

Busca em Profundidade

Prof. Andrei Braga



Conteúdo

- Percorrendo um sistema de passagens
- Busca em profundidade
- Exercícios
- Referências

Percorrendo um sistema de passagens

- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:



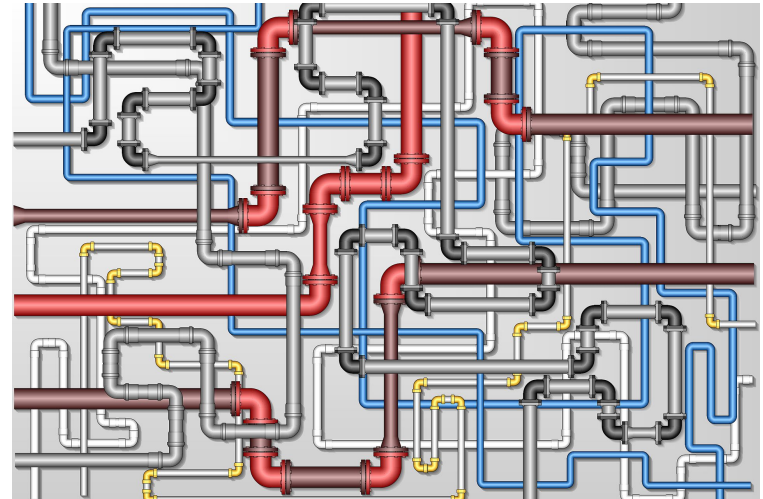
Percorrendo um sistema de passagens

- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:



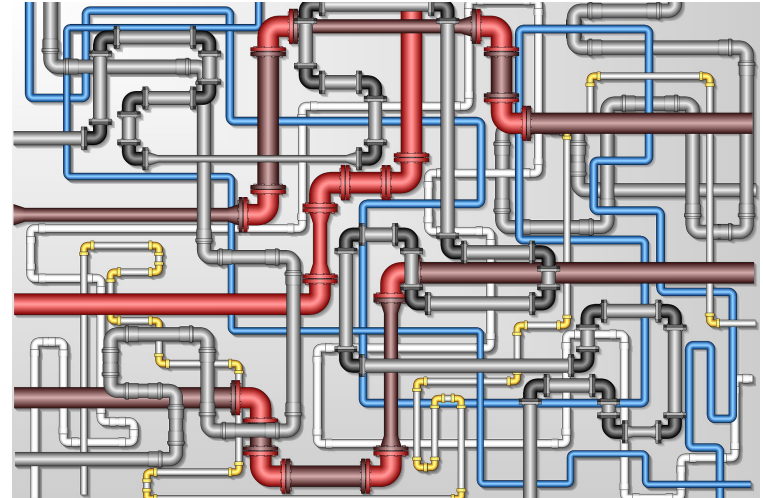
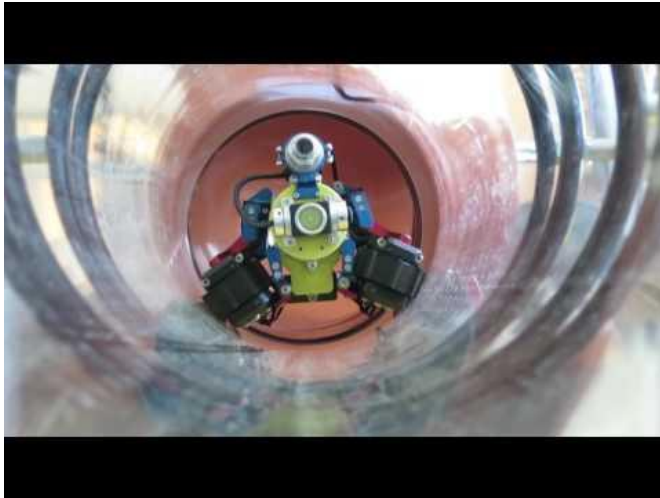
Percorrendo um sistema de passagens

- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:

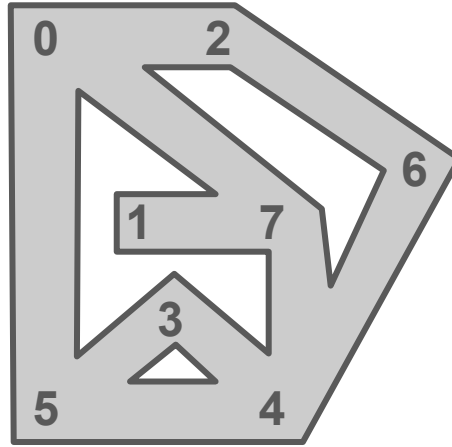


Percorrendo um sistema de passagens

- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:

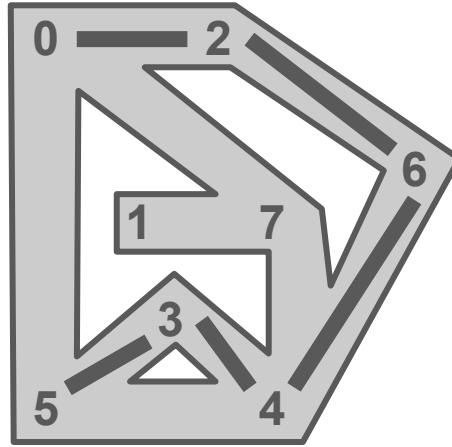


Percorrendo um sistema de passagens



Percorrendo um sistema de passagens

Partindo do 0

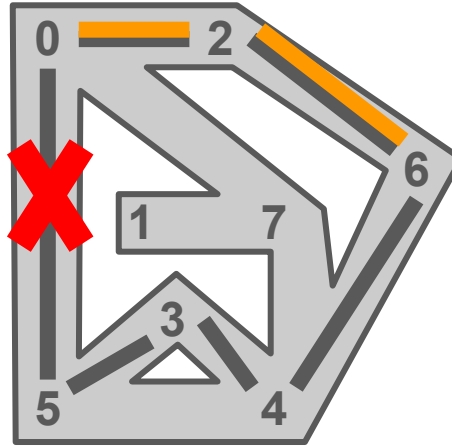


Percorrendo um sistema de passagens

Partindo do 0

**Não vamos visitar novamente
pontos já visitados**

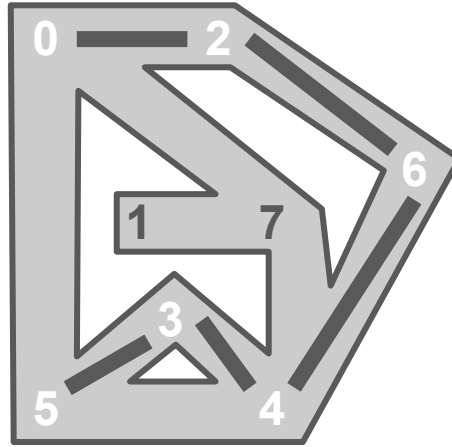
Para isso, vamos marcar os
pontos que vão sendo visitados



Percorrendo um sistema de passagens

Partindo do 0

Para isso, vamos marcar os pontos que vão sendo visitados

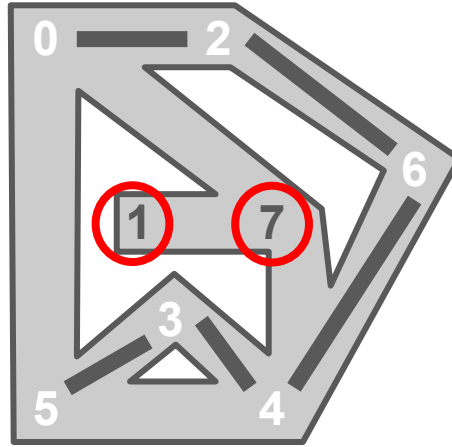


Percorrendo um sistema de passagens

Partindo do 0

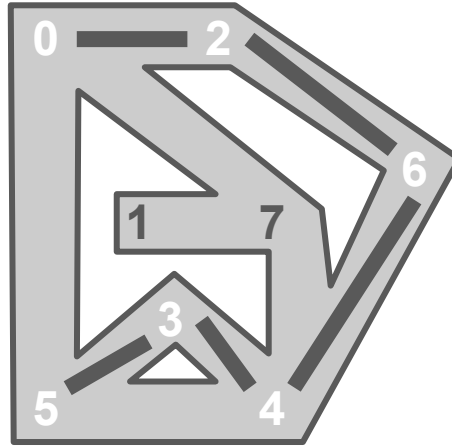
Ainda podem existir pontos não visitados

Por isso, vamos retornar ao ponto de onde viemos e tentar visitar pontos ainda não visitados



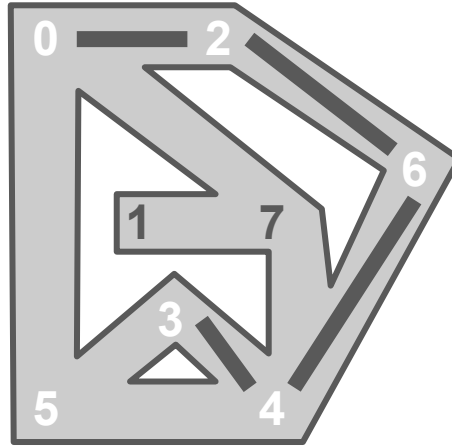
Percorrendo um sistema de passagens

Partindo do 0



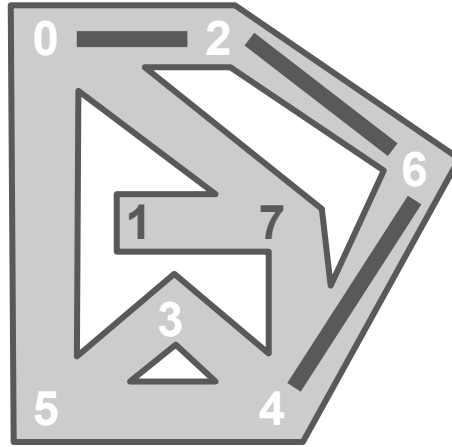
Percorrendo um sistema de passagens

Partindo do 0



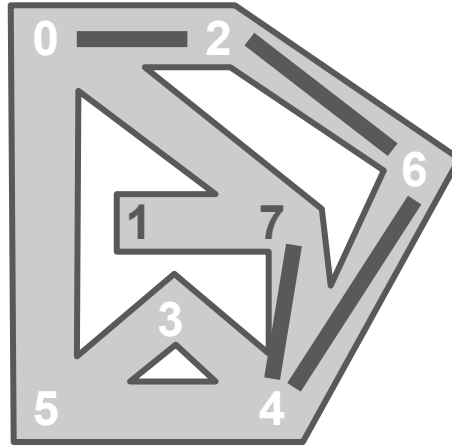
Percorrendo um sistema de passagens

Partindo do 0



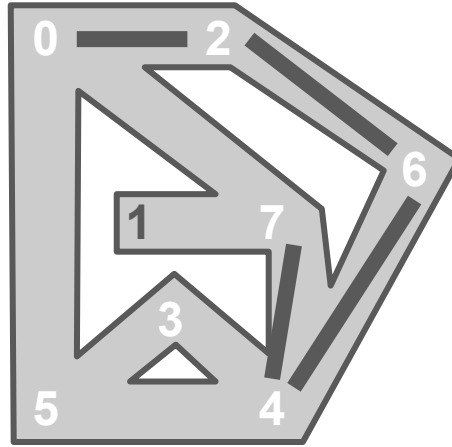
Percorrendo um sistema de passagens

Partindo do 0



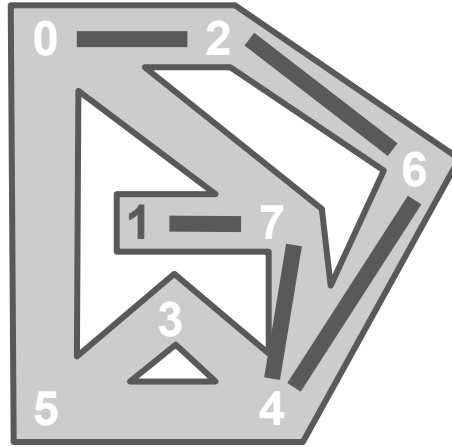
Percorrendo um sistema de passagens

Partindo do 0



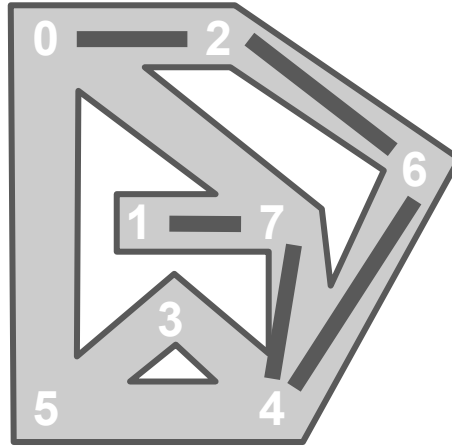
Percorrendo um sistema de passagens

Partindo do 0



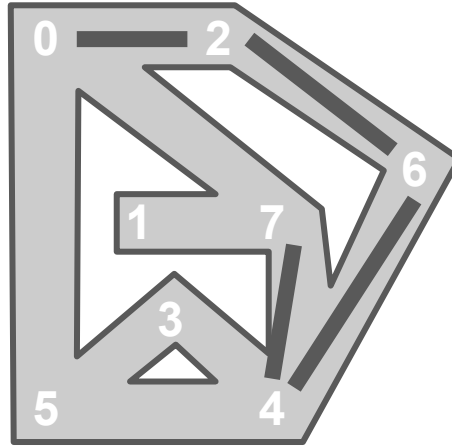
Percorrendo um sistema de passagens

Partindo do 0



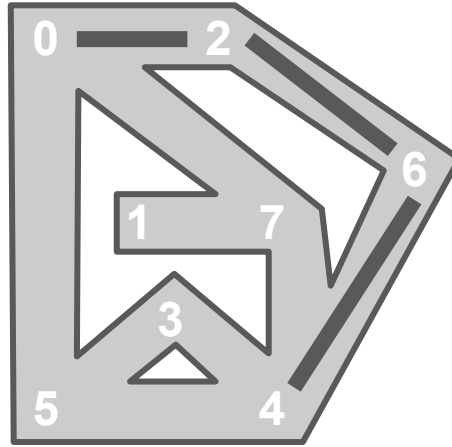
Percorrendo um sistema de passagens

Partindo do 0



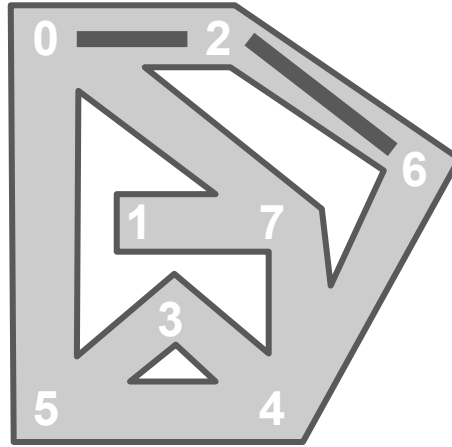
Percorrendo um sistema de passagens

Partindo do 0



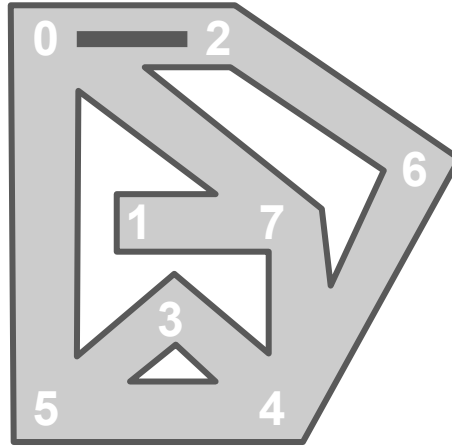
Percorrendo um sistema de passagens

Partindo do 0



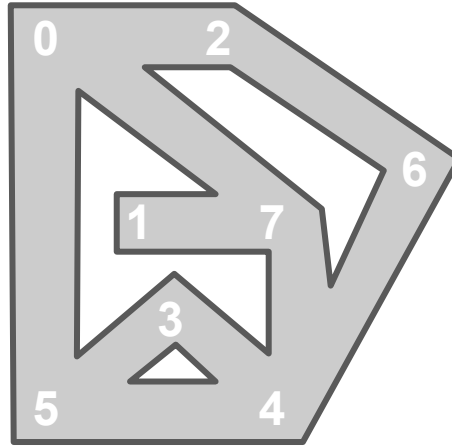
Percorrendo um sistema de passagens

Partindo do 0



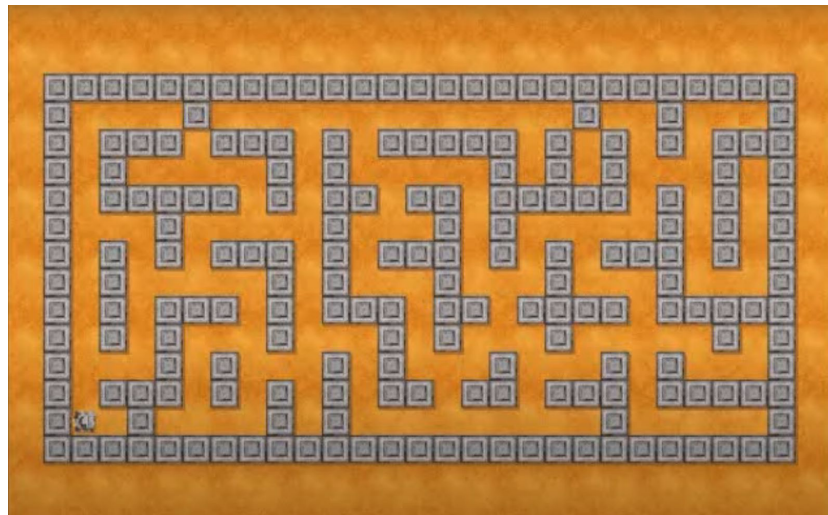
Percorrendo um sistema de passagens

Partindo do 0



Percorrendo um sistema de passagens

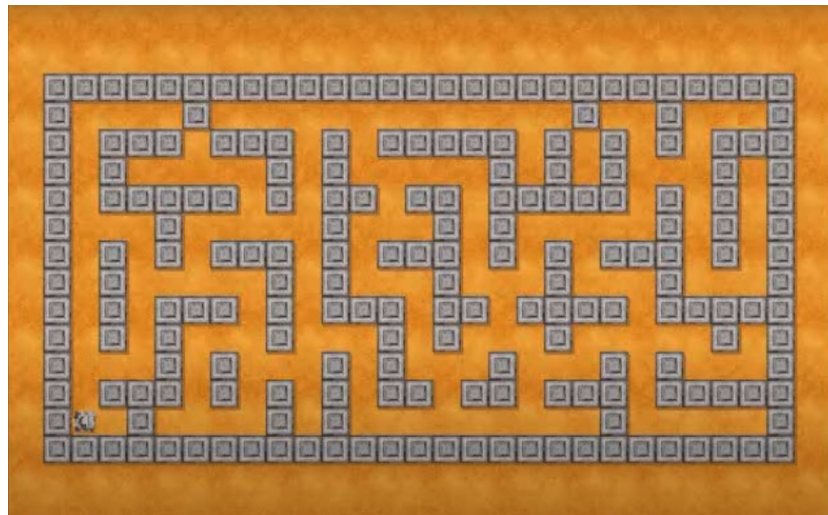
- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:



Percorrendo um sistema de passagens

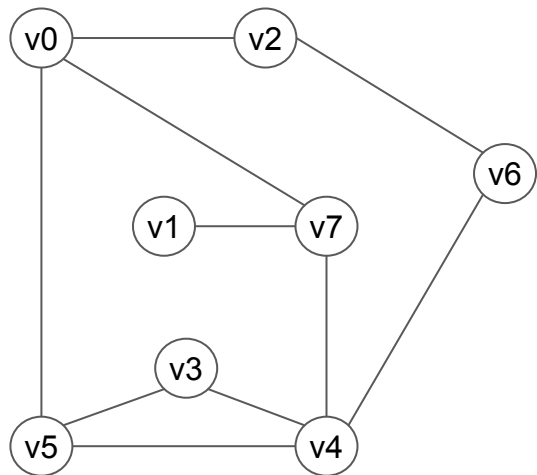
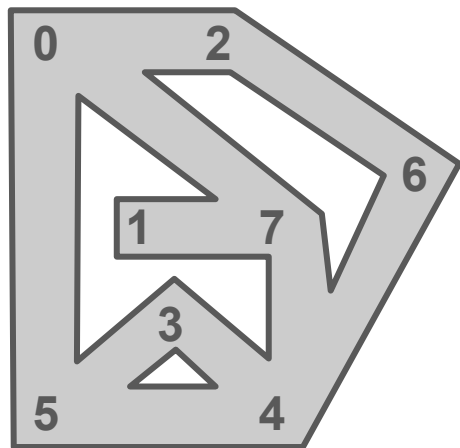
- Em várias situações, temos um sistema de **passagens** e queremos percorrer de maneira eficiente as **junções** e **extremidades** destas passagens
- Exemplo:

Queremos **continuar** percorrendo as **junções** e **extremidades** mesmo depois de ter achado algum ponto de interesse



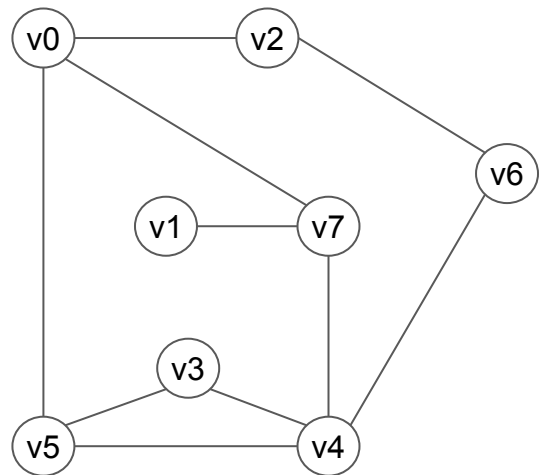
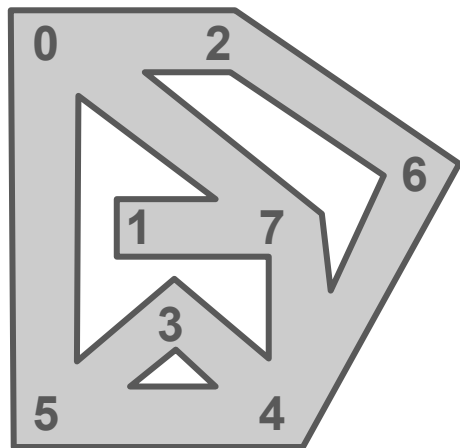
Percorrendo um sistema de passagens

- Vamos representar o sistema de passagens como um grafo: passagens modeladas por arestas; junções e extremidades modeladas por vértices
- Exemplo:



Percorrendo um sistema de passagens

- Vamos representar o sistema de passagens como um grafo: passagens modeladas por arestas; junções e extremidades modeladas por vértices
- Vamos implementar um algoritmo para **percorrer os vértices** do grafo
- Exemplo:



Percorrendo um sistema de passagens

- Estratégia:
 - Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0



Percorrendo um sistema de passagens

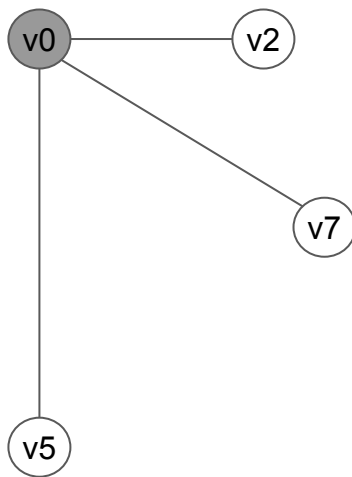
- Estratégia:
 - Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0



Percorrendo um sistema de passagens

- Estratégia:

- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0



Percorrendo um sistema de passagens

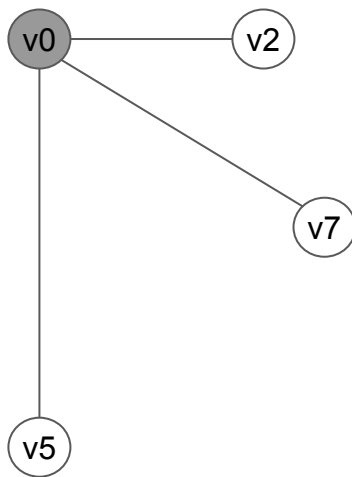
- Estratégia:

- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0

1. Comece em v_0

2. Considere os vizinhos de v_0

Observe que, para percorrer os demais vértices do grafo, podemos

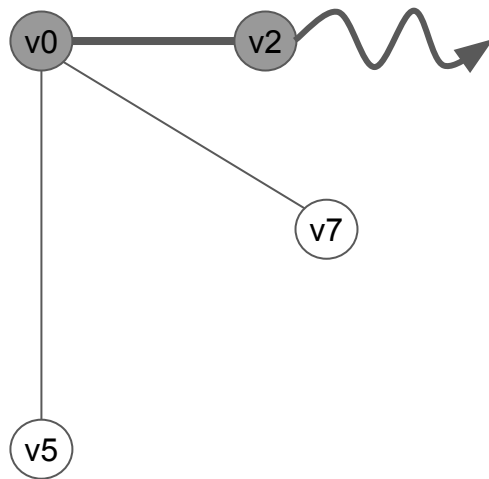


Percorrendo um sistema de passagens

- Estratégia:

- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0

Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2

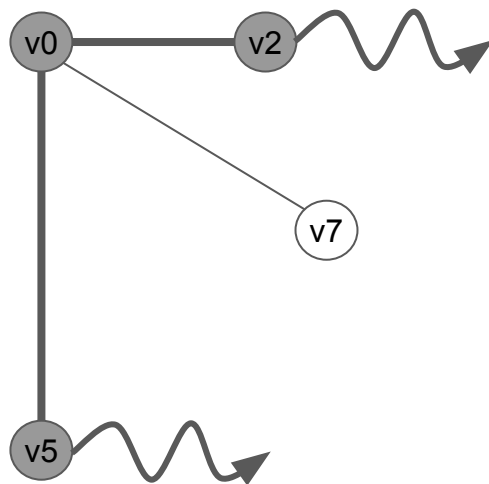


Percorrendo um sistema de passagens

- Estratégia:

- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0

Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2 , percorrer os vértices partindo de v_5

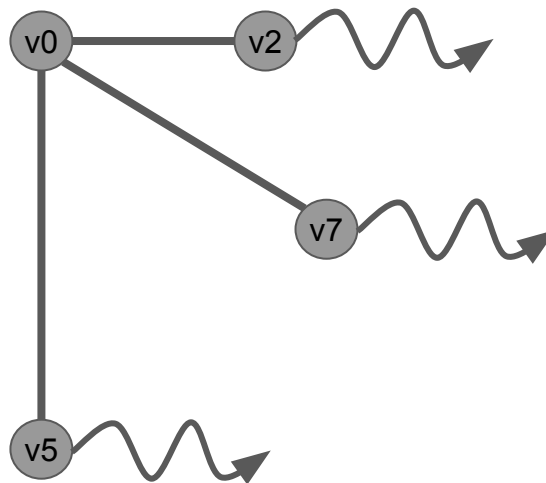


Percorrendo um sistema de passagens

- Estratégia:

- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0

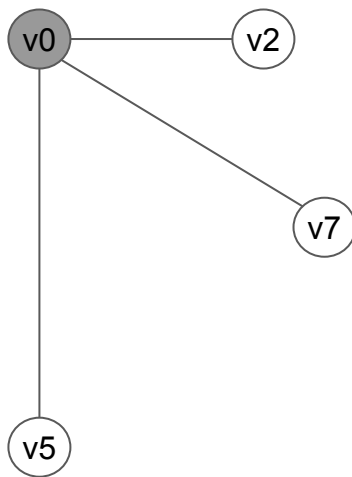
Observe que, para percorrer os demais vértices do grafo, podemos percorrer os vértices partindo de v_2 , percorrer os vértices partindo de v_5 e percorrer os vértices partindo de v_7



Percorrendo um sistema de passagens

- Estratégia:

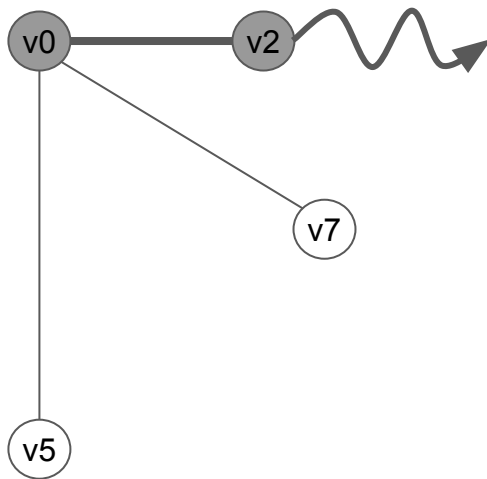
- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
1. Comece em v_0
 2. Considere os vizinhos de v_0



Percorrendo um sistema de passagens

- Estratégia:

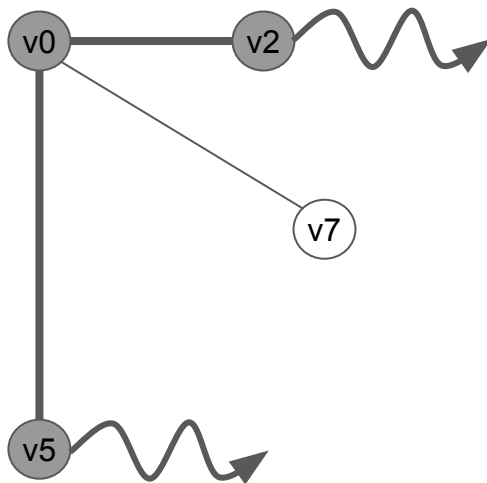
- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0
- 3. Percorra recursivamente os vértices do grafo partindo de v_2



Percorrendo um sistema de passagens

- Estratégia:

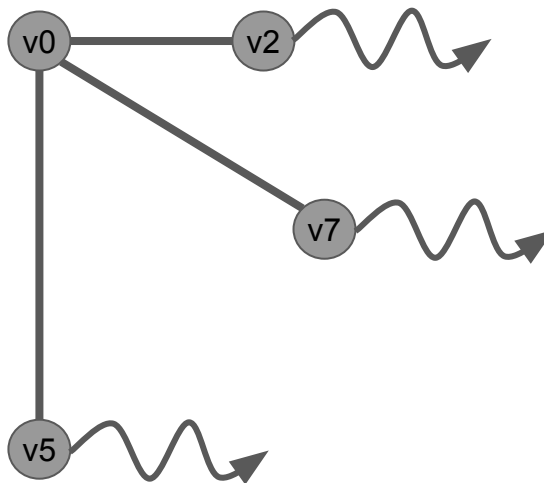
- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0
- 3. Percorra recursivamente os vértices do grafo partindo de v_2
- 4. Percorra recursivamente os vértices do grafo partindo de v_5



Percorrendo um sistema de passagens

- Estratégia:

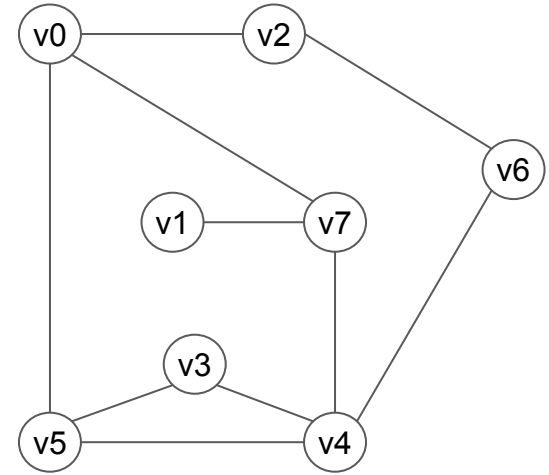
- Vamos percorrer os vértices do grafo partindo de um vértice especificado; por ex., v_0
- 1. Comece em v_0
- 2. Considere os vizinhos de v_0
- 3. Percorra recursivamente os vértices do grafo partindo de v_2
- 4. Percorra recursivamente os vértices do grafo partindo de v_5 e
- 5. Percorra recursivamente os vértices do grafo partindo de v_7



Não queremos visitar novamente vértices já visitados. Por isso, vamos marcar os vértices que vão sendo visitados

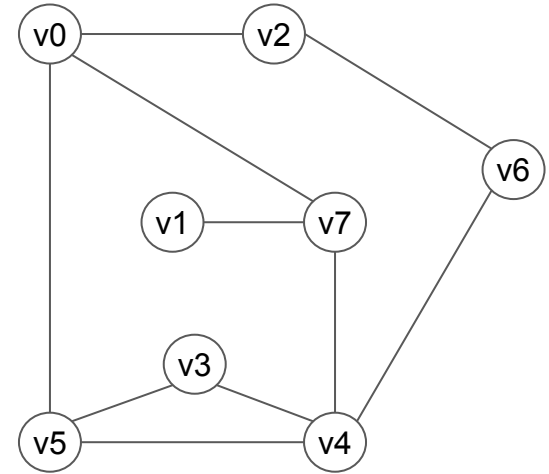
Percorrendo um sistema de passagens - Implementação

```
void Grafo::percorre(int v) {
```



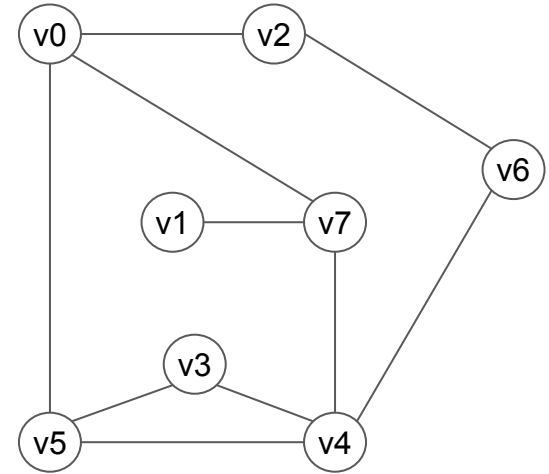
Percorrendo um sistema de passagens - Implementação

```
void Grafo::percorre(int v) {  
    printf("%d\n", v);  
  
}
```



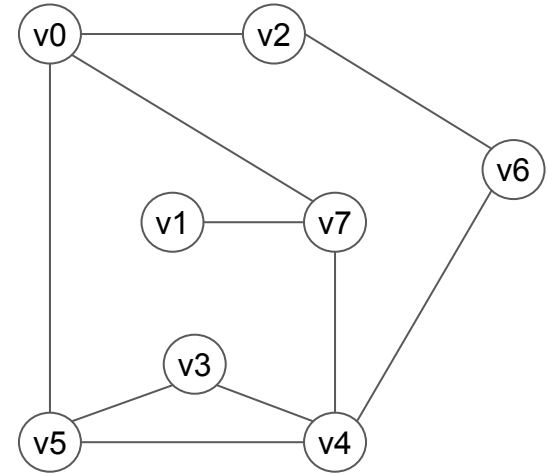
Percorrendo um sistema de passagens - Implementação

```
void Grafo::percorre(int v) {  
    printf("%d\n", v);  
  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
  
}
```



Percorrendo um sistema de passagens - Implementação

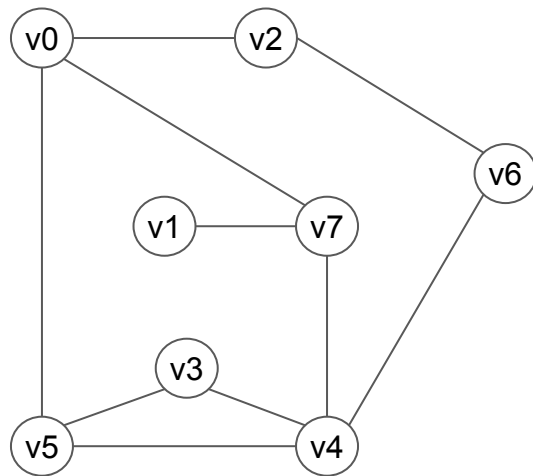
```
void Grafo::percorre(int v) {  
    printf("%d\n", v);  
  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            percorre(u);  
}
```



Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo percorre ser chamado
```

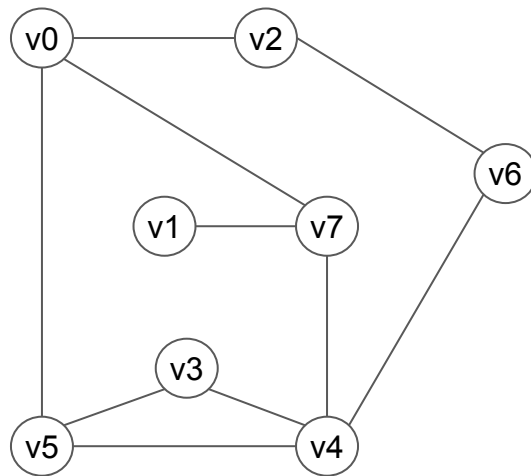
```
void Grafo::percorre(int v, int marcado[]) {  
    printf("%d\n", v);  
  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            percorre(u, marcado);  
}
```



Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo percorre ser chamado
```

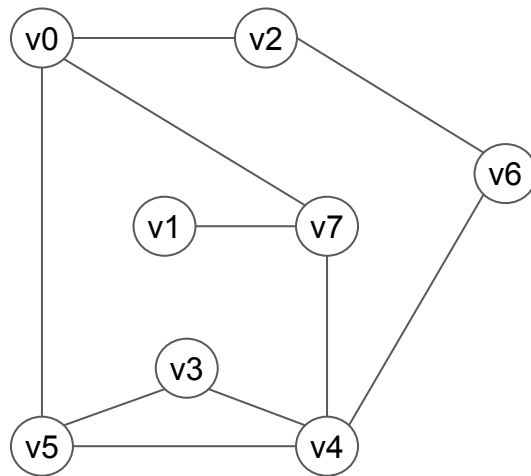
```
void Grafo::percorre(int v, int marcado[]) {  
    printf("%d\n", v);  
    marcado[v] = 1;  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            if (marcado[u] == 0)  
                percorre(u, marcado);  
}
```



Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo percorre ser chamado
```

```
void Grafo::percorre(int v, int marcado[]) {  
    printf("%d\n", v);  
    marcado[v] = 1;  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            if (marcado[u] == 0)  
                percorre(u, marcado);  
}
```

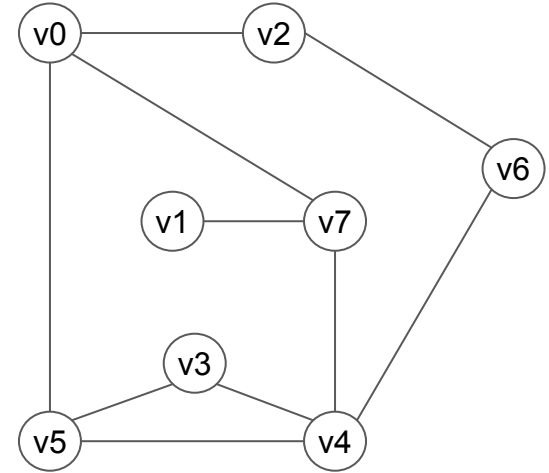


O processo de percorrer um grafo também é chamado de **busca**

Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo busca ser chamado
```

```
void Grafo::busca(int v, int marcado[]) {  
    printf("%d\n", v);  
    marcado[v] = 1;  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            if (marcado[u] == 0)  
                busca(u, marcado);  
}
```

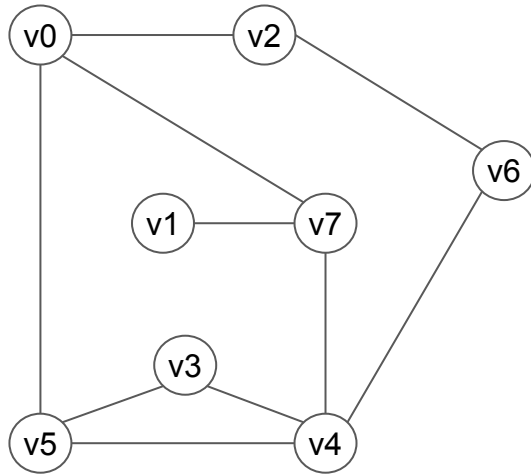


O processo de percorrer um grafo também é chamado de **busca**

Percorrendo um sistema de passagens - Dinâmica

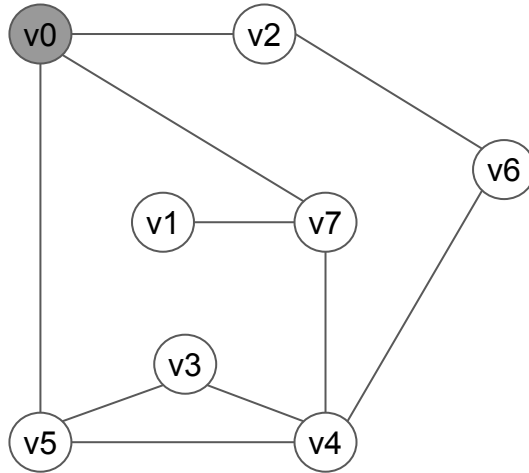
- Vamos construir um grafo H que representa a dinâmica da busca realizada
 - Quando o vértice inicial da busca é visitado, o adicionamos a H
 - Quando um novo vértice v é visitado, se chegamos a v através da aresta wv , então adicionamos a H a aresta wv e o vértice v

Percorrendo um sistema de passagens - Dinâmica



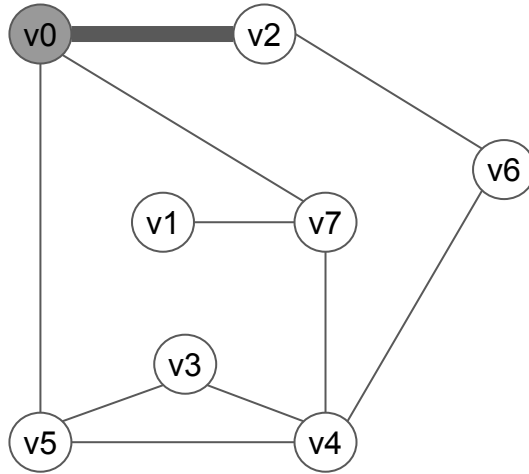
Percorrendo um sistema de passagens - Dinâmica

H :

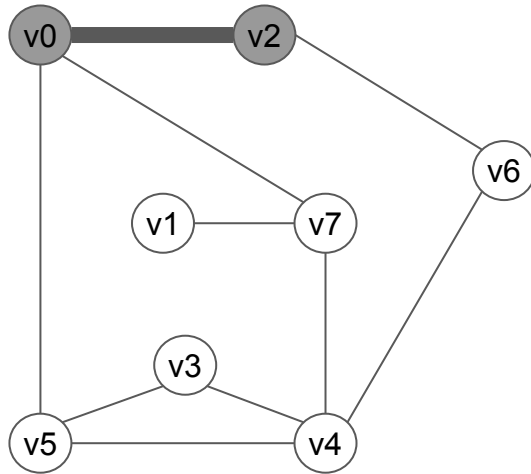


Percorrendo um sistema de passagens - Dinâmica

H :



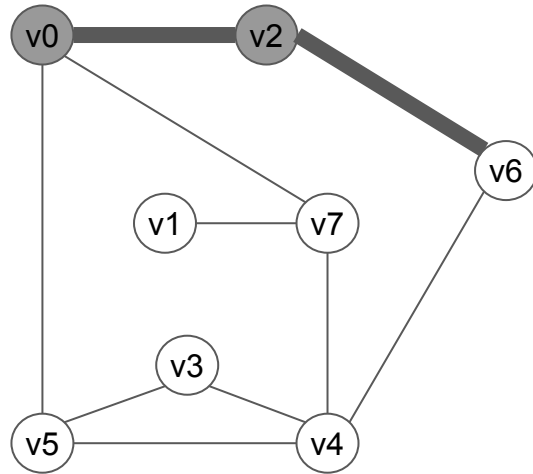
Percorrendo um sistema de passagens - Dinâmica



H :



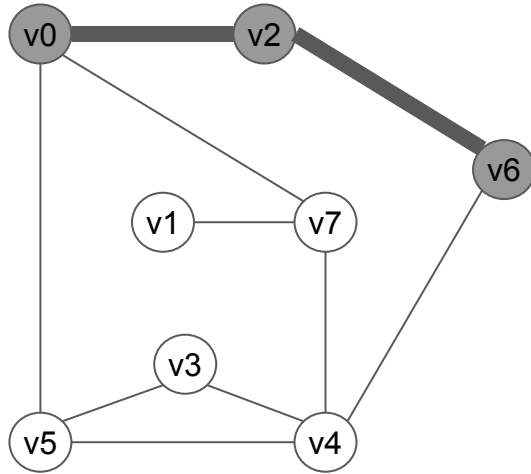
Percorrendo um sistema de passagens - Dinâmica



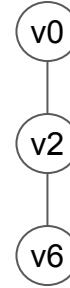
H :



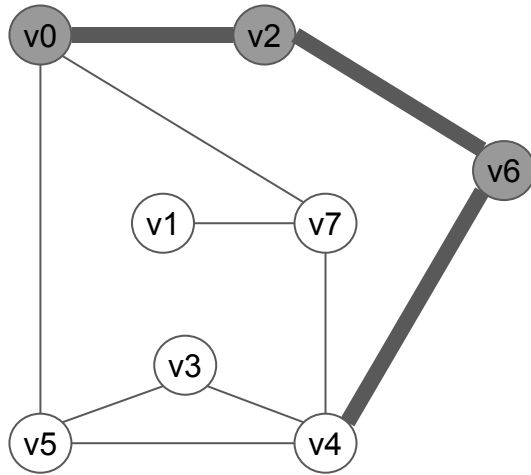
Percorrendo um sistema de passagens - Dinâmica



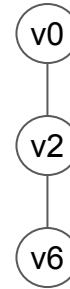
H :



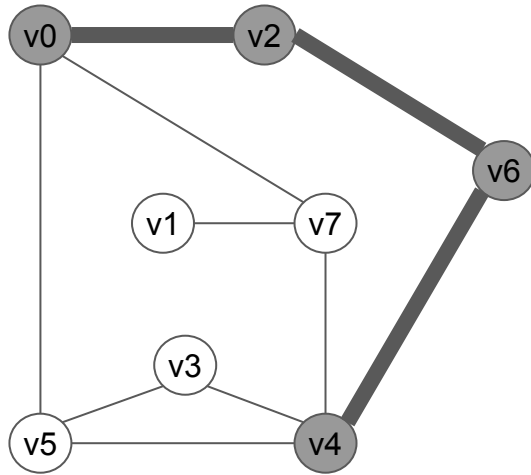
Percorrendo um sistema de passagens - Dinâmica



H :



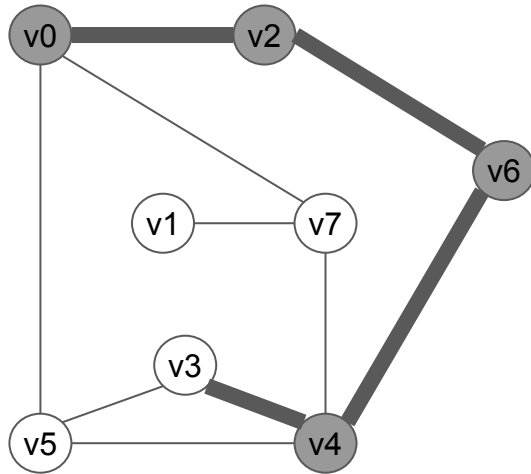
Percorrendo um sistema de passagens - Dinâmica



H :



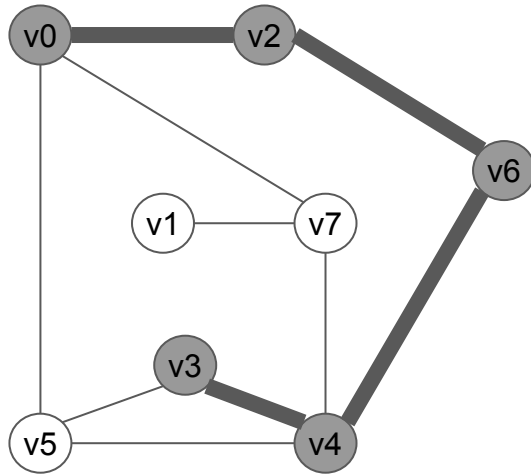
Percorrendo um sistema de passagens - Dinâmica



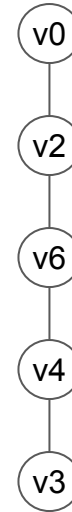
H :



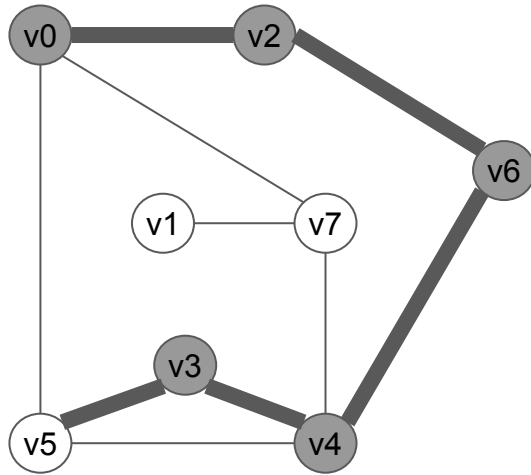
Percorrendo um sistema de passagens - Dinâmica



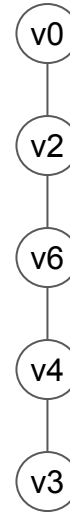
H :



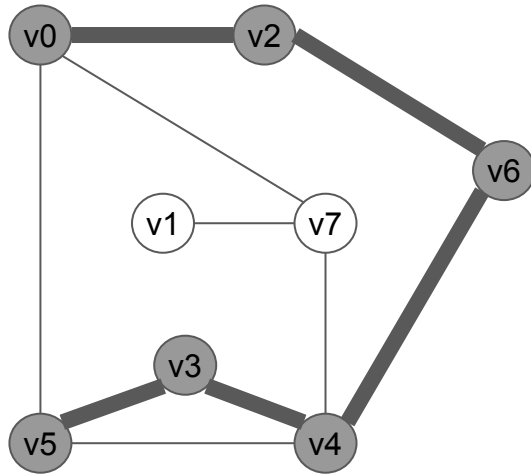
Percorrendo um sistema de passagens - Dinâmica



H :



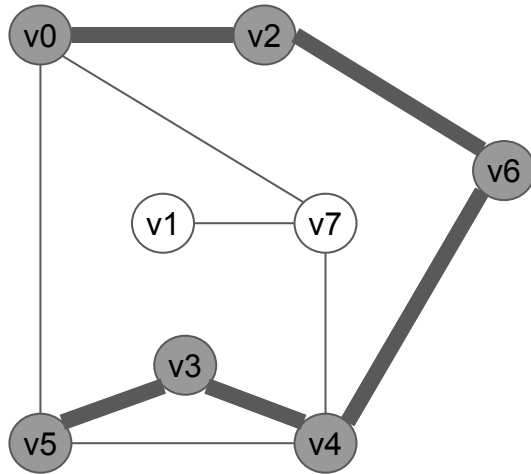
Percorrendo um sistema de passagens - Dinâmica



H :



Percorrendo um sistema de passagens - Dinâmica

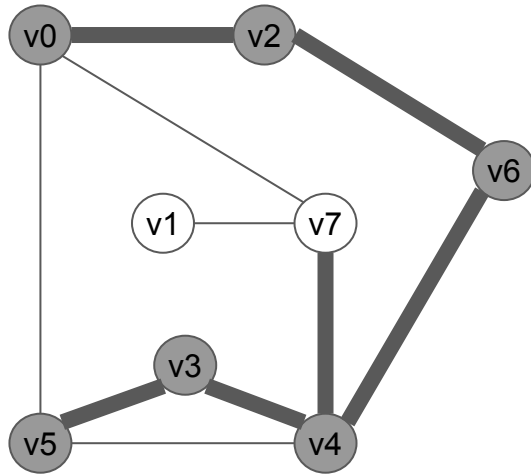


H:



A busca segue em **profundidade** até não ser mais possível, para depois retornar

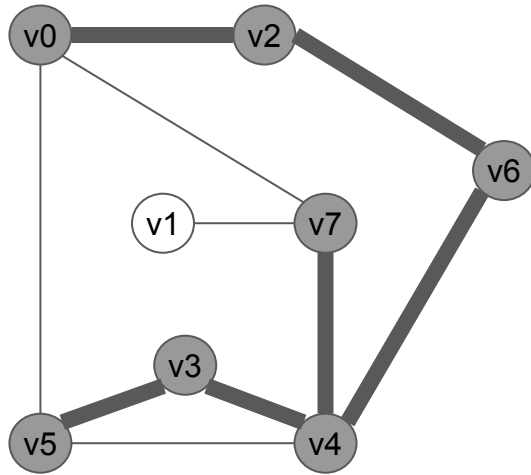
Percorrendo um sistema de passagens - Dinâmica



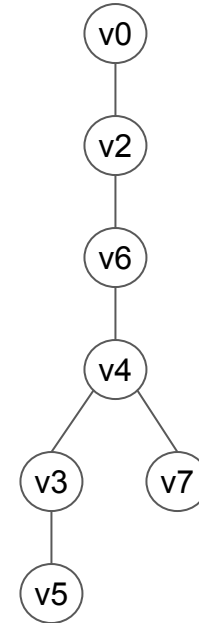
H:



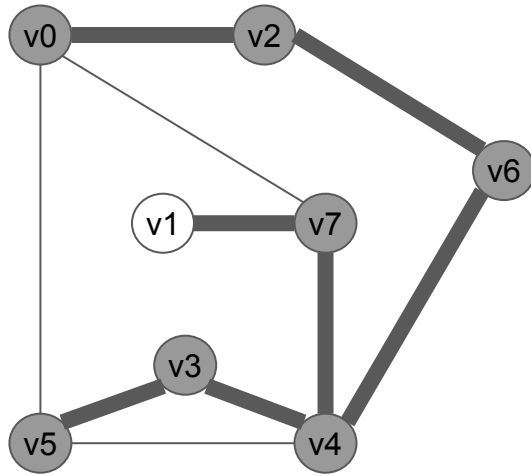
Percorrendo um sistema de passagens - Dinâmica



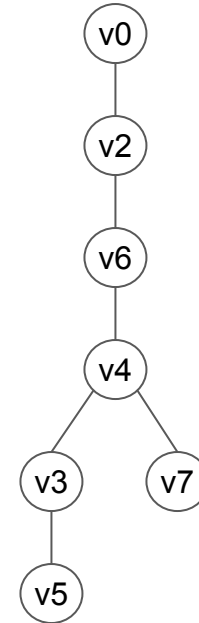
H :



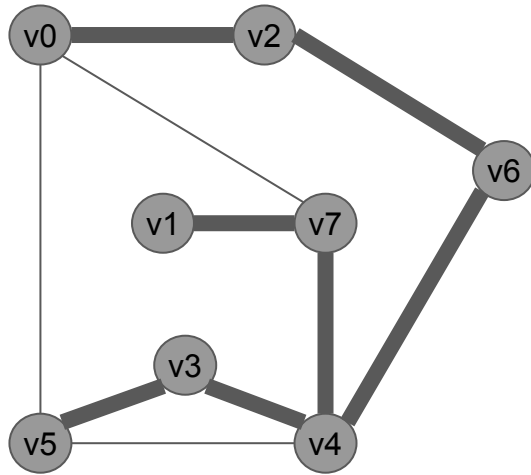
Percorrendo um sistema de passagens - Dinâmica



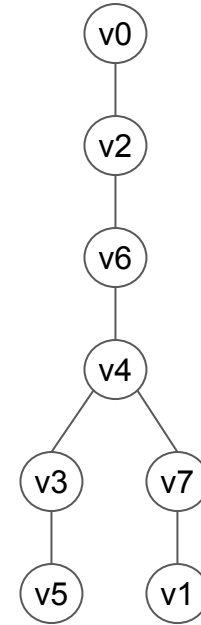
H :



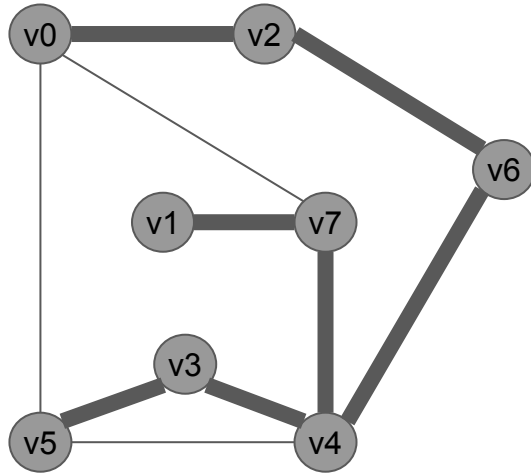
Percorrendo um sistema de passagens - Dinâmica



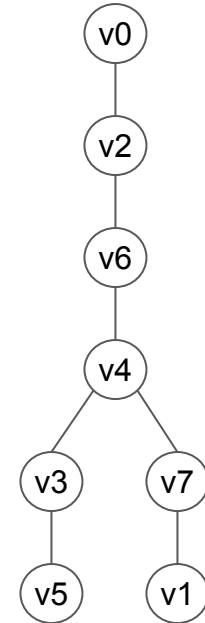
H :



Percorrendo um sistema de passagens - Dinâmica



H :

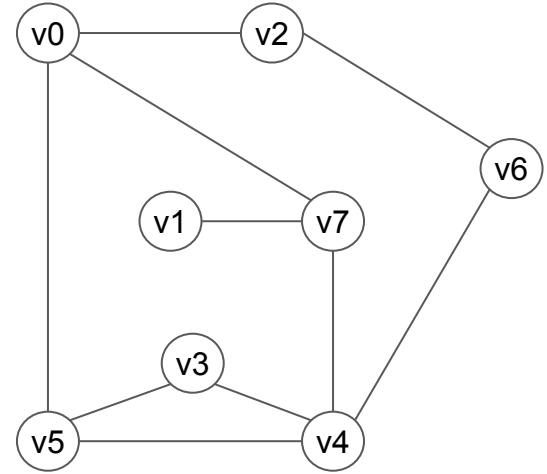


H é uma **árvore**

Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo busca ser chamado
```

```
void Grafo::busca(int v, int marcado[]) {  
    printf("%d\n", v);  
    marcado[v] = 1;  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            if (marcado[u] == 0)  
                busca(u, marcado);  
}
```

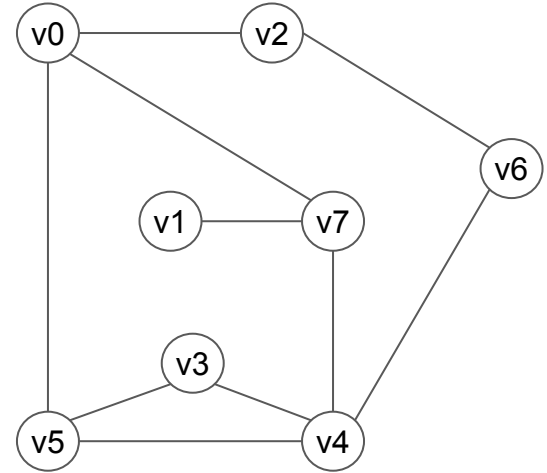


A busca segue em **profundidade** até não ser mais possível, para depois retornar

Percorrendo um sistema de passagens - Implementação

```
// 0 vetor marcado eh criado e inicializado antes  
// do metodo busca_prof ser chamado
```

```
void Grafo::busca_prof(int v, int marcado[]) {  
    printf("%d\n", v);  
    marcado[v] = 1;  
    for (int u = 0; u < num_vertices_; u++)  
        if (matriz_adj_[v][u] != 0)  
            if (marcado[u] == 0)  
                busca_prof(u, marcado);  
}
```



A busca segue em **profundidade** até não ser mais possível, para depois retornar

Busca em um grafo

- A estratégia de busca em um grafo vista nos slides anteriores é conhecida como o algoritmo de **busca em profundidade**
- Ao realizar uma busca em um grafo, conseguimos visitar **todo vértice** w do grafo tal que existe um **caminho** entre o **vértice inicial** da busca e w
- Em outras palavras, conseguimos visitar todos os vértices da **componente conexa** do grafo que contém o vértice inicial da busca
- Se o grafo é conexo, então conseguimos visitar todos os seus vértices

Exercícios

- Exercício 1 da Lista de Exercícios “Busca em profundidade e em largura”.

Exercícios – Solução

- Exercício 1 da Lista de Exercícios “Busca em profundidade e em largura”.

Podemos realizar a busca e em seguida verificar se todos os vértices do grafo foram marcados como visitados

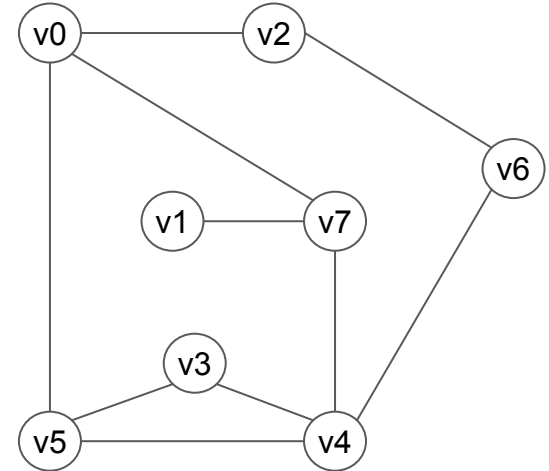
A solução descrita acima é mais eficiente que a solução comentada anteriormente para resolver o mesmo problema

Busca em um grafo

- Considere agora o seguinte objetivo:
Dado um grafo, queremos determinar um caminho de comprimento mínimo entre um certo vértice e cada um dos vértices do grafo
- Podemos atingir este objetivo através de uma estratégia de busca chamada **busca em largura**
- Para este objetivo, o algoritmo de busca em profundidade não é útil, pois a estratégia utilizada não tem relação com calcular caminhos de comprimento mínimo
- Antes de entender como funciona uma busca em largura, vamos examinar uma maneira alternativa de implementar uma busca em profundidade

Busca em um grafo - Estratégia geral

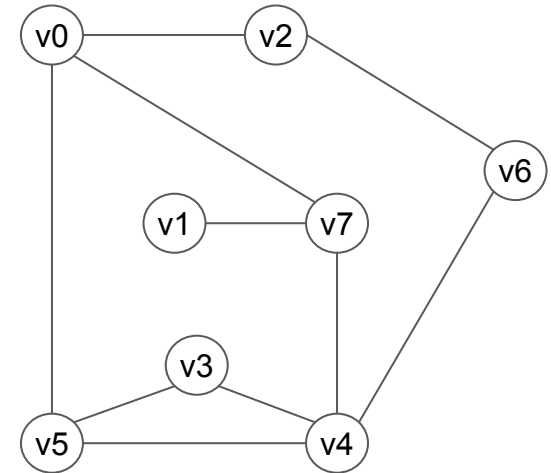
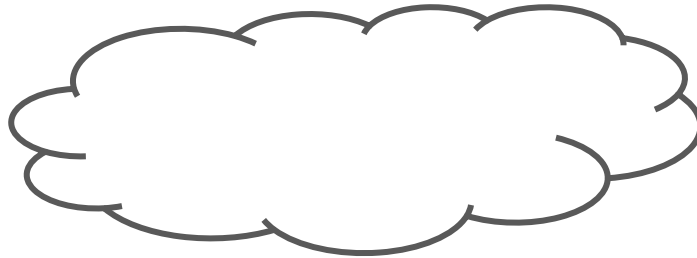
- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o próximo vértice
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação
- A estratégia acima pode ser implementada com o uso de uma estrutura de dados



Busca em um grafo - Estratégia geral

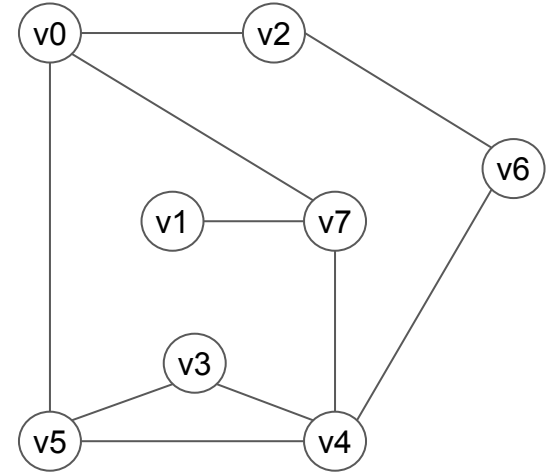
- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o próximo vértice
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

Estrutura de dados:

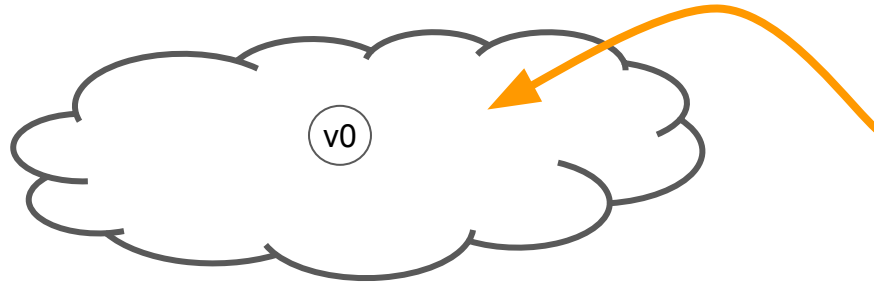


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. **Agenda** o vértice inicial **para visitação**
 2. Enquanto existe vértice a ser visitado:
 3. Visita o próximo vértice
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

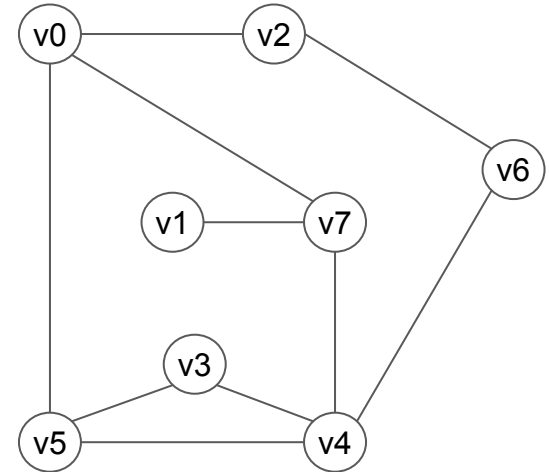


Estrutura de dados:

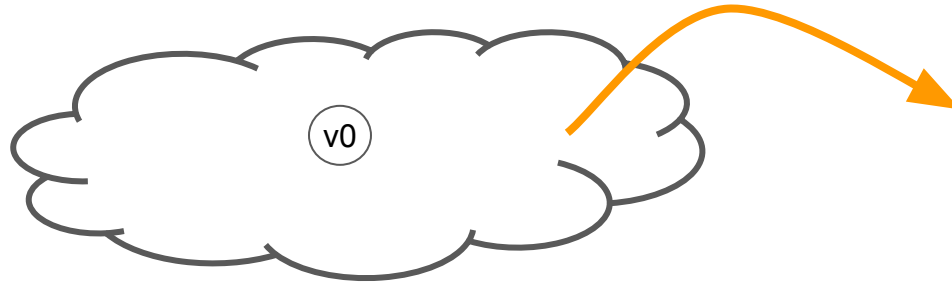


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o **próximo vértice**
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

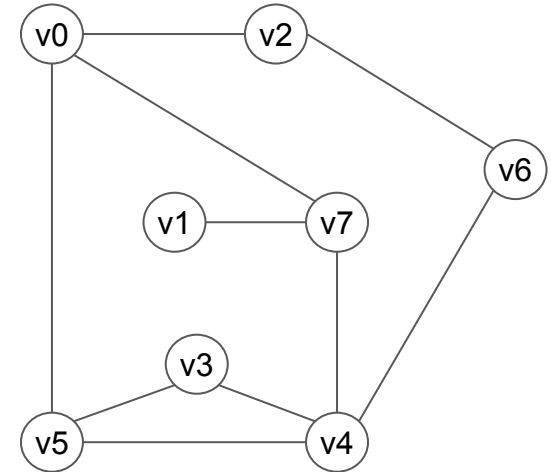


Estrutura de dados:

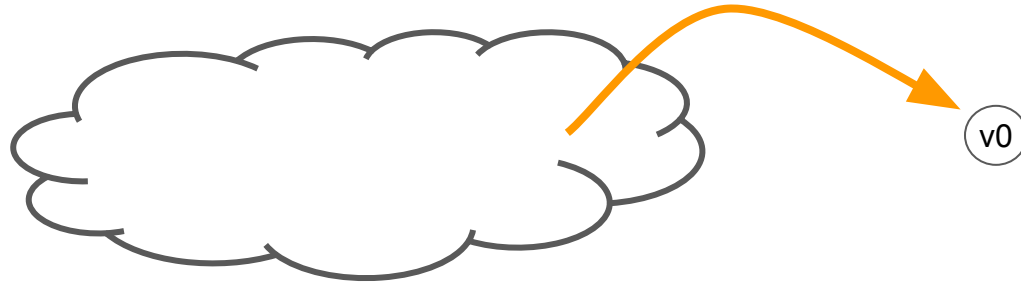


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o **próximo vértice**
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

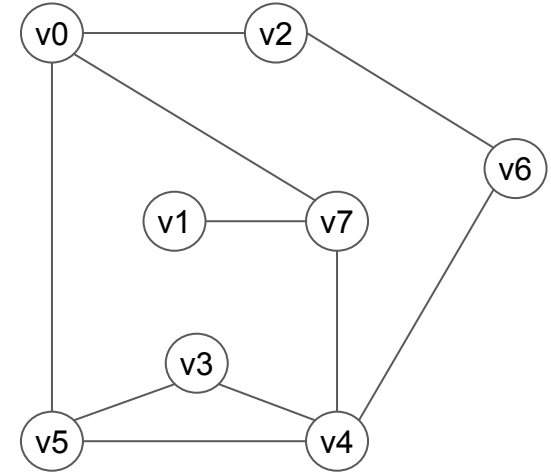


Estrutura de dados:

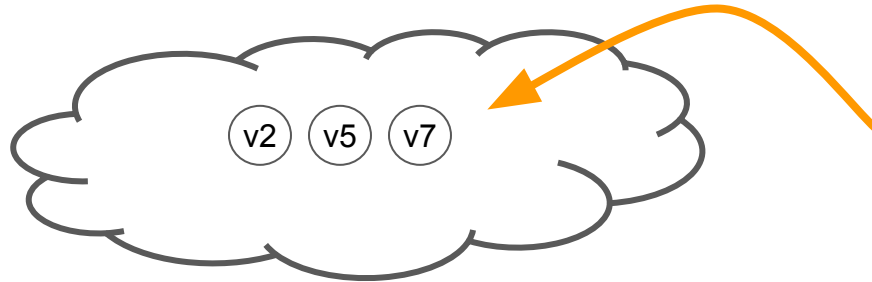


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o próximo vértice
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. **Agenda** o vizinho **para visitação**

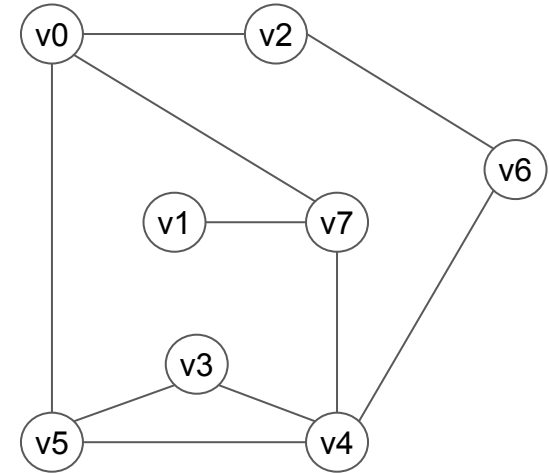


Estrutura de dados:

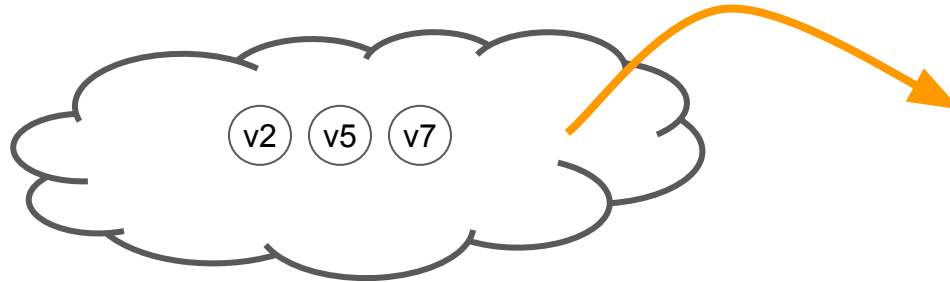


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o **próximo vértice**
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

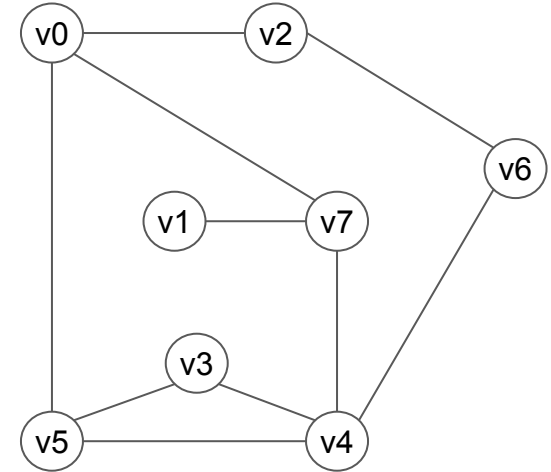


Estrutura de dados:

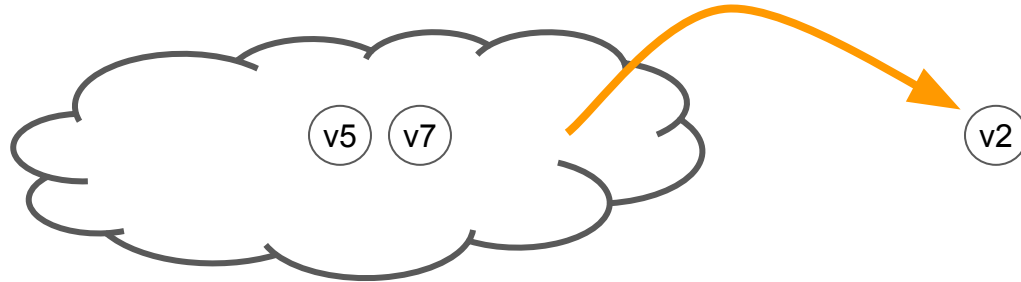


Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto existe vértice a ser visitado:
 3. Visita o **próximo vértice**
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação

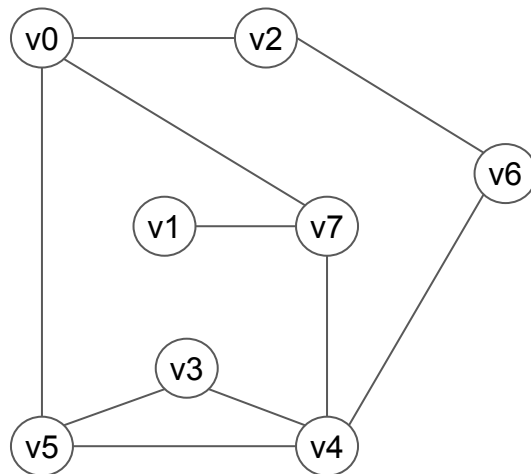


Estrutura de dados:



Busca em um grafo - Estratégia geral

- Estratégia geral:
 1. Agenda o vértice inicial para visitação
 2. Enquanto **existe vértice a ser visitado**:
 3. Visita o próximo vértice
 4. Para todo vizinho do vértice:
 5. Se o vizinho ainda não foi visitado
 6. Agenda o vizinho para visitação



Estrutura de dados:

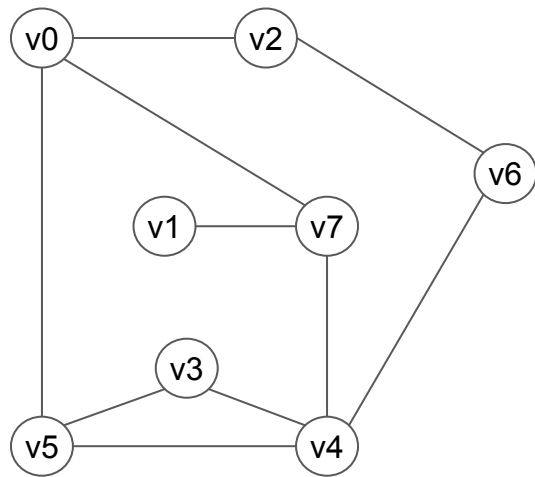


Busca em profundidade - Implementação alternativa

- Considerando a estratégia geral descrita no slide anterior, o que podemos dizer da lógica através da qual os vértices são visitados no algoritmo de busca em profundidade?
 - É uma lógica de **pilha**
- Vamos implementar este algoritmo usando explicitamente uma pilha

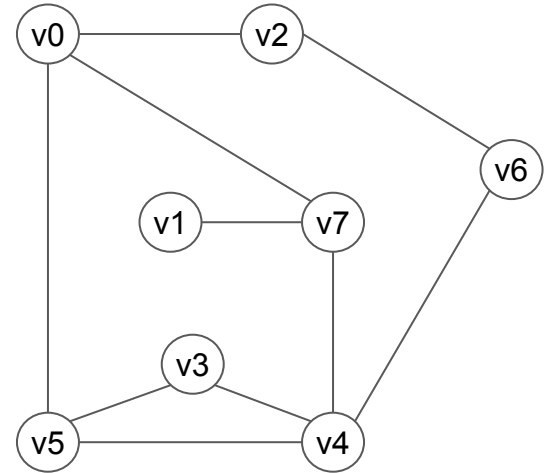
Busca em profundidade - Implementação alternativa

```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
  
    while () {  
  
        printf("%d\n", w);  
        marcado[w] = 1;  
        for (int u = 0; u < num_vertices_; u++)  
            if (matriz_adj_[w][u] != 0)  
                if (marcado[u] == 0)  
  
    }  
}
```



Busca em profundidade - Implementação alternativa

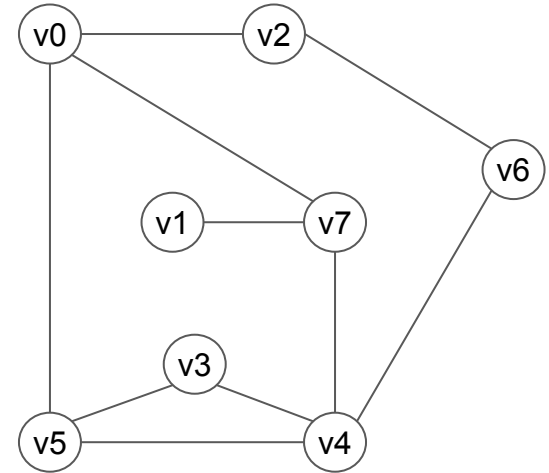
```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
    stack<int> pilha;  
    pilha.push(v);  
    while (!pilha.empty()) {  
        int w = pilha.top();  
        pilha.pop();  
        printf("%d\n", w);  
        marcado[w] = 1;  
        for (int u = 0; u < num_vertices_; u++)  
            if (matriz_adj_[w][u] != 0)  
                if (marcado[u] == 0)  
                    pilha.push(u);  
    }  
}
```



Nesta implementação, o **percurso** realizado pode ser **diferente** do realizado na **versão recursiva**

Busca em profundidade - Implementação alternativa

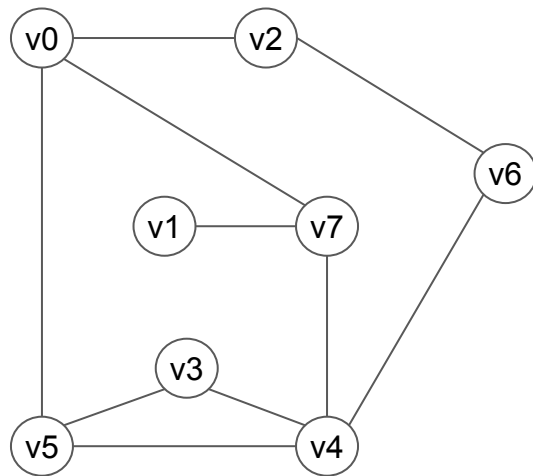
```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
    stack<int> pilha;  
    pilha.push(v);  
    while (!pilha.empty()) {  
        int w = pilha.top();  
        pilha.pop();  
        printf("%d\n", w);  
        marcado[w] = 1;  
        for (int u = (num_vertices_ - 1); u >= 0; u--)  
            if (matriz_adj_[w][u] != 0)  
                if (marcado[u] == 0)  
                    pilha.push(u);  
    }  
}
```



Com a alteração destacada, obtemos o **mesmo percurso** realizado na **versão recursiva**

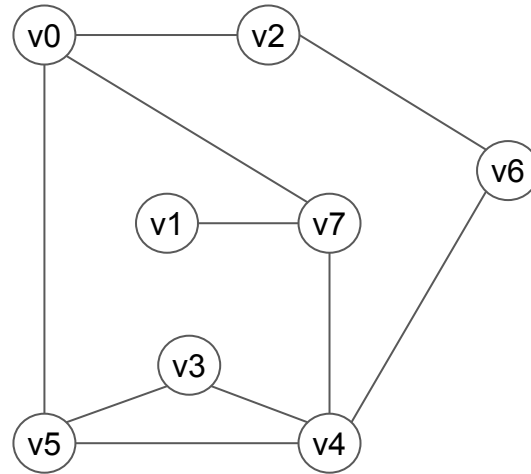
Busca em profundidade - Implementação alternativa

```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
    stack<int> pilha;  
    pilha.push(v);  
    while (!pilha.empty()) {  
        int w = pilha.top();  
        pilha.pop();  
        printf("%d\n", w);  
        marcado[w] = 1;  
        for (int u = (num_vertices_ - 1); u >= 0; u--)  
            if (matriz_adj_[w][u] != 0)  
                if (marcado[u] == 0)  
                    pilha.push(u);  
    }  
}
```



Nesta implementação, um vértice pode ser **visitado mais de uma vez!**

Busca em profundidade - Implementação alternativa

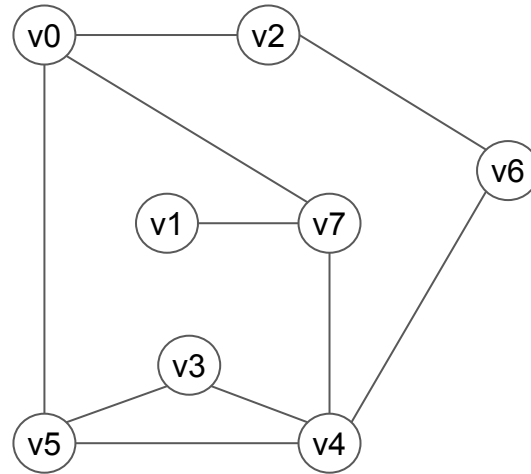


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

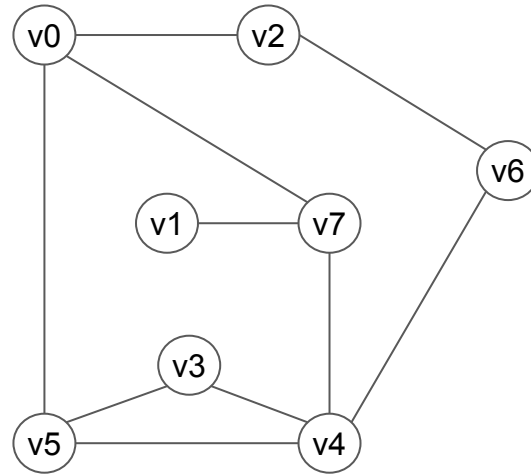


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

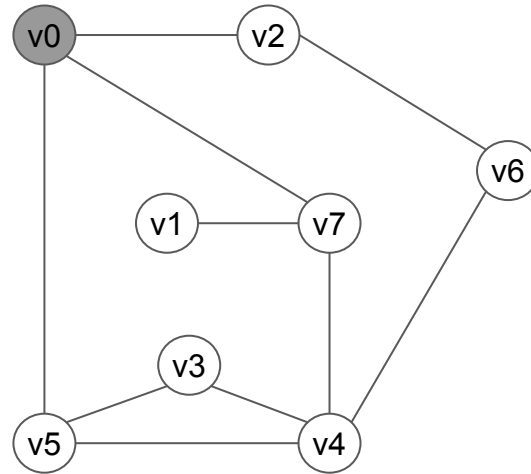


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

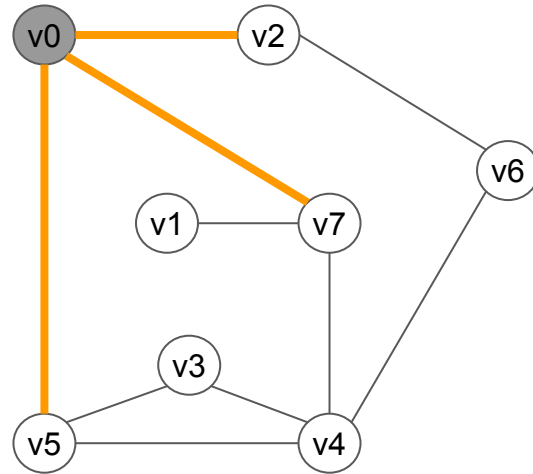


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

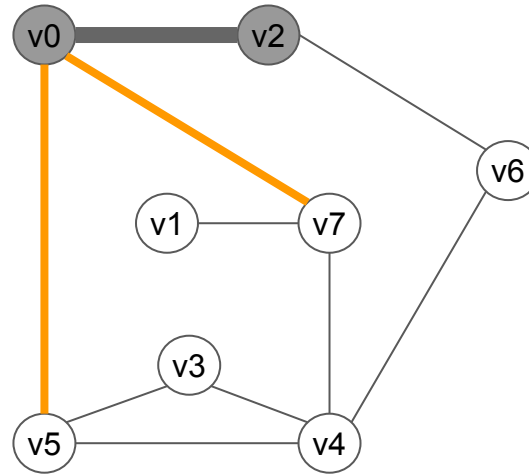


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

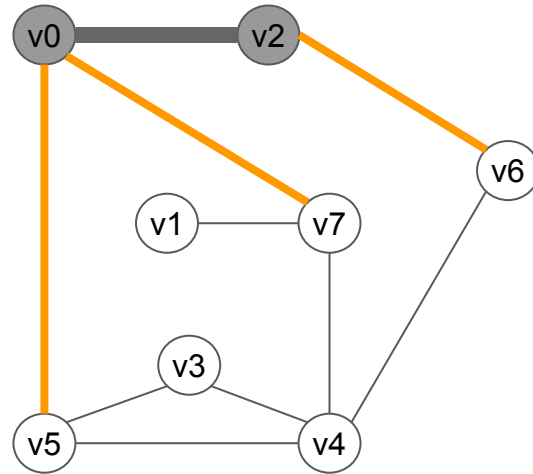


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

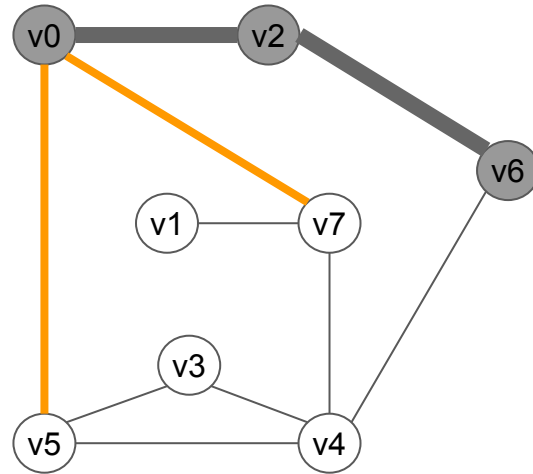


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

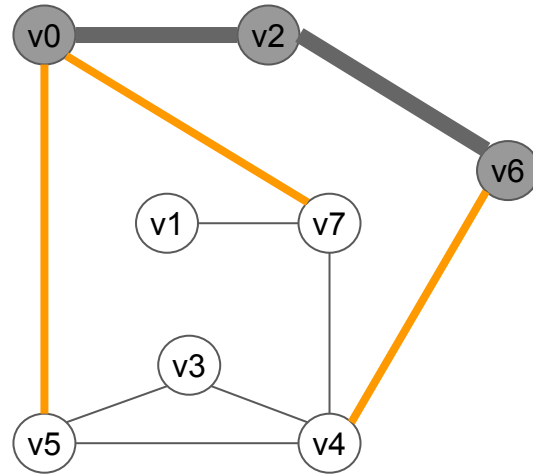


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

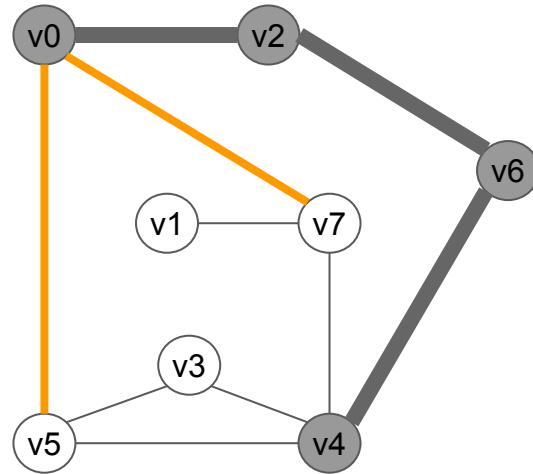


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

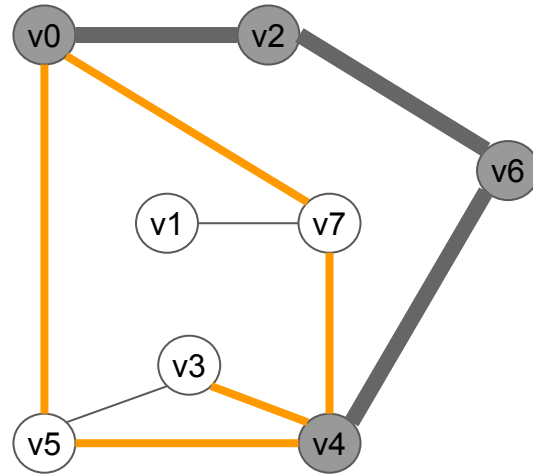


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

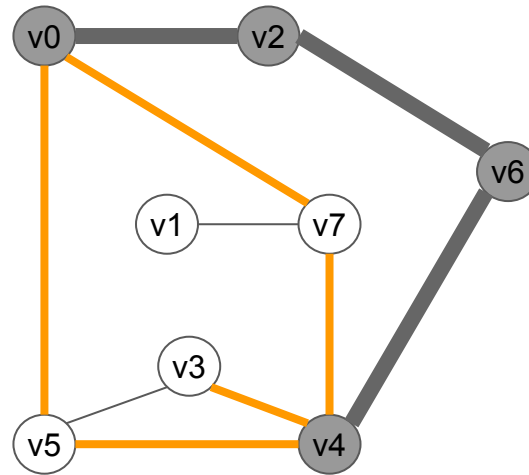


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**



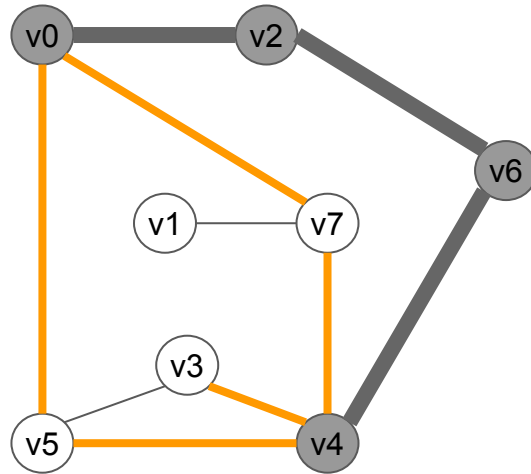
Pilha:



Se não alterarmos a implementação, os vértices **v5** e **v7** vão ser **visitados mais de uma vez!**

Busca em profundidade - Implementação alternativa

Partindo do **v0**



Pilha:



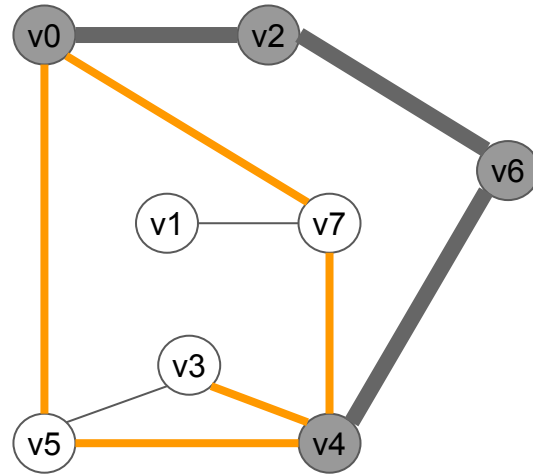
Podemos tratar deste problema em dois momentos:

- **Quando vamos inserir** um vértice na pilha
- **Quando vamos remover** um vértice da pilha

Se não alterarmos a implementação, os vértices **v5** e **v7** vão ser **visitados mais de uma vez!**

Busca em profundidade - Implementação alternativa

Partindo do **v0**

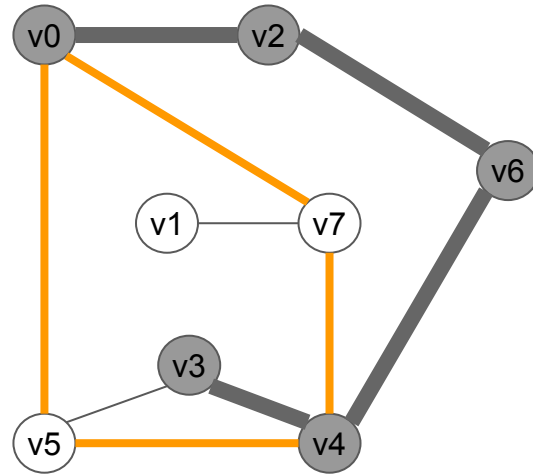


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

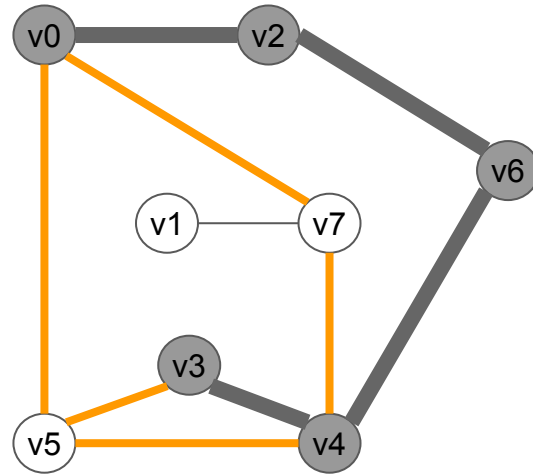


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

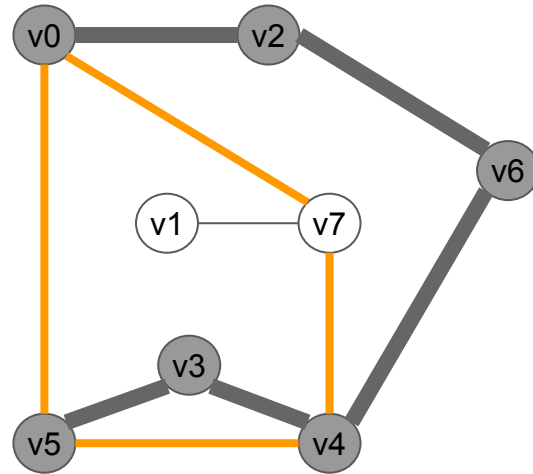


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**

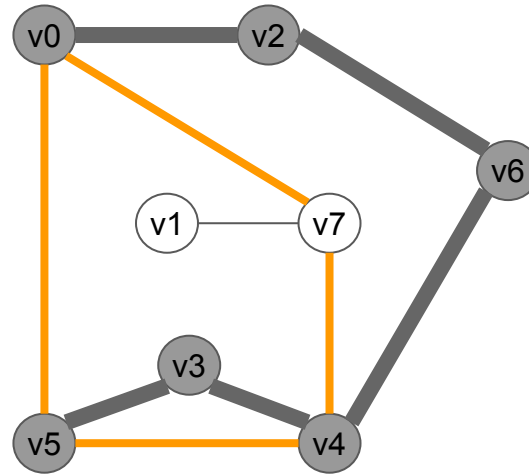


Pilha:



Busca em profundidade - Implementação alternativa

Partindo do **v0**



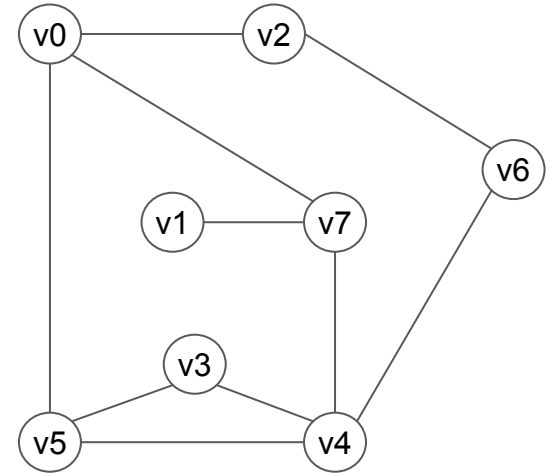
Pilha:



A **primeira vez** que v5 **sai** da pilha corresponde ao **momento correto** em que queremos visitá-lo (o mesmo vale para v7)

Busca em profundidade - Implementação alternativa

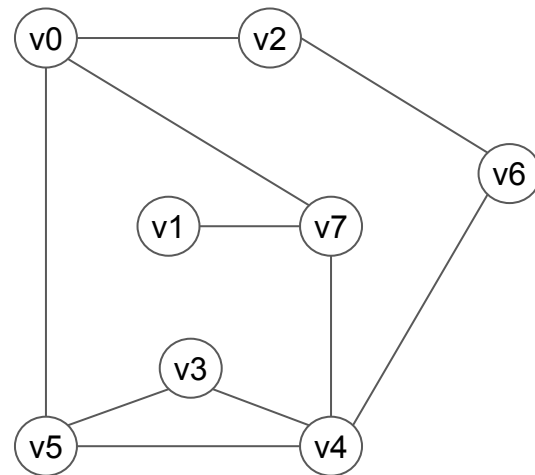
```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
    stack<int> pilha;  
    pilha.push(v);  
    while (!pilha.empty()) {  
        int w = pilha.top();  
        pilha.pop();  
        printf("%d\n", w);  
        marcado[w] = 1;  
        for (int u = (num_vertices_ - 1); u >= 0; u--)  
            if (matriz_adj_[w][u] != 0)  
                if (marcado[u] == 0)  
                    pilha.push(u);  
    }  
}
```



Nesta implementação, um vértice pode ser **visitado mais de uma vez!**

Busca em profundidade - Implementação alternativa

```
void Grafo::busca_prof(int v) {  
    // Criacao e inicializacao do vetor marcado  
    stack<int> pilha;  
    pilha.push(v);  
    while (!pilha.empty()) {  
        int w = pilha.top();  
        pilha.pop();  
        if (marcado[w] == 0) {  
            printf("%d\n", w);  
            marcado[w] = 1;  
            for (int u = (num_vertices_ - 1); u >= 0; u--)  
                if (matriz_adj_[w][u] != 0)  
                    if (marcado[u] == 0)  
                        pilha.push(u);  
        }  
    }  
}
```



Exercícios

- Exercício 2 da Lista de Exercícios “Busca em profundidade e em largura”.

Exercícios

- Exercício 3 da Lista de Exercícios “Busca em profundidade e em largura”.

Exercícios

- Exercício 4 da Lista de Exercícios “Busca em profundidade e em largura”.

Referências

- Esta apresentação é baseada nos seguintes materiais:
 - Capítulo 22 do livro
Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. 3rd. ed. MIT Press, 2009.
 - Capítulo 18 do livro
Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed. Addison-Wesley, 2002.