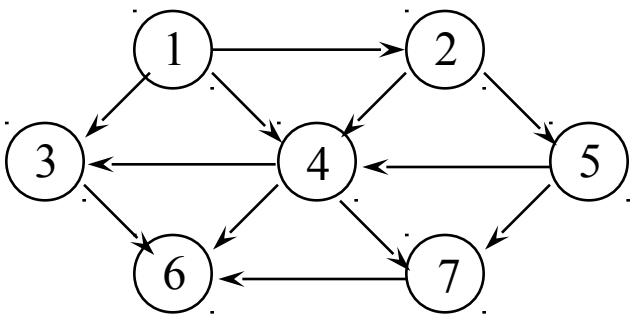
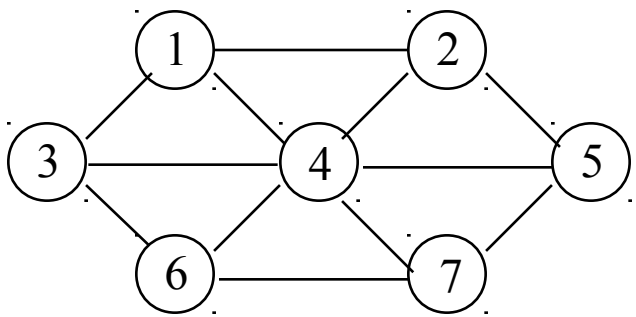
Grafos

- Grafo G = (V, E)
  - V — conjunto de vértices
  - E — conjunto de arestas (ou arcos)
    - cada aresta é um par de vértices (v, w), em que v, w ∈ V
    - se o par for ordenado, o grafo é dirigido, ou digrafo
    - um vértice w é adjacente a um vértice v se e só se (v, w) ∈ E
    - num grafo não dirigido com aresta (v, w) e, logo, (w, v) w é adjacente a v e v adjacente a w
    - as arestas têm por vezes associado um custo ou peso



G1



G2

Mais definições

- **caminho** — sequência de vértices $v_1, v_2, \ldots, v_n$ tais que $(v_i, v_{i+1}) \in E$, $1 \leq i < n$
  - comprimento do caminho é o número de arestas, n-1
    - se n = 1, o caminho reduz-se a um vértice v_1; comprimento = 0
  - laço — caminho $v, v \Rightarrow (v, v) \in E$, comprimento 1;
  - caminho simples — todos os vértices distintos

- **ciclo** — caminho de comprimento ≥ 1 com v1 = vn
  - num grafo não dirigido requer-se que as arestas sejam diferentes
  - DAG — grafo dirigido acíclico
- **conectividade**
  - grafo não dirigido é conexo sse houver um caminho entre qualquer par de vértices
  - digrafo com a mesma propriedade — fortemente conexo
- **densidade**
  - grafo completo — existe uma aresta entre qualquer par de nós
  - grafo denso — $|E| = \Theta(V^2)$
  - grafo esparso — $|E| = \Theta(V)$

Representação

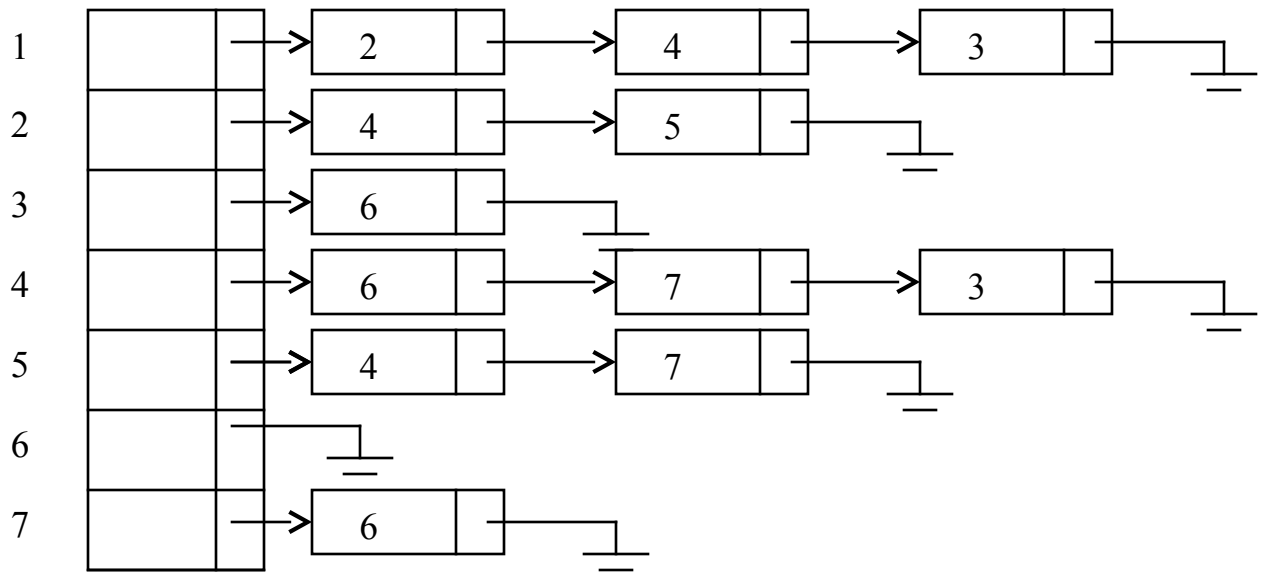
- matriz de adjacências

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

  - a[u][v] = 1 sse $(u, v) \in E$
  - elementos da matriz podem ser os pesos
  - apropriada para grafos densos

Lista de adjacências

- estrutura típica para grafos esparsos
  - para cada vértice, mantém-se a lista dos vértices adjacentes
  - vetor de cabeças de lista, indexado pelos vértices
  - espaço é O(|E| + |V|)
  - pesquisa dos adjacentes em tempo proporcional ao número destes



Grafo Matriz

```
class GrafoMatriz{

        private:
                bool** MatrizAdj;
                 int verticeCont;

        public:
                GrafoMatriz(int verticeNum);
                void adAresta(int i, int j) ;
                void removeAresta(int i, int j);
                bool existeAresta(int i, int j);
                void imprimeGrafo();
                void imprimeMatrizGrafo();
                ~GrafoMatriz();
};
```

Grafo Matriz

```
GrafoMatriz::GrafoMatriz (int verticeNum) {

        verticeCont = verticeNum;
       MatrizAdj = new bool*[verticeCont];

        for (int i = 0; i < verticeCont; i++) {

             MatrizAdj[i] = new bool[verticeCont];
             for (int j = 0; j < verticeCont; j++)
                        MatrizAdj[i][j] = false;
        }

}
```

Grafo Matriz

```
GrafoMatriz::~GrafoMatriz() {

        for (int i = 0; i < verticeCont; i++)
                delete[] MatrizAdj[i];

         delete[] MatrizAdj;

}
```

Grafo Matriz

```
void GrafoMatriz::imprimeGrafo() {

        for (int i = 0; i < verticeCont; i++)
            for (int j = i; j < verticeCont ; j++)
                if (MatrizAdj[i][j])
                    cout<<" Aresta " << i << " " << j << "\n";

}
```

Grafo Matriz

```
main() {

        GrafoMatriz G(4);

        G.adAresta(1,2);
        G.adAresta(0,3);
        G.adAresta(2,3);
        G.adAresta(0,1);

        G.imprimeGrafo();
        .
        .
        .
```

Grafo Matriz

```
main( ){...
        G.imprimeMatrizGrafo();



        G.removeAresta(0,3);
        G.imprimeGrafo();
        G.imprimeMatrizGrafo();

        if (G.existeAresta(3,2))
            cout<< "\nSao adjacentes\n\n ";
        else
            cout << "\nNao sao adjacentes\n\n";
        .
        .
}
```

Grafo Lista

```
#include "lista.h"
class GrafoListaAdj{
        private:
                 lista* vizinhos;
                int verticeNum;
        public:
                GrafoListaAdj(int vertices);
                void adAresta(int i, int j) ;
                void removeAresta(int i, int j);
                bool existeAresta(int i, int j);
                void imprimeGrafo();
                ~GrafoListaAdj();

};
```

Grafo Lista

```
GrafoListaAdj::GrafoListaAdj(int Nvertices){

        if (Nvertices > 0) {

                vizinhos = new lista[ Nvertices ];
                verticeNum = Nvertices;
        }
}
```

Grafo Lista

```
GrafoListaAdj::~GrafoListaAdj(){

        delete [] vizinhos;
}
```

Grafo Lista

```
void GrafoListaAdj::imprimeGrafo(){

        cout << "\n\n Grafo G: \n ";

        for ( int i = 0; i < verticeNum; i++ ){

                cout << "\n Vizinhos [ " << i  << "]: " ;
                vizinhos[i].imprime();
        }

}
```

Grafo Lista

```
main() {

        GrafoListaAdj G(4);

        G.adAresta(1,2);
        G.adAresta(0,3);
        G.adAresta(2,3);
        G.adAresta(0,1);

        G.imprimeGrafo();
        .
        .
        .
```

Grafo Lista

```
main( ){...

        G.removeAresta(0,3);
        G.imprimeGrafo();


        if (G.existeAresta(3,2))
            cout<< "\nSao adjacentes\n\n ";
        else
            cout << "\nNao sao adjacentes\n\n";
        .
        .
}
```