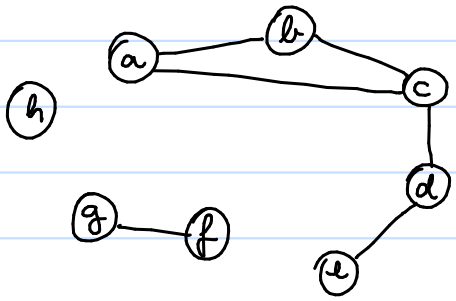
# Grafos

DEFINIÇÃO: Um grafo $G$ é um par de conjuntos $(V,E)$ onde $V$ é um conj. de elementos chamados vértices e $E$ é um conj. de elem. chamados de arestas, sendo que cada aresta é um par de vértices

→ Exemplo: $H = (V,E)$ com $V = \{a, b, c, d, e, f, g, h\}$
e $E = \{\{a,b\}, \{a,c\}, \{b,c\}, \{c,d\}, \{d,e\}, \{f,g\}\}$




Obs: essa definição é de grafos simples
Obs 2: $E = \{ab, ac, bc, cd, de, fg\}$.
Obs 3: Usaremos $V(\cdot)$ e $E(\cdot)$

## Terminologias básicas em grafos

→ Se $e = uv$ é uma aresta de um grafo $G$, então:
↳ $u$ e $v$ são vizinhos / adjacentes
↳ $u$ e $v$ são extremos de $e$
↳ $e$ incide em $u$ e em $v$

→ Dado $u \in V(G)$:
↳ o grau de $u$ é o nº de arestas incidentes a $u$ ($d_G(u)$)
↳ a vizinhança de $u$ é o conj. de vizinhos de $u$ ($N_G(u)$)

→ Dado um grafo $G$:
↳ o grau mínimo é $\delta(G) = \min\{d(v) : v \in V(G)\}$
↳ o grau máximo é $\Delta(G) = \max\{d(v) : v \in V(G)\}$

## O resultado mais básico

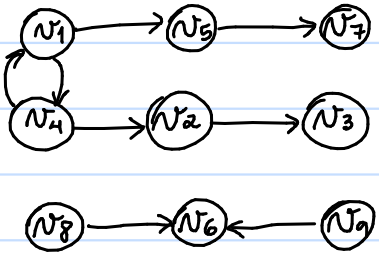
TEOREMA DO APERTO DE MÃOS: Para todo grafo $G$ vale que

$$\sum_{v \in V(G)} d(v) = 2|E(G)|.$$

# Dígrafos

DEFINIÇÃO: Um dígrafo $D$ é um par de conjuntos $(V,E)$ onde $V$ é um conj. de elementos chamados vértices e $E$ é um conj. de elem. chamados de arcos, sendo que cada arco é um par ordenado de vértices.

→ Exemplo: $H = (V,E)$ com $V = \{v_1, v_2, \dots, v_9\}$ e $E = \{(v_1,v_5), (v_2,v_3), (v_1,v_4), (v_4,v_2), (v_4,v_1), (v_5,v_7), (v_8,v_6), (v_9,v_6)\}$.




Obs: essa definição é de dígrafo simples

Obs 2: $(x,y) \neq (y,x)$. $E = \{v_1v_5, v_2v_3, \dots\}$

Obs 3: Usaremos $V(\cdot)$ e $E(\cdot)$

→ Todo grafo é um dígrafo (pensamos em $\{x,y\}$ como $(x,y)$ e $(y,x)$).

## Terminologias básicas em dígrafos

→ Se $e = uv$ é um arco de um dígrafo $D$, então:
- ↳ $u$ é cabeça e $v$ é cauda
- ↳ $e$ sai de $u$ e entra em $v$

→ Dado $u \in V(D)$:
- ↳ o grau de entrada de $u$ é o nº de arcos que entram em $u$ ($d_D^-(u)$) e o grau de saída é o nº de arcos que saem de $u$ ($d_D^+(u)$)
- ↳ a vizinhança de entrada de $u$ é o conj. de vértices extremos dos arcos que entram em $u$ ($N_D^-(u)$) e a vizinhança de saída é o conj. de vértices extremos dos arcos que saem de $u$, exceto por $u$ ($N_D^+(u)$).

TEOREMA DO APERTO DE MÃOS: Para todo dígrafo $D$ vale que

$$\sum_{v \in V(G)} d^+(v) = \sum_{v \in V(G)} d^-(v) = |E(G)|.$$

## Grafos e dígrafos ponderados

→ Podemos associar valores (pesos) a vértices e/ou arestas dos (di)grafos.
  ↳ $w : V(G) \to \mathcal{N}$
  ↳ $w : E(G) \to \mathcal{N}$

## Subgrafos

→ Dados grafos $H$ e $G$, $H$ é subgrafo de $G$, denotado $H \subseteq G$, se $V(H) \subseteq V(G)$ e $E(H) \subseteq E(G)$.

## Passeios

→ Uma sequência de vértices $W = (v_0, v_1, \dots, v_k)$ é um passeio de um grafo $G$ se $v_i \in V(G)$ para todo $0 \leq i \leq k$ e $v_i v_{i+1} \in E(G)$ para todo $0 \leq i \leq k-1$.
→ comprimento = nº de arestas
→ aberto: se $v_0 \neq v_k$
  fechado: se $v_0 = v_k$.
  $v_0$ e $v_k$ são extremos e $v_1, \dots, v_{k-1}$ são vértices internos.
→ caminho: passeio que não repete vértices.
  ciclo: passeio fechado que não repete vértices internos.
→ $uv$- caminho: caminho de $u$ até $v$

## Classes de grafos

→ Completos: grafos $K_n$ com $V(K_n) = \{v_1, \dots, v_n\}$ e $E(K_n) = \{xy : x, y \in V(K_n)\}$
  ↳ $|E(K_n)| = \binom{n}{2} = \frac{n(n-1)}{2}$
→ Fixado número $n$ de vértices, todo grafo com $n$ vértices é subgrafo de um grafo completo
  Para todo $G$, $|E(G)| \leq \frac{n(n-1)}{2}$. ★

## Conexidade em grafos

→ Um grafo $G$ é **conexo** se existe $uv$-caminho para todo $u, v \in V(G)$.
Caso contrário, $G$ é **desconexo** e consiste de várias **componentes conexas**, que são subgrafos conexos maximais.

## Conexidade em digrafos

→ Um digrafo $D$ é **fortemente conexo** se existe $uv$-caminho para todo $u, v \in V(G)$.
Caso contrário, $D$ consiste de várias **componentes fortemente conexas**, que são subdigrafos fortemente conexos maximais.

## Distância em grafos / digrafos

→ Dados um (di)grafo $G$ e dois vértices $u, v \in V(G)$:

$$dist_G(u,v) = \begin{cases} \text{comprimento de um } uv\text{-caminho de menor compr.} & \text{se há } uv\text{-cam.} \\ \infty & \text{c.c.} \end{cases}$$

## Distância em grafos / digrafos ponderados

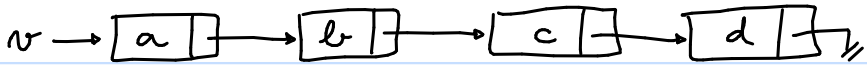
→ Dados um (di)grafo $G$, uma função $w: E(G) \to \mathbb{R}$ e dois vértices $u, v \in V(G)$:

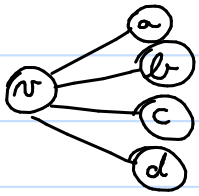
$$dist^w_G(u,v) = \begin{cases} \text{peso de um } uv\text{-caminho de menor peso} & \text{se há } uv\text{-cam.} \\ \infty & \text{c.c.} \end{cases}$$

# Representação de grafos

→ Assuma que o conjunto de vértices é $\{1, 2, \ldots, n\}$.

→ Na representação por **listas de adjacências** há uma lista ligada para cada vértice, contendo os vizinhos do vértice.



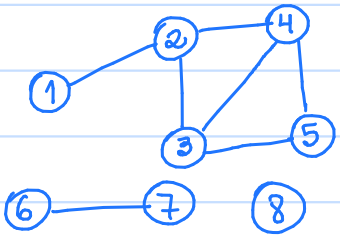



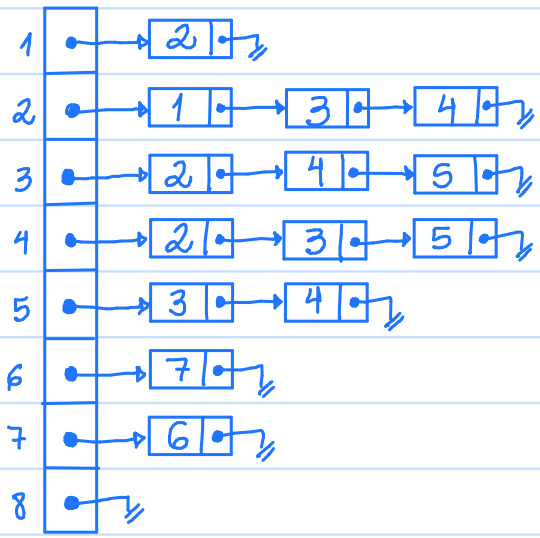

↳ Então uma lista tem $d(v)$ elementos

↳ O espaço total é $\Theta(|V(G)|) + \sum_{v \in V(G)} \Theta(d(v)) = \Theta(|V(G)| + |E(G)|)$

→ Na representação por **matriz de adjacências**, o grafo é dado por uma matriz $M$ com $|V(G)|$ linhas e $|V(G)|$ colunas, com $M[i][j] = 1$ se $ij \in E(G)$ e $M[i][j] = 0$ caso contrário.

↳ O espaço total é $\Theta(|V(G)|^2)$




| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 |
| 5 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 7 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 8 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

# Pseudocódigos e análise de algoritmos em grafos

→ Vamos resolver o problema de calcular o grau de um vértice.

```
GRAU (G, v)
    Devolve  d_G(v)
```

ou

```
GRAU (G, v)
    grau = 0
    Para cada u ∈ N_G(v)
        grau = grau + 1
    Devolve grau
```

"Realidade":

```
GRAU (M, n, v)
    grau = 0
    Para u = 1 até |V(G)|
        Se M[v][u] == 1
            grau = grau + 1
    Devolve grau
```

```
GRAU (L, n, v)
    grau = 0
    atual = L[v]
    Enquanto atual ≠ NULL
        grau = grau + 1
        atual = atual.prox
    Devolve grau
```

→ Vamos agora encontrar o grau máximo de um grafo ($\Delta(G)$)

```
GRAU_MAXIMO (G)
    max = 0
    Para cada v ∈ V(G)
        Se d_G(v) > max
            max = d_G(v)
    Devolve max
```

API de grafos:

```
typedef struct grafo grafo_t;

grafo_t* cria_grafo(int n, int m);
void adiciona_aresta(grafo_t *G, int u, int v);
void imprime_grafo(grafo_t *G);
void deleta_grafo(grafo_t *G);

int grau_vertice(grafo_t *G, int u);
```

Exemplo de função principal

```
int main(int argc, char *argv[]) {
    int n, m;
    int u, v;
    grafo_t *G;

    /* Esperado: dois numeros indicando qtd vertices e de arestas */
    scanf("%d %d", &n, &m);
    G = cria_grafo(n, m);

    /* Esperado: m arestas dadas por dois inteiros cada: u, v, custo */
    while (m--) {
        scanf("%d %d", &u, &v);
        adiciona_aresta(G, u, v);
    }

    imprime_grafo(G);

    deleta_grafo(G);

    return 0;
}
```

Exemplos de entradas:

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

```
12 16
10 11
9 10
9 11
8 11
8 9
5 7
4 7
3 6
2 5
2 4
1 4
2 3
1 2
0 3
0 2
0 1
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "grafo.h"

struct grafo {
    int n;
    int m;
    int **matriz_adj;
};

grafo_t* cria_grafo(int n, int m) {
    grafo_t* G = malloc(sizeof(grafo_t));
    G->n = n;
    G->m = m;
    G->matriz_adj = malloc(n * sizeof(int*));

    for (int u = 0; u < n; u++) {
        G->matriz_adj[u] = malloc(n * sizeof(int));
        for (int v = 0; v < n; v++)
            G->matriz_adj[u][v] = 0;
    }

    return G;
}
```

```c
void adiciona_aresta(grafo_t *G, int u, int v, int custo) {
    /* Vertice u eh vizinho do v */
    G->matriz_adj[v][u] = 1;

    /* Vertice v eh vizinho do u */
    G->matriz_adj[u][v] = 1;
}

void imprime_grafo(grafo_t *G) {
    /* Para cada vertice u, imprime os vizinhos dele */
    for (int u = 0; u < G->n; u++) {
        printf("%d:", u);
        for (int v = 0; v < G->n; v++) {
            if (G->matriz_adj[u][v])
                printf(" %d", v);
        }
        printf("\n");
    }
}
```

```c
int grau_vertice(grafo_t *G, int u) {
    int grau = 0;

    /* Testa se algum outro vertice eh vizinho de u */
    for (int v = 0; v < G->n; v++) {
        if (G->matriz_adj[u][v]) {
            grau++;
        }
    }

    return grau;
}
```

```c
#include <stdio.h>
#include <stdlib.h>
#include "grafo.h"

struct node {
    int rotulo;
    struct node *prox;
};

struct grafo {
    int n;
    int m;
    node_t **lista_adj;
};

grafo_t* cria_grafo(int n, int m) {
    grafo_t* G = malloc(sizeof(grafo_t));
    G->n = n;
    G->m = m;
    G->lista_adj = malloc(n * sizeof(node_t*));

    for (int u = 0; u < n; u++)
        G->lista_adj[u] = NULL;

    return G;
}
```

```c
void adiciona_aresta(grafo_t *G, int u, int v) {
    node_t *x;

    /* Vertice u eh vizinho do v: adiciona node com rotulo u na lista de v */
    x = malloc(sizeof(node_t));
    x->rotulo = u;
    x->prox = G->lista_adj[v];
    G->lista_adj[v] = x;

    /* Vertice v eh vizinho do u: adiciona node com rotulo v na lista de u */
    x = malloc(sizeof(node_t));
    x->rotulo = v;
    x->prox = G->lista_adj[u];
    G->lista_adj[u] = x;
}

void imprime_grafo(grafo_t *G) {
    node_t *x;

    /* Para cada vertice u, imprime os vertices vizinhos dele */
    for (int u = 0; u < G->n; u++) {
        printf("%d:", u);
        x = G->lista_adj[u];
        while (x) {
            printf(" %d", x->rotulo);
            x = x->prox;
        }
        printf("\n");
    }
}
```

```c
int grau_vertice(grafo_t *G, int u) {
    node_t *x;
    int grau = 0;

    /* Percorre a lista do vertice u */
    x = G->lista_adj[u];
    while (x) {
        grau++;
        x = x->prox;
    }

    return grau;
}
```

# Árvores

→ Se há $n$ componentes conexas com um vértice cada, qual o menor número de arestas a serem acrescentadas para ficar com uma única componente conexa?
  ↳ $n-1$

→ **Árvore**: grafo conexo e sem ciclos.
  ↳ **folha**: vértices de grau 1.
→ **floresta**: grafo que não contém ciclos.

→ Podemos enraizar uma árvore em qualquer um de seus vértices pois não há vértices especiais

**TEOREMA**: Seja $G$ um grafo. As seguintes afirmações são equivalentes:

1) $G$ é uma árvore
2) Existe um único caminho entre quaisquer dois vértices de $G$.
3) $G$ é conexo e para toda $e \in E(G)$, $G-e$ é desconexo.
4) $G$ é conexo e $|E(G)| = |V(G)| - 1$.
5) $G$ não tem ciclos e $|E(G)| = |V(G)| - 1$.
6) $G$ não tem ciclos e para todo par $x, y \in V(G)$ com $xy \notin E(G)$, $G+xy$ tem exatamente um ciclo.

## Consequências do teorema

→ Se $T$ é árvore e $xy \in E(T)$ com $x, y \in V(T)$, então $T + xy$ tem exatamente um ciclo.
Se $f \in E(T)$ pertence ao ciclo, então $T + xy - f$ é uma árvore.

→ Se $T \subseteq G$ é árvore e $V(T) = V(G)$, então $G$ é conexo.

→ Se $T \subseteq G$ é árvore, $V(T) \neq V(G)$ e existe $xy \in E(G)$ com $x \in V(T)$ e $y \in V(G) \setminus V(T)$, então $T + xy$ é árvore.

## Árvore geradora

→ $T \subseteq G$ é árvore geradora se $T$ é árvore e $V(T) = V(G)$.

# Busca em grafos

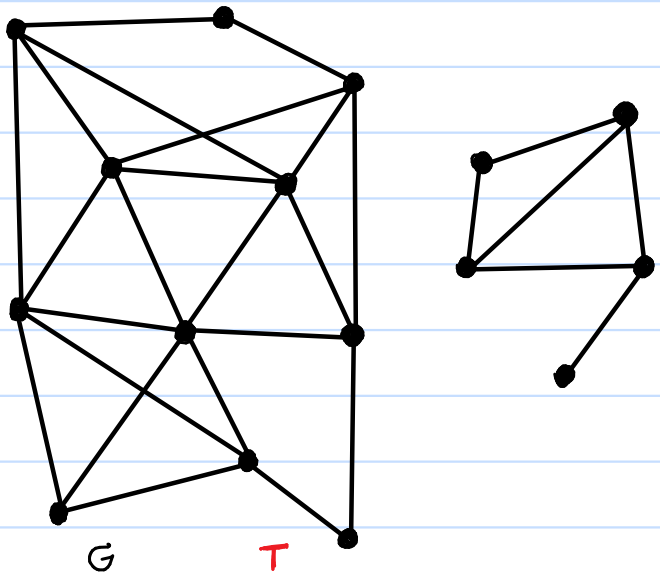
→ Nos dão informação sobre a estrutura de um grafo (caminhos)

→ Com poucos ajustes, resolvem outros problemas (conexão, ciclos, distância, bipartição, arestas de corte, geração de labirintos, componentes fortemente conexas, ...)

→ Inspiram a criação de outros algoritmos.

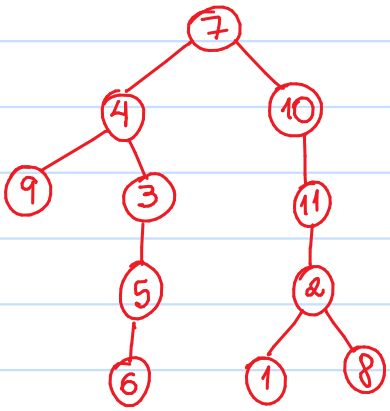
## Ideia dos algoritmos de busca em grafos

Crescer uma árvore $T \subseteq G$.
Se $V(T) = V(G)$, então $G$ é conexo.
Caso contrário, tentamos adicionar um vértice de $V(G) \setminus V(T)$ a $T$.




## Árvore de busca

→ Todo vértice tem um predecessor na árvore (v.predecessor)




$$E(T) = \{ \{v.predecessor, v\} : v \in V(T) \setminus \{s\} \}$$

→ v.visitado = 1 se e somente se v estiver na árvore

# Algoritmo genérico de busca (árvore implícita)

```
BUSCA(G, s)
1 s.visitado = 1
2 enquanto houver aresta com um extremo visitado e outro não faça
3 |   Seja xy uma aresta com x.visitado == 1 e y.visitado == 0
4 |   y.visitado = 1
5 |   y.pred = x
```

Invariante: no início da t-ésima iteração, v.visitado = 1 sse existe sv-caminho.

```
CHAMABUSCA(G)
1 para todo vértice v ∈ V(G) faça
2 |   v.visitado = 0
3 |   v.pred = null
4 seja s ∈ V(G) qualquer
5 BUSCA(G, s)
```

# Consequências do algoritmo de busca

Após a execução de Busca(G,s):

- v.visitado = 1 se e somente se há sv-caminho em G
- v.visitado = 1 se e somente se v e s estão na mesma componente conexa
- podemos construir um sv-caminho:

  (s, ...., v.predecessor, v)

  (s, ...., v.predecessor.predecessor, v.predecessor, v)

(outros sv-caminhos podem existir).

## Construindo um sv-caminho

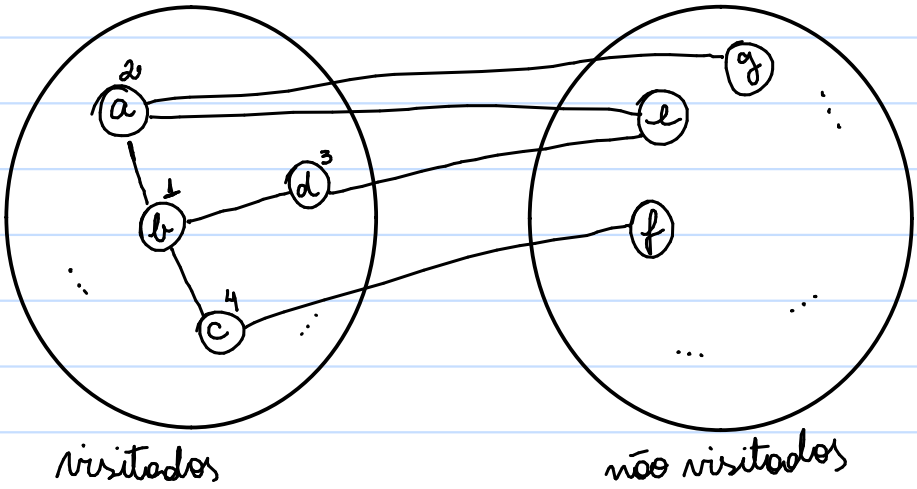
```
ConstroiCaminho(G, s, v)
1 seja L uma lista vazia
2 se v.visitado == 0 então
3     devolve L
4 atual = v
5 enquanto atual ≠ s faça
6     InsereNoInicioLista(L, atual)
7     atual = atual.pred
8 InsereNoInicioLista(L, s)
9 devolve L
```

## Como expandir a árvore?

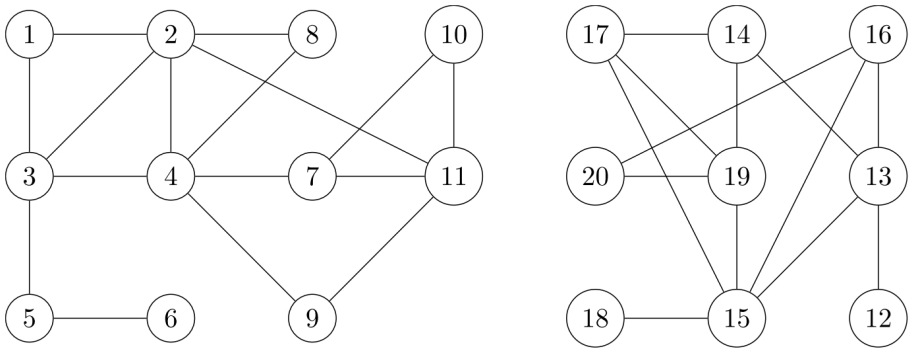
```
Busca(G, s)
1 s.visitado = 1
2 enquanto houver aresta com um extremo visitado e outro não faça
3     Seja xy uma aresta com x.visitado == 1 e y.visitado == 0
4     y.visitado = 1
5     y.pred = x
```

# Busca em largura – BFS (Breadth-first search)

→ Explora a vizinhança dos vértices já visitados na ordem "primeiro a entrar, primeiro a sair".




# Exemplo de execução da BFS (s = 7)




FILA : ( )

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| visitado | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| predecessor | | | | | | | | | | | | | | | | | | | | |
| dist | | | | | | | | | | | | | | | | | | | | |

# Algoritmo – BFS

$n_s$ = nº vértices na comp. de $s$
$m_s$ = nº arestas na comp. de $s$
$n = |V(G)|$ $\quad m = |E(G)|$
$V_s(G)$ = conj. vértices na comp. de $s$

```
BuscaLarguraDistancia(G, s)
1  s.visitado = 1
2  s.dist = 0
3  cria fila vazia F
4  Enfileira(F, s)
5  enquanto F.tamanho > 0 faça
6      u = Desenfileira(F)
7      para todo vértice v ∈ N(u) faça
8          se v.visitado == 0 então
9              v.visitado = 1
10             v.dist = u.dist + 1
11             v.pred = u
12             Enfileira(F, v)
```

Por fim, note que a árvore $T$ tal que

$$V(T) = \{v \in V(G) : v.\texttt{predecessor} \neq null\} \cup \{s\}$$
$$E(T) = \{\{v.\texttt{predecessor}, v\} : v \in V(T) \setminus \{s\}\}$$

é uma árvore geradora de $G$, contém um único $sv$-caminho para qualquer $v \in V(T)$ e é chamada de *árvore de busca em largura*.

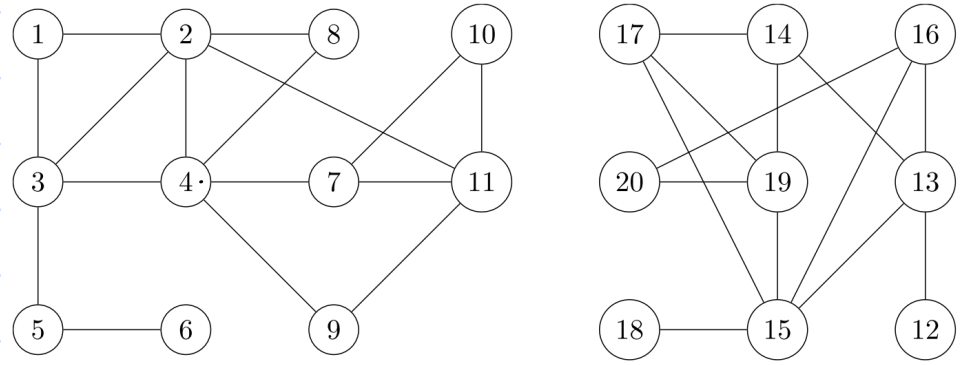
**LEMA:** Sejam $G$ um grafo e $s \in V(G)$. Na execução de BUSCALARGURADISTANCIA$(G, s)$, se $u$ e $v$ são dois vértices que estão na fila e $u$ entrou antes de $v$ na fila, então $\quad u.dist \leq v.dist \leq u.dist + 1$

**TEOREMA:** Sejam $G$ um grafo e $s \in V(G)$. Ao fim de BUSCALARGURADISTANCIA$(G, s)$, para todo $v \in V(G)$ vale que $\quad v.dist = dist_G(s, v)$.

# Busca em profundidade - DFS (Depth-first search)

→ Explora a vizinhança dos vértices já visitados na ordem "último a entrar, primeiro a sair".

## Exemplo de execução da DFS (s = 7)

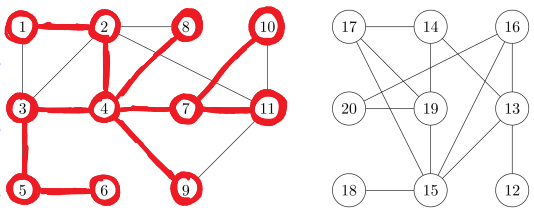


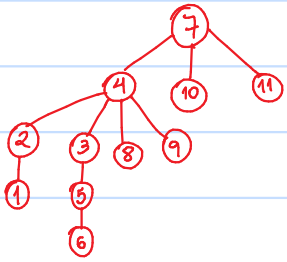

PILHA : ( )

| | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| visitado | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| predecessor | | | | | | | | | | | | | | | | | | | | |

```
BuscaProfRecursiva(G, s)
1 s.visitado = 1
2 para todo vértice v ∈ N(s) faça
3     se v.visitado == 0 então
4         v.pred = s
5         BuscaProfRecursiva(G, v)
```

## Resultado de BuscaLargura(G,7):




→ ciclos ←

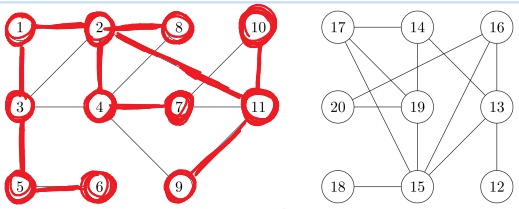
→ caminhos ←

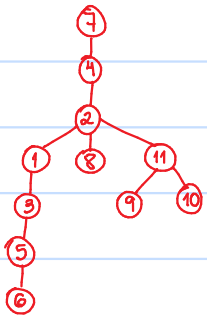
→ componentes ←

→ caminhos mínimos

→ grafos bipartidos

## Resultado de BuscaProfundidade(G,7):




→ arestas de corte

→ labirintos

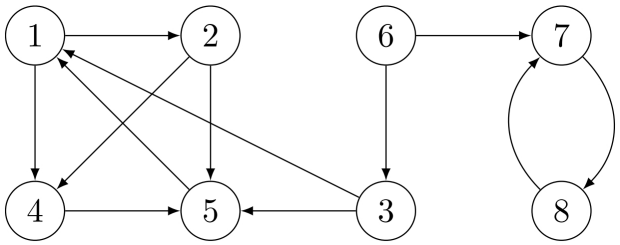
## Componentes conexas

**BUSCACOMPONENTES($G$)**

```
1  para todo vértice v ∈ V(G) faça
2      v.visitado = 0
3      v.pred = null
4  qtdComponentes = 0
5  para todo vértice s ∈ V(G) faça
6      se s.visitado == 0 então
7          s.visitado = 1
8          s.componente = s
9          qtdComponentes = qtdComponentes + 1
10         BUSCA(G, s)
11 devolve qtdComponentes
```

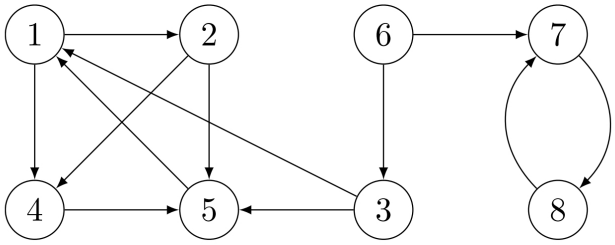
→ Atualizar nos buscas:
v.componente = u.componente
se v for visitado por causa de u

## Exemplo de execução — BFS e DFS em digrafos




BFS (D, 3)

FILA: ( )




DFS (D, 3)

PILHA: ( )

## "Problemas" das buscas em digrafos

→ Não obtemos sempre uma arborescência geradora, mesmo que uma exista.

→ Os vértices visitados não fazem parte de uma componente conexa do grafo subjacente e nem de uma componente fortemente conexa.

## Boas notícias

→ A busca em largura ainda encontra distâncias

→ A busca em profundidade, com poucas alterações, encontra componentes fortemente conexas.