Grafos - Métodos de Percorrimento

Prof. Luiz Gustavo Almeida Martins

Percorrimento de um grafo

- Vários problemas relacionados a grafos envolvem a **obtenção de informações sobre sua estrutura**
  - Envolve uma **forma sistemática para explorar os vértices do grafo**
    - Exploração completa (precisa visitar todos os vértices)
    - Exploração parcial (apenas um subconjunto de vértices precisa ser visitado)
  - Define **como caminhar pelos vértices e arestas**

Percorrimento de um grafo

- Vários problemas relacionados a grafos envolvem a **obtenção de informações sobre sua estrutura**
  - Envolve uma **forma sistemática para explorar os vértices do grafo**
    - Exploração completa (precisa visitar todos os vértices)
    - Exploração parcial (apenas um subconjunto de vértices precisa ser visitado)
  - Define **como caminhar pelos vértices e arestas**

- **Pontos críticos:**
  - Ponto de partida (**vértice inicial**)
  - **Repetição de vértices** já visitados

Percorrimento de um grafo

Vários problemas relacionados a grafos envolvem a **obtenção de informações sobre sua estrutura**

- Envolve uma **forma sistemática para explorar os vértices do grafo**
  - Exploração completa (precisa visitar todos os vértices)
  - Exploração parcial (apenas um subconjunto de vértices precisa ser visitado)
- Define **como caminhar pelos vértices e arestas**

Pontos críticos:

- Ponto de partida (**vértice inicial**)
- **Repetição de vértices** já visitados

Principais tipos de percorrimento em grafos:

- **Busca em profundidade (DFS - *Depth-First Search*)**
- **Busca em largura (BFS - *Breadth-First Search*)**
- **Busca pelo menor caminho**

Percorrimento de um grafo

Busca em Profundidade

Busca em profundidade

Estratégia que visa **explorar o máximo possível cada um dos ramos** do grafo (**mais profundo**)

Busca em profundidade

Estratégia que visa **explorar o máximo possível cada um dos ramos** do grafo (**mais profundo**)

Ideia básica:

- Explorar as arestas do **último vértice explorado**
- Quando todas as arestas do vértice tiverem sido exploradas, a busca retrocede para o seu antecessor (***backtracking***)

Busca em profundidade

Estratégia que visa **explorar o máximo possível cada um dos ramos** do grafo (**mais profundo**)

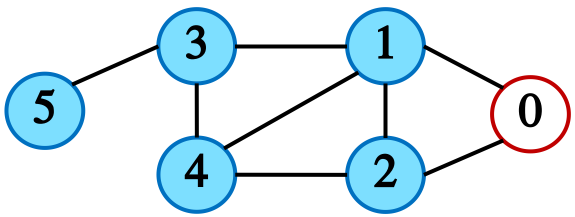
Ideia básica:

- Explorar as arestas do **último vértice explorado**
- Quando todas as arestas do vértice tiverem sido exploradas, a busca retrocede para o seu antecessor (***backtracking***)

Exemplos de utilização:

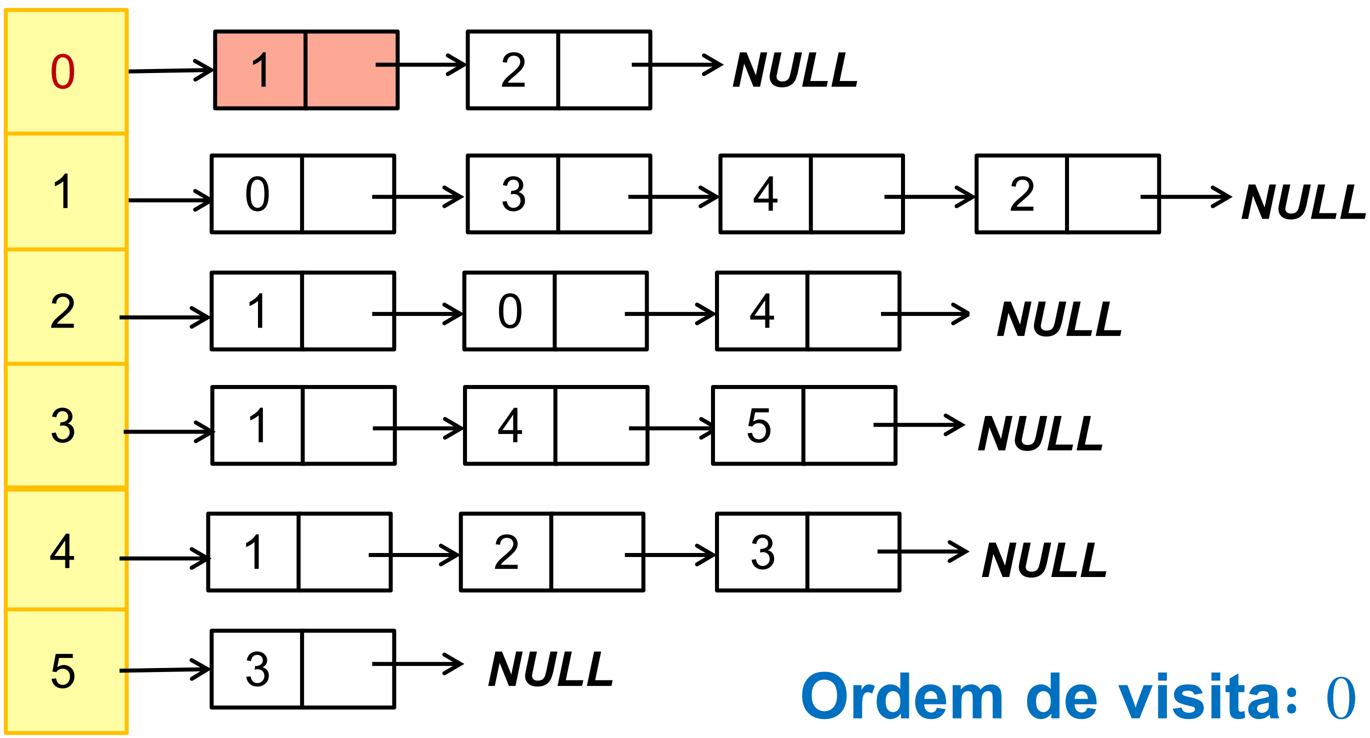
- Encontrar componentes conectados
- Obter uma ordenação topológica do grafo
- Verificar se um grafo é cíclico ou acíclico
- Resolver quebra-cabeças (**ex:** labirintos)

Busca em profundidade: exemplo



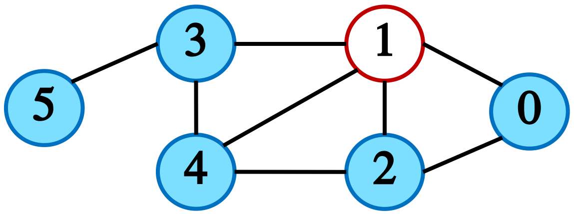
vértice inicial

0

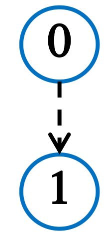


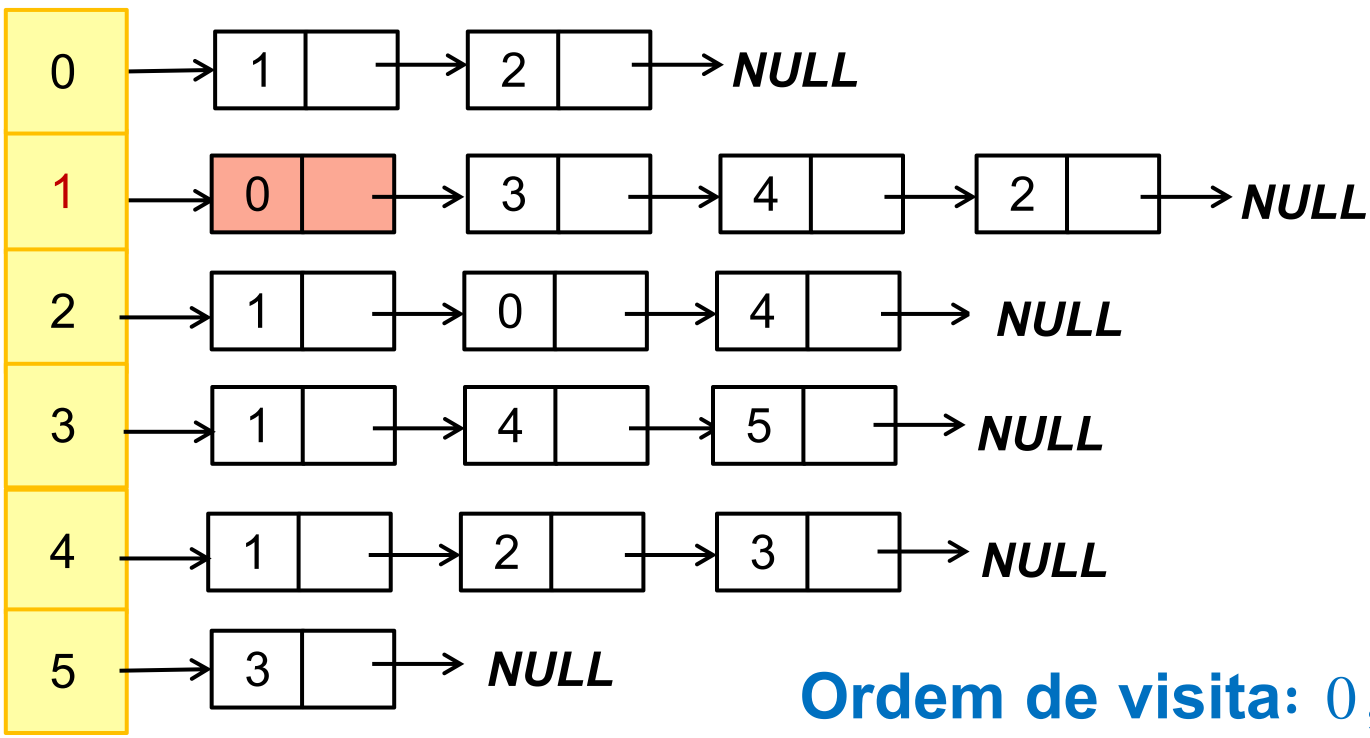
Ordem de visita: 0

Busca em profundidade: exemplo



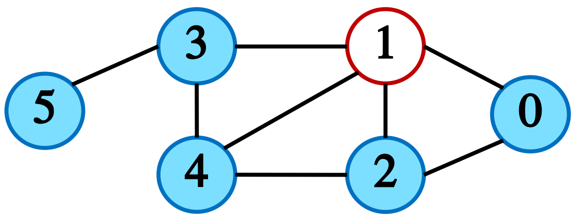
vértice inicial



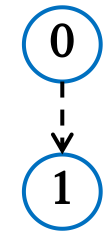


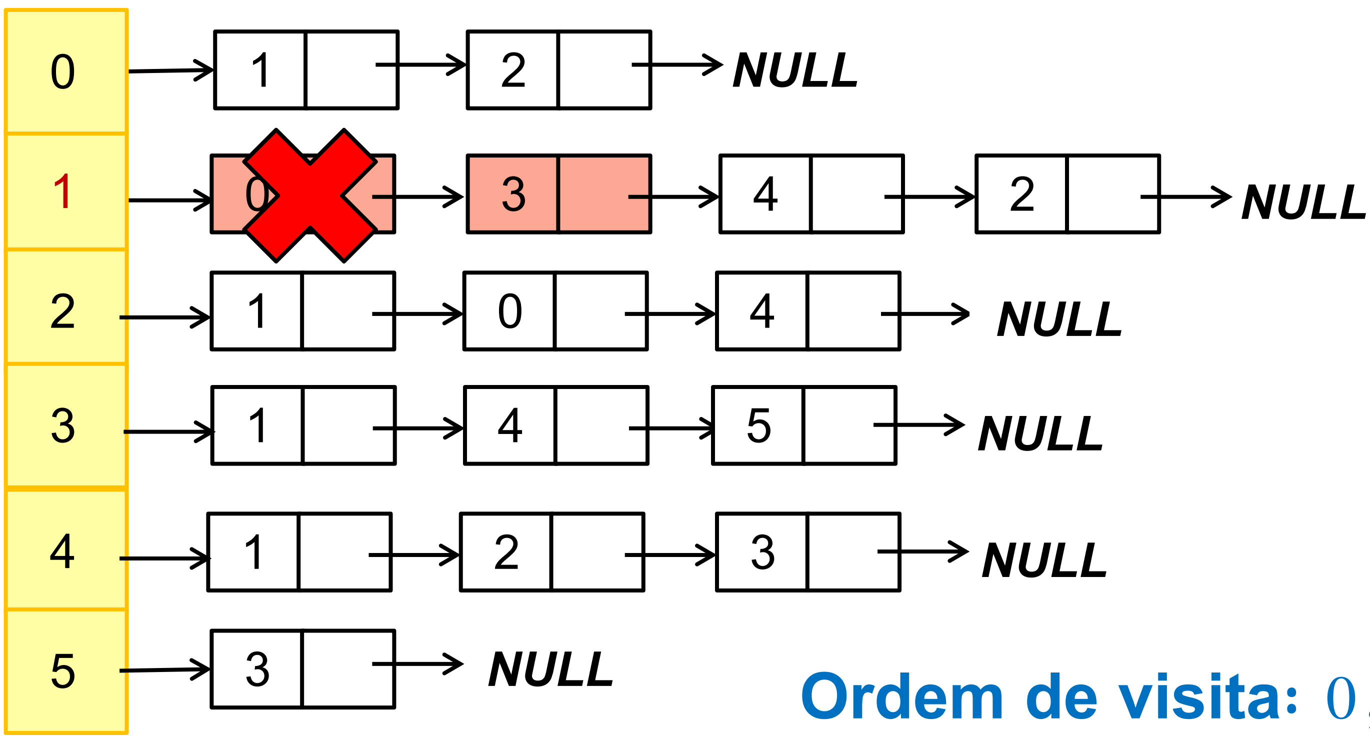
Ordem de visita: 0 , 1

Busca em profundidade: exemplo



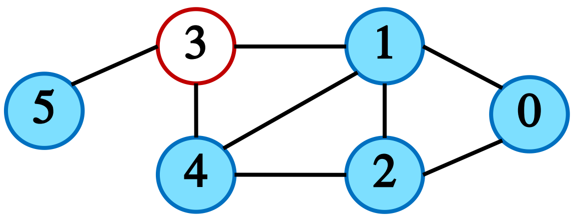
vértice inicial

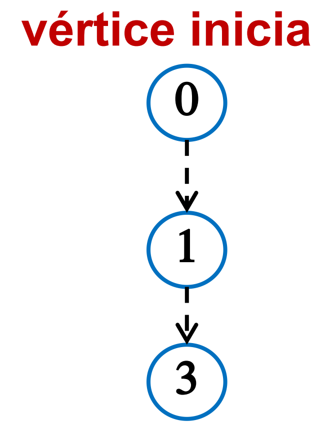


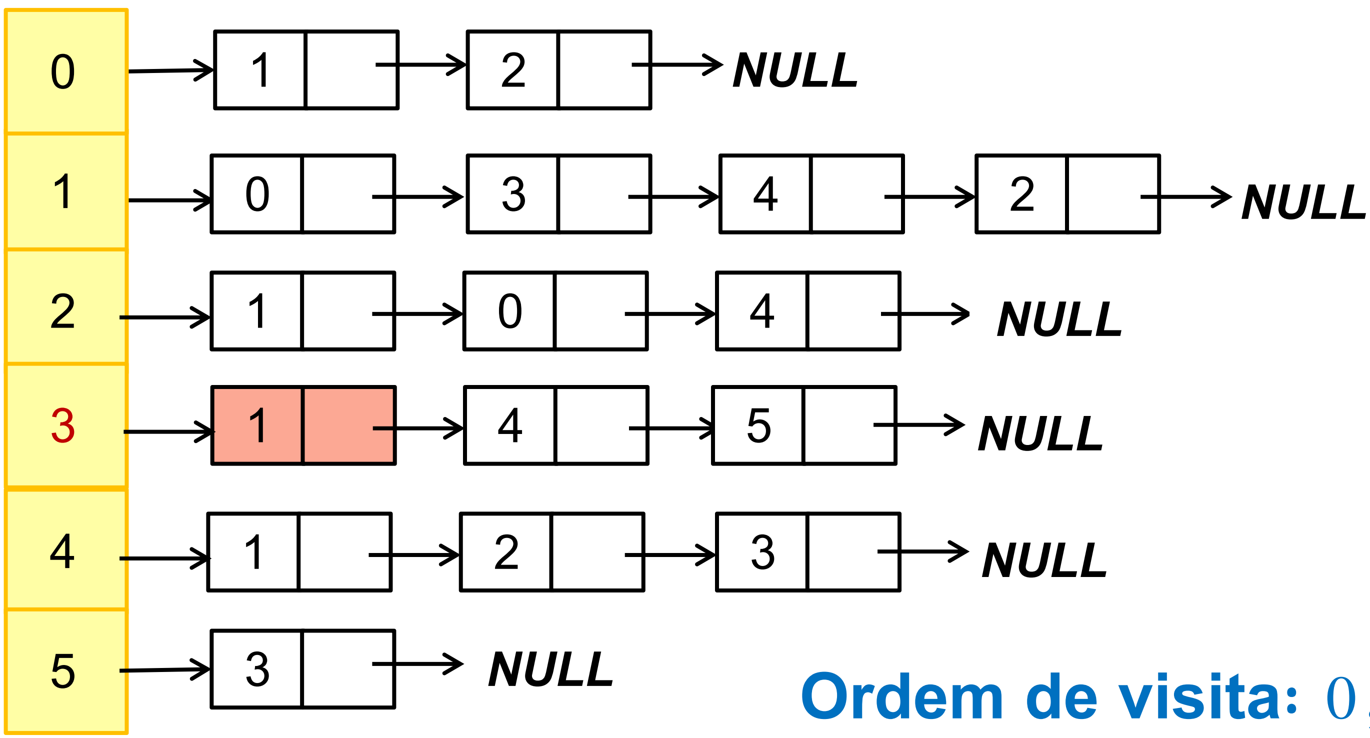


Ordem de visita: 0 , 1

Busca em profundidade: exemplo

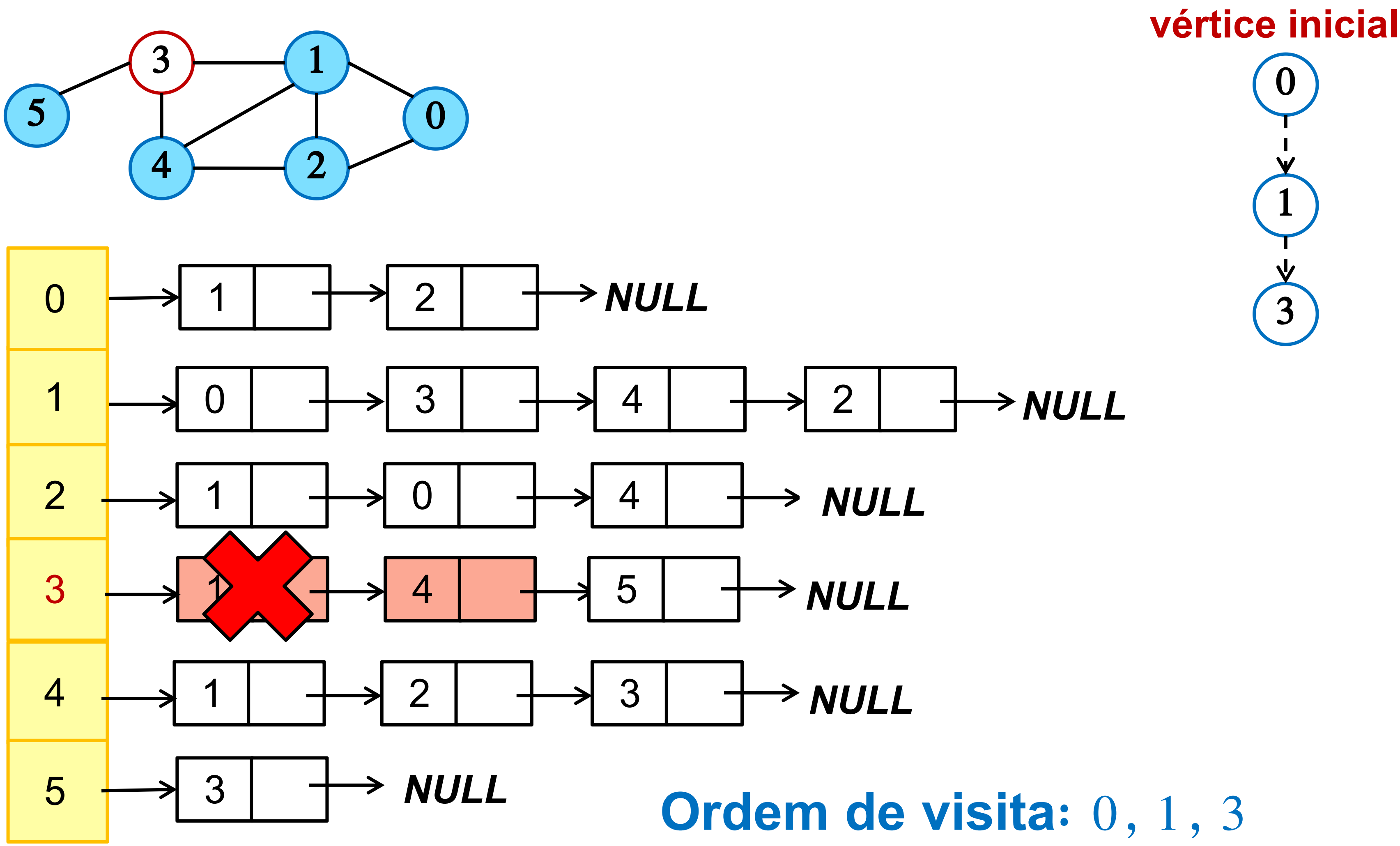






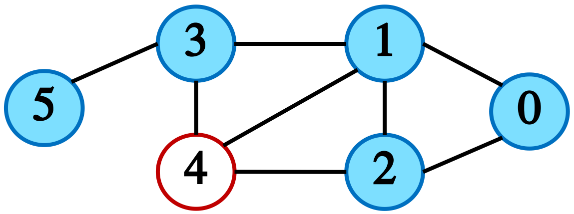
Ordem de visita: 0 , 1 , 3

Busca em profundidade: exemplo

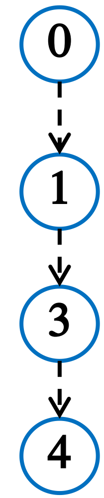


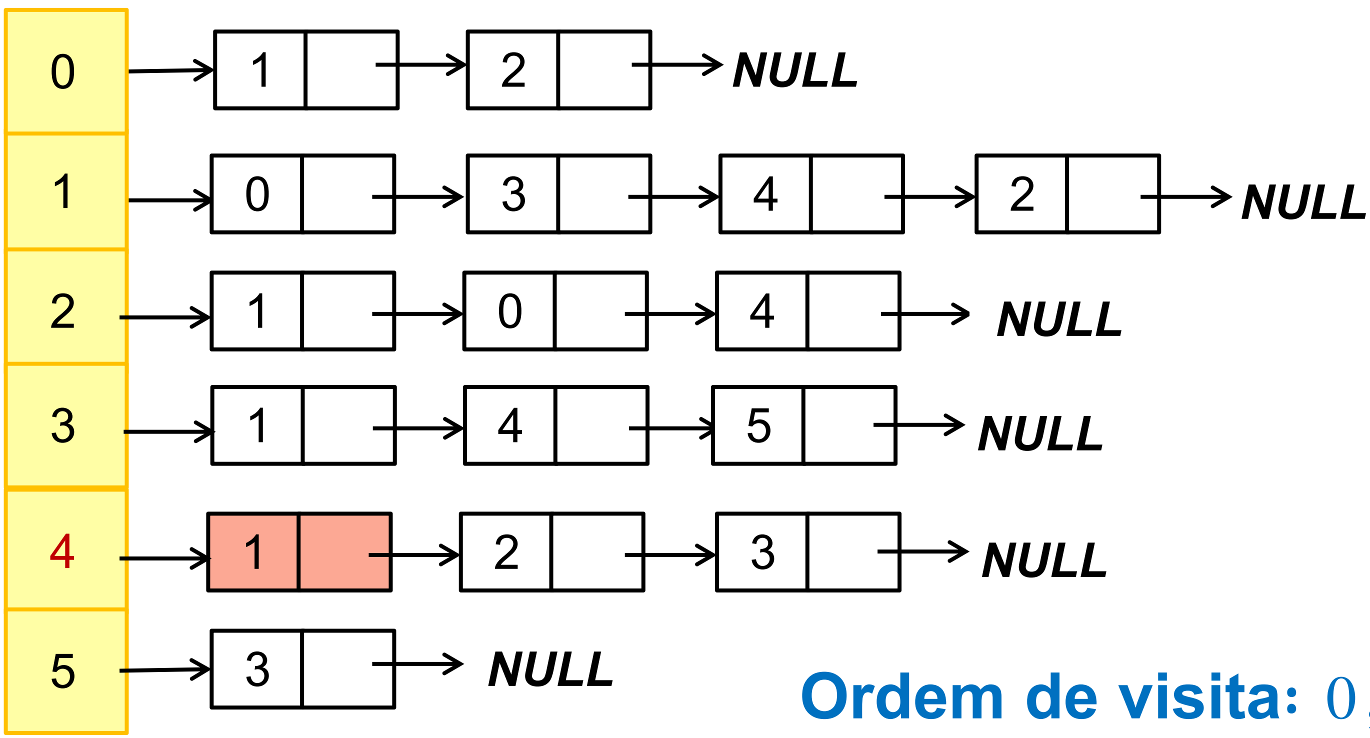
Ordem de visita: 0 , 1 , 3

Busca em profundidade: exemplo



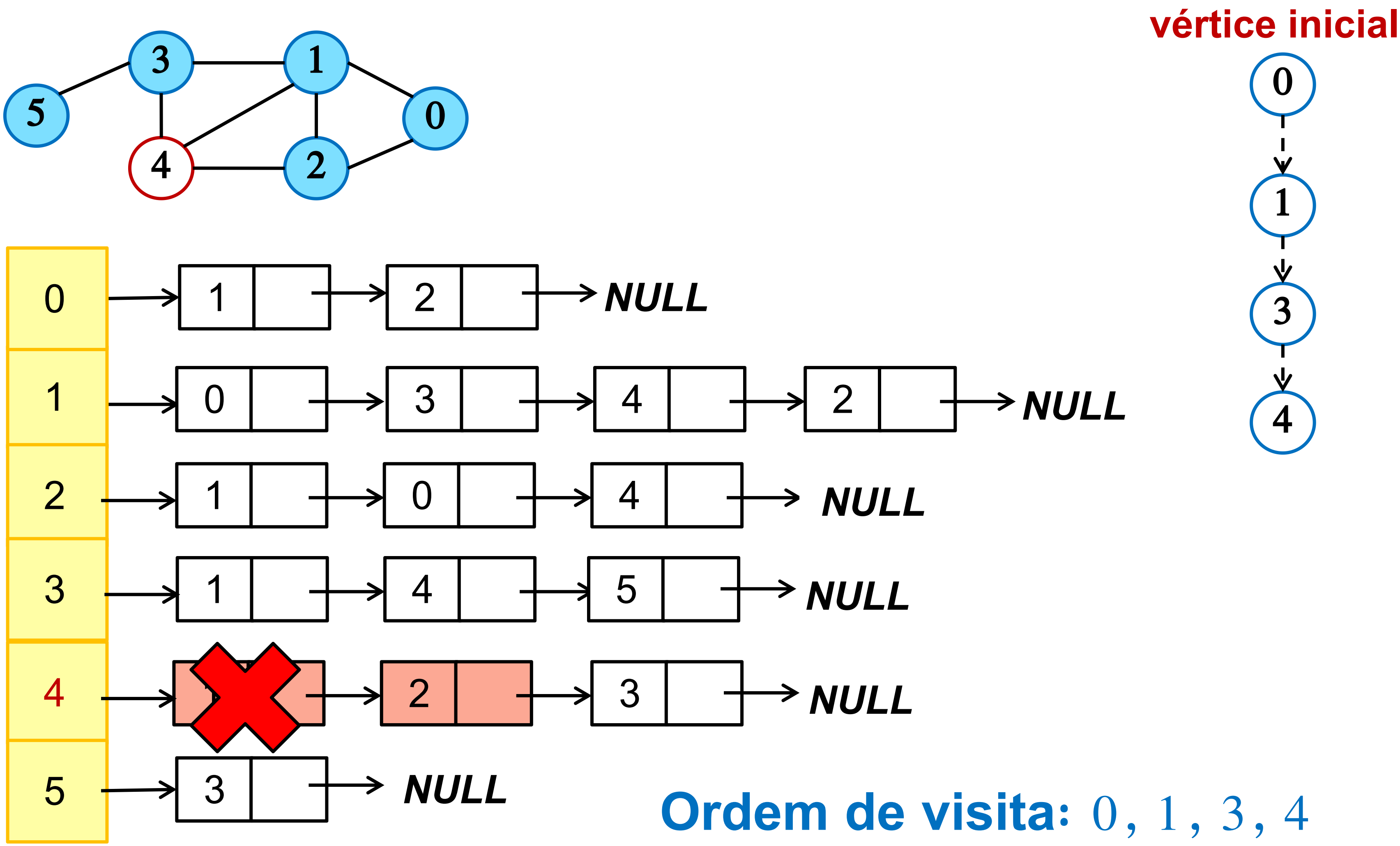
vértice inicial





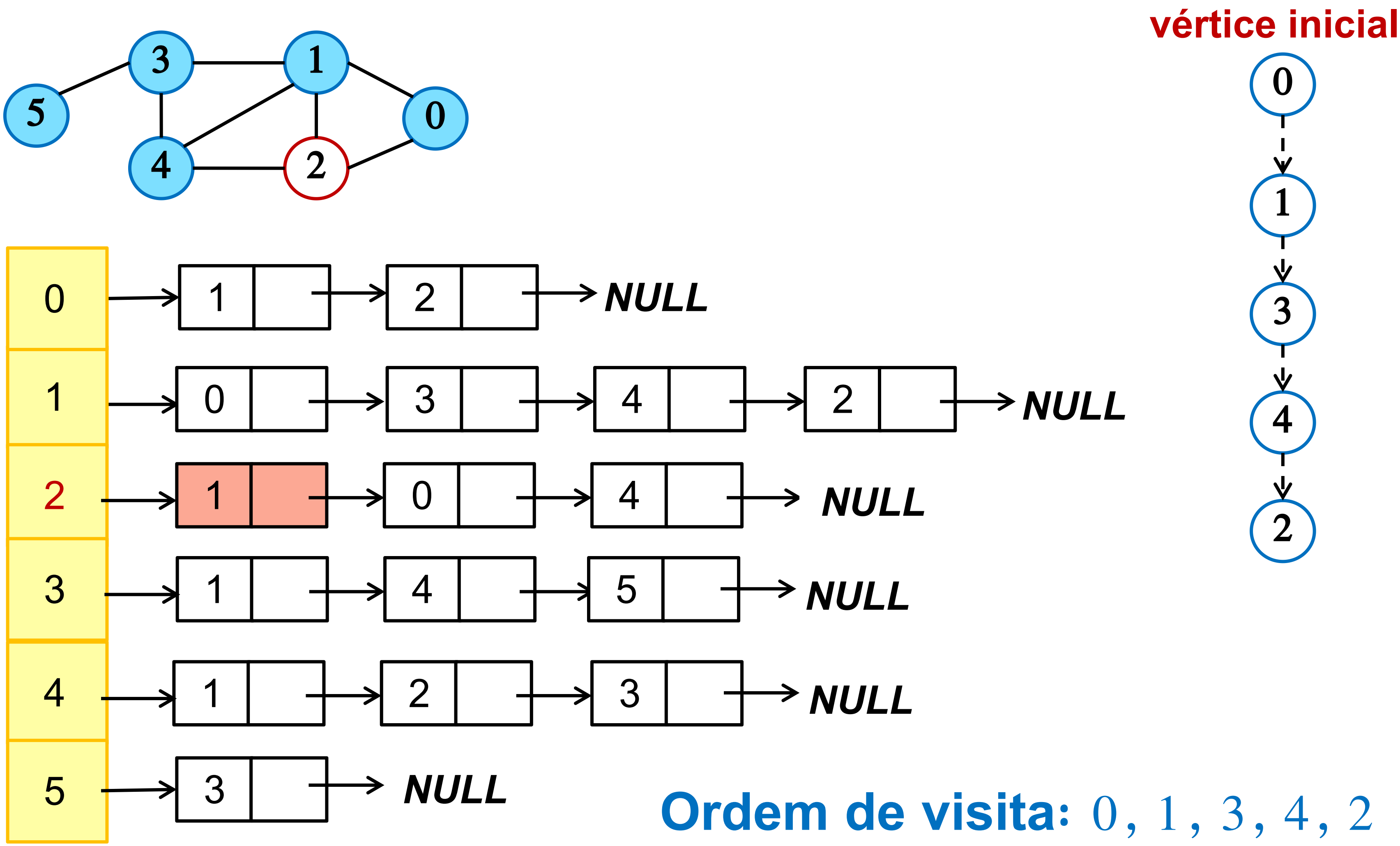
Ordem de visita: 0, 1, 3, 4

Busca em profundidade: exemplo



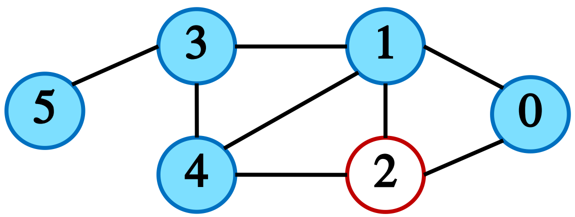
Ordem de visita: 0, 1, 3, 4

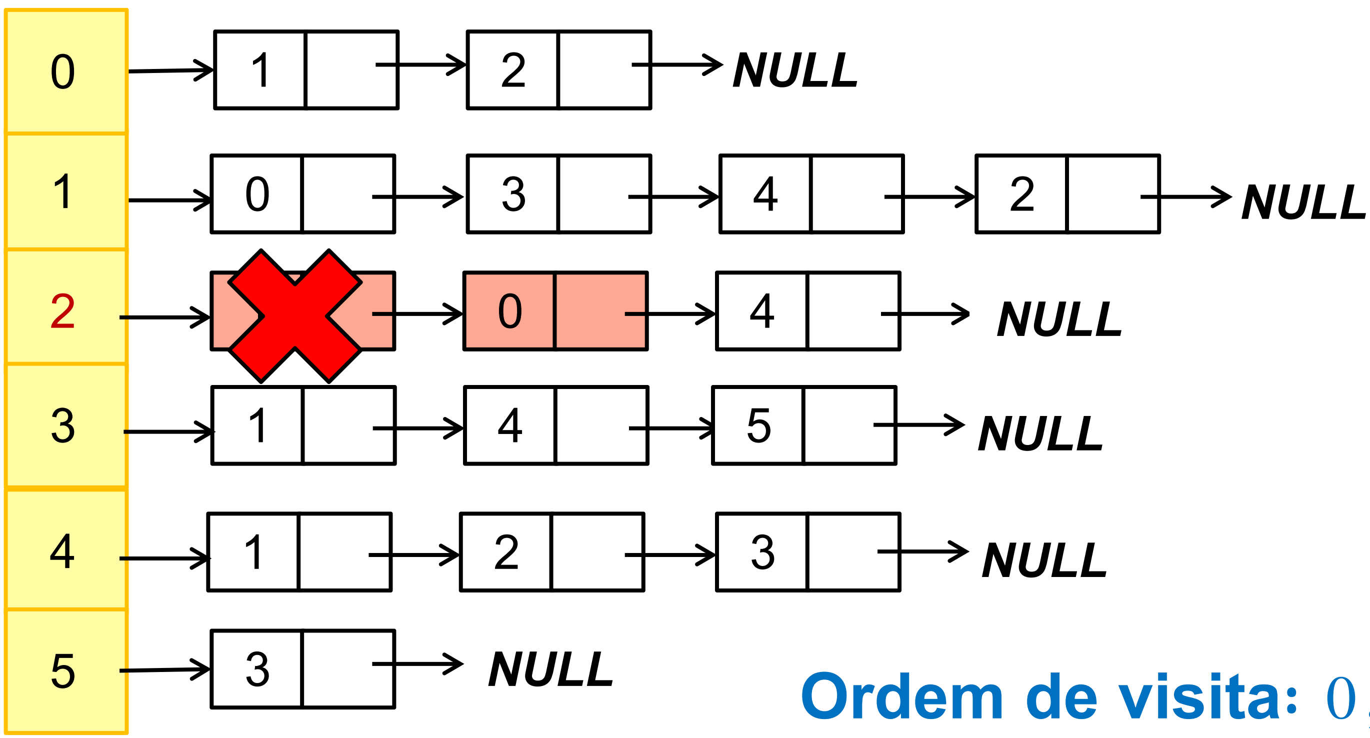
Busca em profundidade: exemplo

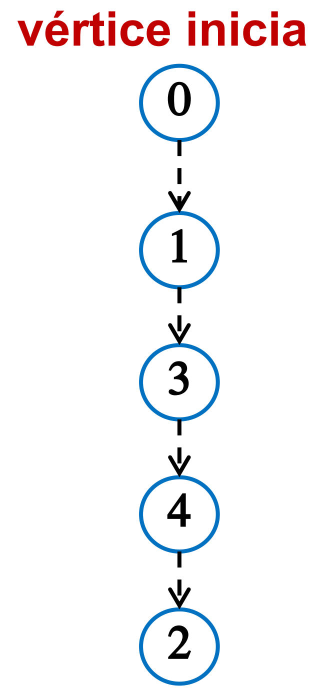


Ordem de visita: 0, 1, 3, 4, 2

Busca em profundidade: exemplo

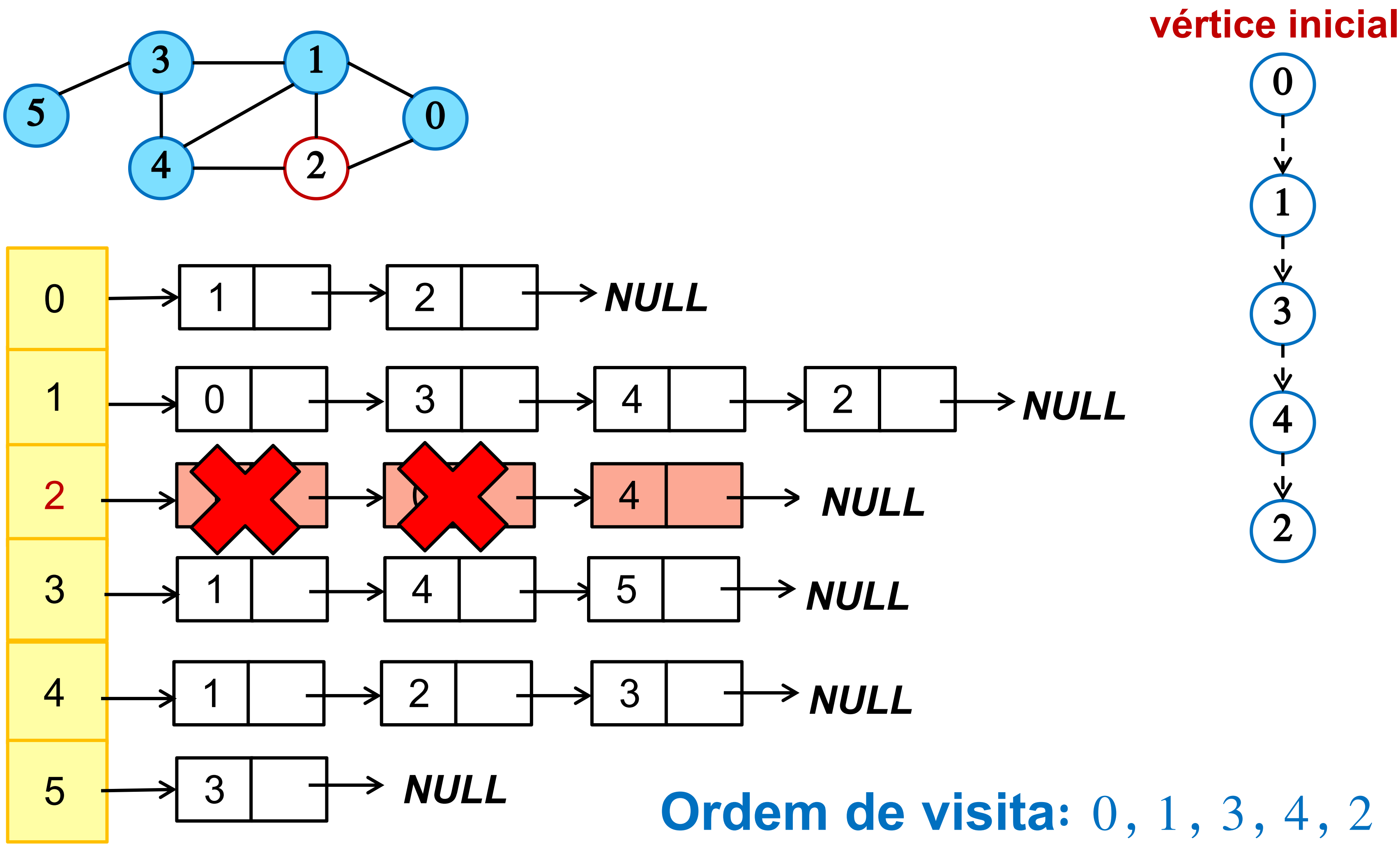






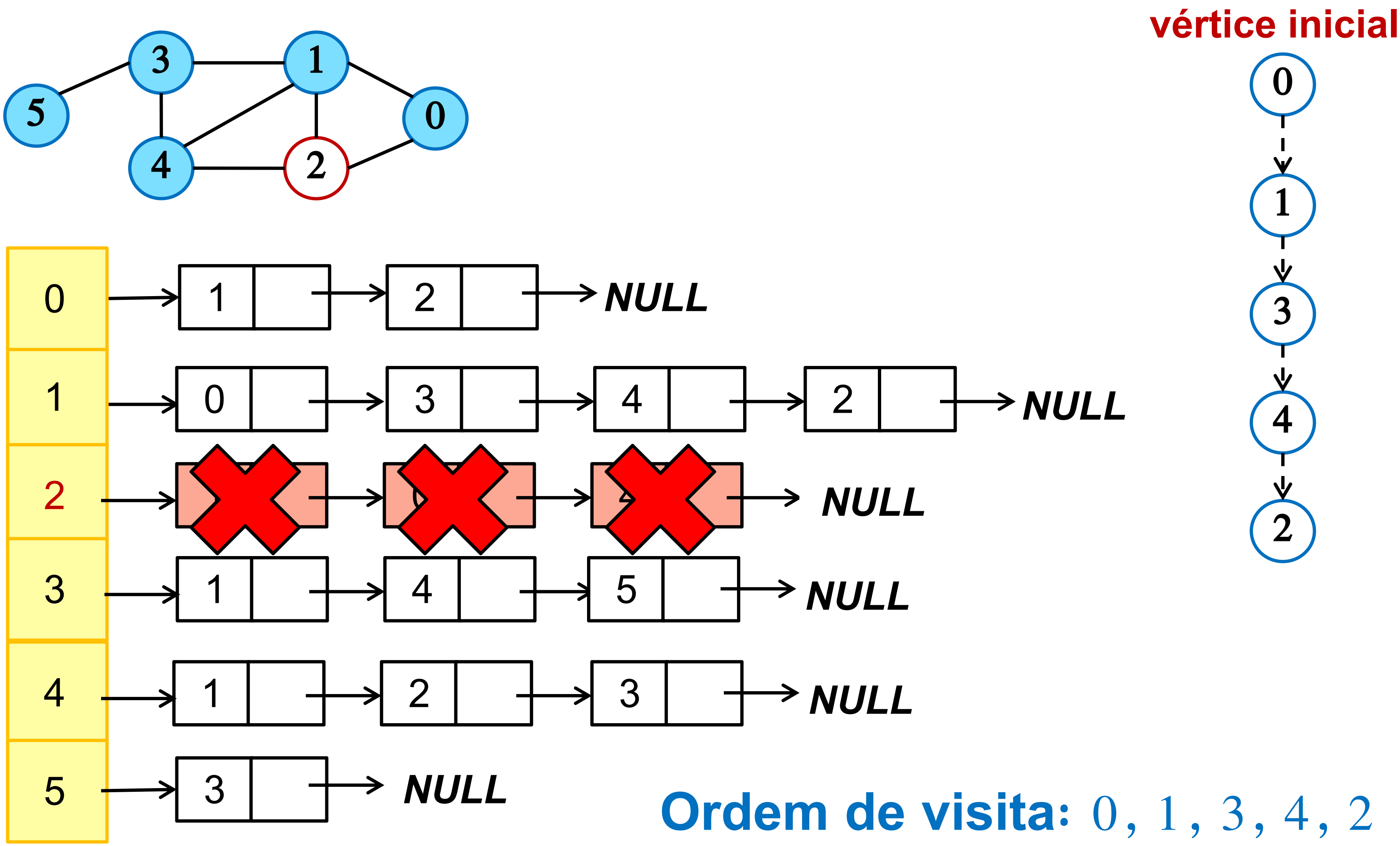
Ordem de visita: 0, 1, 3, 4, 2

Busca em profundidade: exemplo



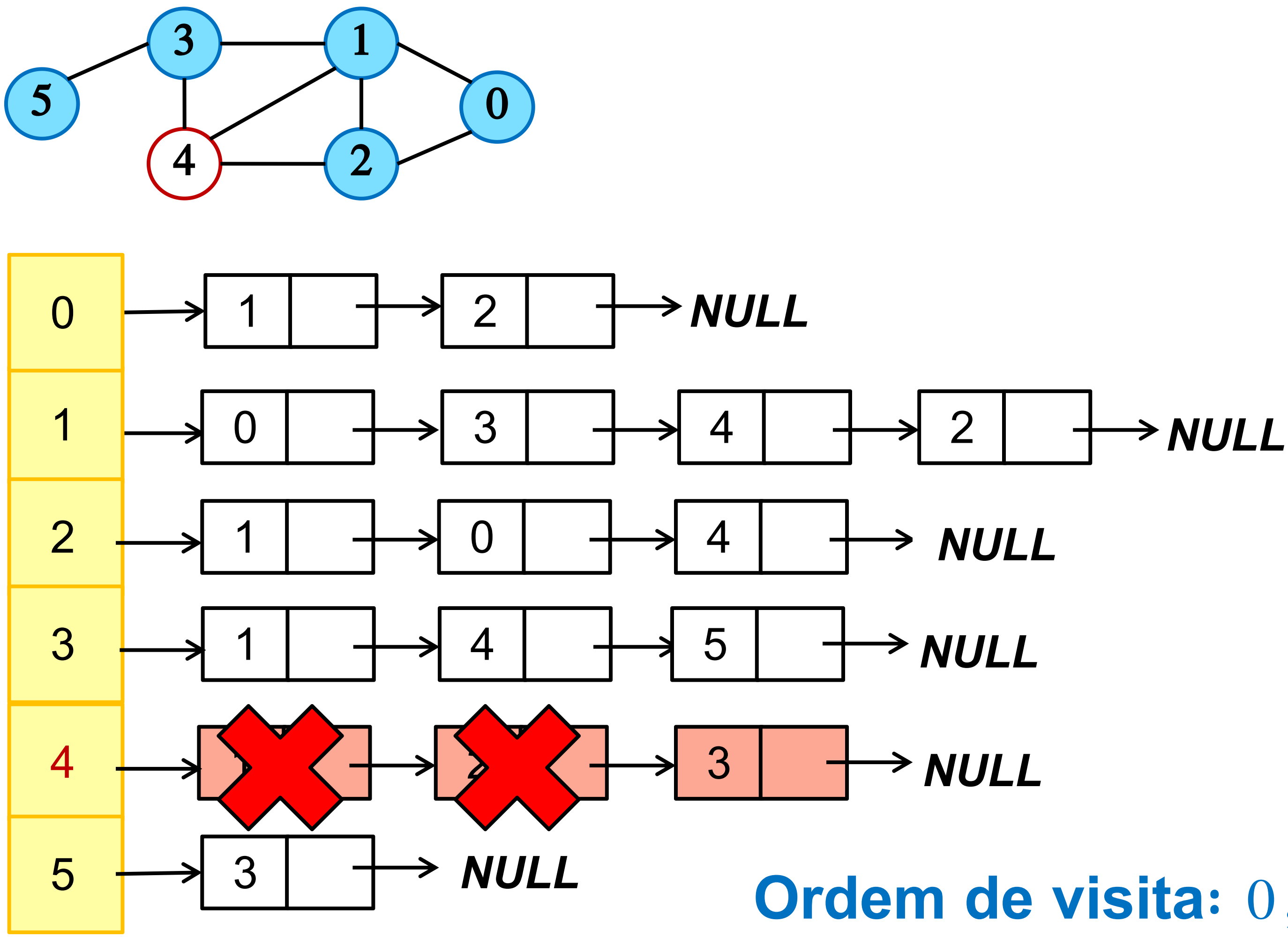
Ordem de visita: 0, 1, 3, 4, 2

Busca em profundidade: exemplo

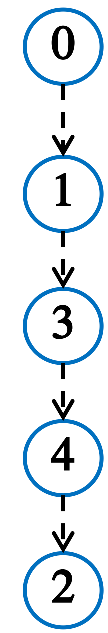


Ordem de visita: 0, 1, 3, 4, 2

Busca em profundidade: exemplo

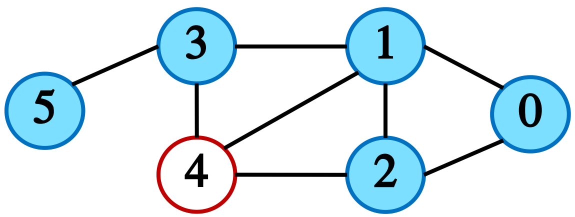


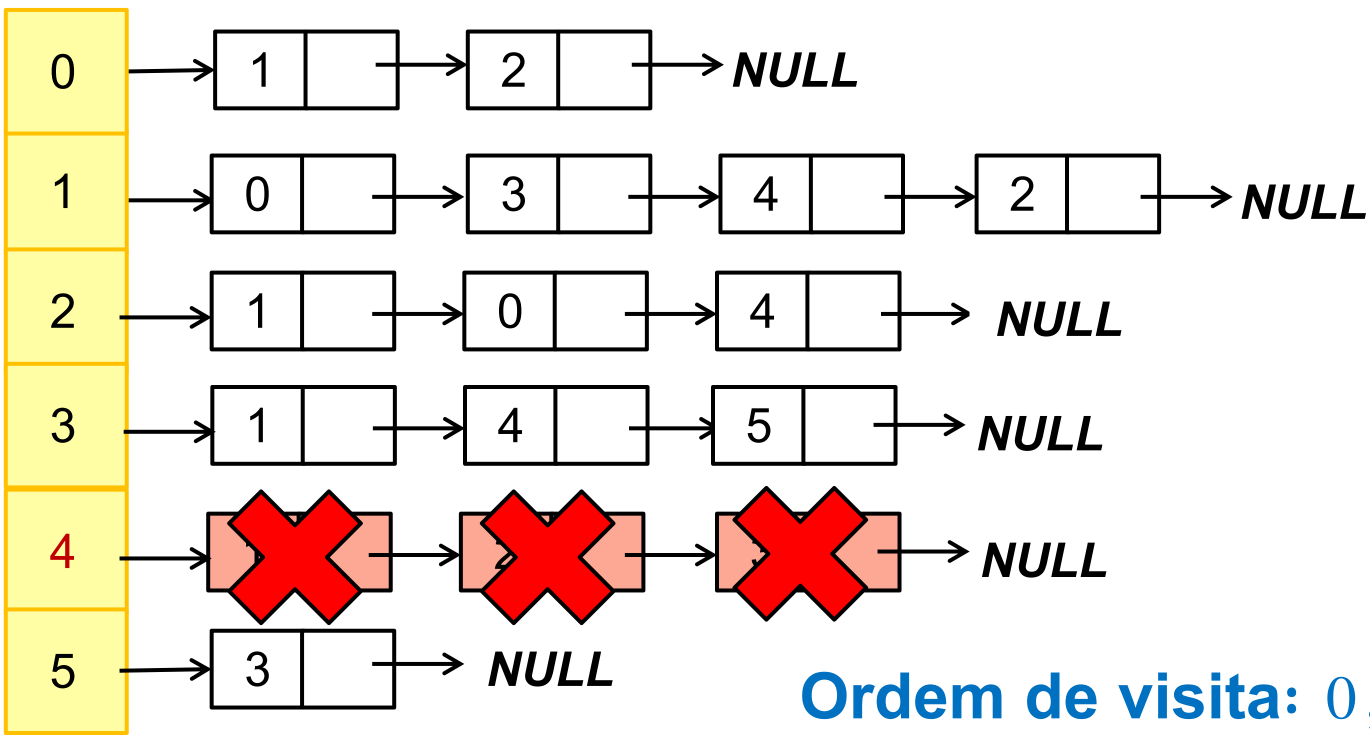
vértice inicial



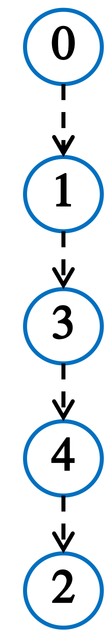
Ordem de visita: 0 , 1 , 3 , 4 , 2

Busca em profundidade: exemplo



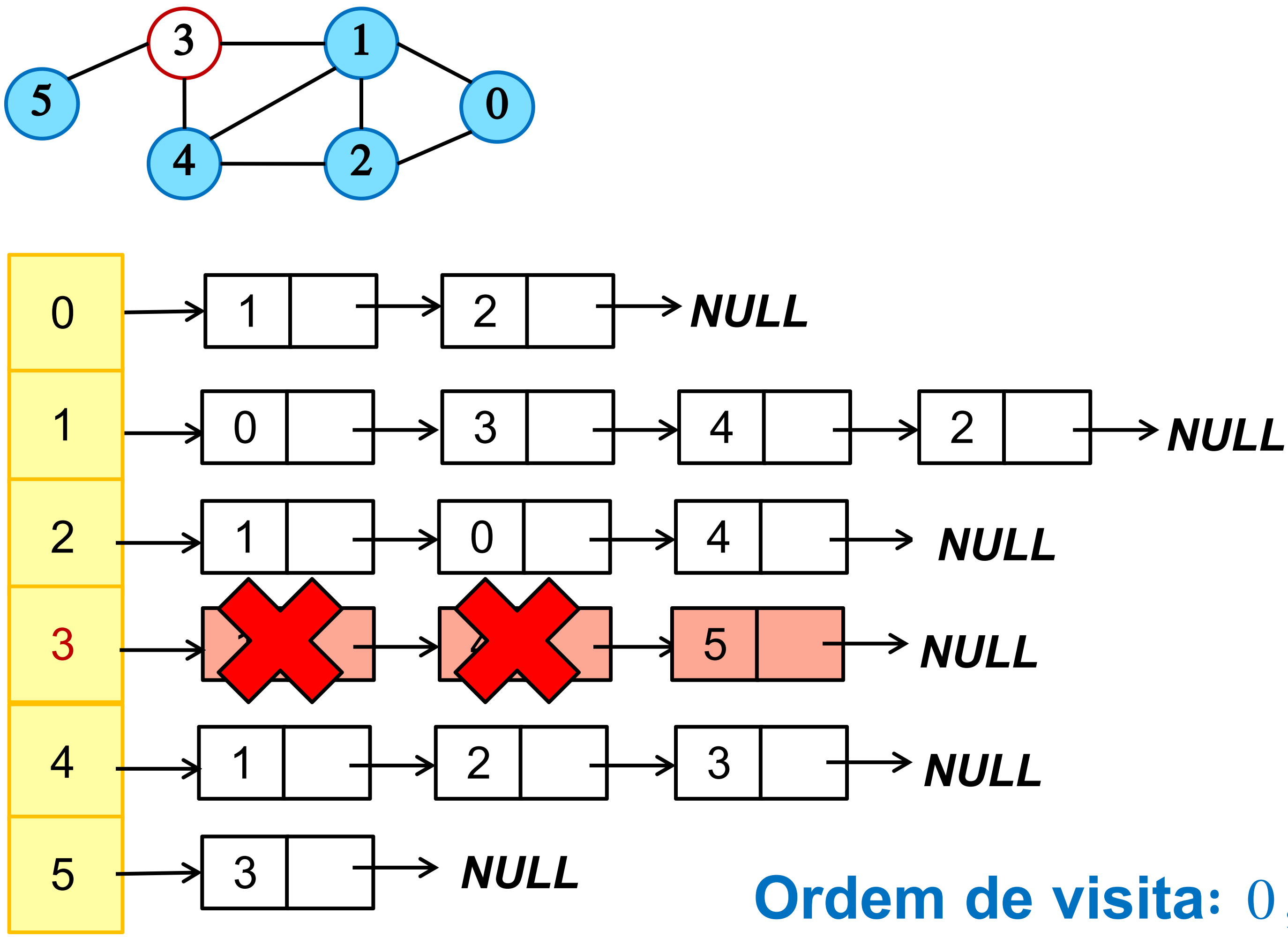


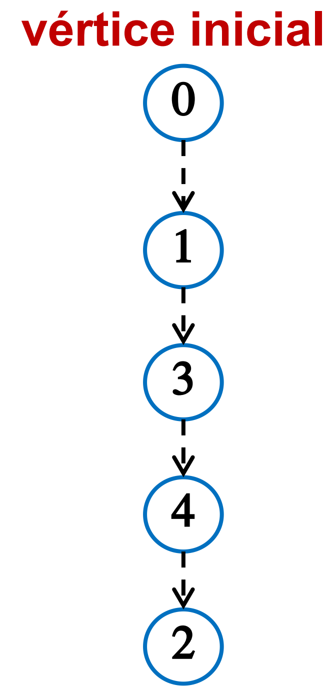
vértice inicial



Ordem de visita: 0, 1, 3, 4, 2

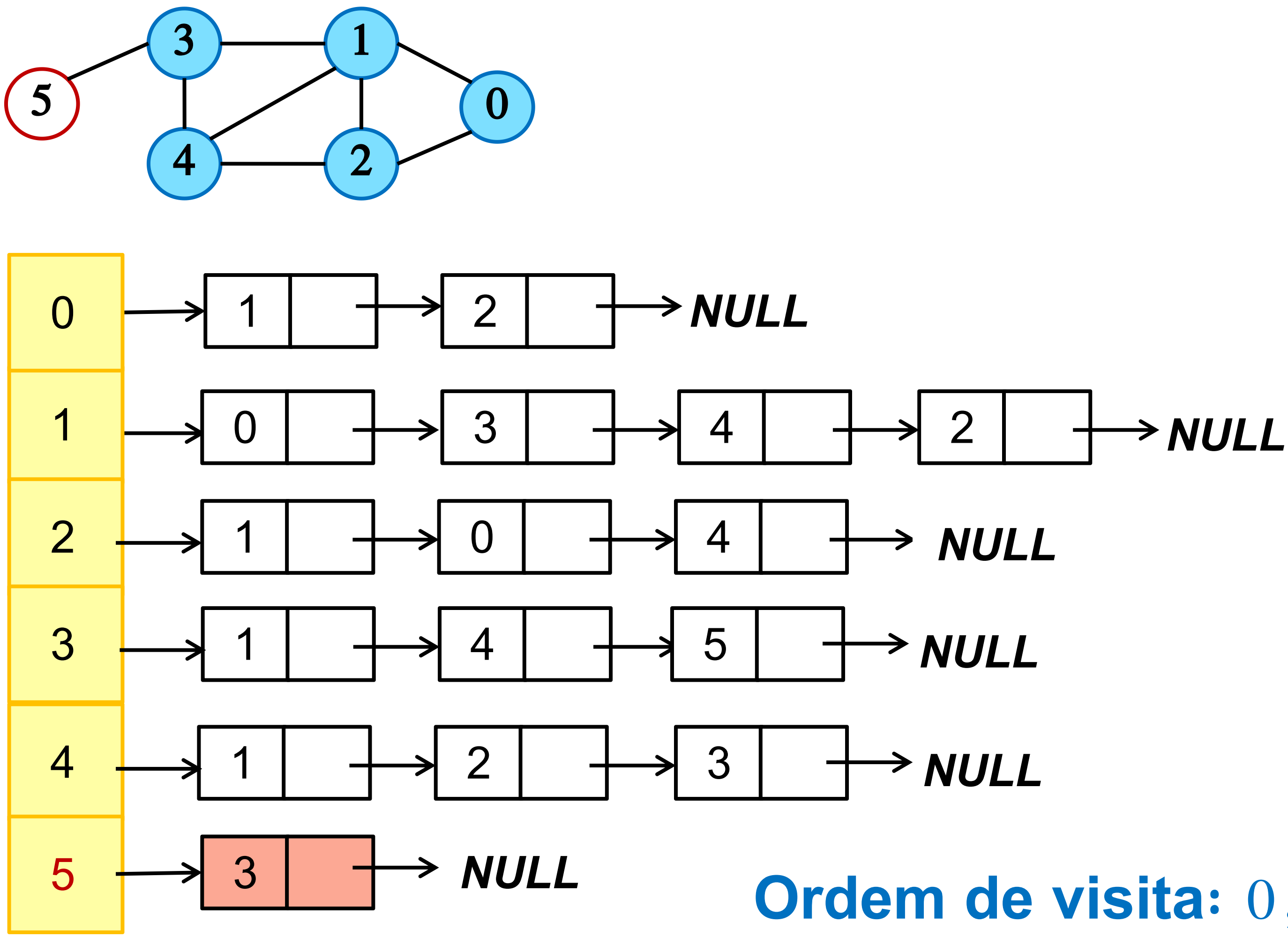
Busca em profundidade: exemplo

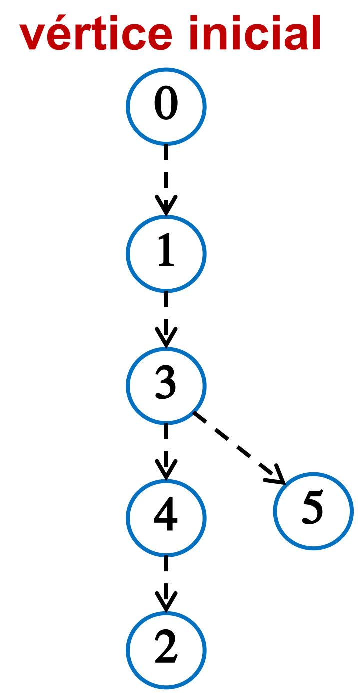




Ordem de visita: 0, 1, 3, 4, 2

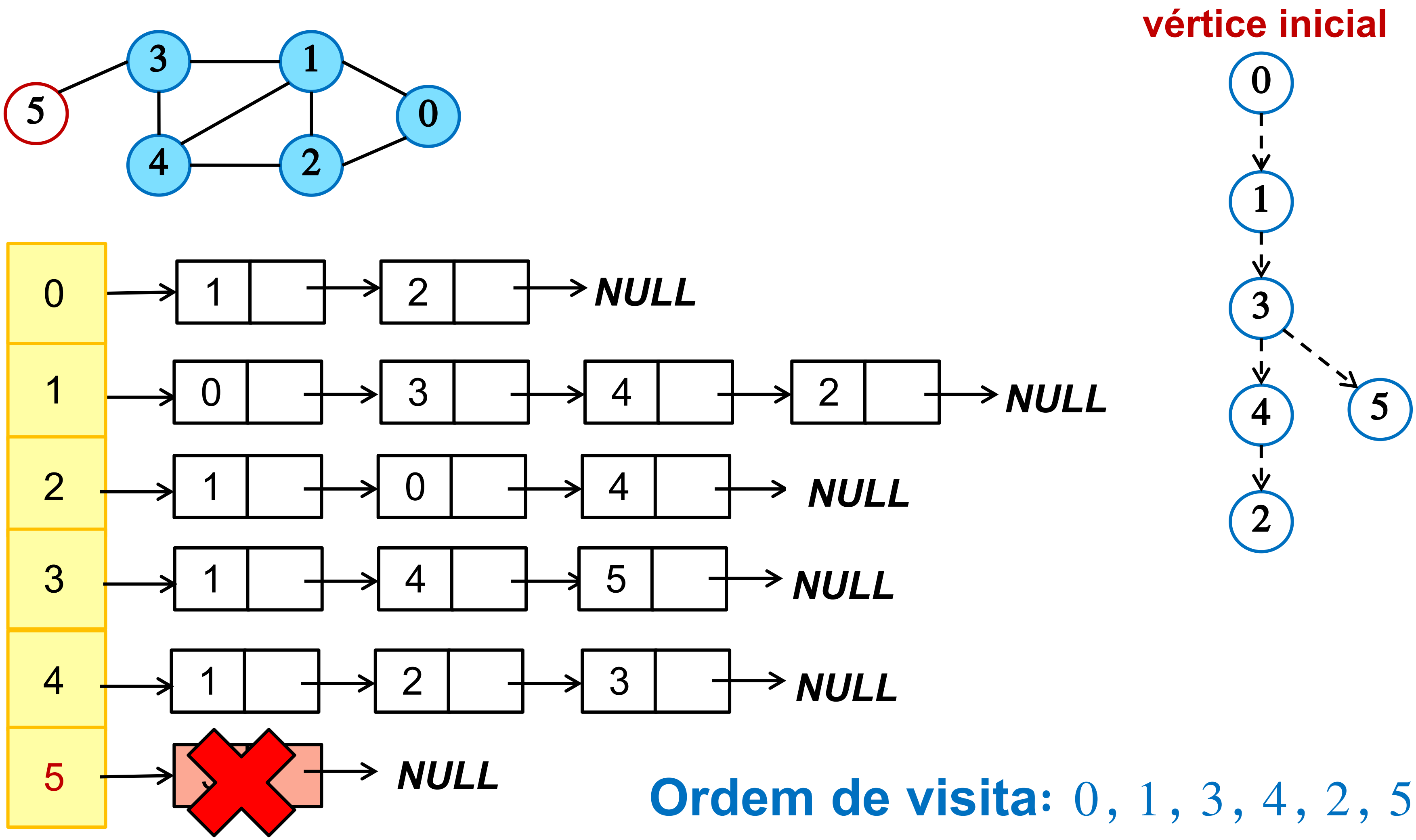
Busca em profundidade: exemplo





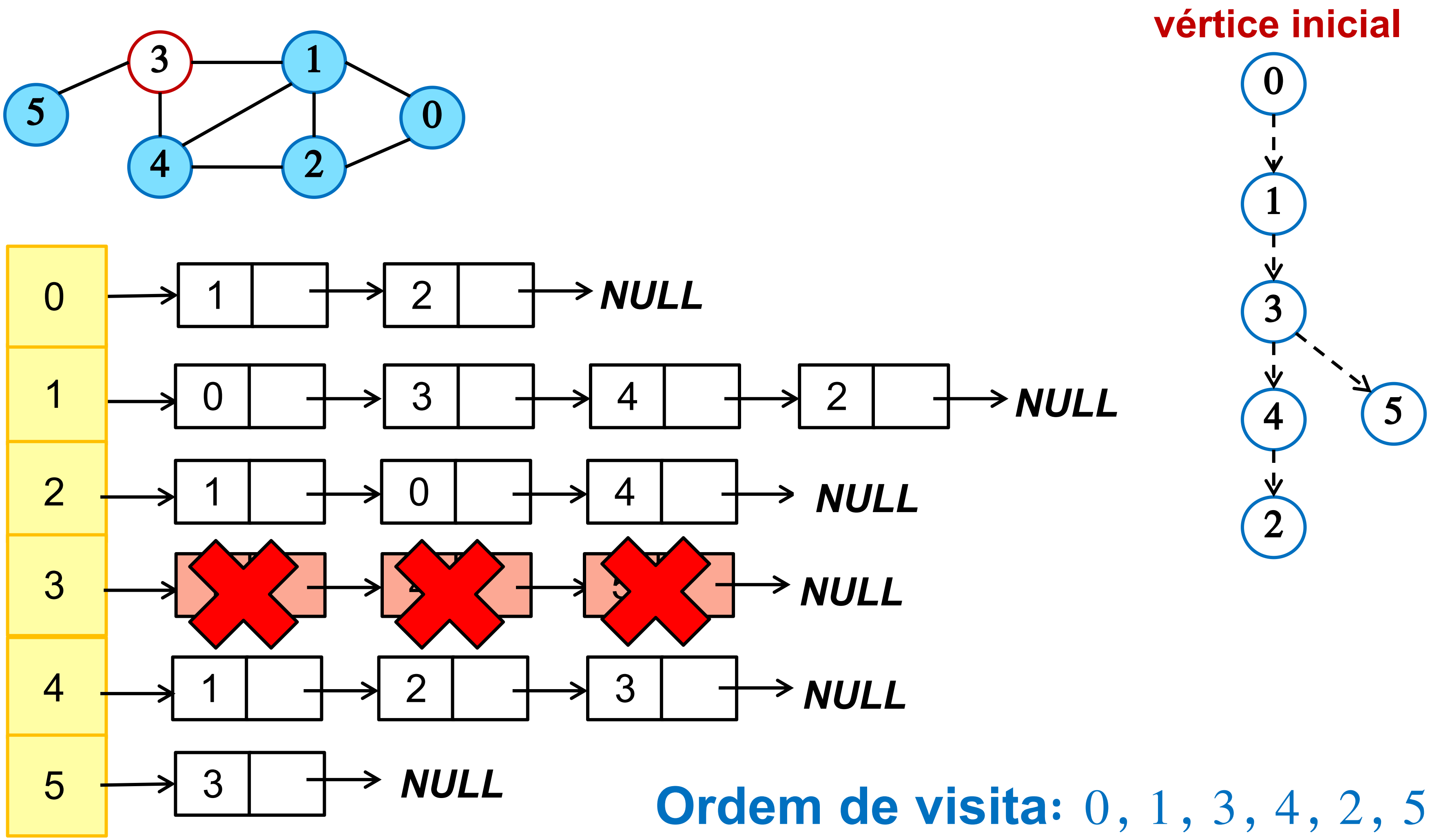
Ordem de visita: 0, 1, 3, 4, 2, 5

Busca em profundidade: exemplo



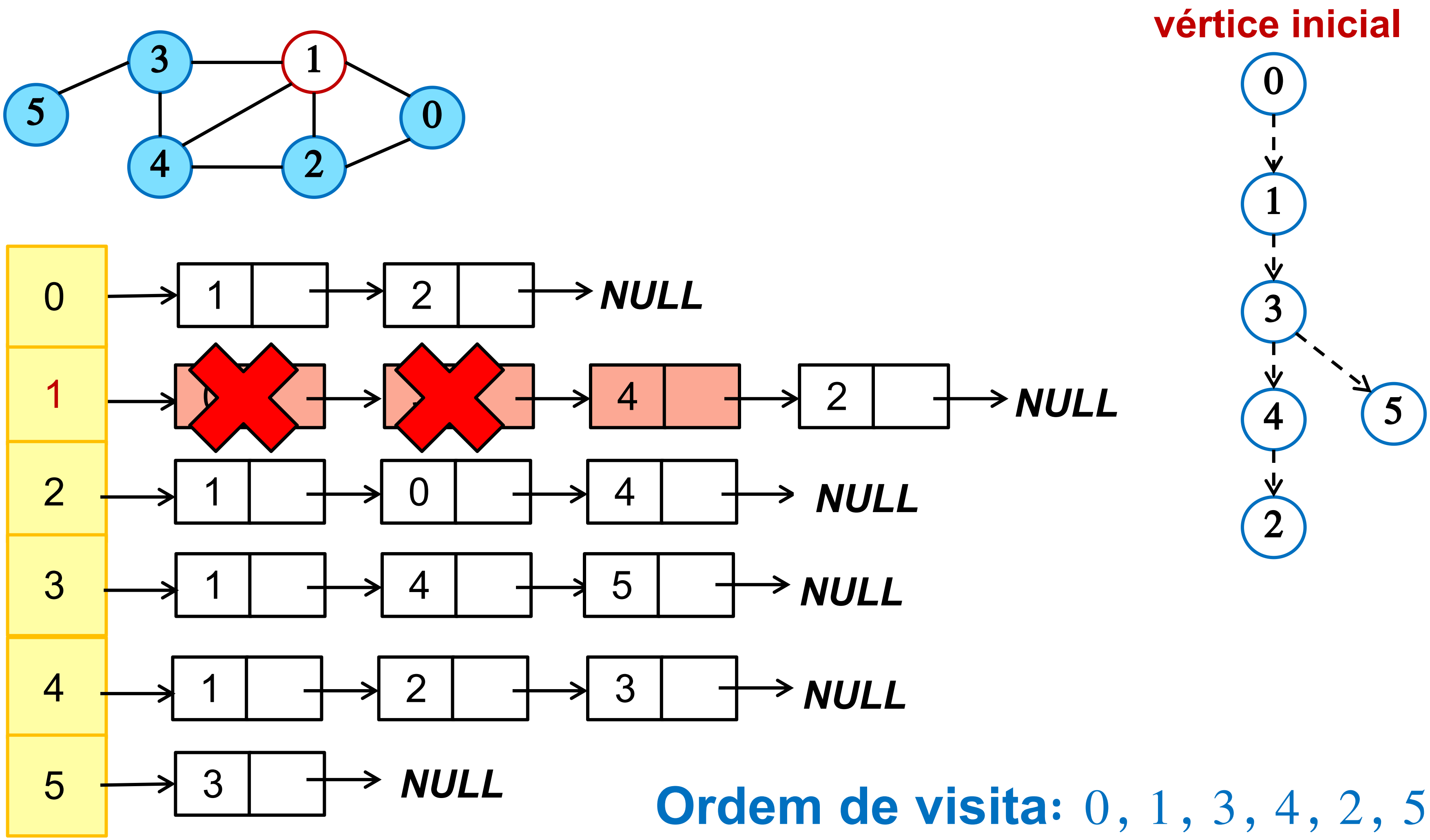
Ordem de visita: 0, 1, 3, 4, 2, 5

Busca em profundidade: exemplo



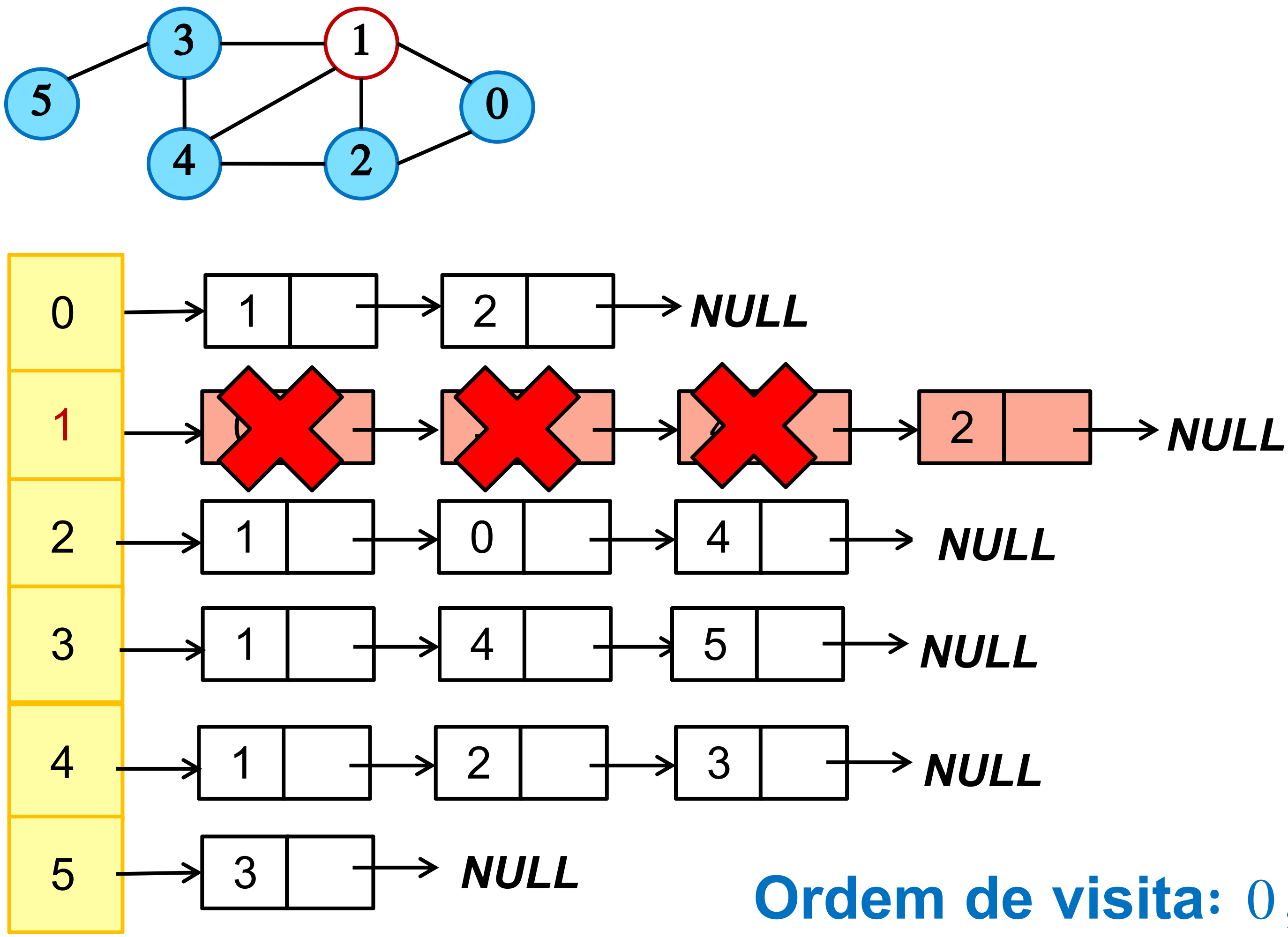
Ordem de visita: 0, 1, 3, 4, 2, 5

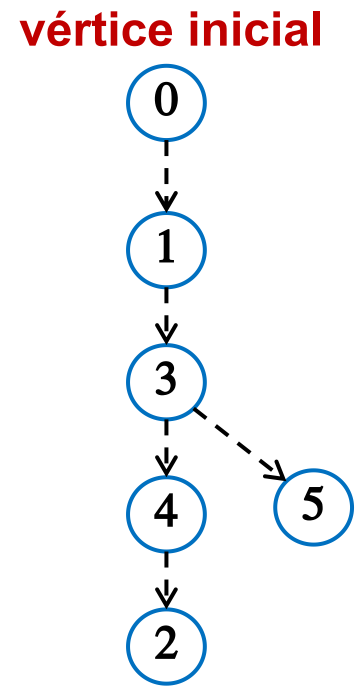
Busca em profundidade: exemplo



Ordem de visita: 0, 1, 3, 4, 2, 5

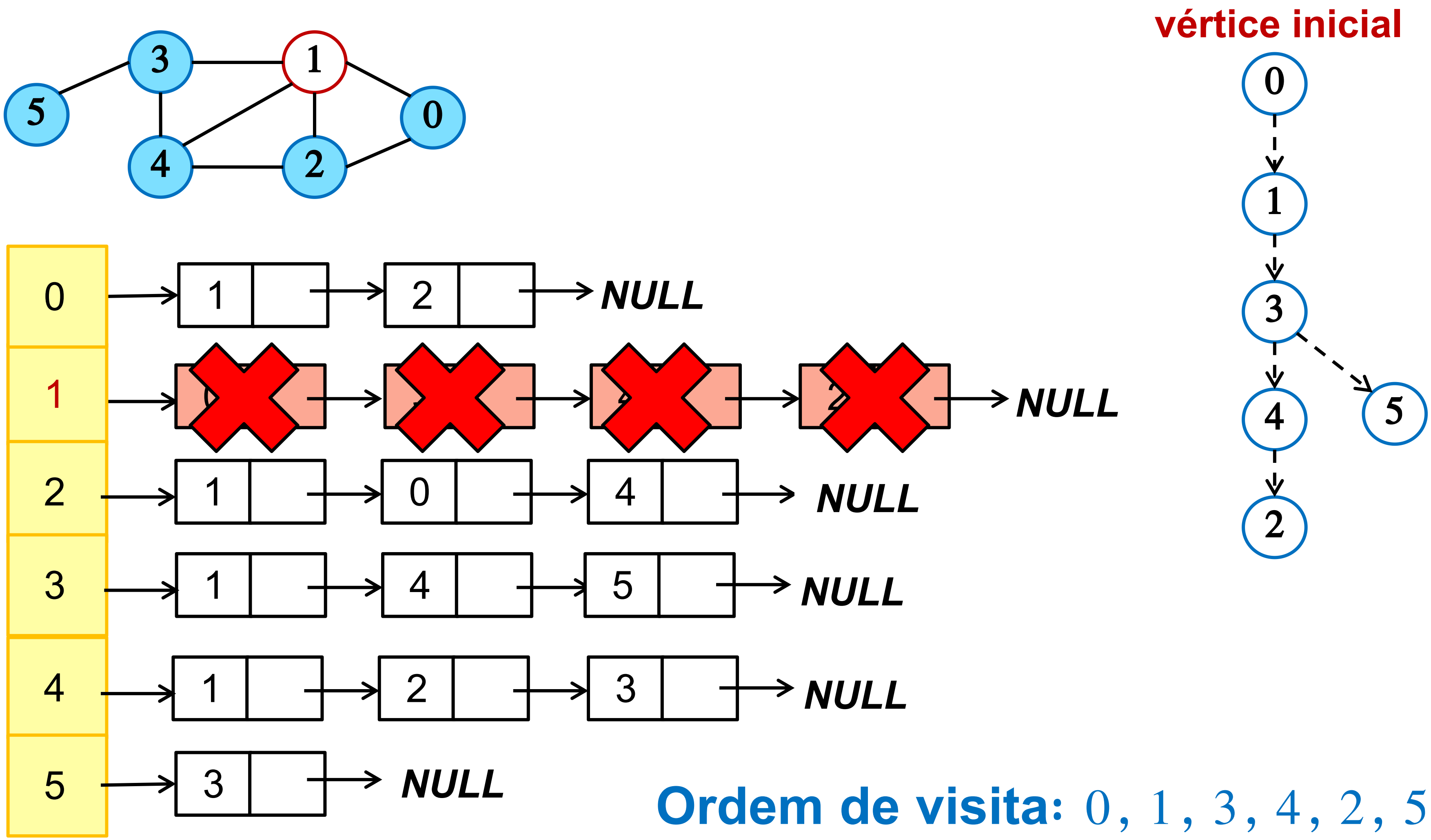
Busca em profundidade: exemplo





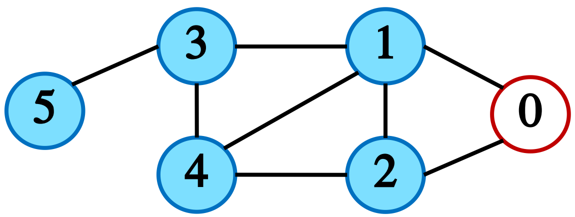
Ordem de visita: 0, 1, 3, 4, 2, 5

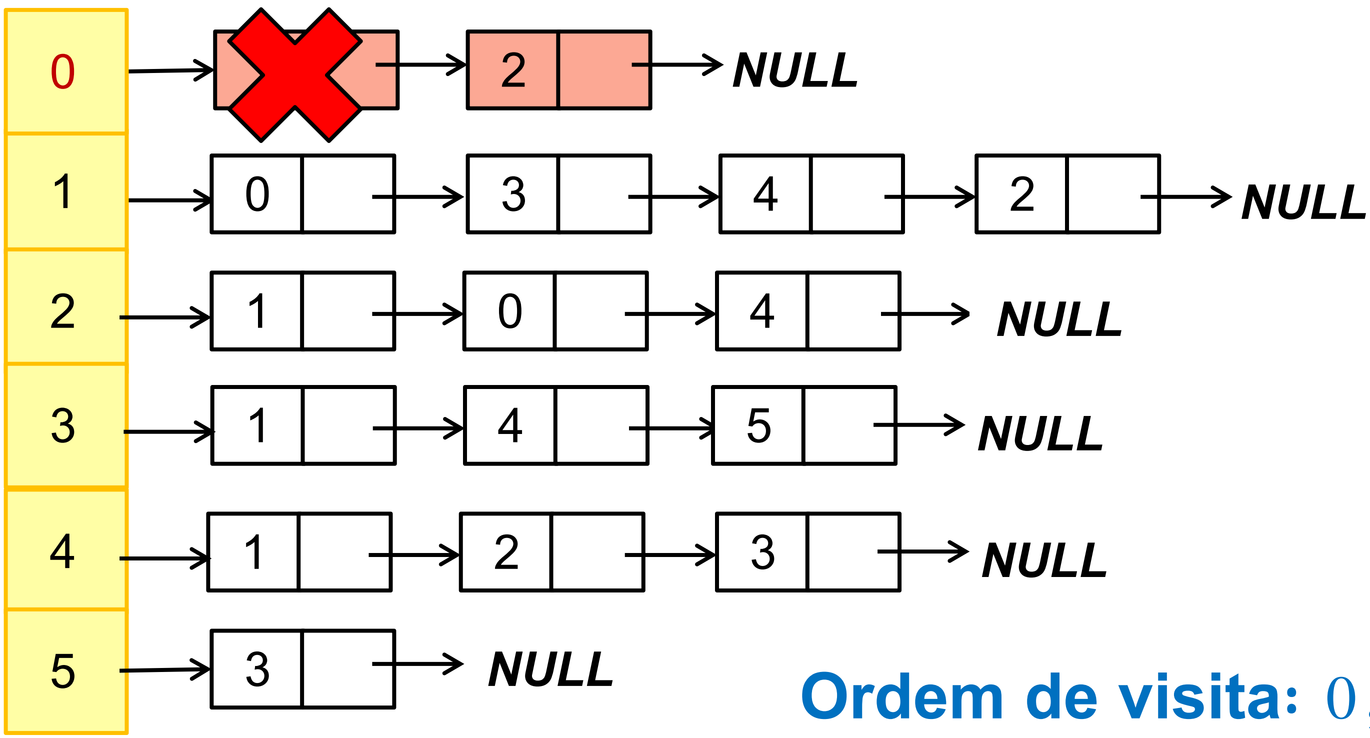
Busca em profundidade: exemplo



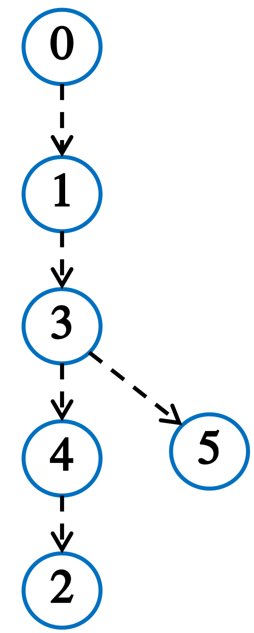
Ordem de visita: 0, 1, 3, 4, 2, 5

Busca em profundidade: exemplo



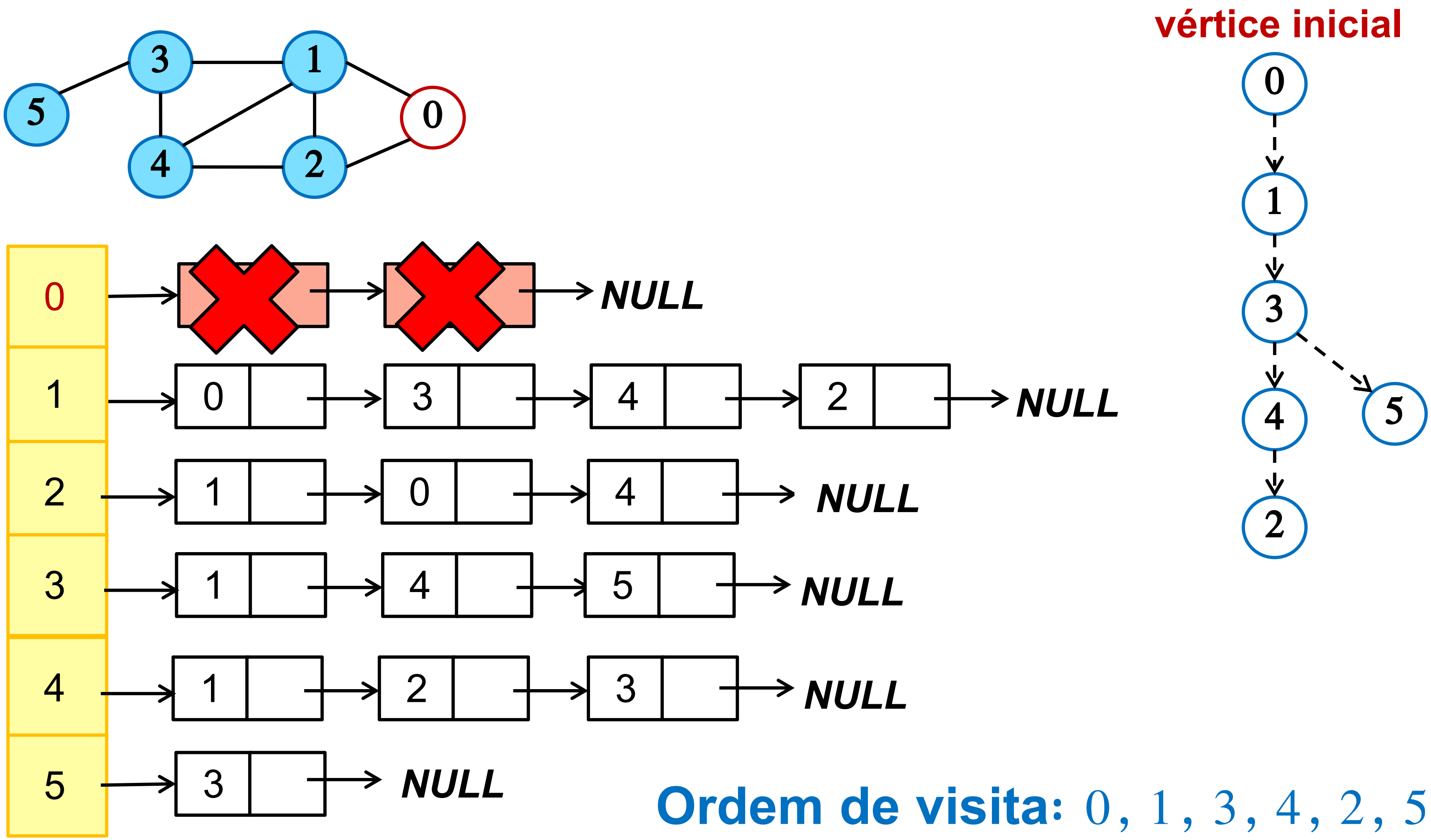


vértice inicial



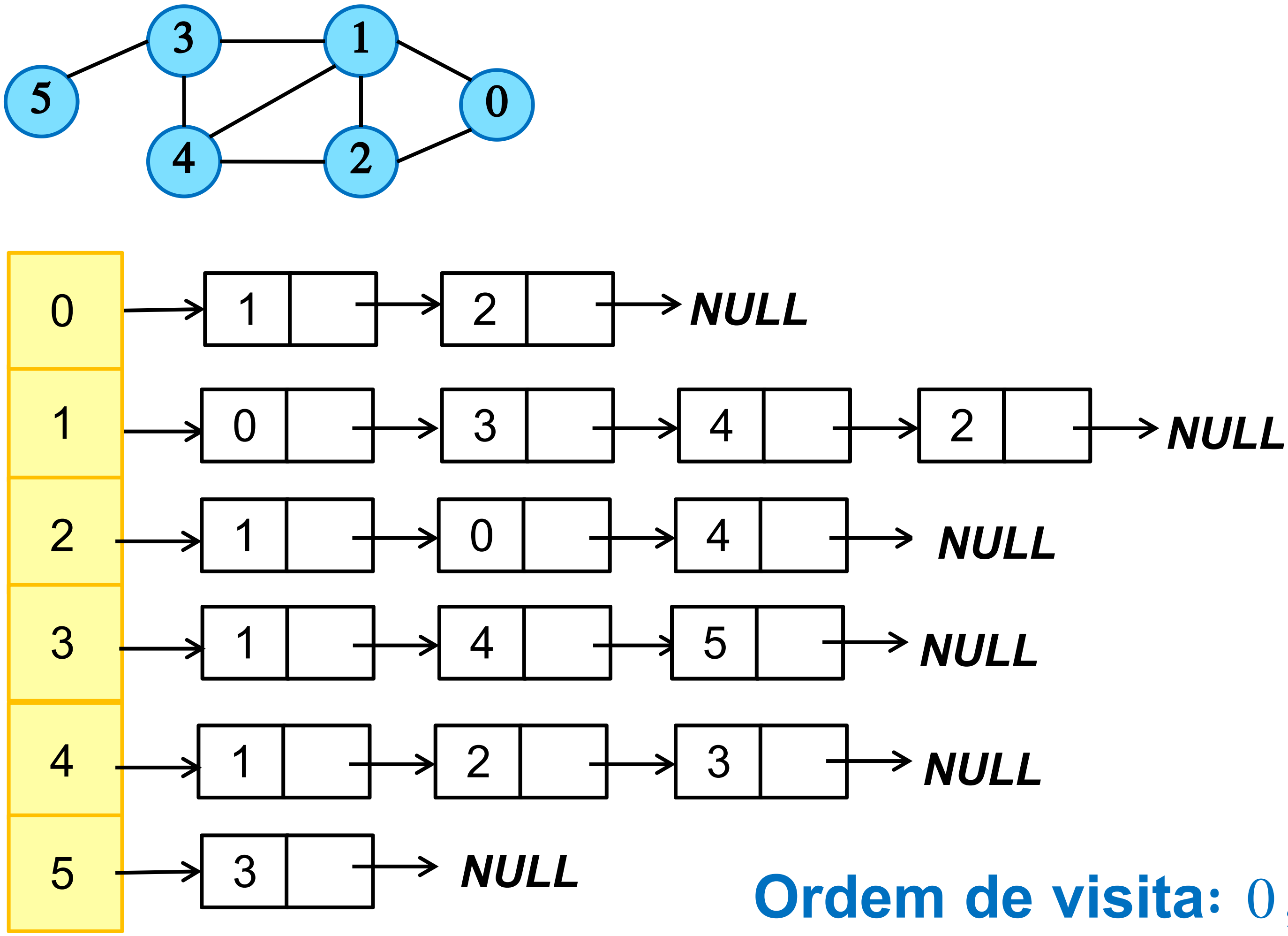
Ordem de visita: 0, 1, 3, 4, 2, 5

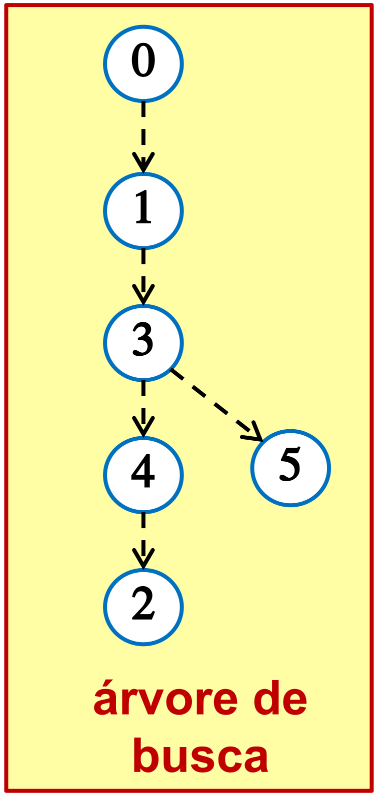
Busca em profundidade: exemplo



Ordem de visita: 0, 1, 3, 4, 2, 5

Busca em profundidade: exemplo





Ordem de visita: 0, 1, 3, 4, 2, 5

Busca em profundidade: algoritmo

- Precisa **armazenar os vértices já visitados**
  - **Vetor binário** onde cada posição indica se um vértice foi visitado ou não

Busca em profundidade: algoritmo

- Precisa **armazenar os vértices já visitados**
  - **Vetor binário** onde cada posição indica se um vértice foi visitado ou não
  - **Vetor de inteiros** onde cada posição indica a ordem na qual um vértice foi visitado
    - ZERO indica não visitado

Busca em profundidade: algoritmo

- Precisa **armazenar os vértices já visitados**
  - **Vetor binário** onde cada posição indica se um vértice foi visitado ou não
  - **Vetor de inteiros** onde cada posição indica a ordem na qual um vértice foi visitado
    - ZERO indica não visitado
  - **Lista linear** onde os nós indicam os vértices e a sequência indica a ordem de visita

Busca em profundidade: algoritmo

- Precisa **armazenar os vértices já visitados**
  - **Vetor binário** onde cada posição indica se um vértice foi visitado ou não
  - **Vetor de inteiros** onde cada posição indica a ordem na qual um vértice foi visitado
    - ZERO indica não visitado
  - **Lista linear** onde os nós indicam os vértices e a sequência indica a ordem de visita

- Pode ser definida como:
  - Variável global
  - Passada como **parâmetro da função**

Busca em profundidade: algoritmo

- Precisa **armazenar os vértices já visitados**
  - **Vetor binário** onde cada posição indica se um vértice foi visitado ou não
  - **Vetor de inteiros** onde cada posição indica a ordem na qual um vértice foi visitado
    - ZERO indica não visitado
  - **Lista linear** onde os nós indicam os vértices e a sequência indica a ordem de visita

- Pode ser definida como:
  - Variável global
  - Passada como **parâmetro da função**

Busca em profundidade: algoritmo

busca_profundidade *(Grafo *G, int V,* ***int *visitado****)*

FIM

Busca em profundidade: algoritmo

busca_profundidade *(Grafo *G, int V, int *visitado)*

Marque V como visitado;

FIM

Busca em profundidade: algoritmo

***busca_profundidade** (Grafo *G, int V, int *visitado)*

Marque V como visitado;

Execute a operação desejada em V (ex: imprimir);

FIM

Busca em profundidade: algoritmo

busca_profundidade *(Grafo *G, int V, int *visitado)*

Marque V como visitado;

Execute a operação desejada em V (ex: imprimir);

PARA cada vértice Adj adjacente a V FAÇA

FIM_PARA

FIM

Busca em profundidade: algoritmo

```
busca_profundidade (Grafo *G, int V, int *visitado)
    Marque V como visitado;

    Execute a operação desejada em V (ex: imprimir);

    PARA cada vértice Adj adjacente a V FAÇA
        SE adj não foi visitado ENTÃO
            busca_profundidade(G, Adj, visitado);
        FIM_SE
    FIM_PARA

FIM
```

Busca em profundidade: algoritmo

Algoritmo de disparo da busca:

DFS *(Grafo *G, int V)*

FIM

Busca em profundidade: algoritmo

Algoritmo de disparo da busca:

DFS *(Grafo *G, int V)*

*Aloca o vetor **visitado** com **qtde_vertices** de inteiros;*

FIM

Busca em profundidade: algoritmo

Algoritmo de disparo da busca:

DFS *(Grafo *G, int V)*

*Aloca o vetor **visitado** com **qtde_vertices** de inteiros;*

Inicializa cada elemento do vetor com ZERO;

FIM

Busca em profundidade: algoritmo

Algoritmo de disparo da busca:

DFS *(Grafo *G, int V)*

*Aloca o vetor **visitado** com **qtde_vertices** de inteiros;*

Inicializa cada elemento do vetor com ZERO;

busca_profundidade(G, V, visitado);

FIM

Busca em profundidade: algoritmo

Algoritmo de disparo da busca:

DFS *(Grafo *G, int V)*

*Aloca o vetor **visitado** com **qtde_vertices** de inteiros;*

Inicializa cada elemento do vetor com ZERO;

busca_profundidade(G, V, visitado);

FIM

Explora somente vértices conectados (ideal para tratar grafos conexos)

Busca em profundidade: análise

- A função ***busca_profundidade()*** é chamada uma única vez para cada vértice
  - Marcar o vértice como visitado tem custo fixo
  - Executar a operação sobre o vértice tem custo fixo
  - Percorrer os vértices adjacentes depende do **seu grau de saída**

Busca em profundidade: análise

- A função ***busca_profundidade()*** é chamada uma única vez para cada vértice
  - Marcar o vértice como visitado tem custo fixo
  - Executar a operação sobre o vértice tem custo fixo
  - Percorrer os vértices adjacentes depende do **seu grau de saída**

- Custo total da função está diretamente relacionado com a quantidade de arestas do grafo (***O(|A|)***)

Busca em profundidade: análise

A função ***busca_profundidade()*** é chamada uma única vez para cada vértice

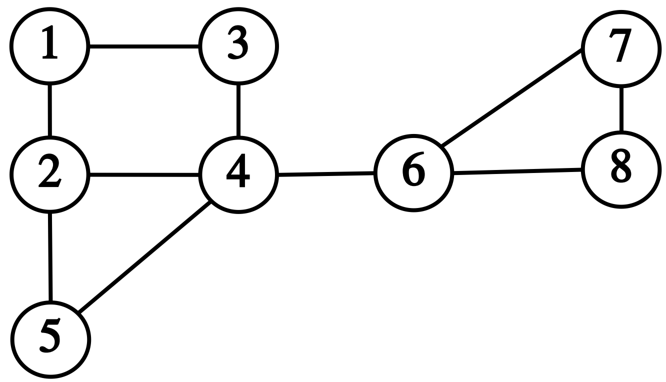
- Marcar o vértice como visitado tem custo fixo
- Executar a operação sobre o vértice tem custo fixo
- Percorrer os vértices adjacentes depende do **seu grau de saída**

Custo total da função está diretamente relacionado com a quantidade de arestas do grafo (***O(|A|)***)

Considerando a função de disparo, **o custo total da busca em profundidade é *O(|V|+|A|)***

Busca em profundidade: exercícios

1- Faça o teste de mesa do algoritmo de busca em profundidade para o grafo abaixo, iniciando pelo vértice 1:



2- Implemente o algoritmo de modo que o tratamento dos vértices seja mostrar o vetor de visitados, a lista de adjacentes do vértice atual, qual vértice será usado na próxima chamada da função e a ordem de visita da busca. Verifique se o resultado apresentado é igual ao do seu teste de mesa.

Busca em profundidade: exercícios

3- Altere a função de disparo de modo a explorar todos os vértices (**grafo não conexo**)

4- Implemente a **versão iterativa** da busca em profundidade (**dica:** use uma **pilha** para guardar os próximos vértices adjacentes a visitar). Neste caso, não existe a necessidade da função de disparo, uma vez que o vetor visitado pode ser criado no início do processo de busca. Por fim, faça a análise de complexidade deste algoritmo.

5- Considerando que o grafo é representado através de uma matriz de adjacências, refaça as implementações recursivas e iterativas e as respectivas análises da complexidade do algoritmo.

Percorrimento de um grafo

Busca em largura

Busca em largura

- Estratégia que visa **explorar os vértices vizinhos** antes de aumentar a profundidade da busca
  - Aumenta **uniformemente** a **largura da fronteira** entre os vértices descobertos e não descobertos
  - Descobre todos os vértices a uma distância *k* do vértice de origem antes de descobrir qualquer vértice a uma distância *k+1*

Busca em largura

Estratégia que visa **explorar os vértices vizinhos** antes de aumentar a profundidade da busca

- Aumenta **uniformemente** a **largura da fronteira** entre os vértices descobertos e não descobertos
- Descobre todos os vértices a uma distância *k* do vértice de origem antes de descobrir qualquer vértice a uma distância *k+1*

Ideia básica:

- Explorar **todas as arestas** do vértice atual
- Quando todas as arestas tiverem sido exploradas, a busca passa a explorar as arestas dos seus vértices adjacentes

Busca em largura

Estratégia que visa **explorar os vértices vizinhos** antes de aumentar a profundidade da busca

- Aumenta **uniformemente** a **largura da fronteira** entre os vértices descobertos e não descobertos
- Descobre todos os vértices a uma distância *k* do vértice de origem antes de descobrir qualquer vértice a uma distância *k+1*

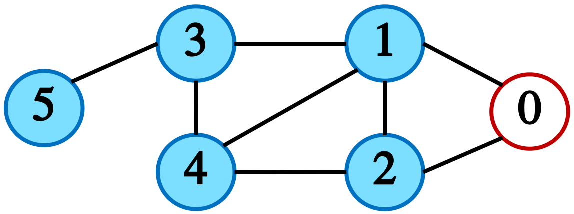
Ideia básica:

- Explorar **todas as arestas** do vértice atual
- Quando todas as arestas tiverem sido exploradas, a busca passa a explorar as arestas dos seus vértices adjacentes

Exemplos de utilização:

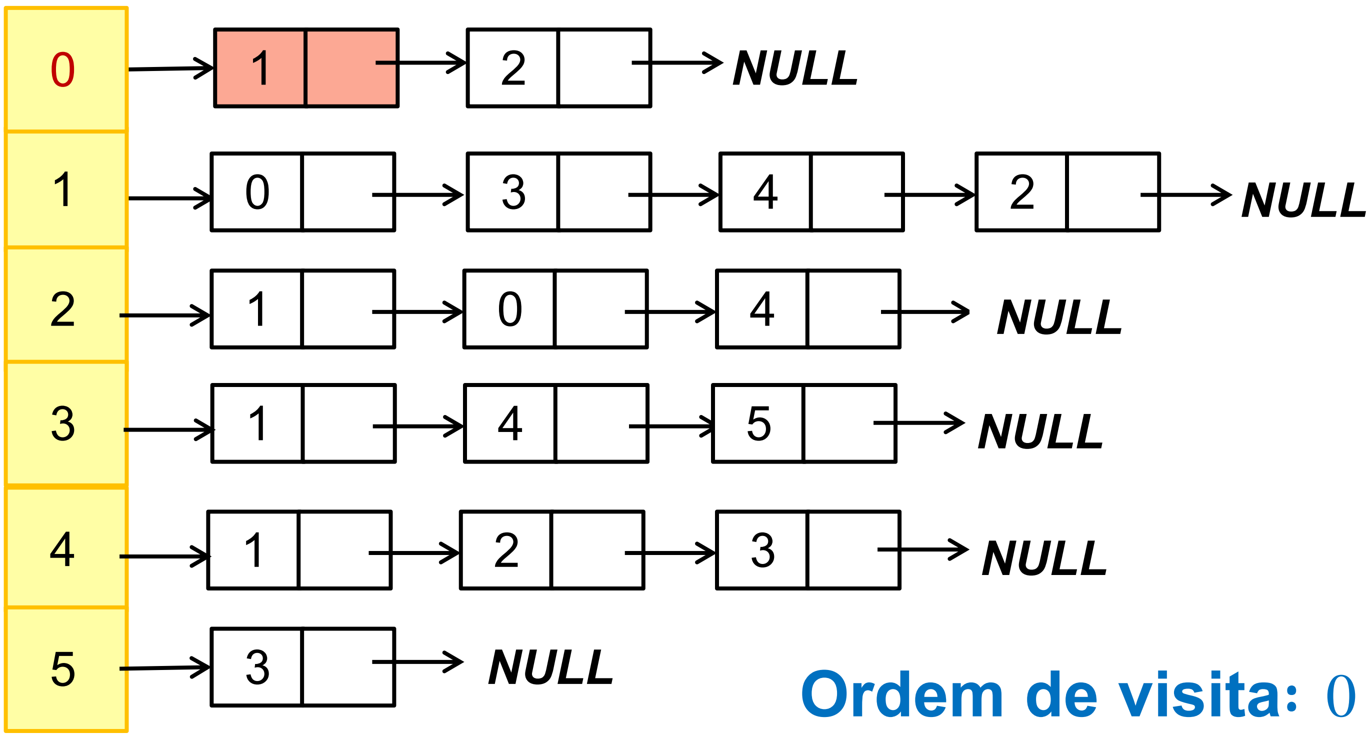
- Verificar bipartição de um grafo
- Achar o menor caminho entre 2 vértices

Busca em largura: exemplo



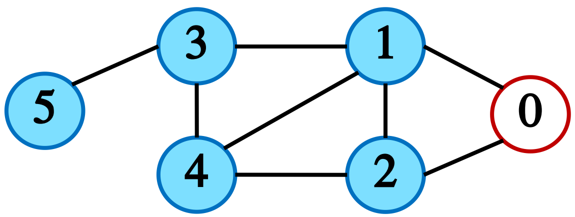
vértice inicial

0

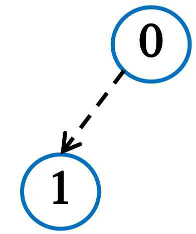


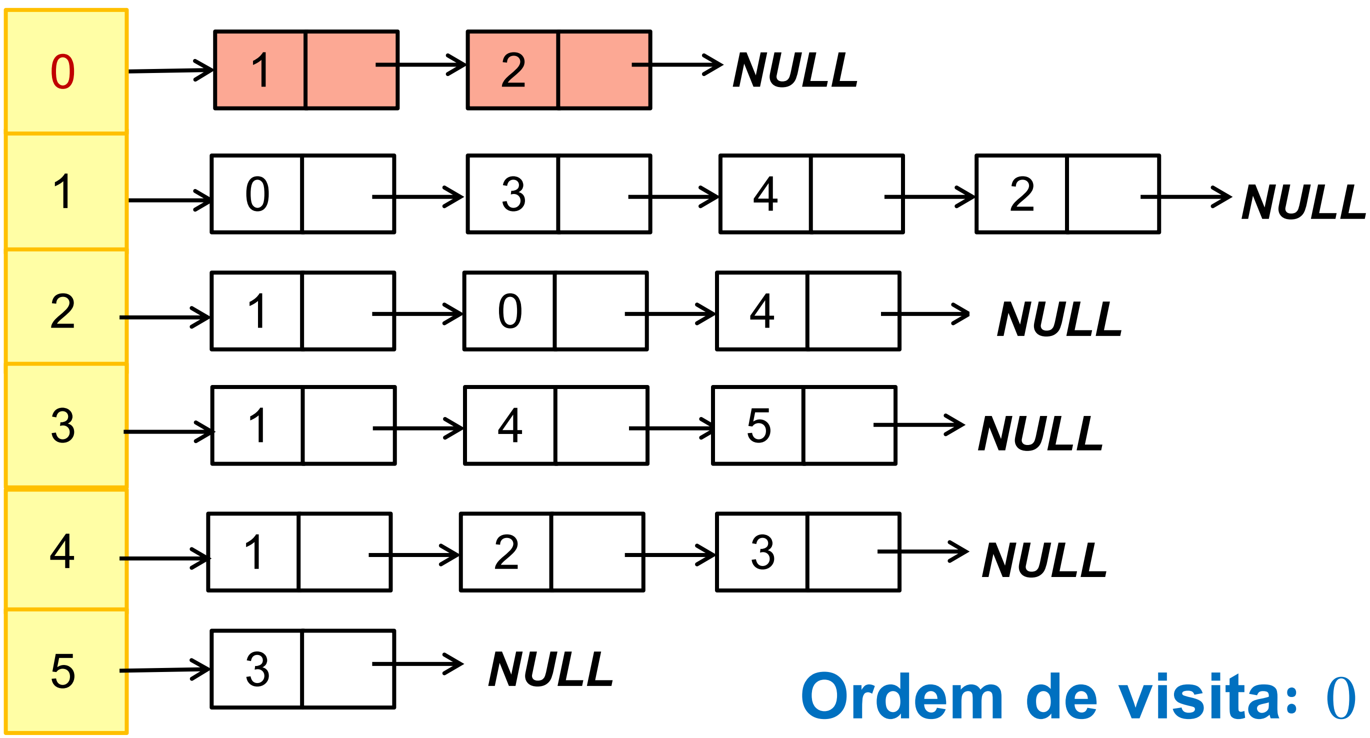
Ordem de visita: 0

Busca em largura: exemplo



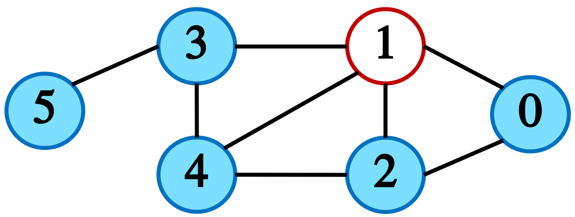
vértice inicial



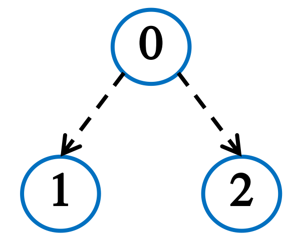


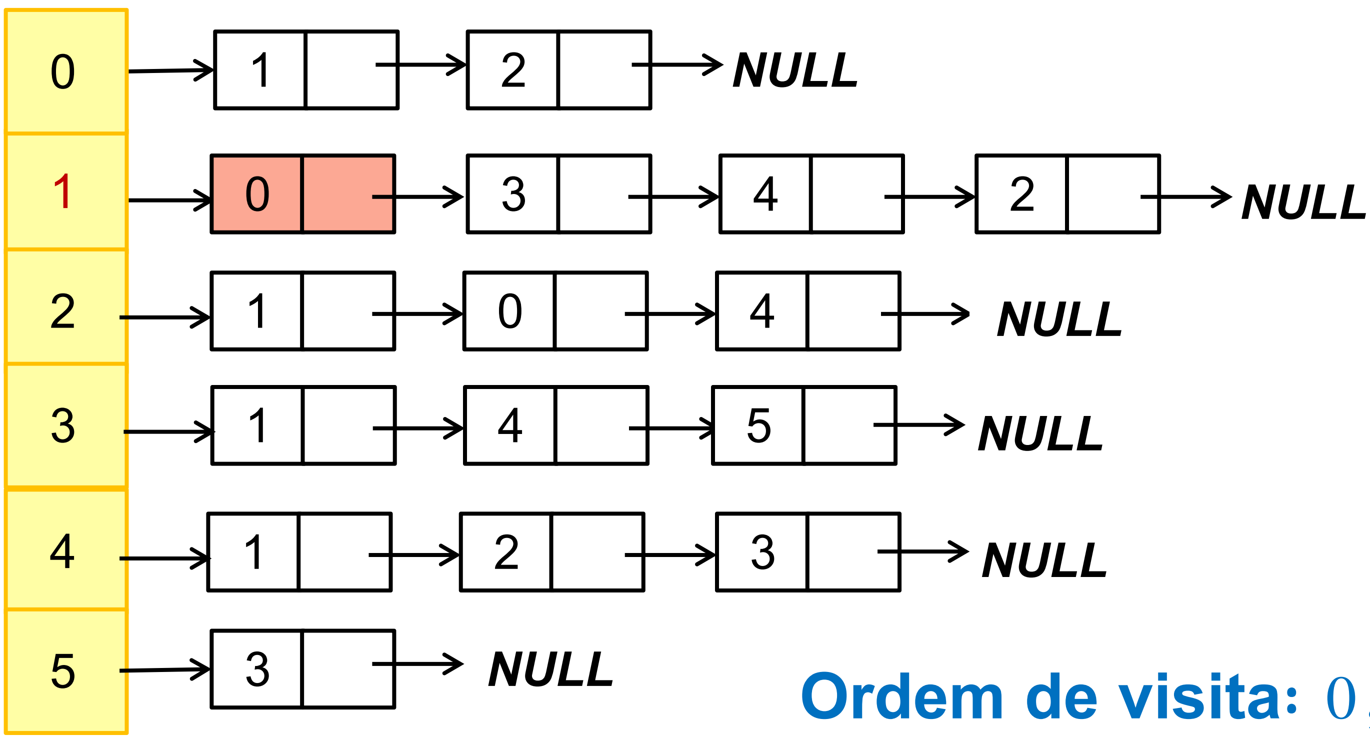
Ordem de visita: 0

Busca em largura: exemplo



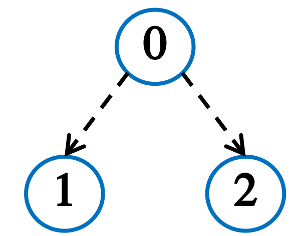
vértice inicial

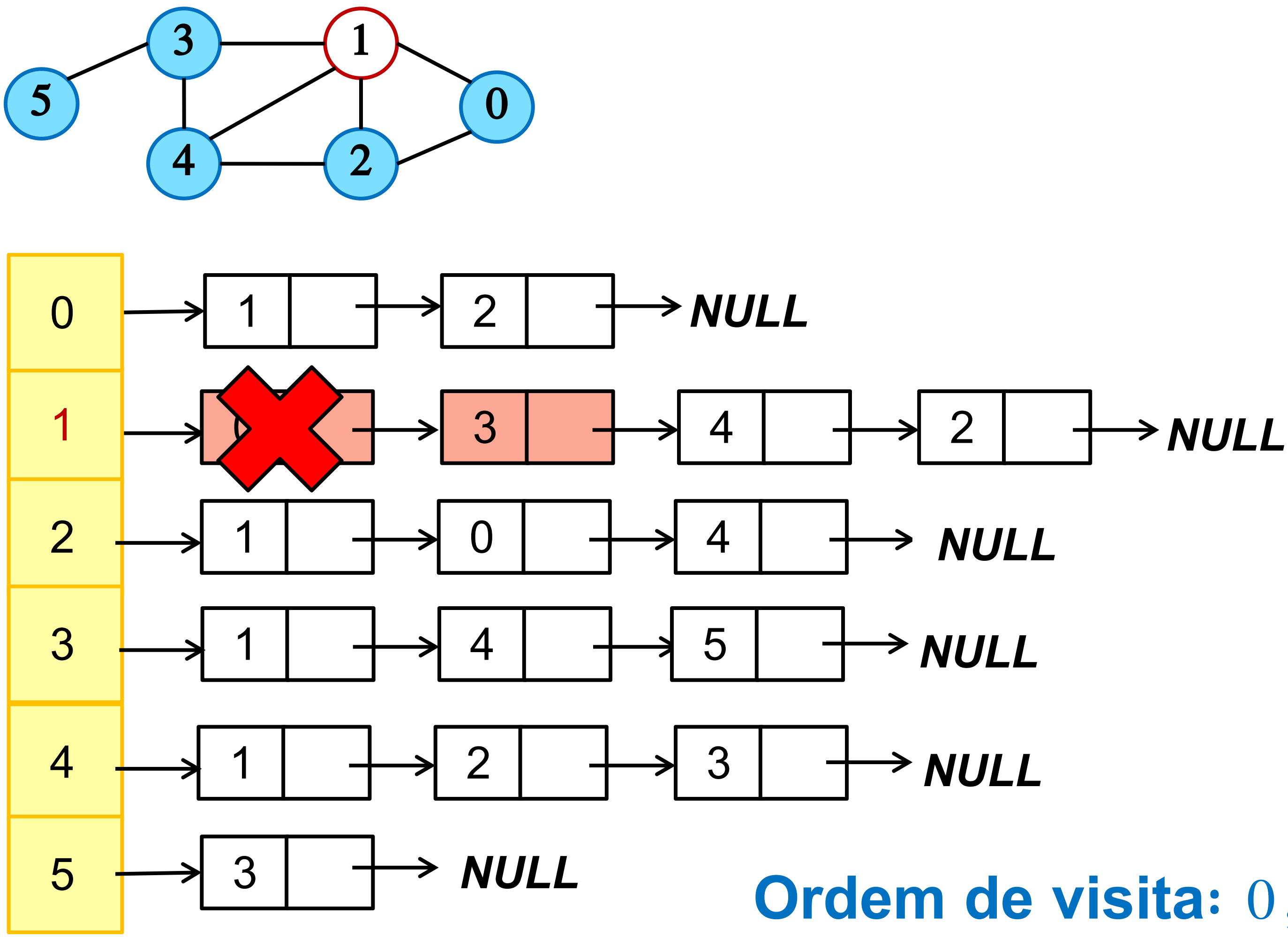




Ordem de visita: 0 , 1

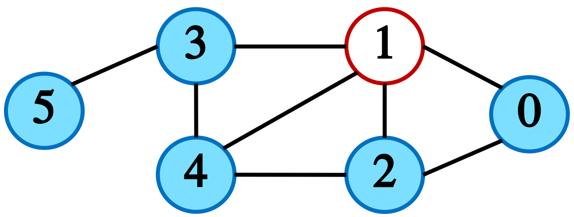
Busca em largura: exemplo



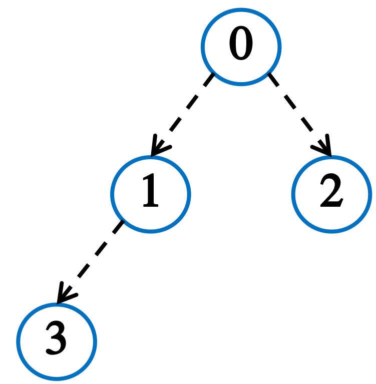


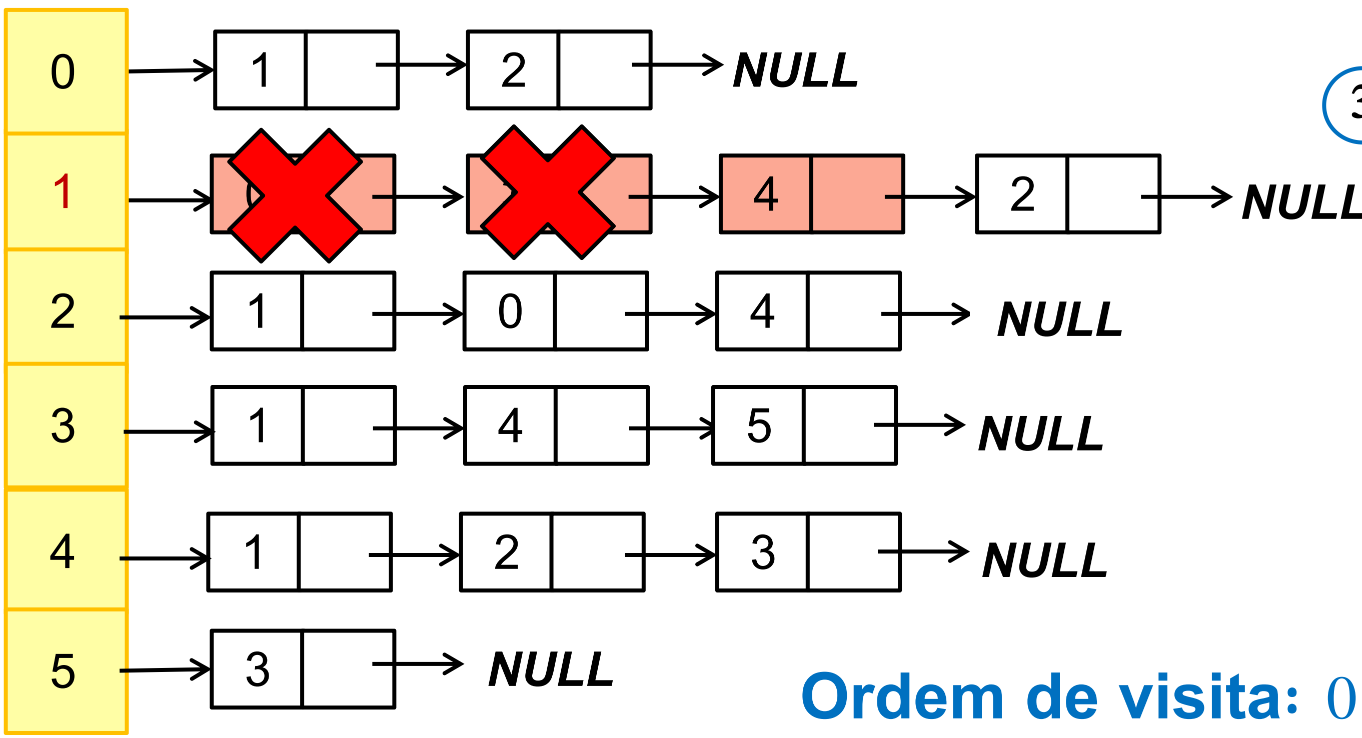
Ordem de visita: 0 , 1

Busca em largura: exemplo



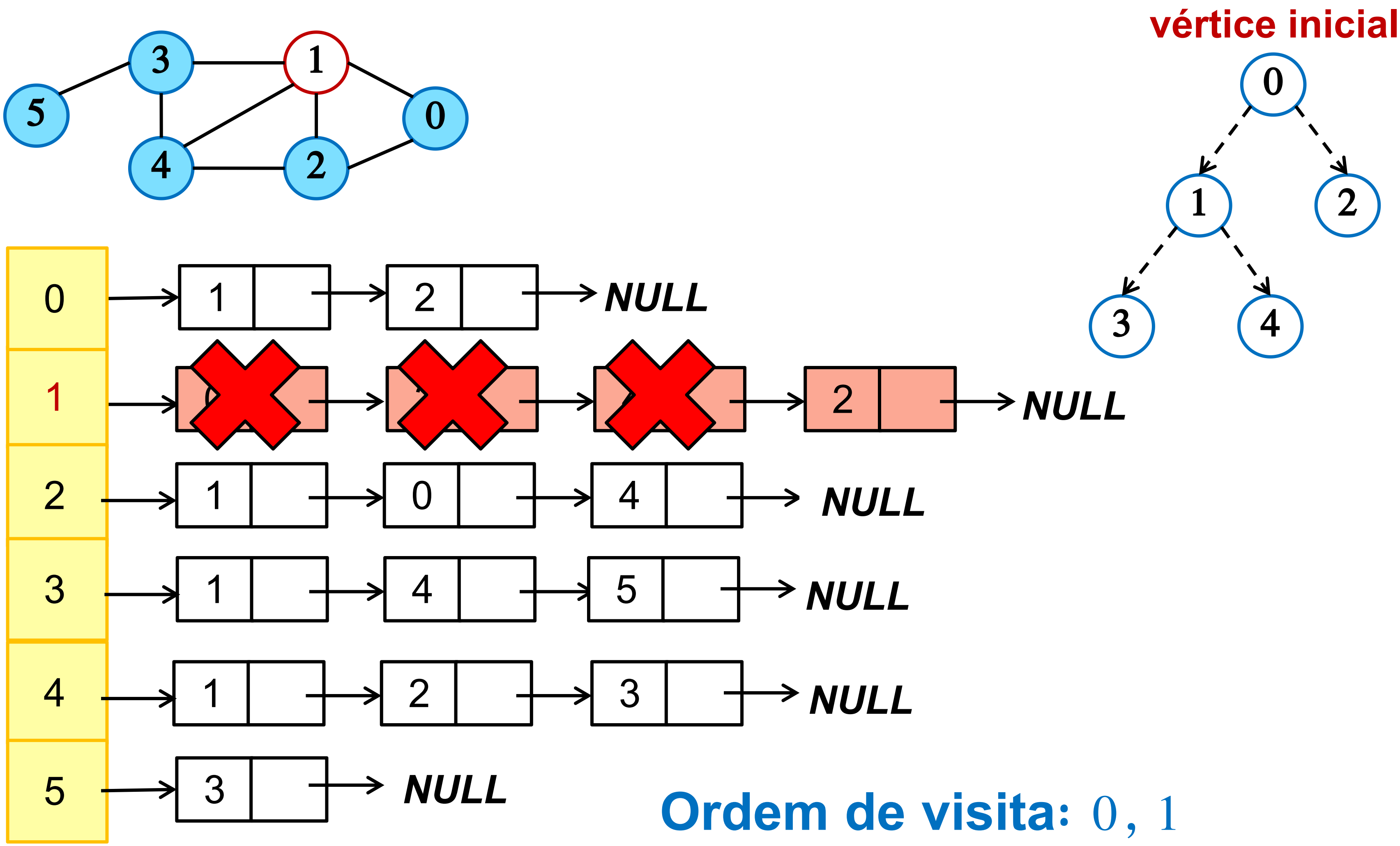
vértice inicial





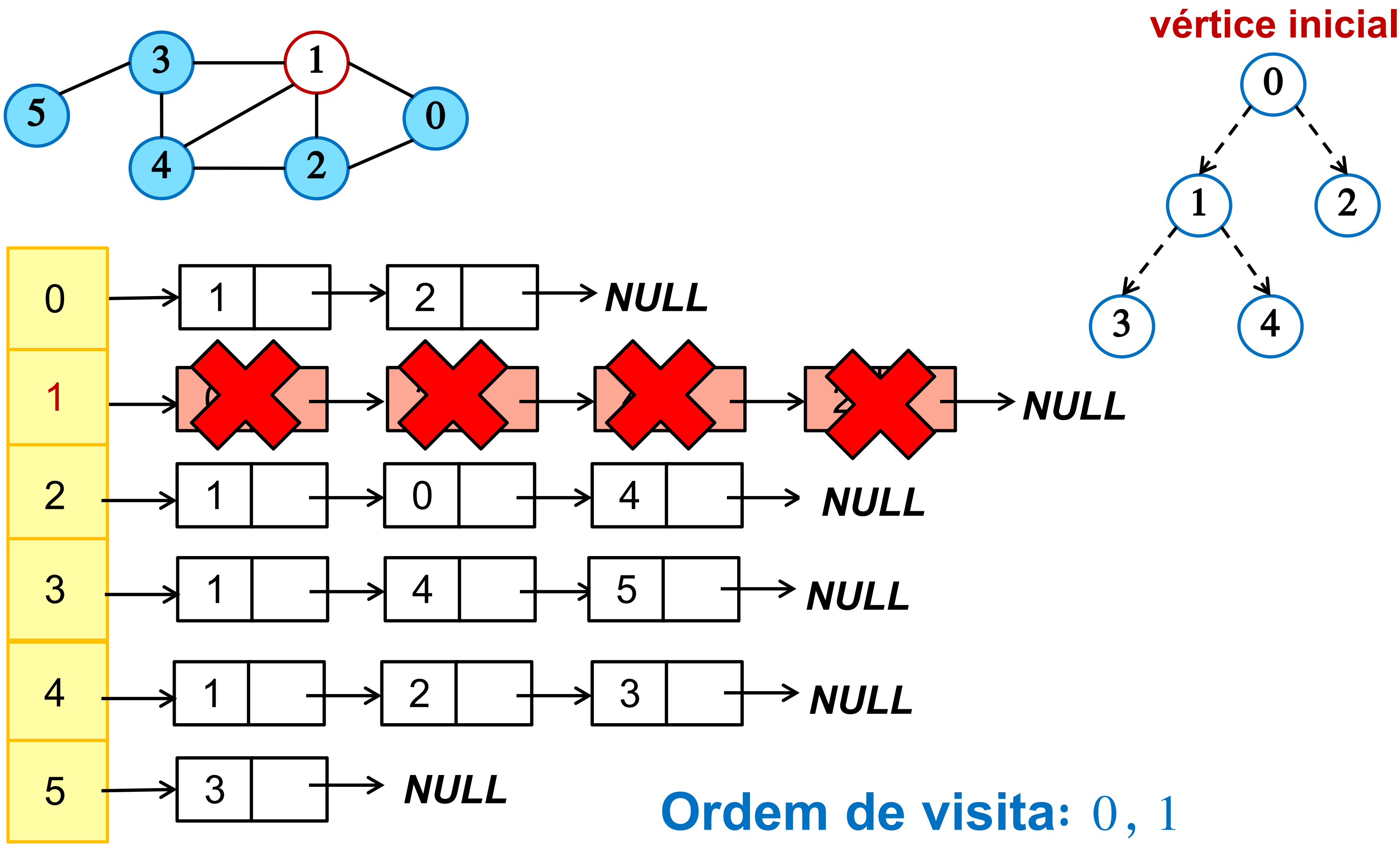
Ordem de visita: 0 , 1

Busca em largura: exemplo



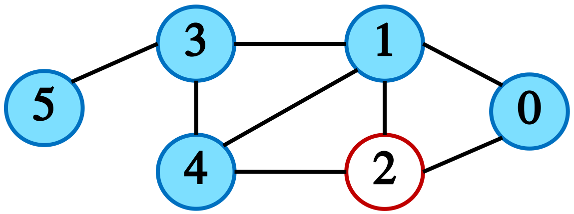
Ordem de visita: 0, 1

Busca em largura: exemplo

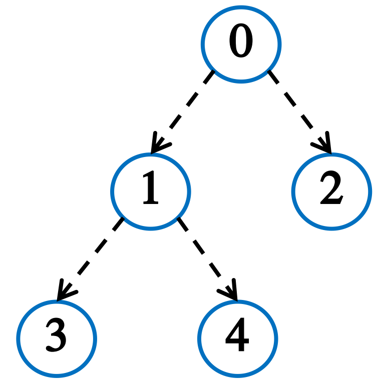


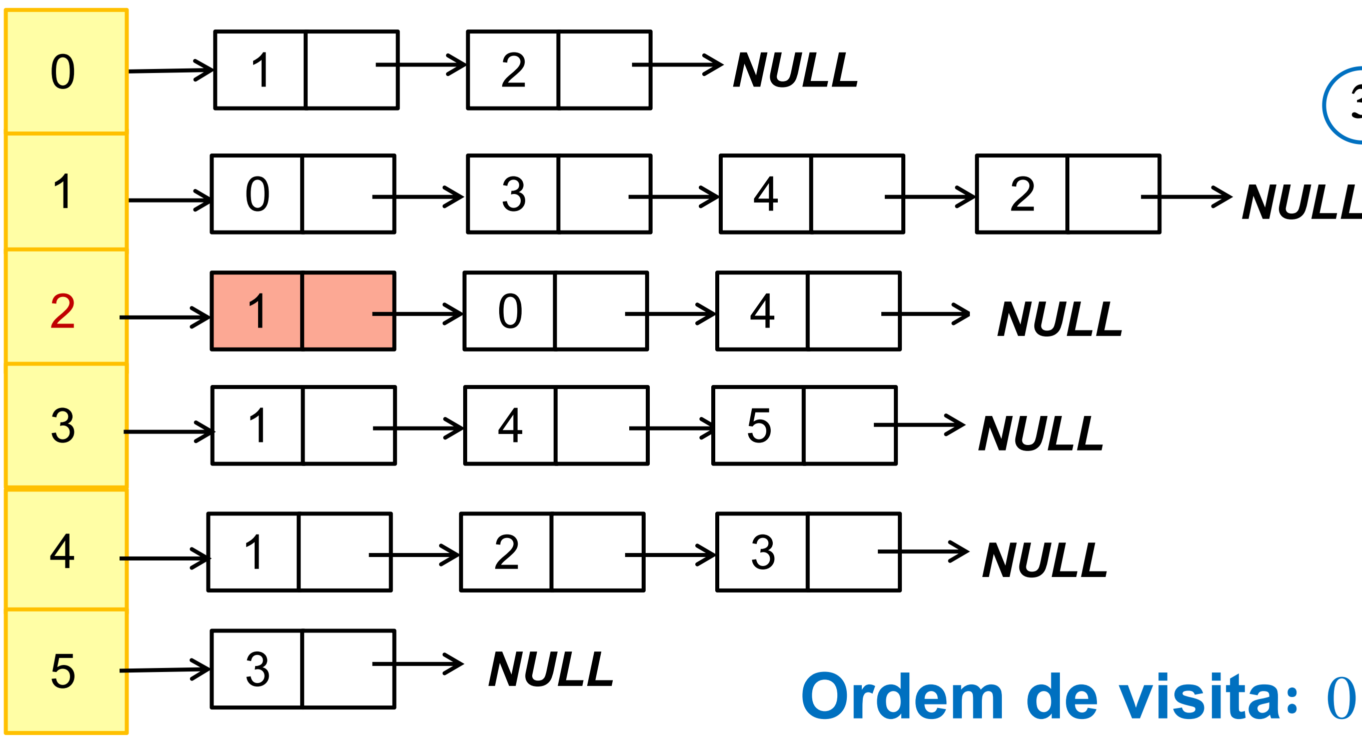
Ordem de visita: 0 , 1

Busca em largura: exemplo



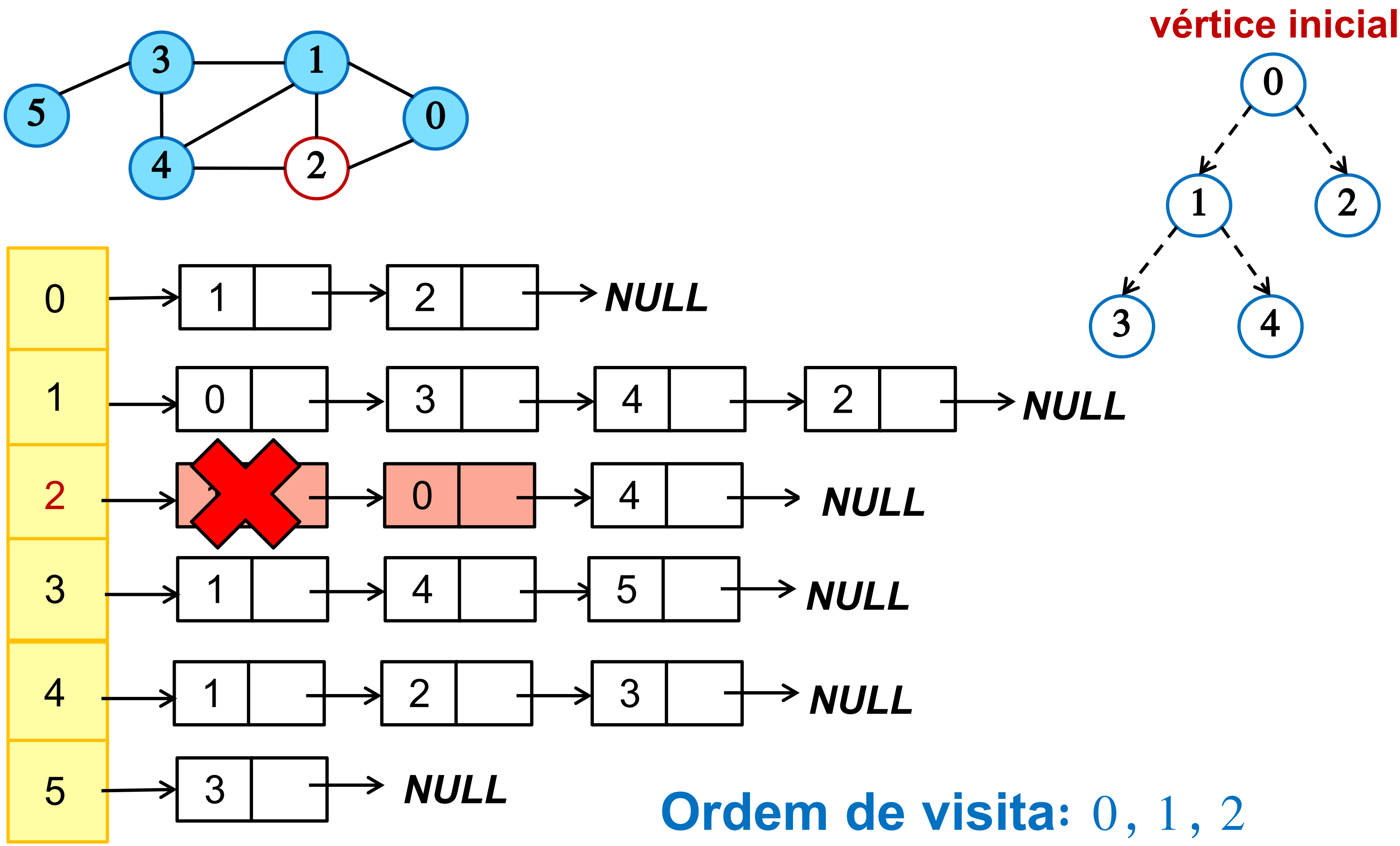
vértice inicial





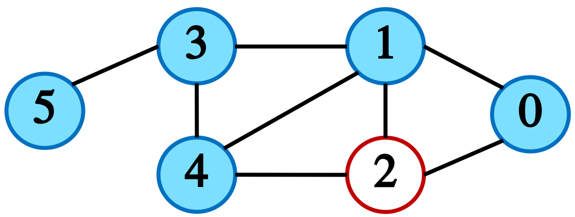
Ordem de visita: 0 , 1 , 2

Busca em largura: exemplo

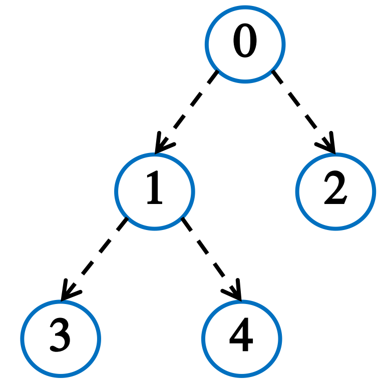


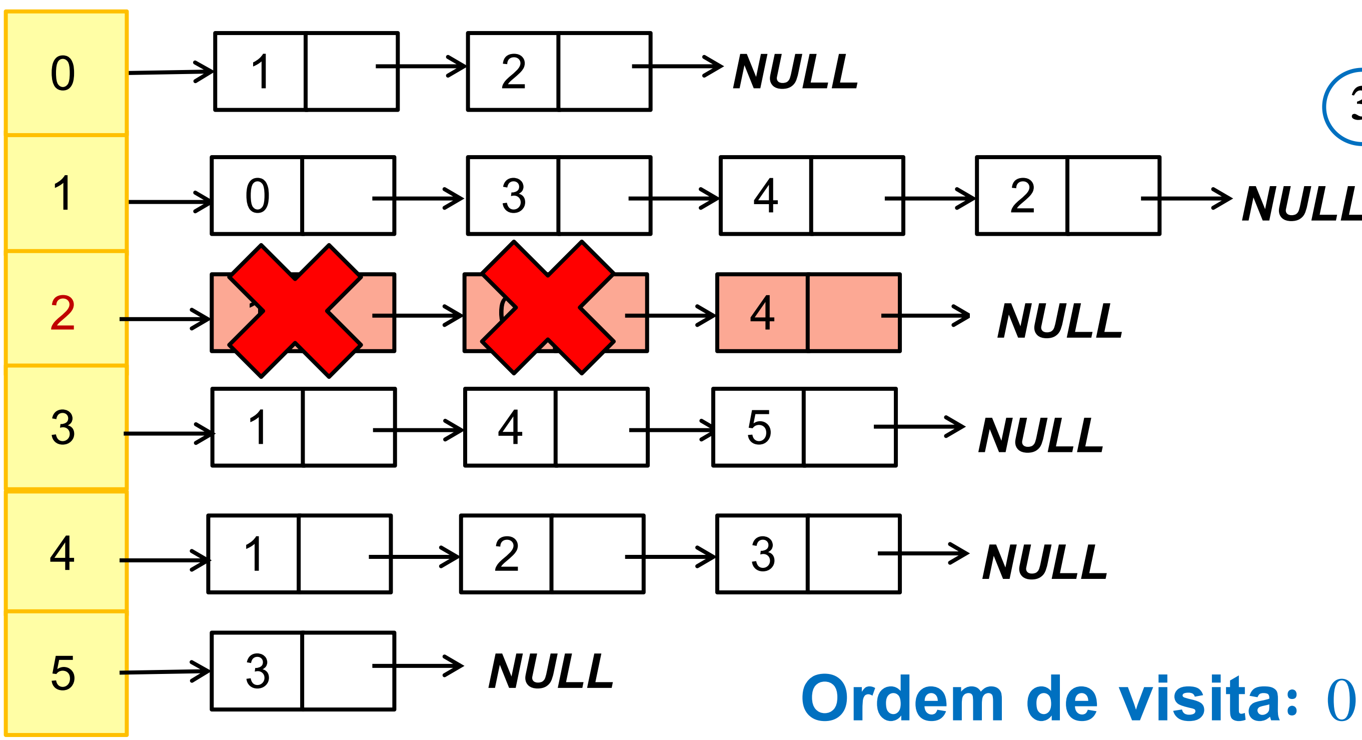
Ordem de visita: 0 , 1 , 2

Busca em largura: exemplo



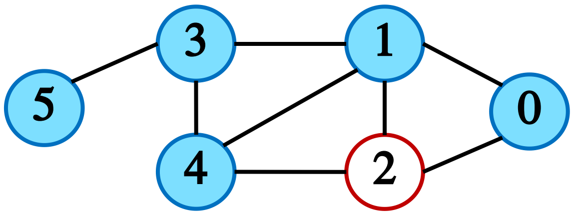
vértice inicial



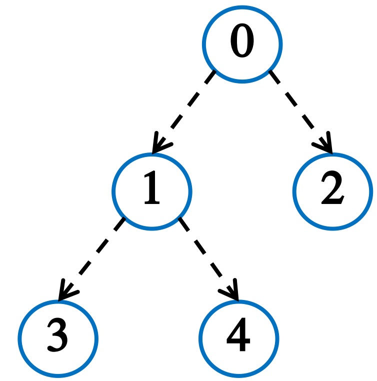


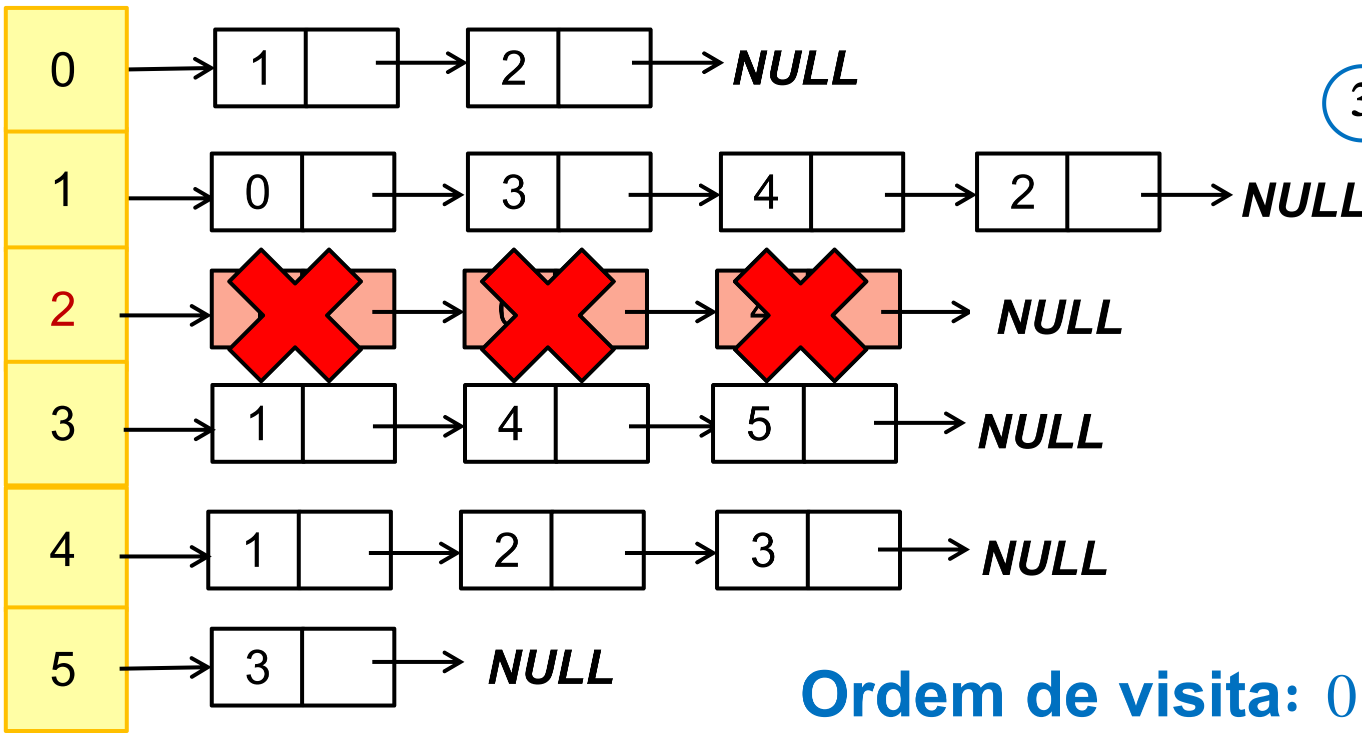
Ordem de visita: 0 , 1 , 2

Busca em largura: exemplo



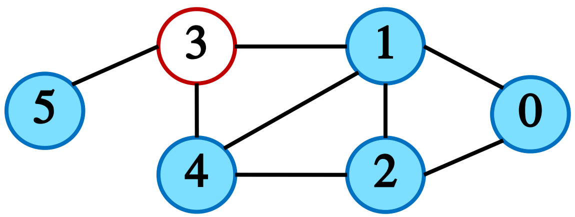
vértice inicial



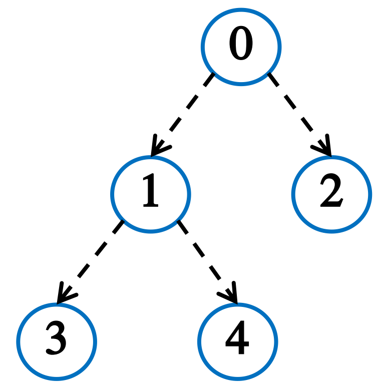


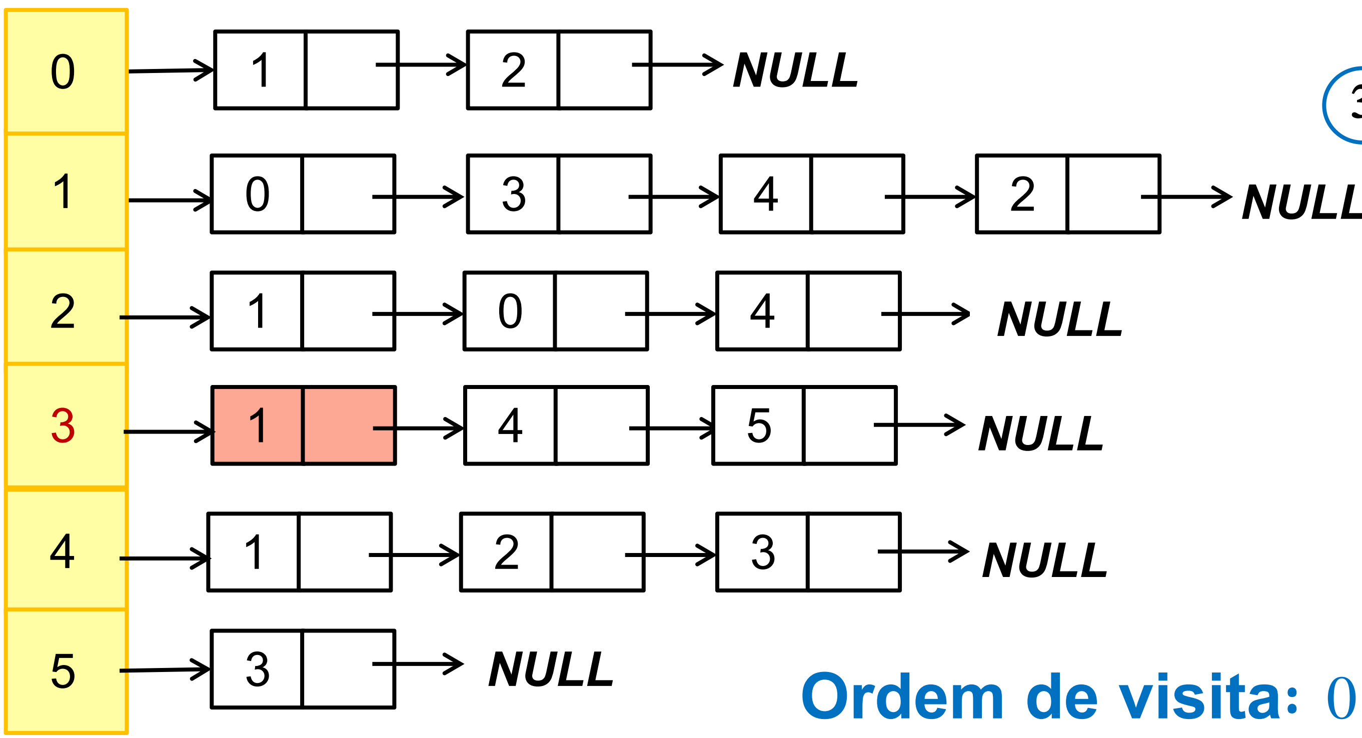
Ordem de visita: 0 , 1 , 2

Busca em largura: exemplo



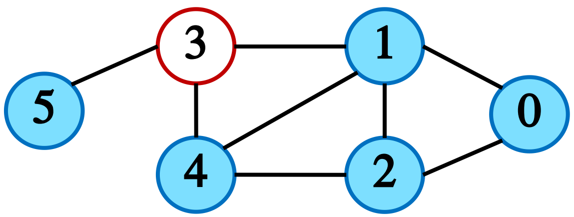
vértice inicial

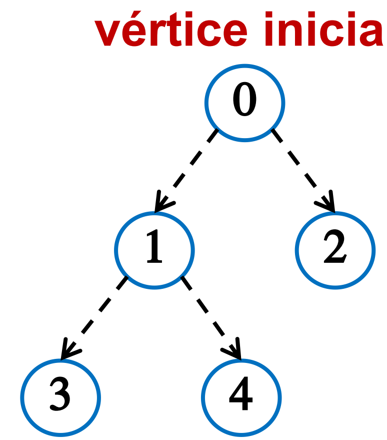


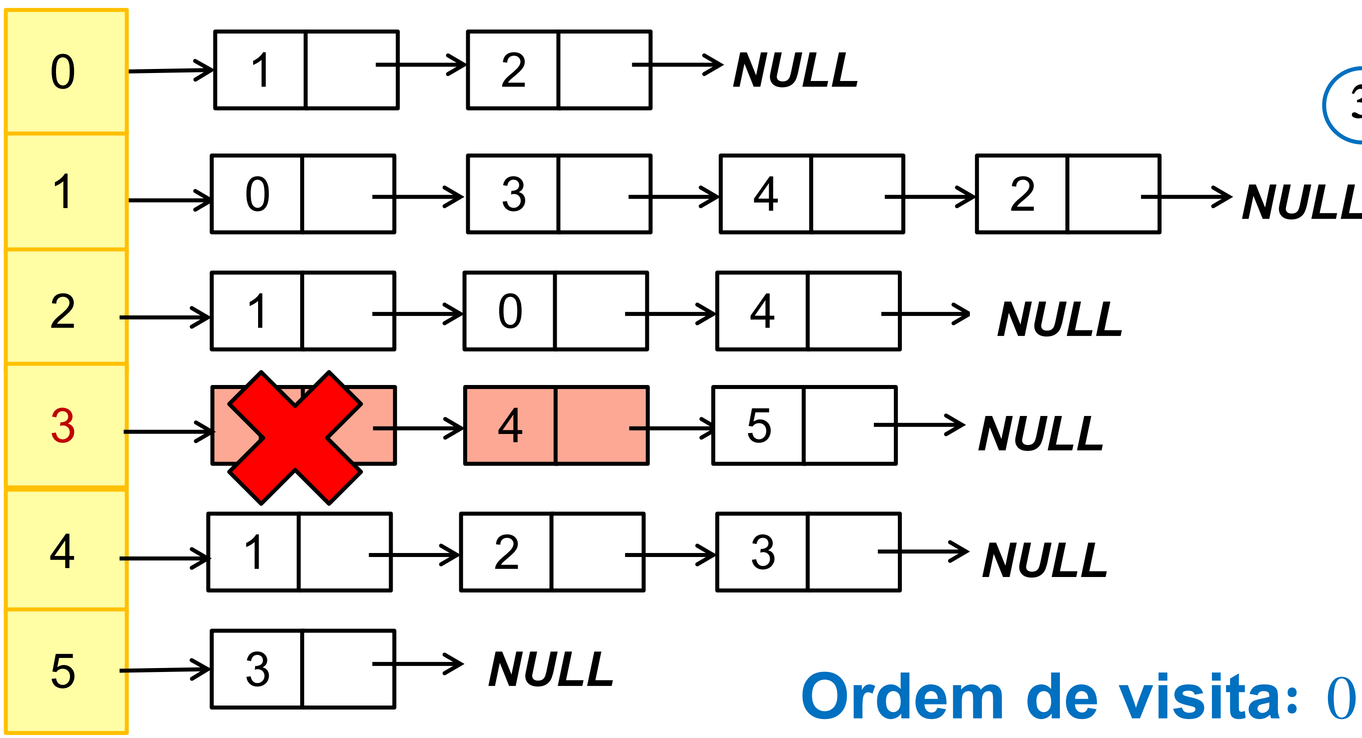


Ordem de visita: 0, 1, 2, 3

Busca em largura: exemplo

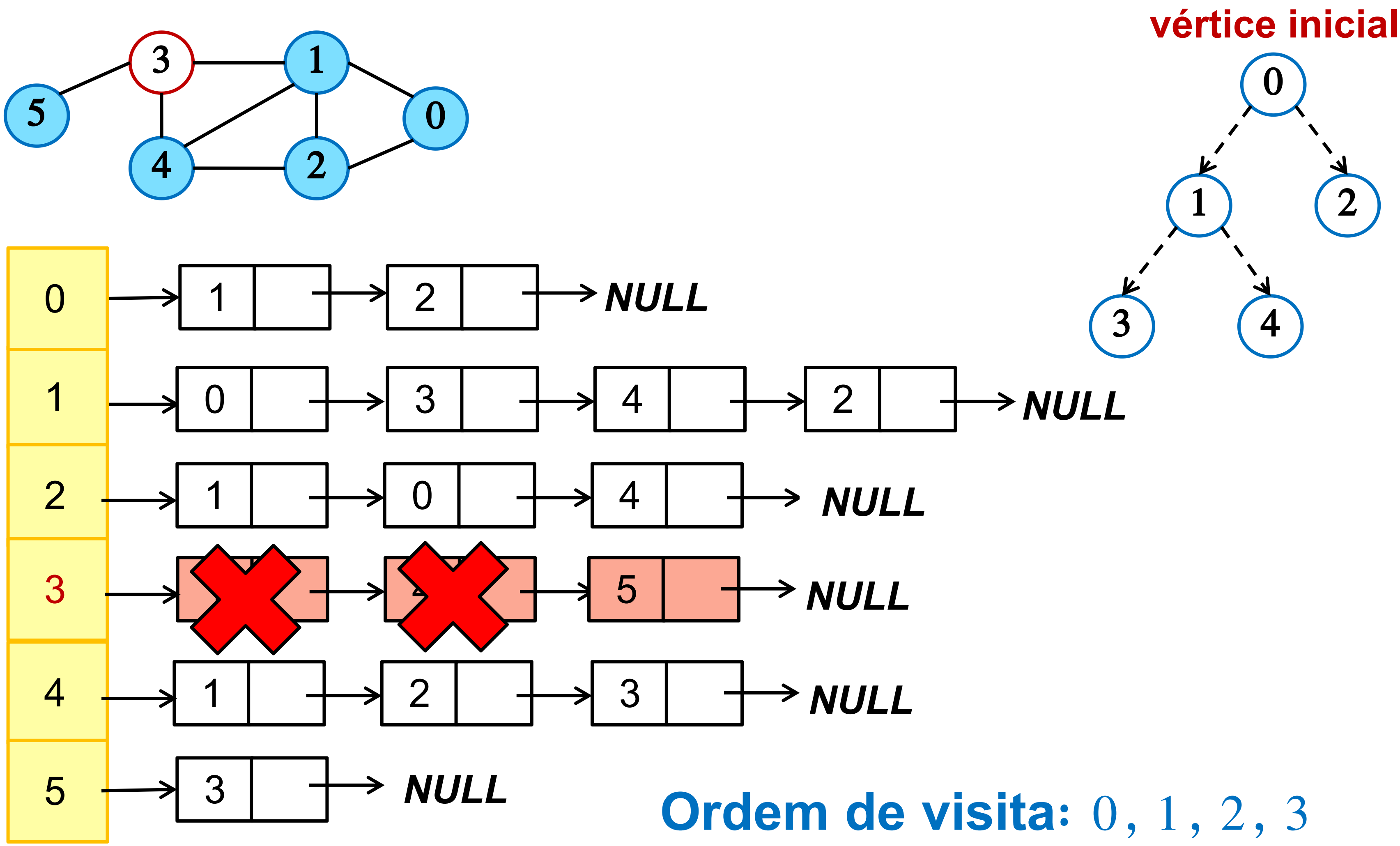






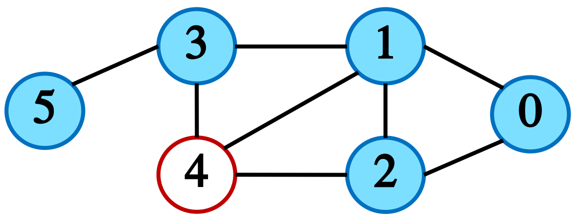
Ordem de visita: 0 , 1 , 2 , 3

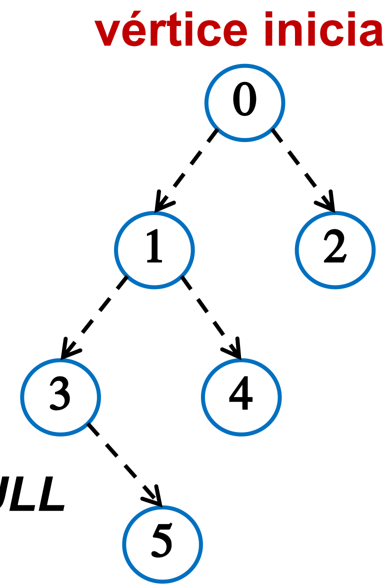
Busca em largura: exemplo

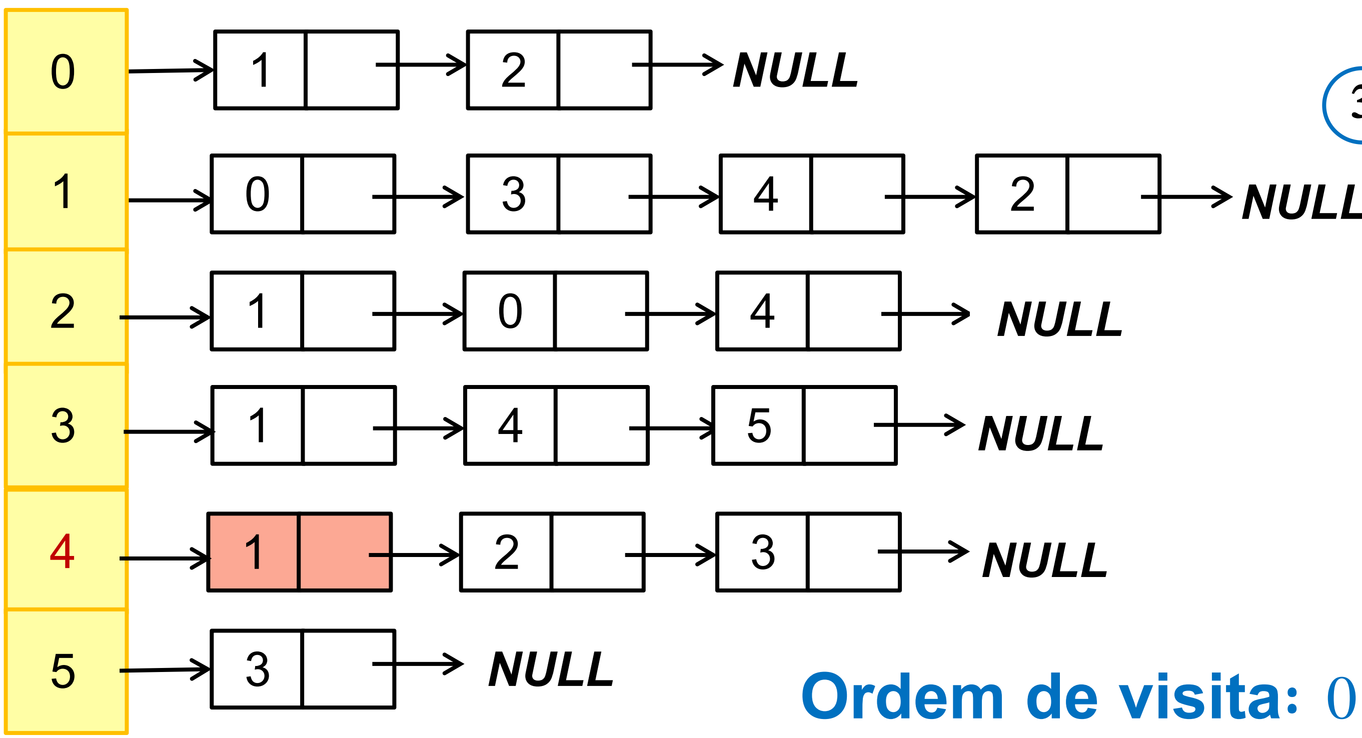


Ordem de visita: 0, 1, 2, 3

Busca em largura: exemplo

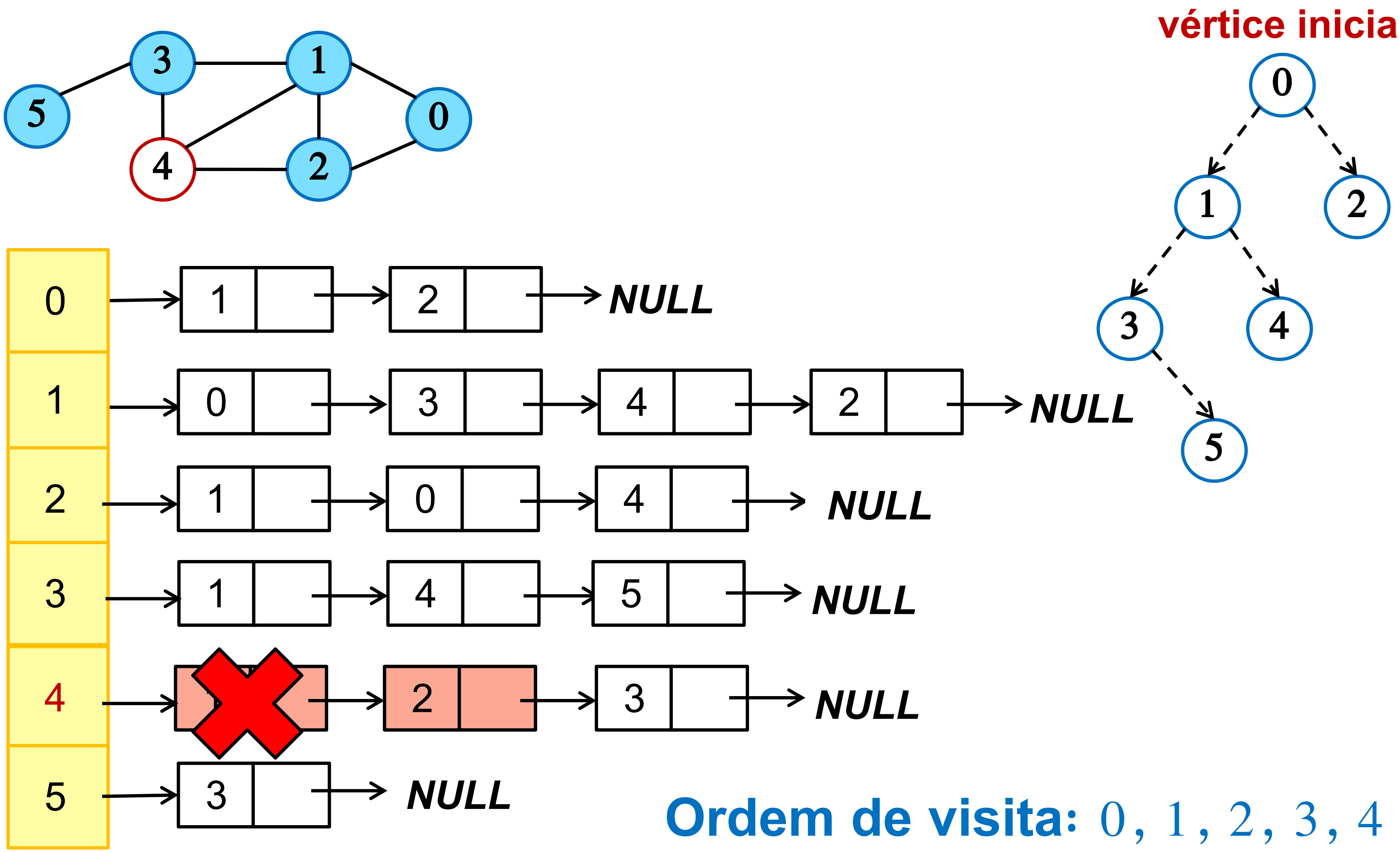






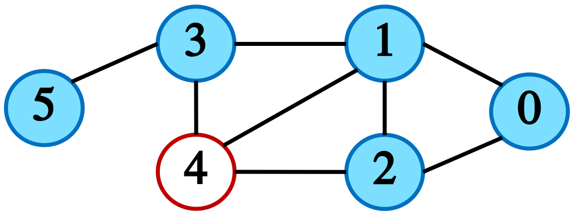
Ordem de visita: 0 , 1 , 2 , 3 , 4

Busca em largura: exemplo

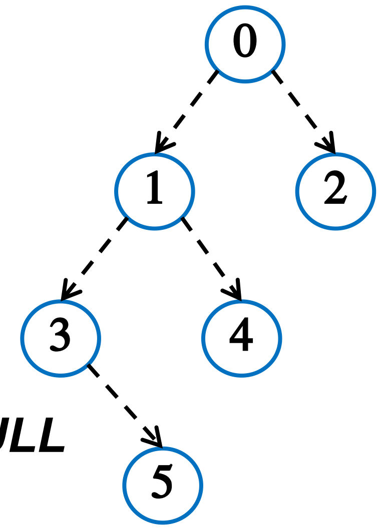


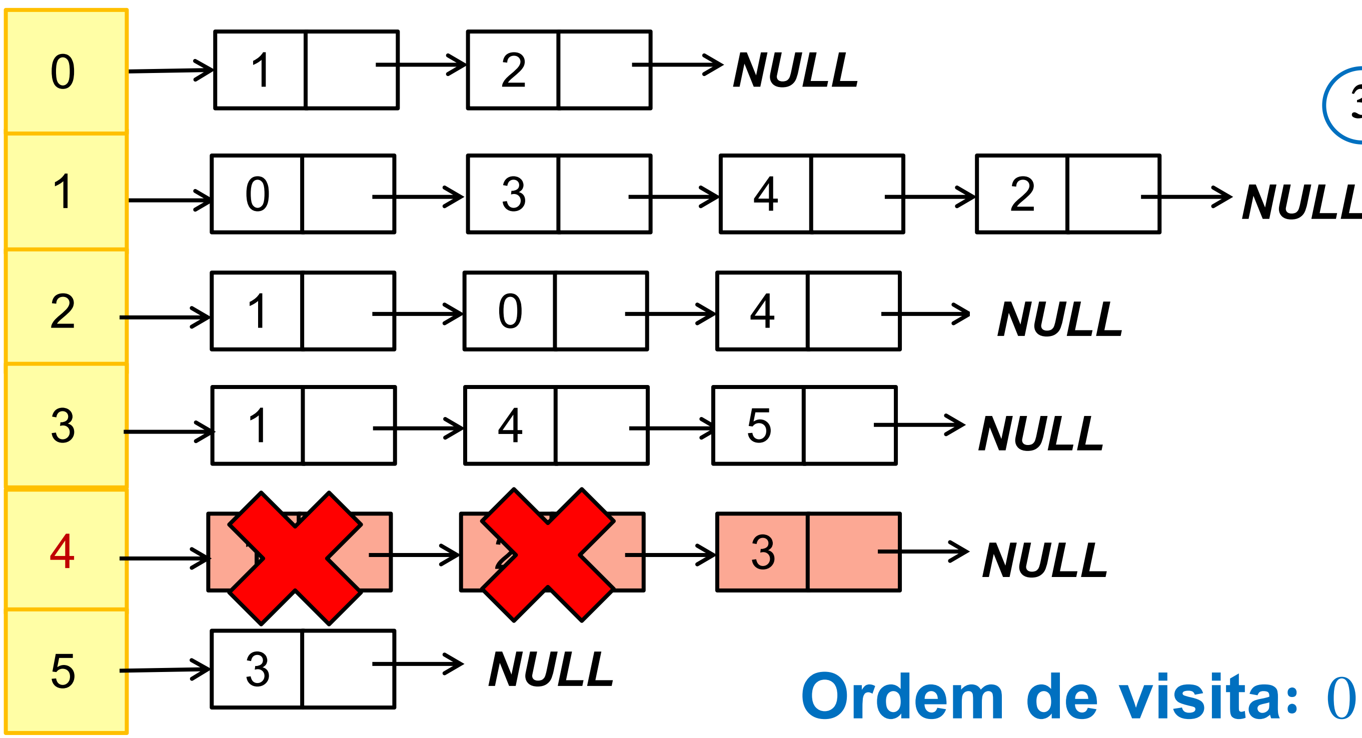
Ordem de visita: 0 , 1 , 2 , 3 , 4

Busca em largura: exemplo



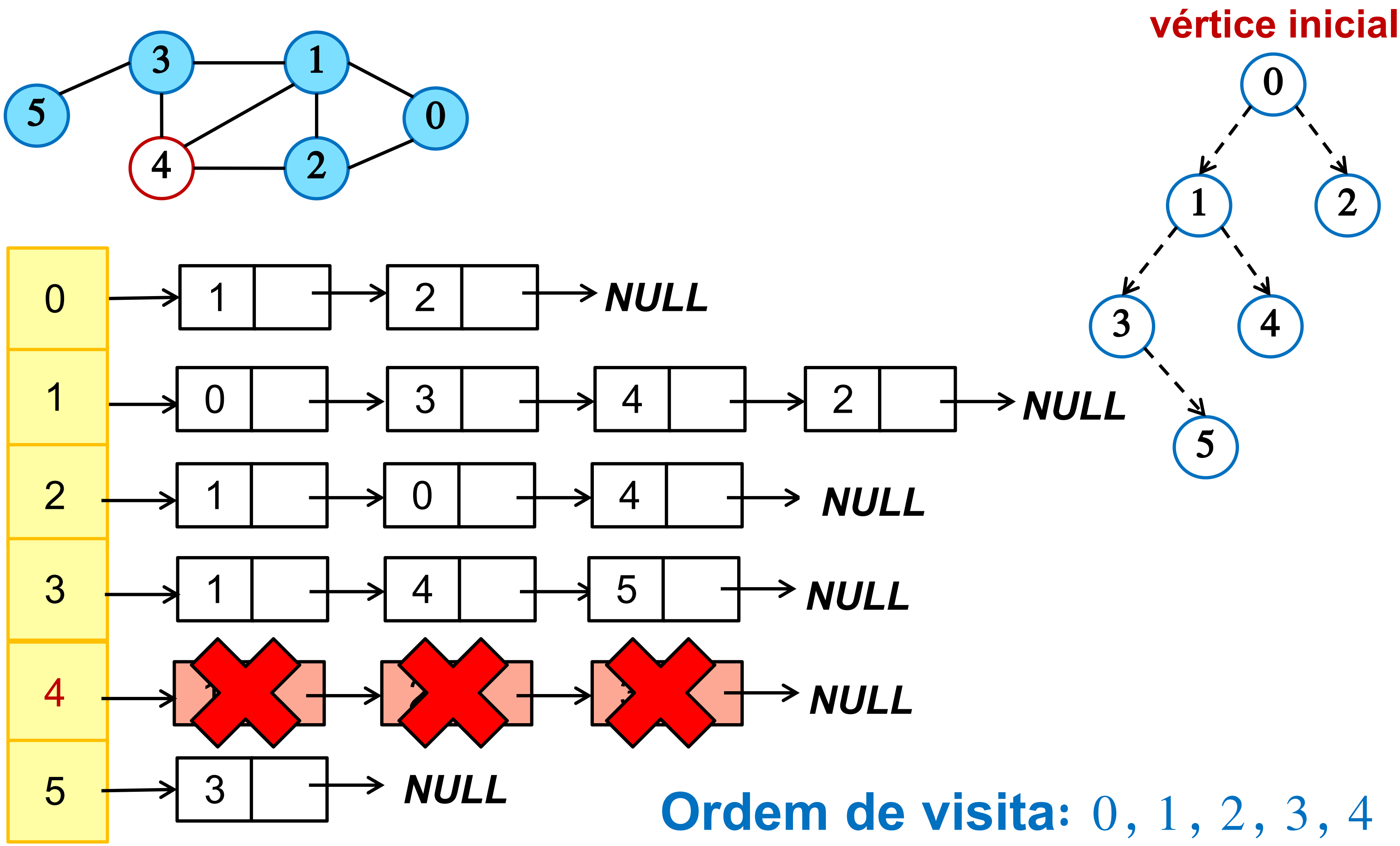
vértice inicial





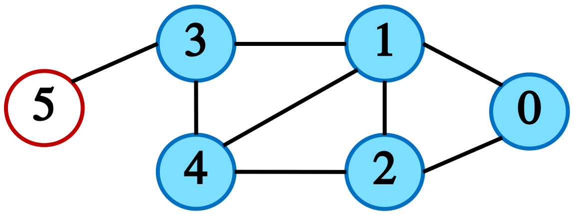
Ordem de visita: 0 , 1 , 2 , 3 , 4

Busca em largura: exemplo

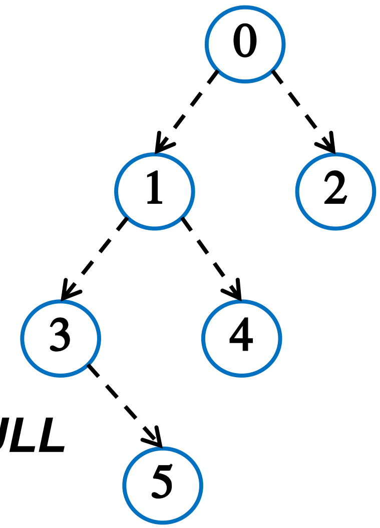


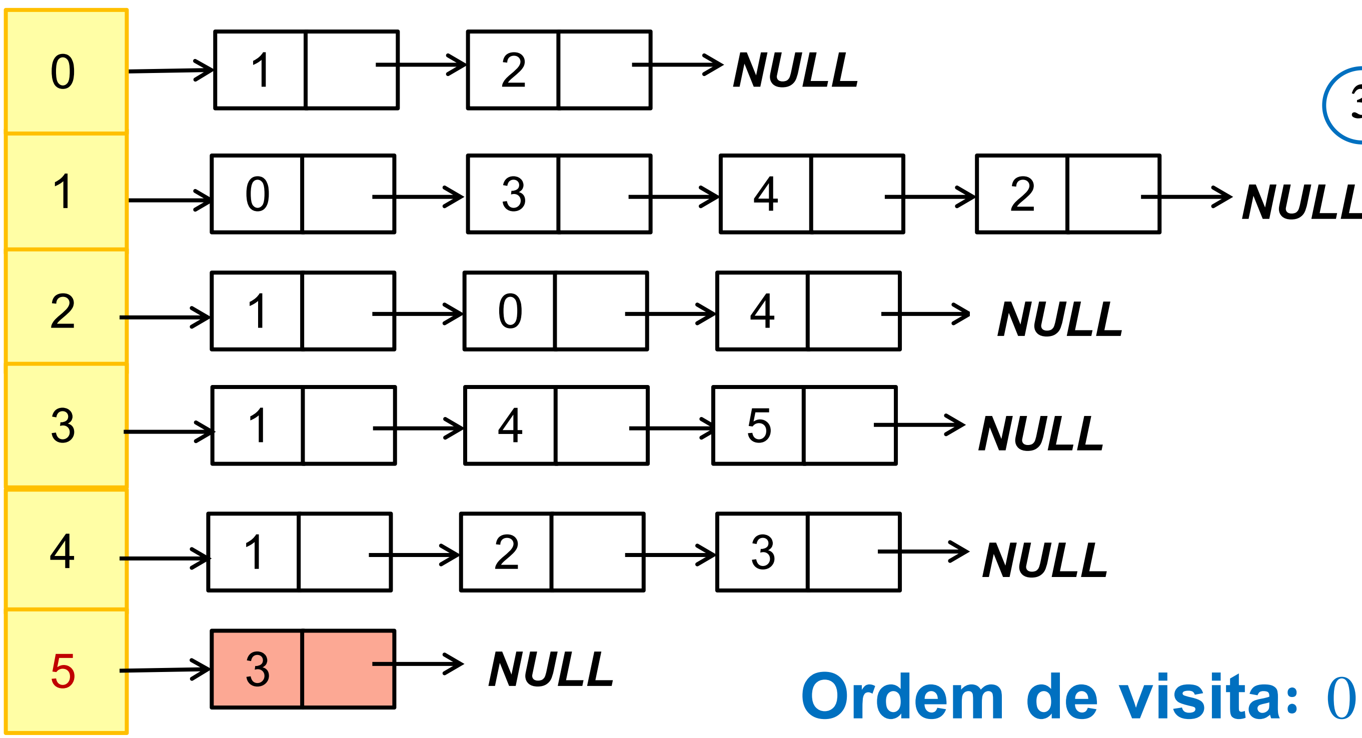
Ordem de visita: 0 , 1 , 2 , 3 , 4

Busca em largura: exemplo



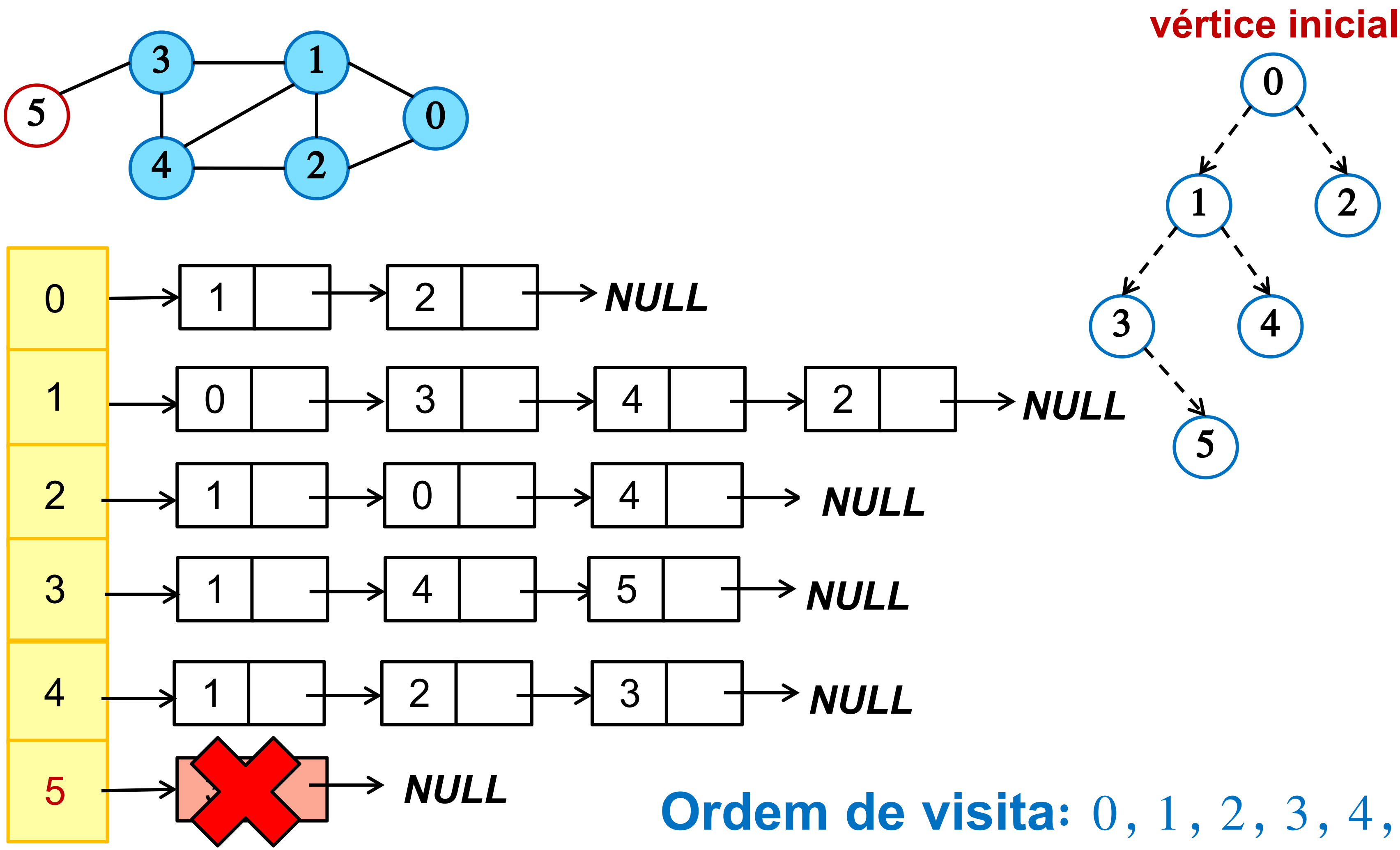
vértice inicial





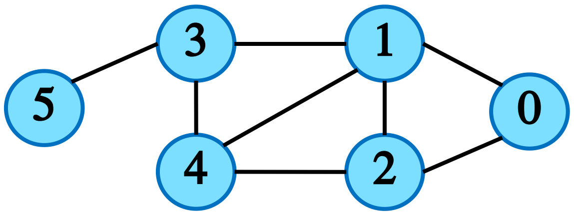
Ordem de visita: 0, 1, 2, 3, 4, 5

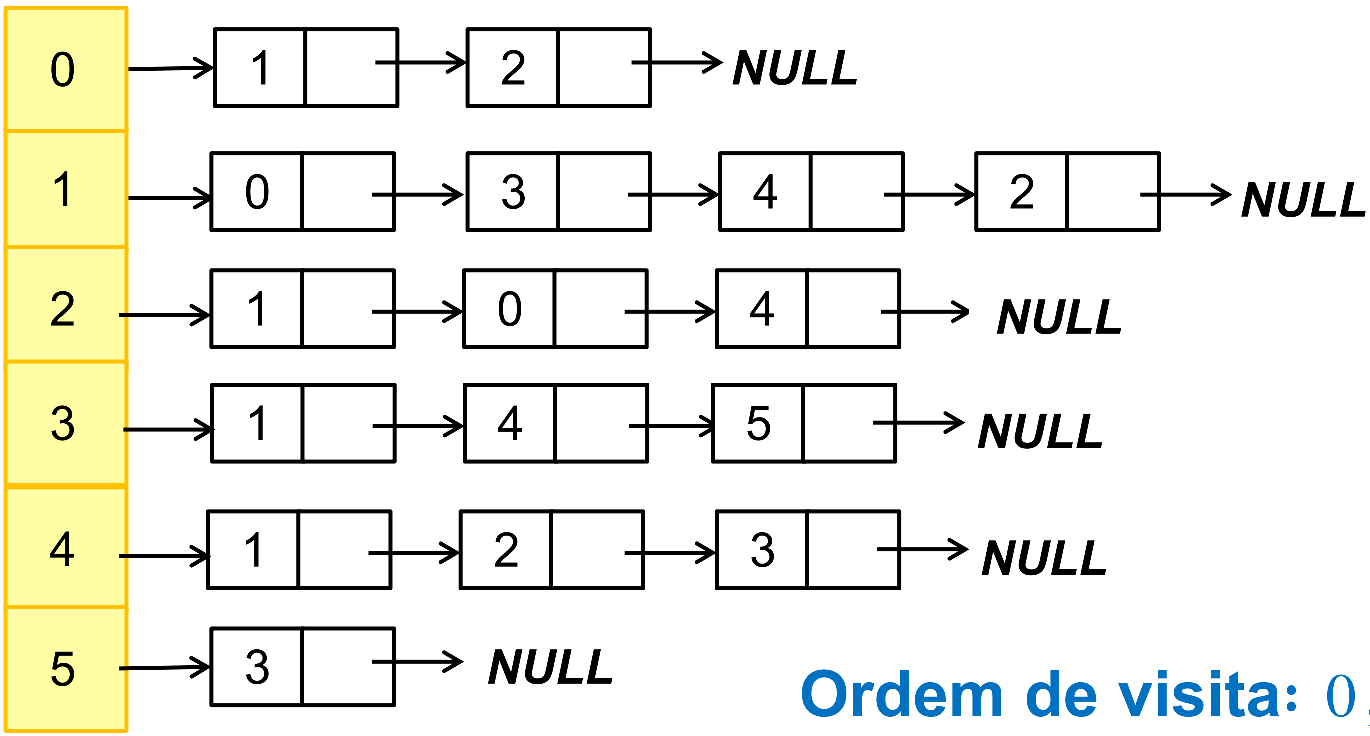
Busca em largura: exemplo

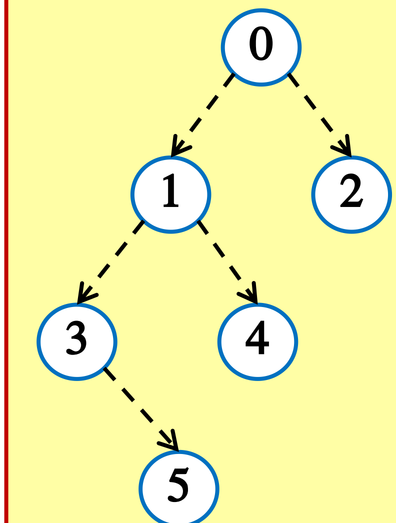


Ordem de visita: 0, 1, 2, 3, 4, 5

Busca em largura: exemplo







árvore de busca

Ordem de visita: 0, 1, 2, 3, 4, 5

Busca em largura: algoritmo

Precisa **armazenar os vértices já visitados**

- Vetor de índice ou lista encadeada

- Variável global ou argumento da função

Busca em largura: algoritmo

Precisa **armazenar os vértices já visitados**

Vetor de índice ou lista encadeada

Variável global ou argumento da função

Utiliza uma **estrutura fila** para guardar os **vértices a visitar** futuramente

Busca em largura: algoritmo

busca_largura *(Grafo *G, int V)*

●●●

Busca em largura: algoritmo

busca_largura *(Grafo *G, int V)*

// Inicializa as estruturas auxiliares

*Aloca o vetor **visitado** com V de inteiros e inicializa cada elemento do vetor com ZERO;*

*Cria uma **fila**;*

•••

Busca em largura: algoritmo

busca_largura *(Grafo *G, int V)*

// Inicializa as estruturas auxiliares

*Aloca o vetor **visitado** com V de inteiros e inicializa cada elemento do vetor com ZERO;*

*Cria uma **fila**;*

// Trata o vértice atual

Marque V como visitado;

Execute a operação desejada em V (ex: imprimir);

Insere V no final da fila;

...

Busca em largura: algoritmo

...

// Percorre o grafo

ENQUANTO ***fila ≠ vazia*** *FAÇA*

FIM_ENQUANTO

FIM

Busca em largura: algoritmo

...

```
// Percorre o grafo
ENQUANTO fila ≠ vazia FAÇA
    Vet = remove inicio da fila;



FIM_ENQUANTO
FIM
```

Busca em largura: algoritmo

...

// Percorre o grafo

*ENQUANTO **fila ≠ vazia** FAÇA*

Vet = remove inicio da fila;

PARA cada vértice Adj adjacente a Vet FAÇA

FIM_PARA

FIM_ENQUANTO

FIM

Busca em largura: algoritmo

...

```
  // Percorre o grafo
  ENQUANTO fila ≠ vazia FAÇA
      Vet = remove inicio da fila;
      PARA cada vértice Adj adjacente a Vet FAÇA
          SE adj não foi visitado ENTÃO



          FIM_SE
      FIM_PARA
  FIM_ENQUANTO
FIM
```

Busca em largura: algoritmo

...

```
// Percorre o grafo
ENQUANTO fila ≠ vazia FAÇA
    Vet = remove inicio da fila;
    PARA cada vértice Adj adjacente a Vet FAÇA
        SE adj não foi visitado ENTÃO
            Marque Adj como visitado;
            Execute a operação desejada em Adj;
            Insere Adj no final da fila;
        FIM_SE
    FIM_PARA
FIM_ENQUANTO
FIM
```

Busca em largura: algoritmo

...

```
// Percorre o grafo
ENQUANTO fila ≠ vazia FAÇA
    Vet = remove inicio da fila;
    PARA cada vértice Adj adjacente a Vet FAÇA
        SE adj não foi visitado ENTÃO
            Marque Adj como visitado;
            Execute a operação desejada em Adj;
            Insere Adj no final da fila;
        FIM_SE
    FIM_PARA
FIM_ENQUANTO
FIM
```

Explora somente vértices conectados (ideal para tratar grafos conexos)

Busca em largura: análise

A alocação e inicialização do vetor de visitados tem custo ***O(|V|)***

Busca em largura: análise

A alocação e inicialização do vetor de visitados tem custo ***O(|V|)***

O tratamento do vértice inicial tem custo ***O(1)***

- As operações de criação e inserção na fila têm custo constante

Busca em largura: análise

A alocação e inicialização do vetor de visitados tem custo ***O(|V|)***

O tratamento do vértice inicial tem custo ***O(1)***

- As operações de criação e inserção na fila têm custo constante

O custo associado ao percorrimento do grafo é ***O(|A|)***

- O laço externo (***ENQUANTO***) será executado para todos os vértices
- Para cada vértice, sua lista de adjacências será percorrida uma única vez (laço interno - comando ***PARA***)

Busca em largura: análise

A alocação e inicialização do vetor de visitados tem custo ***O(|V|)***

O tratamento do vértice inicial tem custo ***O(1)***

- As operações de criação e inserção na fila têm custo constante

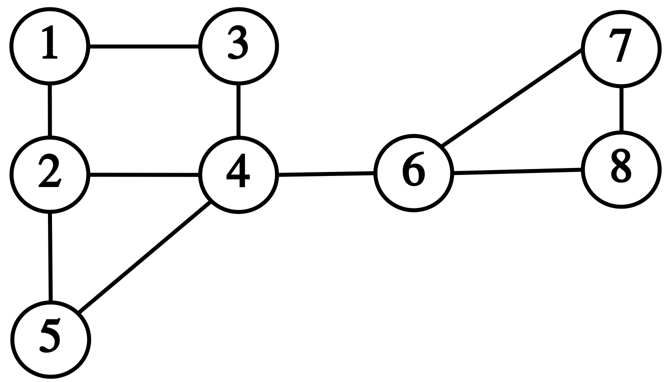
O custo associado ao percorrimento do grafo é ***O(|A|)***

- O laço externo (***ENQUANTO***) será executado para todos os vértices
- Para cada vértice, sua lista de adjacências será percorrida uma única vez (laço interno - comando ***PARA***)

O custo total do algoritmo é *O(|V| + |A|)*

Busca em largura: exercícios

1- Faça o teste de mesa do algoritmo de busca em largura para o grafo abaixo, iniciando pelo vértice 1:



2- Implemente o algoritmo de modo que o tratamento dos vértices seja mostrar o vetor de visitados, a fila, a lista de adjacentes do vértice atual e a ordem de visita da busca. Verifique se o resultado apresentado é igual ao do seu teste de mesa.

Busca em largura: exercícios

3- Altere a função de modo a explorar todos os vértices (**grafo não conexo**)

4- Implemente a **versão recursiva** da busca em largura. Neste caso, é necessário criar uma função de disparo, como na busca em largura.

5- Considerando que o grafo é representado através de uma matriz de adjacências, refaça a implementação iterativa e analise a sua complexidade.

Percorrimento de um grafo

Busca pelo menor caminho

Busca pelo menor caminho

Menor caminho (**caminho geodésico**) entre 2 vértices é aquele que apresenta o **menor comprimento** dentre todos

Busca pelo menor caminho

- **Menor caminho** (**caminho geodésico**) entre 2 vértices é aquele que apresenta o **menor comprimento** dentre todos
  - **Grafo não ponderado:** comprimento é o **número de arestas** que conectam os 2 vértices

Busca pelo menor caminho

Menor caminho (**caminho geodésico**) entre 2 vértices é aquele que apresenta o **menor comprimento** dentre todos

- **Grafo não ponderado:** comprimento é o **número de arestas** que conectam os 2 vértices
- **Grafo ponderado:** comprimento é a **soma dos pesos das arestas** que conectam os 2 vértices
  - **Peso** das arestas **não pode ser negativo**

Busca pelo menor caminho

Menor caminho (**caminho geodésico**) entre 2 vértices é aquele que apresenta o **menor comprimento** dentre todos

- **Grafo não ponderado:** comprimento é o **número de arestas** que conectam os 2 vértices
- **Grafo ponderado:** comprimento é a **soma dos pesos das arestas** que conectam os 2 vértices
  - **Peso** das arestas **não pode ser negativo**

Exemplos de aplicações:

- Definir o grau de relacionamento entre pessoas em uma rede social
- Determinar rotas em um mapa
- Definir estratégias de exploração através de máquinas autônomas
- Algoritmos de roteamento

Busca pelo menor caminho

- Uma forma de achar o menor caminho é através do **algoritmo de Dijkstra**
  - Calcula **o menor comprimento de um vértice a todos os outros**, desde que exista um caminho entre eles

Busca pelo menor caminho

Uma forma de achar o menor caminho é através do **algoritmo de Dijkstra**

- Calcula **o menor comprimento de um vértice a todos os outros**, desde que exista um caminho entre eles

Ideia básica:

- Baseado na **busca em largura**

Busca pelo menor caminho

Uma forma de achar o menor caminho é através do **algoritmo de Dijkstra**

- Calcula **o menor comprimento de um vértice a todos os outros**, desde que exista um caminho entre eles

Ideia básica:

- Baseado na **busca em largura**
- Próximo nó a ser explorado é escolhido de acordo com o **"potencial" de cada vértice adjacente**
  - Aquele com menor comprimento
  - Vértices não adjacentes tem **custo infinito**

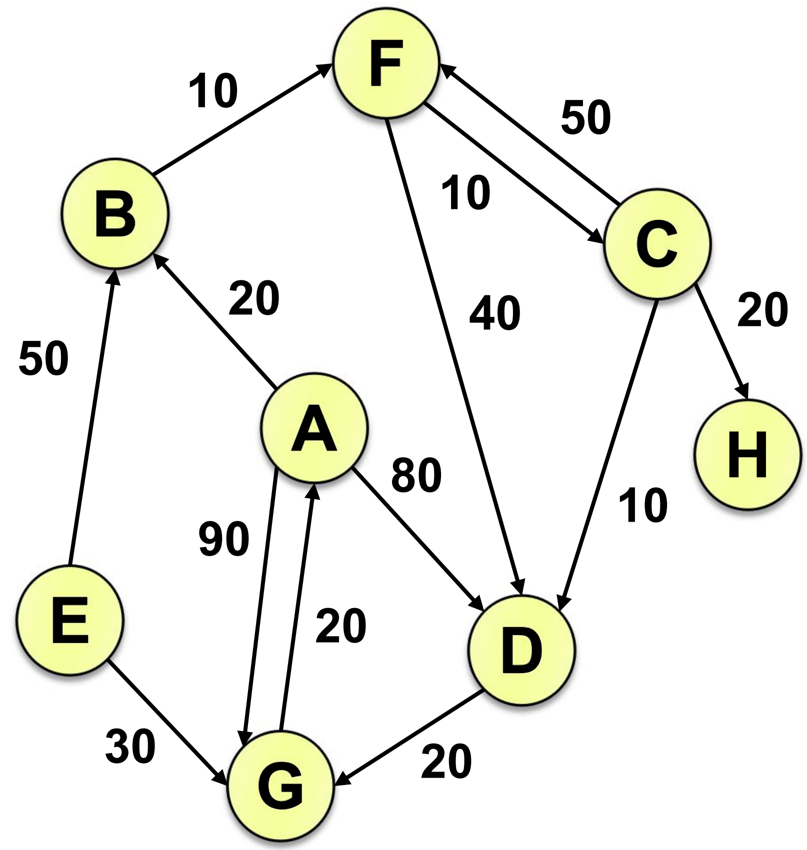
Busca pelo menor caminho

- Uma forma de achar o menor caminho é através do **algoritmo de Dijkstra**
  - Calcula **o menor comprimento de um vértice a todos os outros**, desde que exista um caminho entre eles

- **Ideia básica:**
  - Baseado na **busca em largura**
  - Próximo nó a ser explorado é escolhido de acordo com o **"potencial" de cada vértice adjacente**
    - Aquele com menor comprimento
    - Vértices não adjacentes tem **custo infinito**
- Processo se repete até **todo nó conexo ser explorado**

Algoritmo Dijkstra: Exemplo

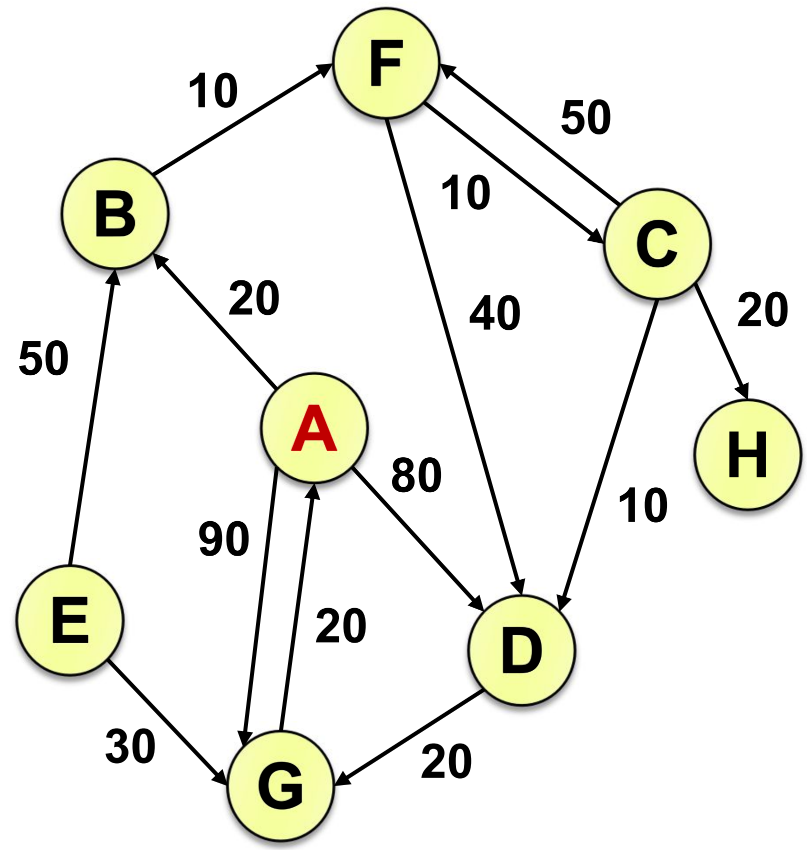
Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?



Algoritmo Dijkstra: Exemplo

Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?

De *A* podemos ir para:

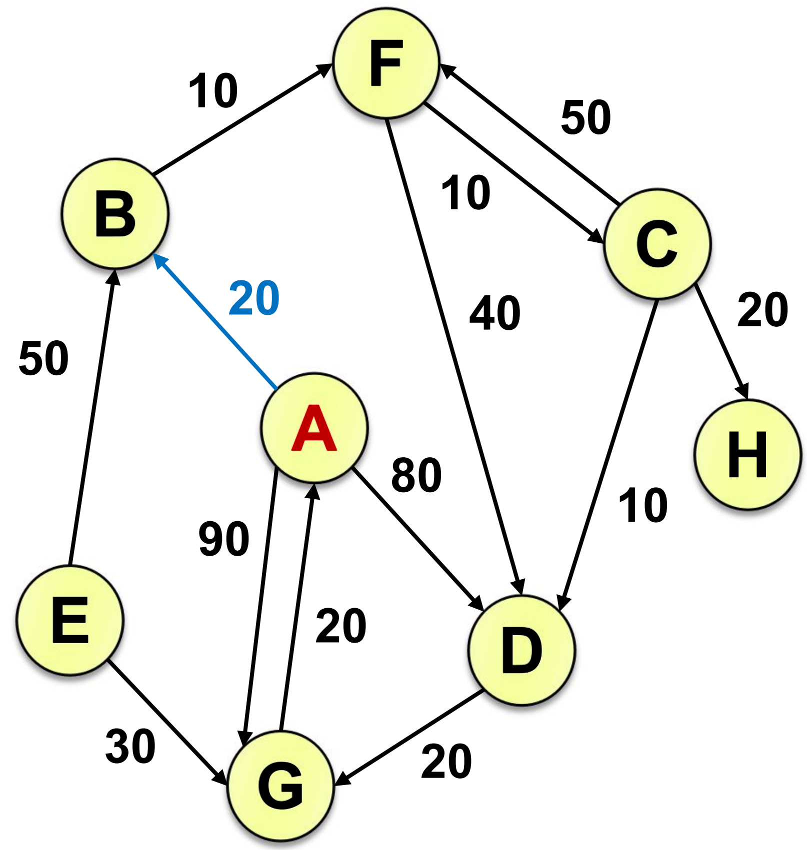


Algoritmo Dijkstra: Exemplo

Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?

De *A* podemos ir para:

***B* (custo = 20)**

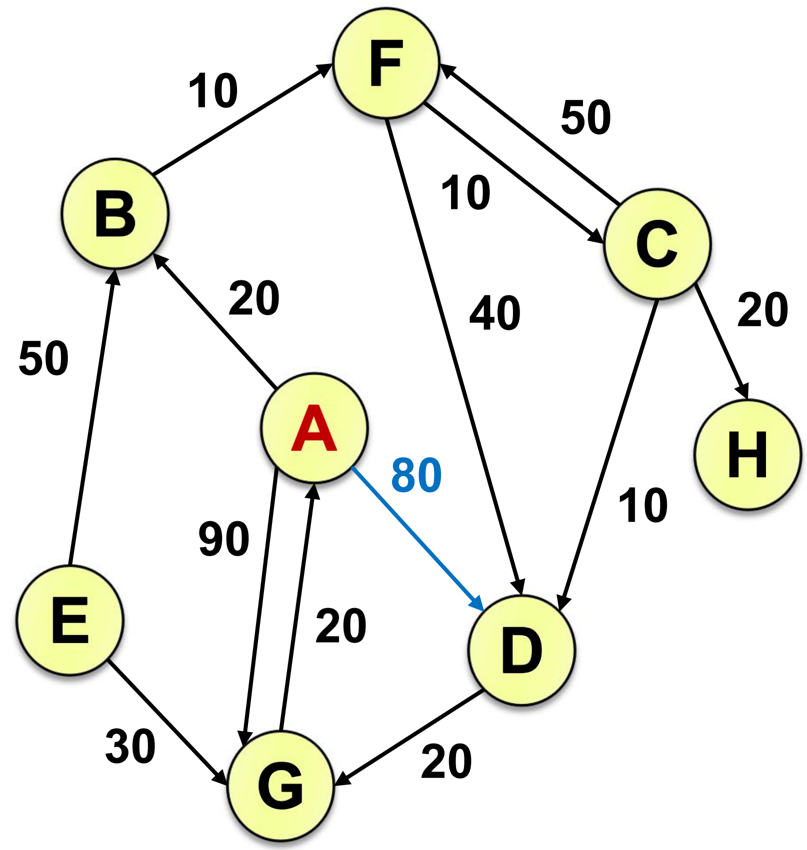


Algoritmo Dijkstra: Exemplo

Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?

De *A* podemos ir para:

- *B* (custo = 20)
- ***D* (custo = 80)**

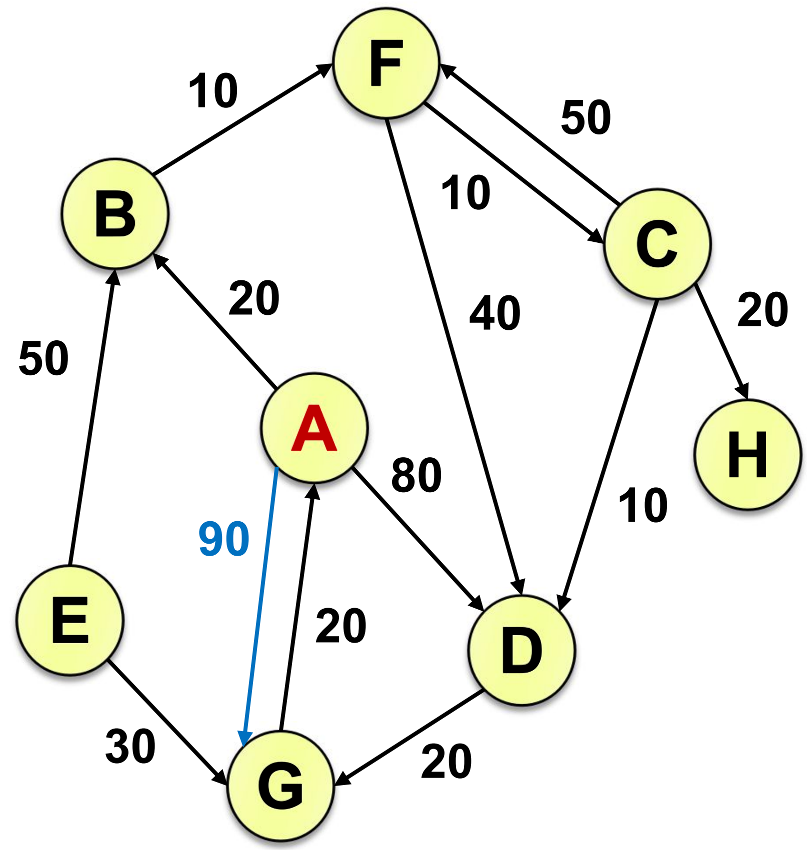


Algoritmo Dijkstra: Exemplo

Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?

De *A* podemos ir para:

- *B* (custo = 20)
- *D* (custo = 80)
- ***G* (custo = 90)**



Algoritmo Dijkstra: Exemplo

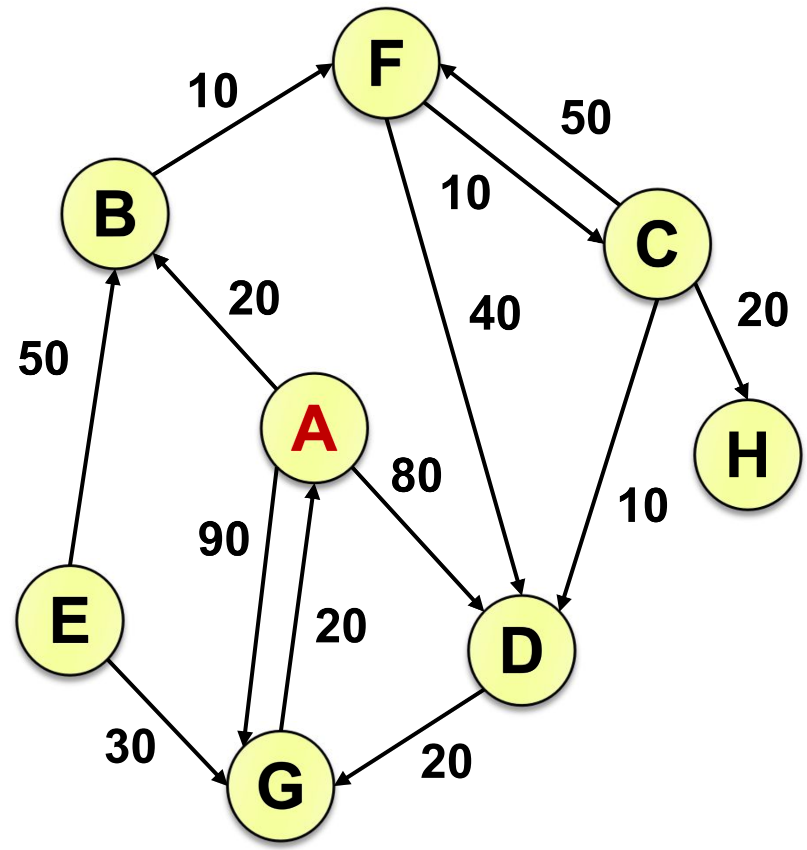
Dado o grafo abaixo, qual é o menor caminho de cada vértice a partir do vértice *A*?

De *A* podemos ir para:

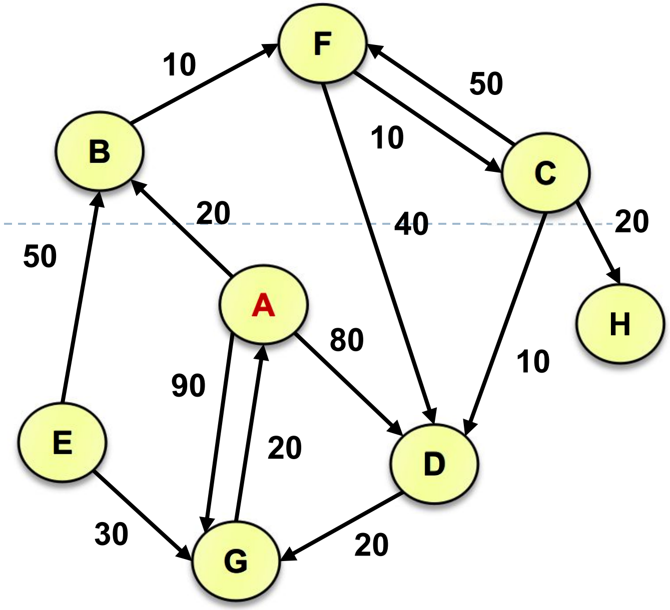
- *B* (custo = 20)
- *D* (custo = 80)
- *G* (custo = 90)

Demais vértices não podem ser alcançados

Custo = ∞ (infinito)



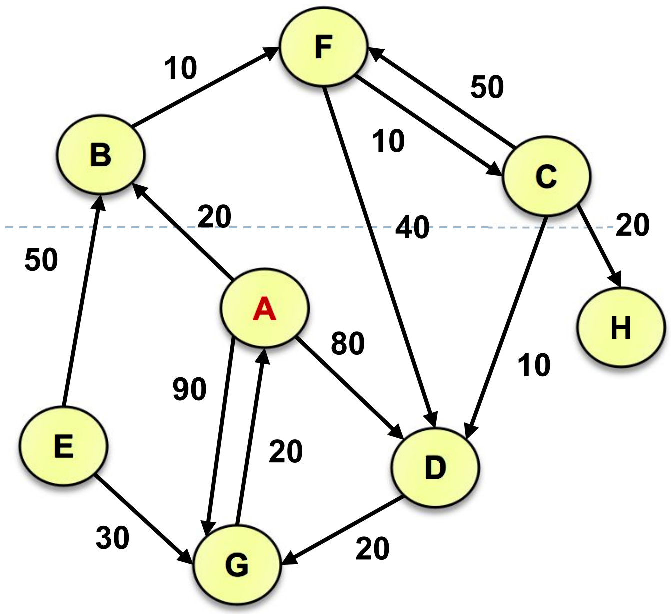
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	20	∞	80	∞	∞	90	∞

Algoritmo Dijkstra: Exemplo

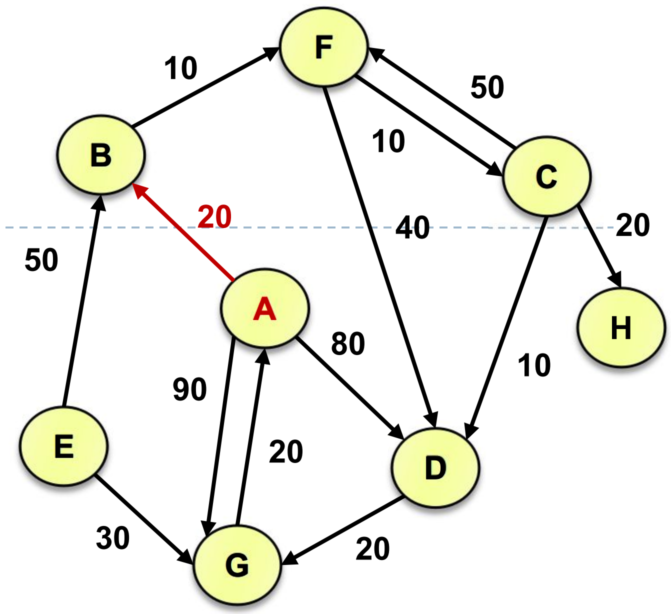


Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	20	∞	80	∞	∞	90	∞

Qual o menor custo?

Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞

Qual o menor custo?

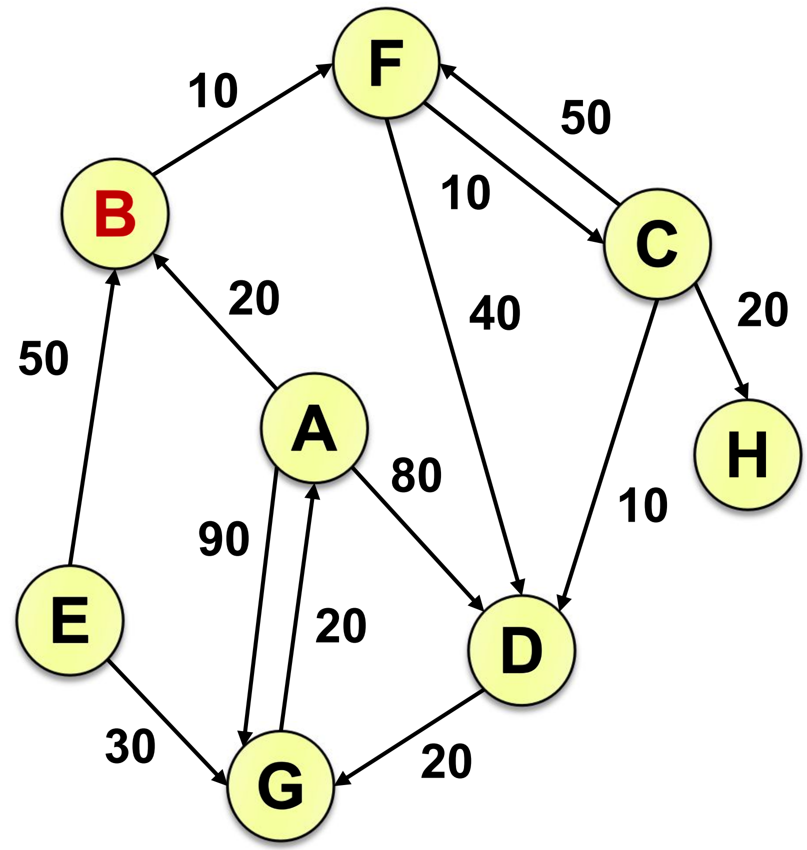
{A,B}* é o menor caminho de *A* até *B

(∄ outro caminho até *B* com menor custo)

Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B*, vamos analisar o menor caminho entre *B* e os outros vértices

De *B* podemos ir para:

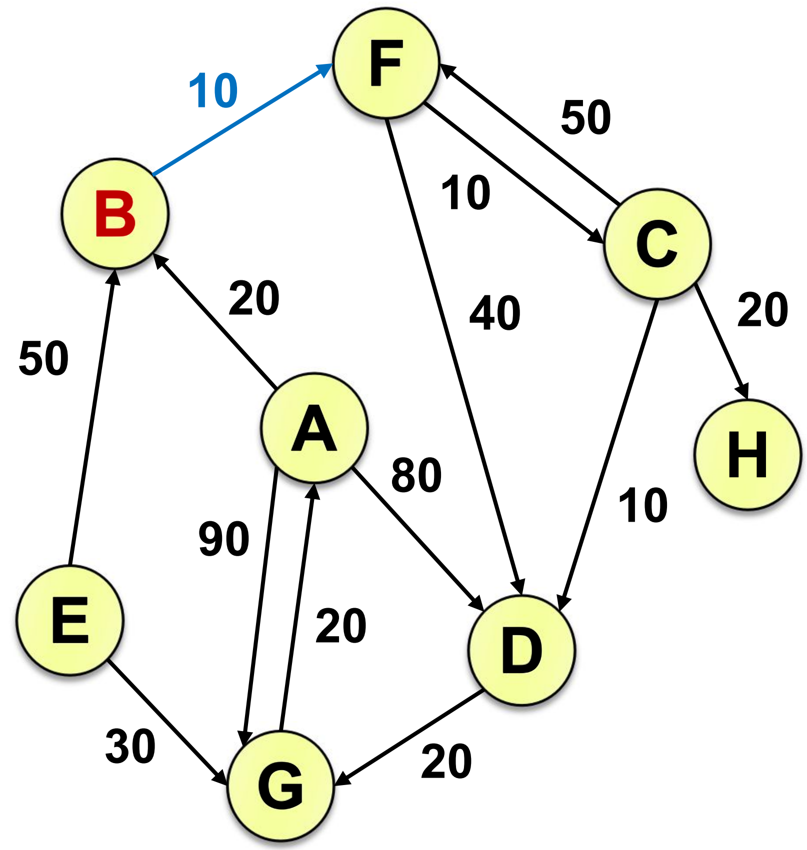


Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B*, vamos analisar o menor caminho entre *B* e os outros vértices

De *B* podemos ir para:

- ***F*** **(custo = 10)**



Algoritmo Dijkstra: Exemplo

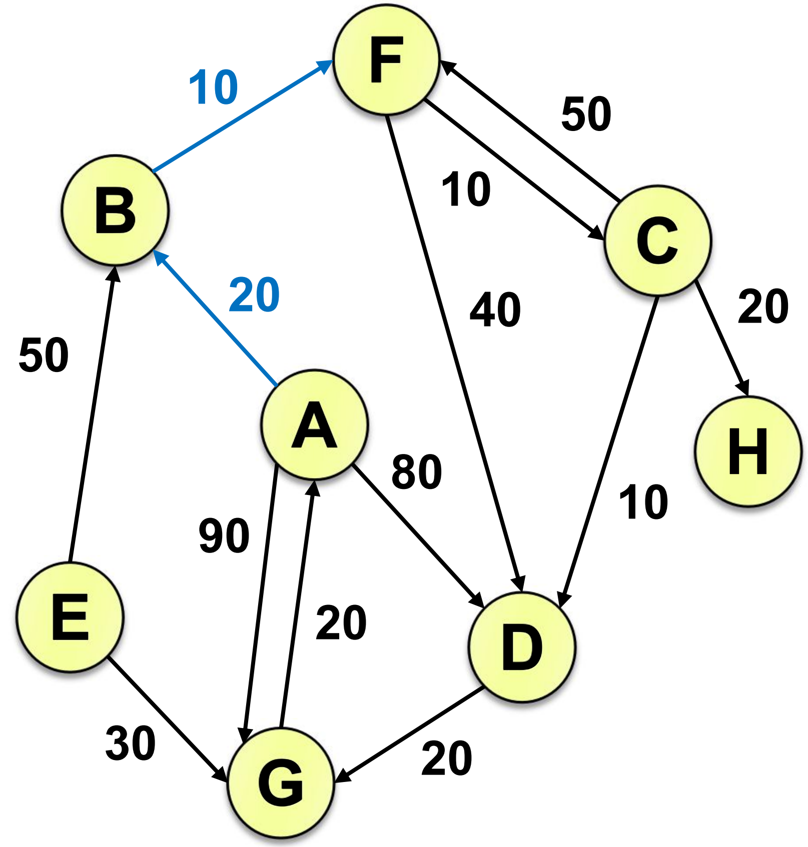
Como já sabemos que o menor caminho saindo de *A* é ir para *B*, vamos analisar o menor caminho entre *B* e os outros vértices

De *B* podemos ir para:

F (custo = 10)

Custo para chegar a *F* é somado ao custo até *B*

***custo(A,F)* = 20 + 10 = 30**



Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B*, vamos analisar o menor caminho entre *B* e os outros vértices

De *B* podemos ir para:

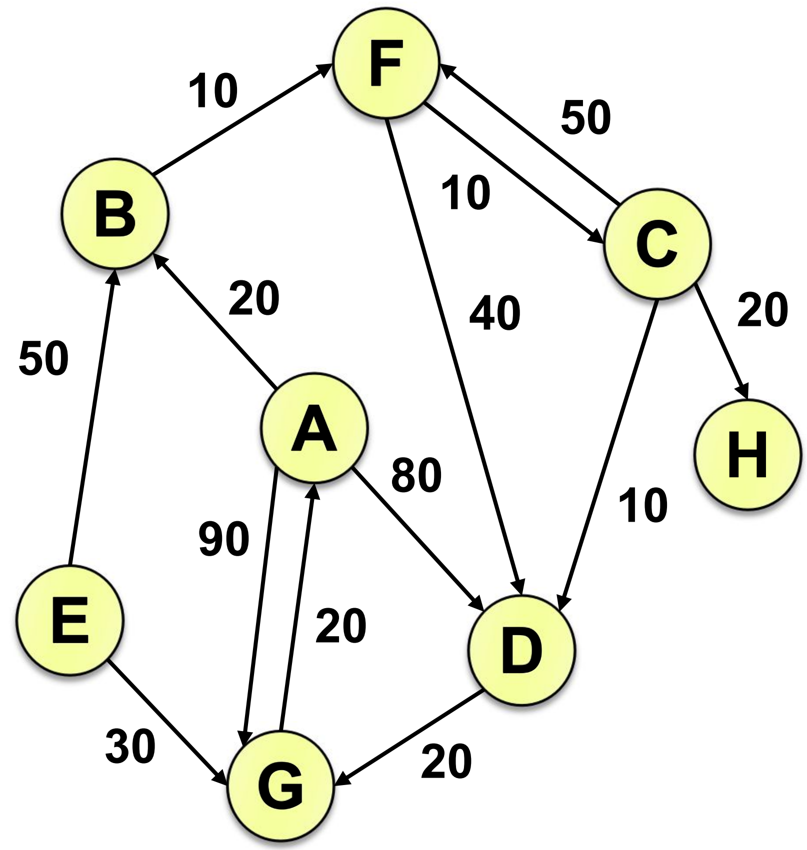
- *F* (custo = 10)

Custo para chegar a *F* é somado ao custo até *B*

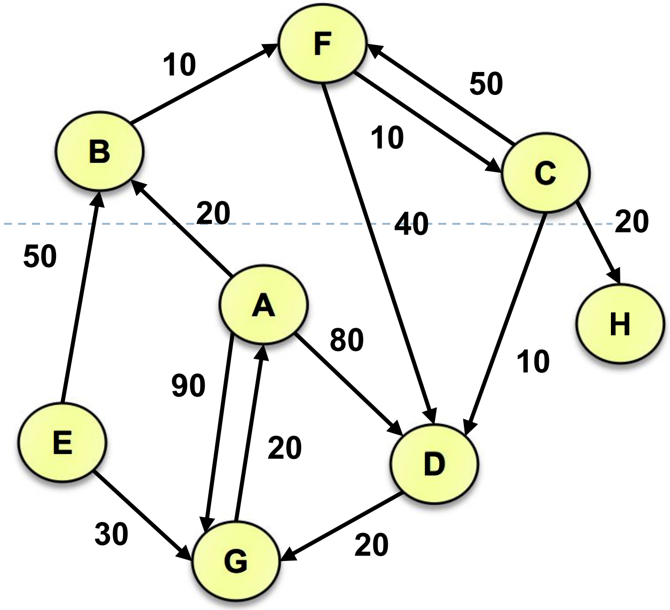
- *custo(A,F)* = 20 + 10 = 30

Demais vértices não podem ser alcançados

- **Custo atual é mantido**



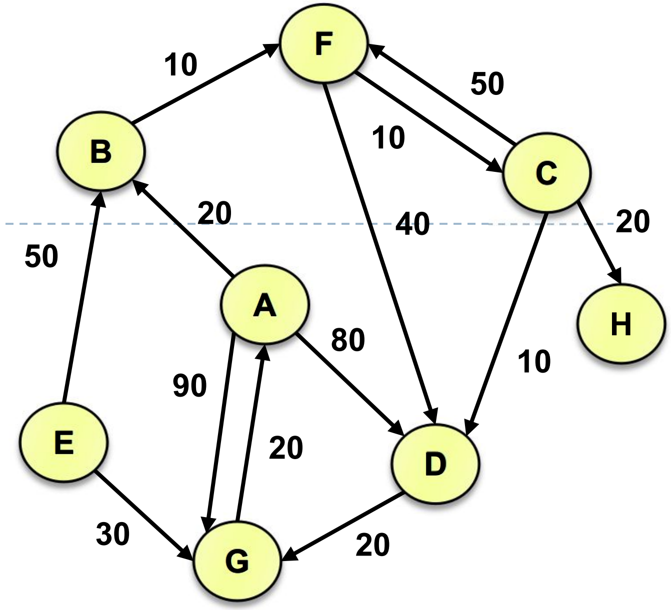
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	20	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	30	90	∞

Algoritmo Dijkstra: Exemplo

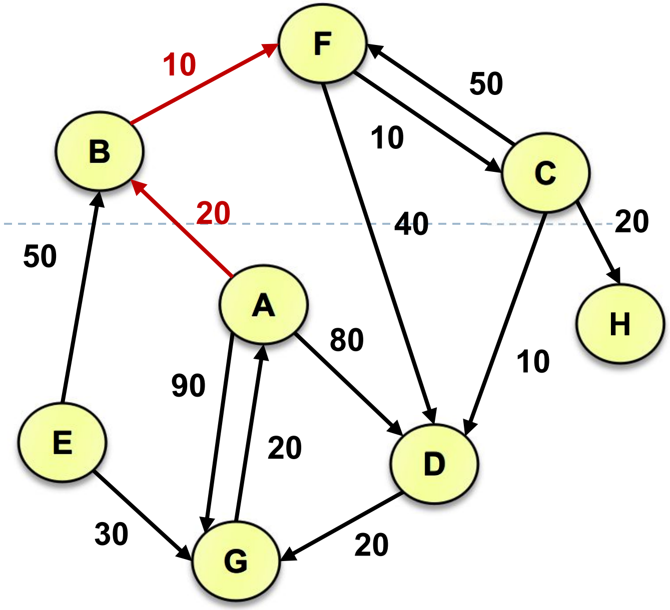


Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	20	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	30	90	∞

Qual o menor custo?

Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞

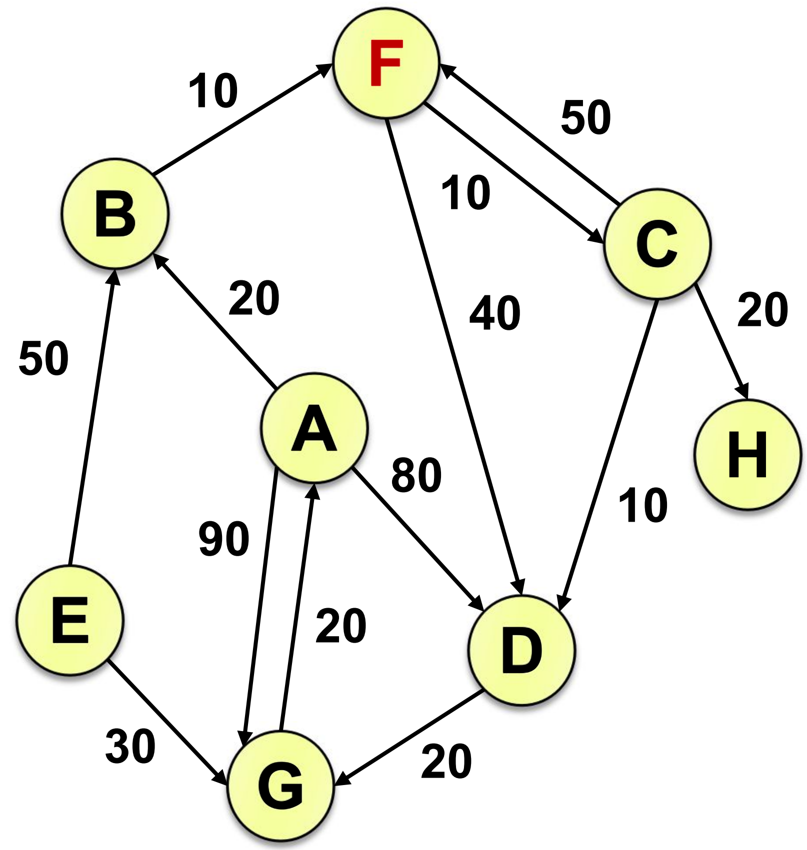
Qual o menor custo?

{A,B,F} **é o menor caminho de *A* até *F***

Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

De *F* podemos ir para:

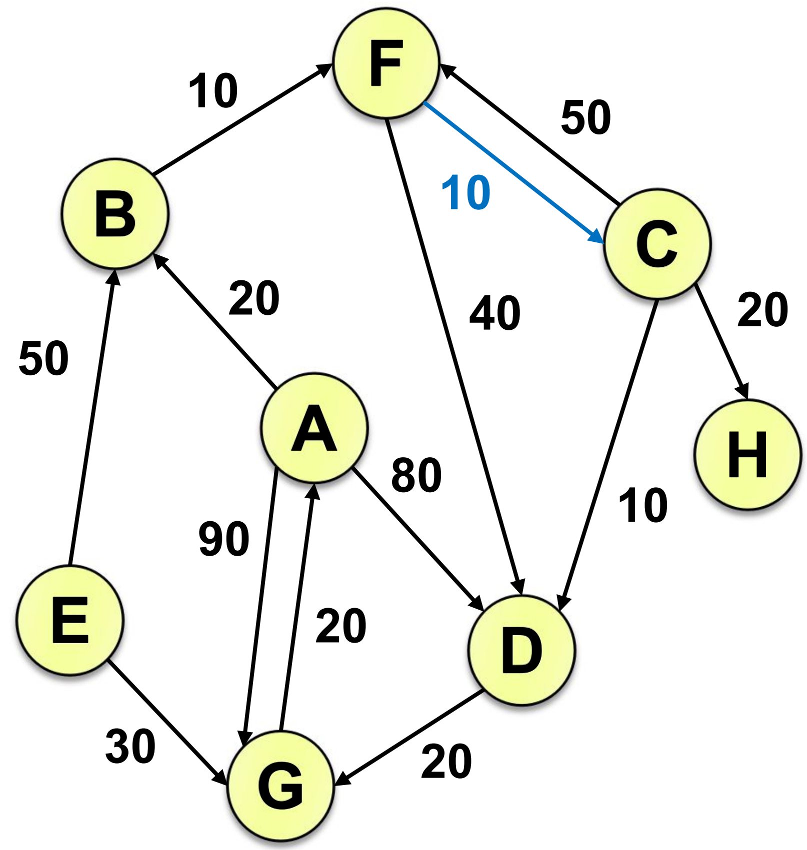


Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

De *F* podemos ir para:

C (custo = 10)



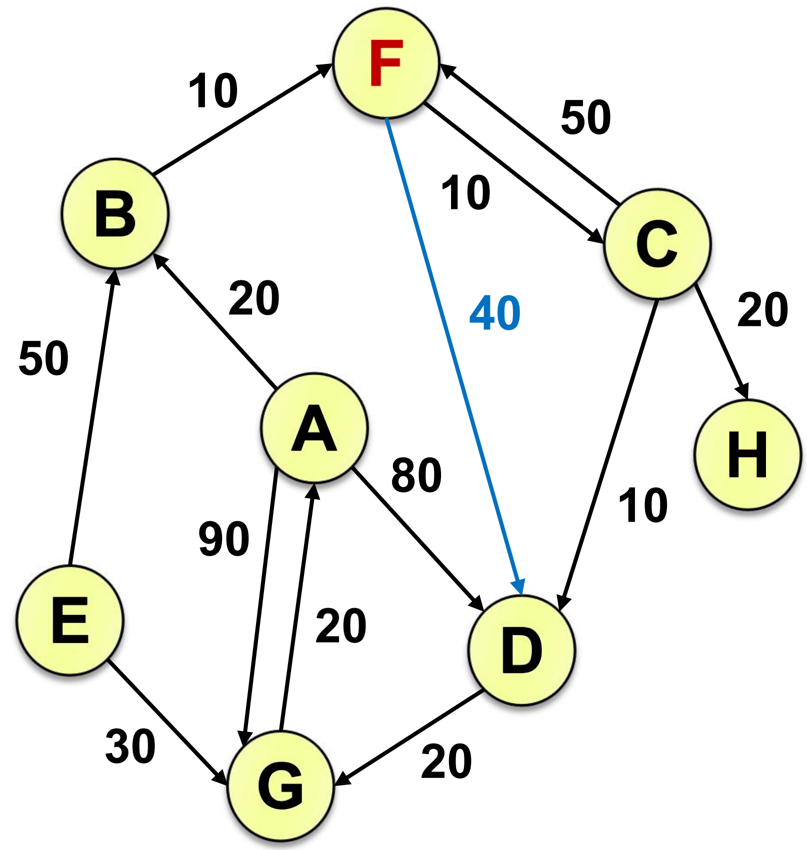
Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

De *F* podemos ir para:

C (custo = 10)

***D* (custo = 40)**



Algoritmo Dijkstra: Exemplo

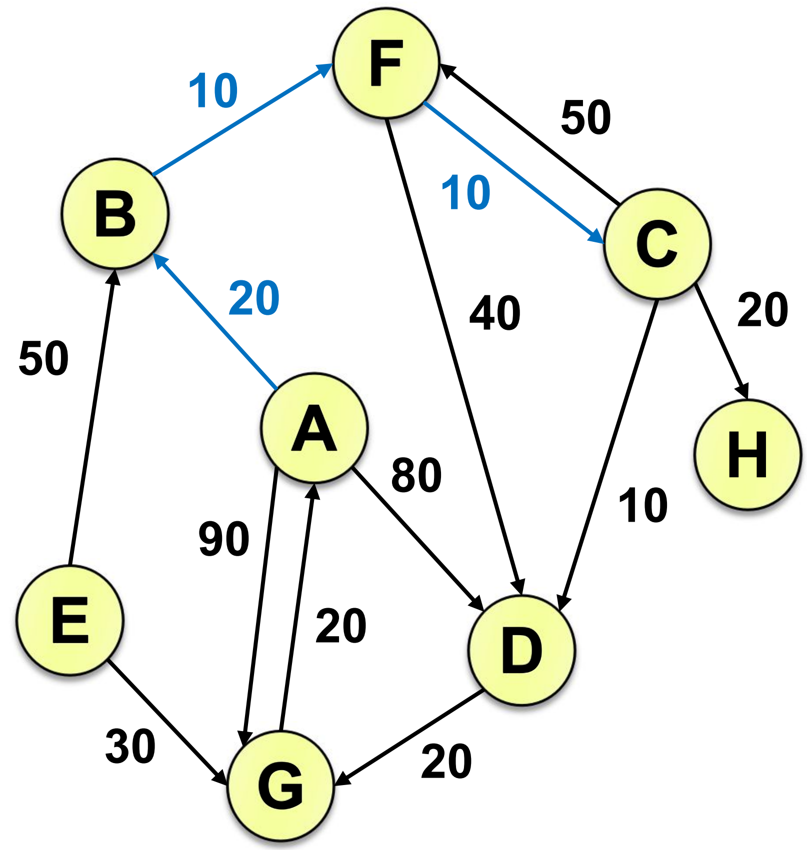
Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

De *F* podemos ir para:

- *C* (custo = 10)
- *D* (custo = 40)

Custo agregado:

- ***custo(A,C)* = 30 + 10 = 40**
- *custo (A,D)* = 30 + 40 = 70

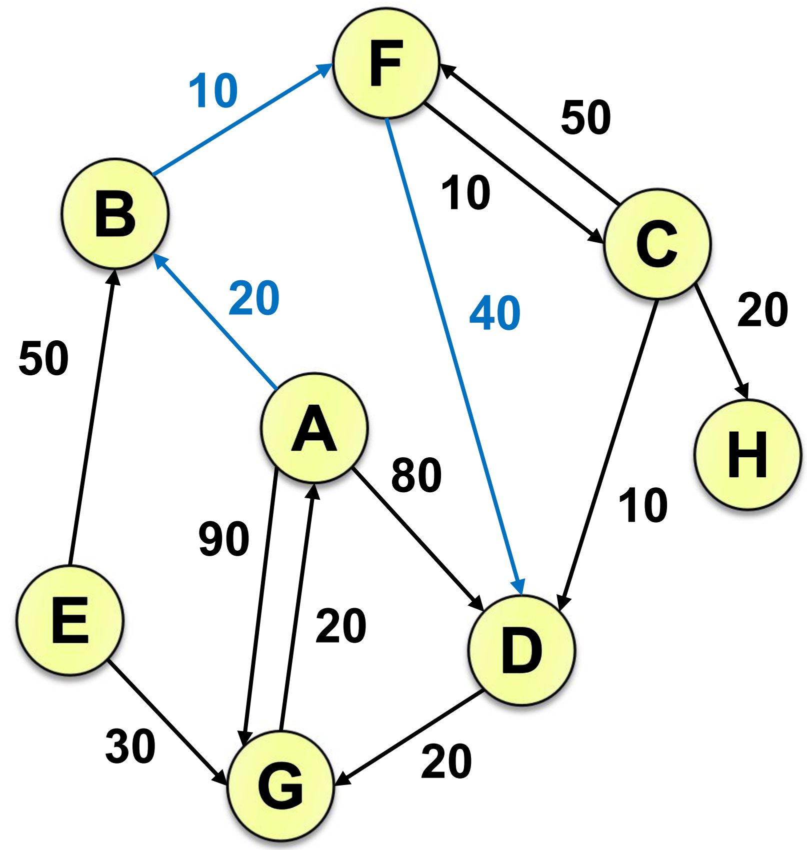


Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

- De *F* podemos ir para:
  - *C* (custo = 10)
  - *D* (custo = 40)

- Custo agregado:
  - *custo(A,C)* = 30 + 10 = 40
  - ***custo (A,D)* = 30 + 40 = 70**



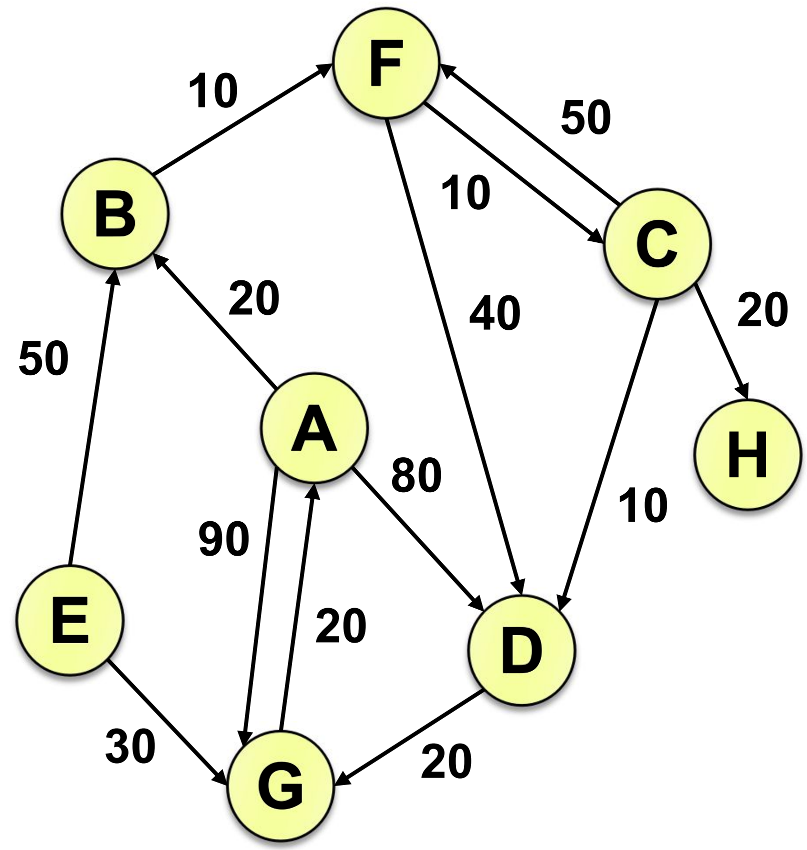
Algoritmo Dijkstra: Exemplo

Como já sabemos que o menor caminho saindo de *A* é ir para *B* e depois para *F*, vamos analisar o menor caminho entre *F* e os outros vértices:

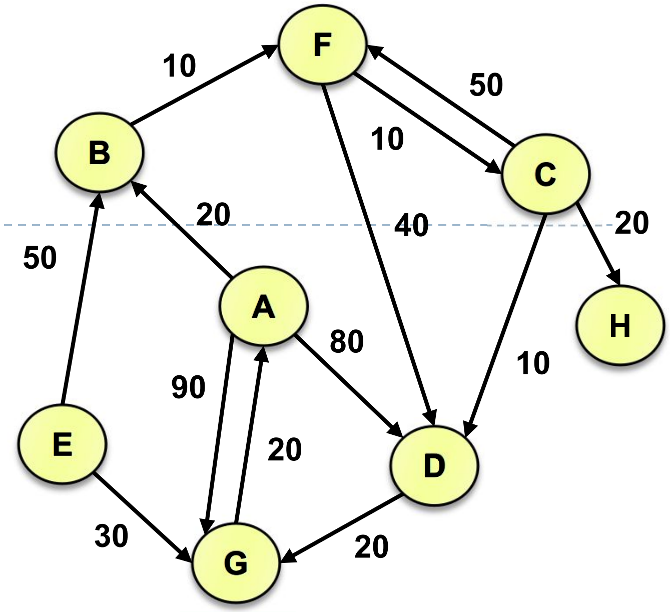
- De *F* podemos ir para:
  - *C* (custo = 10)
  - *D* (custo = 40)

- Custo agregado:
  - *custo(A,C)* = 30 + 10 = 40
  - *custo (A,D)* = 30 + 40 = 70

- Demais vértices não podem ser alcançados
  - **Custo atual é mantido**



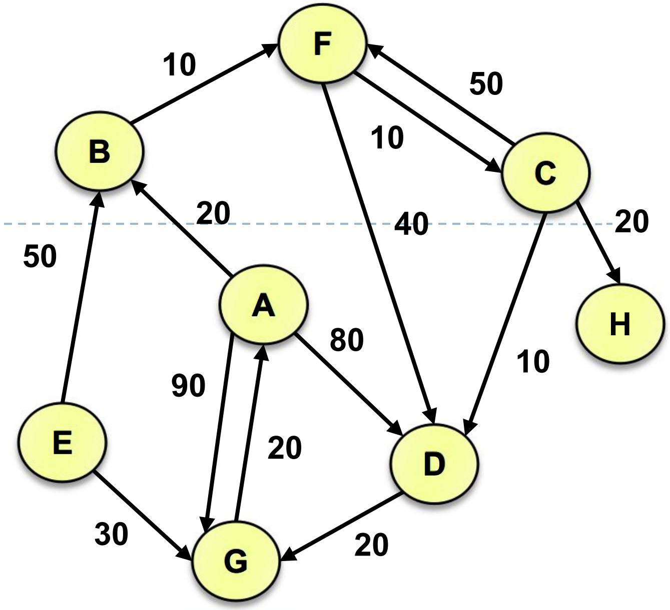
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		40	70	∞		90	∞

Algoritmo Dijkstra: Exemplo

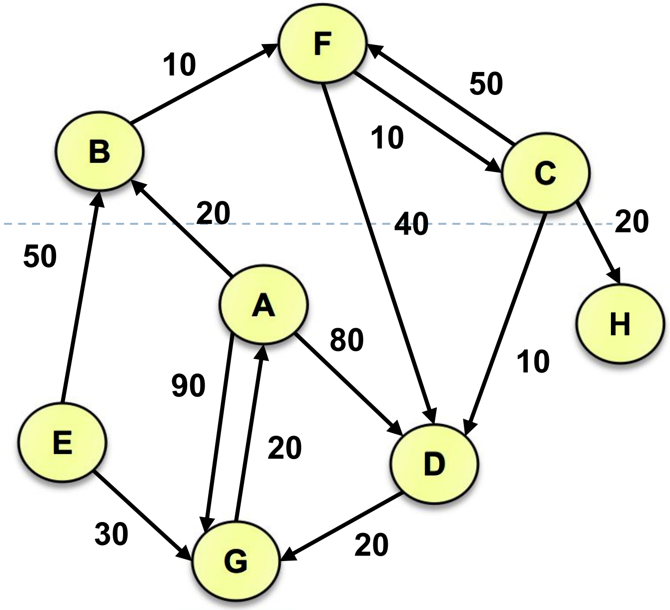


Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		40	**70**	∞		90	∞

custo de alcançar *D* passando por *B* e *F* (***70***) **é menor que** o custo de ir para D direto do *A* (***80***)

Algoritmo Dijkstra: Exemplo

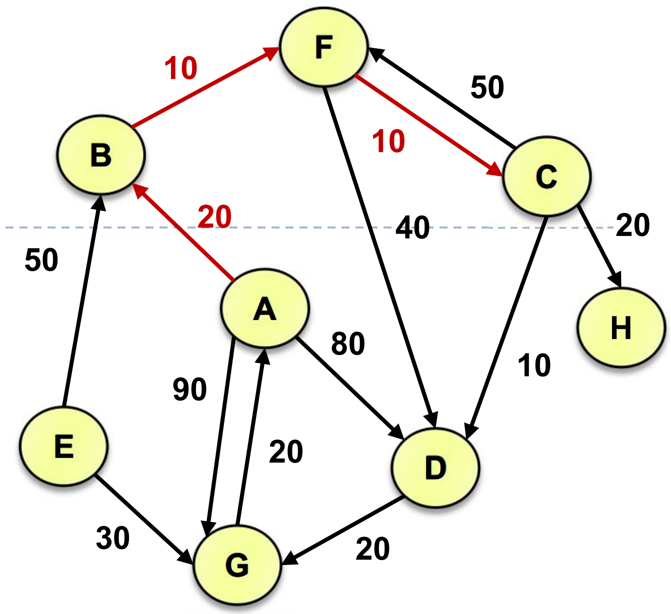


Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		40	70	∞		90	∞

Qual o menor custo?

Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞

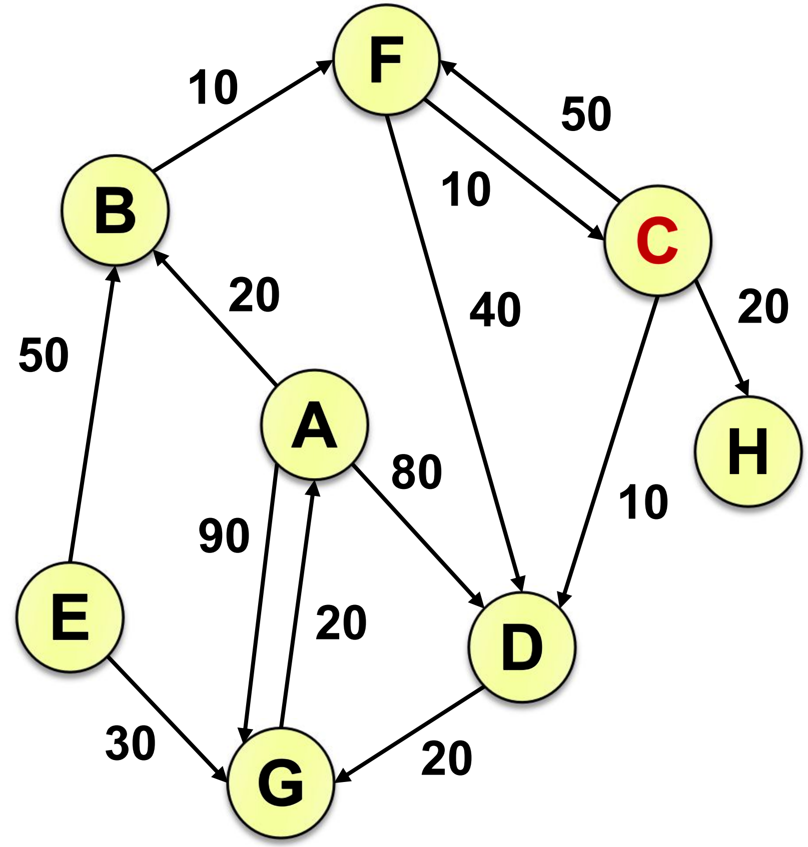
Qual o menor custo?

{A,B,F,C} **é o menor caminho de *A* até *C***

Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

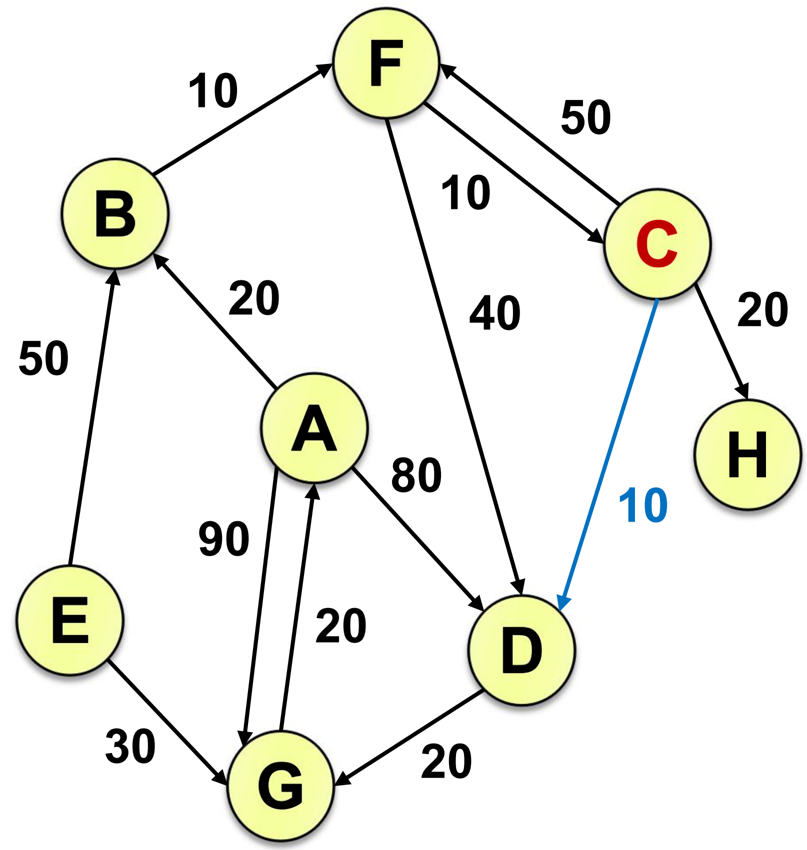


Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

_D_ (custo = 10)



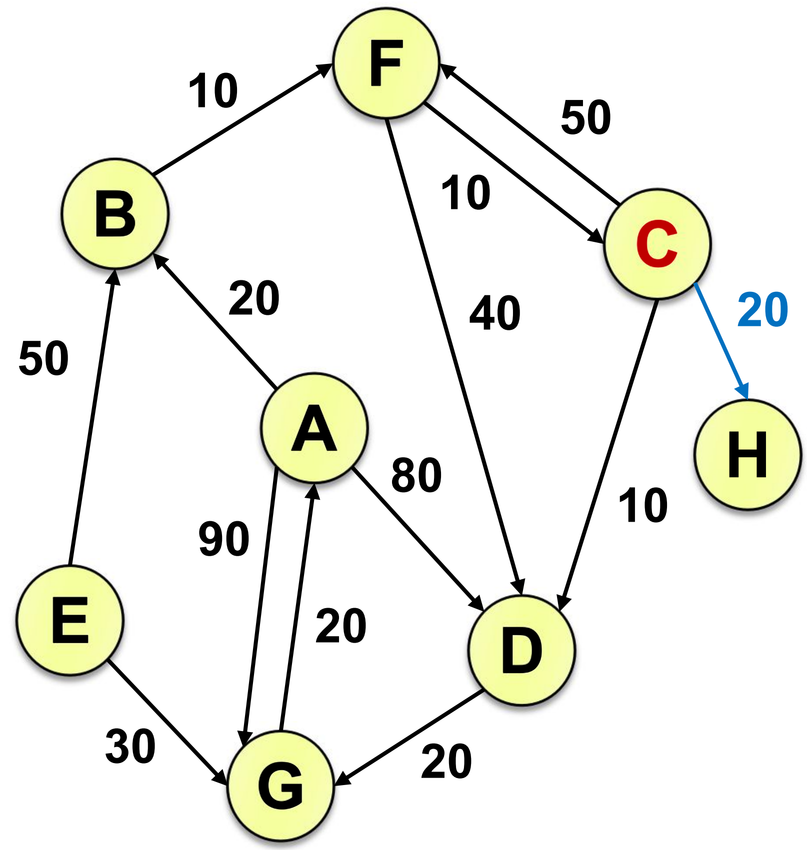
Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

D (custo = 10)

***H* (custo = 20)**

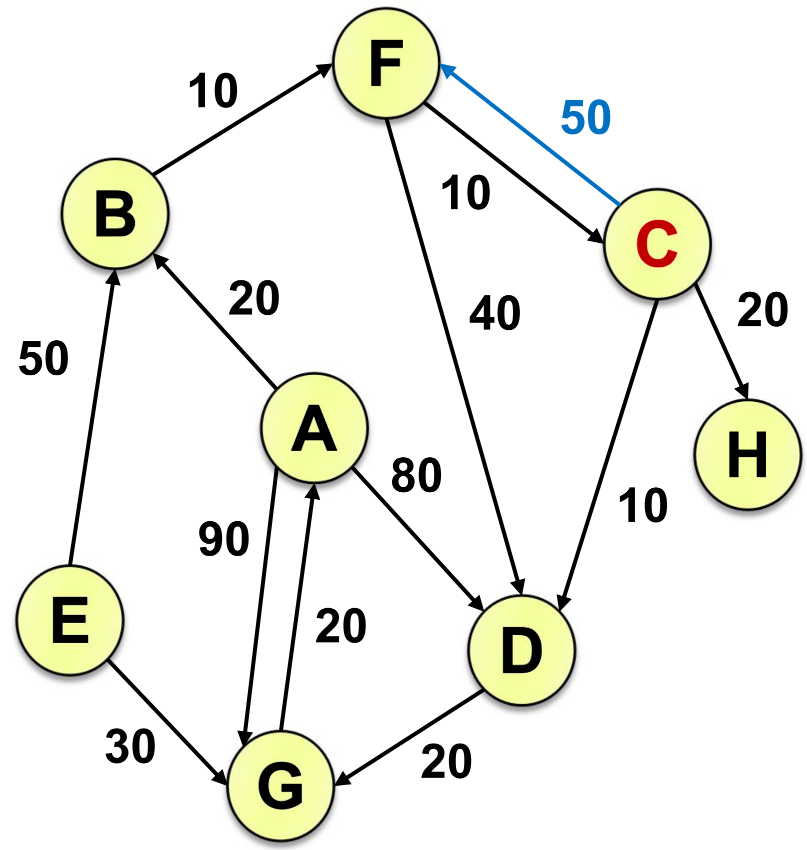


Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

- *D* (custo = 10)
- *H* (custo = 20)
- **F (custo = 50)**



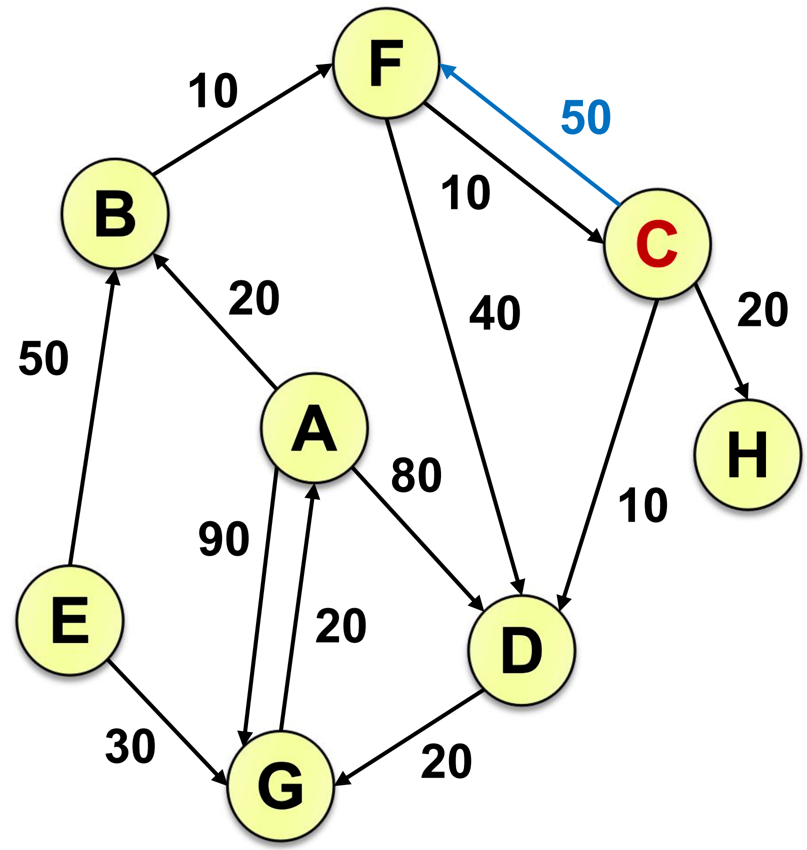
Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

- *D* (custo = 10)
- *H* (custo = 20)
- ~~**F (custo = 50)**~~

Menor caminho de *A* até *F* *já é conhecido*



Algoritmo Dijkstra: Exemplo

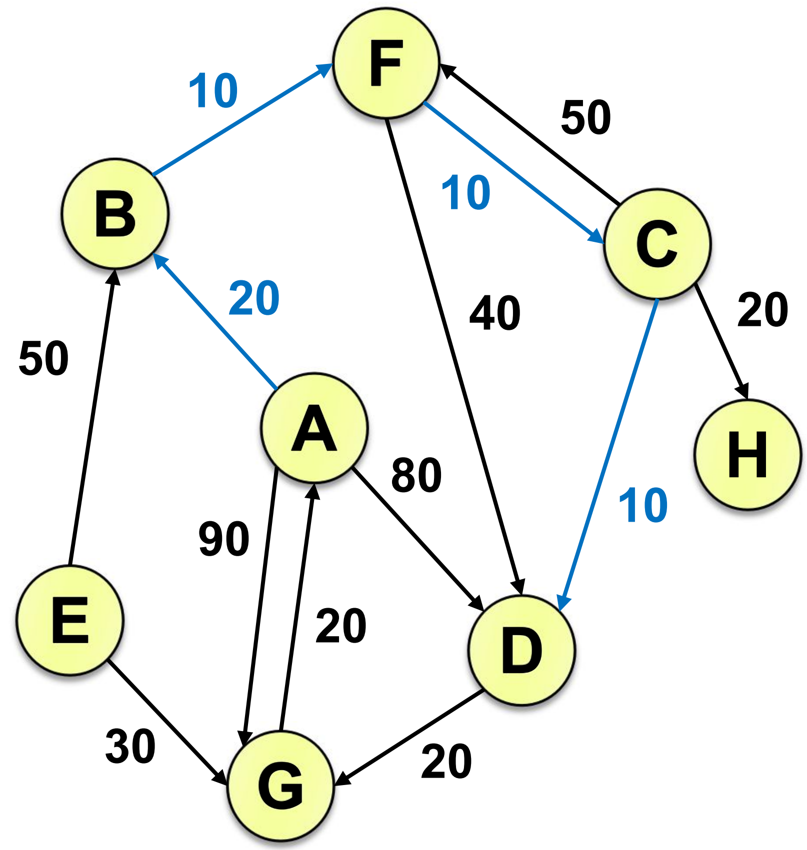
Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

- *D* (custo = 10)
- *H* (custo = 20)
- ~~*F* (custo = 50)~~

Custo agregado:

custo(A,D) = 40 + 10 = 50



Algoritmo Dijkstra: Exemplo

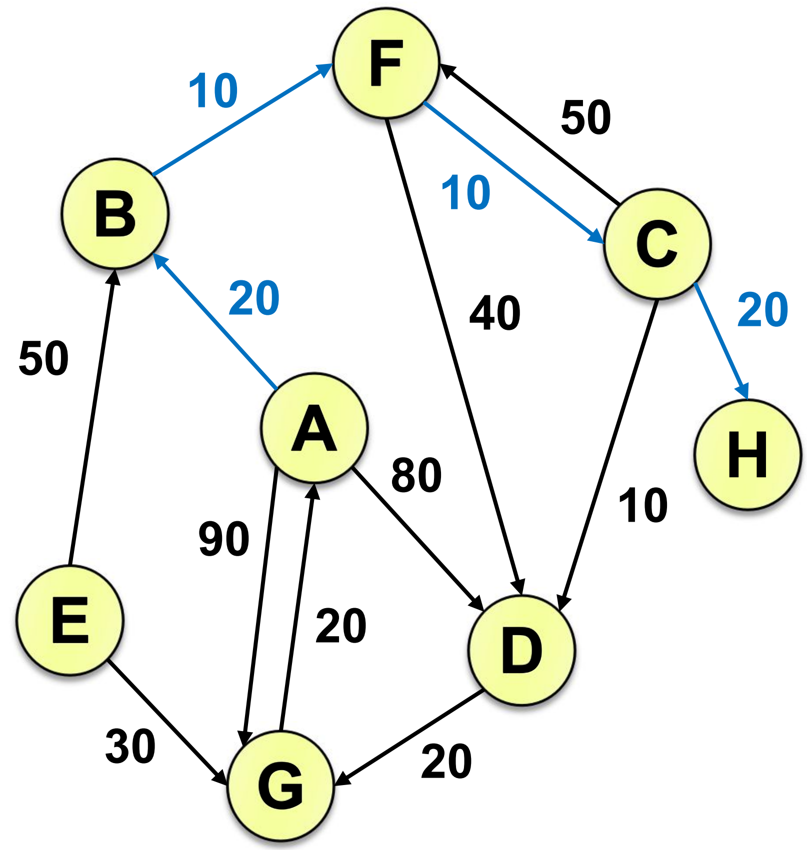
Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

- *D* (custo = 10)
- *H* (custo = 20)
- ~~*F* (custo = 50)~~

Custo agregado:

- *custo(A,D)* = 40 + 10 = 50
- ***custo (A,H)* = 40 + 20 = 60**



Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*}, vamos analisar o menor caminho entre *C* e os outros vértices:

De *C* podemos ir para:

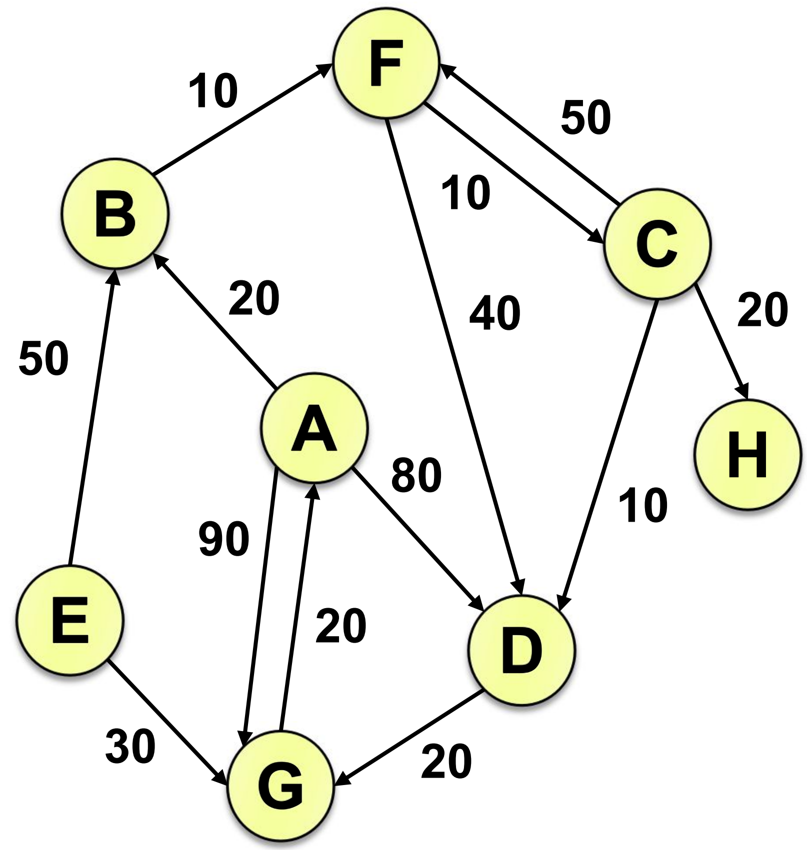
- *D* (custo = 10)
- *H* (custo = 20)
- ~~*F* (custo = 50)~~

Custo agregado:

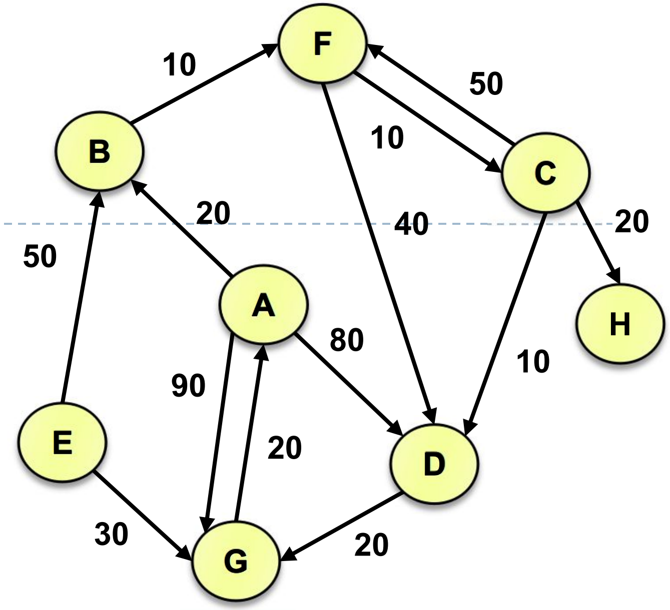
- *custo(A,D)* = 40 + 10 = 50
- *custo (A,H)* = 40 + 20 = 60

Demais vértices não podem ser alcançados

Custo atual é mantido



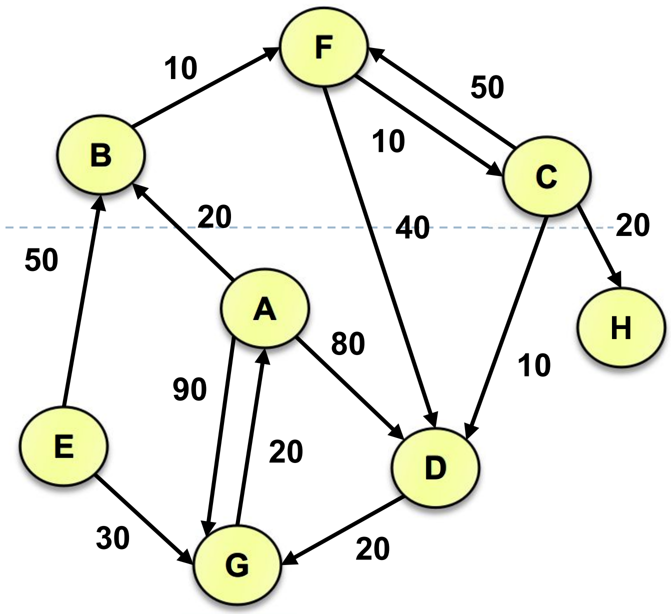
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			50	∞		90	60

Algoritmo Dijkstra: Exemplo

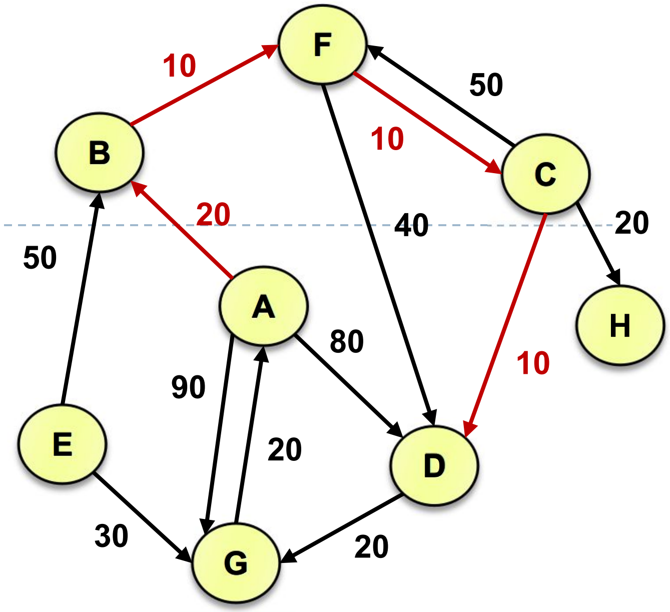


Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			50	∞		90	60

custo de alcançar *D* passando por *C* (***50***) **é menor que** o custo de ir para D a partir de *F* (***70***)

Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60

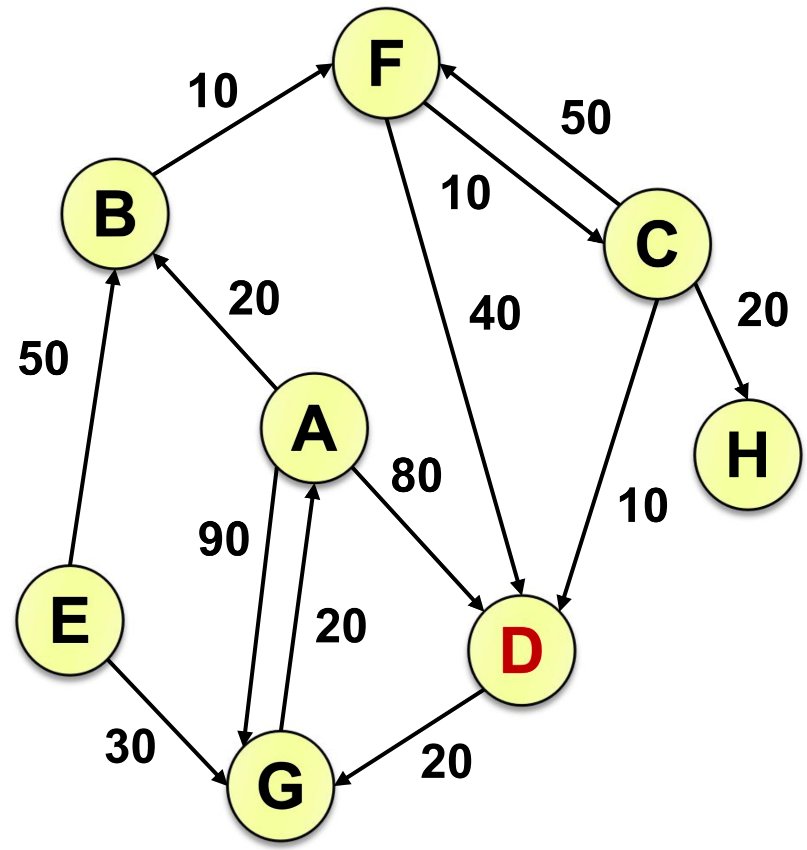
Qual o menor custo?

{A,B,F,C,D}* é o menor caminho de *A* até *D

Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*}, vamos analisar o menor caminho entre *D* e os outros vértices:

De *D* podemos ir para:

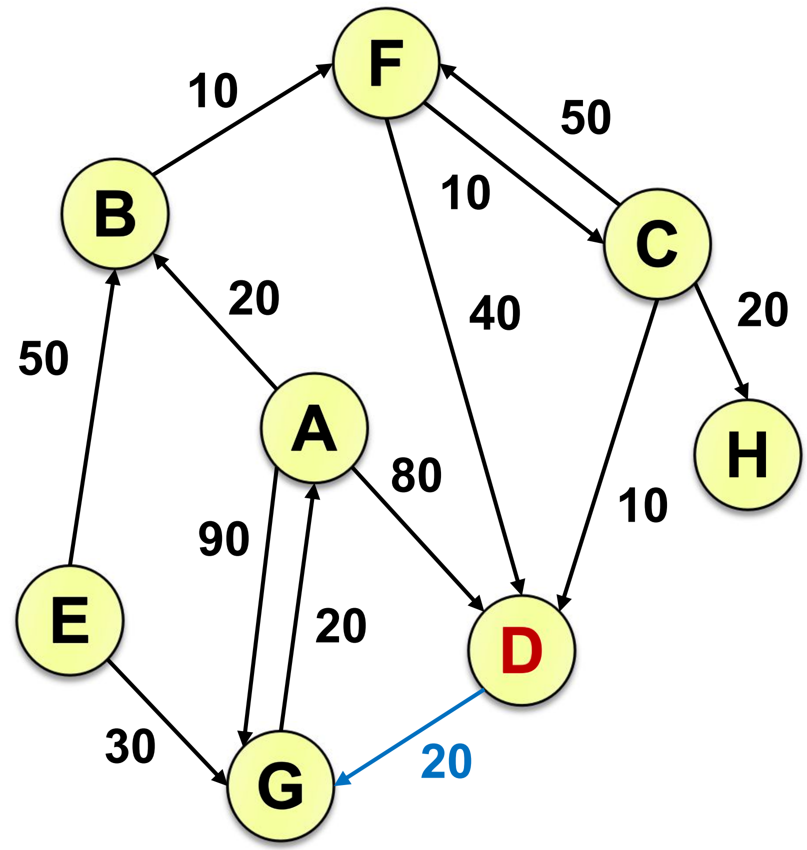


Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*}, vamos analisar o menor caminho entre *D* e os outros vértices:

De *D* podemos ir para:

G (custo = 20)



Algoritmo Dijkstra: Exemplo

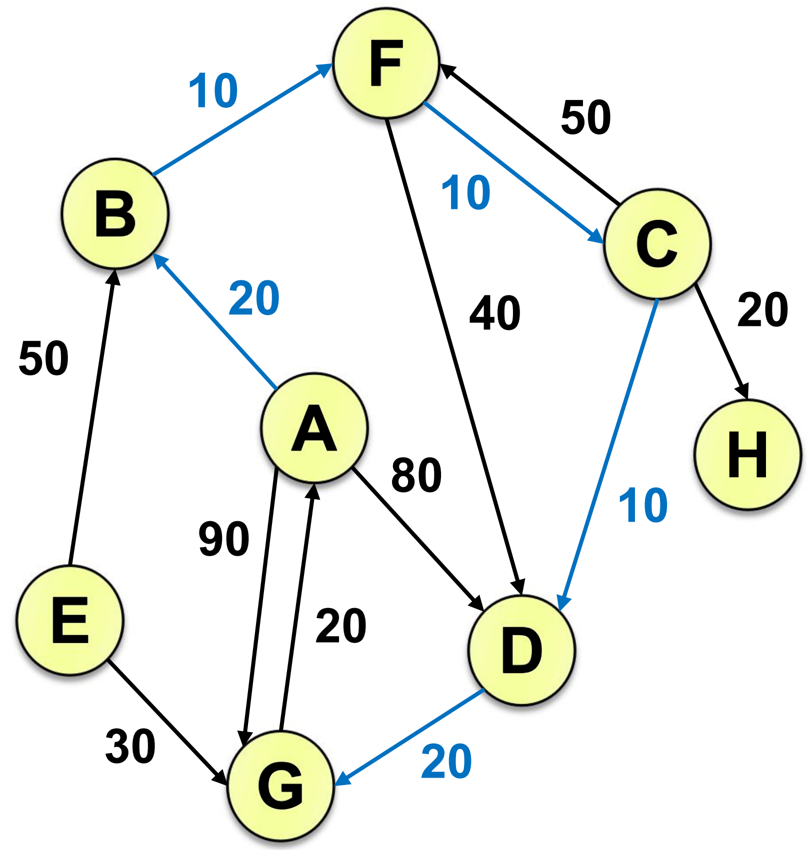
Considerando o caminho {*A*,*B*,*F*,*C*,*D*}, vamos analisar o menor caminho entre *D* e os outros vértices:

De *D* podemos ir para:

G (custo = 20)

Custo agregado:

***custo(A,G)* = 50 + 20 = 70**



Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*}, vamos analisar o menor caminho entre *D* e os outros vértices:

De *D* podemos ir para:

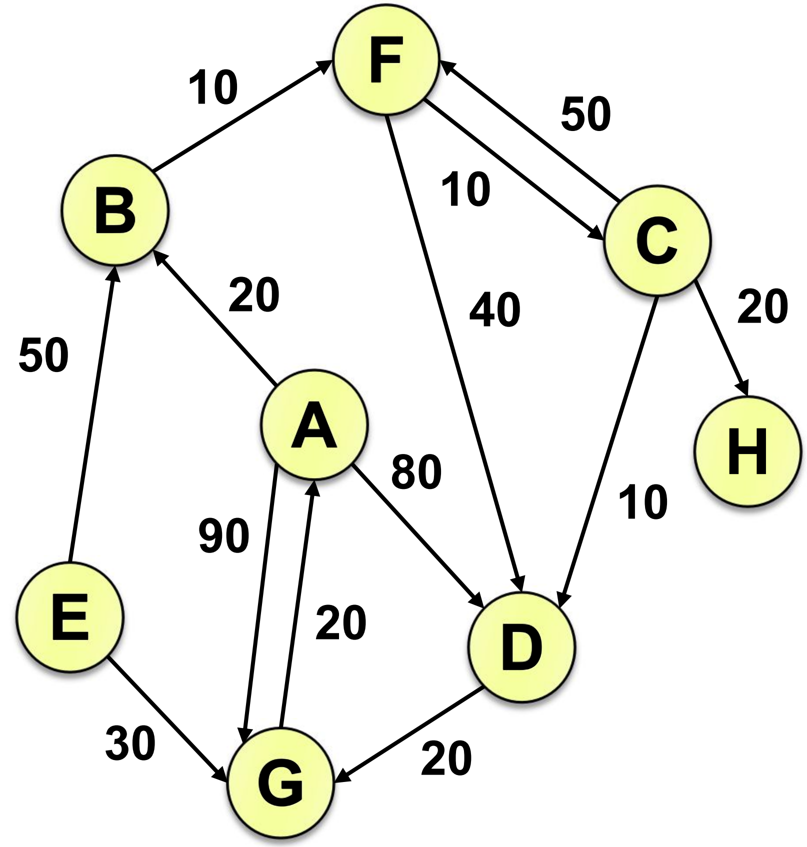
G (custo = 20)

Custo agregado:

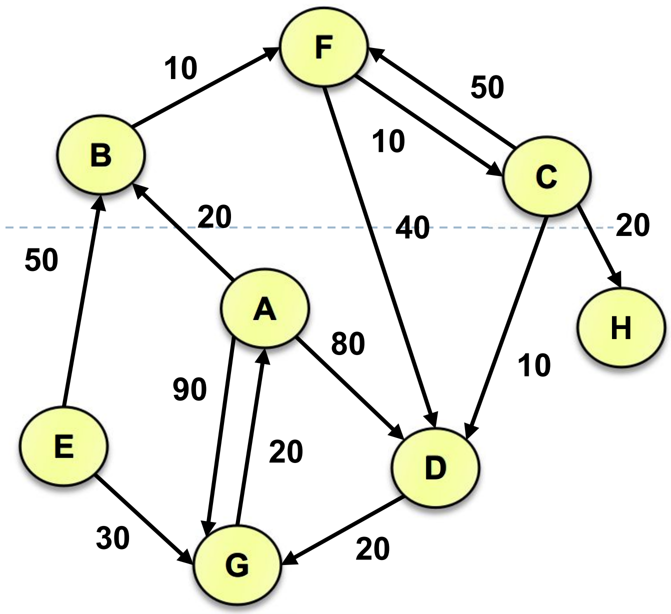
custo(A,G) = 50 + 20 = 70

Demais vértices não podem ser alcançados

Custo atual é mantido



Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

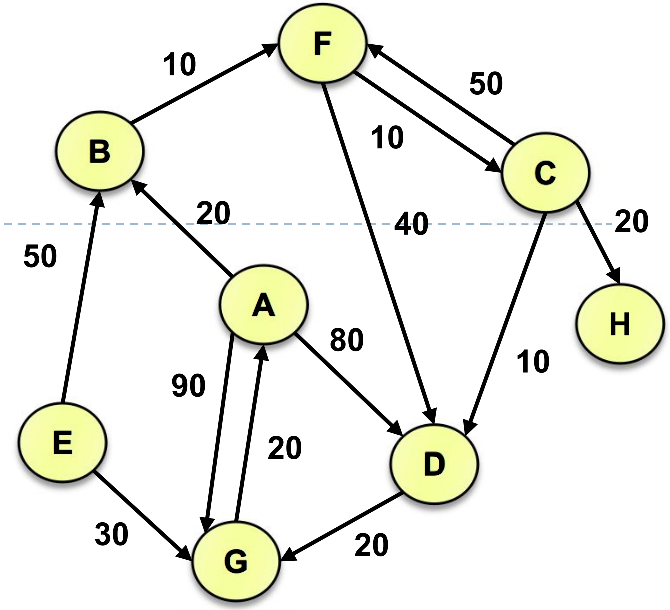
Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	60

custo de alcançar *G* passando por *D* (***70***) **é menor que** o custo de ir para *G* direto de *A* (***90***)

Algoritmo Dijkstra: Exemplo

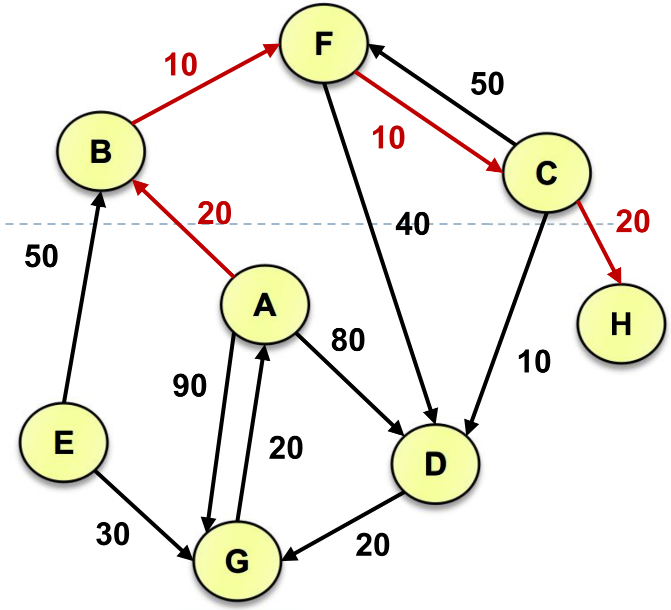
Qual o menor custo?

Tabulando os dados temos:



Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	60

Algoritmo Dijkstra: Exemplo



Qual o menor custo?

Tabulando os dados temos:

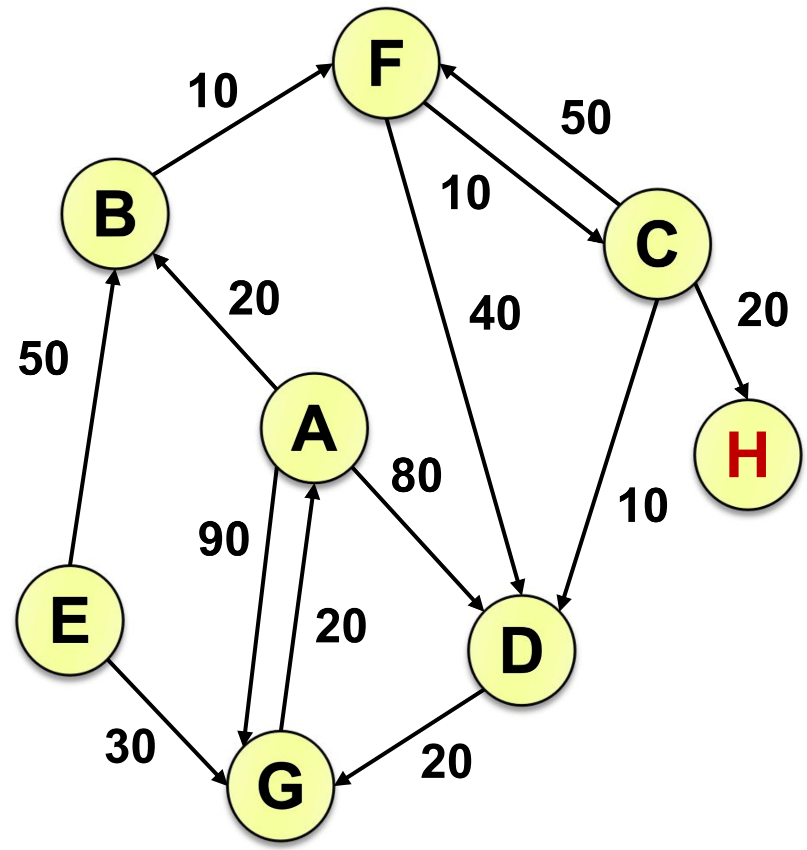
Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**

{A,B,F,C,H}* é o menor caminho de *A* até *H

Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*H*}, vamos analisar o menor caminho entre *H* e os outros vértices:

De *H* **NENHUM vértice é alcançado**



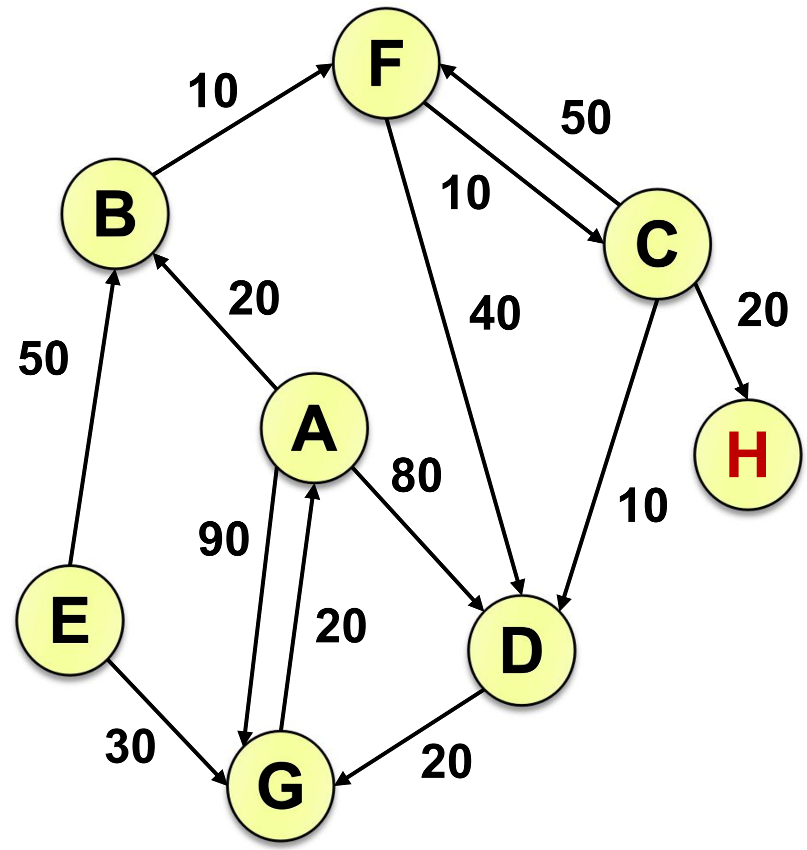
Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*H*}, vamos analisar o menor caminho entre *H* e os outros vértices:

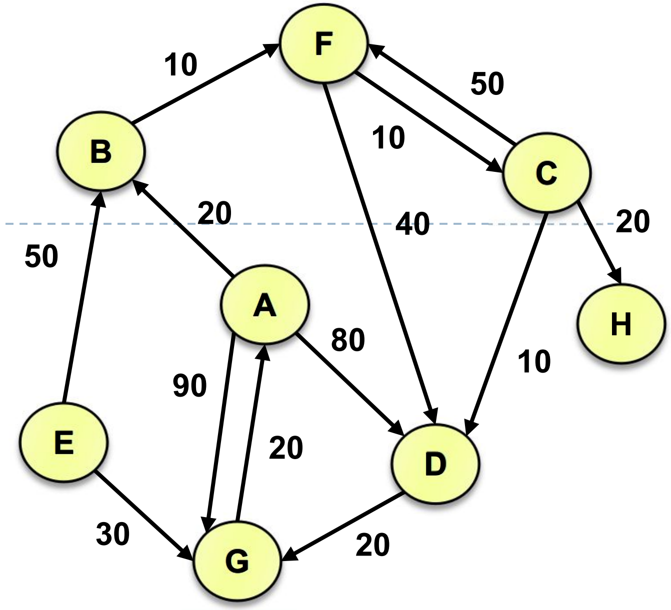
De *H* **NENHUM vértice é alcançado**

Repete-se todos os valores da tabela

Custo atual é mantido



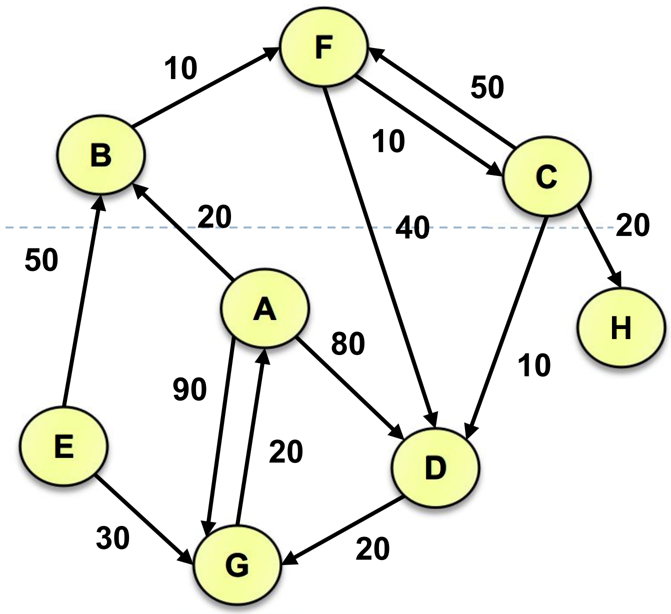
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**
H	{E,G}				∞		70	

Algoritmo Dijkstra: Exemplo

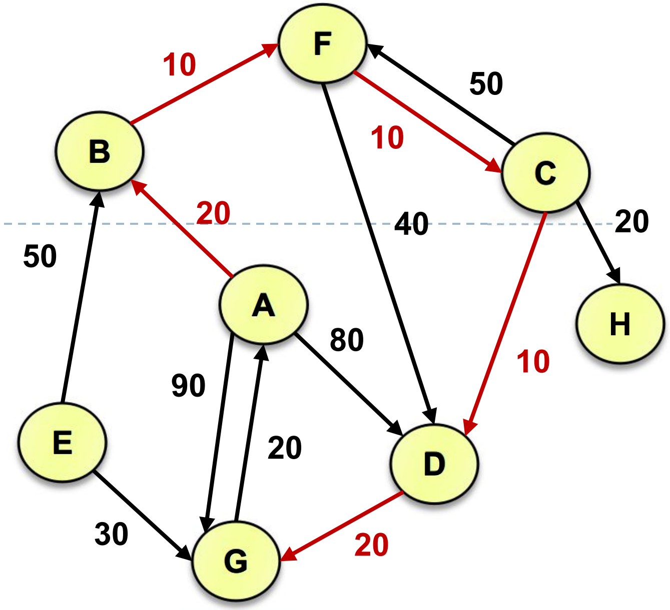


Qual o menor custo?

Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	**C**	**D**	**E**	**F**	**G**	**H**
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**
H	{E,G}				∞		70	

Algoritmo Dijkstra: Exemplo



Qual o menor custo?

Tabulando os dados temos:

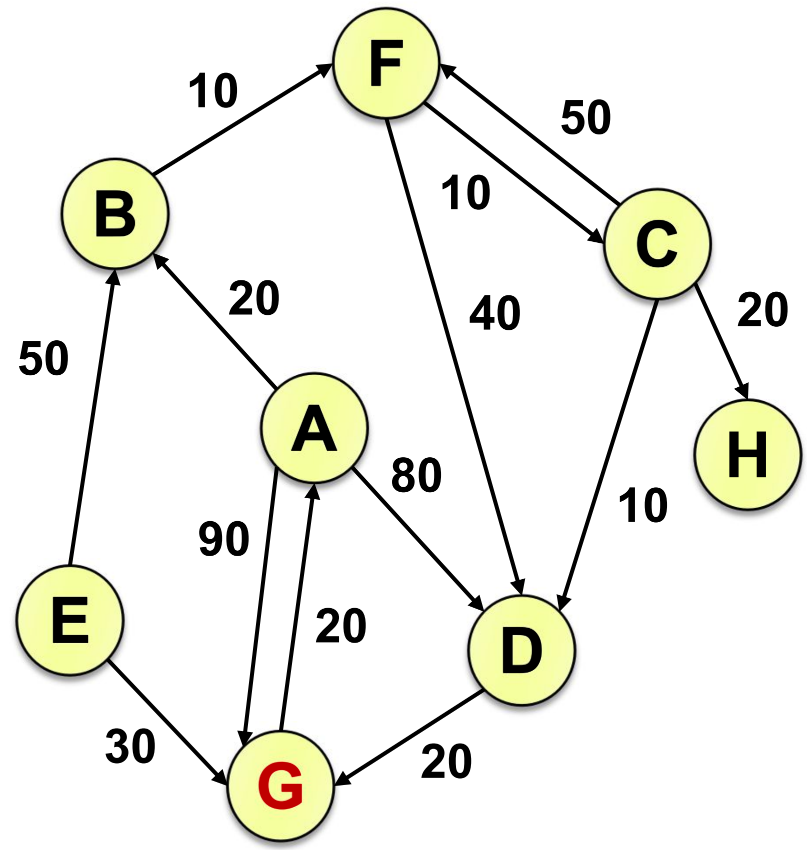
Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**
H	{E,G}				∞		**70**	

{A,B,F,C,D,G}* é o menor caminho de *A* até *G

Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*,*G*}, vamos analisar o menor caminho entre *G* e os outros vértices:

De *G* podemos ir para:

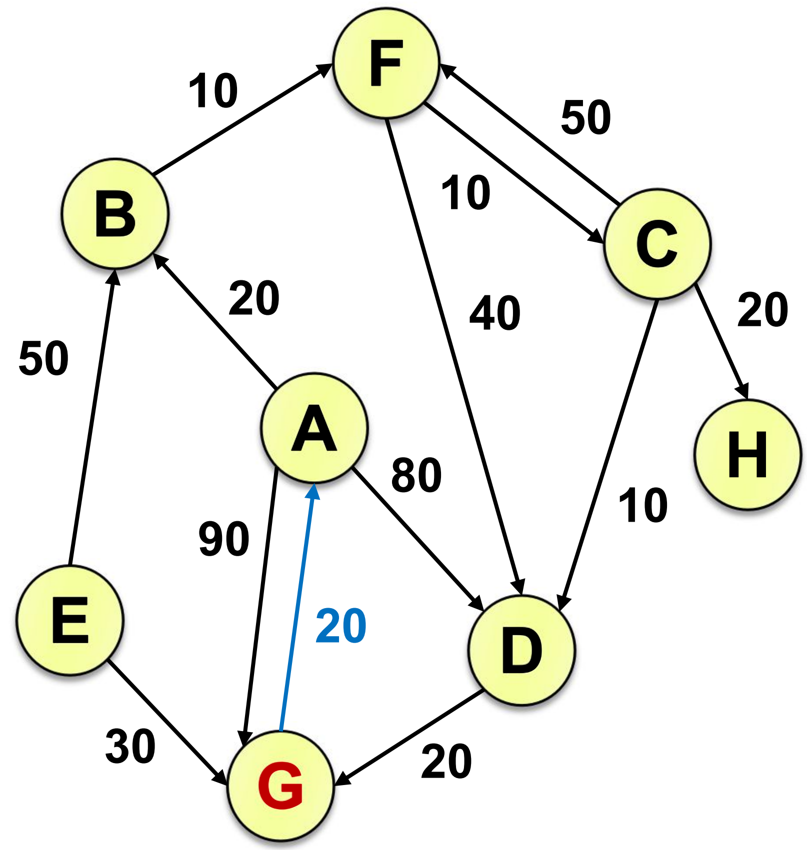


Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*,*G*}, vamos analisar o menor caminho entre *G* e os outros vértices:

De *G* podemos ir para:

***A* (custo = 20)**



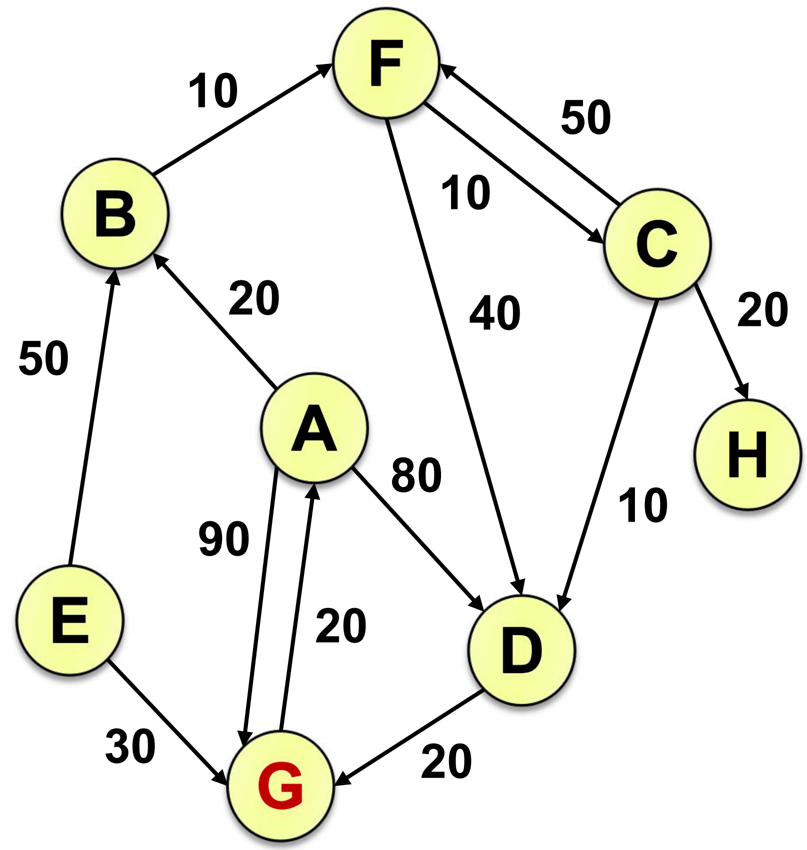
Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*,*G*}, vamos analisar o menor caminho entre *G* e os outros vértices:

De *G* podemos ir para:

- ~~***A* (custo = 20)**~~

***A* é o vértice inicial (custo = 0)**

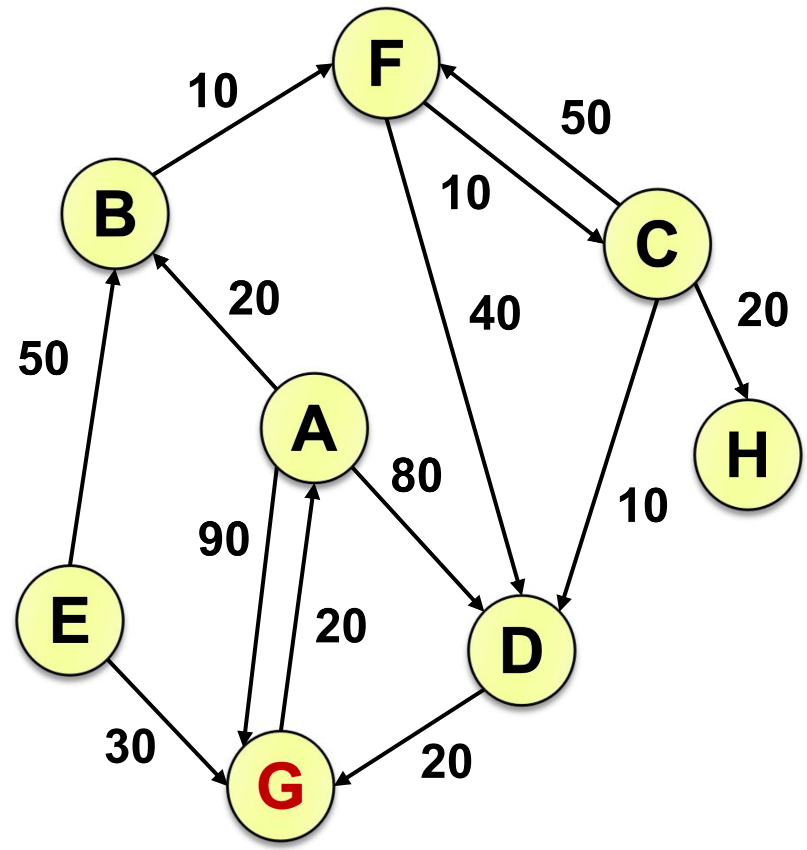


Algoritmo Dijkstra: Exemplo

Considerando o caminho {*A*,*B*,*F*,*C*,*D*,*G*}, vamos analisar o menor caminho entre *G* e os outros vértices:

De *G* podemos ir para:

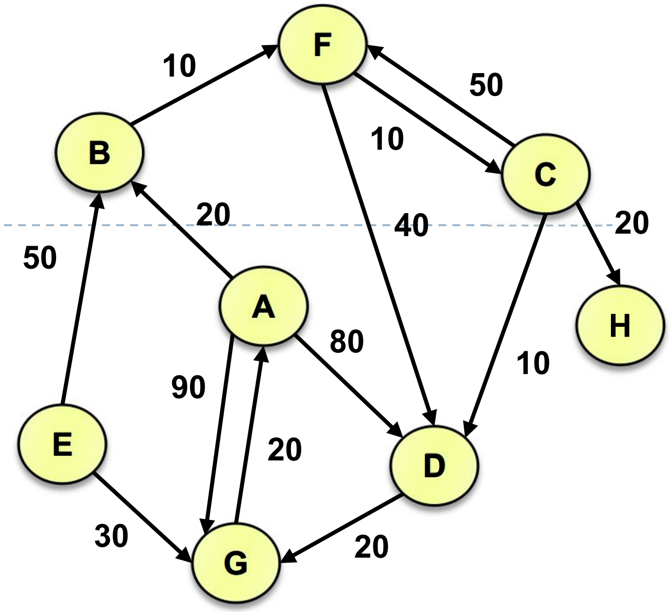
- ~~*A* (custo = 20)~~



Demais vértices não podem ser alcançados

Custo atual é mantido

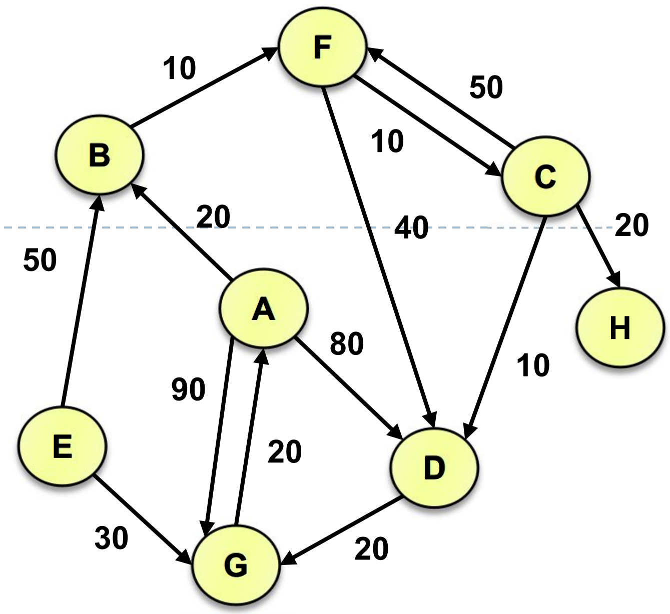
Algoritmo Dijkstra: Exemplo



Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**
H	{E,G}				∞		**70**	
G	{E}				∞			

Algoritmo Dijkstra: Exemplo



Não há caminho para o vértice *E* a partir de *A*

Tabulando os dados temos:

Nó em análise	Nós não visitados	Menor Distância até o momento						
		B	C	D	E	F	G	H
A	{B,C,D,E,F,G,H}	**20**	∞	80	∞	∞	90	∞
B	{C,D,E,F,G,H}		∞	80	∞	**30**	90	∞
F	{C,D,E,G,H}		**40**	70	∞		90	∞
C	{D,E,G,H}			**50**	∞		90	60
D	{E,G,H}				∞		70	**60**
H	{E,G}				∞		**70**	
G	{E}				∞			

Dijkstra: o algoritmo

Entrada:

- **Grafo** $G = \{V, A\}$
- **Vértice inicial** ou de origem v

Dijkstra: o algoritmo

Entrada:

Grafo $G = \{V,A\}$

Vértice inicial ou de origem v

Saída:

Vetor de distâncias D com o custo mínimo entre cada vértice e o vértice inicial

Dijkstra: o algoritmo

Entrada:

Grafo *G* = {*V*,*A*}

Vértice inicial ou de origem v_0

Saída:

Vetor de distâncias *D* com o custo mínimo entre cada vértice e o vértice inicial

Considerações:

O **peso das arestas** é representado por uma **matriz *C***

Os **vértices visitados** são armazenados em um **vetor *S***

Infinito será representado por um **valor muito alto**

Ex: *INT_MAX*

Dijkstra: o algoritmo

int ***dijkstra****(Grafo *G, int v_0, int *D)*

●●●

Dijkstra: o algoritmo

int ***dijkstra****(Grafo *G, int v_0, int *D)*

  Alocar espaço para os vetores ***S*** *e* ***D****;*

...

Dijkstra: o algoritmo

int ***dijkstra****(Grafo *G, int* v_0*, int *D)*

Alocar espaço para os vetores ***S*** *e* ***D****;*

// Inicializa elementos dos vetores

Inicializar todos os elementos do ***vetor S*** *com* ***ZERO****;*

Marcar o vértice $\boldsymbol{v_0}$ *como visitado em* ***S****;*

…

Dijkstra: o algoritmo

*int **dijkstra**(Grafo *G, int v_0, int *D)*

*Alocar espaço para os vetores **S** e **D**;*

// Inicializa elementos dos vetores

*Inicializar todos os elementos do **vetor S** com **ZERO**;*

*Marcar o vértice **v_0** como visitado em **S**;*

*Inicializar todos os elementos do **vetor D** com **infinito**;*

D[v_0] = 0; // Custo do vértice inicial é ZERO

PARA** cada vértice $v_i \in \{V\text{-}S\}$ **FAÇA

*   **SE** C[v_0 , v_i] ≠ 0 **ENTÃO***

*   D[v_i] = C[v_0 , v_i];*

*   **FIM_SE***

FIM_PARA

...

Dijkstra: o algoritmo

...

PARA *k = 2 até qtde_vertices* ***FAÇA***

FIM_PARA

FIM

Dijkstra: o algoritmo

...

PARA *k = 2 até qtde_vertices* ***FAÇA***

Encontre o vértice $v_k \in \{V\text{-}S\}$, tal que ***D[v_k] é o menor valor****;*

FIM_PARA

FIM

Dijkstra: o algoritmo

...

PARA *k = 2 até qtde_vertices* ***FAÇA***

Encontre o vértice $v_k \in \{V\text{-}S\}$, *tal que* ***D[*** $\boldsymbol{v_k}$ ***] é o menor valor****;*

Marcar o vértice $\boldsymbol{v_k}$ *como visitado em* ***S****;*

FIM_PARA

FIM

Dijkstra: o algoritmo

...

PARA *k = 2 até qtde_vertices* ***FAÇA***

Encontre o vértice $v_k \in \{V\text{-}S\}$*, tal que* ***D[*** $\boldsymbol{v_k}$ ***] é o menor valor****;*

Marcar o vértice $\boldsymbol{v_k}$ *como visitado em* ***S****;*

PARA *cada vértice* $v_j \in \{V\text{-}S\}$ ***FAÇA***

FIM_PARA

FIM_PARA

FIM

Dijkstra: o algoritmo

...

PARA *k = 2 até qtde_vertices* ***FAÇA***

Encontre o vértice $v_k \in \{V\text{-}S\}$*, tal que* ***D[*** $\mathbf{v_k}$ ***] é o menor valor****;*

Marcar o vértice $\mathbf{v_k}$ *como visitado em* ***S****;*

PARA *cada vértice* $v_j \in \{V\text{-}S\}$ ***FAÇA***

$D_{novo} = D[\, v_k \,] + C[\, v_k , v_j \,]$; // Calcula custo do caminho $v_0 ,\ldots, v_k , v_j$

FIM_PARA

FIM_PARA

FIM

Dijkstra: o algoritmo

…

PARA *k = 2 até qtde_vertices* ***FAÇA***

Encontre o vértice $v_k \in \{V\text{-}S\}$*, tal que* ***D[*** $\mathbf{v_k}$ ***] é o menor valor****;*

Marcar o vértice $\mathbf{v_k}$ *como visitado em* ***S****;*

PARA *cada vértice* $v_j \in \{V\text{-}S\}$ ***FAÇA***

$D_{novo} = D[\, v_k \,] + C[\, v_k , v_j \,]$; *// Calcula custo do caminho* $v_0 ,\ldots, v_k , v_j$

SE $D_{novo} < D[\, v_j \,]$ ***ENTÃO***

$D[\, v_j \,] = D_{novo}$;

FIM_SE

FIM_PARA

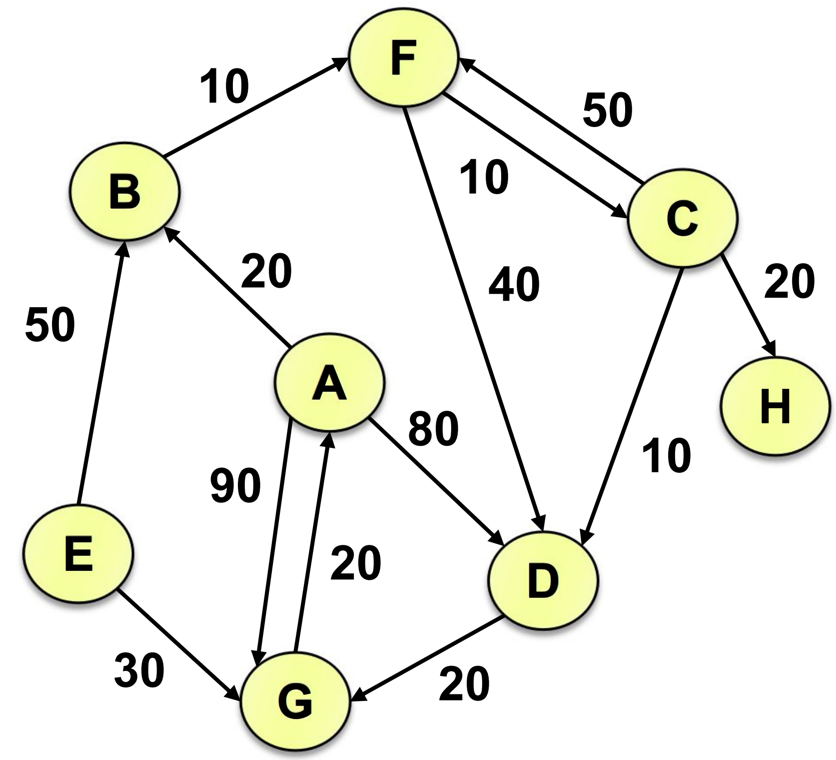
FIM_PARA

FIM

Algoritmo Dijkstra: teste de mesa

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0



Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	inf	80	inf	inf	90	inf

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	inf	80	inf	inf	90	inf

A	B	C	D	E	F	G	H
0	20	inf	80	inf	30	90	inf

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	inf	80	inf	inf	90	inf

A	B	C	D	E	F	G	H
0	20	inf	80	inf	30	90	inf

A	B	C	D	E	F	G	H
0	20	40	70	inf	30	90	inf

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	inf	80	inf	inf	90	inf

A	B	C	D	E	F	G	H
0	20	inf	80	inf	30	90	inf

A	B	C	D	E	F	G	H
0	20	40	70	inf	30	90	inf

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	90	60

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	90	60

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	90	60

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Algoritmo Dijkstra: teste de mesa

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

Vetor D

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	90	60

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

Algoritmo Dijkstra: teste de mesa

Vetor D

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	90	60

Matriz C

	A	B	C	D	E	F	G	H
A	0	20	0	80	0	0	90	0
B	0	0	0	0	0	10	0	0
C	0	0	0	10	0	50	0	20
D	0	0	0	0	0	0	20	0
E	0	50	0	0	0	0	30	0
F	0	0	10	40	0	0	0	0
G	20	0	0	0	0	0	0	0
H	0	0	0	0	0	0	0	0

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

A	B	C	D	E	F	G	H
0	20	40	50	inf	30	70	60

Dijkstra: o algoritmo

Podemos modificar o algoritmo para **guardar o caminho mais curto**

Dijkstra: o algoritmo

- Podemos modificar o algoritmo para **guardar o caminho mais curto**
  - Necessário um **vetor de antecessores** para guardar o **vértice imediatamente anterior no caminho** encontrado

Dijkstra: o algoritmo

*int **dijkstra_modificado**(Grafo *G, int v_0, int *D, **int *A**)*

*Alocar espaço para os vetores **A** , **S** e **D**;*

*Inicializar todos os elementos do **vetor A** com **-1**;*

*Inicializar todos os elementos do **vetor S** com **ZERO**;*

*Marcar o vértice $\boldsymbol{v_0}$ como visitado em **S**;*

*Inicializar todos os elementos do **vetor D** com **infinito**;*

D[v_0] = 0; // Custo do vértice inicial é ZERO

PARA** cada vértice $v_i \in \{V-S\}$ **FAÇA

    SE** C[v_0 , v_i] ≠ 0 **ENTÃO

        D[v_i] = C[v_0 , v_i];   A[v_i] = v_0 ;

    FIM_SE

FIM_PARA

…

Dijkstra: o algoritmo

…

PARA *k = 2 até qtde_vertices* **FAÇA**

Encontre o vértice $v_k \in \{V-S\}$*, tal que* **D[v_k] é o menor valor**;

Marcar o vértice $\mathbf{v_k}$ *como visitado em* **S**;

PARA *cada vértice* $v_j \in \{V-S\}$ **FAÇA**

$D_{novo} = D[\ v_k\ ] + C[\ v_k\ ,\ v_j\ ]$; // Calcula custo do caminho $v_0 ,\dots, v_k , v_j$

SE $D_{novo} < D[\ v_j\ ]$ **ENTÃO**

$D[\ v_j\ ] = D_{novo}$; $A[\ v_j\ ] = v_k$;

FIM_SE

FIM_PARA

FIM_PARA

FIM

Dijkstra: o algoritmo

- Podemos modificar o algoritmo para **guardar o caminho mais curto**
  - Necessário um **vetor de antecessores** para guardar o **vértice imediatamente anterior no caminho** encontrado

- Permite a **reconstrução do caminho** mais curto entre 2 vértices

Dijkstra: o algoritmo

Podemos modificar o algoritmo para **guardar o caminho mais curto**

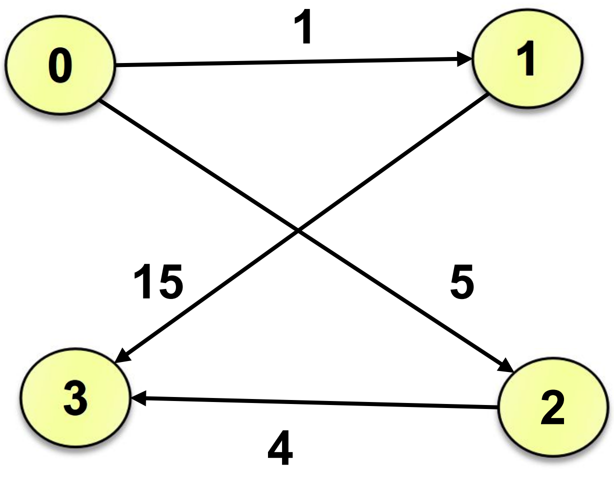
- Necessário um **vetor de antecessores** para guardar o **vértice imediatamente anterior no caminho** encontrado

Permite a **reconstrução do caminho** mais curto entre 2 vértices

```
void mostra_caminho(int Vini, int Vfim, int * A)  {
    if (Vini == Vfim)
        printf("%4d", Vini);
    else if(A[Vfim] == -1)
        printf("não existe caminho de %d para %d", Vini, Vfim);
    else {
        mostra_caminho(Vini, A[Vfim]);
        printf("%4d",Vfim);
    }
}
```

Dijkstra: exercício de fixação

Considerando o grafo abaixo, realize o teste de mesa do algoritmo modificado, apresentando o conteúdo dos vetores *S*, *D* e *A* a cada passo, e determine o caminho mais curto entre os vértices 0 e 3.



Dijkstra: resolução

```
grafo criado
  0   1   5   0
  0   0   0  15
  0   0   0   4
  0   0   0   0
```

```
S      1      0      0      0
D      0      1      5 300000
A     -1      0      0     -1

S      1      1      0      0
D      0      1      5     16
A     -1      0      0      1

S      1      1      1      0
D      0      1      5      9
A     -1      0      0      2
```

```
caminho mais curto entre    0 e    3 tem comprimento    9:    0   2   3
```

Busca do menor caminho: exercícios

1- Implemente um algoritmo Dijkstra modificado para mostrar o menor caminho entre os vértices de um dígrafo representado através de:

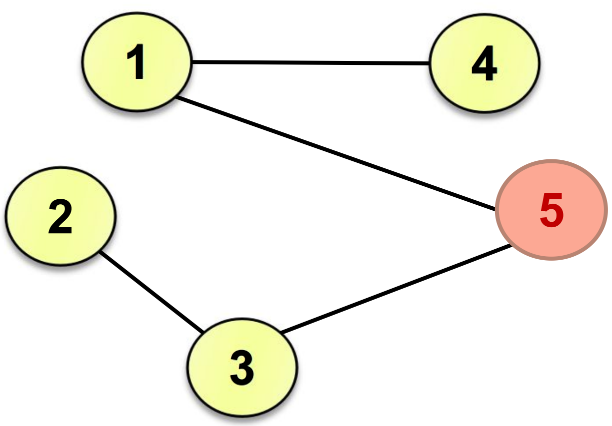
Matriz de adjacências

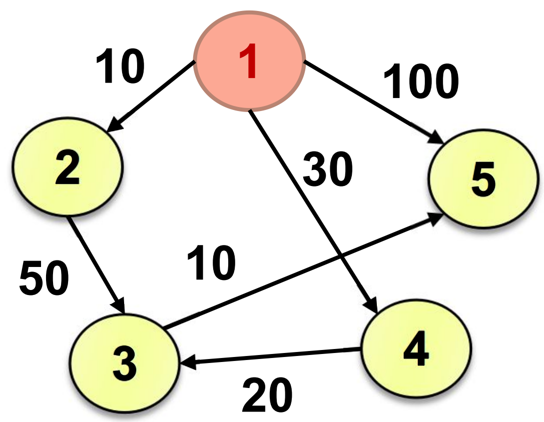
Listas de adjacências

2- Modifique os códigos implementados no exercício anterior para tratar um grafo não direcionado e não ponderado.

Busca em grafos: exercícios

1- Faça o teste de mesa para os 3 métodos de busca, considerando o grafo abaixo e o vértice em vermelho como inicial. Ao final de cada busca, mostre o comprimento do caminho encontrado entre o vértice inicial e os demais. Utilize os códigos implementados para confirmar sua resposta.





Busca em grafos: exercícios

2- Implemente uma função que determine se um vértice *X*, indicado pelo usuário, é conectado aos demais vértices do grafo.

3- Implemente uma função para determinar todos os vértices que não são conectados a nenhum outro vértice do grafo.

Bibliografia

Slides adaptados do material da Profa. Gina M. B. Oliveira, da Profa. Dra. Denise Guliato e do Prof. Dr. Bruno Travençolo.

BACKES, A. Linguagem C Descomplicada: portal de vídeo-aulas para estudo de programação. Disponível em: https://programacaodescomplicada.wordpress.com/indice/estrutura-de-dados/

CORMEN, T.H. et al. Algoritmos: Teoria e Prática, Campus, 2002

ZIVIANI, N. Projeto de algoritmos: com implementações em Pascal e C (2ª ed.), Thomson, 2004

MORAES, C.R. Estruturas de Dados e Algoritmos: uma abordagem didática (2ª ed.), Futura, 2003