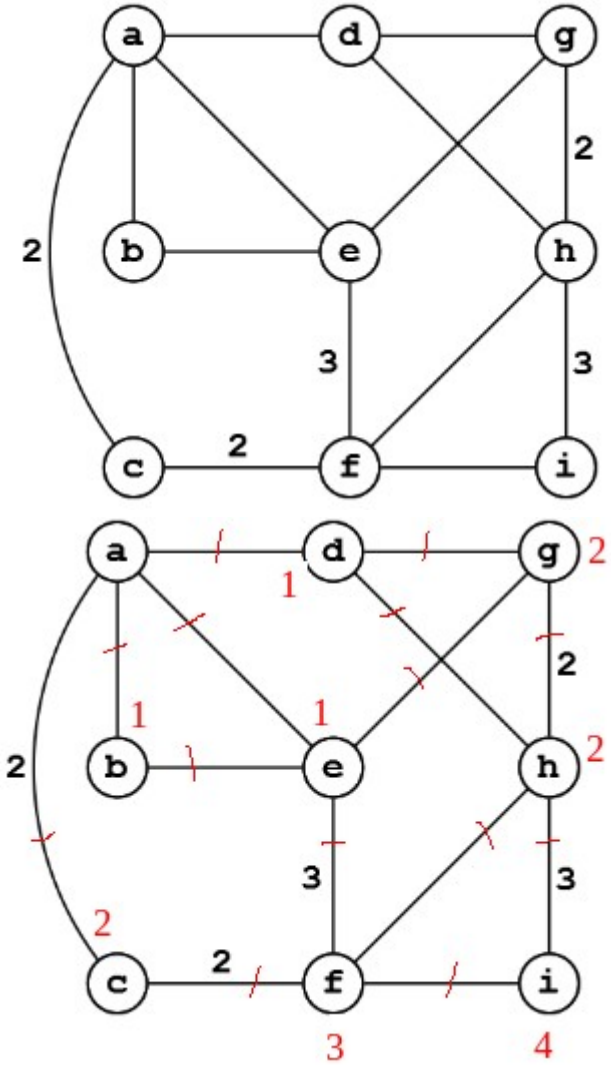
i	0	1	2	3	4	5
Vértice	***	s	d	c	a	b
s	0	***				
a	∞	6	5	5	***	
d	∞	3	***			
b	∞	∞	10	10	8	
c	∞	∞	4	***		

Distancia mais curta: 4
Caminho: s, d, c



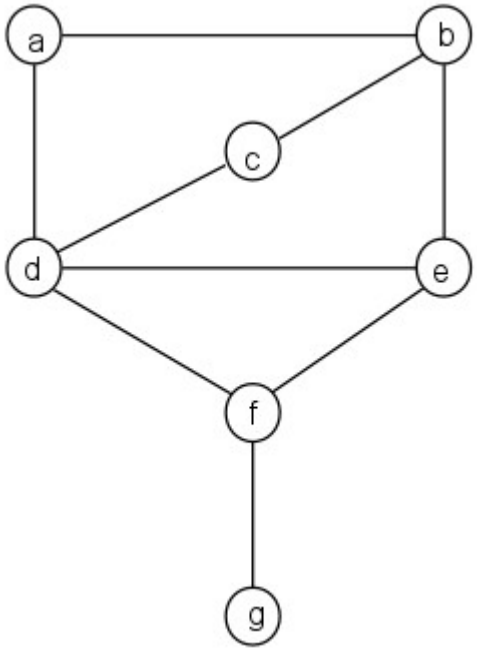
Inicio vértice a

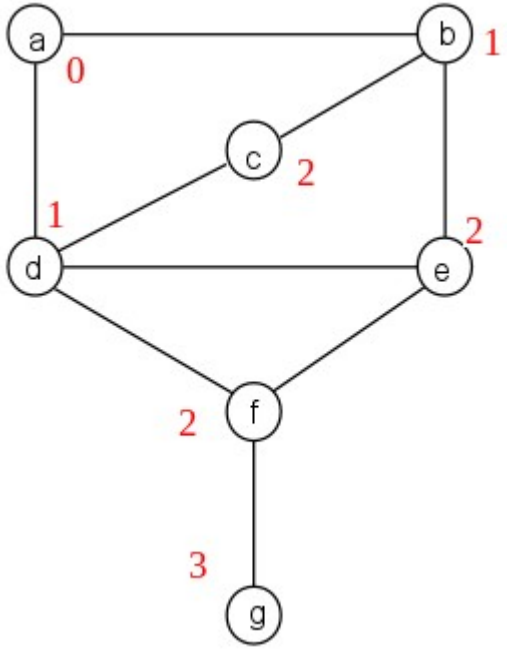
iteração	0	1	2	3	4	5	6	7	8	
Vértice ativo	***	**a**	**b**	**d**	**e**	**c**	**g**	**h**	**f**	**(x)**
Vértce										
a	0									
b	∞	1	***							
c	∞	2	2	2	2	***				
d	∞	1	1	***						
e	∞	1	1	1	***					
f	∞	∞	∞	∞	∞	4	4	3		
g	∞	∞	∞	2	2	2	***			
h	∞	∞	∞	2	2	2	2	***		
i	∞							5	4	
Menor		b	d	e	c	g	h	f		



Inicio vértice a

i	0	1	2	3	4	5	6
Vértice	***	a	b	d	c	e	f
a	0	***					
b	∞	1	***				
c	∞	∞	2	2	***		
d	∞	1	1	***			
e	∞	∞	2	2	2	***	
f	∞	∞	∞	2	2	2	***
g	∞	∞	∞	∞	∞	∞	3



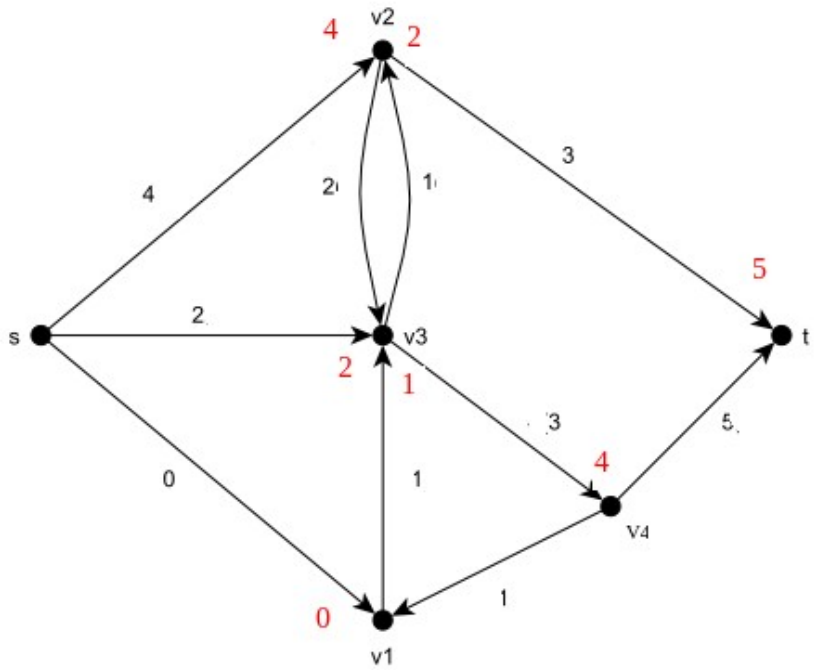


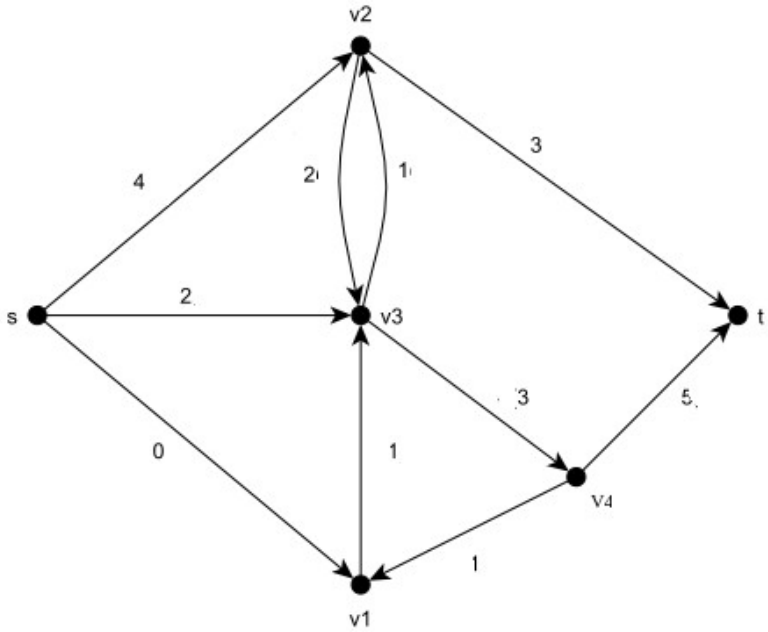
Inicio vértice S

i	0	1	2	3	4	5	6
Vértice	***	a	V1	V3	V2	V4	
s	0	***					
V1	∞	0	***				
V2	∞	4	4	2	***		
V3	∞	2	1	***			
V4	∞	∞	∞	4	4	***	
t	∞	∞	∞	∞	5	5	

Distancia mais curta: 5

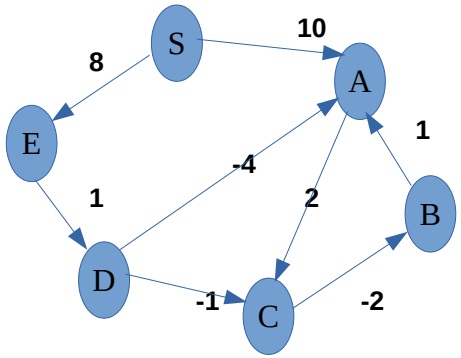
Caminho: s, V1, V3, V2, t





Algoritmo Ford

			Menor		
i	**0**	**1**	**2**	**3**	**4**
S	0	0	0	0	0
A	∞	10	5	5	5
B	∞	10	10	5	5
C	∞	12	8	7	7
D	∞	9	9	9	9
E	∞	8	8	8	8




```
Ford()
    Vinicial.distância = 0;
    Para todos vértices diferente Vinicial
        V.ditancia = inf;
    Enquanto alterar V.distancia
        Seleciona V com menor V.distancia
        Para todos adjacentes U de V
            D = ditancia (U, V);
            Se D < U.distancia
                U.distancia = D;
        Fim
    Fim
Fim
```

i=1

de	S	A	B	C	D	E		
S	0	10				8	S→A (10)	S→E (8)
A				12			A→C (12)	
B							∞	
C			10				C→B (10)	
D							∞	
E					9		E→D (9)	

I=2

de	S	A	B	C	D	E		
S	0	**5**				8	S→A (10)	S→E (8)
A				**8**			A→C (12)	
B							B→A (11)	
C			10				C→B (10)	
D							**D→A(5)**	**D→C(8)**
E					9		E→D (9)	

I=3

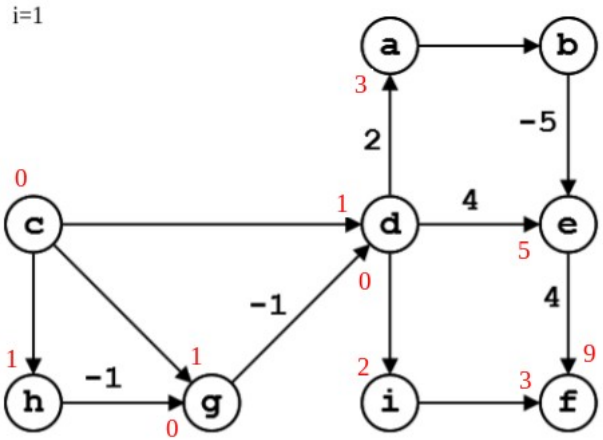
de	S	A	B	C	D	E		
S	0	5				8	S→A (10)	S→E (8)
A				**7**			**A→C (7)**	
B							B→A (11)	
C			**5**				**C→B (5)**	
D							D→A(5)	D→C(8)

I=4

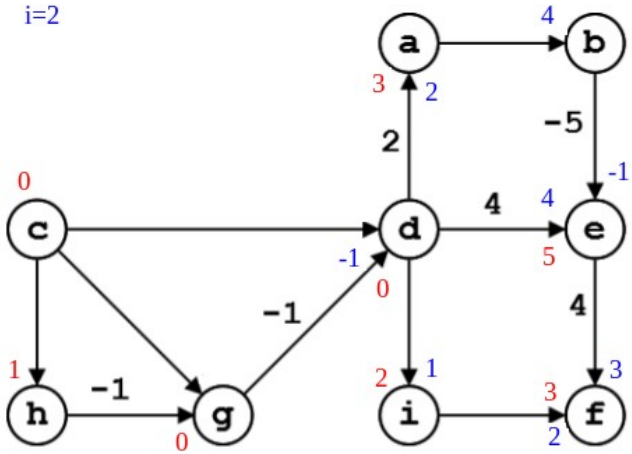
de	S	A	B	C	D	E		
S	0	5				8	S→A (10)	S→E (8)
A				7			**A→C (7)**	
B							B→A (11)	
C			5				**C→B (5)**	
D							D→A(5)	D→C(8)
E					9		E→D (9)	

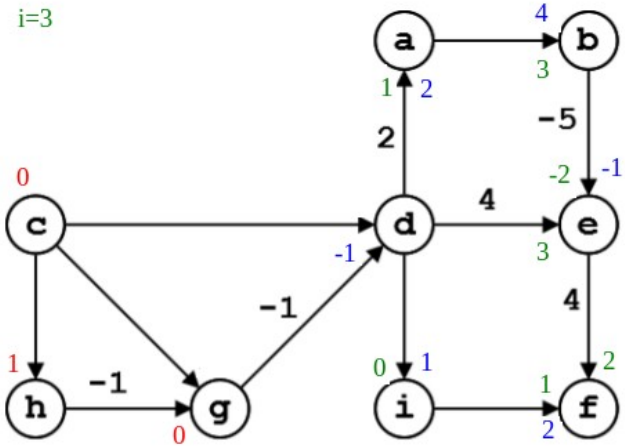
Vértice inicial: c

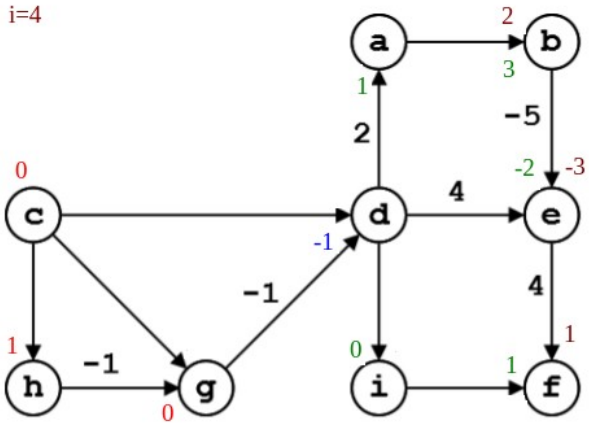
Vértice	Ordem das arestas		
a	ab		
b	be		
c	cd	cg	ch
d	da	de	di
e	ef		
f	***		
g	gd		
h	hg		
i	if		



	Iteração					
Vértice	0	1		2	3	4	5
a	∞	3		2	1		
b	∞			4	3	2	
c	0						
d	∞	1	0	-1			
e	∞	5		-1	-2	-3	
f	∞	9	3	2	1		
g	∞	1	0				
h	∞	1					
i	∞	2		1	0		







Vértice inicial = a

	Iteração			
Vértice	0	1	2	3
a	0	0		
b	∞	2		
c	∞	1		
d	∞	1		
e	∞	3	-1	
f	∞	2		
g	∞	4	3	
h	∞	4		0
i	∞	3		
j	∞	5		1

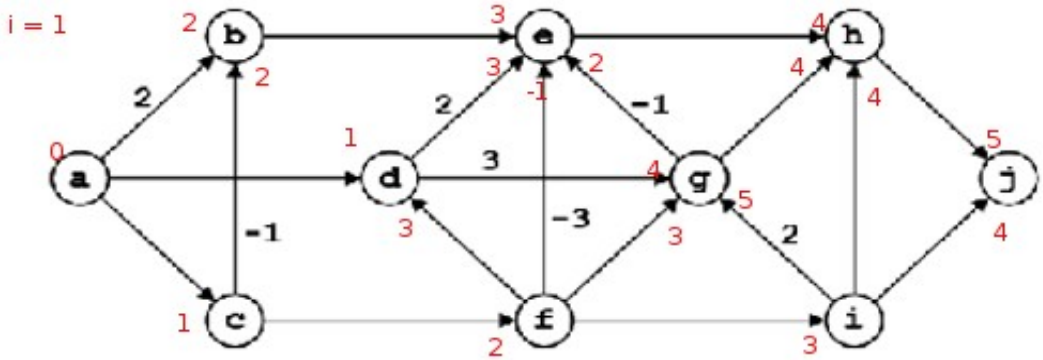
Ordem das arestas			
ab	ac	ad	
be			
cb	cf		
de	dg		
eh			
fd	fe	fg	fi
ge	gh		
hj			
ig	ih	ij	

Caminho mais curto:

a, c, f, e, h, j

Distância = 1

Nota: Ausência de valor nas arestas => 1



i = 1

ab=2; ac=1;ad=1

be=2+1=3

cb=1+1=2; cf=1+1=2

de=1+2=3; dg=4

eh=3+1=4

fd=2+1=3; fe=2-3=-1; fg=2+1=3; fi=2+1=3

ge=3-1=2; gh=3+1=4

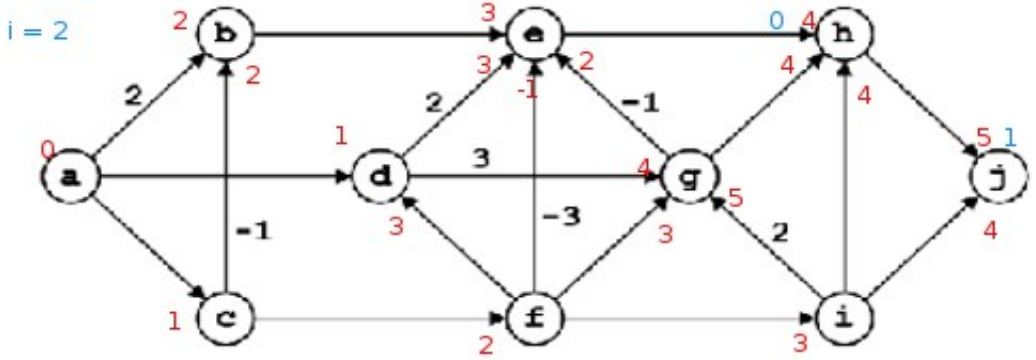
hj=3+1=5

ig=3+2=5; ih=3+1=4; ij=3+1=4

i = 2

eh=-1+1=0

hj=0+1=1



Vértice inicial = S

Ordem das arestas

SA SC
AB
BD BT
CA CB CD
DT

I	0	1	2	3
Vértice				
S	0			
A	∞	5	0	0
B	∞	6	2	1
C	∞	-2		
D	∞	5	2	0
T	∞	9	3	1

Caminho mais curto:

S, C, A, B, D, T

Distância = 1