

Teoria dos Grafos – Algoritmo BFS – Busca em Largura

Atividade Hands-on em Laboratório – 03

Prof. Calvetti

1. Introdução

Um problema fundamental em grafos é descobrir como explorá-lo de forma sistemática. Muitas aplicações são abstraídas como problemas de busca.

Os algoritmos de busca em grafos são a base para o estudo de diversos outros algoritmos mais gerais em grafos.

Como motivação para o uso dos algoritmos de busca em grafos, poderíamos levantar a seguinte questão: Como saber se existem caminhos simples entre dois vértices?

Um algoritmo de busca é um algoritmo que esquadrinha um grafo andando pelos arcos de um vértice a outro. Depois de visitar a ponta inicial de um arco, o algoritmo percorre o arco e visita sua ponta final. Cada arco é percorrido no máximo uma vez.

Há muitas maneiras de organizar uma busca. Cada estratégia de busca é caracterizada pela ordem em que os vértices são visitados.

Neste hands-on, trataremos do Algoritmo de Busca em Largura (= breadth-first search = BFS), ou busca BFS.

Na teoria dos grafos, busca em largura (ou busca em amplitude, também conhecido em inglês por Breadth-First Search - BFS) é um algoritmo de busca em grafos utilizado para realizar uma busca ou travessia num grafo e estrutura de dados do tipo árvore.

Intuitivamente, começa-se pelo vértice raiz e explora todos os vértices vizinhos. Então, para cada um desses vértices mais próximos, exploramos os seus vértices vizinhos inexplorados e assim por diante, até que ele encontre o alvo da busca.

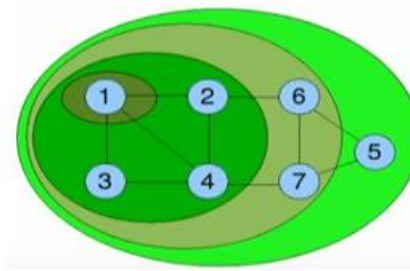
Formalmente, uma busca em largura é um método de busca que expande e examina sistematicamente todos os vértices de um grafo direcionado ou não-direcionado. Em outras palavras, podemos dizer que o algoritmo realiza uma busca exaustiva num grafo passando por todas as arestas e vértices do grafo. Sendo assim, o algoritmo deve garantir que nenhum vértice ou aresta será visitado mais de uma vez e, para isso, utiliza uma estrutura de dados fila para garantir a ordem de chegada dos vértices. Dessa maneira, as visitas aos vértices são realizadas através da ordem de chegada na estrutura fila e um vértice que já foi marcado não pode entrar novamente a esta estrutura.

Uma analogia muito conhecida para demonstrar o funcionamento do algoritmo é pintando os vértices de branco, cinza e preto. Os vértices na cor branca ainda não foram marcados e nem enfileirados, os da cor cinza são os vértices que estão na estrutura fila e os pretos são aqueles que já tiveram todos os seus vértices vizinhos enfileirados e marcados pelo algoritmo.

Seja $G = (V, A)$ e um vértice s , o Algoritmo de Busca em Largura (BFS) percorre as arestas de G descobrindo todos os vértices atingíveis a partir de s .

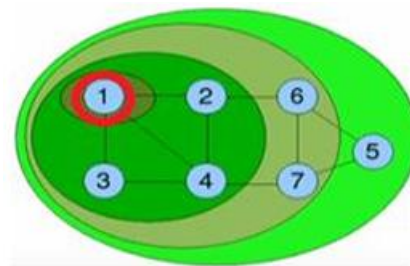
BFS determina a distância (em número de arestas) de cada um desses vértices a s .

Antes de se encontrar um vértice à distância $K+1$ de s , todos os vértices à distância k são encontrados.

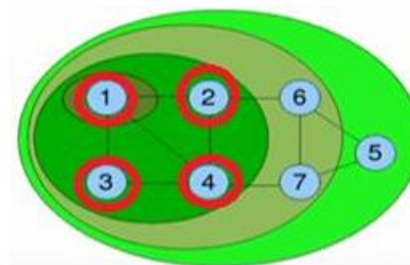


Por exemplo, vamos considerar que o algoritmo BFS será aplicado inicialmente ao Vértice 1.

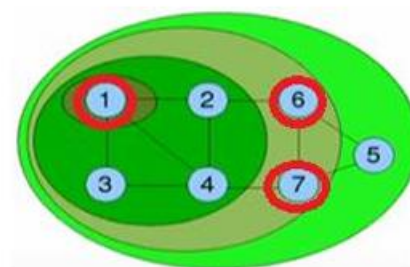
O vértice 1 está à distância zero dele mesmo.



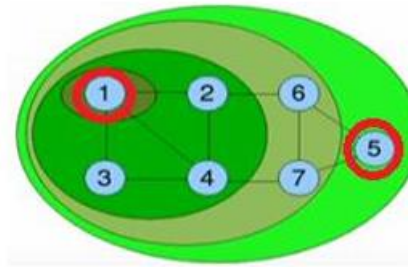
Em seguida, o algoritmo irá encontrar todos os vértices que estão à distância 1: 2, 3 e 4



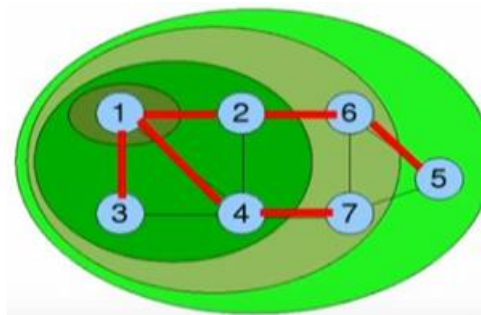
Em seguida, o algoritmo irá encontrar todos os vértices que estão à distância 2: 6 e 7



Em seguida, o algoritmo irá encontrar todos os vértices que estão à distância 3: 5



O algoritmo BFS produz uma árvore BFS com raiz em s , que contém todos os vértices acessíveis determinando o caminho mais curto (Caminho que contém o número mínimo de arestas) de s a t (em que t é um vértice acessível).

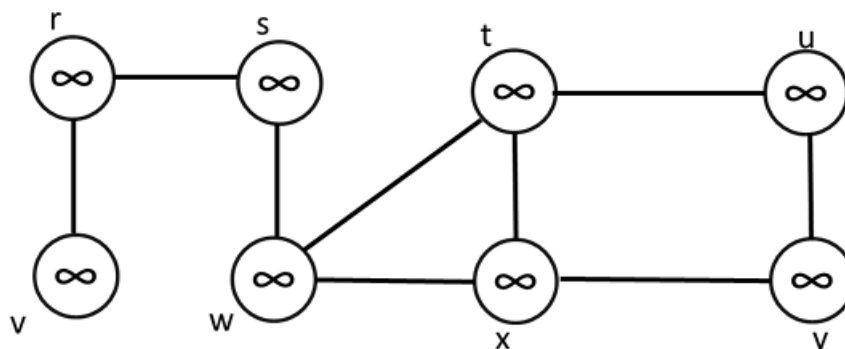


Para se organizar o processo de busca, pintam-se os vértices:

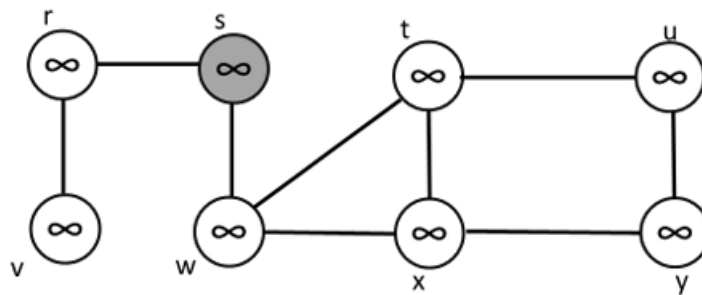
- ✓ Branco: ainda não foram descobertos
- ✓ Cinza: correspondem à fronteira. O vértice já foi descoberto, mas ainda não foram examinados seus vizinhos;
- ✓ Preto: São os vértices já descobertos e seus vizinhos já foram examinados.

Utiliza-se uma fila para se manter os vértices cinzas.

Exemplo: No início, todos os vértices são brancos e a distância é infinita.



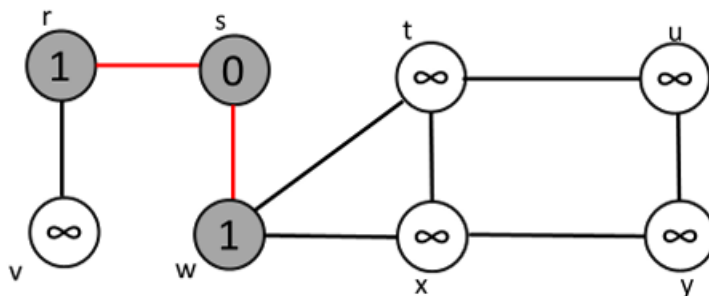
Vamos agora aplicar o algoritmo a partir do vértice inicial s .
O vértice inicial s é pintado de cinza (ele é considerado descoberto) e é colocado na fila.



Q:

s

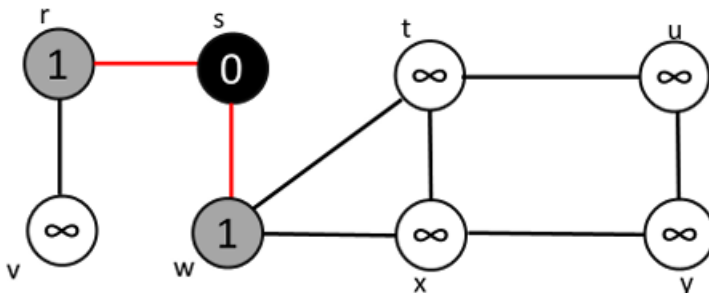
Retira-se o primeiro elemento da Fila e os adjacentes a ele são colocados em Q e pintados de cinza. Além disso, é atualizada a distância dos nós adjacentes e do pai.



Q:

w	r
---	---

Em seguida, o vértice s é colorido com preto, uma vez que seus vizinhos já foram visitados (descobertos).

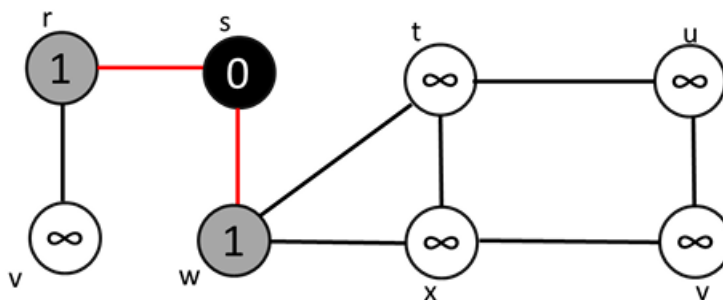


Q:

↓	
w	r

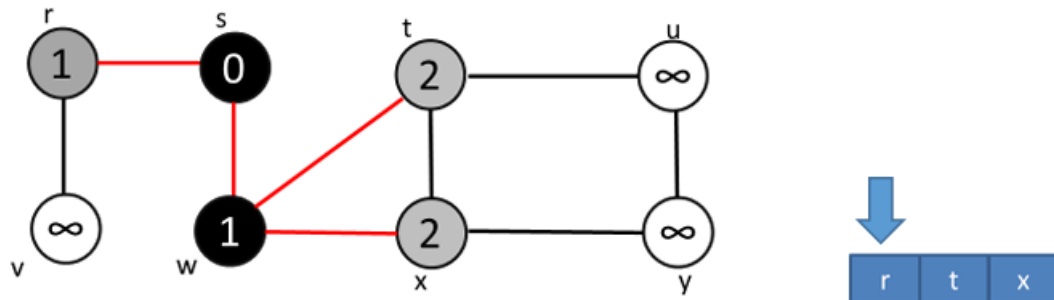
Agora, os dados da fila serão processados. O primeiro elemento da fila a ser processado é w e seus vizinhos são: t e x

w é removido da fila e entram na fila t e x .



r	t	x
---	---	---

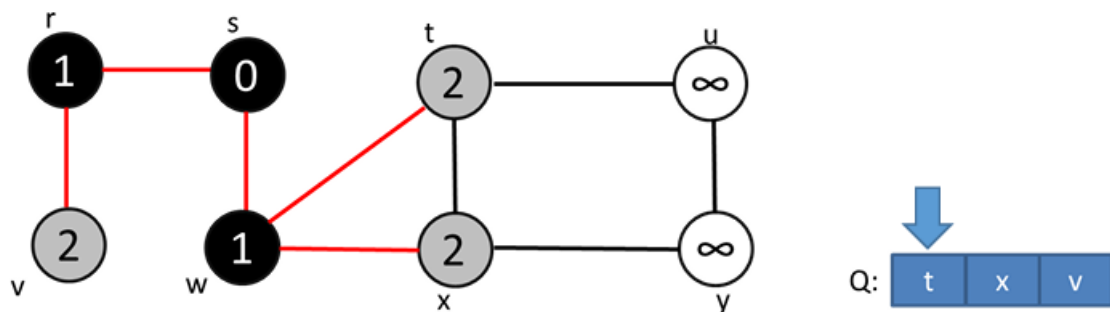
Os vértices t e x são coloridos de cinza e o vértice w é colorido de preto. As distâncias também são atualizadas.



O próximo elemento que deverá ser retirado da fila é r.

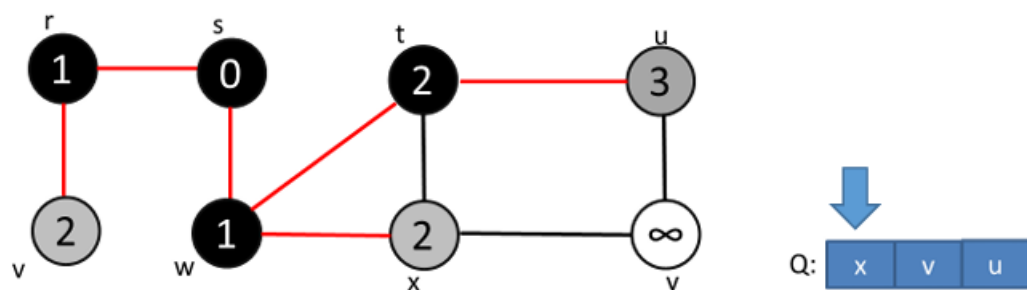
Os vizinhos de r são v. Assim, r é retirado da fila e insere-se na fila o vértice v. O nó r é pintado de preto e o nó v de cinza.

A distância também é atualizada.

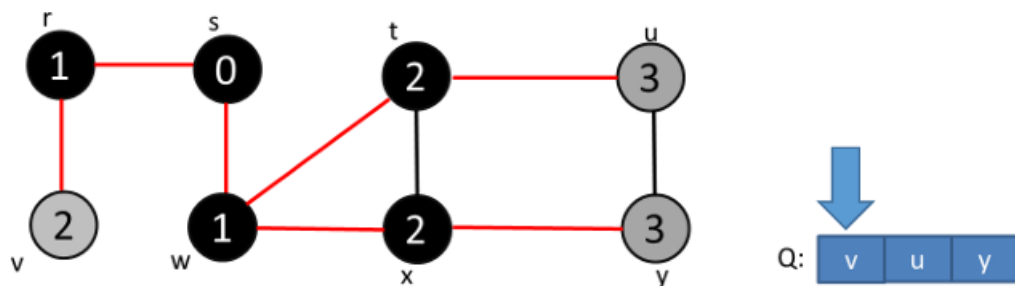


O próximo a ser retirado da fila é t. Os vizinhos de t são u e x, mas x já foi visitado. Observe, portanto, que somente se coloca na fila vértices brancos, que são imediatamente coloridos de cinza ao entrar na fila.

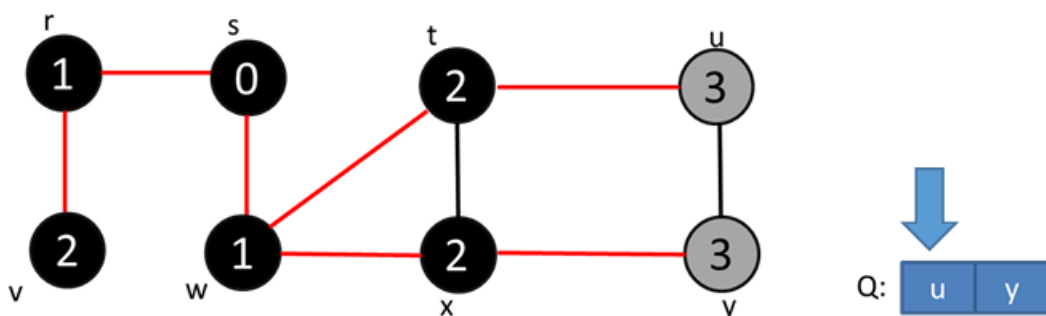
Assim, considera-se apenas o vértice u que é vizinho branco de t. Com isso, u entra na fila, é colorido de cinza, atualizam se as distâncias.



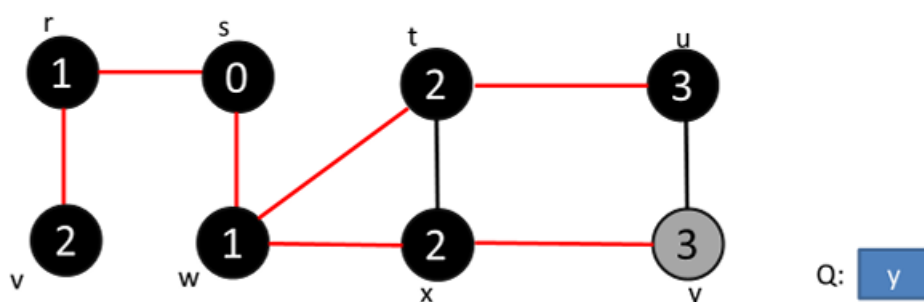
O próximo elemento que deverá ser retirado da fila é x. Os vizinhos de x são 1 e y. Mas 1 já foi visitado. Então, x é retirado da fila e pintado de preto. O vértice y entra fila e é pintado de cinza. As distâncias também são atualizadas.



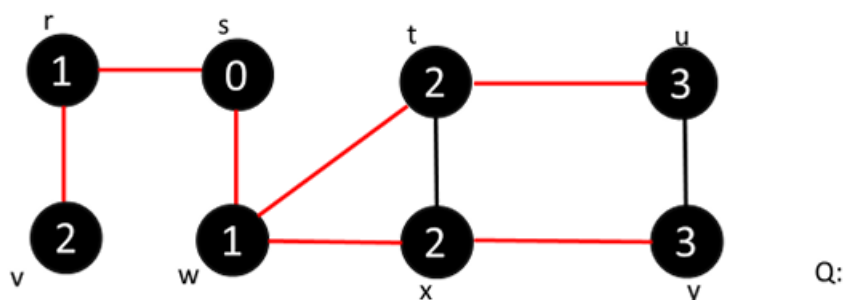
O próximo vértice que deverá ser retirado da fila é v. Os vizinhos de v são r. Mas, r já foi visitado. Portanto, apenas remove-se v da fila. O vértice v é pintado de preto.



O próximo que está na fila é u. Os vizinhos de u são t e y. Mas, t e y já foram visitados. Assim, u é retirado da fila e pintado de preto.



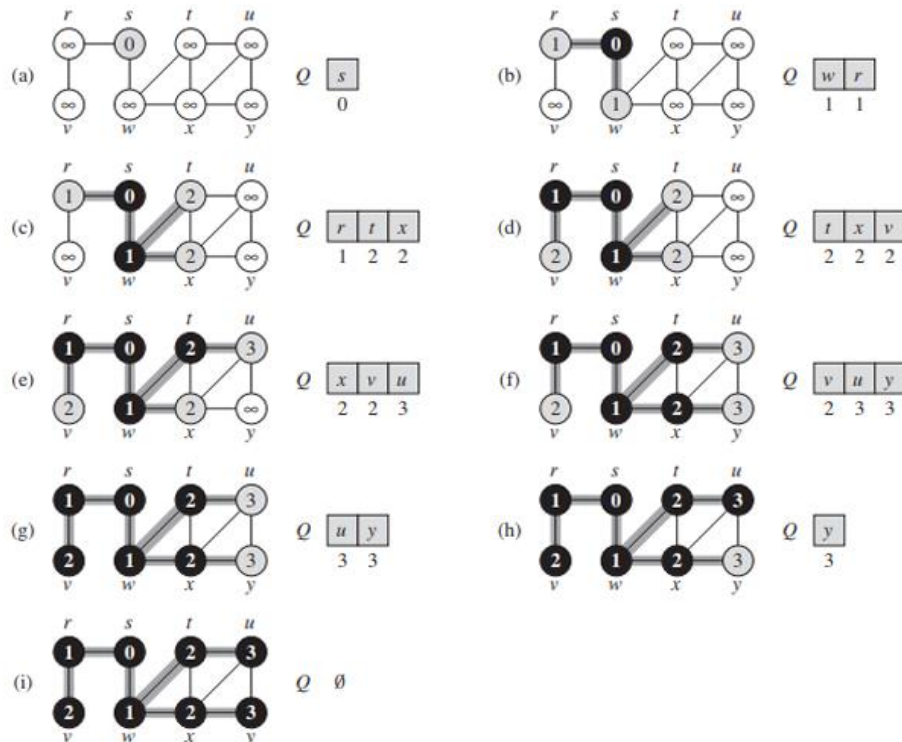
O próximo vértice a ser processado é y. Os vizinhos de y são x e u. Mas, x e u já foram visitados. Assim y é retirado da fila.



De acordo com Cormen, o pseudocódigo do algoritmo BFS é dado por:

```
BFS( $G, s$ )
1  for each vertex  $u \in G.V - \{s\}$ 
2       $u.color = \text{WHITE}$ 
3       $u.d = \infty$ 
4       $u.\pi = \text{NIL}$ 
5   $s.color = \text{GRAY}$ 
6   $s.d = 0$ 
7   $s.\pi = \text{NIL}$ 
8   $Q = \emptyset$ 
9  ENQUEUE( $Q, s$ )
10 while  $Q \neq \emptyset$ 
11      $u = \text{DEQUEUE}(Q)$ 
12     for each  $v \in G.Adj[u]$ 
13         if  $v.color == \text{WHITE}$ 
14              $v.color = \text{GRAY}$ 
15              $v.d = u.d + 1$ 
16              $v.\pi = u$ 
17             ENQUEUE( $Q, v$ )
18      $u.color = \text{BLACK}$ 
```

Ainda de acordo com Cormen, a figura abaixo ilustra o algoritmo BFS:

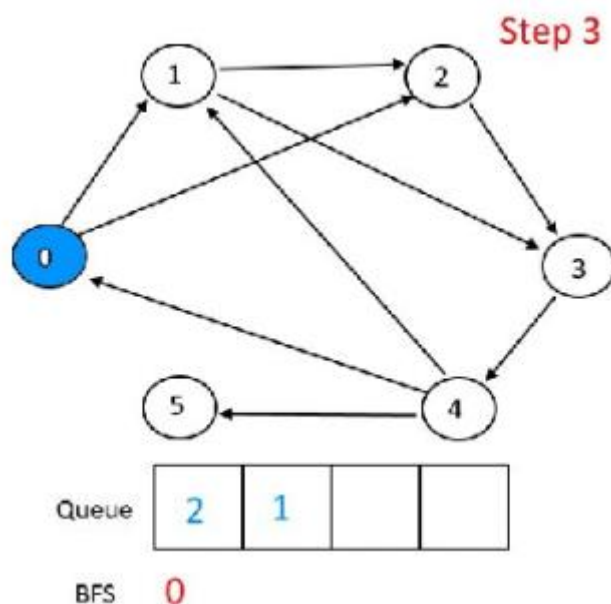
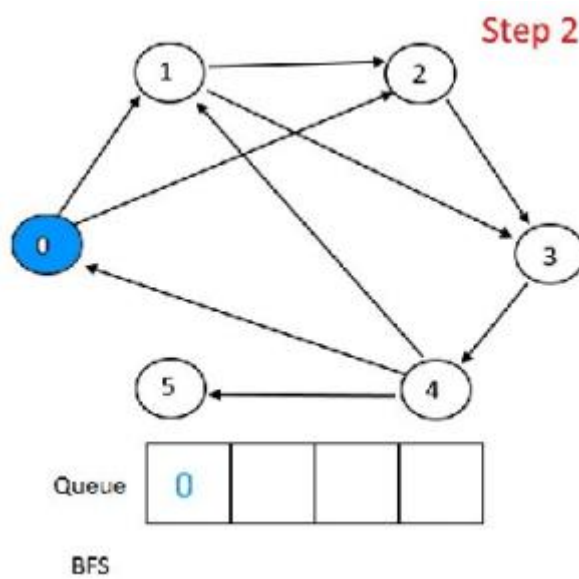
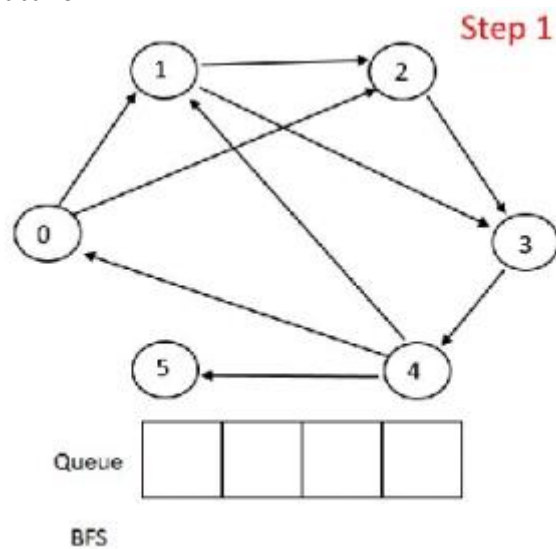


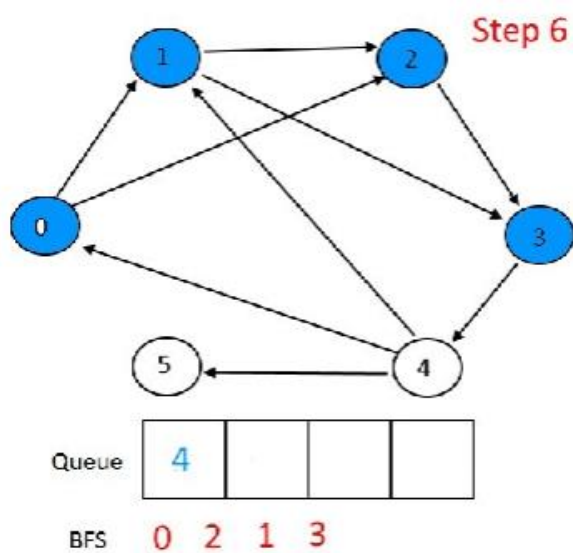
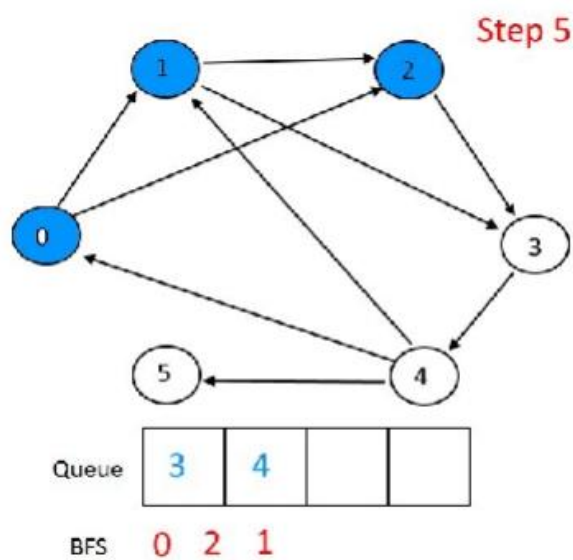
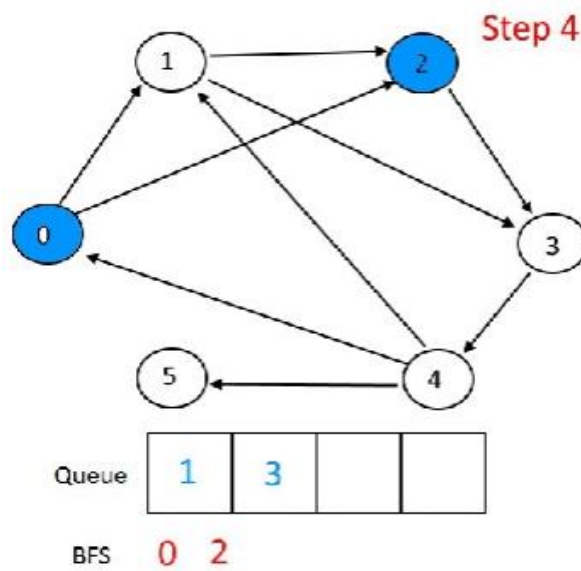
Busca em largura

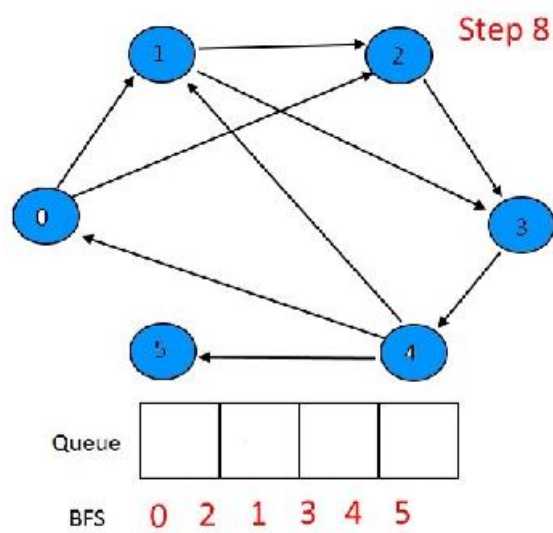
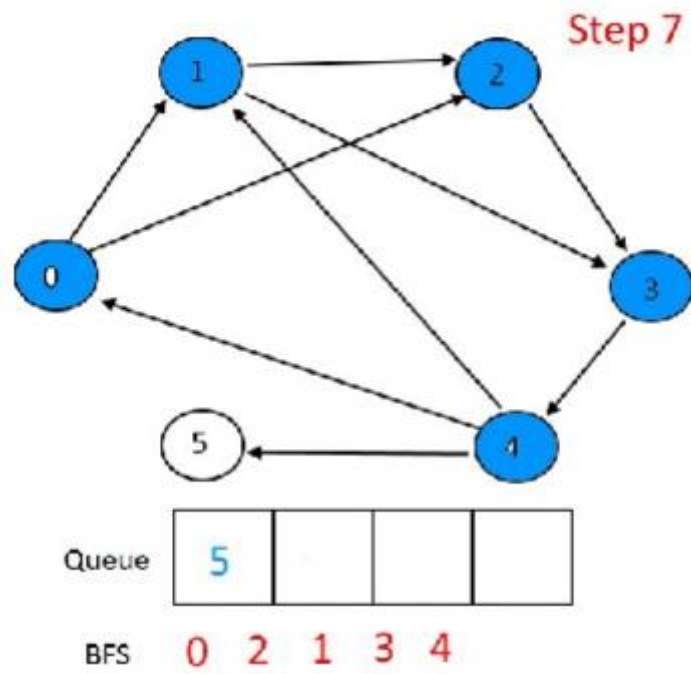
BFS(V, A, s)

1. for each u in $V - \{s\}$
 - ▷ para cada vértice u em V exceto s
 - 2. $color[u] \leftarrow WHITE$
 - ▷ no início todos os vértices são brancos
 - 3. $d[u] \leftarrow infinity$
 - 4. $\pi[u] \leftarrow NIL$
5. $color[s] \leftarrow GRAY$
 - ▷ Vértice origem descoberto
6. $d[s] \leftarrow 0$
7. $\pi[s] \leftarrow NIL$
8. $Q \leftarrow \{ \}$
9. ENQUEUE(Q, s)
 - ▷ Colocar o vértice origem na fila Q
10. while Q is non-empty
 - ▷ Enquanto existam vértices cinzas
 - 11. $u \leftarrow DEQUEUE(Q)$
 - ▷ i.e., $u = primeiro(Q)$
 - 12. for each v adjacent to u
 - ▷ para cada vértice adjacente a u
 - 13. if $color[v] = WHITE$
 - ▷ se é branco (ele ainda não foi descoberto)
 - 14. then $color[v] \leftarrow GRAY$
 - 15. $d[v] \leftarrow d[u] + 1$
 - 16. $\pi[v] \leftarrow u$
 - 17. ENQUEUE(Q, v)
 - 18. $color[u] \leftarrow BLACK$

Efetuar a análise do algoritmo Busca em Largura e, com o emprego de Listas de Adjacências, aplicar o algoritmo BFS para o grafo abaixo:







```

import java.util.LinkedList;
import java.util.Queue;

public class GraphBFS {
    public static void main(String args[]) {
        Graph g = new Graph(6);
        g.addEdge(0, 1);
        g.addEdge(0, 2);
        g.addEdge(1, 2);
        g.addEdge(1, 3);
        g.addEdge(3, 4);
        g.addEdge(2, 3);
        g.addEdge(4, 0);
        g.addEdge(4, 1);
        g.addEdge(4, 5);
        g.BFS(0);
    }
}

class Node {
    int dest;
    Node next;

    public Node(int d) {
        dest = d;
        next = null;
    }
}

class adjList {
    Node head;
}

class Graph {
    int V;
    adjList[] array;

    public Graph(int V) {
        this.V = V;
        array = new adjList[V]; // linked lists = number of Nodes
in Graph

        for (int i = 0; i < V; i++) {
            array[i] = new adjList();
            array[i].head = null;
        }
    }
}

```

```

public void addEdge(int src, int dest) {
    Node n = new Node(dest);
    n.next = array[src].head;
    array[src].head = n;
}

public void BFS(int startVertex) {
    boolean[] visited = new boolean[V];
    Queue<Integer> s = new LinkedList<Integer>();

    s.add(startVertex);
    while (s.isEmpty() == false) {
        int n = s.poll(); //
        System.out.print(" " + n);
        visited[n] = true;
        Node head = array[n].head;
        while (head != null) {
            if (visited[head.dest] == false) {
                s.add(head.dest);
                visited[head.dest] = true;
            }
            head = head.next;
        }
    }
}

```