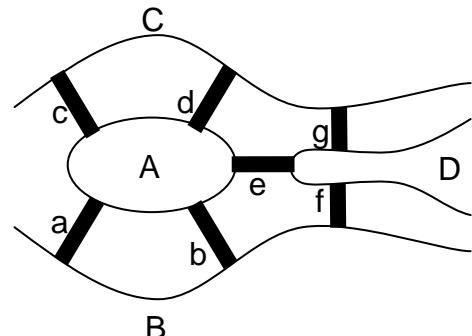


# GRAFOS

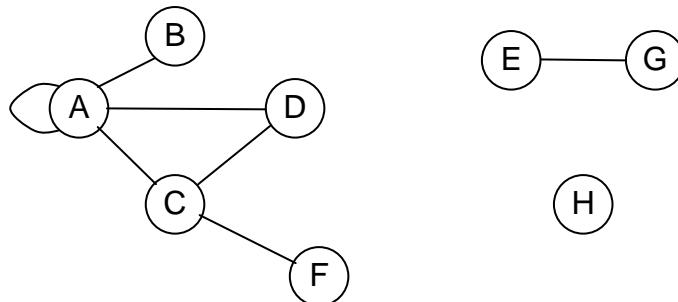
- Estrutura de dados
- História:
  - Problema das pontes de Koenigsberg (enunciado abaixo)
  - Solucionado por Euler em 1736 utilizando grafos

Determinar se, a partir de alguma área de terra (A, B, C, D), é possível atravessar todas as pontes (a, b, c, d, e, f, g) exatamente uma vez e retornar à área de terra inicial (veja figura ao lado)<sup>1</sup>.



## Terminologia

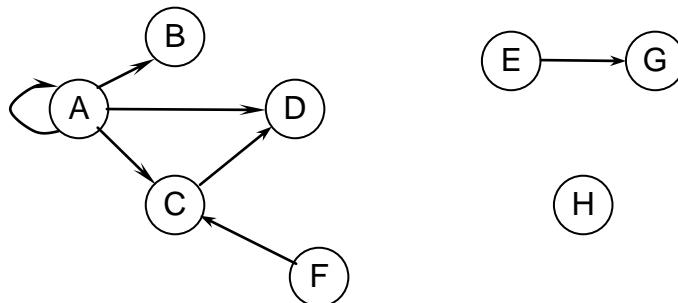
Um grafo consiste de um conjunto de nós (ou vértices) e um conjunto de arcos (ou bordas). Cada arco em um grafo é especificado por um par de nós. Exemplo:



$$\text{Nós} = \{ A, B, C, D, E, F, G, H \}$$

$$\text{Arcos} = \{ (A,B), (A,D), (A,C), (C,D), (C,F), (E,G), (A,A) \}$$

Se os pares de nós que constituem os arcos são ordenados, o grafo é dito ser um grafo dirigido (ou dígrafo). Ex.:



$$\text{Arcos} = \{ <A,B>, <A,D>, <A,C>, <C,D>, <F,C>, <E,G>, <A,A> \}$$

No caso de <A,B>, A é chamado de cauda e B de cabeça.

<sup>1</sup> Resposta: não é possível!

## Observações:

- Uma árvore é um grafo
- Um nó não necessita possuir arcos associados.

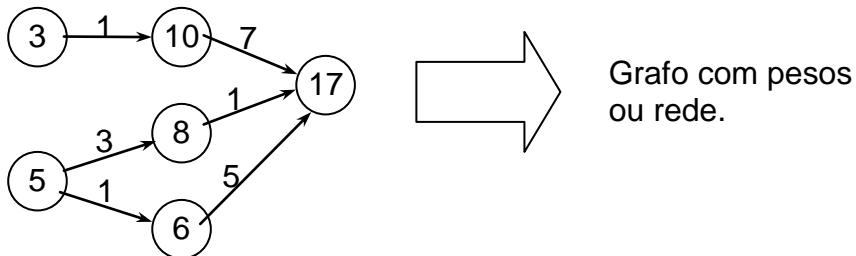
Sendo  $(n_1, n_2)$  um arco, dizemos que os nós  $n_1$  e  $n_2$  são adjacentes e que o arco  $(n_1, n_2)$  é incidente nos nós  $n_1$  e  $n_2$ .

O grau de um nó é o número de arcos incidentes nele. O grau de entrada de um nó  $n$  é o número de arcos que possuem  $n$  como cabeça, e o grau de saída de  $n$  é o número de arcos que possuem  $n$  como cauda (dígrafo).

Se  $A$  é adjacente a  $B$  ( $\langle A, B \rangle$ ),  $B$  é chamado sucessor de  $A$ , e  $A$  um predecessor de  $B$ .

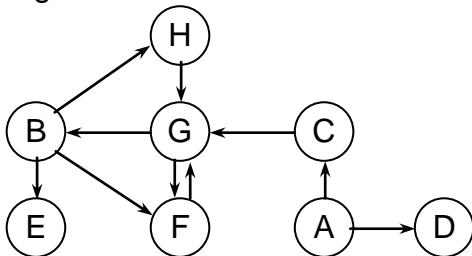
Uma relação  $R$  sobre um conjunto  $A$  é um conjunto de pares ordenados de elementos de  $A$ . Por exemplo, se  $A = \{3, 5, 6, 8, 10, 17\}$ , o conjunto  $R = \{\langle 3, 10 \rangle, \langle 5, 6 \rangle, \langle 5, 8 \rangle, \langle 6, 17 \rangle, \langle 8, 17 \rangle, \langle 10, 17 \rangle\}$  é uma relação. Se  $\langle x, y \rangle$  é um membro de uma relação  $R$ ,  $x$  é dito estar relacionado a  $y$  em  $R$ .

A relação acima pode ser descrita por:  $x$  está relacionado com  $y$  se  $x$  é menor que  $y$  e o resto da divisão de  $y$  por  $x$  é ímpar. Exemplo:



Um caminho de comprimento  $K$  do nó  $A$  para o nó  $B$  é definido como uma seqüência de  $K + 1$  nós,  $n_1, n_2, \dots, n_{k+1}$  tal que  $n_1 = A$ ,  $n_{k+1} = B$  e adjacente  $(n_i, n_{i+1})$  é verdadeiro para todo  $i$  entre 1 e  $K$ . Se para algum inteiro  $K$ , um caminho de tamanho  $K$  existe entre  $A$  e  $B$ , então há um caminho de  $A$  para  $B$ . Um caminho de um nó para ele mesmo é chamado ciclo. Se um grafo contém um ciclo, ele é cíclico, caso contrário ele é acíclico.

Considere o dígrafo:



Existe um caminho de tamanho 1 de  $A$  para  $C$ , dois caminhos de tamanho 2 de  $B$  para  $G$ , e um caminho de tamanho 3 de  $A$  para  $F$ . Não existe caminho de  $B$  para  $C$ . Existem ciclos de  $B$  para  $B$ , de  $F$  para  $F$  e de  $H$  para  $H$ .

Caminho simples é aquele em que são diferentes todos os nós com a possível exceção do primeiro e do último.

## Operações sobre grafos

Algumas operações básicas sobre grafos:

1. inserir (A,B) → insere o arco de A até B, se ainda não existir;
2. eliminar (A,B) → elimina o arco de A até B, se existir;
3. adjacente (A,B) → verifica se o arco de A até B existe ou não (V ou F).

**Aplicação:** grafo cujos nós representam cidades e os arcos as estradas que ligam as cidades.

**Problema:** Verificar a existência de um caminho de comprimento nr, a partir de A até B.

**Solução:** procurar um nó C tal que exista o arco de A até C e exista um caminho de comprimento nr – 1 de C até B. Se essas condições são satisfeitas para qualquer nó C, então o caminho desejado existe. Caso contrário, o caminho não existe.

Algoritmo:

```
.
.
.
leia(n)      // número de cidades
enquanto "existir mais dados" faca
    leia(cidd1,cidd2) // crie n nós e denomine-os 1 a n
    inserir(cidd1, cidd2)
fim-enquanto
leia(a,b)    // cidades para busca do caminho
leia(nr)      // comprimento do caminho para ir de A até B
se busca(nr, a, b) entao
    escreva("existe caminho de ", a, " até ", b, " de comprimento ", nr)
senao
    escreva("não existe caminho de", a, " até ", b, "de comprimento", nr)
fim-programa

funcao busca(k, a, b): logico
    se k = 1 entao // procura um caminho de comprimento 1
        se adjacente(a,b) entao busca := V
            senao busca := F
    senao inicio    // determina se existe um caminho através de C
        busca := F
        para c := 1 ate n faca
            se adjacente(a,c)
                entao se busca(k-1, c, b) entao busca := V
        fim
```

### Observação:

O algoritmo apresenta várias deficiências:

- ❑ muitos caminhos são investigados várias vezes durante o processo recursivo;
- ❑ resultado final somente aponta se um caminho existe ou não , sem indicar o caminho propriamente dito;
- ❑ não testa a existência de uma caminho sem considerar o comprimento; somente busca caminhos de tamanhos específicos.

## Representação de grafos

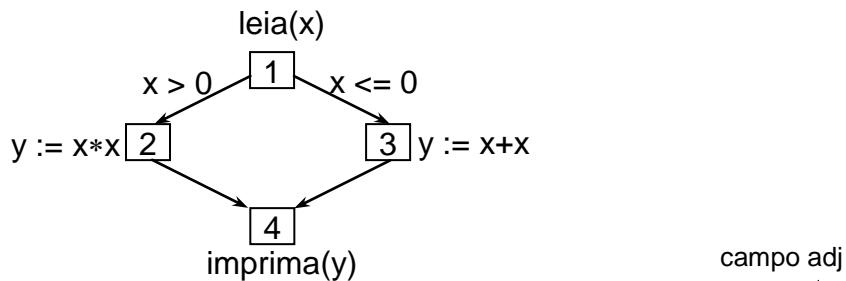
- Várias formas para a representação interna de um grafo

Supondo que o número de nós é constante podendo apenas ser variado o número de arcos do grafo, tem-se:

```
#define MAX 50
struct arco {
    int adj; // valor lógico
    // outras informações associadas com cada arco
};
struct no {
    // informações associadas com cada nó
};
struct grafo {
    struct no nos[MAX];
    struct arco arcos[MAX,MAX];
};
struct grafo g;
```

- Cada nó do grafo é representado por um número inteiro entre 1 e MAX
- O arranjo NOS contém as informações associadas a cada um dos nós
- O arranjo ARCOS é bi-dimensional e contém toda possível ligação de um nó a outro, além das informações associadas a esse arco, se ele existir
- O valor de `g.arcos[i, j].adj` é um valor lógico V ou F:
  - (V) → verdadeiro se existe o arco  $< i, j >$
  - (F) → falso, caso contrário

**Exemplo:** seja o dígrafo representando o fluxo de um programa



NOS		ARCOS						
1	leia(x)	F		V	x > 0	V	x <= 0	F
2	y := x*x	F		F		F		V *
3	y := x+x	F		F		F		V *
4	imprima(y)	F		F		F		F
		1		2		3		4

```

struct arco {
    int adj; // valor lógico
    char predicado[30];
};
struct no {
    char instrucao[30];
};
struct grafo {
    struct no nos[MAX];
    struct arco arcos[MAX, MAX];
};
struct grafo g;

```

### **Observações:**

- O arranjo bi-dimensional ARCOS é chamado de matriz de adjacências
- As informações associadas tanto aos nós como aos arcos são chamadas de pesos. O grafo ou dígrafo nessas condições é chamado de grafo ponderado ou dígrafo ponderado

Muitas aplicações não possuem pesos nos arcos e nos nós. Nesse caso o grafo pode ser representado unicamente por sua matriz de adjacência.

```
#define MAX 50
int mat_adj[MAX,MAX]; // valor lógico
```

Pode ser utilizada uma representação que use parte da primeira definição. Isto é, os nós possuírem peso, mas os arcos não e vice-versa.

**Exercício:** construir o algoritmo das operações INSERIR, ELIMINAR e ADJACENTE, utilizando a representação para grafos não ponderados (sem pesos).

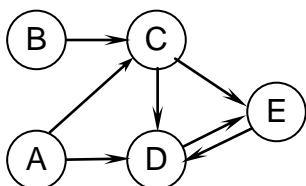
### **Matriz de adjacência**

Um dígrafo com n nós:  $a_1, a_2, a_3, \dots, a_n$  pode ser representado por uma matriz.

**Definição:** seja  $D = < A, R >$  um dígrafo com  $A = \{ a_1, a_2, a_3, \dots, a_n \}$ . A matriz de adjacência  $M$  de  $D$  é definida por:

$$M_{i,j} = \begin{cases} 1; & \text{se } < a_i, a_j > \in R \\ 0; & \text{se } < a_i, a_j > \notin R \end{cases}$$

### **Exemplo:**



$$M = \begin{array}{c|ccccc} & A & B & C & D & E \\ \hline A & 0 & 0 & 1 & 1 & 0 \\ B & 0 & 0 & 1 & 0 & 0 \\ C & 0 & 0 & 0 & 1 & 1 \\ D & 0 & 0 & 0 & 0 & 1 \\ E & 0 & 0 & 0 & 1 & 0 \end{array}$$

**Definição:** seja  $D = \langle A, R \rangle$  um dígrafo com  $A = \{a_1, a_2, a_3, \dots, a_n\}$ . A matriz de caminhos  $P$  de  $D$  é definida por:

$$P_{i,j} = \begin{cases} 1; & \text{se existe um caminho, não nulo, de } i \text{ até } j \\ 0; & \text{se não existe caminho de } i \text{ até } j \end{cases}$$

**Exemplo:** de acordo com o dígrafo anterior, tem-se:

$$P = \begin{pmatrix} 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 1 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix}$$

## Caminho num dígrafo

Os elementos da matriz de adjacência foram definidos como lógicos deliberadamente, a fim de permitir que operações fossem realizadas.

Vamos assumir que um dígrafo é completamente descrito por sua matriz de adjacência  $M$ . O valor da expressão  $M[i,k] \wedge M[k,j]$  será verdadeiro se, e somente se, existir um arco do nó  $i$  para  $k$  e do nó  $k$  para  $j$ . Portanto,  $M[i,k] \wedge M[k,j]$  é verdade se existe um caminho de tamanho 2 de  $i$  para  $j$  passando por  $k$ .

Considerando a expressão:

$$(M[i,1] \wedge M[1,j]) \vee (M[i,2] \wedge M[2,j]) \vee \dots \vee (M[i,MAX] \wedge M[MAX,j])$$

temos que o valor será verdadeiro somente se existir um caminho de tamanho 2 de  $i$  para  $j$ .

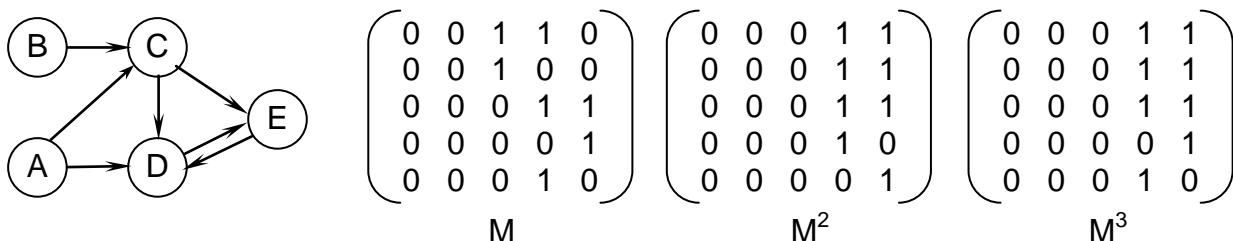
Considere um arranjo  $M^2$  tal que  $M^2[i,j]$  é o valor da expressão anterior.  $M^2$  é chamado matriz de caminhos de comprimento 2 e indica se existe um caminho de tamanho 2 entre  $i$  e  $j$ .

$$M^2 = M \times M, \text{ sendo } \text{multiplicação} = E \text{ e } \text{adição} = OU \text{ (produto booleano).}$$

Analogamente,  $M^3$  ( $M^2 \times M$ ) indica os caminhos de comprimento 3,  $M^4$  de comprimento 4 e assim por diante.

**Observação:** para um dígrafo de  $n$  nós, o caminho mais longo tem comprimento máximo de  $n-1$  nós (sem repetição).

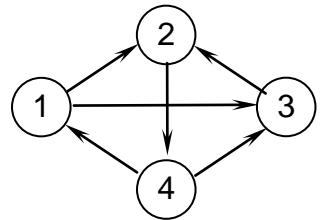
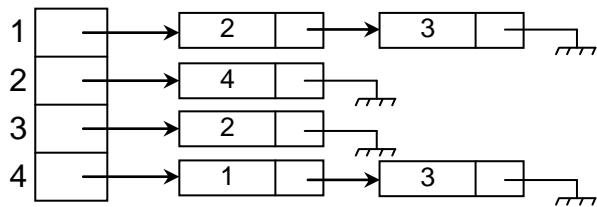
**Exemplo:** para o dígrafo abaixo, temos:



## Representação com listas ligadas

Seja o dígrafo ao lado:

- Uma possível representação utilizando vetor e listas ligadas (chamado listas de adjacências) seria:



Matriz de adjacência apresenta deficiências:

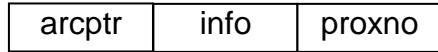
- requer o prévio conhecimento do número de nós
- grafo com  $n$  nós requer  $n^2$  posições (mesmo esparso)

**Solução:** alocação dinâmica.

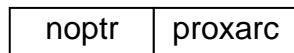
**Dificuldade:** prévio conhecimento do número máximo de arcos (necessário para a definição da estrutura).

**Alternativa:** estrutura multiligada:

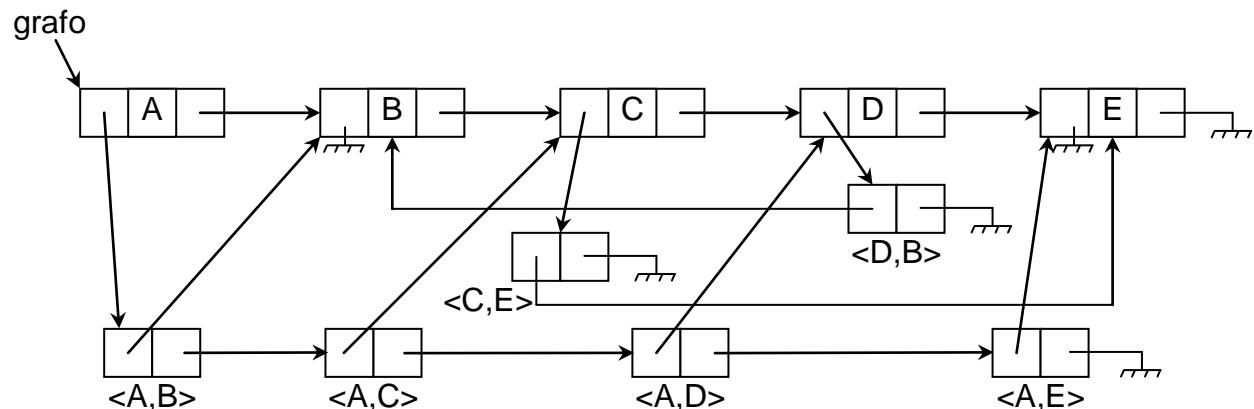
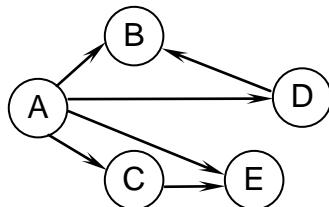
- os nós são representados por uma lista ligada de “nós de cabeçalho”:



- Lista de adjacências representa os arcos (nós de lista):



Exemplo:



**Observação:** grafos com pesos – necessário acrescentar campos aos nós.

Assumindo que ambas as listas possuem o mesmo formato e contém dois ponteiros e um campo de informação do tipo inteiro, temos a seguinte declaração utilizando arranjos:

```
#define max 50
struct no {
    int info, point, prox;
};
struct no nos[MAX];
```

No caso de um nó de cabeçalho, *nos[p]* representa um nó; *nos[p].info* representa a informação associada ao nó; *nos[p].prox* aponta para o próximo nó do dígrafo; e *nos[p].point* aponta para a lista de adjacência representando os arcos que saem do nó.

No caso da lista de adjacência (nó de lista), *nos[p]* representa um arco  $\langle A, B \rangle$ , *nos[p].info* representa o peso do arco; *nos[p].prox* aponta para o próximo arco que sai de A; e *nos[p].point* aponta para um nó de cabeçalho.

Implementação utilizando ponteiros:

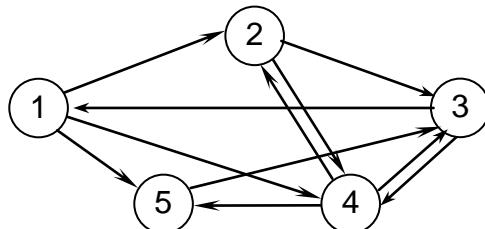
```
struct no {
    int info;
    struct no *point, *prox;
};
struct no *nos;
```

## Árvores estendidas

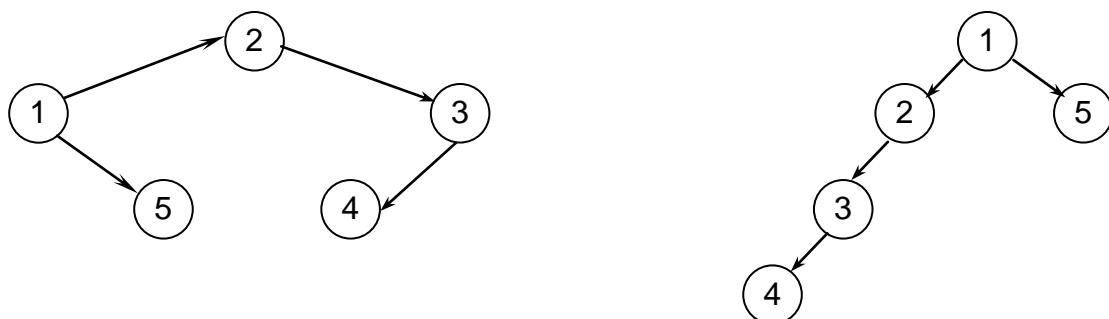
Seja  $G = \langle A, R \rangle$  um grafo.

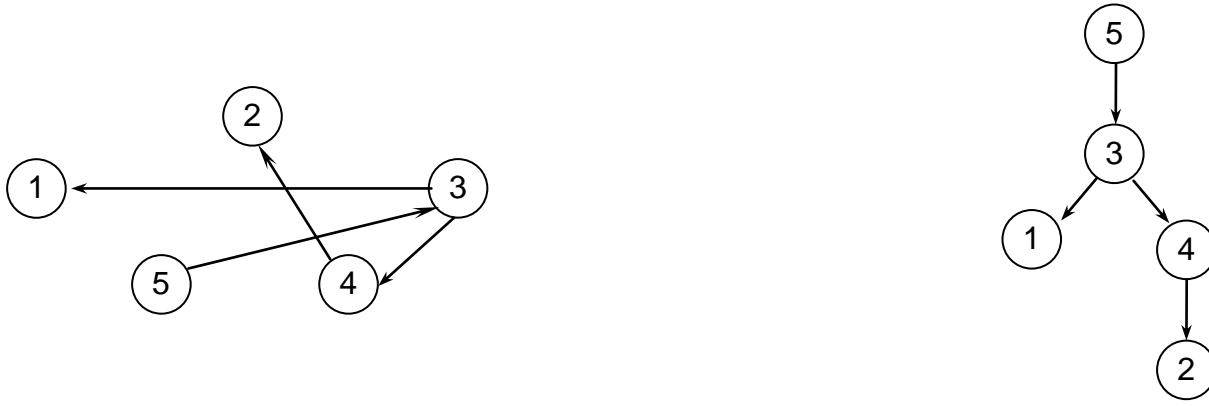
Árvore estendida é uma árvore que consiste unicamente de arcos de  $G$  e que inclui todos os nós de  $G$ .

**Exemplo:** seja o dígrafo



Árvores estendidas:

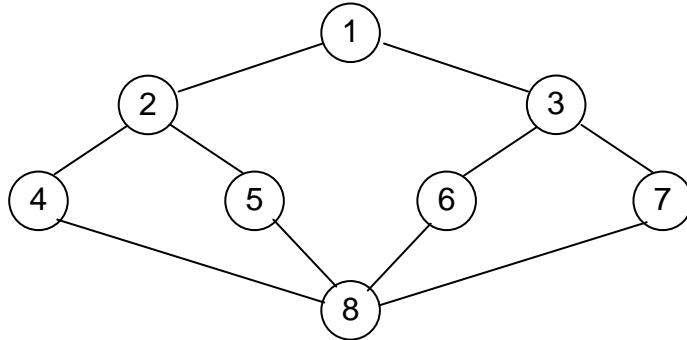




## Pesquisa em profundidade e em largura

- Formas de percorrer um grafo, gerando árvores estendidas
- Forma sistemática de acessar todos os nós de um grafo
- Mais complexo do que percorrer árvores e listas pelas seguintes razões:
  1. em geral não existe um “primeiro” nó para iniciar o percurso. Além disso, após a escolha de um nó inicial e iniciado o percurso, podem existir nós não visitados por não serem alcançáveis;
  2. não existe uma ordem natural entre os sucessores de um determinado nó;
  3. um nó pode ter mais do que um predecessor.
- Pesquisa em profundidade → pilha
- Pesquisa em largura → fila

Seja o grafo:



Objetivo: visitar todos os nós.

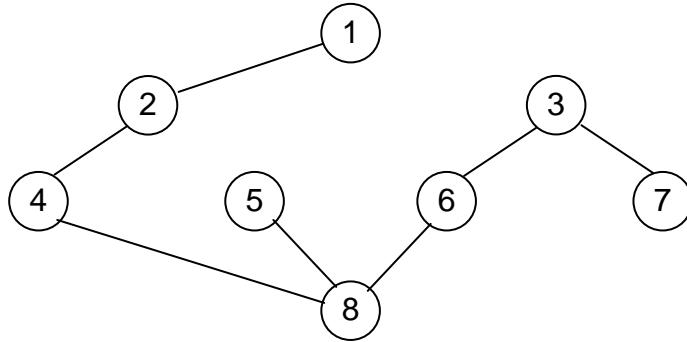
### 1) Em profundidade: DFS (*depth first search*)

- tendo um grafo não-dirigido  $G$  com  $n$  nós e um arranjo  $\text{visite}(n)$  inicialmente zerado, o algoritmo abaixo visita todos os nós alcançáveis a partir de  $v$ ;
- $\text{visite}$  é o vetor que conterá as posições visitadas. Tamanho = número de nós:

```

procedimento DFS(v)      { v é o nó de partida }
inicio
  visite(v) := 1
  para cada nó w adjacente à v faça
    se visite(w) = 0 então DFS(w)
fim
  
```

Árvore estendida gerada:

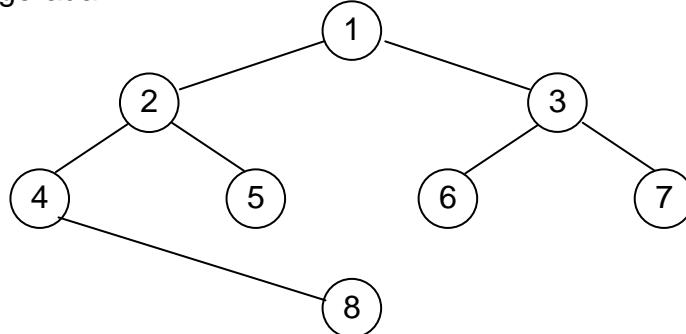


### 1) Em largura: BFS (*breadth first search*)

```

procedimento BFS(v)      { v é o nó de partida }
inicio
    visite(v) := 1
    CRIAR(Q,I,F)      { cria uma fila Q vazia }
    repita
        para cada nó w adjacente à v faça
            se visite(w) = 0 então INSERIR(Q,I,F,w) E visite(w) := 1
            se I = F então flag := 0      { testa se a fila está vazia }
                senão ELIMINAR (Q,I,F,v)
        até flag = 0
fim
  
```

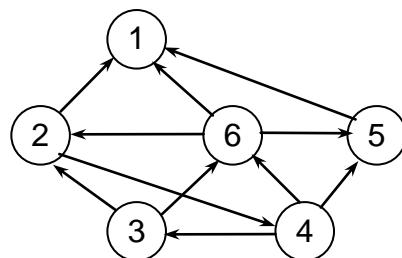
Árvore estendida gerada:



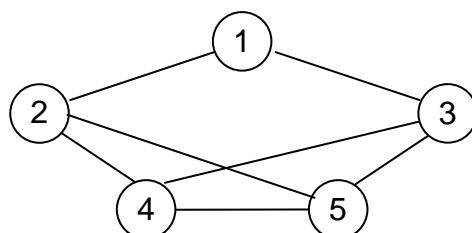
### Exercícios:

1. Para o dígrafo ao lado obtenha:

- i) o grau de entrada e saída de cada nó;
- ii) a matriz de adjacência;
- iii) a representação em uma estrutura multiligada.



2. Aplique a pesquisa em profundidade e a pesquisa em largura para o seguinte grafo (relacione os nós segundo a ordem em que seriam visitados):



## Um algoritmo de menor caminho

Em um grafo com pesos, ou rede, deseja-se freqüentemente achar o menor caminho entre dois nós “s” e “t”, de modo que a soma dos pesos dos arcos do caminho seja minimizada. A seguir é apresentado um algoritmo que soluciona tal problema:

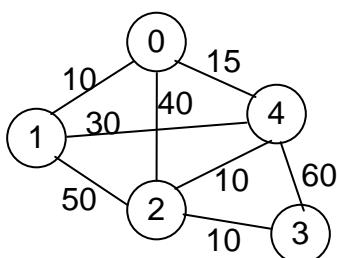
```

Algoritmo grafo
const infinito = ... {valor atribuído quando adjacente(A,B) for falso}
    max = ... {número máximo de nós}
    membro = 1
    naomembro = 0
tipo matriz = arranjo [max,max] de inteiro
    vetor = arranjo [max] de inteiro

procedimento menor_caminho(peso: matriz; var precede: vetor; s,t: inteiro; var
                            total: inteiro)
var
    distancia, perm: vetor
    atual, i, k, da, menordist, novadist: inteiro
inicio
    para i:=0 até max-1 faça
        perm[i] := naomembro
        distancia[i] := infinito
    fim_para
    perm[s] := membro
    distancia[s] := 0
    atual := s
    enquanto atual != t faça
        inicio
            menordist := infinito
            da := distancia[atual]
            para i:=0 até max-1 faça
                se perm[i] = naomembro então
                    novadist := da + peso[atual,i]
                    se novadist < distancia[i] então
                        distancia[i] := novadist
                        precede[i] := atual
                    fim_se
                    se distancia[i] < menordist então
                        menordist := distancia[i]
                        k := i
                    fim_se
                fim_se
            atual := k
            perm[atual] := membro
        fim
    total := distancia[t]
fim

```

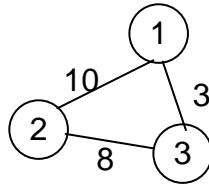
**Exemplo:** dado o grafo abaixo, e considerando s=1, t=3 e l=Infinito, temos:



	0	1	2	3	4
0	l	10	40	l	15
1	10	l	50	l	30
2	40	50	l	10	10
3	l	l	10	l	60
4	15	30	10	60	l

### Exercício:

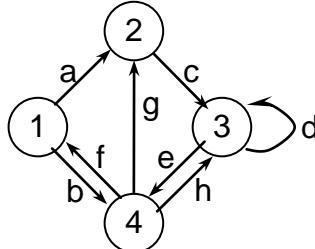
Identifique a função de cada uma das variáveis existentes no algoritmo anterior.  
Siga o algoritmo utilizando o seguinte grafo:



### Enumeração de todos os caminhos

Problema: relacionar todos os possíveis caminhos simples (caminho em que são diferentes todos os nós com a possível exceção do primeiro e do último) entre dois nós, por ordem de comprimento de caminho.

Seja o dígrafo:



Representação do dígrafo anterior:

$$V_0 = \begin{pmatrix} \wedge & a & \wedge & b \\ \wedge & \wedge & c & \wedge \\ \wedge & \wedge & d & e \\ f & g & h & \wedge \end{pmatrix}$$

Definir:

$$V_1: (v_{ij})_1 = v_{ij} \text{ para } i \neq j \quad e \quad (v_{ij}) = \wedge \text{ para } i = j$$

$$D_1: (d_i) = v_{ii}$$

$$V_L = V_0 \cdot V_k \quad e \quad V_R = V_k \cdot V_0 \text{ para } k = 1 \text{ até } n$$

$$V_{k+1}: (v_{ij})_{k+1} = \wedge \text{ para } i = j \text{ e}$$

$$(v_{ij})_{k+1} = \text{a interseção dos termos que ocorrem tanto em } (v_{ij})_L \text{ como em } (v_{ij})_R$$

assim,

$$V_1 = \begin{pmatrix} \wedge & a & \wedge & b \\ \wedge & \wedge & c & \wedge \\ \wedge & \wedge & \wedge & e \\ f & g & h & \wedge \end{pmatrix} \quad D_1 = \begin{pmatrix} \wedge \\ \wedge \\ d \\ \wedge \end{pmatrix}$$

$$V_L = V_0 \cdot V_1 = \begin{pmatrix} bf & bg & ac \wedge bh & \wedge \\ \wedge & \wedge & \wedge & ce \\ ef & eg & eh & de \\ \wedge & fa & gc & fb \wedge he \end{pmatrix} \quad V_R = V_1 \cdot V_0 = \begin{pmatrix} bf & bg & ac \wedge bh & \wedge \\ \wedge & \wedge & cd & ce \\ ef & eg & eh & \wedge \\ \wedge & fa & gc & fb \wedge he \end{pmatrix}$$

$$V_2 = \begin{pmatrix} \wedge & bg & ac \wedge bh & \wedge \\ \wedge & \wedge & \wedge & ce \\ ef & eg & \wedge & \wedge \\ \wedge & fa & gc & \wedge \end{pmatrix}$$

$$D_2 = \begin{pmatrix} bf \\ \wedge \\ eh \\ fb \wedge he \end{pmatrix}$$

$$V_L = V_0 \cdot V_2 = \begin{pmatrix} \wedge & bfa & bgc & ace \\ cef & ceg & \wedge & \wedge \\ def & deg \wedge efa & egc & \wedge \\ hef & fbg \wedge heg & fac \wedge fbh & gce \end{pmatrix}$$

$$V_R = V_2 \cdot V_0 = \begin{pmatrix} \wedge & \wedge & bgc \wedge acd \wedge bhd & ace \wedge bhe \\ cef & ceg & ceh & \wedge \\ \wedge & efa & egc & efb \\ \wedge & \wedge & fac \wedge gcd & gce \end{pmatrix}$$

$$V_3 = \begin{pmatrix} \wedge & \wedge & bgc & ace \\ cef & \wedge & \wedge & \wedge \\ \wedge & efa & \wedge & \wedge \\ \wedge & \wedge & fac & \wedge \end{pmatrix}$$

$$D_3 = \begin{pmatrix} \wedge \\ ceg \\ egc \\ gce \end{pmatrix}$$

### Observações:

$V_3$ : conjunto de todos os caminhos simples de comprimento 3.

$D_3$ : conjunto de todos os ciclos simples de comprimento 3.

$$V_L = V_0 \cdot V_3 = \begin{pmatrix} acef & \wedge & bfac & \wedge \\ \wedge & cefa & \wedge & \wedge \\ \wedge & \wedge & efac & \wedge \\ gcef & hefa & fbgc & face \end{pmatrix}$$

$$V_R = V_3 \cdot V_0 = \begin{pmatrix} acef & aceg & aceh & bgce \\ \wedge & cefa & \wedge & cefb \\ \wedge & \wedge & efac & \wedge \\ \wedge & \wedge & \wedge & face \end{pmatrix}$$

$$V_4 = \begin{pmatrix} \wedge & \wedge & \wedge & \wedge \\ \wedge & \wedge & \wedge & \wedge \\ \wedge & \wedge & \wedge & \wedge \\ \wedge & \wedge & \wedge & \wedge \end{pmatrix}$$

$$D_4 = \begin{pmatrix} acef \\ cefa \\ efac \\ face \end{pmatrix}$$

### Bibliografia

HOROWITZ, E.; SAHNI, S. Fundamentos de Estrutura de Dados, Rio de Janeiro, Campus Ed., 1986.

SZWARCFITER, J.L. Grafos e Algoritmos Computacionais. Editora Campus, 1986.

TENENBAUM, A. M.; LANGSAN, Y.; AU-GENSTEIN, M. Estrutura de Dados usando C. São Paulo: Makron Books Ed., 1995.