TEG

Gilmário B. Santos

gilmario.santos@udesc.br

http://www.joinville.udesc.br/portal/pagina/gilmario

Percursos

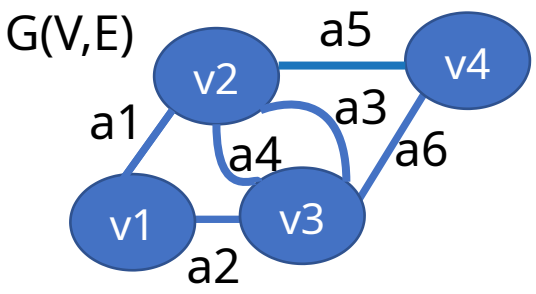
Passeio *P* em um grafo G(V,E):

Passeio: sequência de vértices $v_1, \ldots, v_k$,

tal que $(v_j, v_{j+1}) \in E$, $1 \leq j < |k|$

O comprimento de um passeio corresponde ao seu número de arestas.

Passeio aberto de *n* vértices → comprimento = n-1.



Exemplos de passeios em G:

P1: v1, a2, v3, a4, v2, a5, v4, a6, v3

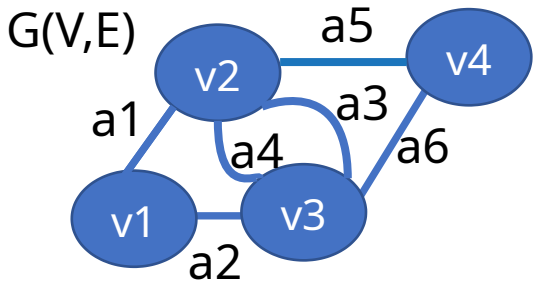
P2: v1, a1, v2, a3, v3, a4, v2, a1, v1

P3: v3, a6, v4, a5, v2, a1, v1

Percursos

Trilha *T* em um grafo:

Corresponde a um passeio sem a repetição de arestas.



Exemplos:

P1: v1, a2, v3, a4, v2, a5, v4, a6, v3, a3 v2

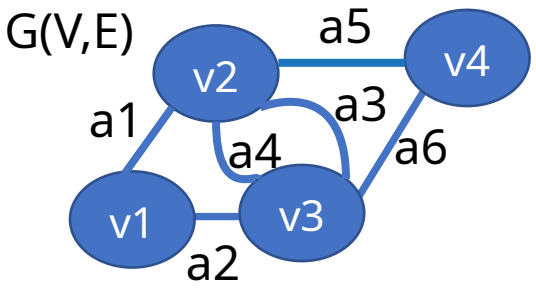
P2: v1, a1, v2, a3, v3, a4, v2, a1, v1

P2 não é uma trilha!

Percursos

Trilha fechada:

- Composta de vértices $v_1, \ldots, v_{k+1}$;
- $k \geq 3$ e
- $v_1 = v_{k+1}$



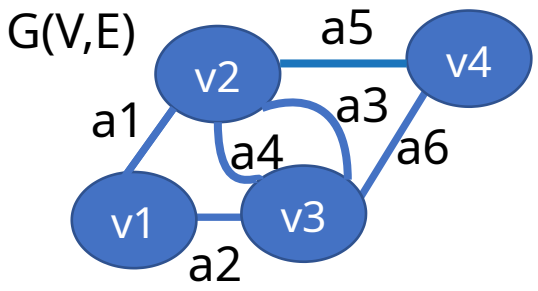
Exemplo de Trilha fechada:

v1, a2, v3, a4, v2, a5, v4, a6, v3, a3 v2, a1, v1

Percursos

Caminho *C* em um grafo:

Corresponde a um passeio sem a repetição de vértices.



Exemplos:

P1: v1, a2, v3, a4, v2, a5, v4, a6, v3

P2: v3, a6, v4, a5, v2, a1, v1

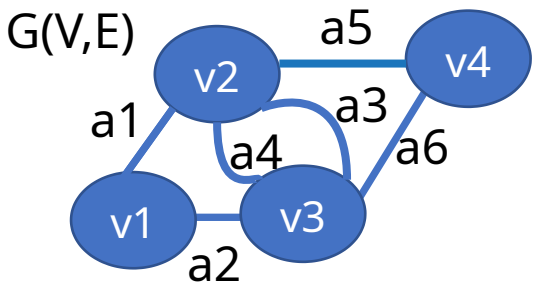
P1 não é um caminho

Percursos

Caminho fechado (ciclo):

- Composto de vértices $v_1, \ldots, v_{k+1}$;
- $k \geq 3$ e
- $v_1 = v_{k+1}$

Ciclo→ não há repetições de vértices (exceto nas extremidades do caminho).



P1: v3, a6, v4, a5, v2, a1, v1, a2, v3

P4: v1, a2, v3, a6, v4, a5, v2, a1, v1

P1 e P4 são ciclos

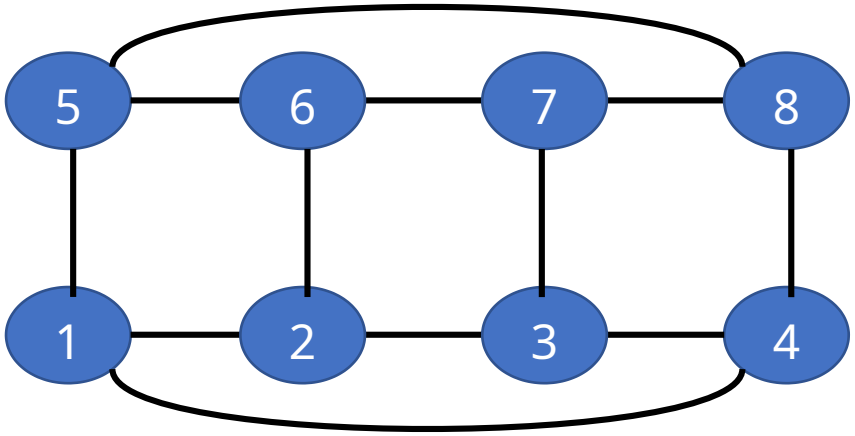
Percursos

Um grafo que não possui ciclos é acíclico.

Um triângulo é um ciclo de comprimento 3.

Dois ciclos são considerados idênticos se um deles puder ser obtido do outro, através de uma permutação circular de seus vértices.

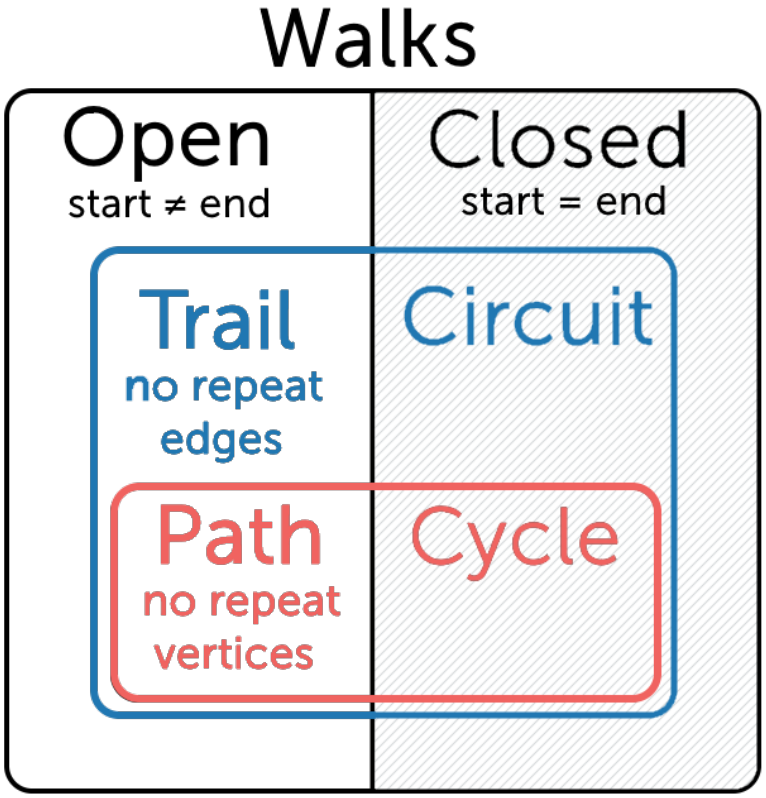
Por exemplo, no grafo da figura, os percursos *2, 3, 7, 6, 2* e *7, 6, 2, 3, 7* são ciclos idênticos,



Percursos - Resumo

1) Passeio: sequência de vértices e arestas;
2) Trilha: passeio sem repetição de arestas;
3) Caminho: passeio sem repetição de vértices;
4) Ciclo: caminho fechado;

Passeio: walk
Trilha: Trial
Caminho: Path



https://mathspace.co/textbooks/syllabuses/Syllabus-1030/topics/Topic-20297/subtopics/Subtopic-266706/
https://www.javatpoint.com/walks-trails-path-circuit-and-cycle-in-discrete-mathematics

Percursos - Resumo

1)Passeio: sequência de vértices e arestas;

2)Trilha: passeio sem repetição de arestas;

3)Caminho: passeio sem repetição de vértices, todos os vértices são distintos entre si;

Perceba que a parte “distinta” da definição de caminho proíbe a ocorrência de arestas repetidas, pois se uma aresta é repetida, um vértice é necessariamente repetido.

4)Ciclo: caminho fechado.

Percursos

Dois percursos particularmente especiais:

Trilha fechada Euleriana: contém todas as arestas do grafo sem repeti-las (é permitido repetir vértices);

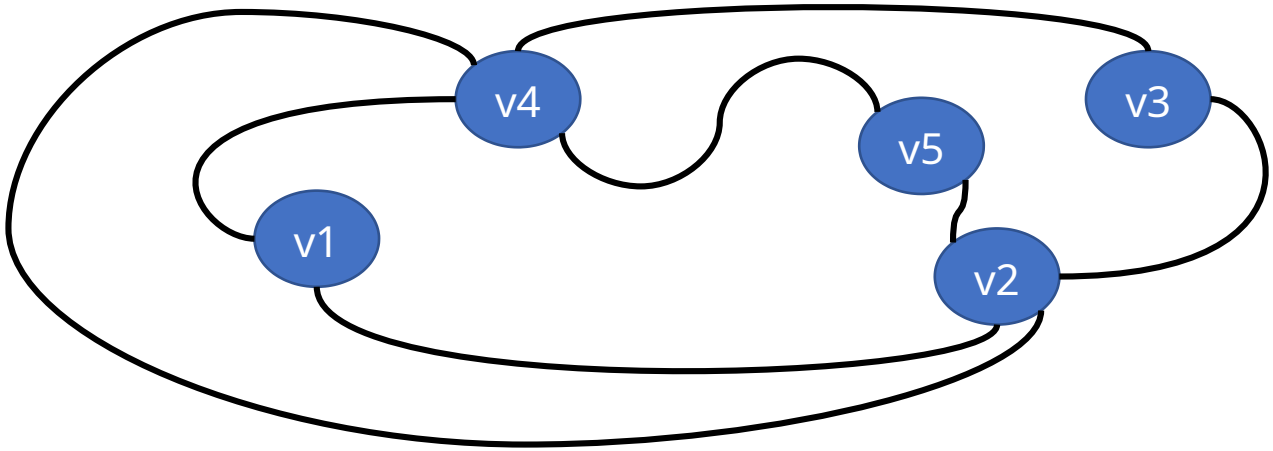
Ciclo Hamiltoniano: contém todos os vértices do grafo sem repetições, ou seja, todos os vértices e arestas são distintos entre si.

Percursos

Seja G(V,E) um grafo com uma trilha fechada contendo todas as arestas de G sem repetições, então G possui uma trilha fechada Euleriana e G é um grafo Euleriano.

Exemplo: a trilha {v4, v5, v2, v3, v4, v1, v2, v4} é uma trilha fechada euleriana, G é euleriano.

É permitido ocorrer repetições de vértices.

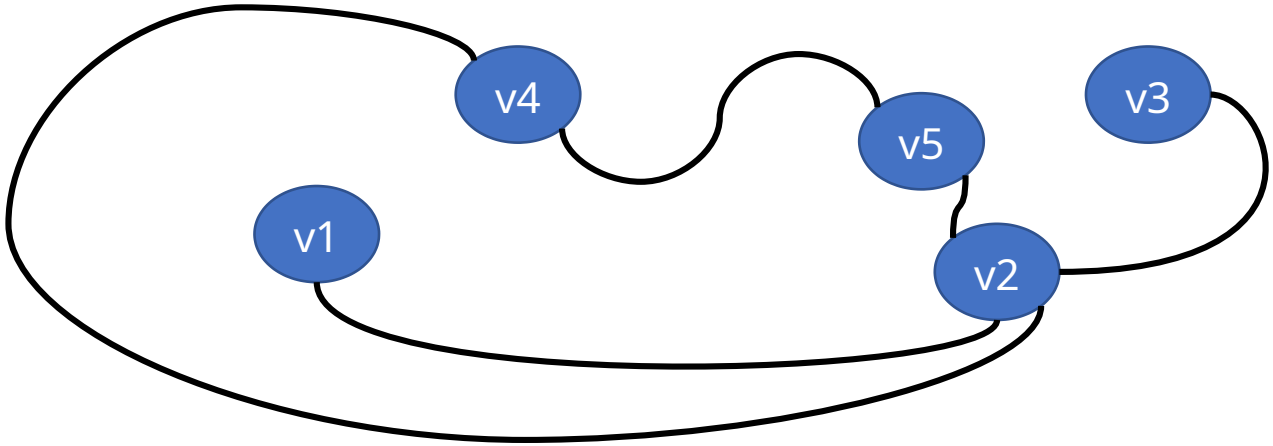


Percursos

Seja uma trilha contendo todas as arestas de G(V,E) sem repetições, então G possui apenas uma trilha Euleriana e será chamado de grafo semi-Euleriano

Exemplo: a trilha {v1, v2, v5, v4, v2, 34} é apenas uma trilha euleriana, G é semi-Euleriano.

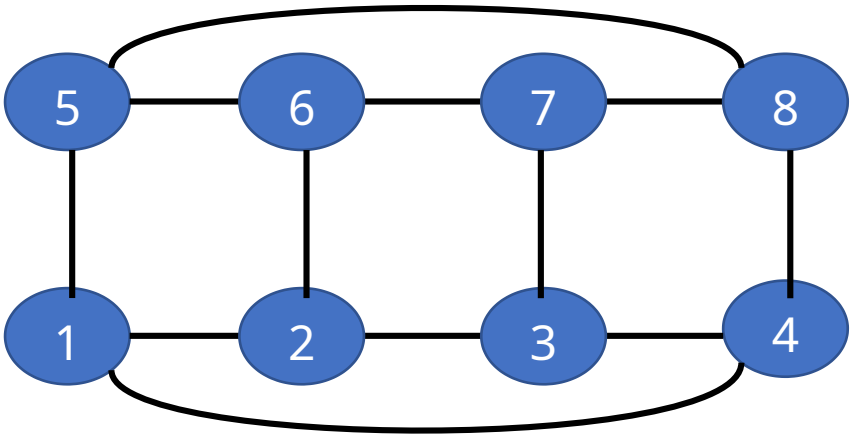
É permitido ocorrer repetições de vértices.



Percursos

Se G possui um caminho Hamiltoniano fechado, ele terá um ciclo Hamiltoniano, então G é denominado grafo Hamiltoniano.

O percurso *1, 5, 6, 2, 3, 7, 8, 4, 1* é um ciclo Hamiltoniano, G é Hamiltoniano. Nesse caso não pode ocorrer repetições de vértices nem de arestas.

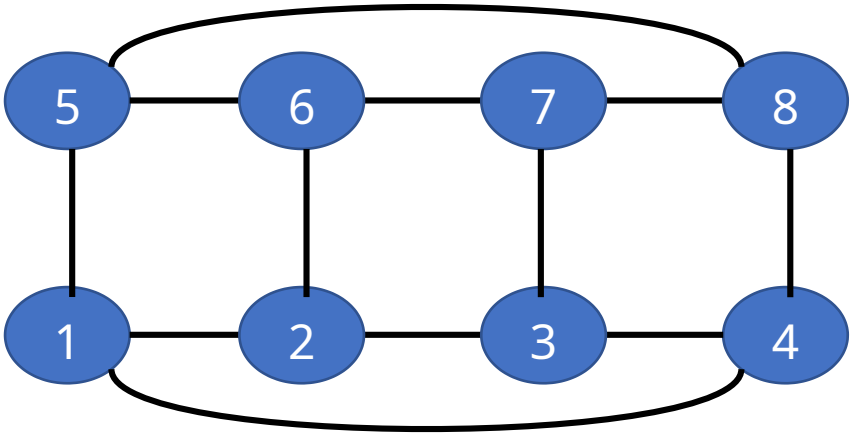


Se grafo G possui apenas um passeio por todos os seus vértices sem repetições, então G possui um caminho Hamiltoniano, e será chamado de grafo semi-Hamiltoniano.

Distância

Denomina-se distância d(v,w) entre dois vértices v,w de um grafo ao comprimento do menor caminho entre v e w (menor contagem de arestas entre v e w).

A distância entre os vértices 1 e 8, no grafo da figura é igual a 2, isto é, d(1,8) = 2.

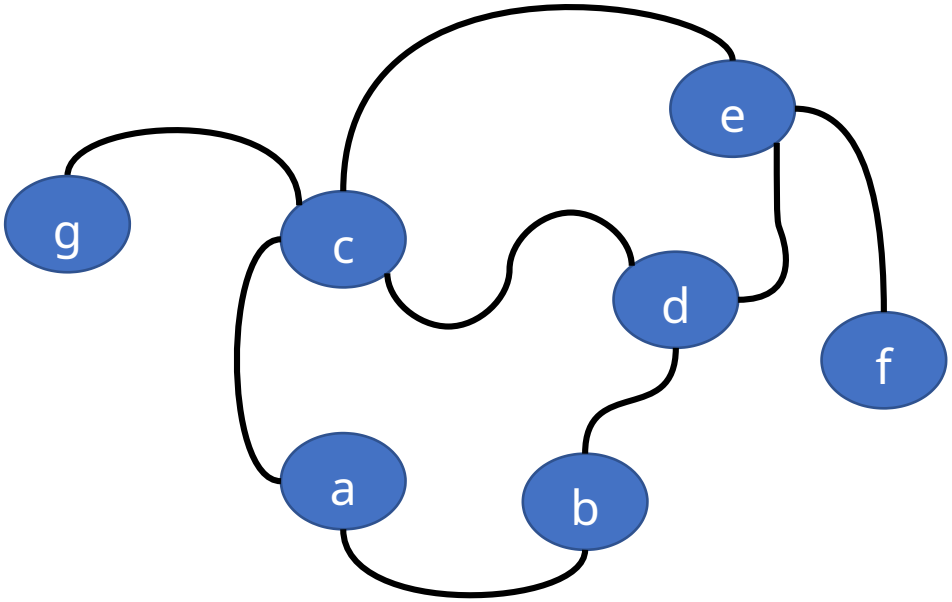


Distância

Seja G(V,E), a **excentricidade** de um vértice *v* é igual à **máxima distância** entre v e w, para todo v ∈ V e w ∈ V.

O centro de G é o subconjunto dos vértices de excentricidade mínima.

A tabela abaixo apresenta as distâncias entre *a* e os demais vértices do grafo



	distâncias de *a* (comprimento do menor caminho)
b	1
c	1
d	2
e	2
f	3
g	2

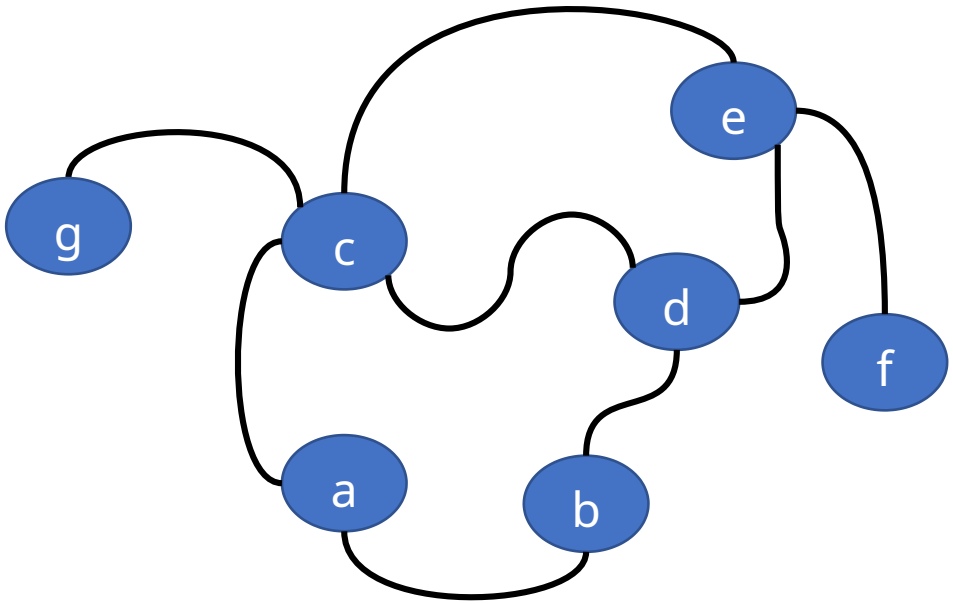
Excentricidade de em relação ao vértice "*a*" *(o maior dentre os mais curtos caminhos)*

Distância

Seja G(V,E), a excentricidade de um vértice *v* é igual à máxima distância entre v e w, para todo v ∈ V e w ∈ V.

O **centro** de G é o subconjunto dos vértices de **excentricidade mínima**.

A tabela abaixo apresenta as excentricidades dos vértices do grafo cujo centro corresponde ao subconjunto dos vértices {c,d,e}.



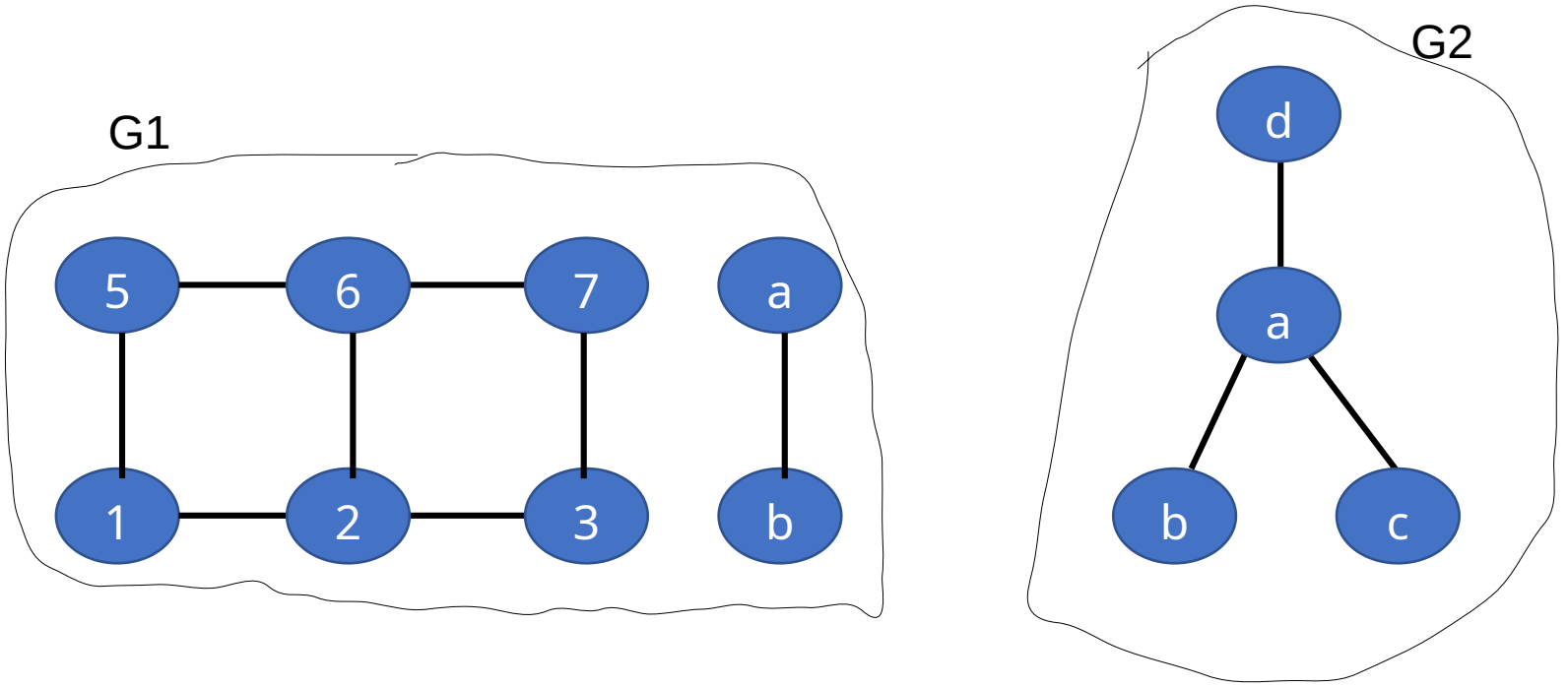
	excentricidades
a	3
b	3
c	2
d	2
e	2
f	3
g	3

Componente Conexo

Um grafo G(V,E) é denominado conexo quando existe percurso entre cada par de vértices de G. Caso contrário G é desconexo.

A representação geométrica de um grafo desconexo é necessariamente não contígua.

Um grafo G sem arestas é totalmente desconexo.



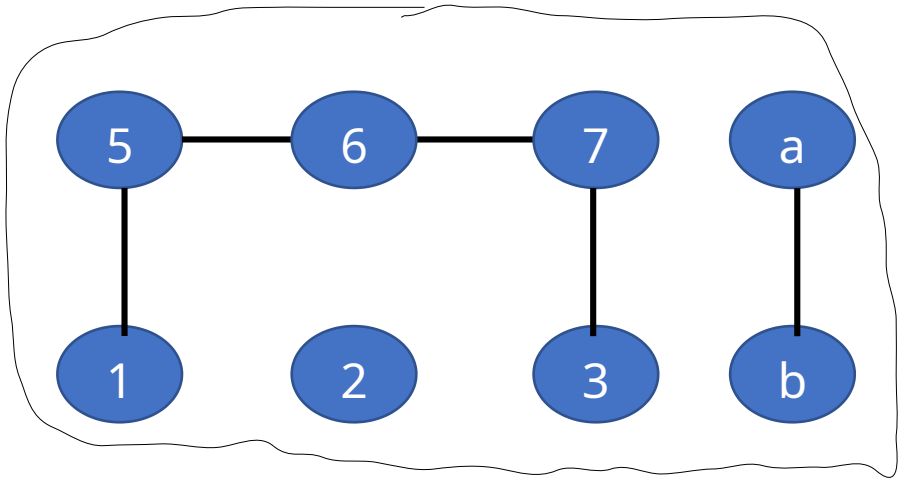
Componente Conexo

Um grafo G(V,E) é denominado conexo quando existe percurso entre cada par de vértices de G. Caso contrário G é desconexo.

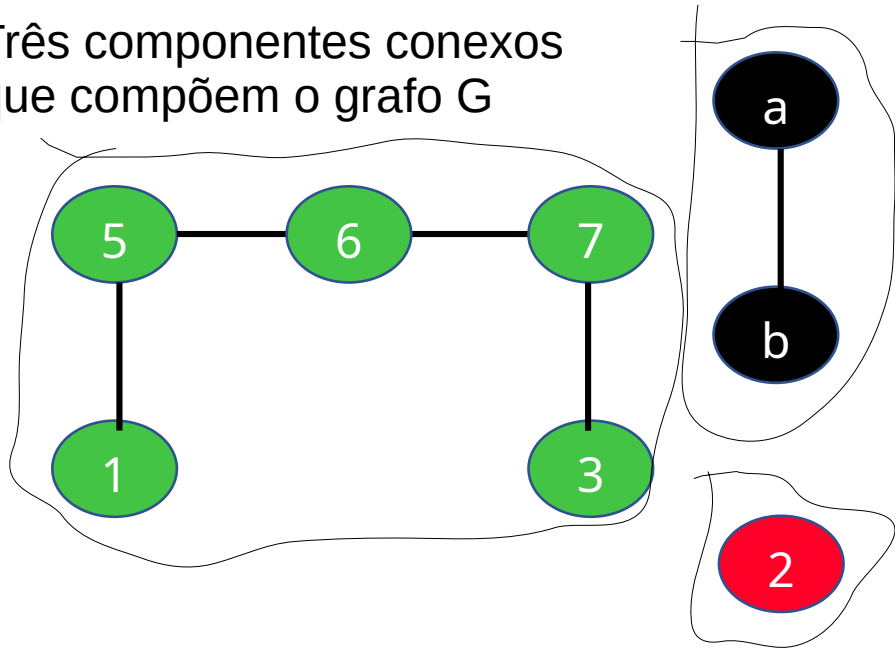
Um componente conexo de um grafo é um subgrafo maximal.

Um componente conexo é o maior subgrafo possível que mantém a conectividade entre seus vértices: caso você acrescente um novo nó do grafo ao subgrafo, este subgrafo deixará de ser um componente conexo.

Grafo G

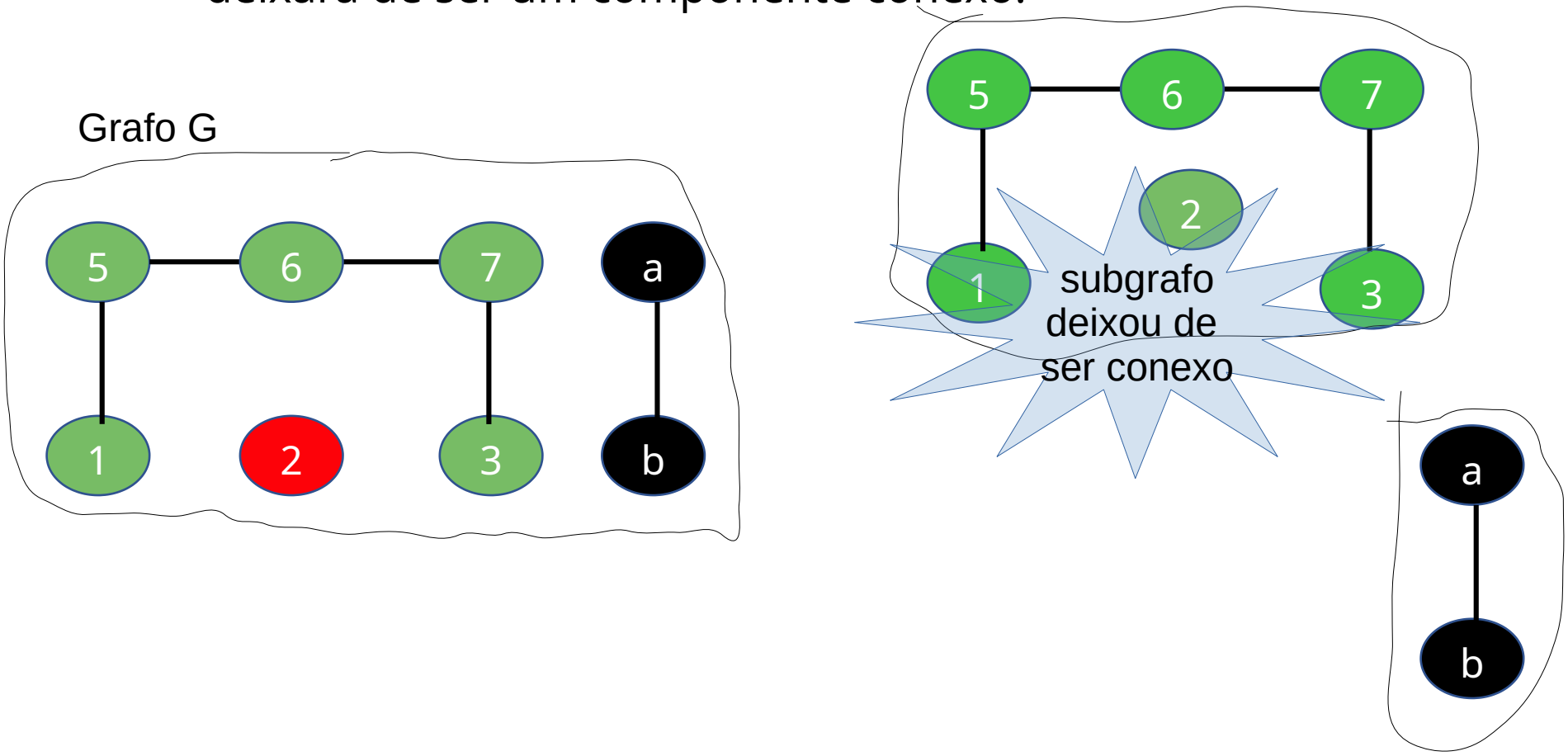


Três componentes conexos que compõem o grafo G



Componente Conexo

Um componente conexo é o maior subgrafo possível que mantém a conectividade entre seus vértices: caso você acrescente um novo nó do grafo ao subgrafo, este subgrafo deixará de ser um componente conexo.



Buscas sistemáticas em grafos

Busca em um Grafo

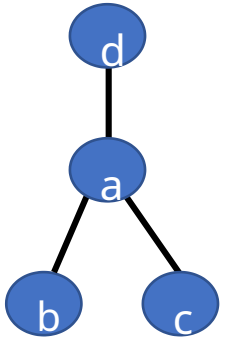
Eventualmente queremos apenas acessar todos os nós de um grafo simples e conexo G(V,E) em alguma ordem sem levar em conta algum peso de aresta (como se cada aresta apresentasse peso unitário);

A ordem dos nós acessados determina um percurso no grafo, usualmente esse processo é conhecido como uma “busca em um grafo”;

Para percorrer todos os vértices os algoritmos podem precisar repetir certas arestas/vértices, porém, essas repetições não são listadas na composição final do percurso de busca;

Exemplo:

no grafo ao lado, o algoritmo de busca em profundidade precisa repetir uma visita ao vértice ‘a’, porém, na sua listagem final teremos o seguinte resultado: d, a, b, c



Percurso/Busca em um Grafo

Os processos de busca são de dois tipos: em profundidade (DFS) e em largura (BFS)

Vamos ao algoritmo de Busca em Largura (BFS) e Busca em Profundidade (DFS) também apresentados na seção CAMINHO MÍNIMO E ÁRVORE GERADORA MÍNIMA do livro da Judith Gersting (disponível no formato ebook e fisicamente no acervo da biblioteca do CCT).

Fundamentos Matemáticos para a Ciência da Computação

Judith L. Gersting

Busca em profundidade

Busca em Profundidade (DFS): já foi estudado em EDA para grafos do tipo árvore binária e pode ser aplicado a grafos em geral

Partindo de um *v* arbitrário, marca-se esse nó como "visitado" e se processa esse nó (um *print* do seu conteúdo, por exemplo);

A partir de *v*, visitando e processando os nós, descendo tão longe quanto possível até não existirem vértices não visitados nesse caminho. Então, retorna-se pelo mesmo caminho explorando quaisquer caminhos laterais, até voltar ao vértice inicial *v*.

Se for o caso exploramos possíveis caminhos ainda inexplorados restantes a partir de *v*.

Busca em profundidade

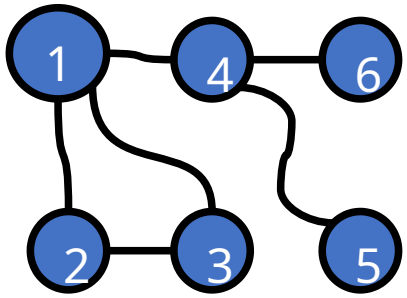
Grafo G(V,E) grafo simples e conexo:
Estruturas auxiliares:

- Matriz de adjacências (MatAdj)
- Vetor de marcações (vetMarca) dos visitados inicialmente zerada

Chamada:
DFS(grafo, raiz, MatAdj, vetMarca)

Função:

```
DFS(G,v, M, marca)
    marca[v]=1
    print(v)
    para cada w adjacente a v
        se marca[w]=0
            DFS(G,n,M,marca)
        senão
            retorna
    retorna
```

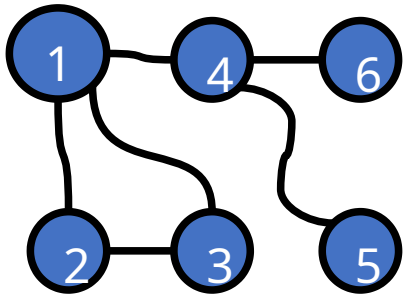


Busca em profundidade

Raiz= v1

vértice	1	2	3	4	5	6
Marca	0	0	0	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)



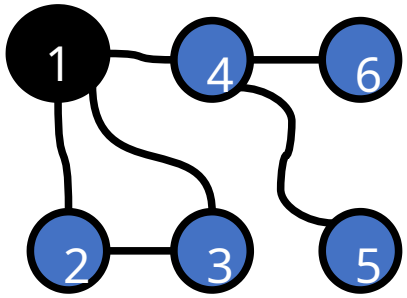
Busca em profundidade

Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	0	0	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)
… print v1

DFS(grafo,v2,MatAdj,vetMarca)



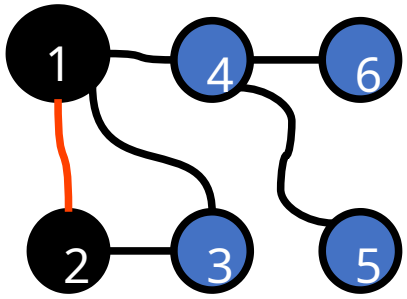
Busca em profundidade

Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	x	0	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v2,MatAdj,vetMarca)
...print v2



Busca em profundidade

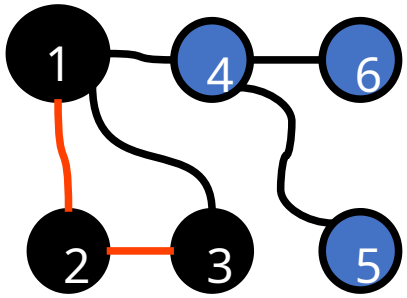
Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	x	x	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v2,MatAdj,vetMarca)

DFS(grafo,v3,MatAdj,vetMarca)
...print v3



Busca em profundidade

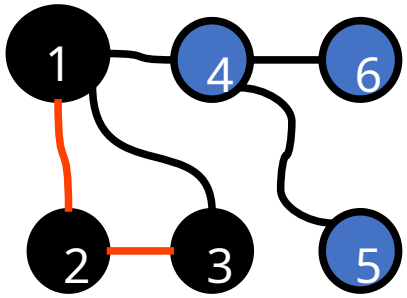
Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	x	x	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v2,MatAdj,vetMarca)

DFS(grafo,v3,MatAdj,vetMarca)



Busca em profundidade

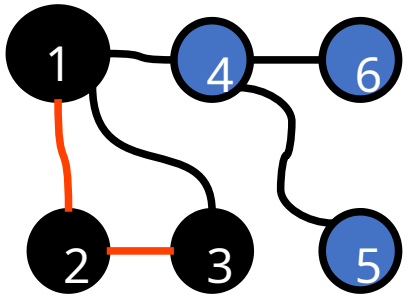
Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	x	x	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v2,MatAdj,vetMarca)

DFS(grafo,v3,MatAdj,vetMarca)



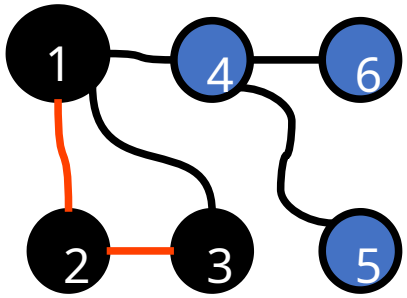
Busca em profundidade

Raiz= v1

vértice	1	2	3	4	5	6
Marca	x	x	x	0	0	0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

para cada *w* adjacente a *v*
DFS(grafo,v4,MatAdj,vetMarca)



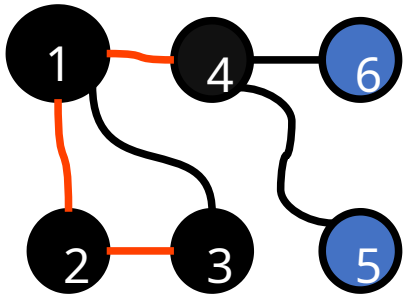
Busca em profundidade

Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x 0 0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)
...print v4



Busca em profundidade

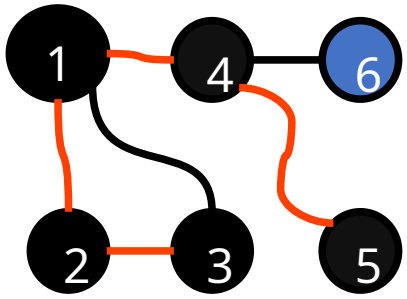
Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x x 0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)

DFS(grafo,v5,MatAdj,vetMarca)
...print v5



Busca em profundidade

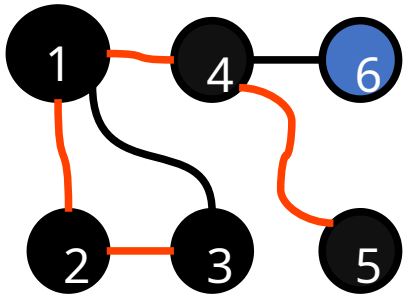
Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x x 0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)

DFS(grafo,v5,MatAdj,vetMarca)



Busca em profundidade

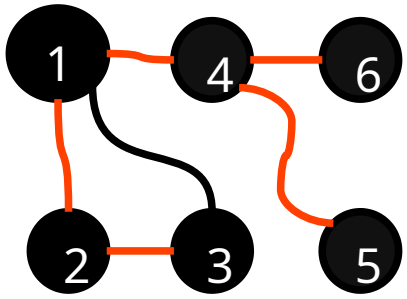
Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x x 0

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)

DFS(grafo,v6,MatAdj,vetMarca)
...print v6



Busca em profundidade

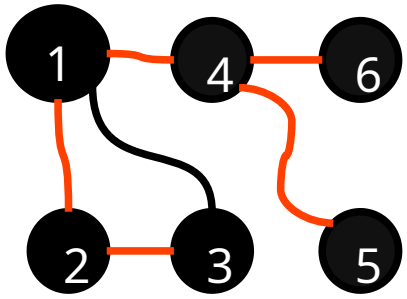
Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x x x

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)

DFS(grafo,v6,MatAdj,vetMarca)



Busca em profundidade

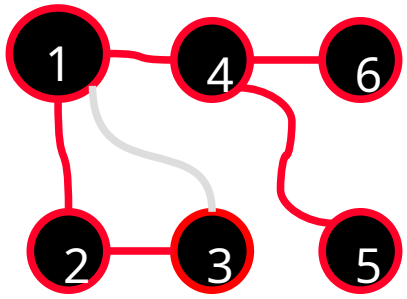
Raiz= v1

-------- 1 2 3 4 5 6
Marca x x x x x x

Chamada:
DFS(grafo,v1,MatAdj,vetMarca)

DFS(grafo,v4,MatAdj,vetMarca)

DFS(grafo,v6,MatAdj,vetMarca)



Busca em Largura

Busca em Largura (BFS): também já foi estudado em EDA para grafos do tipo árvore binária e pode ser aplicado a grafos em geral

A partir de um vértice arbitrário *v*, procuram-se todos os seus vértices adjacentes, depois os adjacentes a esses e assim por diante, como círculos concêntricos de ondas em um pequeno lago.

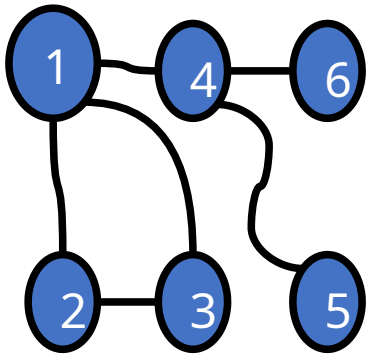
Busca em Largura

Grafo G(V,E) simples e conexo:
Estruturas auxiliares:

- Matriz de adjacências (MatAdj);
- Vetor de marcações (vetMarca) dos visitados inicialmente zerada

Chamada: BFS(grafo,raiz,MatAdj)
Função: BFS(G,v,M)

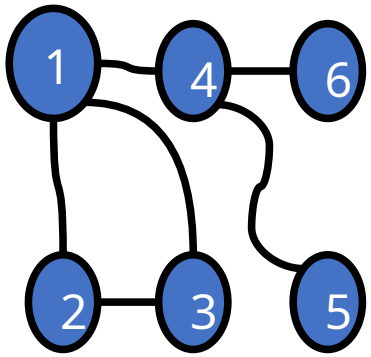
```
marca[ ] = zeros
F = cria(Fila) /* F é uma fila */
marca[v]=1
print(v)
insere(F,v)
enquanto F não é vazia:
    para cada w adjacente à frente(F):
        se marca[w]=0
            marca[w]=1
            print(w)
            insere(F,w)
    remove(F)
```



Busca em Largura

Raiz= v1

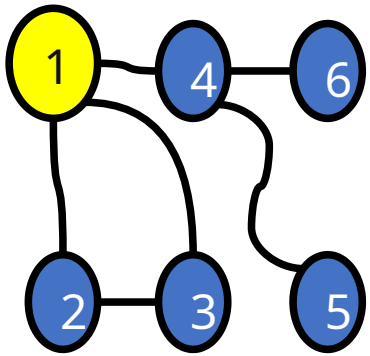
Chamada:
BFS(grafo,v1,M)



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca 0 0 0 0 0 0 Frente:-- Cauda:-- Fila: --	vértice v1 v2 v3 v4 v5 v6 Marca x 0 0 0 0 0 Frente: v1 Cauda: v1 Fila: v1
print	v1

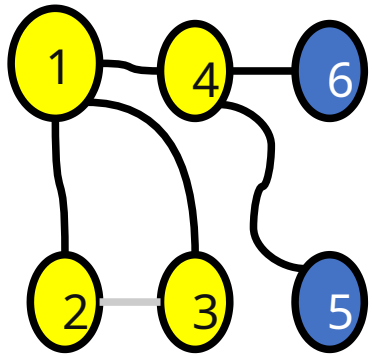
```
BFS(grafo,v,M)
    cria marca[ ] e cria fila[ ]
    marca[v]=x
    print(v)
    insere(F,v)
```



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x 0 0 0 0 0 Frente: v1 Cauda: v1 Fila: v1	vértice 1 2 3 4 5 6 Marca x x x x 0 0 Frente: v1 Cauda: v4 Fila: v1 v2 v3 v4
print	v1 v2 v3 v4

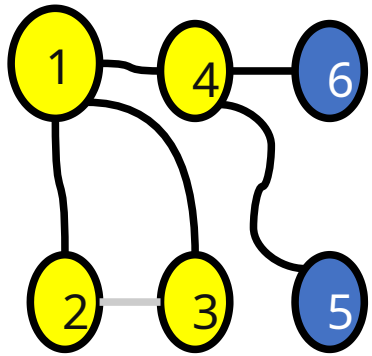
```
BFS(grafo,v,M)
    …
    enquanto fila F não vazia (1ª iteração)
        para cada w adjacente à frente(F)
            se marca[w]=0
                marca[w]=x;
                print(w);
                insere(F,w)
        remove(F)
```



Busca em Largura

Raiz= v1	
antes	depois
vértice 1 2 3 4 5 6 Marca x x x x 0 0 Frente: v1 Cauda: v4 Fila: v1 v2 v3 v4	vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v2 Cauda: v4 Fila: ~~v1~~ v2 v3 v4
print	v1 v2 v3 v4

```
BFS(grafo,v,M)
    …
    enquanto fila F não vazia (1ª iteração)
        para cada w adjacente à frente(F)
            se marca[w]=0
                marca[w]=x;
                print(w);
                insere(F,w)
        remove(F)
```



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v2 Cauda: v4 Fila: v2 v3 v4	vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v3 Cauda: v4 Fila: ~~v2~~ v3 v4
print	v1 v2 v3 v4

BFS(grafo,v,M)

…

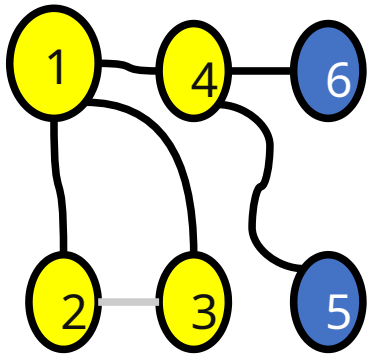
enquanto fila F não vazia (2ª iteração)

para cada *w* adjacente à frente(F)

Frente da fila = v2: cada *w* adjacente a v2 já foi marcado. Aqui ocorre apenas a remoção de v2 da fila

insere(F,w)

remove(F)



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v3 Cauda: v4 Fila: v3 v4	vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v4 Cauda: v4 Fila: ~~v3~~ v4
print	v1 v2 v3 v4

BFS(grafo,v,M)

…

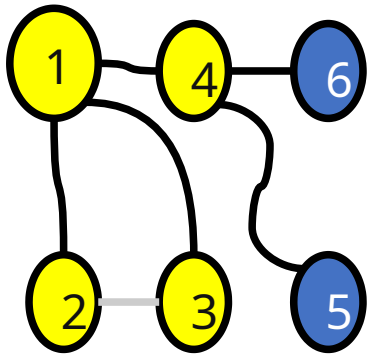
enquanto fila F não vazia (3ª iteração)

para cada *w* adjacente à frente(F)

Frente da fila = v3: cada *w* adjacente a v3 já foi marcado. Aqui ocorre apenas a remoção de v3 da fila

insere(F,w)

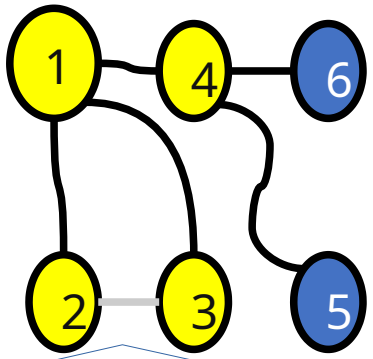
remove(F)



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x 0 0 Frente: v4 Cauda: v4 Fila: v4	vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v4 Cauda: v6 Fila: v4 v5 v6
print	v1 v2 v3 v4 v5 v6

```
BFS(grafo,v,M)
    ...
    enquanto fila F não vazia (4ª iteração)
        para cada w adjacente à frente(F)
            se marca[w]=0
                marca[w]=x;
                print(w);
                insere(F,w)
        remove(F)
```

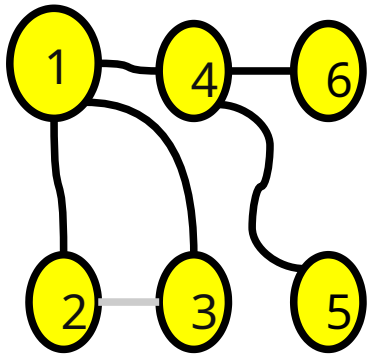


Para v4 há adjacentes v5 e v6 não marcados.

Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v4 Cauda: v6 Fila: v4 v5 v6	vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v5 Cauda: v6 Fila: ~~v4~~ v5 v6
print	v1 v2 v3 v4 v5 v6

```
BFS(grafo,v,M)
    …
    enquanto fila F não vazia (4ª iteração)
        para cada w adjacente à frente(F)
            se marca[w]=0
                marca[w]=x;
                print(w);
                insere(F,w)
        remove(F)
```



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v5 Cauda: v6 Fila: v5 v6	vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v6 Cauda: v6 Fila: ~~v5~~ v6
print	v1 v2 v3 v4 v5 v6

BFS(grafo,v,M)

…

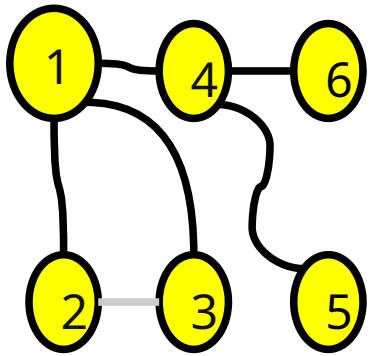
enquanto fila F não vazia (5ª iteração)

para cada *w* adjacente à frente(F)

v5: seus adjacentes já foram todos marcados. Ocorre apenas a remoção de v5 da fila

insere(F,w)

remove(F)



Busca em Largura

Raiz= v1	
antes	depois
vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: v6 Cauda: v6 Fila: v6	vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: 7 Cauda: 6 Fila: ~~v6~~ FILA VAZIA
print	v1 v2 v3 v4 v5 v6

BFS(grafo,v,M)

…

enquanto fila F não vazia (6ª iteração)

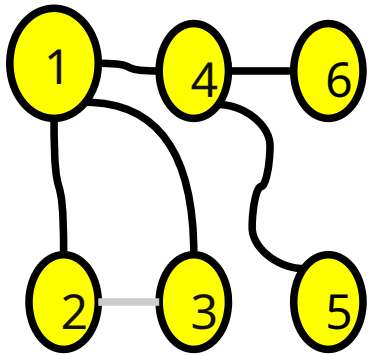
para cada *w* adjacente à frente(F)

se marca[w]=0

v6: seus adjacentes já foram todos marcados. Ocorre apenas a remoção de v6 da fila

insere(F,w)

remove(F)

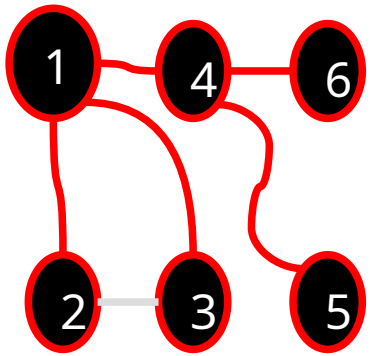


Busca em Largura

Raiz= v1	
vértice v1 v2 v3 v4 v5 v6 Marca x x x x x x Frente: Cauda: Fila: VAZIA	
Percurso BFS	v1 v2 v3 v4 v5 v6

Função:
BFS(grafo,v,M)

…
Fila vazia:
Fim do “enquanto fila F não vazia”



Exercício

Como você poderia aplicar percursos (DFS ou BFS) para determinar se um grafo é desconexo? Descreva uma estratégia.

Suponha que se deseja conhecer a distribuição (tamanho e quantidade) de cada componente conexo em um grafo, como você aplicaria BFS ou DFS para obter esse histograma de componentes conexos de um grafo?

Suponha que se deseje levantar uma medida que descreve a frequência na qual cada aresta se encontra no caminho mais curto entre pares de nós em uma rede (grafo). Descreva uma abordagem para tal computação. O que essa frequência significa em termos da intermediação operada por essas arestas na estrutura do grafo como uma rede de comunidades? O que acontece se for operada uma remoção sucessiva dessas arestas em termos do isolamento de comunidades subjacentes da rede?

TEG

Bibliografia

Básica

LUCCHESI, C. L. et alli. Aspectos Teóricos da Computação, Parte C: Teoria dos Grafos, projeto Euclides, 1979.

SANTOS, J. P. O. et alli. Introdução à Análise Combinatória. UNICAMP; 1995.

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

GERSTING, Judith L. Fundamentos Matemáticos para a Ciência da Computação. Rio de Janeiro. 3a Ed. Editora.

Complementar:

1.) CORMEN, T. Introduction to Algorithms, third edition, MIT press, 2009

2.) ROSEN, K. Discrete Mathematics and its applications, seventh edition, McGraw Hill, 2011.

3.) WEST, Douglas, B. Introduction to Graph Theory, second edition, Pearson, 2001.

4.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984.

5.) SEDGEWICK, R. Algorithms in C - part 5 - Graph Algorithms, third edition, 2002, Addison-Wesley.

6.) GOLDBARG, M., GOLDBARG E., Grafos: Conceitos, algoritmos e aplicações. Editora Elsevier, 2012.

7.) BONDY, J.A., MURTY, U.S.R., Graph Theory with applications , Springer, 1984

8.) FEOFILOFF, P., KOHAYAKAWA, Y., WAKABAYASHI, Y., uma introdução sucinta à teoria dos grafos. 2011. (www.ime.usp.br/~pf/teoriadosgrafos)

9.) DIESTEL, R. Graph Theory, second edition, springer, 2000

10.) FURTADO, A. L. Teoria de grafos. Rio de janeiro. Editora LTC. 1973.

11.) WILSON, R.J. Introduction to Graph Theory. John Wiley & Sons Inc., 1985

12.) BOAVENTURA NETTO , P. O. Grafos: Teoria, Modelos, Algoritmos. Edgard Blucher, SP, quinta edição

Tutoriais, artigos, notas de aula...

Vários livros podem ser acessados np formato eletrônico (e-book) via

https://www.udesc.br/bu/acervos/ebook

Exemplos:

Teoria Computacional de Grafos - Os Algoritmos

Jayme Luiz Szwarcfiter

Fundamentos Matemáticos para a Ciência da Computação

Judith L. Gersting

Grafos

Marco Goldbarg

Algoritmos - Teoria e Prática

Thomas Cormen