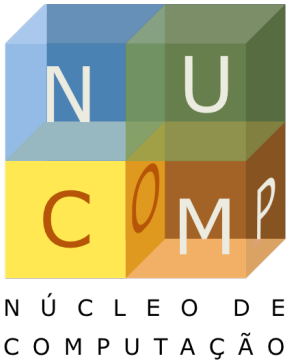
UNIVERSIDADE DO ESTADO DO AMAZONAS
ESCOLA SUPERIOR DE TECNOLOGIA
NÚCLEO DE COMPUTAÇÃO - NUCOMP



Algoritmos e Estruturas de Dados II

Tipo Abstrato de Dados Grafo

Prof. Flávio José Mendes Coelho
fcoelho@uea.edu.br

Plano de aula

1. Lista de adjacência
2. Matriz de adjacência

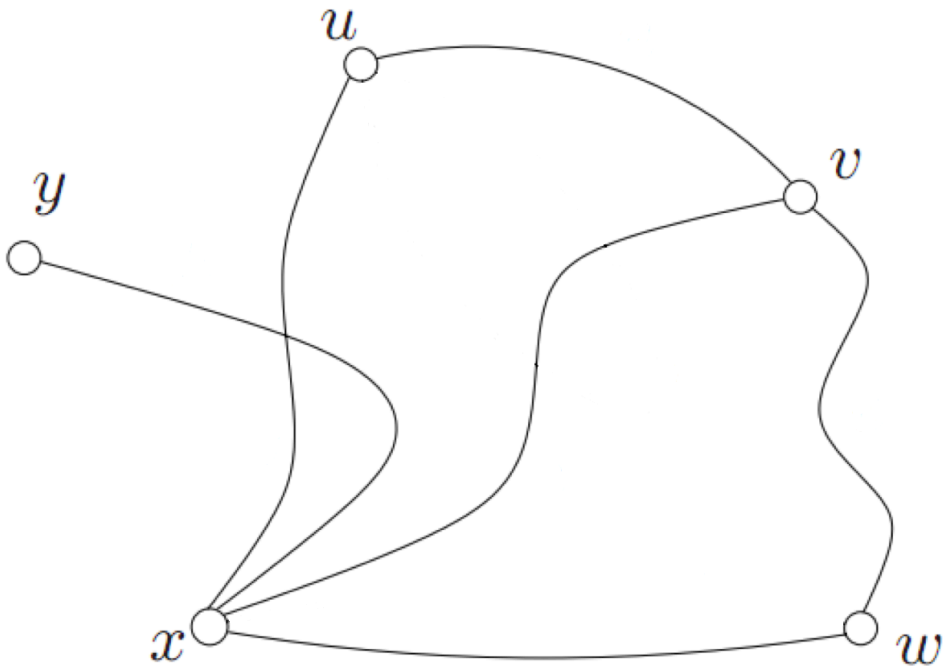
Lista de adjacência

Lista de adjacência

A lista de adjacência de um grafo G é a lista que associa cada vértice $v \in V$ de G à sua vizinhança $N_G(v)$.

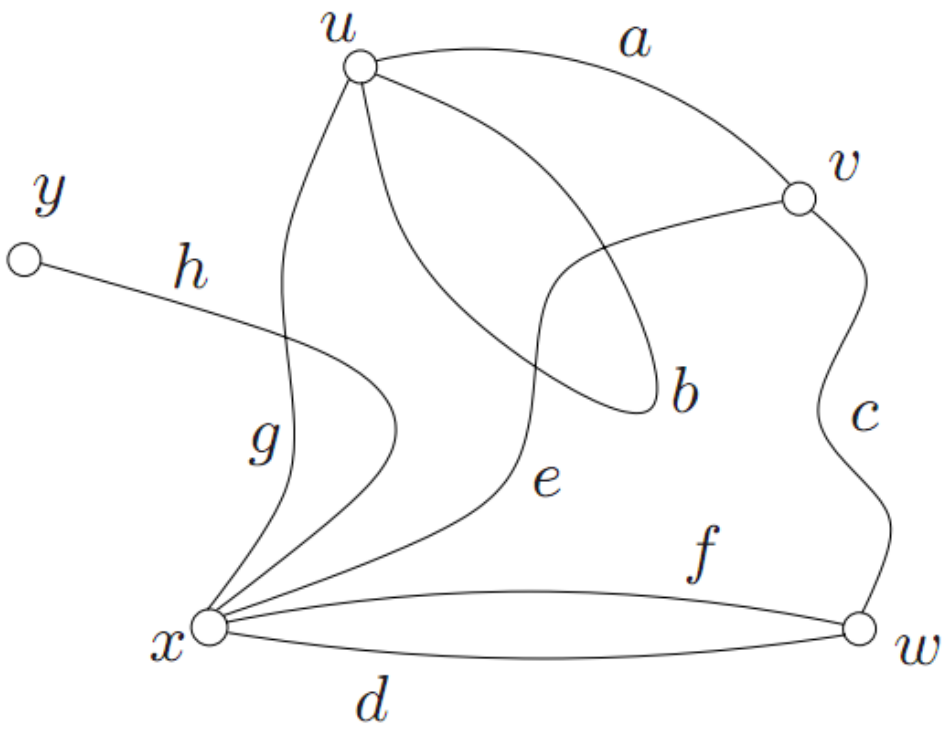
Usualmente, grafos simples são armazenados por meio de listas de adjacência.

Lista de adjacência



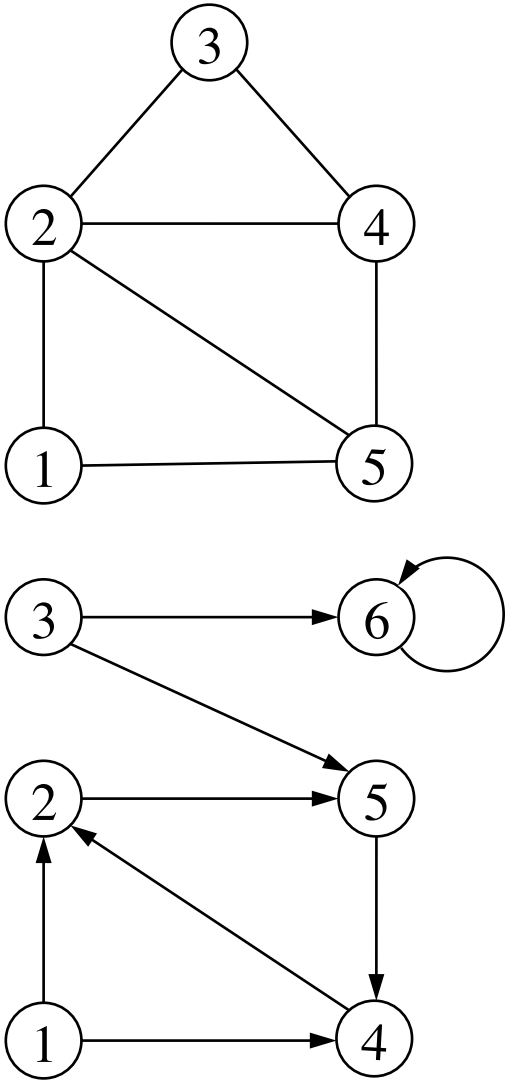
	$N_G(v),\ v \in V$
u	$\{x, v\}$
v	$\{u, x, w\}$
w	$\{v, x\}$
x	$\{y, u, v, w\}$
y	$\{x\}$

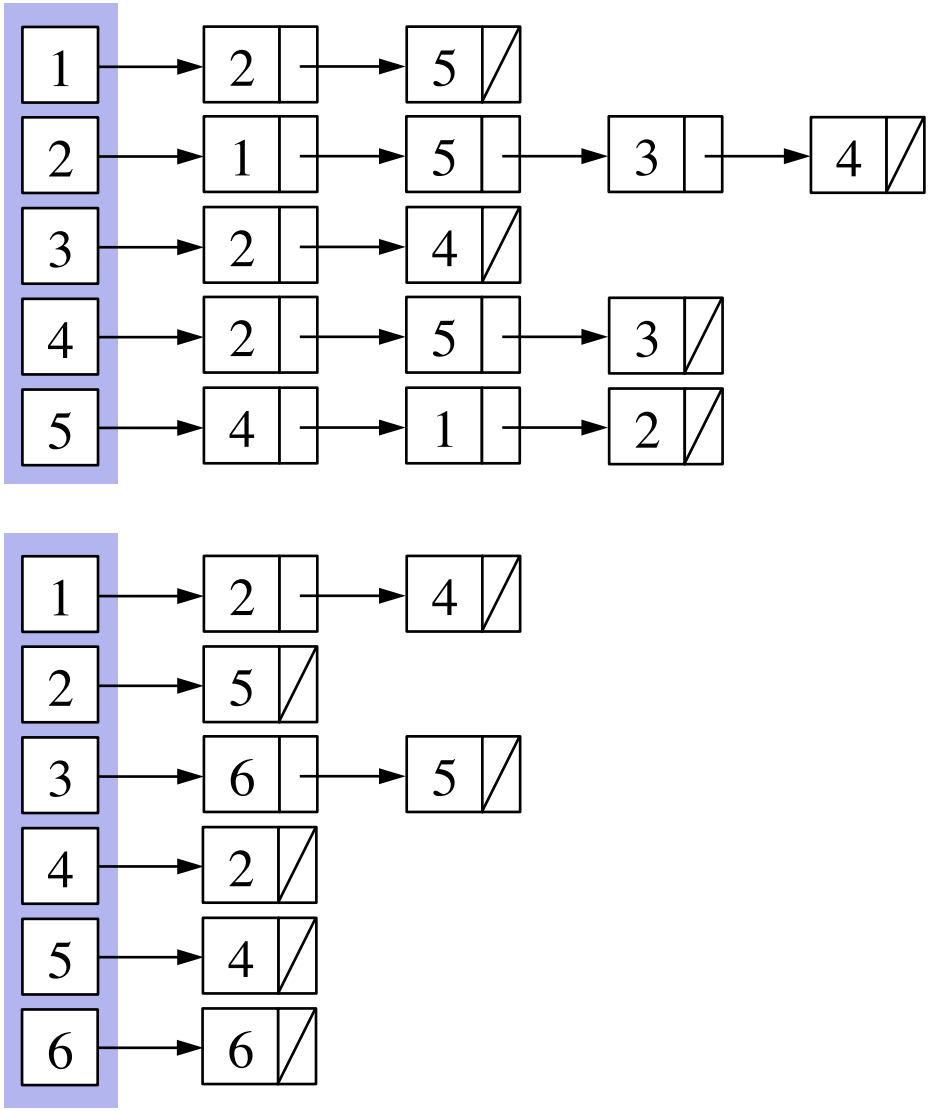
Lista de adjacência



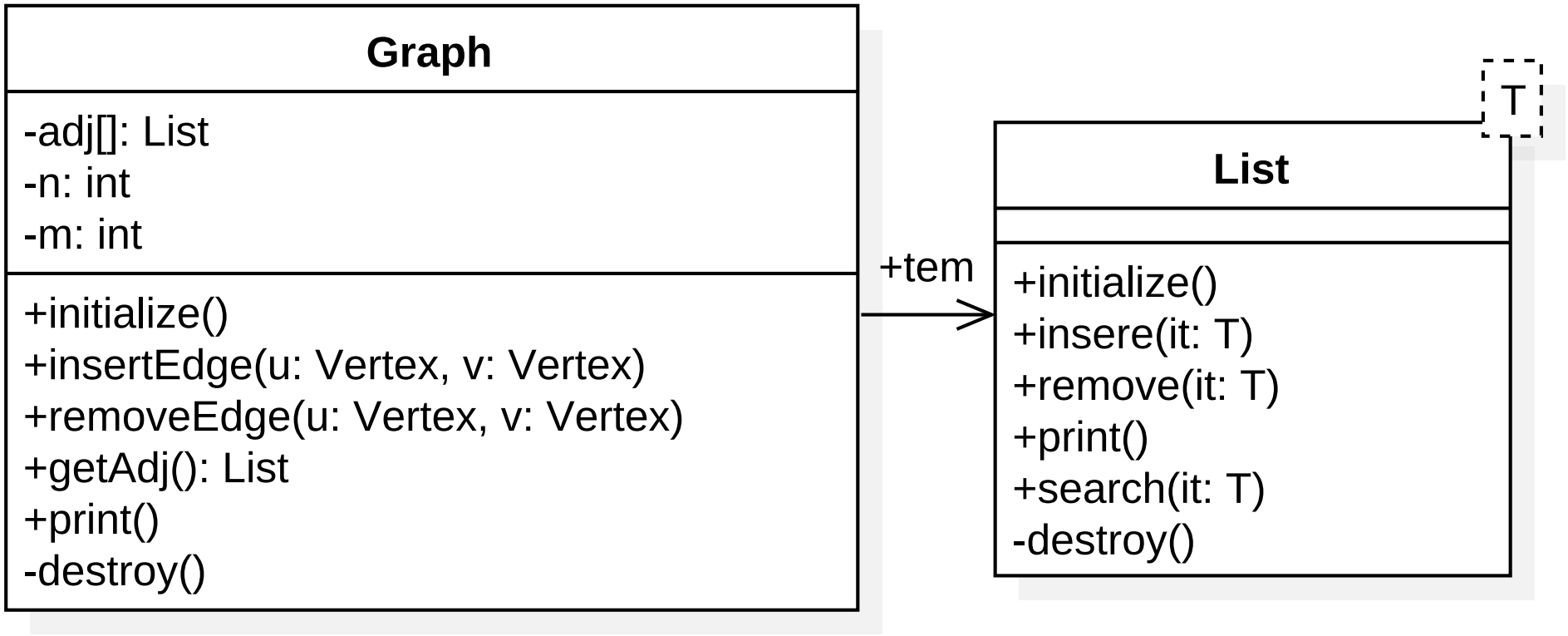
	$N_G(v),\ v \in V$
u	$\{x_g, u_b, v_a\}$
v	$\{u_a, x_e, w_c\}$
w	$\{v_c, x_f, x_d\}$
x	$\{y_h, u_g, v_e, w_f, w_d\}$
y	$\{x_h\}$

Lista de adjacência (computacional)





Lista de adjacência (projeto)



Lista de adjacência (código)

```
typedef int Vertex;

// Utilizamos a classe Lista (lista encadeada)
```

Lista de adjacência (código)

```
class Graph { // Não-direcionado
    List<Vertex>  *adj;
    int n, m; // ordem e tamanho
    void destroy();
public:
    Graph(int); // construtor
    void initialize(int);
    void insertEdge(Vertex, Vertex);
    void print();
    // métodos get/set para n, m e adj.
};
```

Lista de adjacência (código)

```
void Graph::Graph (int n) {
    initialize(n);
}

void Graph::initialize(int n) {
    if (this->n != 0) destroy();
    this->n = n;
    adj = new List<Vertex>[n+1]; // Vetor usa
                                 // células de 1..n
}
```

Lista de adjacência (código)

```
void Graph::insertEdge(Vertex u, Vertex v) {
    Item x = {v}; // chave = vértice
    adj[u].insert(x);  // Insere na lista
    x = {u};
    adj[v].insert(x);  // Insere na lista
    m++;
}
```

Lista de adjacência (código)

```
void Graph::print() {
    for (int i = 1; i <= n; i++) {
        cout << "v[" << i << "] = ";
        adj[i].print();
    }
}
```

Lista de adjacência (código)

```
void Graph::destroy() {
    for (int i = 0; i <= n; i++) {
        adj[i].destroy();    // destroi lista
    }
    delete( adj );
    n = m = 0;
}
```

Lista de adjacência (código)

```
// Função auxiliar
void testaGrafo(Grafo &g) {
    g.insertEdge(1, 2);
    g.insertEdge(2, 3);
    g.insertEdge(3, 4);
    g.insertEdge(4, 5);
    g.insertEdge(5, 1);
    g.insertEdge(5, 2);
    g.insertEdge(2, 4);
    g.print();
}
```

Lista de adjacência (código)

```
int main(int argc, const char * argv[]) {
    int n, m;
    cout << "ordem: "; cin >> n;
    Grafo g(n);
    testaGrafo(g);

    return 0;
}
```

Matriz de adjacência

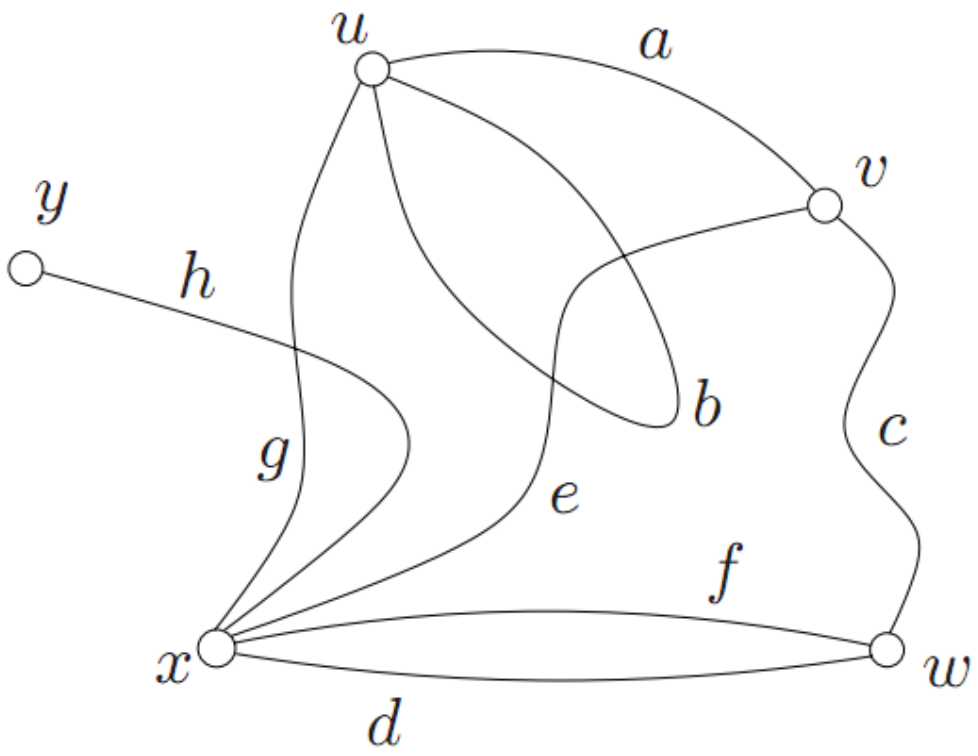
Matriz de adjacência

Seja $G = (V, E)$. A matriz de adjacência de G é a matriz nxn

$$A_G = (a_{uv}),$$

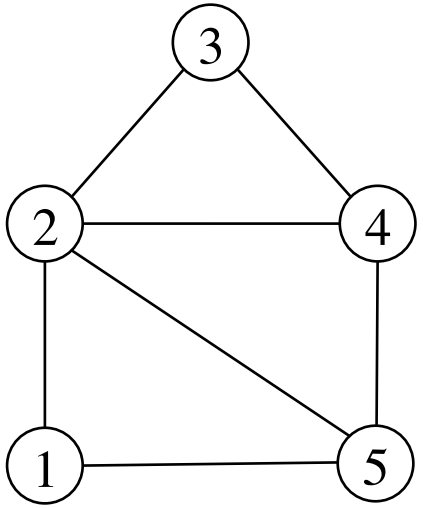
onde $\boldsymbol{a_{uv}}$ é o número de arestas unindo os vértices $\boldsymbol{u}$ e $\boldsymbol{v}$, sendo cada laço contado duas vezes.

Matriz de adjacência

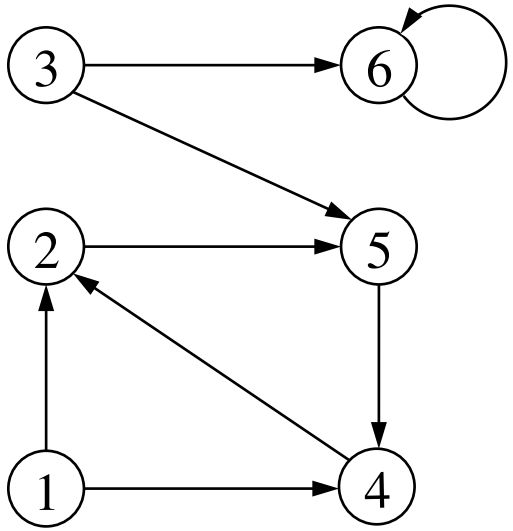


	u	v	w	x	y
u	2	1	0	1	0
v	1	0	1	1	0
w	0	1	0	2	0
x	1	1	2	0	1
y	0	0	0	1	0

Matriz de adjacência (computacional)



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Lista de adjacência (código)

```
#define NUMVERTICES 100

typedef int Vertex;
typedef int Weight;   // Peso normalmente é 0 ou1
struct Graph {
   // Matriz usa as linhas 1..n e colunas 1..n, e não
   // utiliza a linha 0 e coluna 0.
   Weight mat[NUMVERTICES+1][NUMVERTICES+1];
   int n, m;
};
```

Lista de adjacência (código)

```
void initialize(Graph &g) {
    for (int i = 0; i <= g.n; i++) {
        for(int j = 0; j<= g.n; j++)
            g.mat[i][j] = 0;
    }
}
```

Lista de adjacência (código)

```
void insertEdge(Vertex u, Vertex v, Weight w, Graph
&g) {
    g.mat[u][v] = w;
    g.mat[v][u] = w;
    g.m++;
}
```

Lista de adjacência (código)

```
void print(Graph g) {
  int k = 3; // largura de campo
  cout << "   ";
  for (int j = 1; j <= g.n; j++) cout << setw(k) << j;
  cout << endl;
  for (int j = 1; j <= g.n*k + 3; j++) cout << "-";
  cout << endl;
  for (int i = 1; i <= g.n; i++) {
        cout << setw(1) << i; cout << " |";
        for (int j = 1; j <= g.n; j++)
            cout << setw(k) << g.mat[i][j];
        cout << endl; }}
```

Bibliografia

- BONDY, J.; MURTY, U. ***Graduate Texts in Mathematics series: Graph Theory***. Springer, 2008. Volume 244.
- DEITEL, Harvey; DEITEL, Paul. ***C++ How to Program***. 5ª Edição. Prentice Hall, 2004.
- DIESTEL, Reinhard. ***Graduate Texts in Mathematics series: Graph Theory***. New York: Springer-Verlag, 2000. Volume 173.
- CORMEM, Thomas; LEISERSON, Charles; RIVEST, Ronald; STEIN, Clifford. ***Algoritmos: teoria e prática***. Tradução da terceira edição Americana. Rio de Janeiro: Elsevier, 2012.
- CORMEN, T. H. , LEISERSON, C. E., RIVEST, R.L., STEIN, C. ***Introduction to Algorithms***, 3rd edition, MIT Press, 2009.

Bibliografia

- LEVITIN, Anany. ***Introduction to the Design and Analysis of Algorithms***. 3a Edição. USA: Addison Wesley, 2011.
- SKIENA, Steven. ***The Algorithm Design Manual***. 4a Edição. London: Springer-Verlag, 2008.
- SZWARCFITER, J. ***Grafos e Algoritmos Computacionais***. Rio de Janeiro: 2ª. Ed. Campus, 1986.
- ZIVIANI, Nivio. ***Projeto de Algoritmos com Implementação em Pascal e C***. 3a Edição revisada e ampliada de 2010. São Paulo: Thomson Learning, 2010.