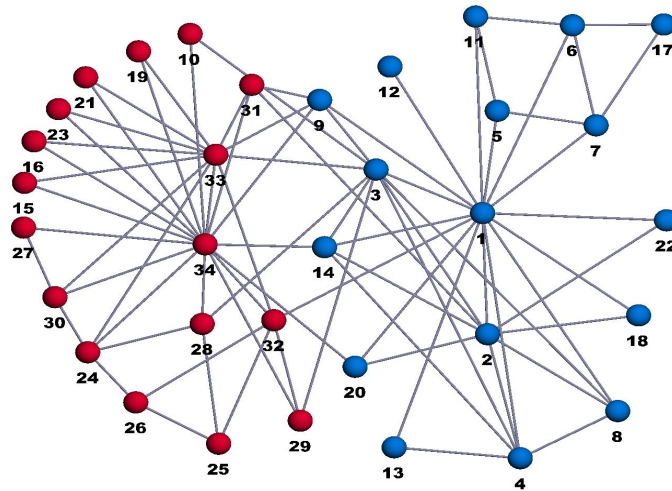


PROJETO 3: BUSCA EM LARGURA E PROFUNDIDADE



- Implementar os algoritmos BFS e DFS

Em grafos, um algoritmo de busca tem como objetivo, dado um vértice, buscá-lo passando pelos vértices que o “antecedem”. Há duas principais estratégias para realizar uma busca em um grafo. Busca em Largura e Busca em Profundidade.

Busca em Largura:

A busca em largura começa por um vértice s . O algoritmo visita s , depois visita todos os vizinhos de s , depois todos os vizinhos de todos os “filhos” de s , e assim por diante. Para implementar essa ideia, o algoritmo usa uma fila *de vértices*. No começo de cada iteração, a fila contém os vértices que já foram visitados mas têm vizinhos ainda não visitados. No começo da primeira iteração, a fila contém o vértice s apenas. Pode-se dizer que a BFS realiza a busca em níveis.

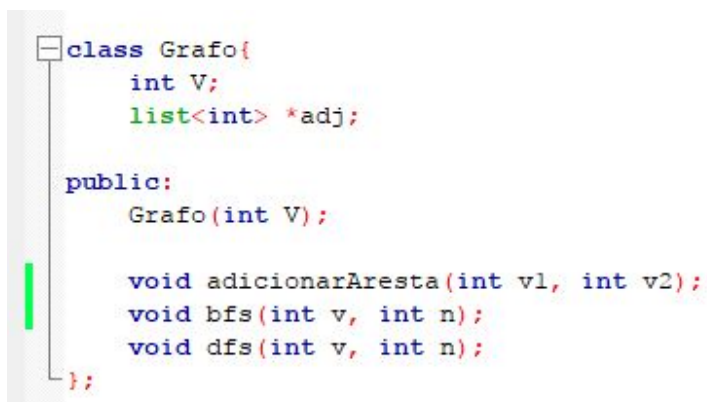
Pseudocódigo:

```
enquanto a fila não estiver vazia faça
    V é o primeiro vértice da fila
    retire V da fila
    para cada vizinho W de V que ainda não foi visitado
        visite w e coloque-o na fila
```

A busca em largura a partir de um vértice s constrói uma árvore com raiz s : cada aresta $v-w$ percorrido até um vértice w não visitado é acrescentado à árvore. Essa árvore é conhecida como árvore de busca em largura = *BFS tree*). Podemos representar essa árvore explicitamente por um vetor.

Implementação:

Antes de iniciar a explicação é importante definir o grafo, que no caso deste projeto foi implementado como uma classe.



```
class Grafo{
    int V;
    list<int> *adj;

public:
    Grafo(int V);

    void adicionarAresta(int v1, int v2);
    void bfs(int v, int n);
    void dfs(int v, int n);
};
```

(Imagem - Classe Grafo)

A classe possui como atributos o número de arestas e a lista de adjacência de cada vértice.

Os métodos implementados são, além do seu construtor:

- adicionarAresta: é passado como parâmetro 2 vértices. Cada um é inserido na lista de adjacência do outro
- bfs: é realizada a busca em largura partindo de um vértice V , buscando um vértice n
- dfs: é realizada a busca em profundidade partindo de um vértice V , buscando um vértice n

Para a busca em largura, primeiramente é criado uma fila. Para isso foi utilizada a biblioteca “queue” que facilita a manipulação desta estrutura.

Para cada vértice do grafo é atribuído “falso”, para a variável “visitados”. Isto significa que todos os vértices são inicializados como não-visitados.

Após isso, o vértice raiz é marcado como visitado.

Enquanto o algoritmo não encontrou o vértice em questão ele repete o seguinte processo:

Para cada vértice vizinho (que está na lista de adjacência), se este vértice não foi visitado, ele é marcado como visitado e mostrado na tela, a fim de representar um elemento da árvore. Também é feita a comparação para saber se este é o vértice buscado. Se for, ele para o loop da busca, e informa que o vértice foi encontrado. Se o vértice não foi encontrado ainda, ele adiciona este vizinho na fila.

Quando todos os vizinhos forem visitados e verificados, ele remove o primeiro elemento da fila, realizando uma nova busca pelos vizinhos do elemento seguinte (que agora é o primeiro elemento) da fila.

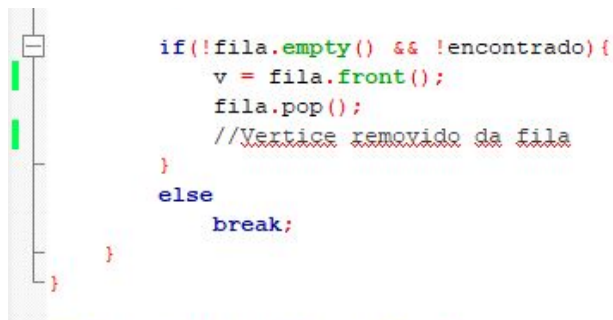
A árvore é mostrada conforme o programa é executado.

```
void Grafo::bfs(int v, int n){
    queue<int> fila;
    bool visitados[V];
    bool encontrado = false;

    for(int i = 0; i < V; i++){
        visitados[i] = false;
    }

    visitados[v] = true;
    cout << "Vertice " << v << " visitado." << endl;

    while(!encontrado){
        list<int>::iterator it;
        for(it = adj[v].begin(); it != adj[v].end(); it++){
            if(!visitados[*it]){
                visitados[*it] = true;
                cout << "Vertice " << *it << " visitado." << endl;
                if(*it == n){
                    cout << "VERTICE " << *it << " ENCONTRADO!!!\n";
                    encontrado = true;
                }
            }
            if(!encontrado){
                fila.push(*it);
                //Vertice inserido Na fila
            }
        }
    }
}
```



(Imagem - Implementação da BFS em C++)

Busca em profundidade:

A busca em profundidade visita todos os vértices de um mesmo “ramo”. Após chegar ao final desta ramificação ele explora as ramificações mais próximas. Este algoritmo também é aplicado na exploração de labirintos, visto que deve-se chegar ao limite de cada caminho, e retornando a bifurcações anteriores a fim de explorá-las.

Em grafos o algoritmo pode ser implementado com a estrutura de pilha. A medida que um caminho é criado, o algoritmo insere os vértices na pilha. Quando todos os vizinhos do vértice em questão já foram visitados ou não existe mais vizinhos(alcançou o limite), é desempilhado o vértice, como se estivesse retornando do caminho feito. Desta forma, também é criada uma árvore de busca (DFS- Tree), que contém os caminhos realizados. O algoritmo também pode ser feito usando recursividade.

Pseudocódigo:

DFS(G,v):

 sendo S é uma pilha

 Empilha V

 Enquanto S não está vazia

 desempilha elemento em v

 Se v não foi visitado

 v é marcado como visitado

 para cada aresta v-w na lista de adjacencia de G faça

 empilha w

Implementação:

Para esta busca é criada uma pilha e uma variável de marcação (visitados). Novamente todos os vértices são inicializados como não visitados.

Enquanto o vértice não foi encontrado, o algoritmo verifica se o vértice v (inicialmente a raiz), já foi visitado. Caso não tenha sido visitado, ele é marcado como visitado, e mostrado na tela com a finalidade de representar a árvore. Quando ele é marcado como visitado, o vértice é empilhado.

É feita então a seguinte verificação com todos os seus vizinhos: se ele for um vizinho que não foi visitado, este vizinho se tornará o próximo vértice a ser feita as verificações anteriores. Além disso é verificado se este é o vértice que está sendo buscado.

É feito esse processo repetidas vezes até que não exista mais vizinhos para visitar (chegou na folha). Quando isto ocorre, é desempilhado, fazendo a verificação se todos os vértices vizinhos do "nó pai" (pai do nó que foi desempilhado) foi visitado. O algoritmo se encerra quando a pilha está vazia, ou seja, todos os caminhos foram realizados.

```
void Grafo::dfs(int v, int n){
    int x;
    stack<int> pilha;
    bool visitados[V];

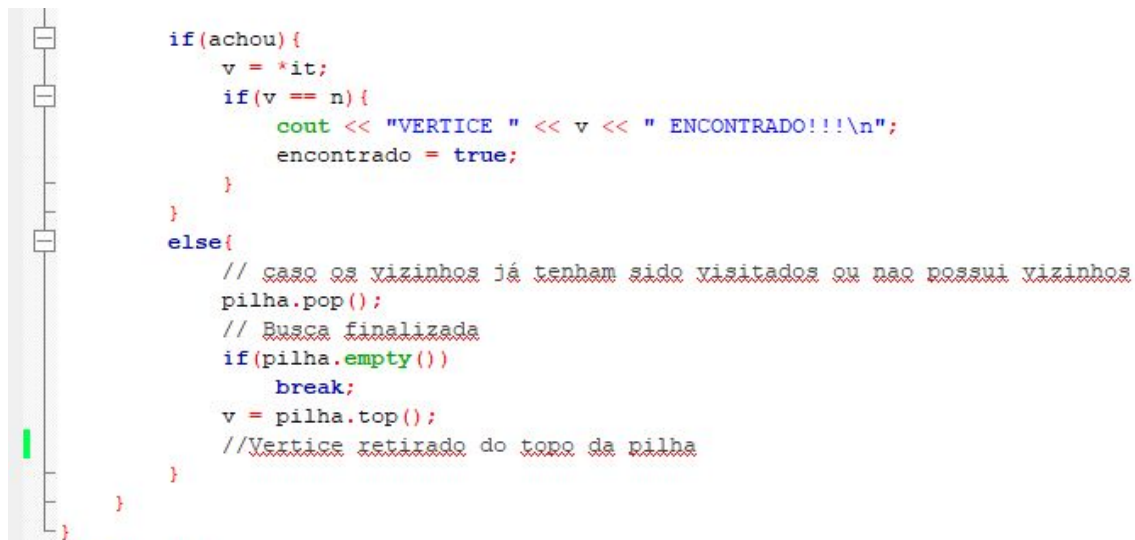
    for(int i = 0; i < V; i++)
        visitados[i] = false;

    bool encontrado = false;

    while(!encontrado){
        if(!visitados[v]){
            visitados[v] = true;
            cout << "Vertice " << v << " visitado\n";
            pilha.push(v);
            //Vertice inserido na Pilha
        }

        bool achou = false;
        list<int>::iterator it;

        for(it = adj[v].begin(); it != adj[v].end(); it++){
            if(!visitados[*it]){
                achou = true;
                break;
            }
        }
    }
}
```



(Imagem 2 - Implementação da DFS em C++)

As arestas dos dois datasets foram adicionadas pelo método “adicionarAresta”, da classe grafos. Desta forma é possível realizar a busca, partindo de um vértice raiz. O vértice da busca em questão foram selecionados aleatoriamente. No exemplo abaixo é buscado para o dataset “Karate”, o vértice 34 e para o dataset “Dolphins”, o vértice 62. O resultado é cada vértice sendo visitado, obtendo então uma árvore de busca. A árvore é apenas mostrada, mas pode ser armazenada em um vetor, por exemplo.

Buscando vertice 34 partindo de 1

BFS Karate:

Vertice 1 visitado.
Vertice 2 visitado.
Vertice 3 visitado.
Vertice 4 visitado.
Vertice 5 visitado.
Vertice 6 visitado.
Vertice 7 visitado.
Vertice 8 visitado.
Vertice 9 visitado.
Vertice 11 visitado.
Vertice 12 visitado.
Vertice 13 visitado.
Vertice 14 visitado.
Vertice 18 visitado.
Vertice 20 visitado.
Vertice 22 visitado.
Vertice 32 visitado.
Vertice 31 visitado.
Vertice 10 visitado.
Vertice 28 visitado.
Vertice 29 visitado.
Vertice 33 visitado.
Vertice 17 visitado.
Vertice 34 visitado.
VERTICE 34 ENCONTRADO!!!

DFS Karate:

Vertice 1 visitado
Vertice 2 visitado
Vertice 3 visitado
Vertice 4 visitado
Vertice 8 visitado
Vertice 13 visitado
Vertice 14 visitado
VERTICE 34 ENCONTRADO!!!

(Resultado das buscas para o “karate”)

```
Buscando vertice 62 partindo de 1
```

```
BFS Dolphins:
```

```
Vertice 1 visitado.  
Vertice 11 visitado.  
Vertice 15 visitado.  
Vertice 16 visitado.  
Vertice 41 visitado.  
Vertice 43 visitado.  
Vertice 48 visitado.  
Vertice 30 visitado.  
Vertice 17 visitado.  
Vertice 25 visitado.  
Vertice 34 visitado.  
Vertice 35 visitado.  
Vertice 38 visitado.  
Vertice 39 visitado.  
Vertice 44 visitado.  
Vertice 51 visitado.  
Vertice 53 visitado.  
Vertice 19 visitado.  
Vertice 46 visitado.  
Vertice 56 visitado.  
Vertice 60 visitado.  
Vertice 36 visitado.  
Vertice 52 visitado.  
Vertice 21 visitado.  
Vertice 45 visitado.  
Vertice 50 visitado.  
Vertice 62 visitado.  
VERTICE 62 ENCONTRADO!!!
```

```
DFS Dolphins:
```

```
Vertice 1 visitado  
Vertice 11 visitado  
Vertice 30 visitado  
Vertice 36 visitado  
Vertice 44 visitado  
Vertice 47 visitado  
Vertice 50 visitado  
Vertice 54 visitado  
VERTICE 62 ENCONTRADO!!!
```

Datasets:

<http://networkdata.ics.uci.edu/data/karate/karate.paj>

<http://networkdata.ics.uci.edu/data/dolphins/dolphins.paj>