UnB - Universidade de Brasília
Departamento de Ciência da Computação
PPGI - Programa de Pós-graduação em Informática

Breadth First Search(BFS)

PPGI2223- PROJETO E COMPLEXIDADE DE ALGORITMOS *- Dr. RODRIGO BONIFACIO DE ALMEIDA*
Trabalho 01

James Taylor Faria Chaves (210005912)
Luisa Sinzker Fantin
Wedrey Nunes da Silva (210006048)

Brasília-DF, 01 de Dezembro de 2022

Roteiro

- Introdução
- Problemas
- Representações em Grafos
- Algoritmo BFS
- Análise
- Implementações

Introdução

Breadth-first search (BFS) é um dos algoritmos mais simples para transpassar um grafo.

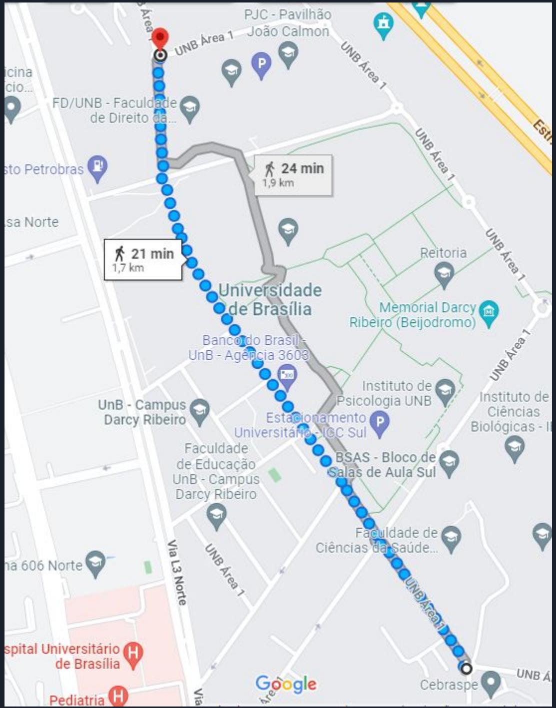
Dado um grafo G (V, E) e um vértice de origem (s), BFS explora sistematicamente as arestas de G para:

- Descobrir cada vértice que é acessível a partir de (s)
- Calcular a distância (menor número de arestas) de (s) a cada vértice alcançável.
- Produzir uma "árvore em largura" com raiz (s) que contém todos os vértices alcançáveis.

Base para: Prim's minimum-spanning tree algorithm / Dijkstra's single-source shortest-paths algorithm

Uso

Locomoção urbana



Teoria dos Grafos

- Estudo das relações entre objetos de um conjunto
- Grafo: estrutura usada para representar relações entre objetos de um determinado conjunto
- grafo trivial: grafo com um único vértice
- orientados ou não-orientado

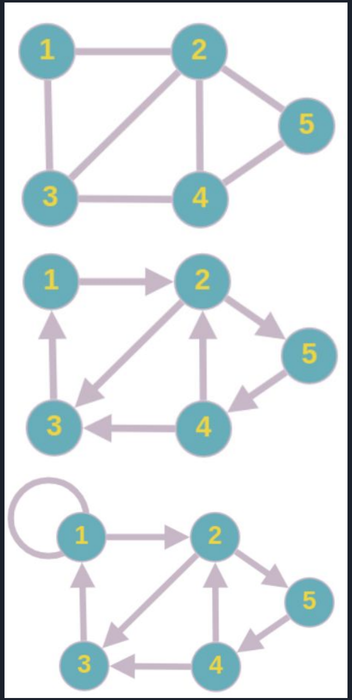
Teoria dos Grafos

G(V, E)

- Grafos não-orientados
    - V(*vertex/node*): conjunto não-vazio de vértices
    - E(*edges*): subconjunto de pares não ordenados ($\{v1, v2\} = \{v2, v1\}$) de V
        - $E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$
- Grafos orientados
    - V(*vertex/node*): conjunto não-vazio de vértices
    - E(*edges*): subconjunto de pares ordenados ($\{v1, v2\} \neq \{v2, v1\}$) de V
        - $E \subseteq \{\{x,y\} \mid y, x \in V^2 \text{ e } x \neq y\}$
        - {v1, v2}: v1 □ v2
        - {v2, v1}: v2 □ v1

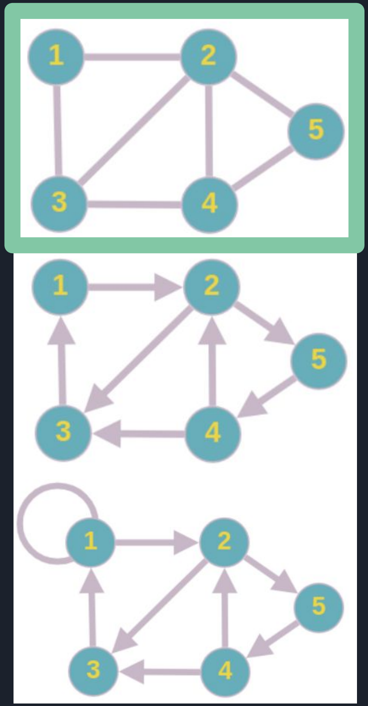
Representações comuns na computação

- Não-orientado ($E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$)
  - Mapas (pedestres, metro)
  - *Seven Bridges of Königsberg**
- Orientado ($E \subseteq \{\{x,y\} \mid y, x \in V^2 \text{ e } x \neq y\}$)
  - Mapas (automóveis)
  - *Control-flow graphs (CFG)*, grafo de fluxo de controle
- Multigrafo ($E \subseteq \{\{x,y\} \mid y, x \in V^2\}$)
  - Máquina de Estados



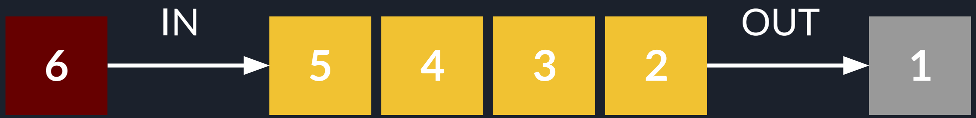
Representações comuns na computação

- Não-orientado ($E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$)
  - Mapas (pedestres, metro)
  - *Seven Bridges of Königsberg**
- Orientado ($E \subseteq \{\{x,y\} \mid y, x \in V^2 \text{ e } x \neq y\}$)
  - Mapas (automóveis)
  - *Control-flow graphs (CFG)*, grafo de fluxo de controle
- Multigrafo ($E \subseteq \{\{x,y\} \mid y, x \in V^2\}$)
  - Máquina de Estados



Fila e Pilha

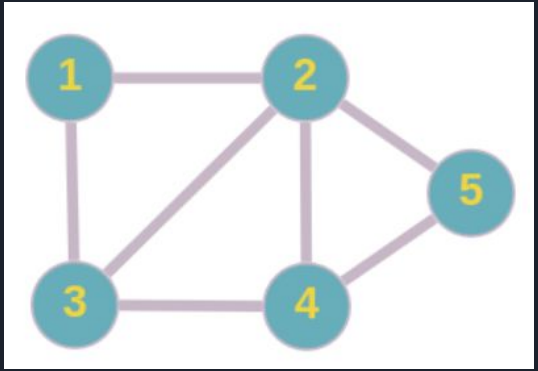
- Fila: tipo de lista encadeada que segue o padrão *First In, First Out (FIFO)*



- Pilha: tipo de lista encadeada que segue o padrão *Last In, First Out (LIFO)*

Exemplo

$$E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$$

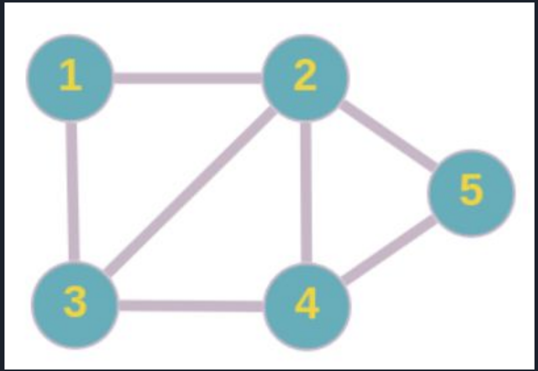


$V = \{1, 2, 3, 4, 5\}$

$E = \{ \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{4, 5\} \}$

Exemplo

$E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$



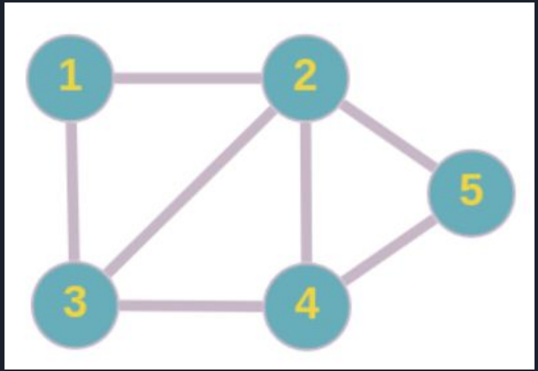
$V = \{1, 2, 3, 4, 5\}$

$E = \{ \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{4, 5\} \}$

Como encontrar o menor caminho entre dois vértices?

Exemplo

$E \subseteq \{\{x,y\} \mid y, x \in V \text{ e } x \neq y\}$



V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

Como encontrar o menor caminho entre dois vértices?

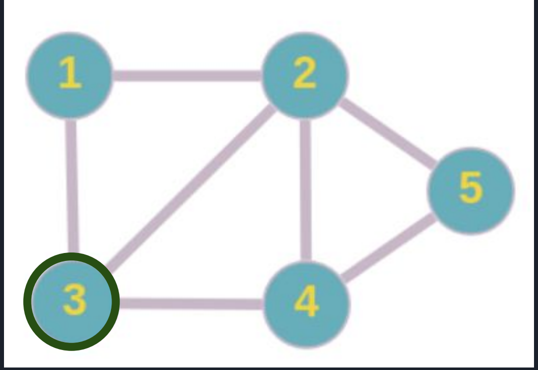
Breadth-first search (*Busca em largura*)

Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

1. A partir do vértice inicial escolhido

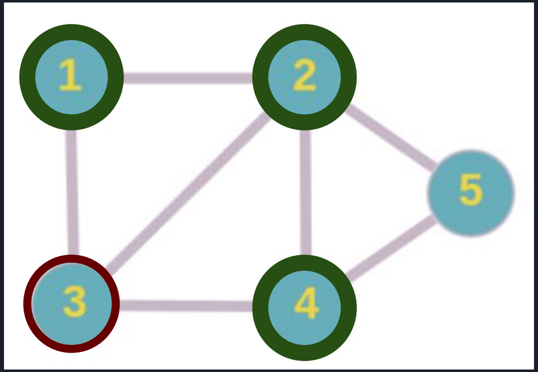


3

Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }



1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados

3	1	2	4

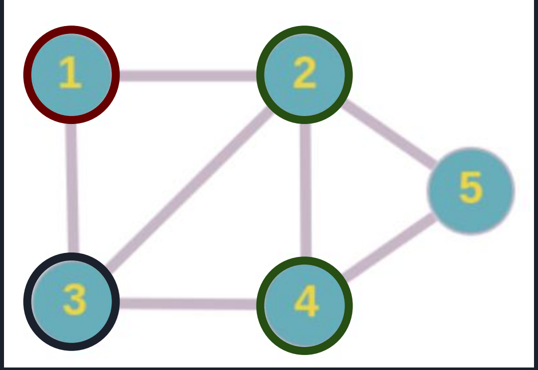
Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados
3. Repete-se o processo para cada vértice adjacente

3	1	2	4



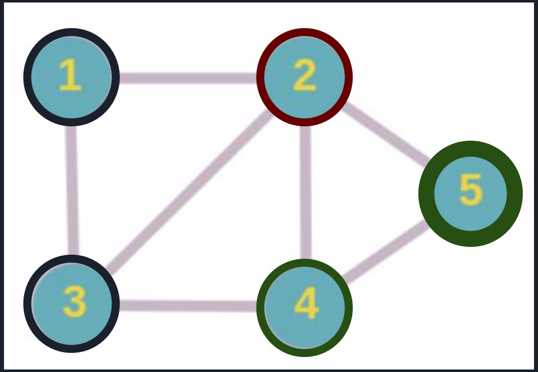
Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados
3. Repete-se o processo para cada vértice adjacente

3	1	2	4	5



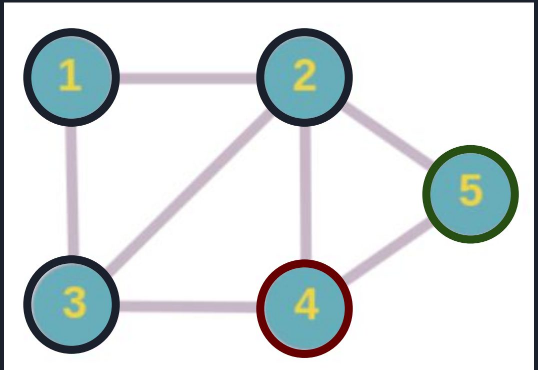
Busca em largura
Breadth-first Search

$V = \{1, 2, 3, 4, 5\}$

$E = \{ \{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 4\}, \{4, 5\} \}$

1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados
3. Repete-se o processo para cada vértice adjacente

3	1	2	4	5



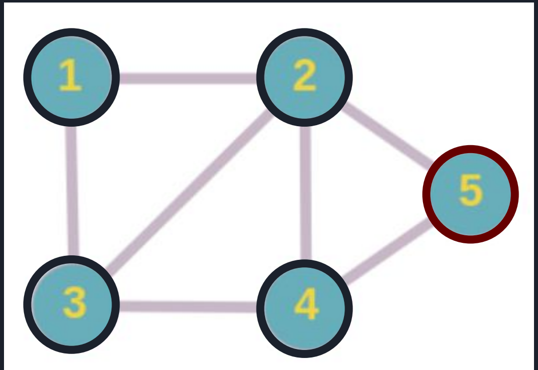
Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados
3. Repete-se o processo para cada vértice adjacente

3	1	2	4	5



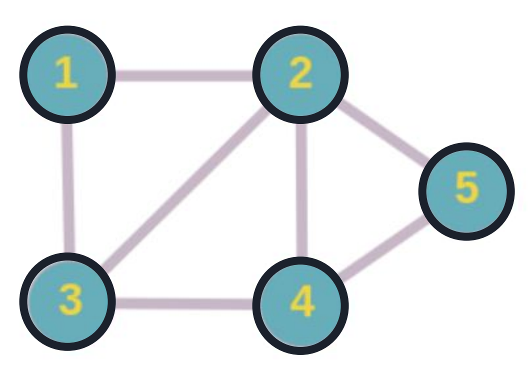
Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

1. A partir do vértice inicial escolhido
2. Visita-se os vértices adjacentes ainda não visitados
3. Repete-se o processo para cada vértice adjacente

3	1	2	4	5

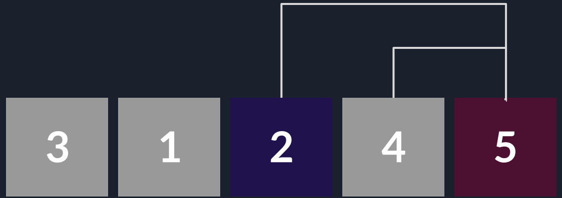


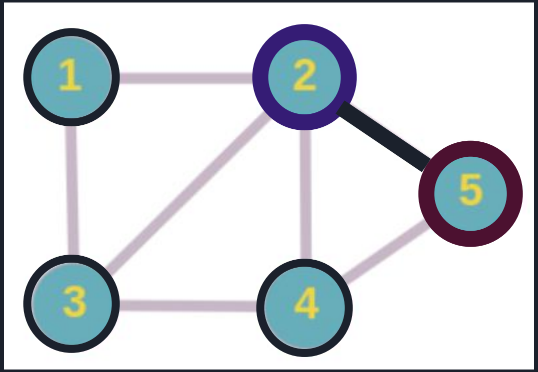
Busca em largura
Breadth-first Search

V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

Caminho inverso: 5□2



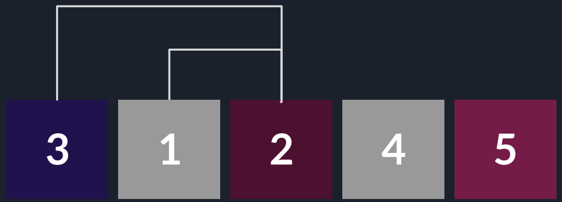


Busca em largura
Breadth-first Search

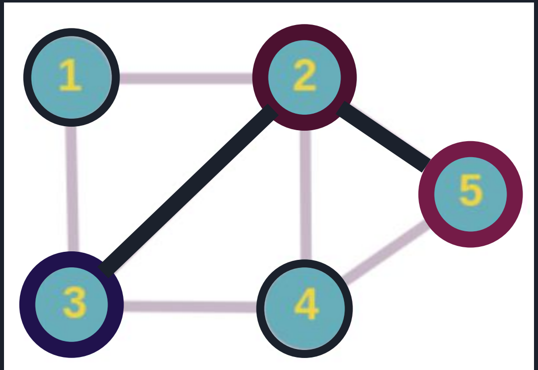
V = {1, 2, 3, 4, 5}

E = { {1, 2}, {1, 3}, {2, 3}, {2, 4}, {2, 5}, {3, 4}, {4, 5} }

Caminho inverso: 5□2□3

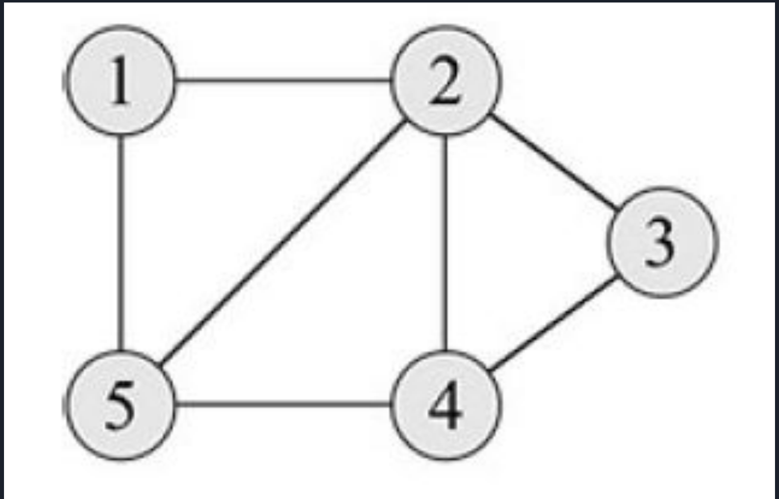


3□2□5



Algoritmo BFS

```
BFS(G, s)
    for cada vértice u ∈ V[G] − {s}
        u.cor = BRANCO
        u.d = ∞
        u.π = NIL
    s.cor = CINZENTO
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for cada v = Adj[u]
            if v.cor == BRANCO
                v.cor == CINZENTO
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.cor = PRETO
```



Representação de um grafo não dirigido

Algoritmo BFS - Implementação C#

```csharp
namespace BfsAlgorithm.BFS
{
    3 referências
    public class BreadthFirstSearch
    {
        private Vertex[] adjacentList;

        1 referência
        public BreadthFirstSearch(int s)
        {
            adjacentList = new Vertex[s];
            for (int i = 1; i < adjacentList.Length; i++)
                adjacentList[i] = new Vertex(i);
        }

        1 referência
        public void BFS(int source) ...

        14 referências
        public void AddVertex(int u, int v)...

        2 referências
        private void Print(Vertex u, Vertex v)...

        1 referência
        public void PathVertex(int u, int v)...
    }
}
```

Obtém o número de vértices através do construtor da classe e inicializa a lista de adjacências para o comprimento especificado.

```csharp
namespace BfsAlgorithm.BFS
{
    19 referências
    public class Vertex
    {
        5 referências
        public int vertex { get; set; } //vertex number
        8 referências
        public Vertex next { get; set; }  //reachable vertex
        6 referências
        public char color { get; set; }
        5 referências
        public int d { get; set; }
        5 referências
        public Vertex p { get; set; }
        3 referências
        public Vertex(int val)
        {
            vertex = val;
            d = Int32.MaxValue;
            color = 'W';
        }
    }
}
```

Estrutura do vértice

Algoritmo BFS - Implementação C#

```csharp
public void BFS(int source)
{
    Queue<Vertex> queue = new Queue<Vertex>();
    Vertex s = adjacentList[source];
    for (int i = 1; i < adjacentList.Length; i++)
    {
        Vertex u = adjacentList[i];

        if (u.vertex!= source)
        {
            u.color = 'W';
            u.d = Int32.MaxValue;
            u.p = null;
        }
    }
    s.color = 'G';
    s.d = 0;
    s.p = null;
    queue.Enqueue(s);
    while (queue.Count > 0)
    {
        Vertex u = queue.Dequeue();
        Vertex v = u.next;
        while (v != null)
        {
            Vertex actualV = adjacentList[v.vertex];

            if (actualV.color == 'W')
            {
                actualV.color = 'G';
                actualV.d = u.d + 1;
                actualV.p = u;
                queue.Enqueue(actualV);
            }
            v = v.next;
        }
        u.color = 'B';
    }
}
```

```
BFS(G, s)
    for cada vértice u ∈ V[G] − {s}
        u.cor = BRANCO
        u.d = ∞
        u.π = NIL
    s.cor = CINZENTO
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for cada v = Adj[u]
            if v.cor == BRANCO
                v.cor == CINZENTO
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.cor = PRETO
```

Algoritmo BFS - Implementação C#

```csharp
public void AddVertex(int u, int v)
{
    Vertex uTmp = adjacentList[u];
    while (uTmp.next != null)
        uTmp = uTmp.next;

    uTmp.next = new Vertex(v);

    Vertex vTmp = adjacentList[v];
    while (vTmp.next != null)
        vTmp = vTmp.next;

    vTmp.next = new Vertex(u);
}
```

O método AddVertex, adiciona um novo vértice à lista de adjacências.

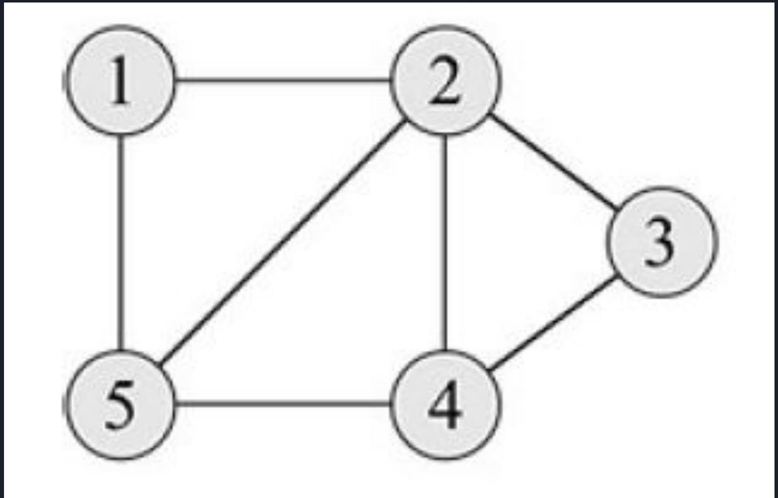
Parâmetro ‘u’ =>vértice onde ‘v’ tem que ser ligado.

Parâmetro ‘v’ => vértice que deve ser vinculado.

Algoritmo BFS - Implementação C#

Criando um grafo através de uma coleção de listas de adjacências

```
public static void Search()
{
    BreadthFirstSearch bfs = new BreadthFirstSearch(6);
    bfs.AddVertex(1, 2);
    bfs.AddVertex(1, 5);

    bfs.AddVertex(2, 1);
    bfs.AddVertex(2, 5);
    bfs.AddVertex(2, 3);
    bfs.AddVertex(2, 4);

    bfs.AddVertex(3, 2);
    bfs.AddVertex(3, 4);

    bfs.AddVertex(4, 2);
    bfs.AddVertex(4, 5);
    bfs.AddVertex(4, 3);

    bfs.AddVertex(5, 4);
    bfs.AddVertex(5, 1);
    bfs.AddVertex(5, 2);

    bfs.PathVertex(1, 4);
}
```



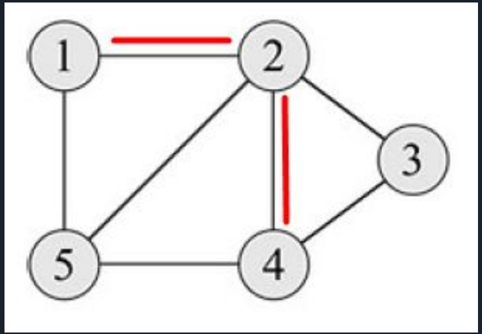
Duas representações de um grafo não dirigido

Algoritmo BFS - Implementação C#

O menor número de arestas de 1 a 4 é:

```
public void PathVertex(int u, int v)
{
    BFS(u);
    Console.Write("Distance (smallest number of edges) from ("+u + ") to (" + v + ") is: ");
    Print(adjacentList[u], adjacentList[v]);
}
```

```
private void Print(Vertex u, Vertex v)
{
    if (u == v)
        Console.Write(u.vertex + " ");
    else if (v.p == null)
        Console.Write("Unreachable path from u to v");
    else
    {
        Print(u, v.p);
        Console.Write(v.vertex + " ");
    }
}
```



Console de Depuração do Microsoft Visual Studio

```
Distance (smallest number of edges) from (1) to (4) is: 1 2 4
```

Disponível em: github.com/wedrey/bfs

Análise

Complexidade de tempo quando se conhece a quantidade de Vértices.

$$O(V + E)$$

Complexidade de espaço

$$O(V)$$

Complexidade de Tempo e Espaço para quantidade de V extremamente grande ou infinita

$$O(b^{d+1})$$

d => Distância da raiz

b => Branching factor (número de filhos tem cada nó)

```
BFS(G, s)
    for cada vértice u ∈ V[G] − {s}
        u.cor = BRANCO
        u.d = ∞
        u.π = NIL
    s.cor = CINZENTO
    s.d = 0
    s.π = NIL
    Q = ∅
    ENQUEUE(Q, s)
    while Q ≠ ∅
        u = DEQUEUE(Q)
        for cada v = Adj[u]
            if v.cor == BRANCO
                v.cor == CINZENTO
                v.d = u.d + 1
                v.π = u
                ENQUEUE(Q, v)
        u.cor = PRETO
```

Referências

CORMEN, T. H. et al. Algorithms: theory and practice. **Rio de Janeiro-RJ-Brazil: Elsevier**, 2012.

AS PONTES DE KÃ□NIGSBERG | MATEMATECA. (n.d.). https://matemateca.ime.usp.br/acervo/pontes_konigsberg.html

Russell, Stuart; Norvig, Peter (2003) [1995]. Artificial Intelligence: A Modern Approach (2nd ed.). Prentice Hall. ISBN 978-0137903955.