Existência de um caminho entre dois vértices

Prof. Andrei Braga

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Conteúdo

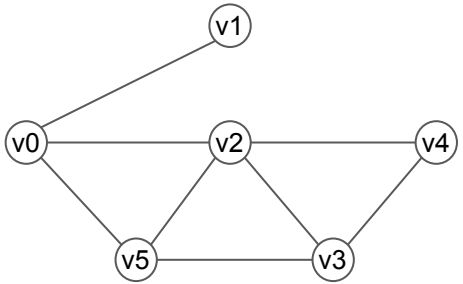
- Estratégia para descobrir se existe um caminho entre dois vértices
- Implementação
- Exercícios
- Referências

Operações em um grafo

- Em uma classe que representa um grafo (como uma matriz de adjacências ou listas de adjacência), podemos implementar operações que determinam propriedades locais do grafo
  Exemplos:
  - Operação: determinar se existe ou não uma aresta entre dois vértices
  - Operação: calcular o grau de um vértice
- Também podemos implementar operações que determinam propriedades globais do grafo
- A seguir, vamos estudar como implementar uma primeira operação deste tipo:
  - Operação: determinar se existe ou não um caminho entre dois vértices

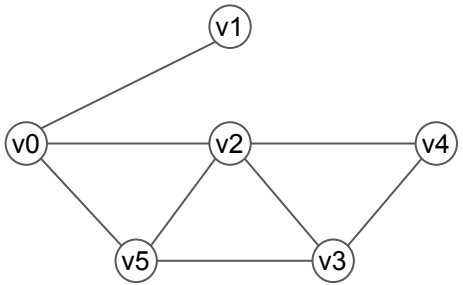
Existência de um caminho entre dois vértices

- Quando estamos processando os valores de uma matriz de adjacências ou de listas de adjacência de um grafo, **não temos** uma **visão global** do grafo
- Em vez disso, temos apenas uma visão local do grafo
- Neste contexto, que estratégia podemos usar para descobrir se existe um caminho entre os vértices v_0 e v_4?



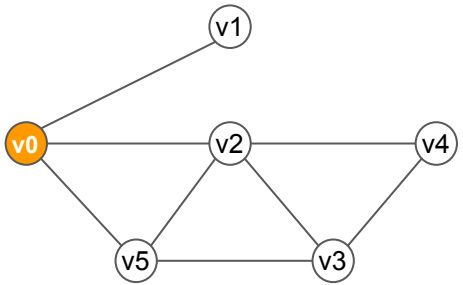
Existência de um caminho entre dois vértices

- Estratégia:
  - Vamos percorrer as arestas do grafo tendo apenas uma visão local do grafo, sem ter uma visão global dele
  - Para escolher o próximo vértice para onde ir, vamos simplesmente escolher o vizinho de menor índice



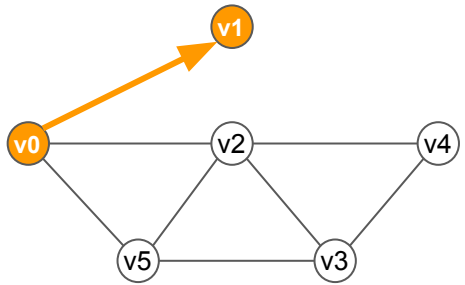
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0



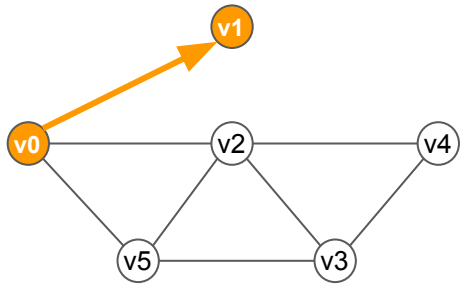
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1



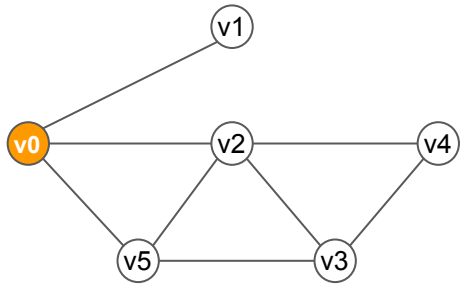
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante



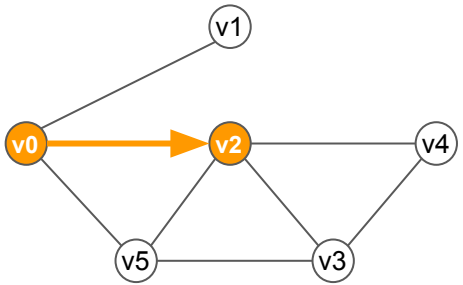
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante
  3. Volte para v_0



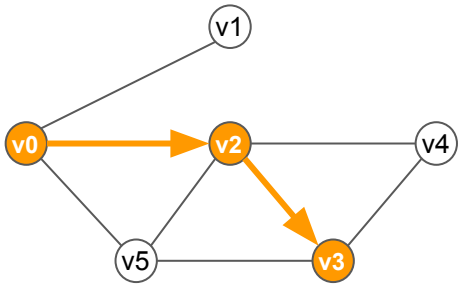
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante
  3. Volte para v_0
  4. Vá para v_2



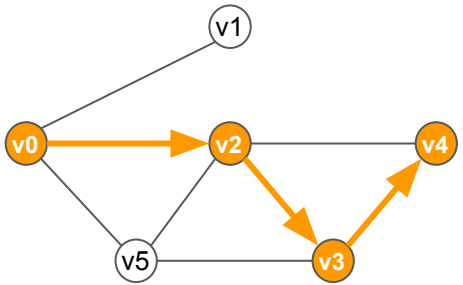
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante
  3. Volte para v_0
  4. Vá para v_2
  5. Vá para v_3



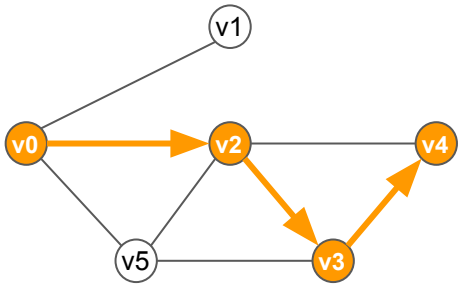
Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante
  3. Volte para v_0
  4. Vá para v_2
  5. Vá para v_3
  6. Vá para v_4



Existência de um caminho entre dois vértices

- Estratégia:
  1. Comece em v_0
  2. Vá para v_1
     Não é possível seguir adiante
  3. Volte para v_0
  4. Vá para v_2
  5. Vá para v_3
  6. Vá para v_4
  7. Existe um caminho entre v_0 e v_4! 🏆👏



Existência de um caminho entre dois vértices

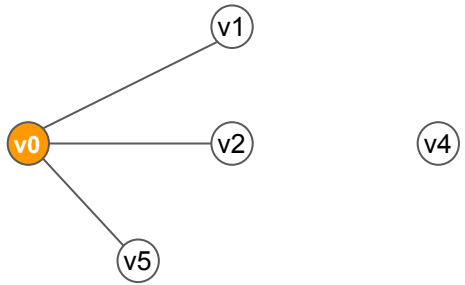
- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4

Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0

Existência de um caminho entre dois vértices

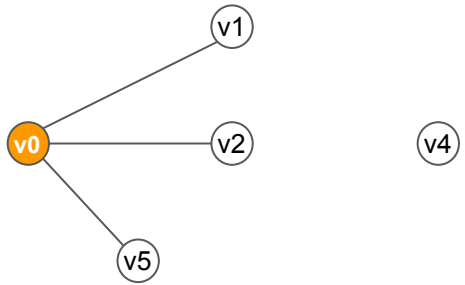
- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0



Existência de um caminho entre dois vértices

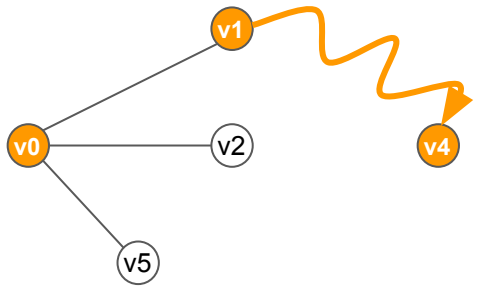
- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0

     Observe que,



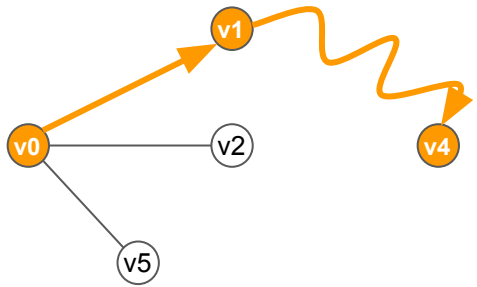
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0
     Observe que,
     se existe um caminho entre
     v_1 e v_4,



Existência de um caminho entre dois vértices

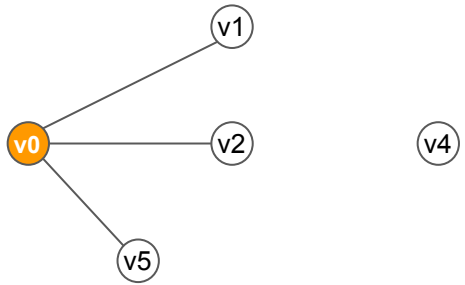
- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0
     Observe que,
     se existe um caminho entre v_1 e v_4,
     então existe um caminho entre v_0 e v_4



Para que este caminho entre v_0 e v_4 seja de fato um caminho, não pode haver vértices repetidos

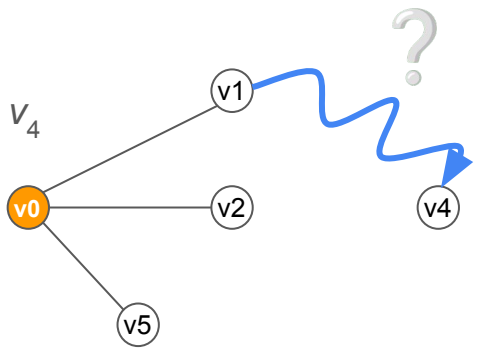
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0



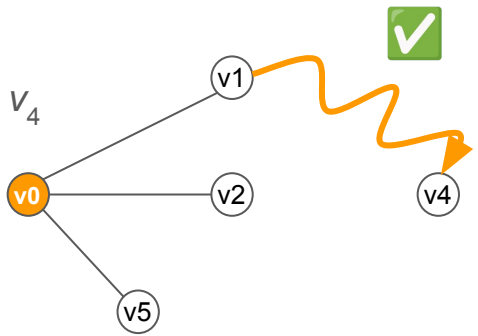
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0
  3. Descubra recursivamente se existe um caminho entre v_1 e v_4



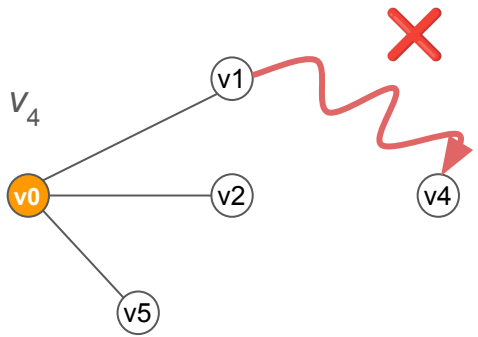
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0
  3. Descubra recursivamente se existe um caminho entre v_1 e v_4
  4. Se sim, existe um caminho entre v_0 e v_4!



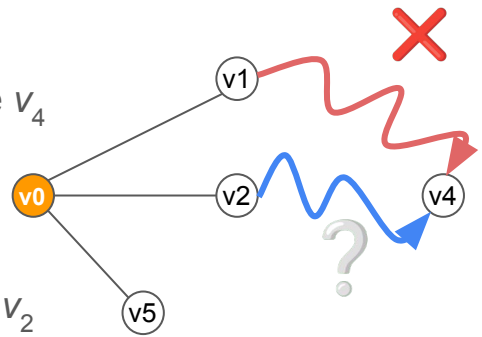
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
  1. Comece em v_0
  2. Considere os vizinhos de v_0
  3. Descubra recursivamente se existe um caminho entre v_1 e v_4
  4. Se sim, existe um caminho entre v_0 e v_4!
  5. Senão,



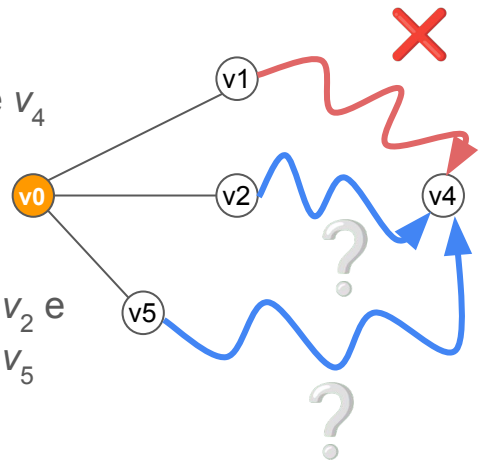
Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
1. Comece em v_0
2. Considere os vizinhos de v_0
3. Descubra recursivamente se existe um caminho entre v_1 e v_4
4. Se sim, existe um caminho entre v_0 e v_4!
5. Senão,
6. faça os passos 3 a 5 para v_2

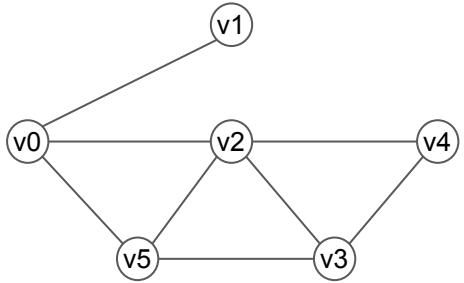


Existência de um caminho entre dois vértices

- Como podemos descrever a estratégia anterior de uma forma geral?
  - Queremos descobrir se existe um caminho entre v_0 e v_4
1. Comece em v_0
2. Considere os vizinhos de v_0
3. Descubra recursivamente se existe um caminho entre v_1 e v_4
4. Se sim, existe um caminho entre v_0 e v_4!
5. Senão,
6. faça os passos 3 a 5 para v_2 e faça os passos 3 a 5 para v_5

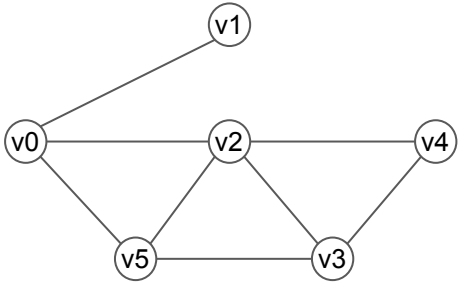


Existência de um caminho entre dois vértices - Implementação



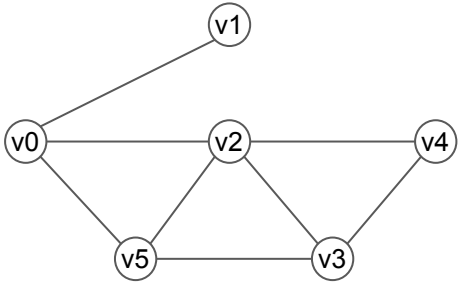

```
bool Grafo::caminho(int v, int w) {



}
```

Existência de um caminho entre dois vértices - Implementação



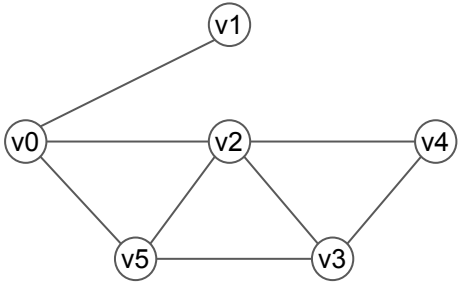

```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)



}
```

Existência de um caminho entre dois vértices - Implementação




```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;

}
```

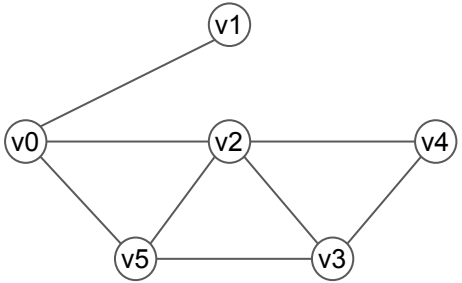
Existência de um caminho entre dois vértices - Implementação




```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```

Existência de um caminho entre dois vértices - Implementação

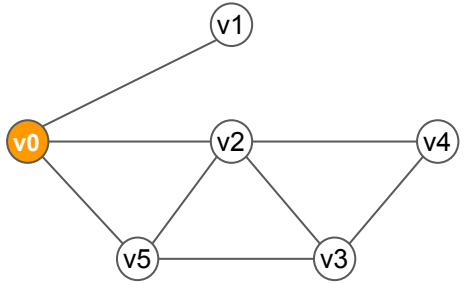
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

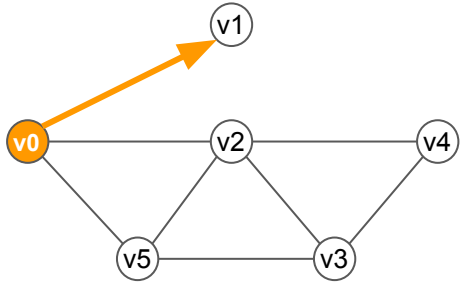
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

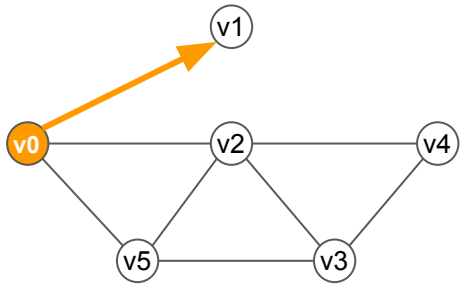
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

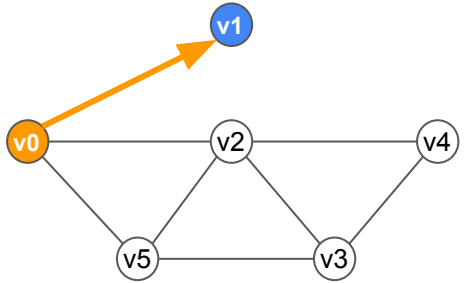
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

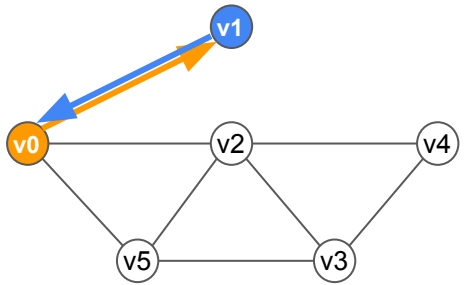
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

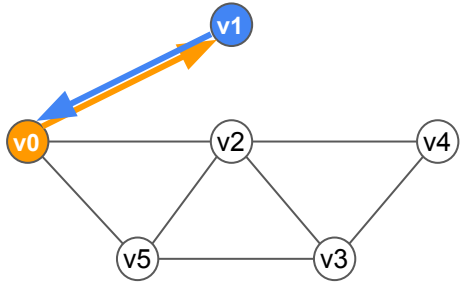
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)
  caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

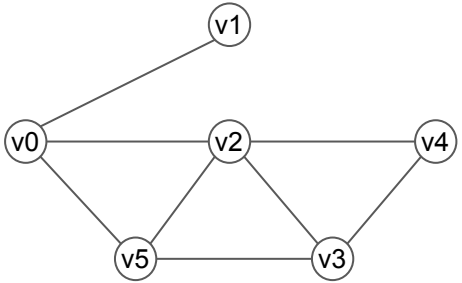
```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```



caminho(0, 4)
  caminho(1, 4)

Nesta implementação, estamos considerando **caminhos onde vértices podem se repetir!**

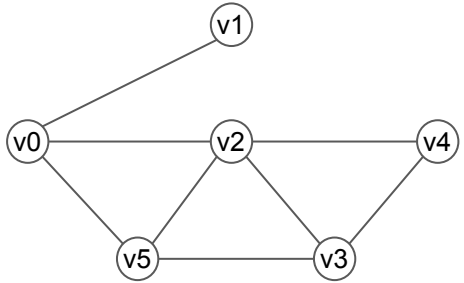
Existência de um caminho entre dois vértices - Implementação




```
bool Grafo::caminho(int v, int w) {
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (caminho(u, w))
                return true;
    return false;
}
```

Existência de um caminho entre dois vértices - Implementação

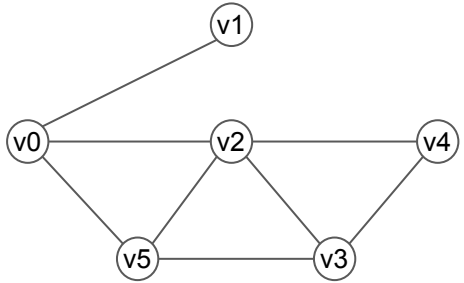
```
bool Grafo::caminho(int v, int w) {

    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)

                if (caminho(u, w))
                    return true;
    return false;
}
```



Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

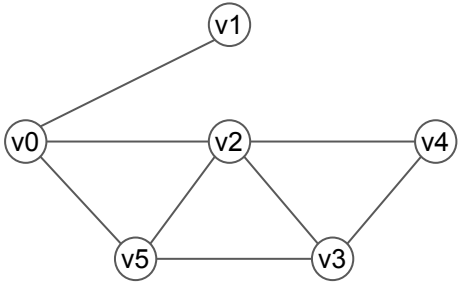
```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```



Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

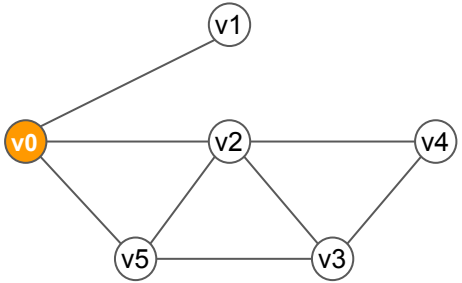


caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

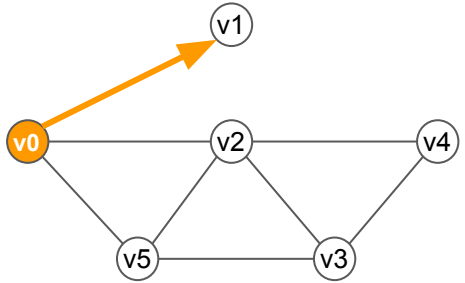


caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

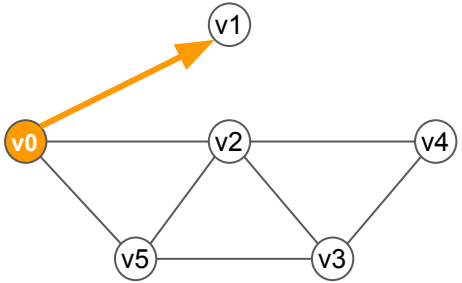


caminho(0, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

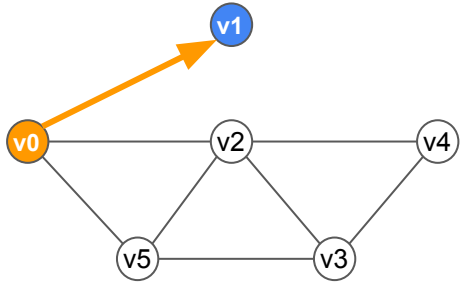


caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

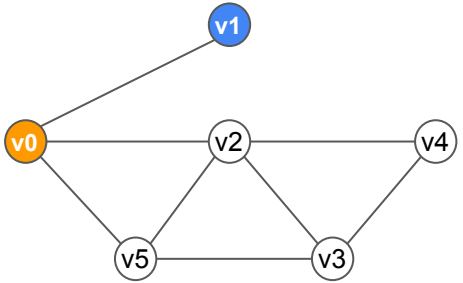


caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

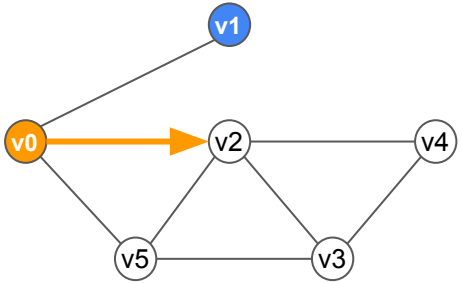


caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

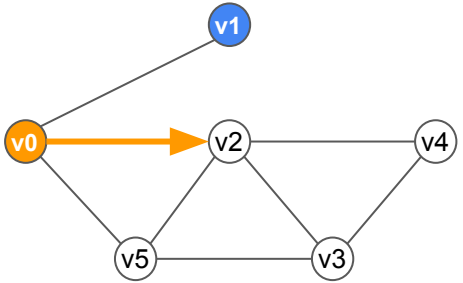


caminho(0, 4)
caminho(1, 4)

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

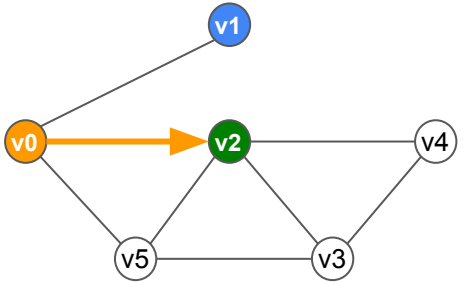



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

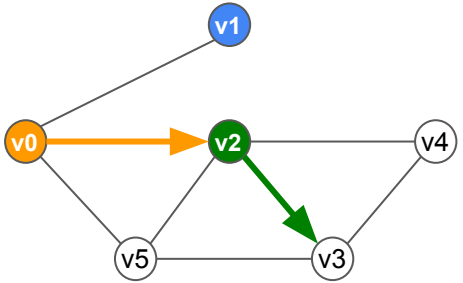



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

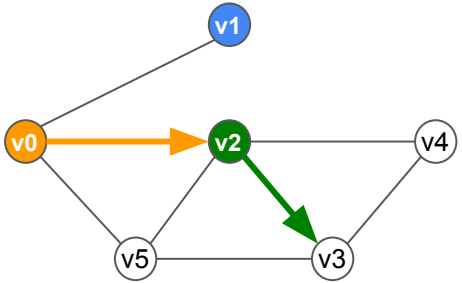



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

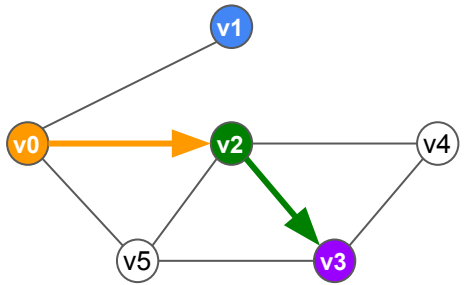



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

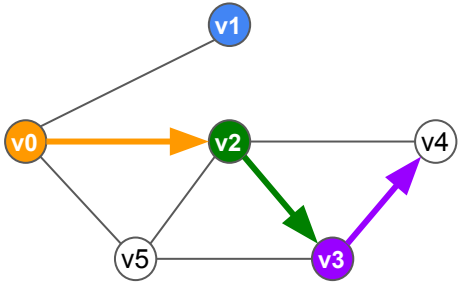



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

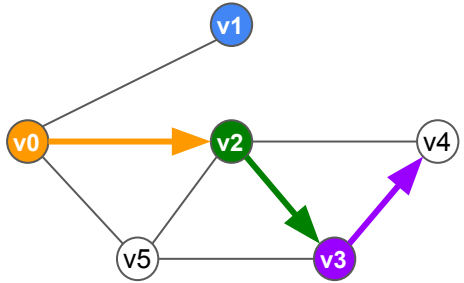



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

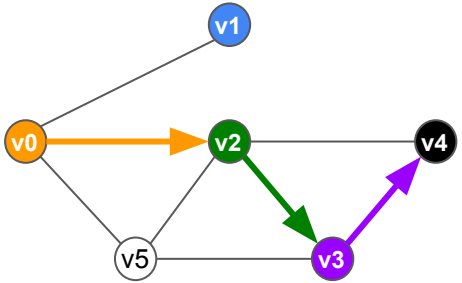



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
      caminho(4, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

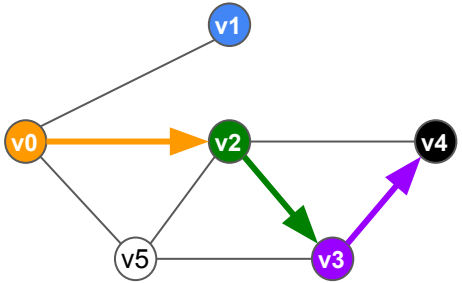



```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
      caminho(4, 4)
```

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```



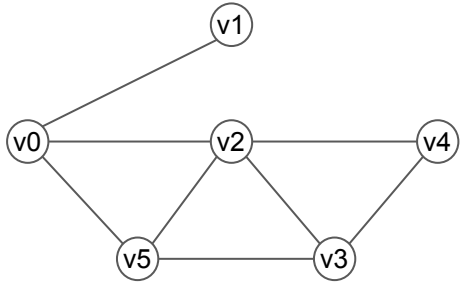

```
caminho(0, 4)
  caminho(1, 4)
  caminho(2, 4)
    caminho(3, 4)
      caminho(4, 4)
```

Em uma chamada **caminho(v, v)**, o retorno será **false**, mas deveria ser `true`!

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

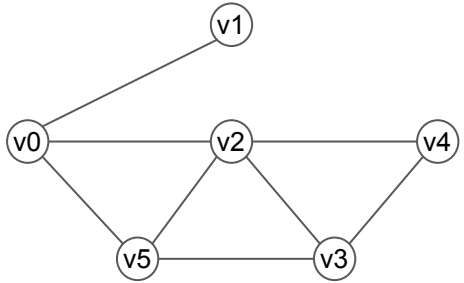
```
bool Grafo::caminho(int v, int w, int marcado[]) {
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```



Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

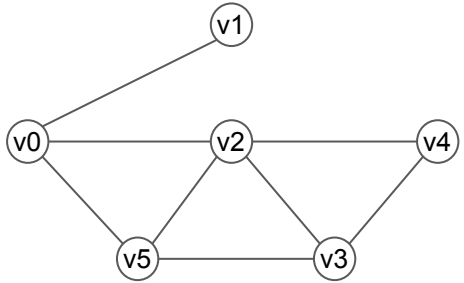
```
bool Grafo::caminho(int v, int w, int marcado[]) {


    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```



Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

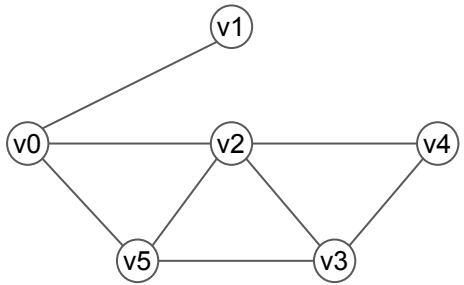
```
bool Grafo::caminho(int v, int w, int marcado[]) {
    if (v == w)
        return true;
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```



Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    if (v == w)
        return true;
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))
                    return true;
    return false;
}
```

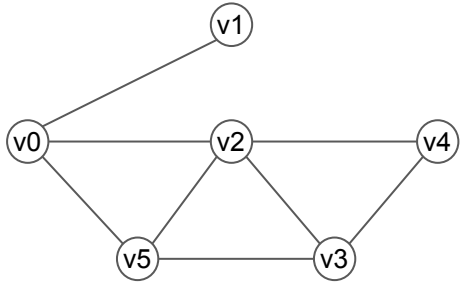


Como imprimir o caminho encontrado?

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    if (v == w)

        return true;

    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado))

                    return true;

    return false;
}
```

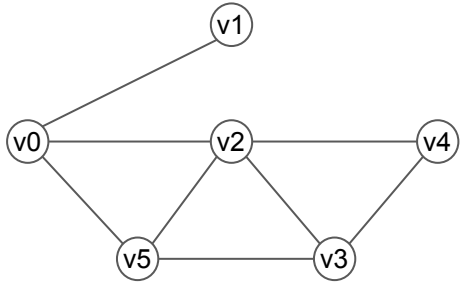


Como imprimir o caminho encontrado?

Existência de um caminho entre dois vértices - Implementação

```
// O vetor marcado eh criado e inicializado antes
// do metodo caminho ser chamado
```

```
bool Grafo::caminho(int v, int w, int marcado[]) {
    if (v == w) {
        printf("%d-", v);
        return true;
    }
    marcado[v] = 1;
    for (int u = 0; u < num_vertices_; u++)
        if (matriz_adj_[v][u] != 0)
            if (marcado[u] == 0)
                if (caminho(u, w, marcado)) {
                    printf("%d-", v);
                    return true;
                }
    return false;
}
```



Como imprimir o caminho encontrado?

Os vértices do caminho estão sendo impressos na **direção de w para v**

Exercícios

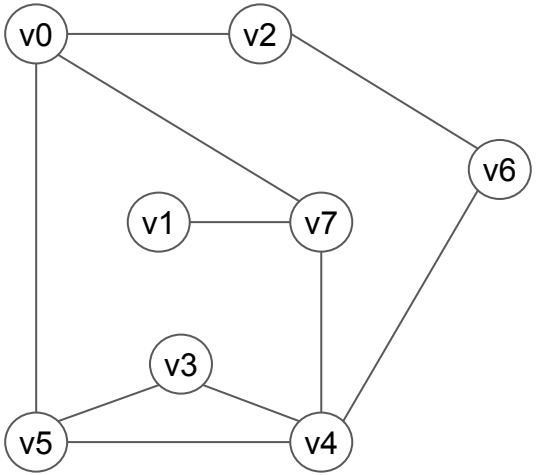
1. Modifique o método `caminho` para imprimir uma linha contendo

   `caminho(v, w)`

   sendo v e w os vértices recebidos por parâmetro pelo método. Faça isto de modo que, para uma chamada recursiva de um nível mais profundo, a linha acima seja impressa com um nível de indentação a mais em relação à linha impressa para a chamada recursiva do nível anterior.
   Veja o exemplo deste slide.

Exercícios

2. Execute o método implementado no Exercício 1 para o grafo abaixo recebendo como parâmetro os vértices v0 e v7.



Referências

- Esta apresentação é baseada nos seguintes materiais:
  - Capítulo 17 do livro
    Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed.
    Addison-Wesley, 2002.