ESTRUTURAS DE DADOS

Grafos (busca)

Roteiro

- Motivação
  - Busca em Profundidade
  - Busca em Largura
- Algoritmos
- Detalhes de Implementação

Motivação

Algoritmos para problemas diversos muitas vezes recaem em algum tipo de visitação sobre os vértices.

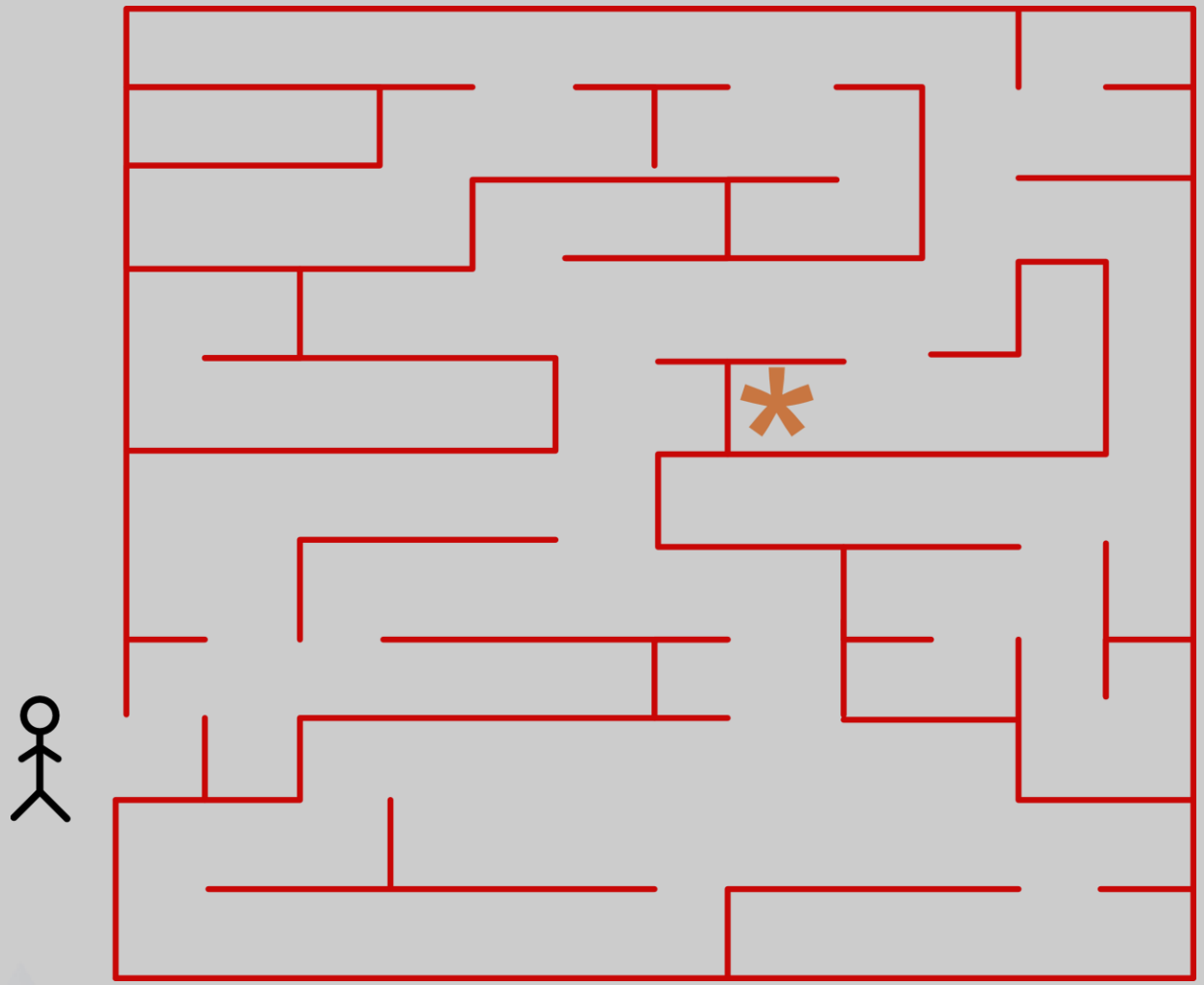
Entender como isso pode ser feito de maneira sistemática ajuda a evitar loops infinitos.

Veremos aqui a Busca em Profundidade e a Busca em Largura.

Busca em Profundidade

- A estratégia consiste em se aprofundar no grafo sempre que possível.
- Se estamos em um ponto do grafo e ainda há uma caminho não percorrido, seguimos esse caminho.
- Se estamos em um ponto do grafo e já percorremos tudo ao redor, voltamos para o vértice anterior (backtracking) procurando caminhos não explorados.
- A busca acaba quando:
  1. Encontramos o que queríamos.
  2. Visitamos todos os vértices e não achamos nada.
- Seguimos intuitivamente a busca em profundidade quando entramos em um labirinto.

Busca em Profundidade

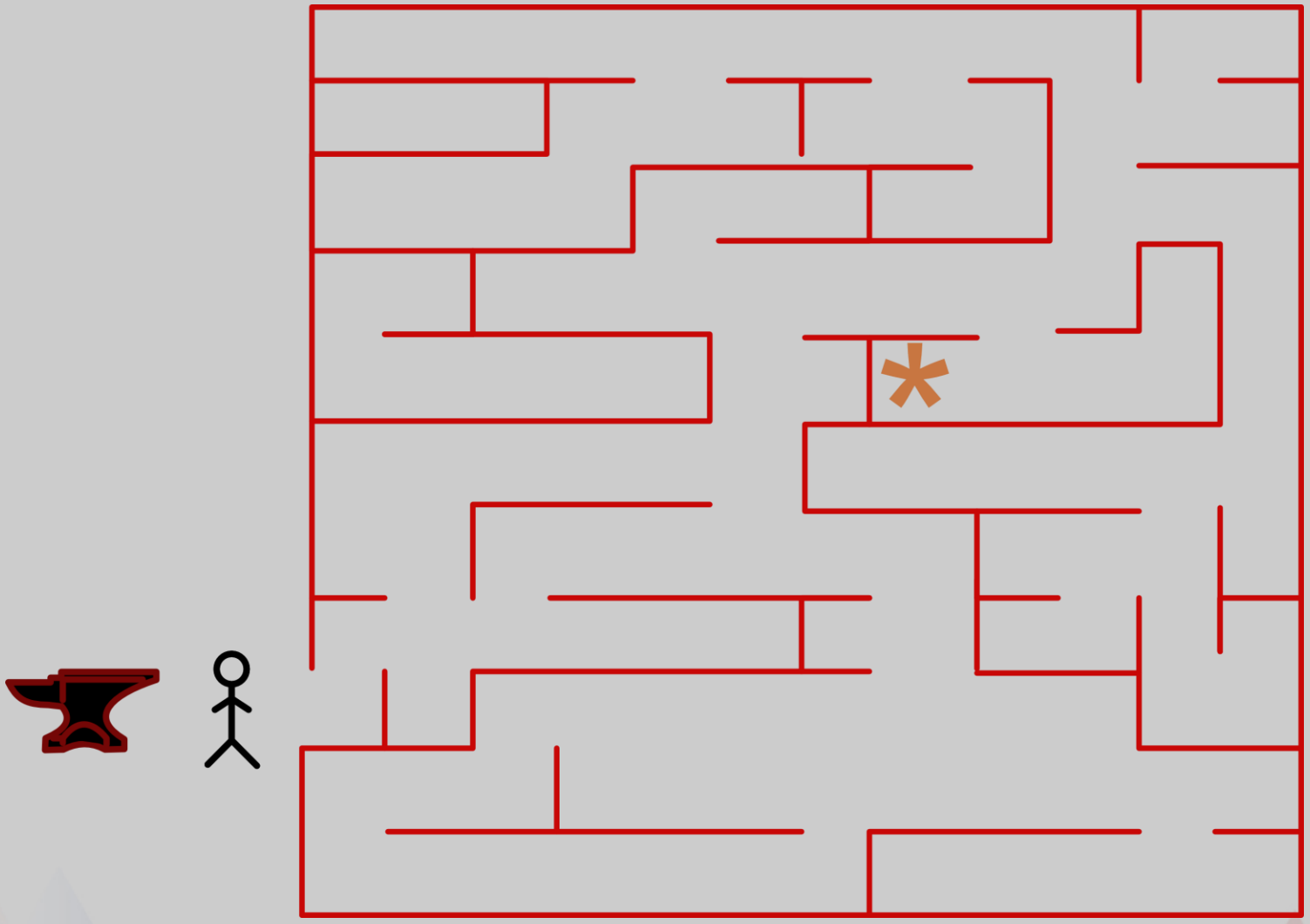


Ao procurar o alvo, andamos pelos quartos até encontrar ou chegar em uma situação em que já vimos tudo o que precisa ser descoberto.

Busca em Largura

- A estratégia consiste em explorar sem se afastar tanto do ponto inicial.
- Primeiramente, seguimos um caminho próximo da origem. Se não acharmos o que queríamos, voltamos para o início e tentamos outro caminho .
- Ao achar o que queríamos, garantimos que sabemos uma maneira rápida de chegar até ele.
- Em geral, só verificamos vértices a uma distância **k+1** se todos os vértices de distância **k** já tiverem sido visitados.
- Intuitivamente, usamos essa ideia quando queremos explorar o ambiente e encontrar um caminho curto até um ponto.

Busca em Largura

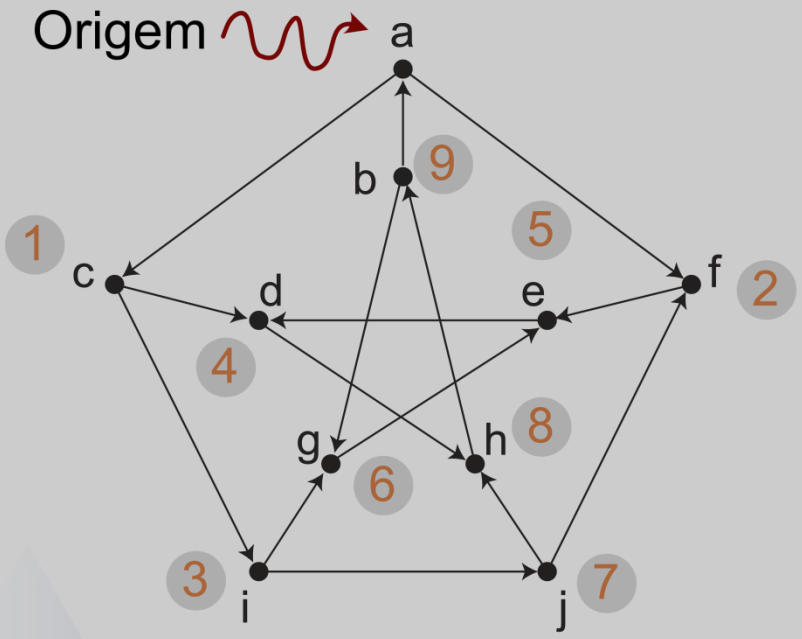


Ao procurar o alvo, queremos um caminho curto para depois carregar algo muito pesado até lá.

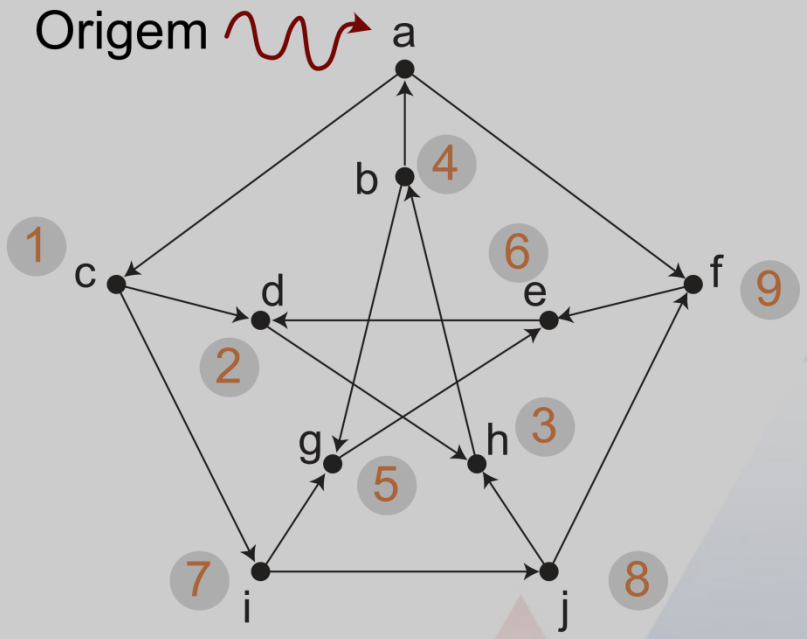
Ordem de Visitação

- **A ordem de visitação muda conforme a estratégia seguida.**
- **Para cada estratégia, existem várias ordens de possíveis visitação .**

Largura



Profundidade



Algoritmos

Em nossos algoritmos, tentaremos resolver o seguinte problema:

Existe um caminho entre o vértice x e o vértice y?

Iniciaremos com a busca em profundidade e depois passaremos para a busca em largura.

Busca em Profundidade

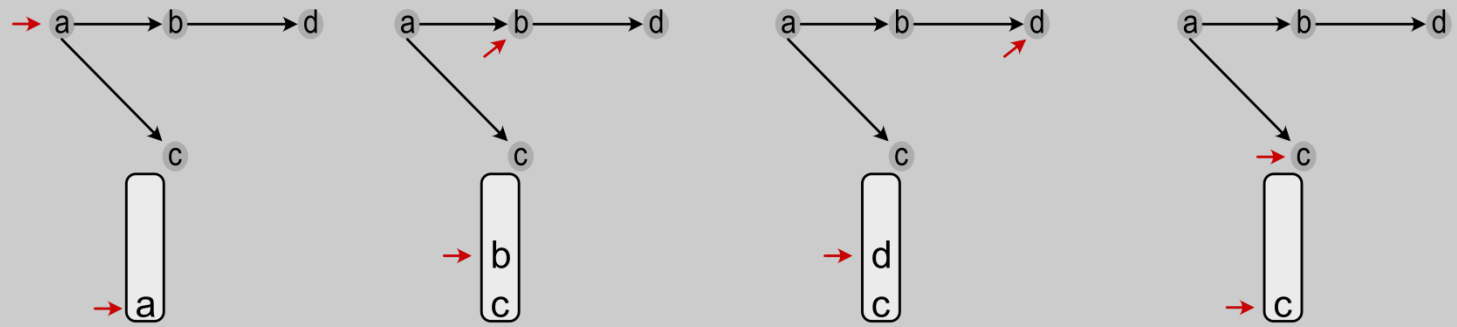
- **Como queremos fazer um *backtracking* sempre que chegarmos a um "beco sem saída", uma pilha será usada para organizarmos os nós.**

Passo 1: visitaremos a porque está no topo. Ele foi colocado por ser origem da busca.
O nó será retirado da pilha e seus filhos adicionados.

Passo 3: ao visitar b, d irá para o topo da pilha. Assim, d será o próximo visitado. Visitar d não adicionará ninguém na pilha.

Grafo:

Pilha:



Passo 2: o nó b ficou no topo, então será visitado antes de c. Assim, o caminho que começa em c será iniciado se a visitação de b não achar o que queremos.

Passo 4: encerramos a busca pelo grafo. Note que aprofundamos em b antes de visitar c. Isso caracteriza a busca em profundidade.

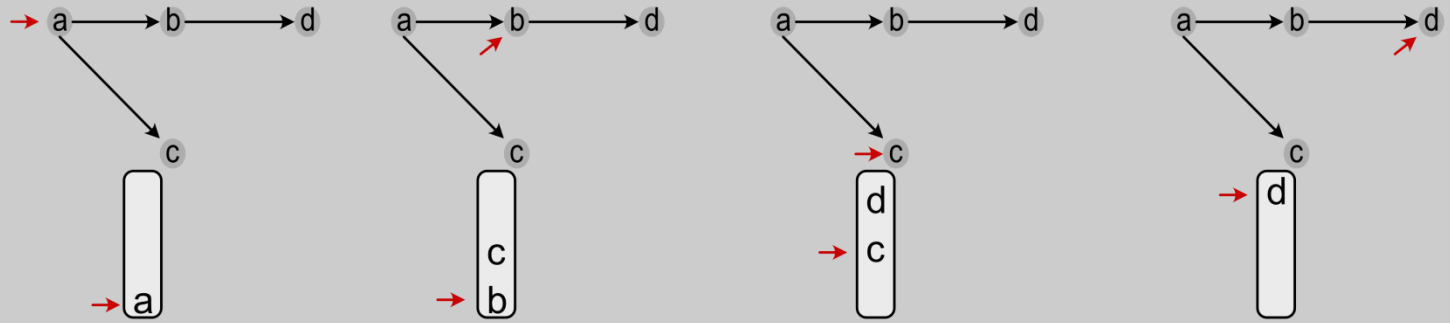
Busca em Largura

- **Quemos que os nós sejam visitados em camadas, sem aprofundar muito na busca, uma fila será usada para organizarmos os nós.**

Passo 1: visitaremos a porque está no topo. Ele foi colocado por ser origem da busca. O nó será retirado da fila e seus filhos adicionados.

Passo 3: ao visitar b, d irá para o final da fila. Assim, c será o próximo visitado. Visitar c não adicionará ninguém na fila.

Grafo:



Fila:

Passo 2: o nó b ficou na frente, então será visitado antes de c, mas queremos que c tenha prioridade sobre os filhos de b.

Passo 4: encerramos a busca pelo grafo. Note que visitamos os nós em camadas. Os filhos de a foram visitados primeiro.

```
BUSCA EM PROFUNDIDADE(grafo, origem, destino)
1 found <- FALSO
2 Inicializar PILHA p vazia
3 p.push(origem)
4
5 FAÇA
6   v <- p.pop()
7   SE v = destino
8     ENTÃO found <- VERDADEIRO
9     SENÃO se v não visitado
10      PARA CADA adj adjacente de v FAÇA
10        SE adj não visitado
11          p.push(adj)
12 ENQUANTO found = FALSO e p não vazia
13
14 SE found = FALSO
15   IMPRIMA "Caminho não existe"
```

```
BUSCA EM LARGURA(grafo, origem, destino)
1 found <- FALSO
2 Inicializar FILA f vazia
3 f.enqueue(origem)
4
5 FAÇA
6   v <- f.dequeue()
7   SE v = destino
8     ENTÃO found <- VERDADEIRO
9     SENÃO se v não visitado
10      PARA CADA adj adjacente de v FAÇA
10        SE adj não visitado
11          f.enqueue(adj)
12 ENQUANTO found = FALSO e f não vazia
13
14 SE found = FALSO
15   IMPRIMA "Caminho não existe"
```

Detalhes de Implementação

Em nossa implementação, iremos utilizar as estruturas de dados fila e pilha já vistas em nosso curso.

Usaremos os métodos clearMarks, isMarked e markVertex para garantir que não visitaremos um determinado nó mais de uma vez.

```
void depthFirstSearch(Graph& graph, Vertex origem, Vertex destino) {
  Stack  vertexStack; bool found = false; Vertex vertex;
  graph.clearMarks();
  vertexStack.push(origem);
  do {
    vertex = vertexStack.pop();
    if (vertex.getNome() == destino.getNome())  {
      cout << "Encontrado: " << vertex.getNome() << ";" << endl;
      found = true;
    } else {
      if (!graph.isMarked(vertex)) {
        graph.markVertex(vertex);
        cout << "Visitando: " << vertex.getNome() << endl;
        Queue adjacents;
        graph.getAdjacents(vertex, adjacents);
        while (!adjacents.isEmpty()) {
          Vertex adjacent = adjacents.dequeue();
          if (!graph.isMarked(  adjacent  )){
            cout << "Empilhando: " << adjacent.getNome() << endl;
            vertexStack.push(adjacent );
          }
        }
      }
    }
  } while (!vertexStack.isEmpty() && !found);
```

```
void breadthFirstSearch(Graph& graph, Vertex origem, Vertex destino) {
  Queue vertexQueue;  bool found = false; Vertex vertex;
  graph.clearMarks();
  vertexQueue.enqueue(origem);
  do {
    vertex  = vertexQueue.dequeue();
    if (vertex.getNome() == destino.getNome()) {
      cout << "Encontrado: " << vertex.getNome() << ";" << endl;
      found = true;
    } else {
      if (!graph.isMarked(vertex)) {
        graph.markVertex(vertex);
        cout << "Visitando: " << vertex.getNome() << endl;
        Queue adjacents;
        graph.getAdjacents(vertex, adjacents);
        while (!adjacents.isEmpty()) {
          Vertex adjacent = adjacents.dequeue();
          if (!graph.isMarked( adjacent ) ){
            cout << "Enfileirando: " << adjacent.getNome() << endl;
            vertexQueue.enqueue( adjacent ) ;
          }
        }
      }
    }
  } while (!vertexQueue.isEmpty() && !found);
```

ESTRUTURAS DE DADOS

Grafos (busca)