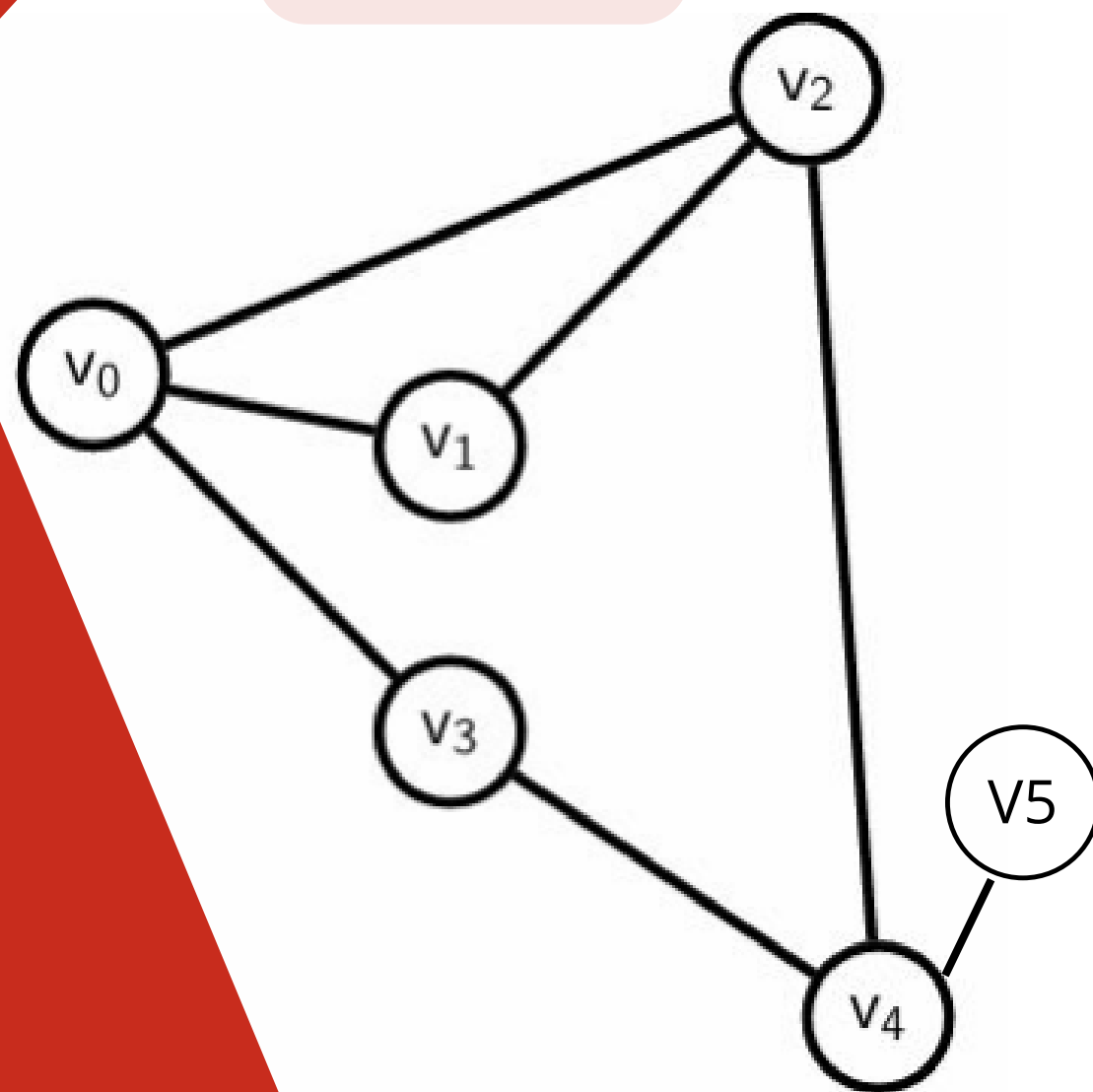


DFS em C++ com lista de adjacência

Arthur de Almeida Brandino

GRAFO EXEMPLO

Visual



Lista de Arestas

0-1

1-2

0-2

2-4

0-3

3-4

4-5

Inserção no Código

```
inserirAresta(grafo, 0, 1, 0);  
inserirAresta(grafo, 1, 0, 0);  
  
inserirAresta(grafo, 0, 2, 0);  
inserirAresta(grafo, 2, 0, 0);  
  
inserirAresta(grafo, 0, 3, 0);  
inserirAresta(grafo, 3, 0, 0);  
  
inserirAresta(grafo, 1, 2, 0);  
inserirAresta(grafo, 2, 1, 0);  
  
inserirAresta(grafo, 2, 4, 0);  
inserirAresta(grafo, 4, 2, 0);  
  
inserirAresta(grafo, 3, 4, 0);  
inserirAresta(grafo, 4, 3, 0);  
  
inserirAresta(grafo, 4, 5, 0);  
inserirAresta(grafo, 5, 4, 0);
```

EXPLICANDO O CÓDIGO

Chamada da Busca na Main()

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
void buscaProfundidade(No** grafo, int origem, int destino, int numV)
```

Passamos:

- O Grafo = grafo
- A Origem = 0
- O Destino = 4
- O Número De Vértices = numV

```
{  
    bool* visitados = new bool[numV];  
    bool existe = false;
```

Variável booleana para verificar se já
passamos por tal vértice
Ela é uma matriz de tamanho numV

Variável booleana para vermos
se já achamos o Caminho

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

Aqui estamos passando em toda matriz
marcando que não visitamos nenhum
vértice ainda

EXPLICANDO O CÓDIGO

```
bool existe = buscaProfundidadeVisitas(grafo, origem, destino, visitados);
```

→ Iniciamos a varredura da origem

```
for (int i = 0; i < numV; i++)  
{  
    if (!visitados[i] && !existe)  
    {  
        existe = buscaProfundidadeVisitas(grafo, i, destino, visitados);  
    }  
}
```

→ Loop para passarmos em todos vértices

→ Aqui verificamos se o Vértice já foi **VISITADO E** se o caminho já **EXISTE**

→ Atribuo a variável **EXISTE** a resposta que a subrotina **buscaProfundidadeVisitas** irá nos dar

```
    if (!existe)  
        cout << "Caminho Não Existe!";  
}
```

→ Retorno visual caso o caminho não seja encontrado

EXPLICANDO O CÓDIGO

```
existe = buscaProfundidadeVisitas(grafo, i, destino, visitados)
```

```
bool buscaProfundidadeVisitas(No** grafo, int verticeAtual, int destino, bool* visitados)
```

Passamos:

- Existe = True/False
- Grafo = Grafo
- i = verticeAtual
- O Destino = destino
- Visitados = visitados

```
if (verticeAtual == destino) {  
    cout << "Acho: " << destino << endl;  
    return true;  
}
```

Verificamos se o vértice vizinho é o que estamos procurando que

Se for, damos um retorno visual mostrando que o caminho foi achado

E retornamos pra variável **existe** que o caminho existe = **True**

```
visitados[verticeAtual] = true;  
No* cabecaFilaVizinho = grafo[verticeAtual];
```

Marcamos que Visitamos o Vértice

Variável para verificarmos se o vértice que acabamos de visitar tem vizinhos

EXPLICANDO O CÓDIGO

```
while(cabecaFilaVizinho != nullptr)
{
```

Loop caso o vértice tenha vizinhos

```
    int vizinho = cabecaFilaVizinho->vertice;
    if (!visitados[vizinho]) {
        if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
            return true;
```

Armazenamos o número do próximo vértice que iremos visitar

Verificamos se o vértice vizinho já foi visitado

Se ele não foi visitado ainda iniciamos uma recursão para visitá-lo

Se a recursão der retorno **true** a sub-rotina tbm retorna **true**

```
cabecaFilaVizinho = cabecaFilaVizinho->proximo;
```

Se não passamos para o vizinho

```
    return false;
}
```

Se sairmos do **while** e o **destino** não for encontrado retornamos **existe = false**

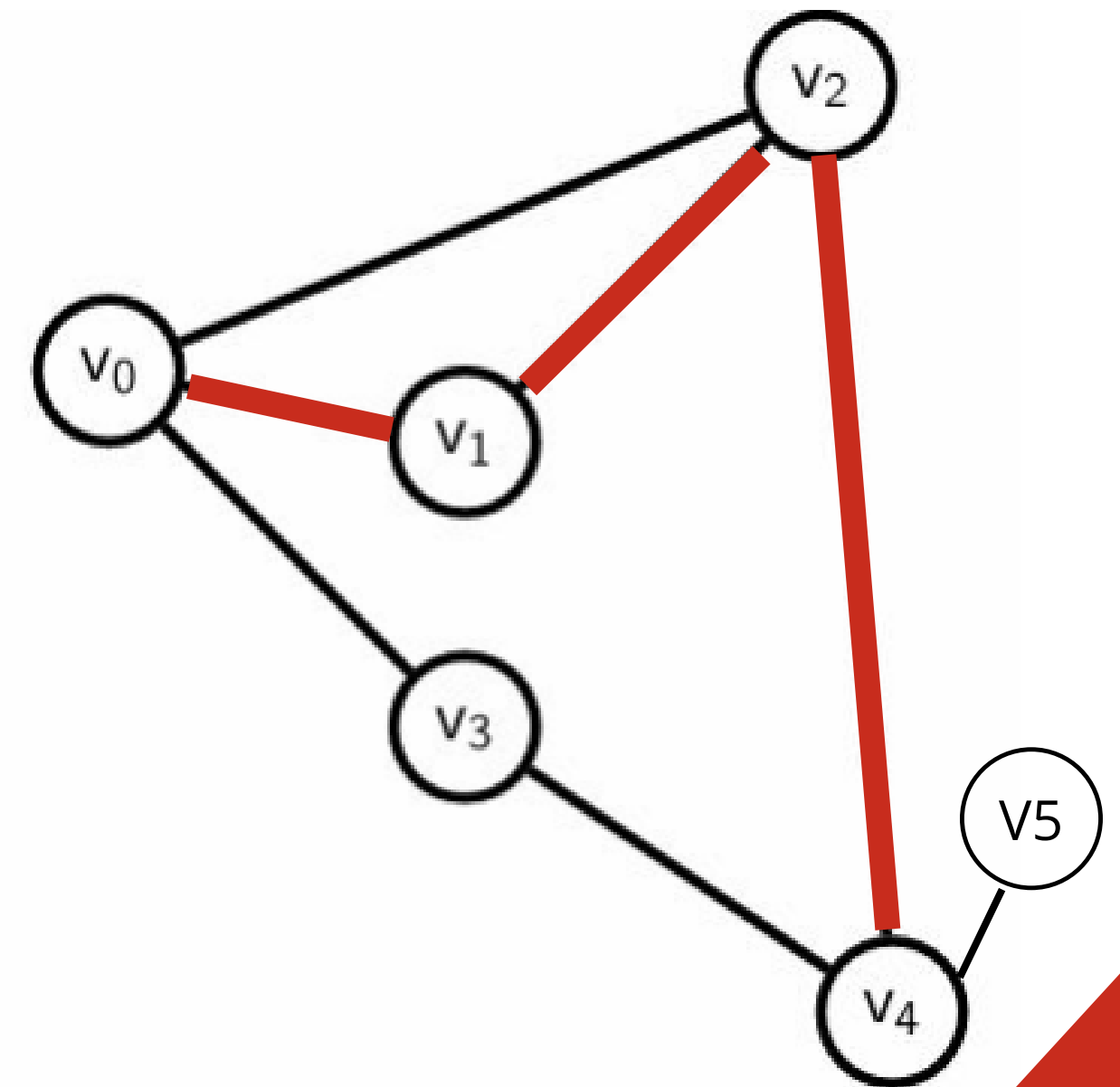
TESTE

Busca de um Caminho Existente

Chamada da Função de Busca na Main()

```
buscaProfundidade(grafo, 0, 4, numV);
```

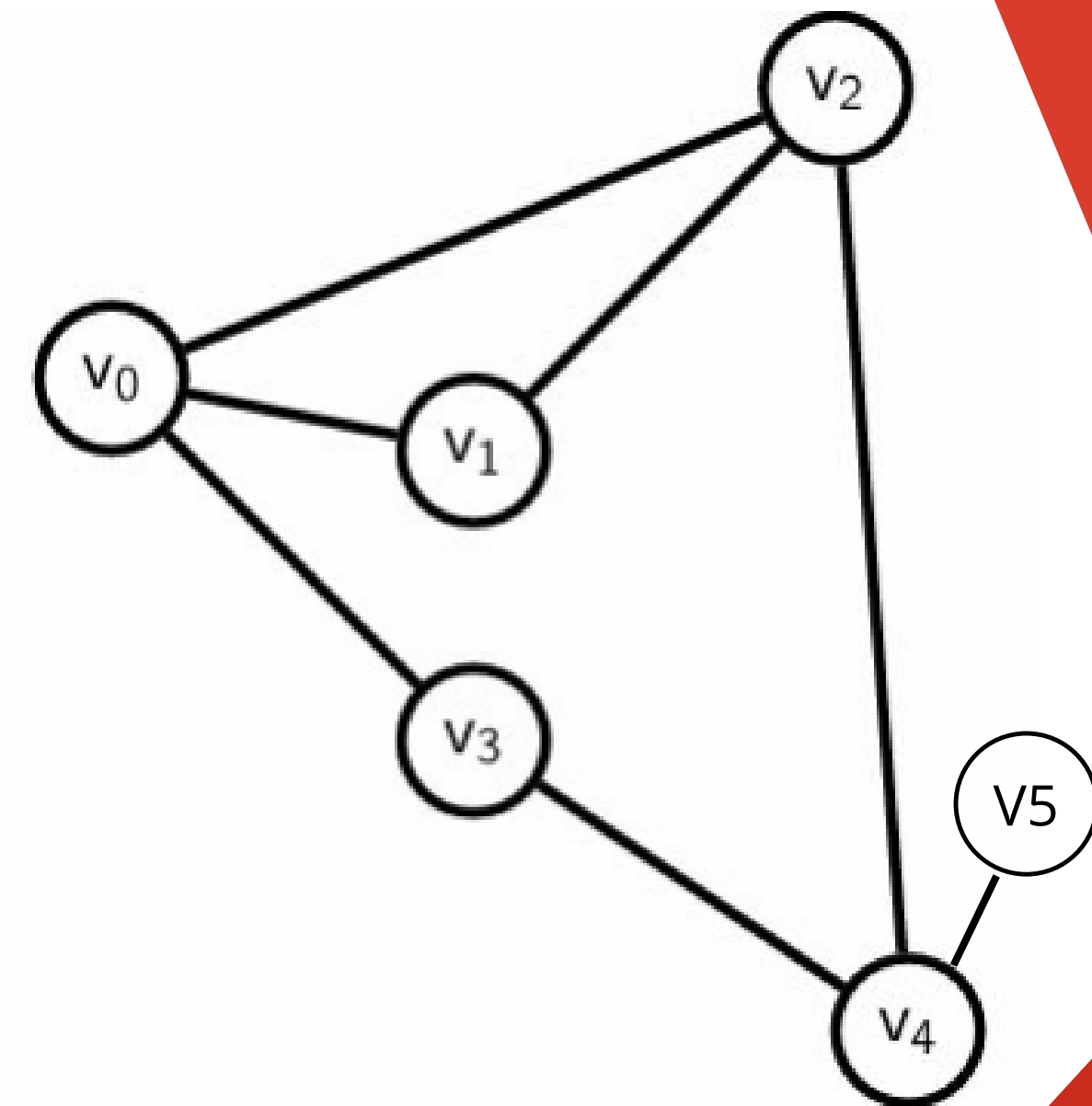
Caminho que será encontrado



TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
void buscaProfundidade(No** grafo, int origem, int destino, int numV)
{
    bool* visitados = new bool[numV];
    bool existe = false;
    for (int i = 0; i < numV; i++)
        visitados[i] = false;
    bool existe = buscaProfundidadeVisitas(grafo, origem, destino, visitados);
}
```



TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
bool existe = buscaProfundidadeVisitas(grafo, origem, destino, visitados);
```

```
bool buscaProfundidadeVisitas(No** grafo, int verticeAtual, int destino, bool* visitados)
```

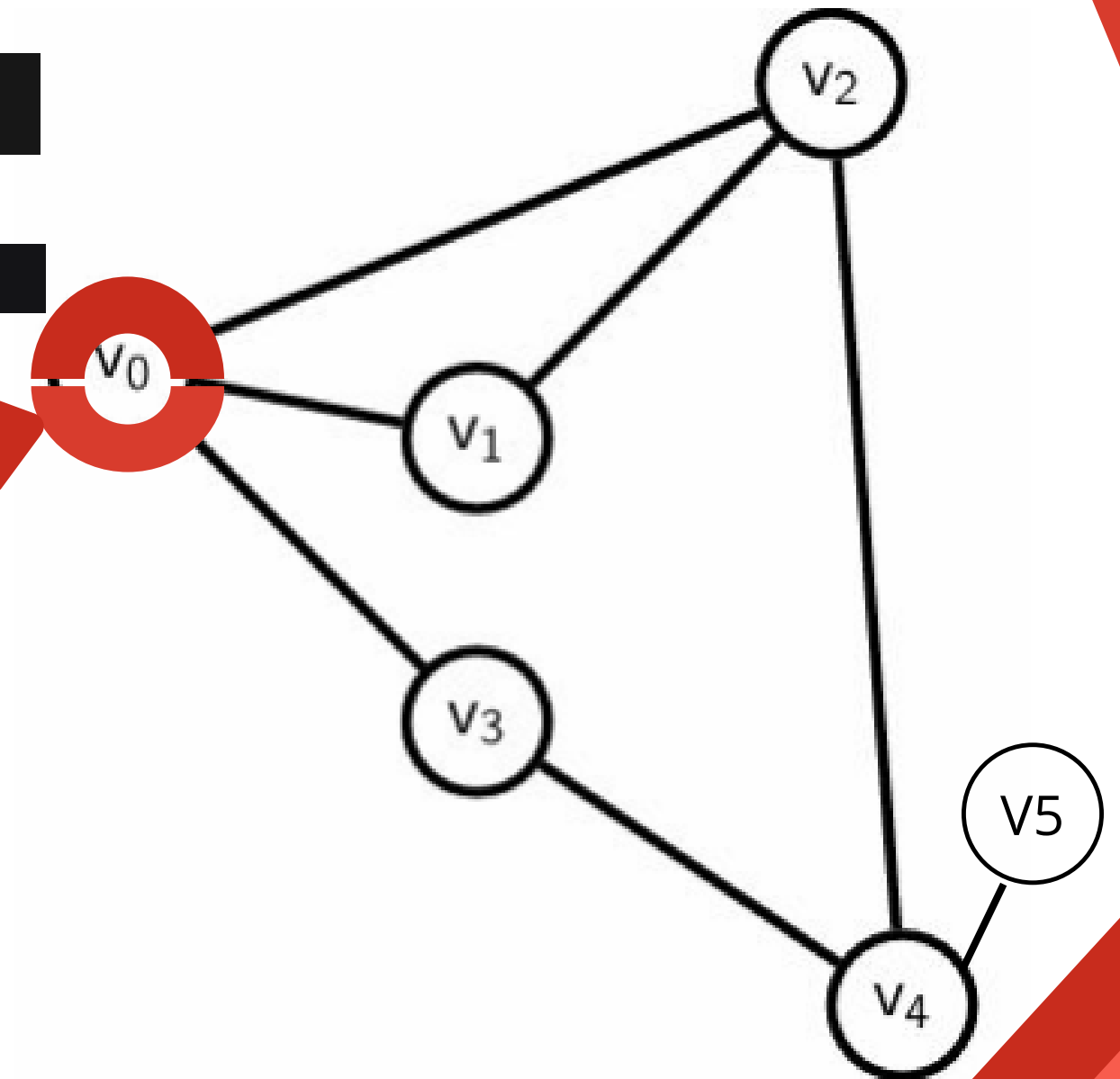
0,

4,

0,

4,

```
if (verticeAtual == destino) {  
    cout << "Achou: " << destino << endl;  
    return true;  
}  
  
visitados[verticeAtual] = true;  
  
No* cabecaFilaVizinho = grafo[verticeAtual];
```

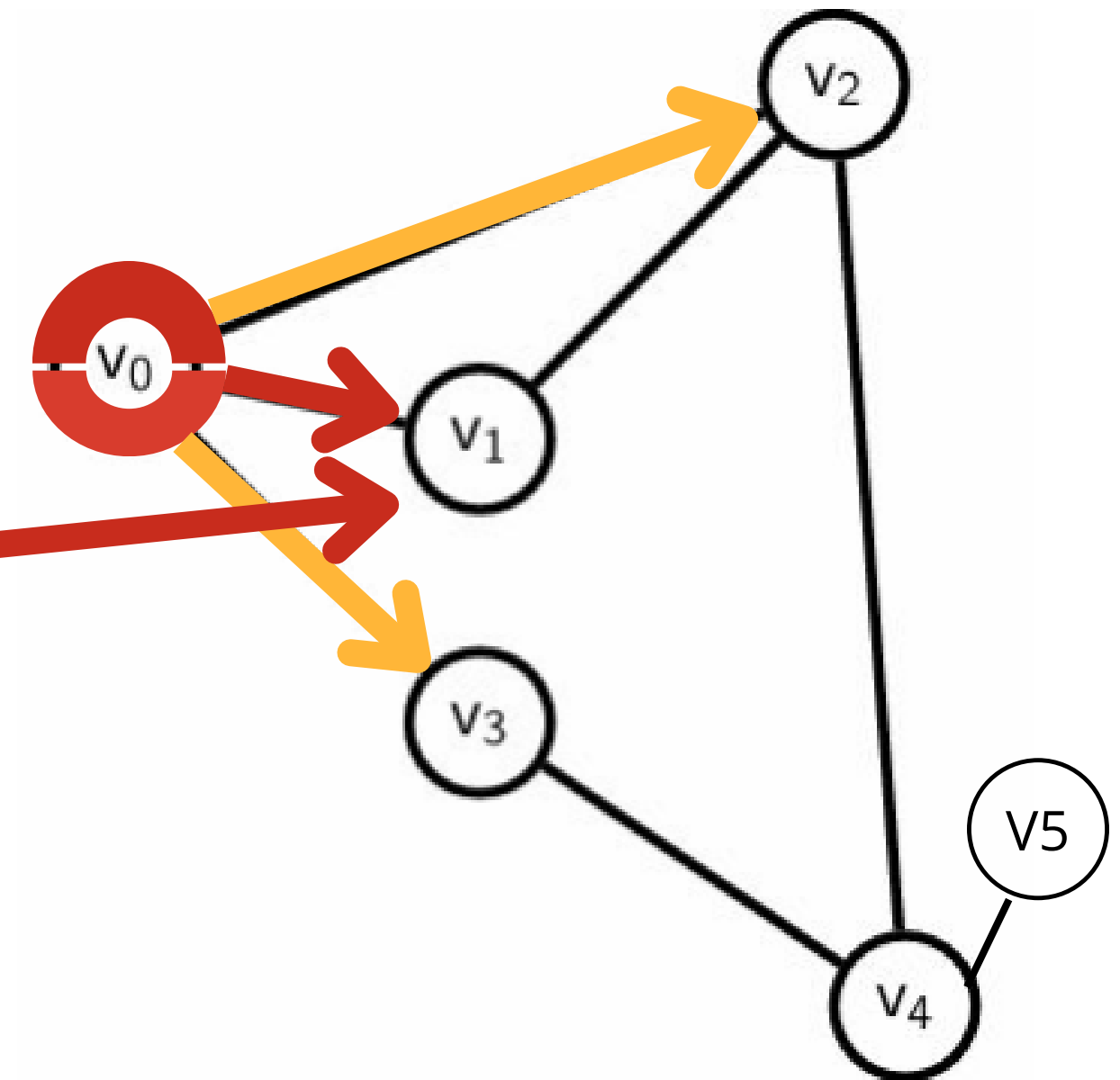


TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
while(cabecaFilaVizinho != nullptr)  
{
```

```
int vizinho = cabecaFilaVizinho->vertice;  
if (!visitados[vizinho]) {  
    if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
```



TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
```

```
bool buscaProfundidadeVisitas(No** grafo, int verticeAtual, int destino, bool* visitados)
```

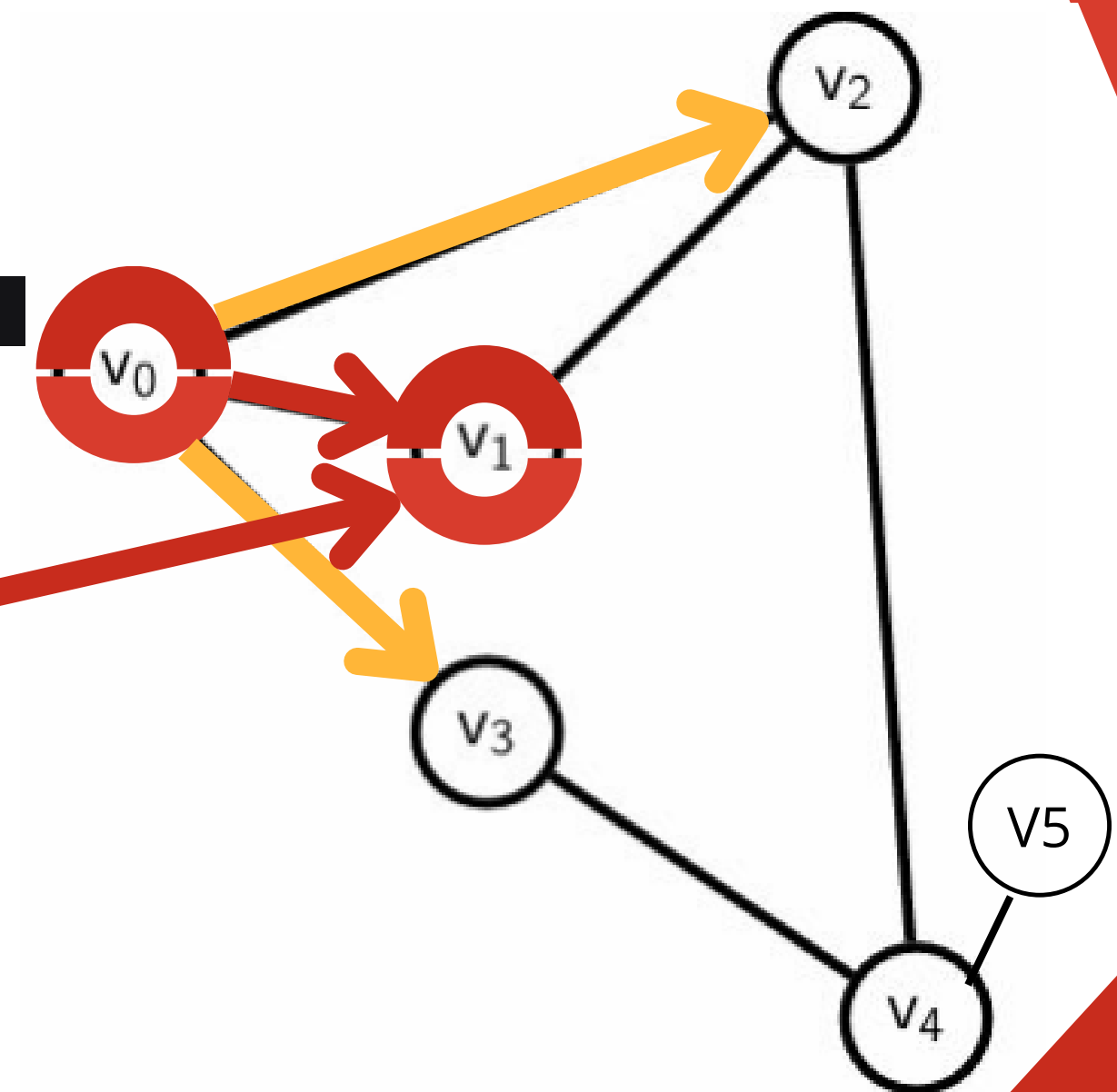
1,

4,

1,

4,

```
if (verticeAtual == destino) {  
    cout << "Achou: " << destino << endl;  
    return true;  
}  
  
visitados[verticeAtual] = true;  
  
No* cabecaFilaVizinho = grafo[verticeAtual];
```



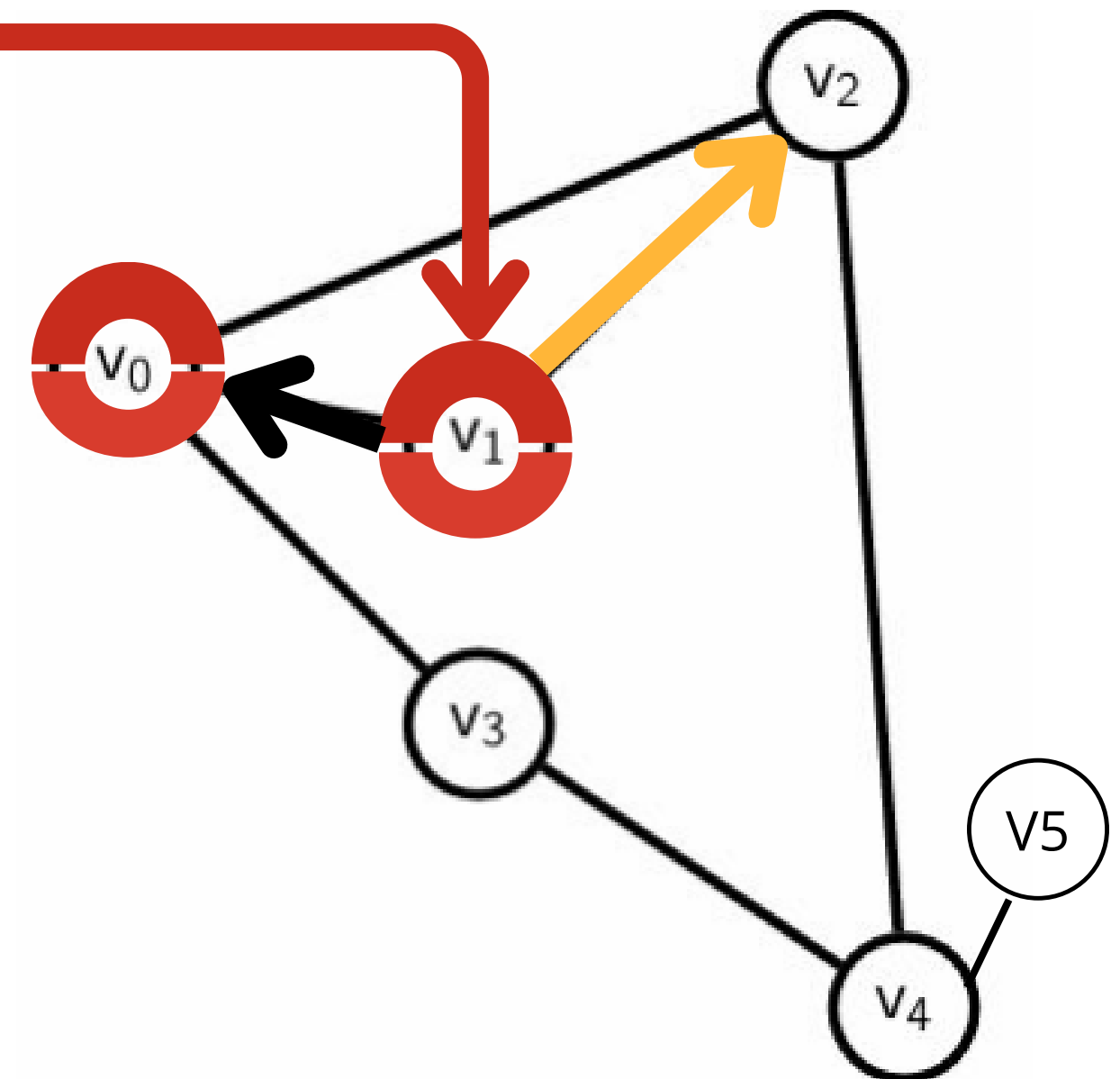
TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
while(cabecaFilaVizinho != nullptr)
{
```

```
int vizinho = cabecaFilaVizinho->vertice;
if (!visitados[vizinho]) {
    if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
        return true;
```

```
cabecaFilaVizinho = cabecaFilaVizinho->proximo;
```



TESTE

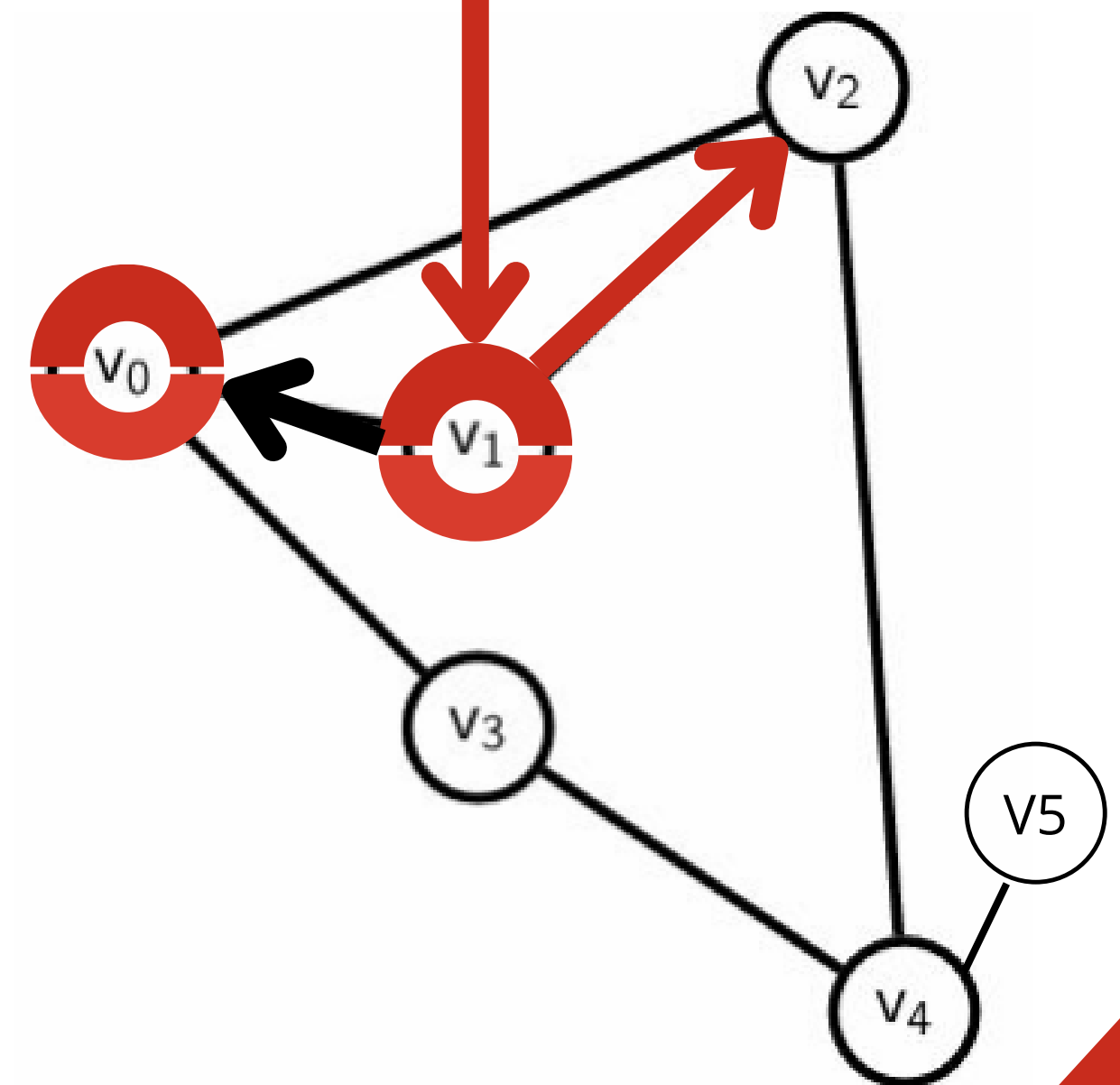
```
buscaProfundidade(grafo, 0, 4, numV);
```

```
while(cabecaFilaVizinho != nullptr)  
{
```

```
    2,
```

```
    2,
```

```
    int vizinho = cabecaFilaVizinho->vertice;  
    if (!visitados[vizinho]) {  
        if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
```



TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

```
if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
```

```
bool buscaProfundidadeVisitas(No** grafo, int verticeAtual, int destino, bool* visitados)
```

2,

4,

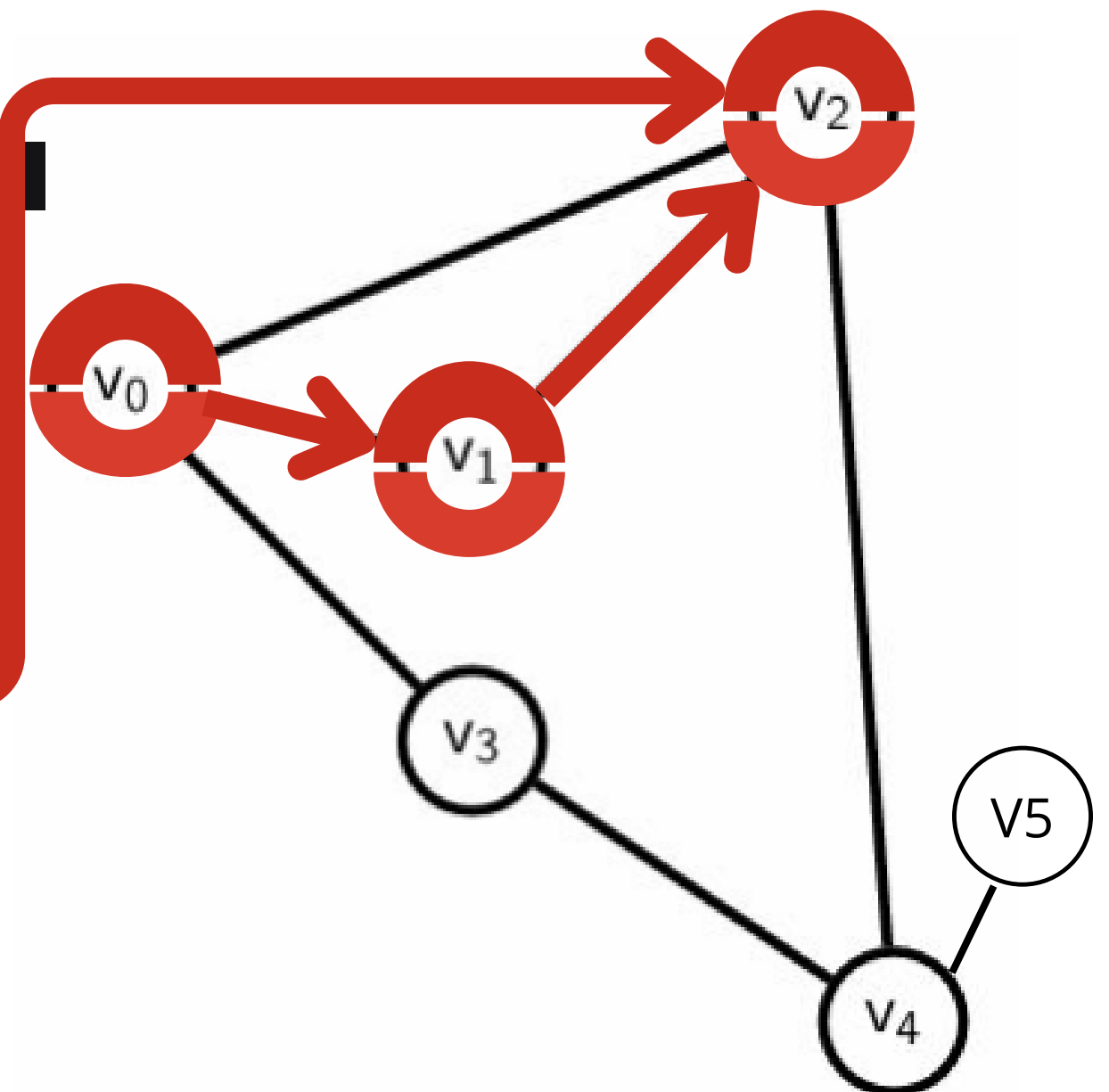
2,

4,

```
if (verticeAtual == destino) {  
    cout << "Acho: " << destino << endl;  
    return true;  
}
```

```
visitados[verticeAtual] = true;
```

```
No* cabecaFilaVizinho = grafo[verticeAtual];
```



TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

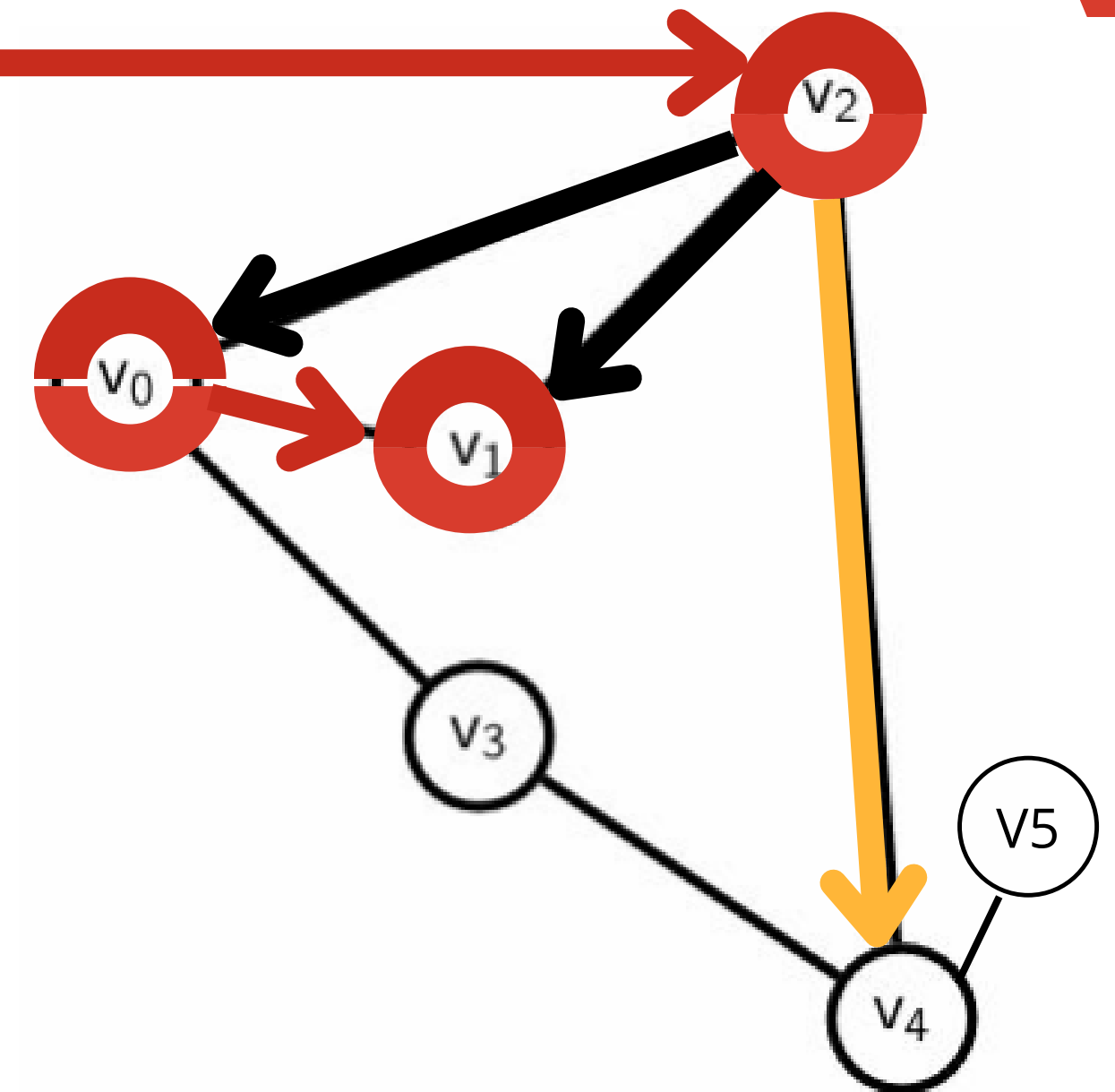
```
while(cabecaFilaVizinho != nullptr)
{
```

```
    int vizinho = cabecaFilaVizinho->vertice;
    if (!visitados[vizinho]) {
        if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
            return true;
```

```
        cabecaFilaVizinho = cabecaFilaVizinho->proximo;
```

```
    int vizinho = cabecaFilaVizinho->vertice;
    if (!visitados[vizinho]) {
        if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
            return true;
```

```
        cabecaFilaVizinho = cabecaFilaVizinho->proximo;
```



TESTE

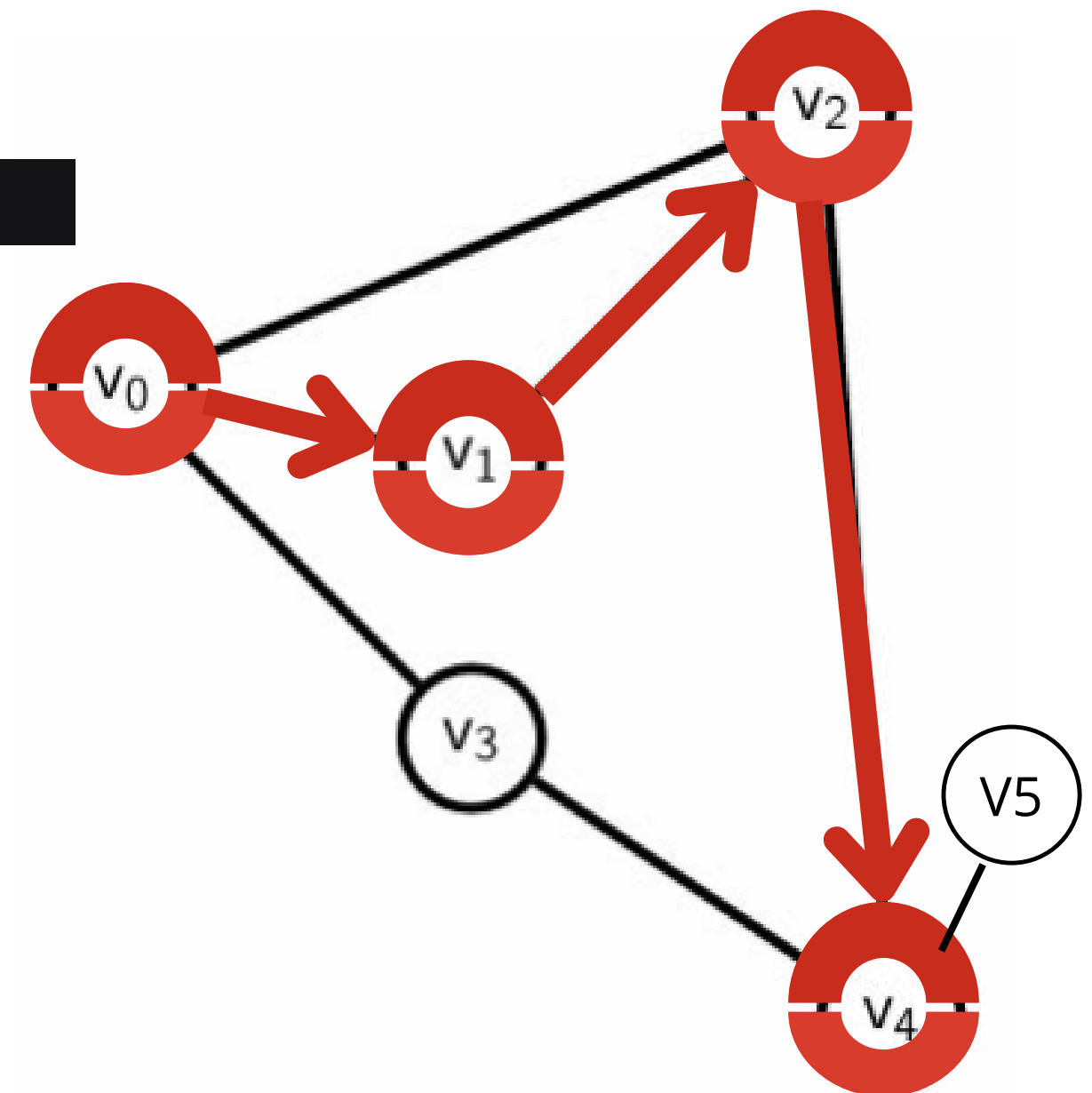
```
buscaProfundidade(grafo, 0, 4, numV);
```

```
if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))
```

```
bool buscaProfundidadeVisitas(No** grafo, int verticeAtual, int destino, bool* v
```

```
if (verticeAtual == destino) {  
    cout << "Acho: " << destino << endl;  
    return true;  
}
```

```
if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))  
    return true;
```

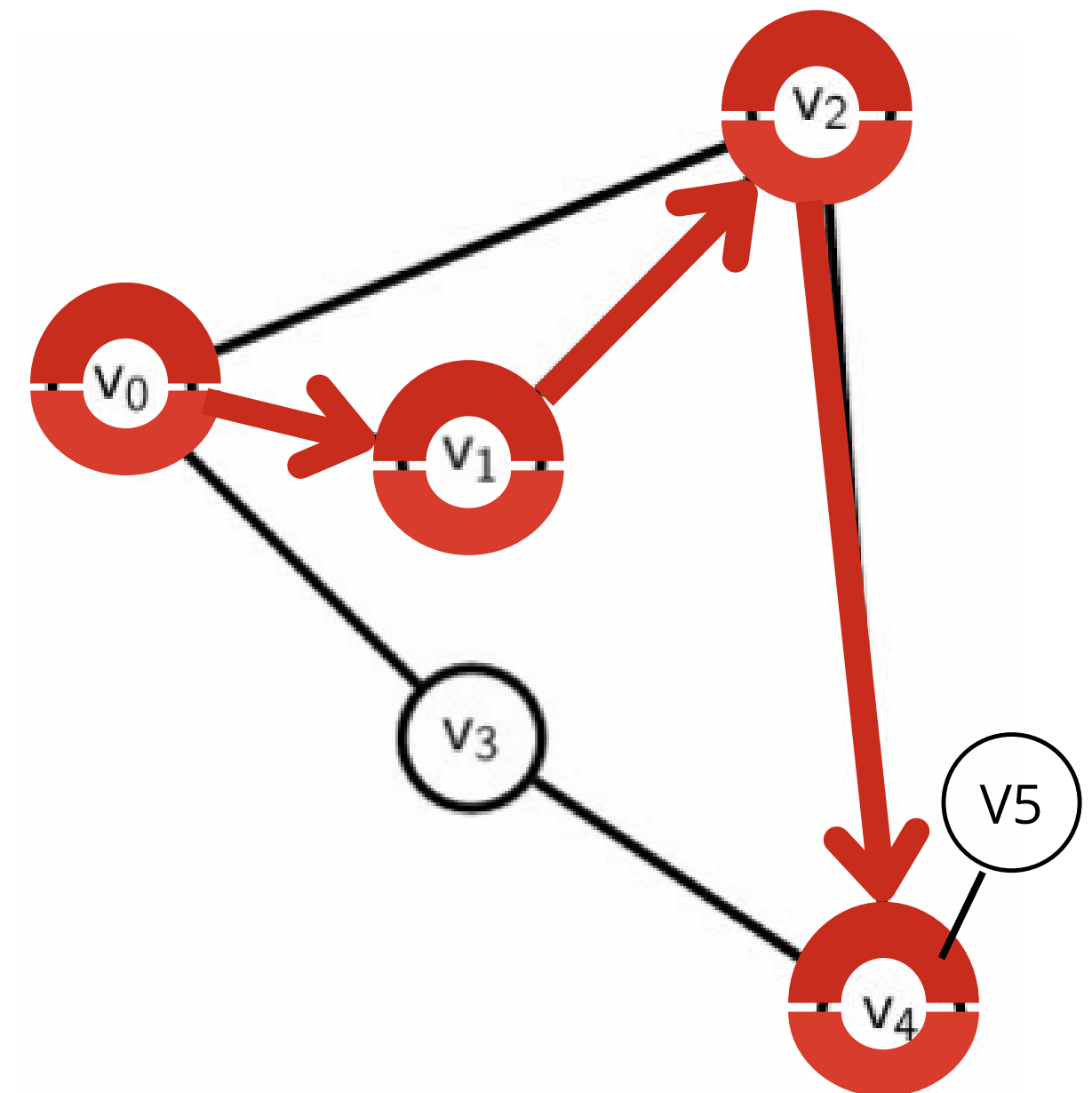


TESTE

```
buscaProfundidade(grafo, 0, 4, numV);
```

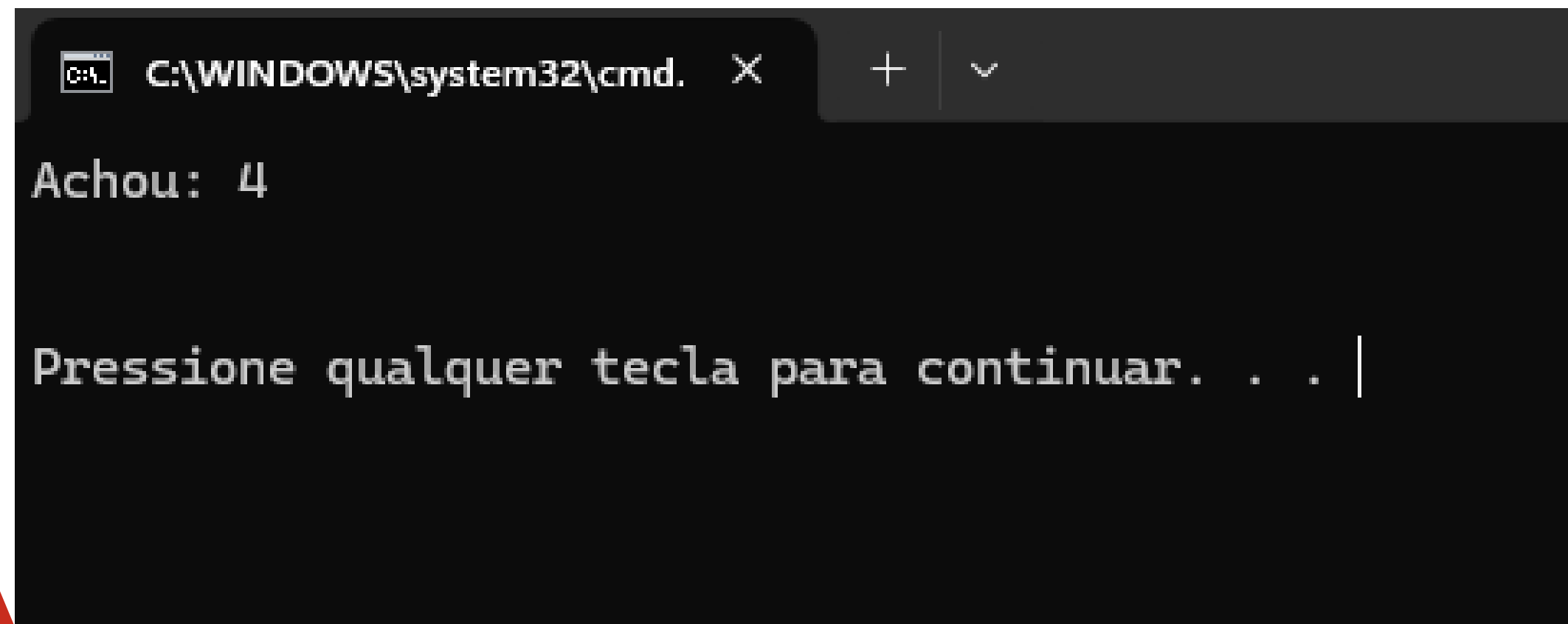
```
if (buscaProfundidadeVisitas(grafo, vizinho, destino, visitados))  
    return true;
```

```
    if (!visitados[i] && !existe)  
    {  
        existe = buscaProfundidadeVisitas(grafo, i, destino, visitados);  
    }  
    if (!existe)  
        cout << "Caminho Não Existe!";
```



TESTE - RESULTADO

```
buscaProfundidade(grafo, 0, 4, numV);
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\WINDOWS\system32\cmd.' and standard window controls. The command prompt displays the text 'Achou: 4' on the first line and 'Pressione qualquer tecla para continuar. . . |' on the second line, indicating the program has finished execution and is waiting for a key press.

