MAUÁ

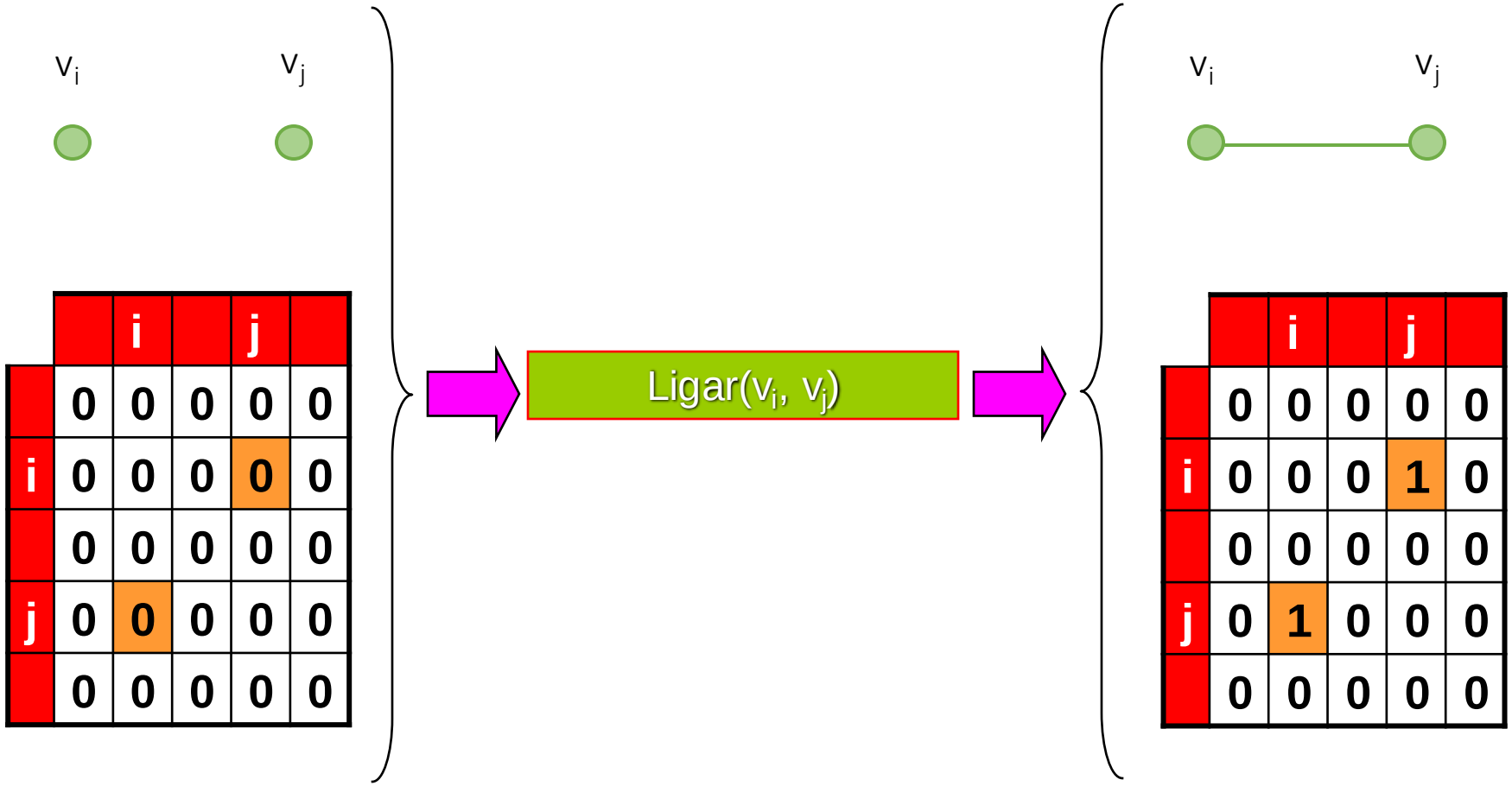
Grafos
Biblioteca

ECM404

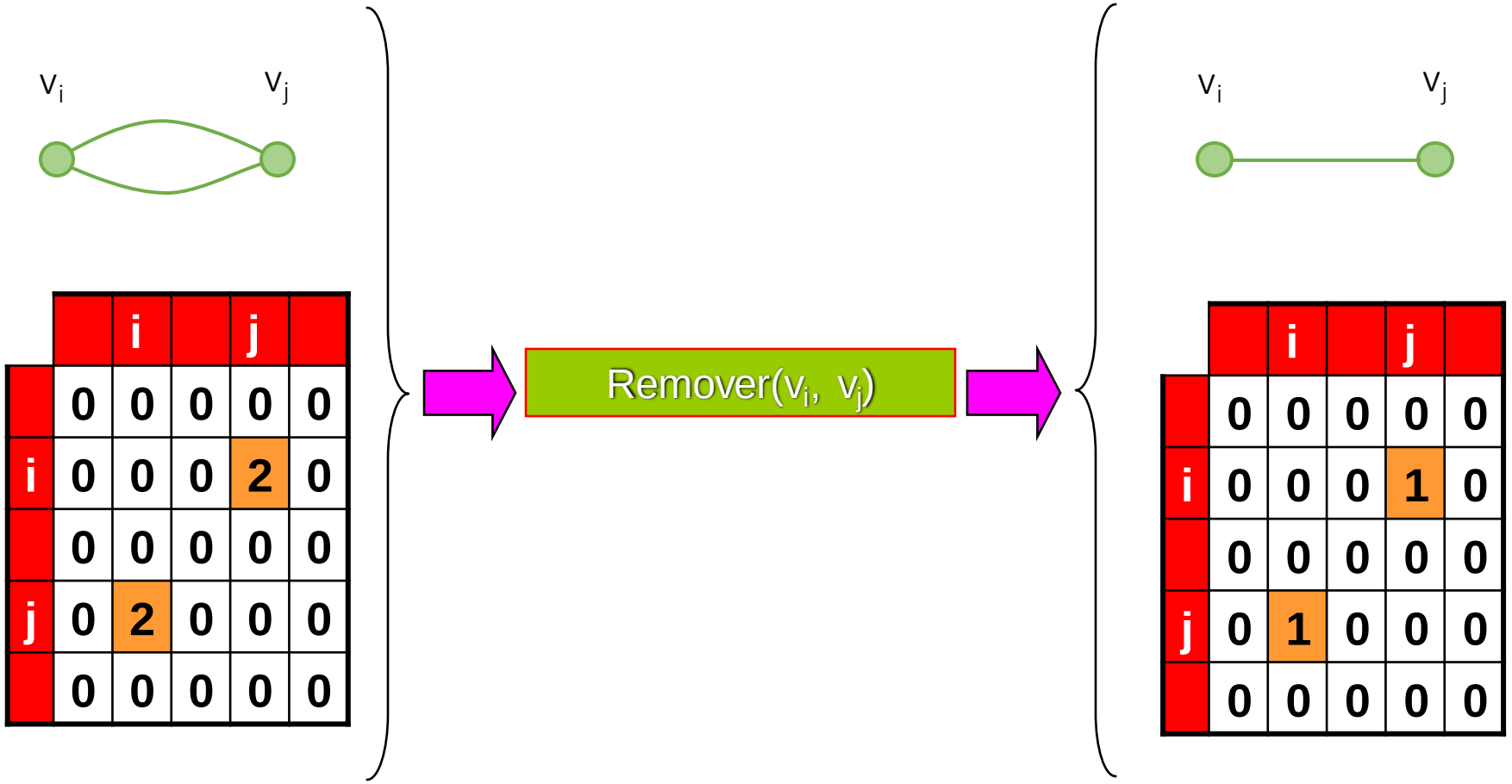
Funções Grafos

```
void CriarGrafo (Grafo *g, int n, int dig);

void InserirAresta (Grafo *g, int de, int para, float peso);
void RemoverAresta (Grafo *g, int de, int para);
int Adjacente (Grafo g, int de, int para);
float PesoDaAresta (Grafo g, int de, int para);

int GrauDeEntrada (Grafo g, int v);
int GrauDeSaida (Grafo g, int v);

int Vertedouro (Grafo g, int v);
int Sorvedouro (Grafo g, int v);

void Warshall (Grafo g, int w[MAX_VERT][MAX_VERT]);
int Alcanca (Grafo g, int de, int para);
```

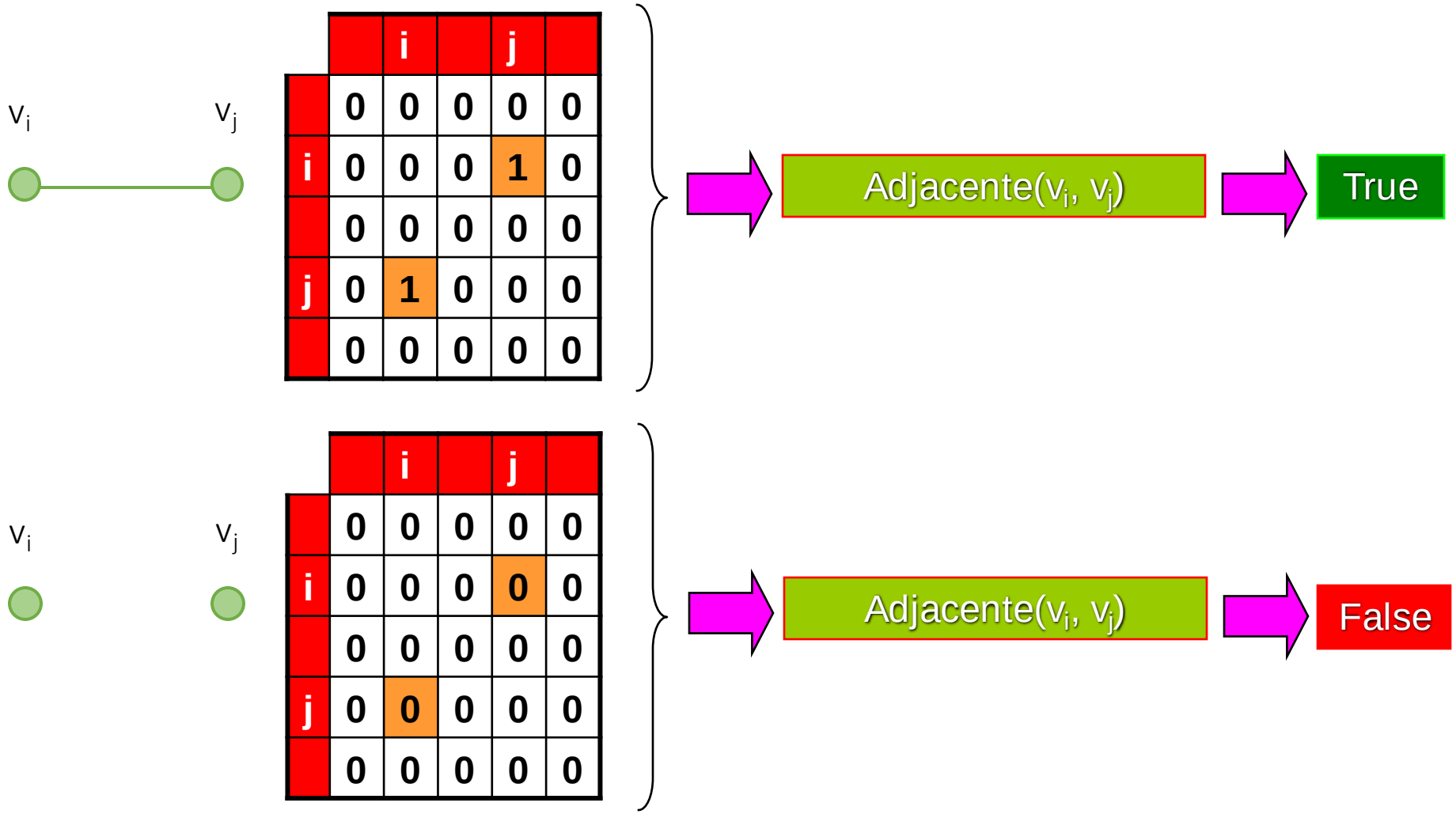
Matriz de Adjacência – Inserir Aresta



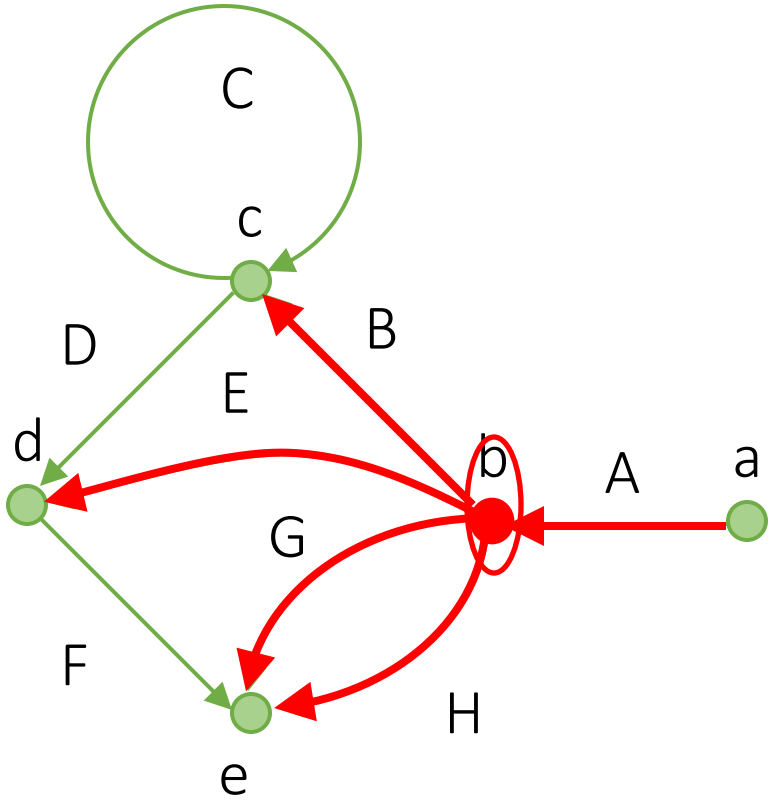
Matriz de Adjacência – Remover Aresta



Matriz de Adjacência – Adjacente



Grau de Entrada e de Saída



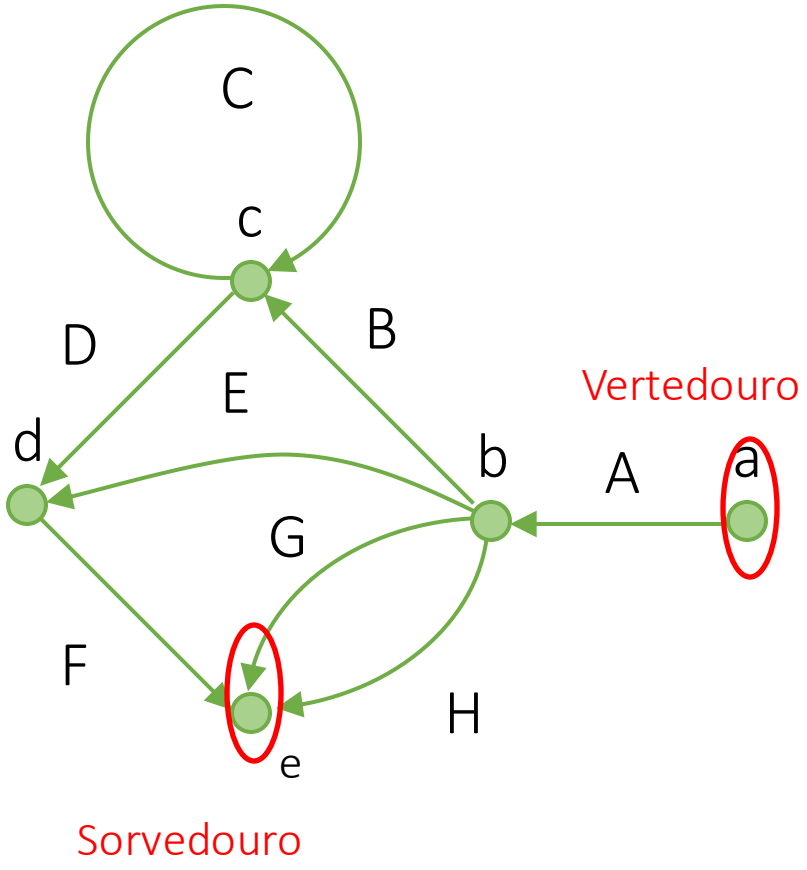
Matriz de adjacências

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	1	2
c	0	0	1	1	0
d	0	0	0	0	1
e	0	0	0	0	0

Grau de saída = 4

Grau de entrada = 1

Vertedouro e Sorvedouro



Matriz de adjacências

	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	1	2
c	0	0	1	1	0
d	0	0	0	0	1
e	0	0	0	0	0

Grau de entrada = 0

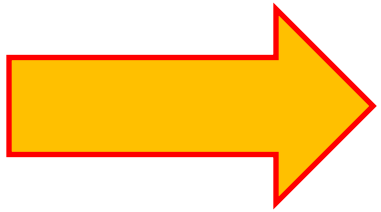
Grau de saída = 0

Algoritmo de Warshall

- Permite verificar quais vértices podem ser alcançáveis;

Matriz de adjacências

g	a	b	c	d	e
a	0	1	0	0	0
b	0	0	1	1	2
c	0	0	1	1	0
d	0	0	0	0	1
e	0	0	0	0	0



Matriz lógica de adjacências

W	a	b	c	d	e
a	F	T	F	F	F
b	F	F	T	T	T
c	F	F	T	T	F
d	F	F	F	F	T
e	F	F	F	F	F

Algoritmo de Warshall

- Permite verificar quais vértices podem ser alcançáveis;

```
Procedimento Warshall (Grafo g, int W[MAX_VERT][MAX_VERT]){
/*
Entrada:
  g: Grafo com matriz de adjacências ou de pesos do dígrafo
Saída:
  W: a matriz "lógica" de vértices Alcançáveis
*/

para i percorrendo todos os vértices
  para j percorrendo todos os vértices
    W[i][j] ← PesoDaAresta(g, i, j) != 0;

para k percorrendo todos os vértices
 para i percorrendo todos os vértices
  para j percorrendo todos os vértices
   W[i][j] ← W[i][j] || (W[i][k] && W[k][j]);
}
```

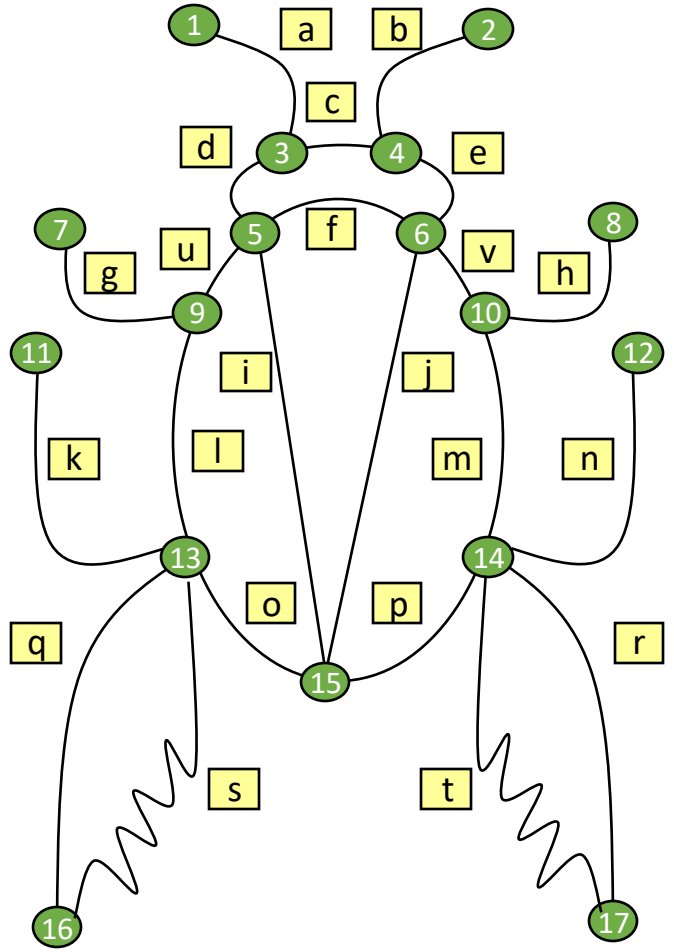
W	a	b	c	d	e
a	F	T	T	T	T
b	F	F	T	T	T
c	F	F	T	T	T
d	F	F	F	F	T
e	F	F	F	F	F

Passeio (walk)

- Sequência não nula, finita e alternada de vértices adjacentes e arestas incidentes.

Exemplos:

- Antenas = 1a3c4b2
- Cabeça = 3c4e6f5d3 (fechado)
- W_1 = 14t17r14n12n14m10
- W_2 = 5f6v10h8h10m14
- Asa Esquerda = 5i15o13l9u5 (fechado)

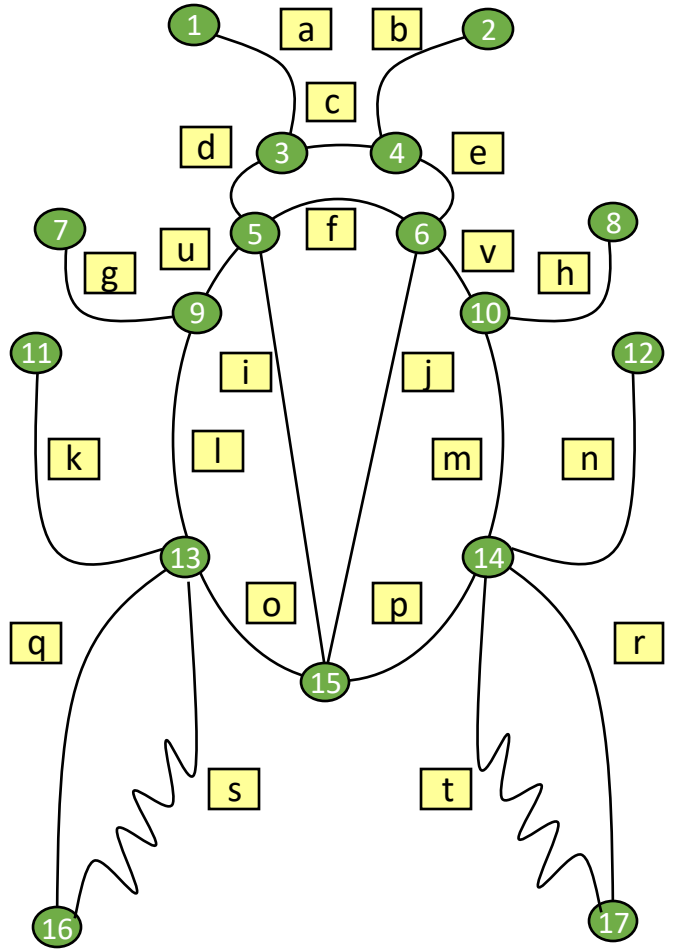


TRAJETO (TRAIL)

- Passeio onde as ***arestas*** não se repetem.

Exemplos:

- Antenas = 1a3c4b2
- Cabeça = 3c4e6f5d3 (fechado)
- T_1 = 2b4e6j15p14t17
- T_2 = 11k13s16q13l9u5i15j6f5
- Patinha Direita Central = 12n14

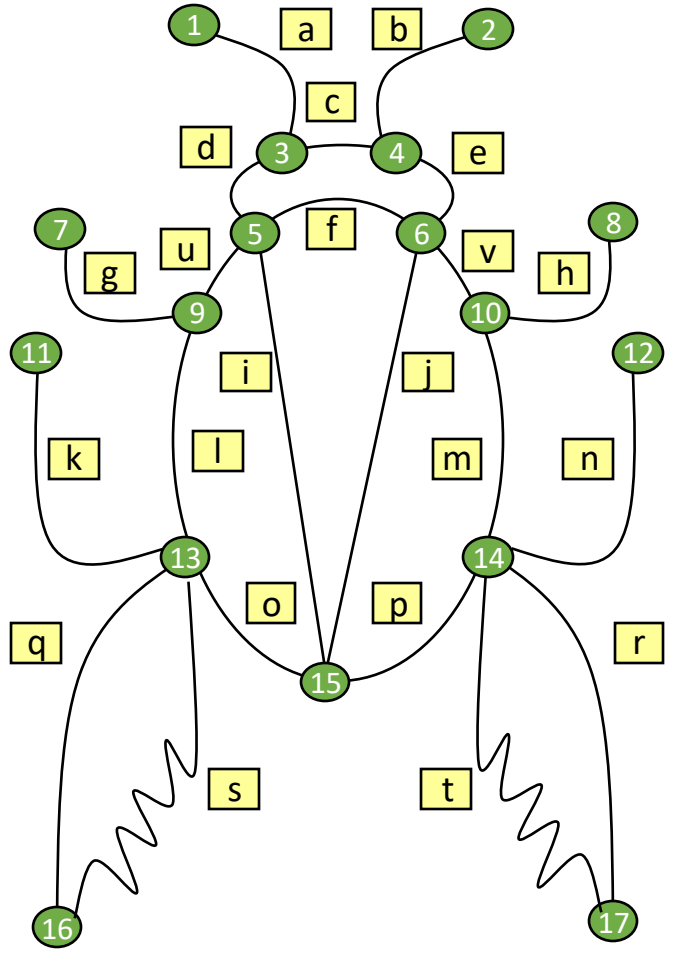


Caminho (path)

- Passeio onde os ***vértices*** não se repetem.

Exemplos:

- Antenas = 1a3c4b2
- P_1 = 2b4e6j15p14t17
- P_2 = 11k13o15j6v10m14t17
- P_1 = 8h10m14
- Patinha Direita Central = 12n14

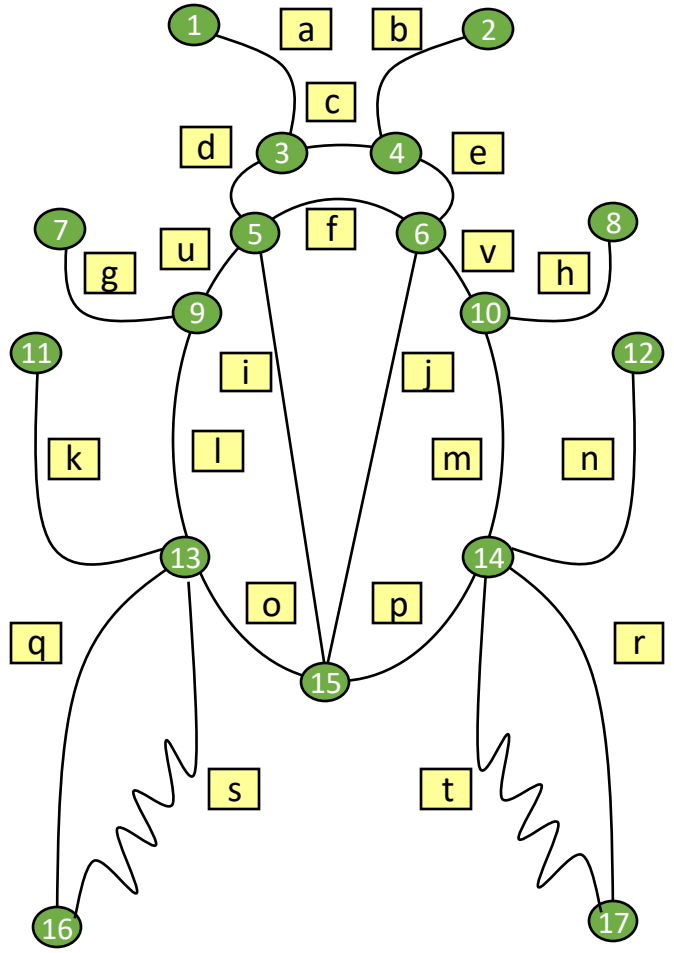


Ciclo (cycle)

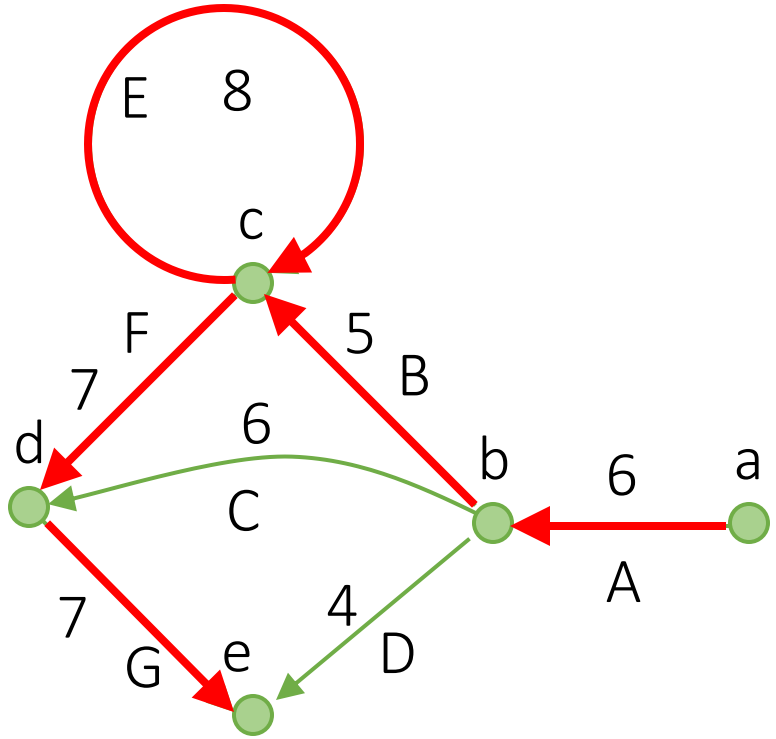
- Trajeto fechado.

Exemplos:

- Cabeça = 3c4e6f5d3 (fechado)
- Asa Esquerda = 5i15o13l9u5 (fechado)
- Patona Direita = 14t17r14 (fechado)
- C_1 = 6v10m14t17r14p15j6 (fechado)
- Tronco = 5f6v10m14p15o13l9u5 (fechado)



Peso do Passeio



Matriz de pesos

	a	b	c	d	e
a	0	6	0	0	0
b	0	0	5	6	4
c	0	0	8	7	0
d	0	0	0	0	7
e	0	0	0	0	0

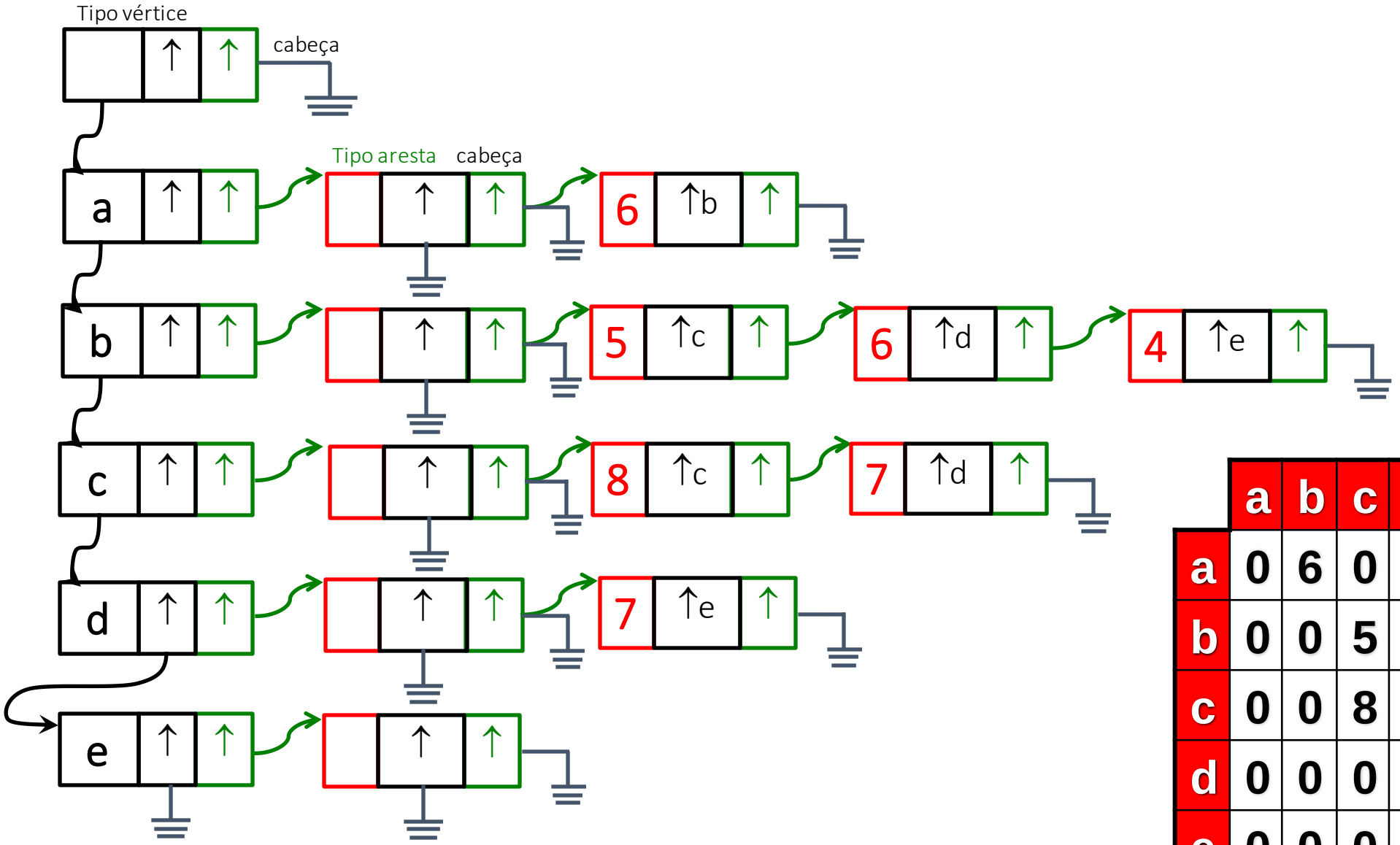
aAbBcEcFdGe = 33

Grafo com Lista Ligada

- A implementação de um grafo como uma lista ligada evita um grande uso de memória, uma vez que a implementação com matriz pode conter uma grande quantidade de zeros.

	a	b	c	d	e
a	0	6	0	0	0
b	0	0	5	6	4
c	0	0	8	7	0
d	0	0	0	0	7
e	0	0	0	0	0

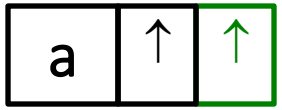
Grafo com Lista Ligada

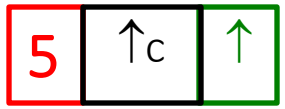


	a	b	c	d	e
a	0	6	0	0	0
b	0	0	5	6	4
c	0	0	8	7	0
d	0	0	0	0	7
e	0	0	0	0	0

Grafo com Lista Ligada

```
typedef struct SVertice;
typedef struct SAresta;

typedef struct SVertice{
    int id;
    struct SVertice *proxVert;
    struct SAresta *proxAresta;
} TVertice;

typedef struct SAresta{
    float peso;
    struct SVertice *destino;
    struct SAresta *proxAresta;
} TAresta;
```





Grafo.h

```
#ifndef GRAFO_H_INCLUDED
#define GRAFO_H_INCLUDED

#define MAX_VERT 100
#define MAX_VERT_LISTA 100

typedef struct{
    float Pesos[MAX_VERT][MAX_VERT];
    int nVertices;
    int digrafo;
} Grafo;

typedef struct{
    int vertices[MAX_VERT_LISTA];
    int nVertices;
} ListaDeVertices;
```

Grafo.h

```
void CriarGrafo (Grafo *g, int n, int dig);
void InserirAresta (Grafo *g, int de, int para, float peso);
void RemoverAresta (Grafo *g, int de, int para);
float PesoDaAresta (Grafo g, int de, int para);

int GrauDeEntrada (Grafo g, int v);
int GrauDeSaida (Grafo g, int v);

int Vertedouro (Grafo g, int v);
int Sorvedouro (Grafo g, int v);

int Adjacente (Grafo g, int de, int para);

void Warshall (Grafo g, int w[MAX_VERT][MAX_VERT]);
int Alcanca (Grafo g, int de, int para);

float PesoDoPasseio (Grafo g, ListaDeVertices p);

#endif // GRAFO_H_INCLUDED
```