Busca em Largura, Caminhos e Distâncias

Prof. Andrei Braga

UNIVERSIDADE FEDERAL DA FRONTEIRA SUL

Conteúdo

Busca em largura

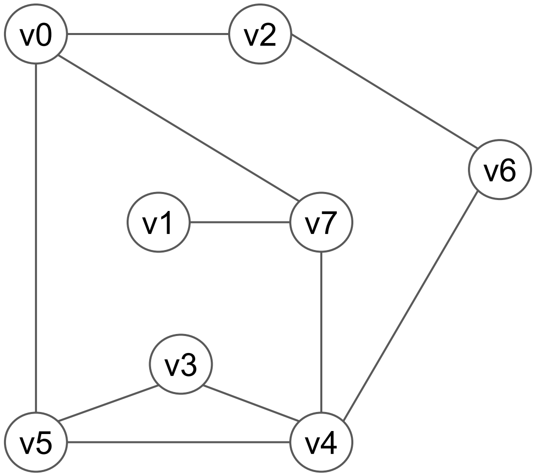
- Falamos antes sobre o seguinte objetivo:
  Dado um grafo, queremos determinar um caminho de comprimento mínimo entre um certo vértice e cada um dos vértices do grafo
- Podemos atingir este objetivo através de uma estratégia de busca chamada **busca em largura**
- Para este objetivo, o algoritmo de busca em profundidade não é útil, pois a estratégia utilizada não tem relação com calcular caminhos de comprimento mínimo

- Em uma busca em largura, vamos percorrer o grafo da seguinte maneira: vamos **visitar primeiro** os vértices **mais próximos** do vértice inicial

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial



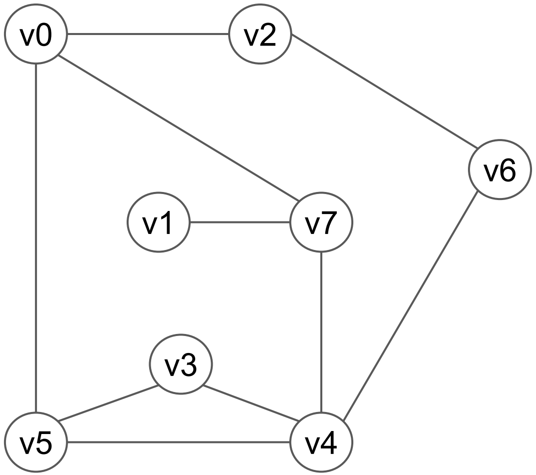
Estrutura de dados:

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



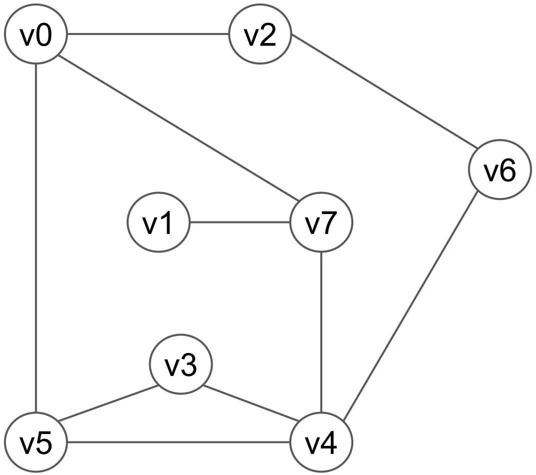
Estrutura de dados:

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



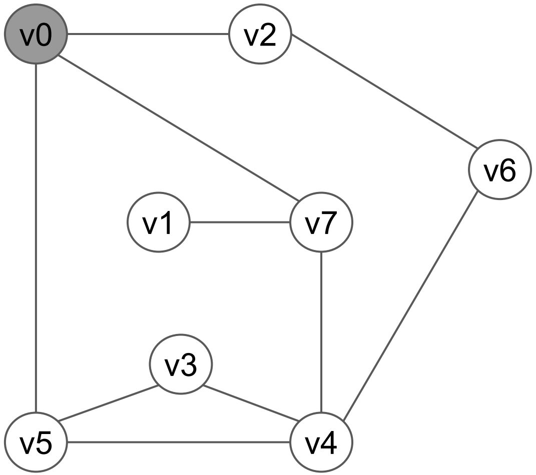
Estrutura de dados: v0

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



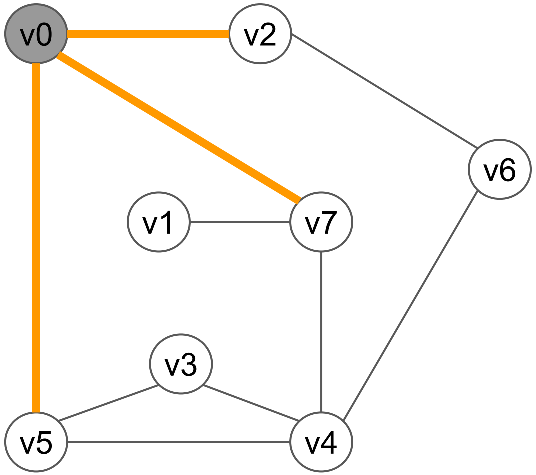
Estrutura de dados:

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



Estrutura de dados:

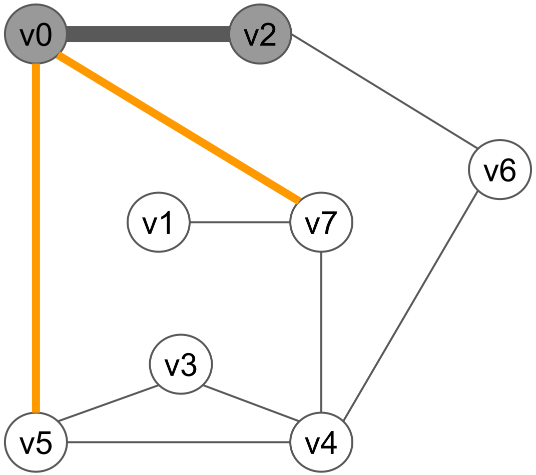
v2 v5 v7

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



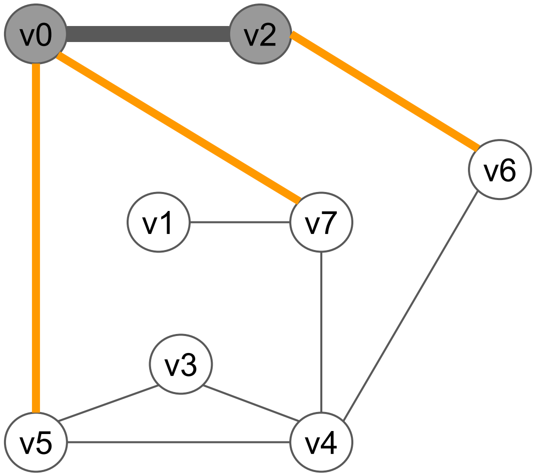
Estrutura de dados: v5 v7

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



Estrutura de dados:

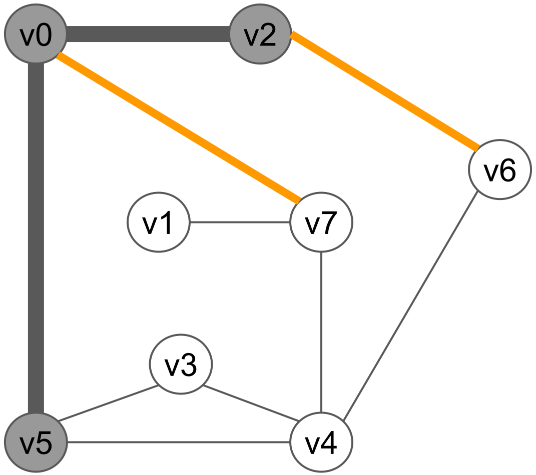
v5 v7 v6

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



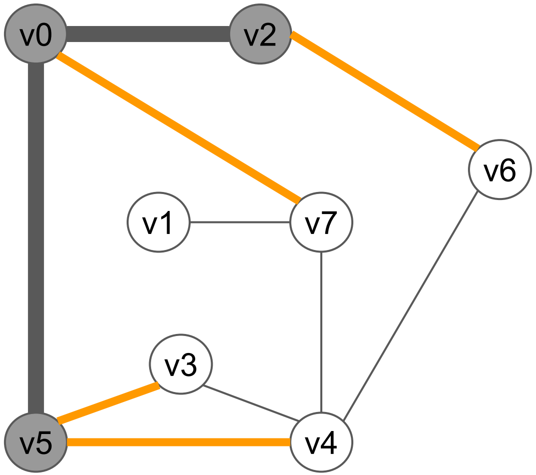
Estrutura de dados: v7 v6

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



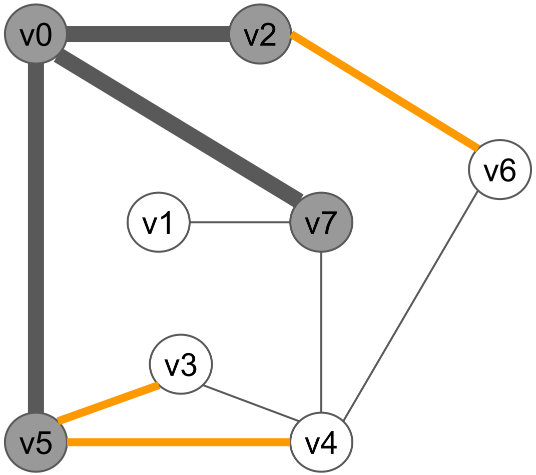
Estrutura de dados: v7 v6 v3 v4

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



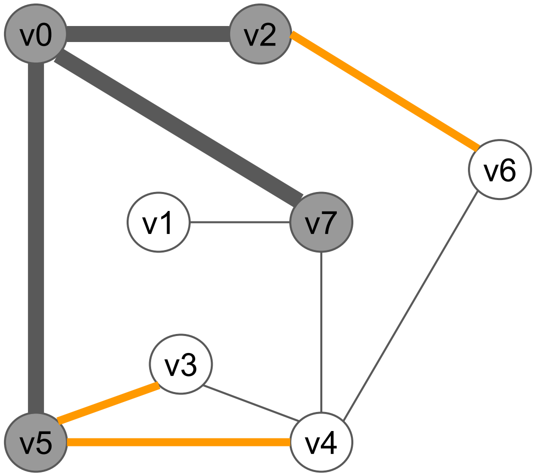
Estrutura de dados: v6 v3 v4

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



Nesta busca, também temos que **evitar** que **um vértice seja visitado mais de uma vez**

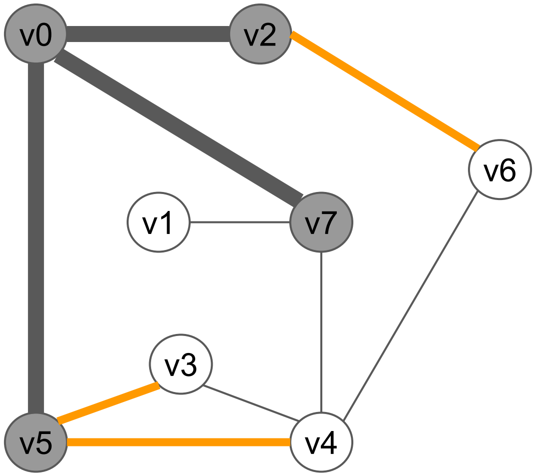
Estrutura de dados: v6 v3 v4

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



A **primeira vez** que v4 **entra** na estrutura de dados corresponde ao **momento correto** em que queremos visitá-lo

Por isso, vamos **evitar** que **um vértice** seja **inserido mais de uma vez** na **estrutura de dados**

Estrutura de dados:

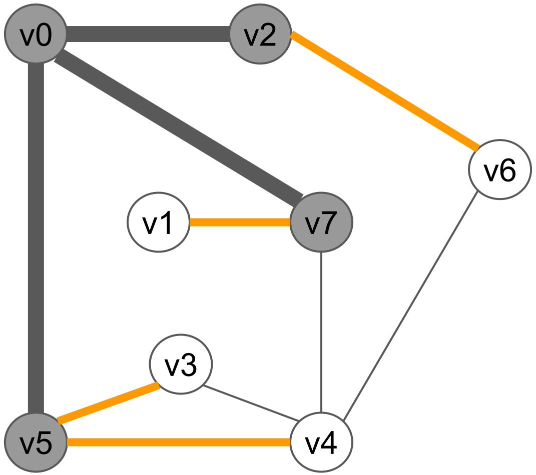
v6 v3 v4

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



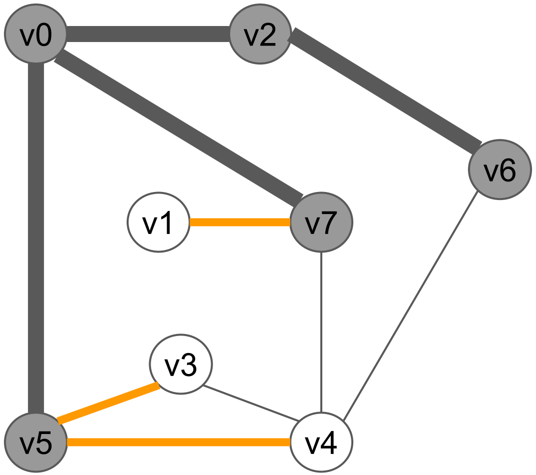
Estrutura de dados: v6 v3 v4 v1

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



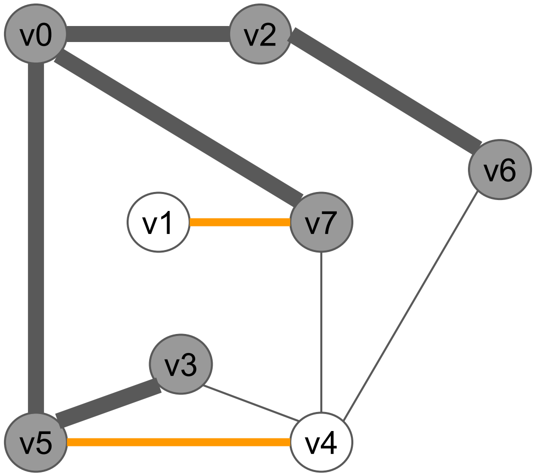
Estrutura de dados: v3 v4 v1

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



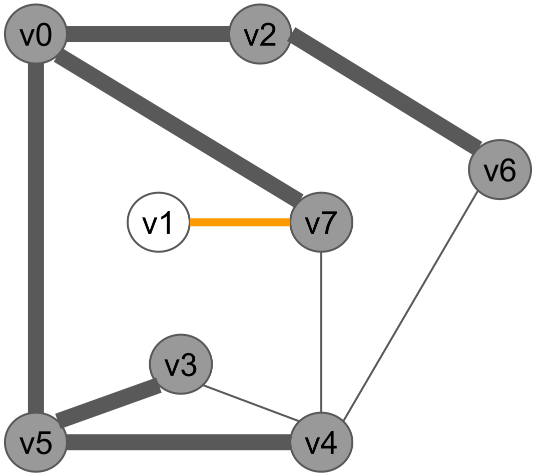
Estrutura de dados: v4 v1

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



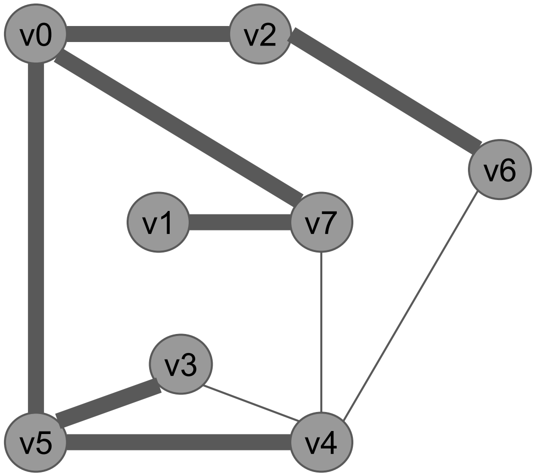
Estrutura de dados: v1

Busca em largura

Vamos usar a **estratégia geral de busca** vista anteriormente

Vamos percorrer o grafo **visitando primeiro** os vértices **mais próximos** do vértice inicial

Partindo do **v0**



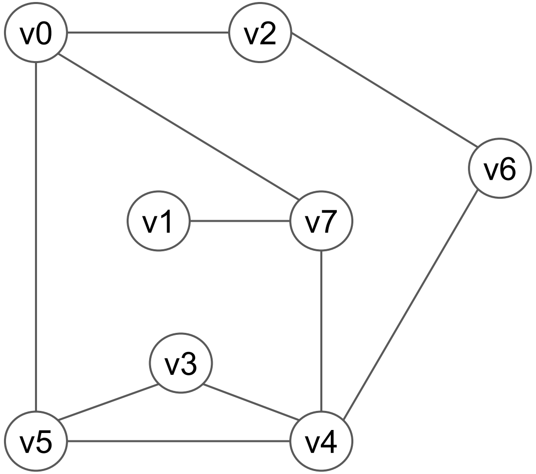
Estrutura de dados:

Busca em largura - Implementação

- Considerando a estratégia geral de busca vista anteriormente, o que podemos dizer da lógica através da qual os vértices são visitados no algoritmo de busca em largura?
    - É uma lógica de **fila**
- Então, vamos implementar este algoritmo usando uma fila

Busca em largura - Implementação

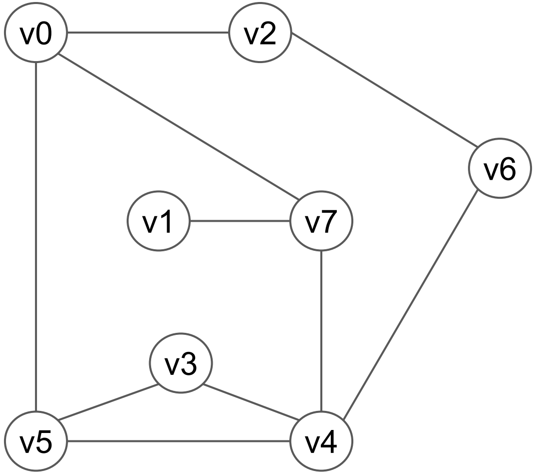
```
void Grafo::busca_larg(int v) {
    // Criacao e inicializacao do vetor marcado
    queue<int> fila;
    fila.push(v);
    while (!fila.empty()) {
        int w = fila.front();
        fila.pop();
        printf("%d\n", w);
        marcado[w] = 1;
        for (int u = 0; u < num_vertices_; u++)
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



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

Busca em largura - Implementação

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

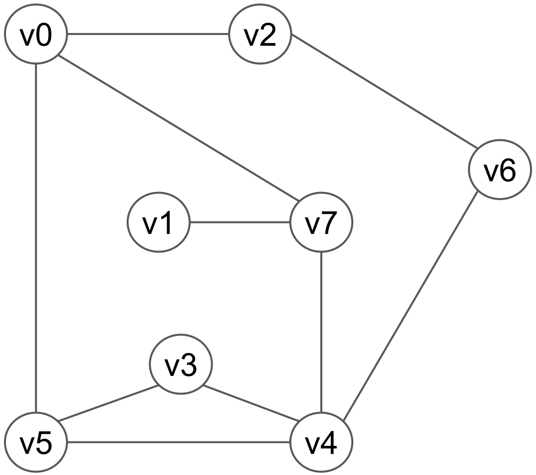


Busca em largura - Dinâmica

- Assim como fizemos para a estratégia de busca anterior, vamos construir um grafo H que representa a dinâmica desta segunda estratégia de busca
  - 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

Busca em largura - Dinâmica

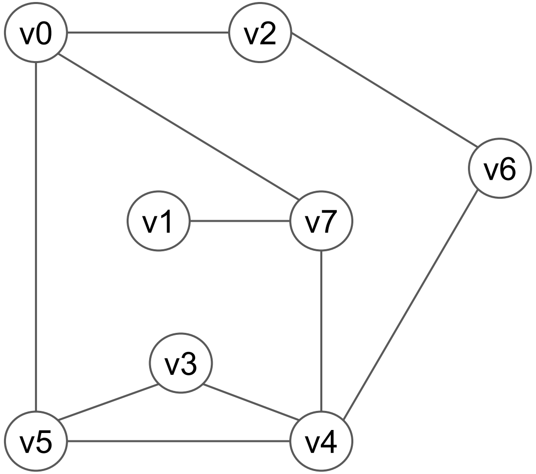
Partindo do **v0**



Estrutura de dados:

Busca em largura - Dinâmica

Partindo do **v0**



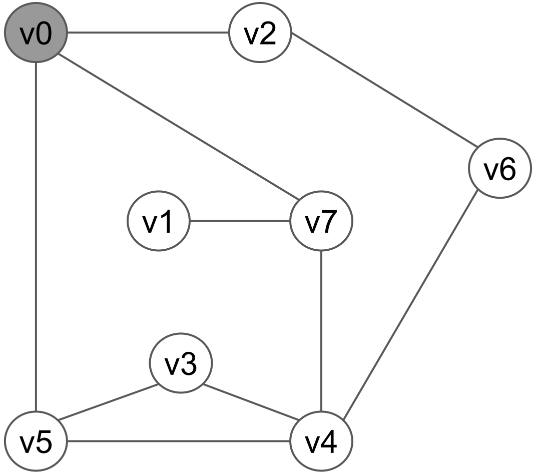
Estrutura de dados:

v0

Busca em largura - Dinâmica

H: 

Partindo do **v0**

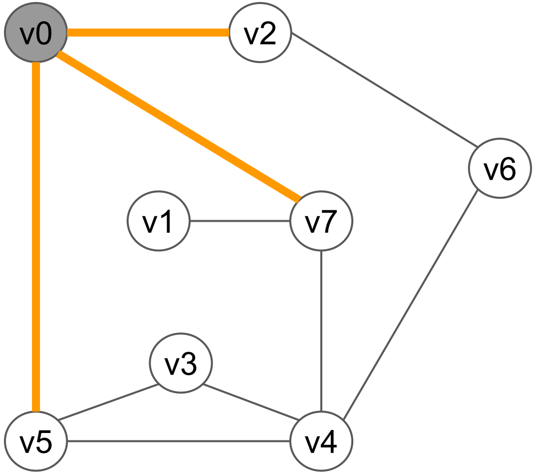


Estrutura de dados:

Busca em largura - Dinâmica

H:

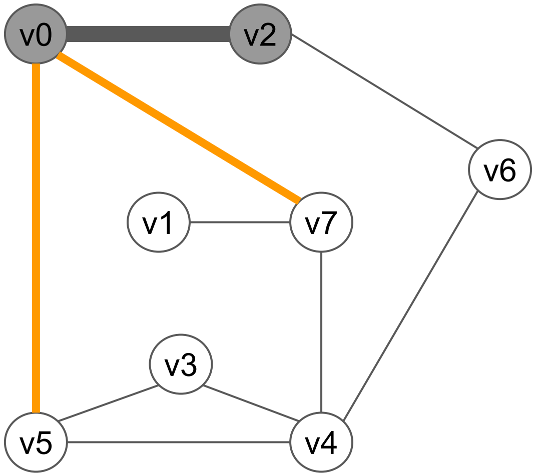
Partindo do **v0**



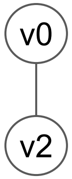
Estrutura de dados: v2 v5 v7

Busca em largura - Dinâmica

Partindo do **v0**



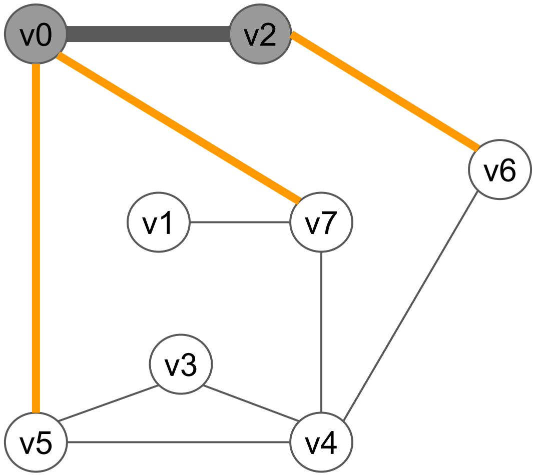
H:



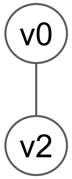
Estrutura de dados: v5 v7

Busca em largura - Dinâmica

Partindo do **v0**



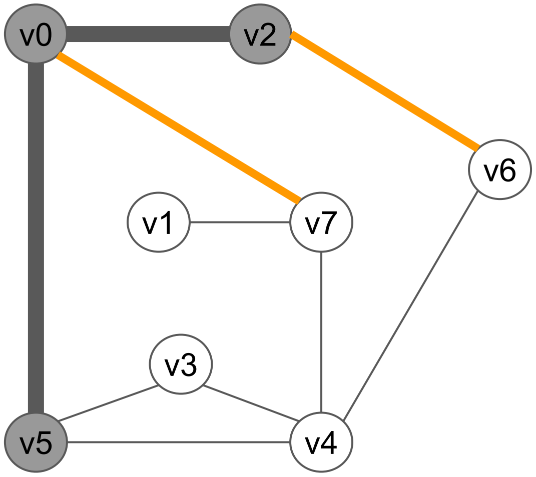
H:



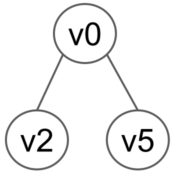
Estrutura de dados: v5 v7 v6

Busca em largura - Dinâmica

Partindo do **v0**



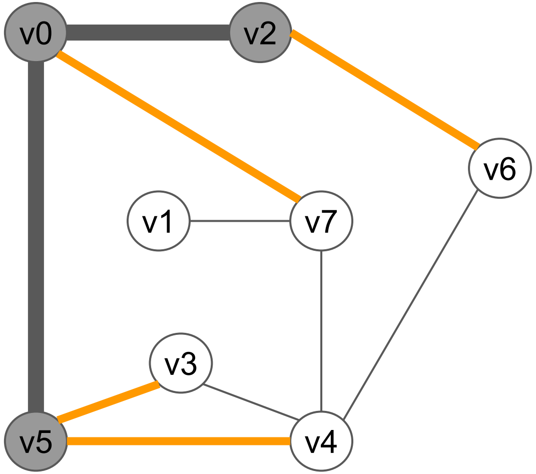
H:



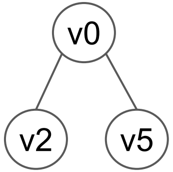
Estrutura de dados: v7 v6

Busca em largura - Dinâmica

Partindo do **v0**



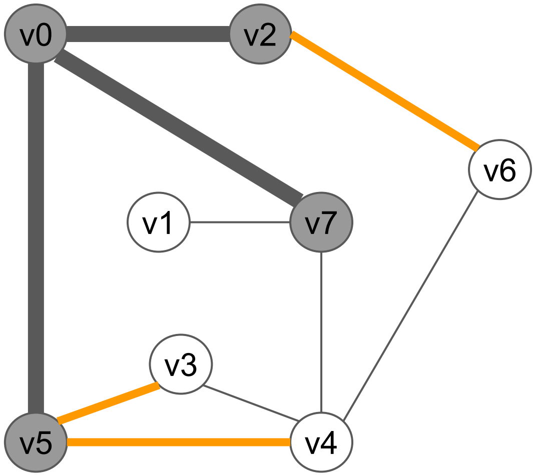
H:



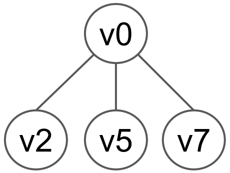
Estrutura de dados: v7 v6 v3 v4

Busca em largura - Dinâmica

Partindo do **v0**



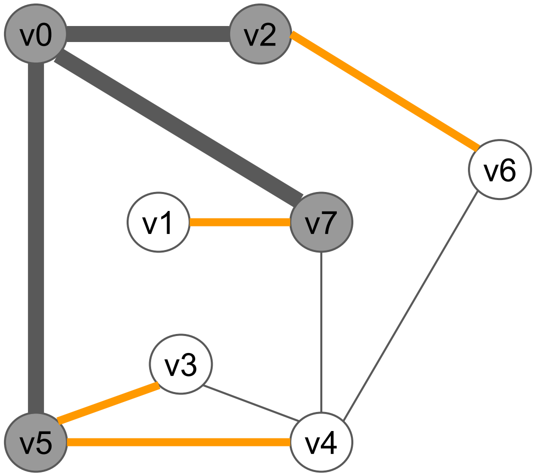
H:



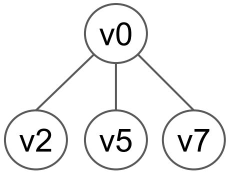
Estrutura de dados: v6 v3 v4

Busca em largura - Dinâmica

Partindo do **v0**



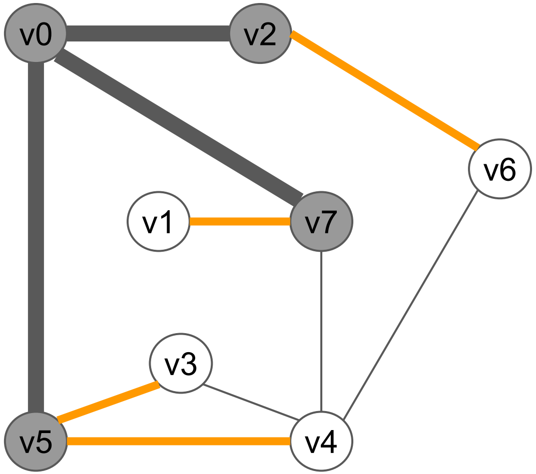
H:



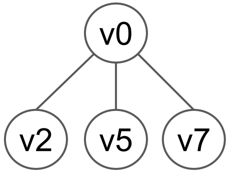
Estrutura de dados: v6 v3 v4 v1

Busca em largura - Dinâmica

Partindo do **v0**



H:

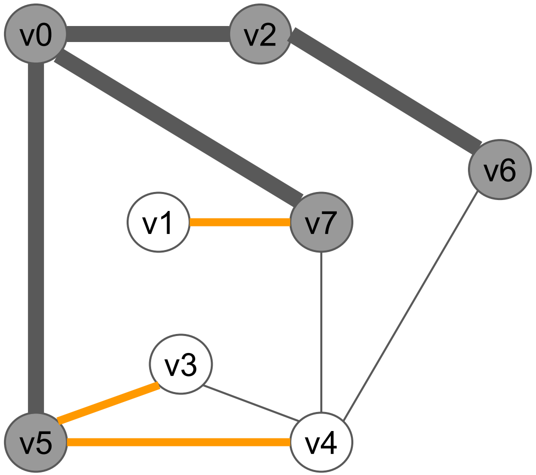


A busca segue em **largura** até não ser mais possível, para depois se aprofundar

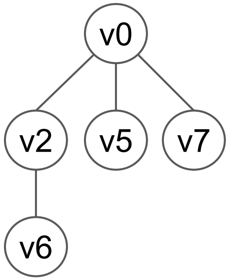
Estrutura de dados: v6 v3 v4 v1

Busca em largura - Dinâmica

Partindo do **v0**



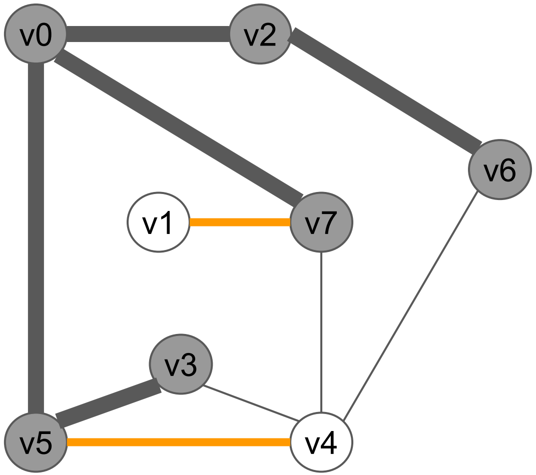
H:



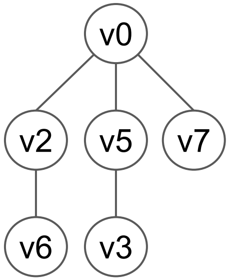
Estrutura de dados: v3 v4 v1

Busca em largura - Dinâmica

Partindo do **v0**



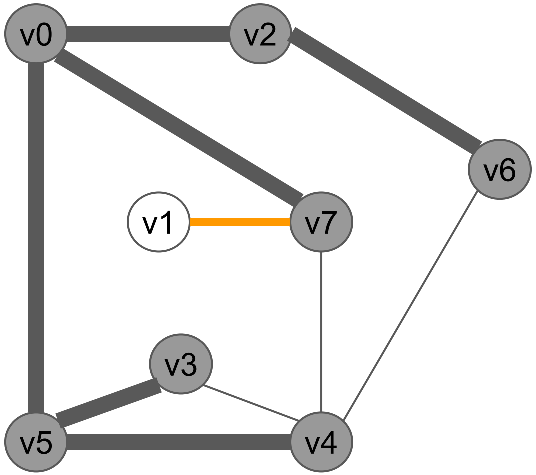
H:



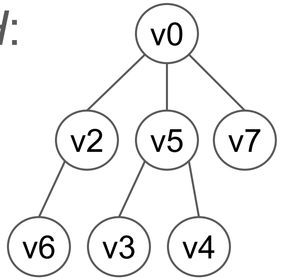
Estrutura de dados: v4 v1

Busca em largura - Dinâmica

Partindo do **v0**



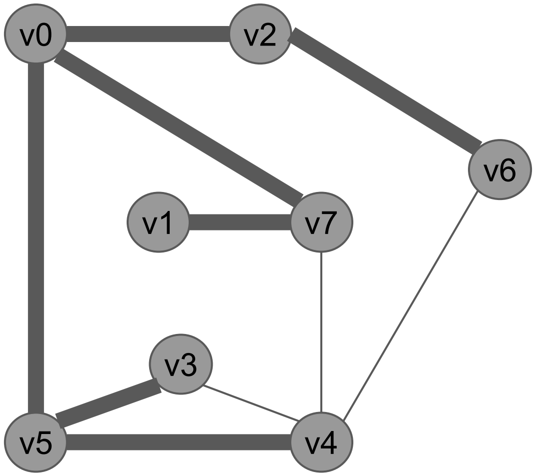
H:



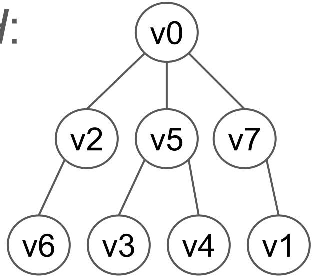
Estrutura de dados: v1

Busca em largura - Dinâmica

Partindo do **v0**



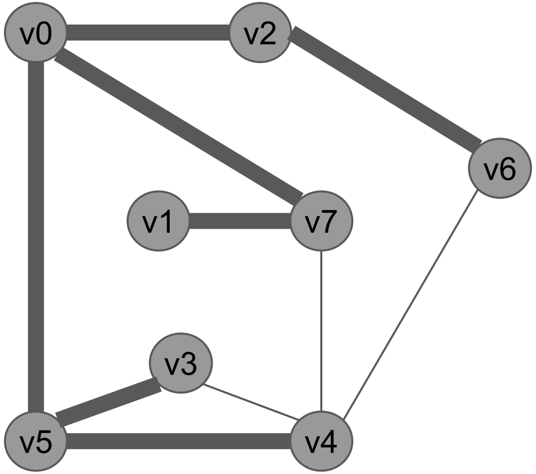
H:



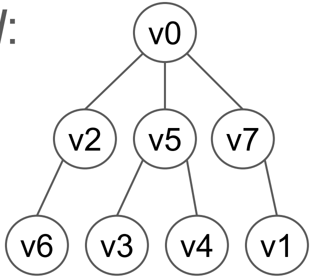
Estrutura de dados:

Busca em largura - Dinâmica

Partindo do **v0**



H:



H é uma **árvore**

Estrutura de dados:

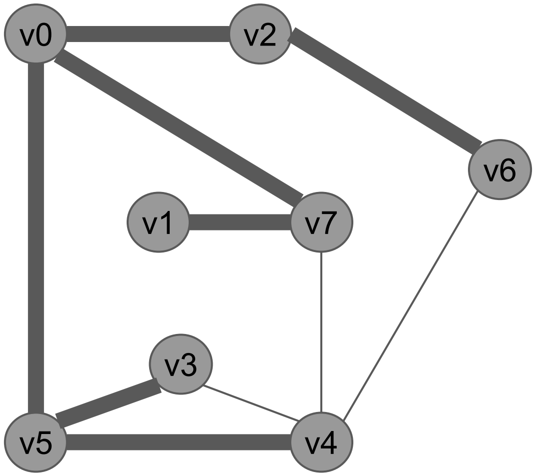
Busca em largura

- Usando o algoritmo de busca em largura, como podemos determinar um caminho de comprimento mínimo entre um dado vértice e cada um dos vértices de um grafo?

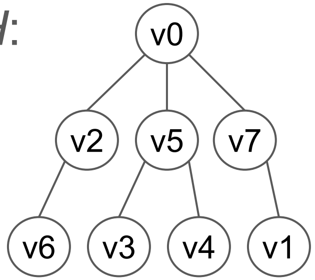
- É possível provar o seguinte teorema:
- **Teorema:** Considere a árvore correspondente à dinâmica do algoritmo de busca em largura executada em um grafo G partindo de um vértice v. Para qualquer vértice u desta árvore, o caminho entre v e u na árvore é um caminho de comprimento mínimo entre v e u em G.
- Para a prova do teorema, veja o capítulo relativo a busca em largura da Ref. 1 desta apresentação

Busca em largura - Dinâmica

Partindo do **v0**



H:



Estrutura de dados:

Busca em largura

- Portanto, a partir da árvore que corresponde à dinâmica do algoritmo de busca em largura, podemos determinar um caminho de comprimento mínimo entre o **vértice inicial da busca** e cada um dos vértices do grafo

- E como podemos obter esta árvore?
- Vamos estender a implementação do algoritmo de busca em largura para que seja gerada uma representação desta árvore
- Vamos representar a árvore através de um **vetor** pai

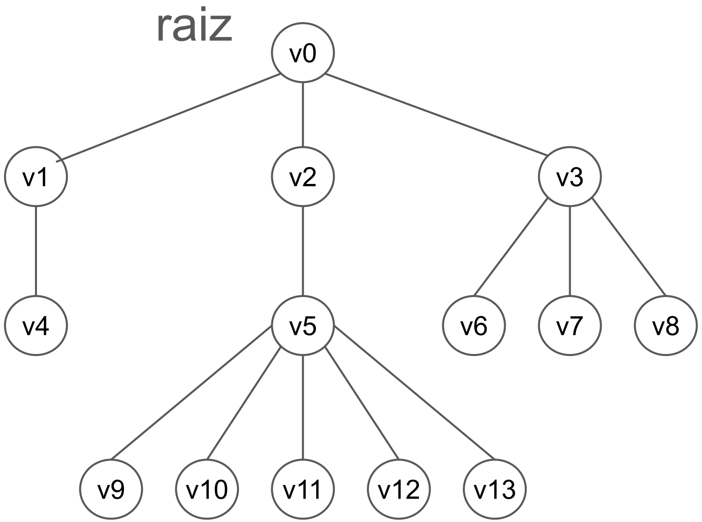
Representação computacional de uma árvore (revisão)

- Uma árvore é um grafo e, portanto, pode ser representada como uma matriz de adjacências ou listas de adjacência ou de outra forma usual de representar um grafo
- Além disso, uma árvore pode ser representada como uma estrutura mais simples

Representação computacional de uma árvore (revisão)

- Dada uma árvore com raiz *r*, se a última aresta do caminho entre o vértice *r* e um vértice *v* na árvore é a aresta *uv*, então dizemos que *u* é o **pai** de *v* e *v* é um **filho** de *u*

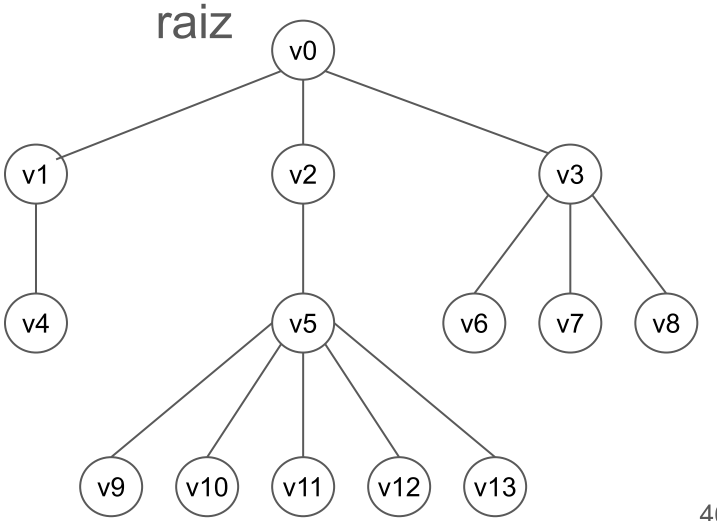
- Exemplo:
    - v2 é filho de v0
    - v5 é pai de v10
    - v13 é filho de v5
    - v6 não é pai de v7 (v6 é **irmão** de v7)



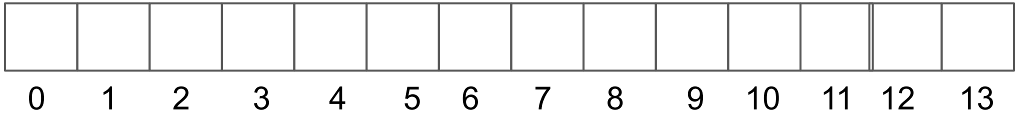
Representação computacional de uma árvore (revisão)

- Podemos representar uma árvore *G* com raiz *r* como um **vetor** *pai* de $|V(G)|$ elementos, com índices 0, 1, …, $|V(G)|$ - 1, tal que
  - *pai*[*i*] é igual ao pai do vértice *i* em *G* caso $i \neq r$ e
  - *pai*[*r*] = -1

- Exemplo:



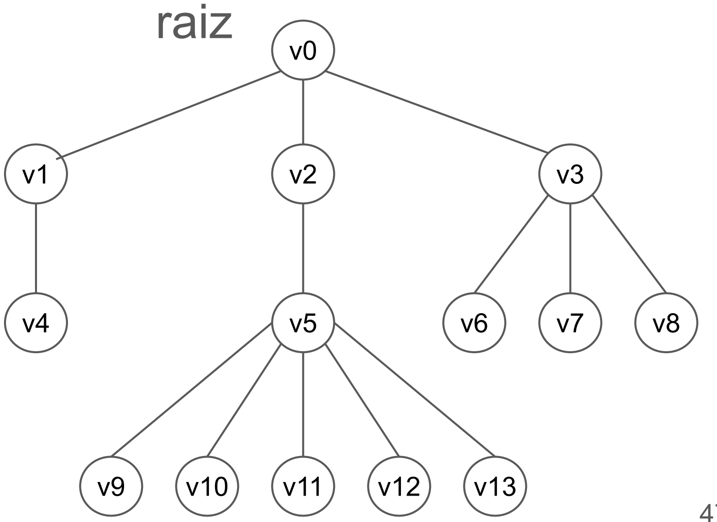
pai



Representação computacional de uma árvore (revisão)

- Podemos representar uma árvore G com raiz r como um **vetor** *pai* de $|V(G)|$ elementos, com índices 0, 1, …, $|V(G)| - 1$, tal que
  - *pai*[i] é igual ao pai do vértice i em G caso $i \neq r$ e
  - *pai*[r] = -1

- Exemplo:



pai

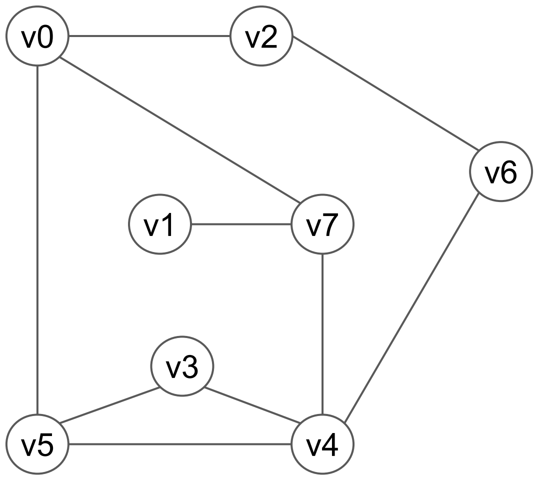
-1	0	0	0	1	2	3	3	3	5	5	5	5	5
0	1	2	3	4	5	6	7	8	9	10	11	12	13

Busca em largura - Representação da árvore de busca

- Lembrando da dinâmica da busca, podemos entender como preencher um vetor *pai* que representa a árvore correspondente

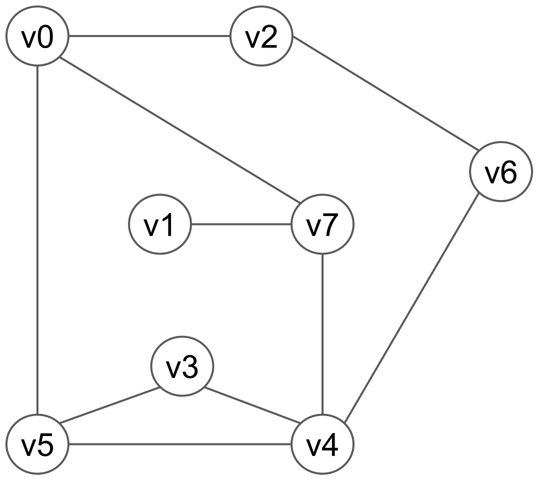
Busca em largura - Implementação

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



Busca em largura - Implementação

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

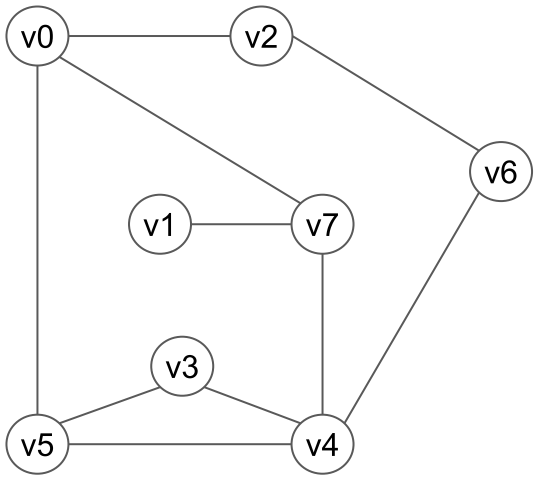


Busca em largura

- Além de determinar caminhos de comprimento mínimo, podemos, durante uma busca em largura, já determinar a **distância** entre o vértice inicial da busca e cada um dos vértices do grafo

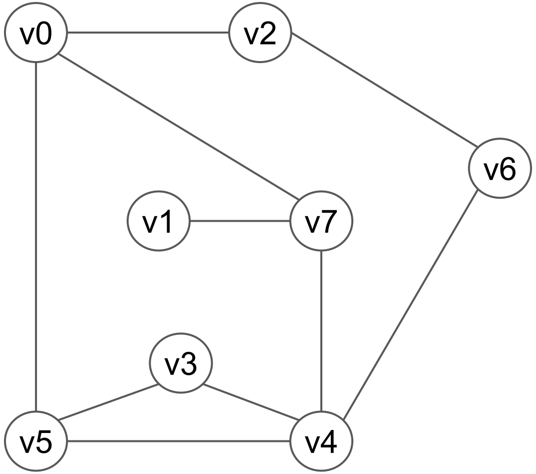
Busca em largura - Implementação

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



Busca em largura - Implementação

```
void Grafo::busca_larg(int v, int pai[], int dist[]) {
    // Criacao e inicializacao do vetor marcado
    // Inicializacao dos vetores pai e dist
    queue<int> fila;
    marcado[v] = 1;
    pai[v] = -1;
    dist[v] = 0;
    fila.push(v);
    while (!fila.empty()) {
        int w = fila.front();
        fila.pop();
        printf("%d\n", w);
        for (int u = 0; u < num_vertices_; u++)
            if (matriz_adj_[w][u] != 0)
                if (marcado[u] == 0) {
                    marcado[u] = 1;
                    pai[u] = w;
                    dist[u] = dist[w] + 1;
                    fila.push(u);
                }
    }
}
```



Exercícios

- Exercício 5 da Lista de Exercícios “Busca em profundidade e em largura”.

Exercícios

- Exercício 6 da Lista de Exercícios “Busca em profundidade e em largura”.

Exercícios

- Demais exercícios da Lista de Exercícios “Busca em profundidade e em largura”.

Referências

- Esta apresentação é baseada nos seguintes materiais:

    1. Capítulo 22 do livro
       Cormen, T. H., Leiserson, C. E., Rivest, R. L., Stein, C. Introduction to Algorithms. 3rd. ed. MIT Press, 2009.
    2. Capítulo 18 do livro
       Sedgewick, R. Algorithms in C++ – Part 5. Graph Algorithms. 3rd. ed. Addison-Wesley, 2002.