DOCUMENTAÇÃO DAS FUNÇÕES:

ANÁLISE DE GRAFOS

bfsConexo()

Propósito:
Realiza uma busca em largura (BFS) para verificar a conectividade de um grafo a partir de um vértice inicial. Essa função percorre todos os vértices alcançáveis a partir do vértice inicial, marcando-os como visitados.

Entrada:

- adjacencia: Um mapa de listas de adjacência que representa o grafo, onde a chave é o nome de um vértice e o valor é um vetor contendo os vértices adjacentes.
- inicio: O vértice inicial a partir do qual a BFS será realizada.
- visitados: Um conjunto de vértices que já foram visitados. Esta estrutura é atualizada conforme a função é executada.

Saída:
A função não possui um retorno explícito, mas atualiza o conjunto `visitados` com todos os vértices que são conectados ao vértice inicial.

Conexão com Grafos:
A função utiliza o algoritmo de Busca em Largura (BFS), que é uma técnica fundamental para explorar grafos e verificar a conectividade entre os vértices. Este algoritmo garante que todos os vértices alcançáveis a partir de um ponto inicial serão visitados, sendo essencial na análise de componentes conexas em grafos.

Explicação das Variáveis e Chamadas

- **adjacencia**: Um map que mapeia um vértice (representado por uma string) para um vetor de vértices adjacentes, formando a lista de adjacência do grafo.
- **inicio**: Representa o vértice a partir do qual a BFS será iniciada. Ele serve como ponto de partida para a exploração do grafo.
- **visitados**: Um set que armazena todos os vértices que já foram visitados durante a execução da BFS. Ele evita que o algoritmo processe novamente um vértice já explorado, prevenindo loops infinitos.
- **fila**: Uma queue utilizada para gerenciar a ordem em que os vértices são explorados. Ela segue a lógica FIFO (First In, First Out), garantindo que a exploração seja feita em camadas, característica fundamental da BFS.
- **u**: A variável que armazena o vértice atualmente sendo processado na BFS.
- **v**: Um vértice adjacente ao vértice u. É examinado para determinar se já foi visitado ou se deve ser inserido na fila para futura exploração.

ehConexo()

Propósito:
Determina se um grafo é conexo (ou fortemente conexo, se direcionado), verificando se todos os vértices podem ser alcançados a partir de qualquer outro vértice. Para grafos não direcionados, verifica se o grafo é conexo, e para grafos direcionados, verifica se é fortemente conexo, analisando tanto o grafo original quanto o grafo com as arestas invertidas.

Entrada:

- arestas: Um vetor de objetos do tipo Aresta, cada um representando uma aresta no grafo, com informações sobre o vértice de origem e o vértice de destino.
- vertices: Um vetor contendo todos os vértices presentes no grafo.
- direcionado: Um booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:
Retorna um valor booleano (true ou false):

- true: Indica que o grafo é conexo (ou fortemente conexo, se direcionado).
- false: Indica que o grafo não é conexo (ou não é fortemente conexo, se direcionado).

Conexão com Grafos:
A função é essencial na teoria dos grafos, pois permite determinar a conectividade de um grafo, um conceito fundamental que tem impacto em várias aplicações práticas, como análise de redes, algoritmos de roteamento e estudo de componentes fortemente conexas em grafos direcionados.

Explicação das Variáveis e Chamadas

- **arestas**: Um vetor de objetos Aresta, onde cada objeto representa uma aresta do grafo com informações sobre a origem e o destino da aresta.
- **vertices**: Um vetor contendo todos os vértices do grafo. Ele é usado para inicializar as listas de adjacência e para garantir que todos os vértices sejam verificados durante a BFS.
- **direcionado**: Um booleano que indica se o grafo é direcionado (true) ou não (false). Isso determina como as arestas são adicionadas ao mapa de adjacência e se será necessário realizar uma BFS no grafo invertido.

- **adjacencia**: Um map que armazena as listas de adjacência do grafo original. As chaves são os vértices e os valores são vetores de vértices adjacentes.
- **adjacenciaInvertida**: Um map que armazena as listas de adjacência do grafo com as arestas invertidas. Isso é usado para verificar a conectividade em grafos direcionados.
- **visitados**: Um set que mantém o registro dos vértices visitados durante a execução da BFS no grafo original.
- **visitadosInvertidos**: Um set que mantém o registro dos vértices visitados durante a execução da BFS no grafo invertido. Usado apenas se o grafo for direcionado.
- **bfsConexo**: Função auxiliar chamada para realizar a busca em largura (BFS) tanto no grafo original quanto no invertido. É utilizada para explorar o grafo e verificar sua conectividade.
- **return true/false**: Retorna true se o grafo é conexo ou fortemente conexo, e false caso contrário. Em um grafo não direcionado, a função retorna false assim que encontra um vértice que não foi visitado. Em um grafo direcionado, ela também verifica a conectividade no grafo invertido.

ehBipartido()

Propósito:

Determina se um grafo é bipartido, ou seja, se é possível colorir os vértices do grafo usando duas cores de modo que nenhum par de vértices adjacentes tenha a mesma cor. Essa verificação é feita utilizando o algoritmo de busca em largura (BFS).

Entrada:

- arestas: Um vetor de objetos do tipo Aresta, onde cada objeto representa uma aresta do grafo, contendo informações sobre o vértice de origem e o vértice de destino.
- vertices: Um vetor contendo todos os vértices presentes no grafo.
- direcionado: Um booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:

Retorna um valor booleano (true ou false):

- true: Indica que o grafo é bipartido.
- false: Indica que o grafo não é bipartido.

Conexão com Grafos:
A bipartição de um grafo é um conceito fundamental na teoria dos grafos, com aplicações em problemas como coloração de grafos, otimização de emparelhamentos e análise de redes. Esta função verifica a bipartição utilizando BFS, que é uma técnica eficiente para explorar grafos e determinar a possibilidade de tal coloração.

Explicação das Variáveis e Chamadas

- **arestas**: Um vetor de objetos Aresta, onde cada objeto representa uma aresta do grafo com informações sobre a origem e o destino da aresta.
- **vertices**: Um vetor contendo todos os vértices do grafo. Ele é usado para inicializar o mapa de cores e para garantir que todos os vértices sejam verificados durante a BFS.
- **direcionado**: Um booleano que indica se o grafo é direcionado (true) ou não (false). Isso determina se as arestas serão adicionadas na direção oposta no mapa de adjacência.
- **adjacencia**: Um map que armazena as listas de adjacência do grafo. As chaves são os vértices, e os valores são vetores de vértices adjacentes.
- **cor**: Um map que armazena a cor atribuída a cada vértice. Inicialmente, todos os vértices têm a cor -1, indicando que ainda não foram coloridos. Durante a BFS, os vértices são coloridos com 0 ou 1.
- **fila**: Uma queue usada para gerenciar a ordem de exploração dos vértices durante a BFS. Ela segue a lógica FIFO (First In, First Out), essencial para a exploração em camadas.
- **u**: A variável que armazena o vértice atualmente sendo processado na BFS.
- **v**: Um vértice adjacente ao vértice u. Ele é examinado para determinar se já foi colorido ou se deve ser colorido e adicionado à fila para futura exploração.
- **bfsConexo**: Embora não seja utilizada diretamente nesta função, o conceito de BFS é aplicado aqui para explorar o grafo e determinar a bipartição.
- **return true/false**: Retorna true se o grafo for bipartido, e false caso contrário. A função retorna false assim que identifica que dois vértices adjacentes têm a mesma cor.

ehEuleriano()

Propósito:
Determina se um grafo é euleriano, ou seja, se existe um ciclo euleriano no grafo. Um ciclo euleriano é um ciclo que percorre todas as arestas do grafo exatamente uma vez, retornando ao vértice de partida. A função realiza essa verificação tanto para grafos direcionados quanto para não direcionados.

Entrada:

- arestas: Um vetor de objetos do tipo Aresta, onde cada objeto representa uma aresta no grafo, contendo informações sobre o vértice de origem e o vértice de destino.
- vertices: Um vetor contendo todos os vértices presentes no grafo.
- direcionado: Um booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:

Retorna um valor booleano (true ou false):

- true: Indica que o grafo é euleriano.
- false: Indica que o grafo não é euleriano.

Conexão com Grafos:

A presença de um ciclo euleriano é um conceito importante na teoria dos grafos, com aplicações em problemas de roteamento, logística e análise de redes. Um grafo é euleriano se for possível percorrer todas as suas arestas uma única vez, retornando ao ponto de partida. A função verifica essa propriedade utilizando critérios clássicos de grafos eulerianos para grafos tanto direcionados quanto não direcionados.

Explicação das Variáveis e Chamadas

- **arestas**: Um vetor de objetos Aresta, onde cada objeto representa uma aresta do grafo com informações sobre a origem e o destino da aresta.
- **vertices**: Um vetor contendo todos os vértices do grafo. Ele é usado para inicializar e verificar os graus de entrada e saída (para grafos direcionados) ou simplesmente o grau (para grafos não direcionados) de cada vértice.
- **direcionado**: Um booleano que indica se o grafo é direcionado (true) ou não (false). Isso determina qual lógica será aplicada para verificar se o grafo é euleriano.
- **grauEntrada**: Um map que armazena o grau de entrada de cada vértice em grafos direcionados. O grau de entrada é o número de arestas que chegam a um vértice.
- **grauSaida**: Um map que armazena o grau de saída de cada vértice em grafos direcionados. O grau de saída é o número de arestas que partem de um vértice.
- **grauVertices**: Um map que armazena o grau de cada vértice em grafos não direcionados. O grau de um vértice é o número de arestas que incidem sobre ele.
- **ehConexo**: Função auxiliar que verifica se o grafo é conexo. Um grafo euleriano precisa ser conexo, ou fortemente conexo no caso de grafos direcionados.
- **return true/false**: Retorna true se o grafo for euleriano, e false caso contrário. A função retorna false se qualquer vértice não satisfaz as condições necessárias para que o grafo seja euleriano.

Função dfsDetectarCiclo

Propósito:
A função dfsDetectarCiclo realiza uma busca em profundidade (DFS) para detectar a presença de ciclos em um grafo. Se um ciclo for encontrado, a função retorna true; caso contrário, retorna false.

Entrada:

- vertice: O vértice atual a partir do qual a DFS está sendo executada.
- pai: O vértice anterior que levou ao vértice atual, utilizado para evitar confundir a aresta de retorno ao pai como um ciclo.
- adjacencia: Um mapa de adjacência representando o grafo, onde as chaves são os vértices e os valores são vetores contendo os vértices adjacentes.
- cor: Um mapa que armazena o estado (cor) de cada vértice, indicando se ele ainda não foi visitado (BRANCO), está sendo visitado (CINZA), ou já foi totalmente explorado (PRETO).
- direcionado: Um valor booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:

- A função retorna true se um ciclo for detectado no grafo e false caso contrário.

Conexão com Grafos:
A função está intimamente relacionada ao conceito de ciclos em grafos. A detecção de ciclos é um problema clássico em teoria dos grafos e é essencial em diversas aplicações, como verificação de dependências em sistemas de software ou na análise de estruturas de dados.

Explicação das Variáveis e Chamadas

- **vertice**: Representa o nó atual sendo explorado na DFS.
- **pai**: Armazena o vértice anterior (pai) na árvore de DFS para evitar falsos positivos na detecção de ciclos.
- **adjacencia**: Mapa que contém a lista de adjacência do grafo.
- **cor**: Mapa que guarda o estado de cada vértice (BRANCO, CINZA, PRETO) durante a execução do algoritmo.
- **direcionado**: Indica se o grafo em questão é direcionado ou não, o que pode afetar a interpretação das arestas.

Função detectarCiclos

Propósito:
A função detectarCiclos verifica se um grafo contém ciclos, utilizando uma busca em profundidade (DFS). O objetivo é determinar a presença de ciclos, o que é crucial para várias aplicações em grafos, como a verificação de consistência em dependências ou a identificação de laços em redes.

Entrada:

- vertices: Um vetor de strings contendo os vértices do grafo.
- arestas: Um vetor de objetos do tipo Aresta, representando as arestas do grafo.
- direcionado: Um booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:

- A função retorna true se algum ciclo for detectado no grafo e false se o grafo for acíclico.

Conexão com Grafos:
A detecção de ciclos é uma operação essencial na teoria dos grafos, especialmente em grafos direcionados, onde os ciclos podem indicar dependências cíclicas. Em grafos não direcionados, ciclos podem representar redundâncias ou rotas alternativas.

Explicação das Variáveis e Chamadas

- **adjacencia**: Um mapa que armazena a lista de adjacências do grafo. Cada chave é um vértice, e o valor é um vetor de vértices adjacentes.
- **cor**: Um mapa que guarda o estado de cada vértice (BRANCO, CINZA, PRETO), representando se o vértice ainda não foi visitado, está em processamento ou já foi totalmente processado.
- **vertices**: Um vetor contendo todos os vértices do grafo.
- **arestas**: Um vetor de objetos Aresta, representando as conexões entre os vértices.
- **direcionado**: Booleano que determina se o grafo é direcionado ou não, influenciando como as arestas são adicionadas à lista de adjacências.

.

Função dfsCompConexas

Propósito:
A função dfsCompConexas realiza uma busca em profundidade (DFS) em um grafo a partir de um vértice inicial, marcando todos os vértices conectados a ele como visitados. É usada para encontrar e marcar todos os vértices em um componente conexo do grafo.

Entrada:

- vertice: O vértice inicial para a busca em profundidade.
- adjacencia: Um mapa que representa a lista de adjacência do grafo, onde as chaves são vértices e os valores são vetores contendo os vizinhos.
- visitados: Um conjunto que armazena os vértices já visitados durante a busca.

Saída:

- A função não retorna nenhum valor. Em vez disso, modifica o conjunto visitados para incluir todos os vértices acessíveis a partir do vertice inicial.

Conexão com Grafos:
A função é fundamental para identificar e explorar componentes conexos em um grafo. Em um grafo não direcionado, um componente conexo é um subconjunto de vértices em que há um caminho entre qualquer par de vértices. Esta função é útil para algoritmos que precisam identificar ou processar todos os vértices em um componente conexo.

Explicação das Variáveis e Chamadas

- **vertice**: O vértice atual sendo explorado na DFS.
- **adjacencia**: Mapa que representa a lista de adjacência do grafo.
- **visitados**: Conjunto que mantém o controle dos vértices já visitados para evitar reprocessamento.

Função contarComponentesConexas

Propósito:
A função contarComponentesConexas calcula o número de componentes conexos em um grafo não direcionado. Um componente conexo é um subconjunto de vértices onde qualquer par de vértices é acessível um ao outro através de caminhos dentro desse subconjunto.

Entrada:

- arestas: Um vetor de objetos do tipo Aresta, representando as arestas do grafo.
- vertices: Um vetor de strings contendo os vértices do grafo.

Saída:

- Retorna um inteiro que representa o número total de componentes conexos no grafo.

Conexão com Grafos:
Contar componentes conexos é essencial para entender a estrutura de um grafo. Isso é útil em aplicações como a análise de redes, a verificação de conectividade e o planejamento de rotas.

Explicação das Variáveis e Chamadas

- **adjacencia**: Mapa que representa a lista de adjacência do grafo. Cada vértice é associado a um vetor de seus vértices adjacentes.
- **visitados**: Conjunto que rastreia os vértices já visitados durante a busca para evitar contagens duplicadas.
- **componentesConexas**: Contador que mantém o número total de componentes conexos encontrados.

Função tarjanDFS

Propósito:

A função tarjanDFS é uma implementação do algoritmo de Tarjan para encontrar componentes fortemente conexos em um grafo direcionado. Um componente fortemente conexo (CFC) é um subconjunto de vértices em que cada vértice é acessível a partir de qualquer outro vértice desse subconjunto.

Entrada:

- vertice: Vértice atual que está sendo visitado.
- adjacencia: Mapa de listas de adjacência que representa o grafo.
- indices: Mapa que armazena o índice de descoberta de cada vértice.
- baixos: Mapa que armazena o menor índice de descoberta alcançável a partir de cada vértice.
- pilha: Pilha que armazena os vértices que estão sendo processados.
- naPilha: Conjunto que rastreia os vértices que estão na pilha.
- componentes: Vetor de vetores que armazena os componentes fortemente conexos encontrados.
- index: Contador global que indica o próximo índice de descoberta a ser atribuído.

Saída:

- A função não retorna um valor diretamente. No entanto, ela modifica o vetor componentes para armazenar os componentes fortemente conexos encontrados no grafo.

Conexão com Grafos:

Encontrar componentes fortemente conexos é fundamental para a análise de grafos direcionados, sendo útil em aplicações como a decomposição de redes, análise de dependências e detecção de ciclos.

Explicação das Variáveis e Chamadas

- **indices**: Armazena o índice de descoberta de cada vértice, indicando a ordem em que os vértices foram visitados.
- **baixos**: Armazena o menor índice de descoberta alcançável a partir de cada vértice, ajudando a identificar os pontos de raiz dos componentes fortemente conexos.
- **pilha**: Mantém os vértices que estão sendo processados pela DFS.
- **naPilha**: Conjunto que rastreia quais vértices estão atualmente na pilha.

- **componentes**: Armazena os componentes fortemente conexos identificados durante a execução do algoritmo.
- **index**: Contador que é incrementado a cada novo vértice descoberto, fornecendo um índice único para cada vértice.

Função contarComponentesFortementeConexas

Propósito:

A função contarComponentesFortementeConexas utiliza o algoritmo de Tarjan para calcular a quantidade de componentes fortemente conexos (CFCs) em um grafo direcionado. Cada CFC é um conjunto de vértices onde cada vértice é alcançável a partir de qualquer outro vértice no mesmo conjunto.

Entrada:

- arestas: Um vetor de arestas que define as conexões direcionadas entre os vértices do grafo.
- vertices: Um vetor de strings que contém todos os vértices do grafo.

Saída:

- A função retorna um inteiro que representa a quantidade de componentes fortemente conexos encontrados no grafo.

Conexão com Grafos:

Identificar componentes fortemente conexos é importante para analisar a estrutura de grafos direcionados. É útil em diversas aplicações, como detectar ciclos e entender a decomposição de redes.

Explicação das Variáveis e Chamadas

- **adjacencia**: Mapa que armazena a lista de adjacências, associando cada vértice aos seus vértices adjacentes (destinos das arestas).
- **indices**: Mapa que armazena o índice de descoberta de cada vértice.
- **baixos**: Mapa que armazena o menor índice de descoberta acessível a partir de cada vértice.
- **pilha**: Pilha que armazena vértices durante a execução da DFS para formar componentes.
- **naPilha**: Conjunto que mantém o rastreamento dos vértices que estão na pilha.
- **componentes**: Vetor de vetores que armazena os componentes fortemente conexos encontrados durante a execução do algoritmo.
- **index**: Variável que mantém o índice de descoberta atual dos vértices.

Função dfsAux

Propósito:
A função dfsAux realiza uma busca em profundidade (DFS) modificada em um grafo para identificar pontos de articulação e pontes. Pontos de articulação são vértices cuja remoção aumenta o número de componentes conectados do grafo, e pontes são arestas cuja remoção também aumenta o número de componentes conectados.

Entrada:

- vertice: O vértice atual sendo visitado na DFS.
- pai: O vértice pai na DFS, ou seja, o vértice a partir do qual a DFS chegou ao vértice atual.
- tempo: Um contador que registra o tempo de descoberta de cada vértice.
- adjacencia: Mapa que representa a lista de adjacências do grafo, onde cada vértice está associado a um vetor de seus vizinhos.
- descoberta: Mapa que armazena o tempo de descoberta de cada vértice.
- maisBaixo: Mapa que armazena o menor tempo de descoberta acessível a partir de cada vértice.
- visitados: Conjunto que mantém o rastreamento dos vértices já visitados.
- articulacoes: Conjunto que armazenará os pontos de articulação encontrados no grafo.
- pontes: Vetor que armazenará as pontes encontradas no grafo.
- filhos: Um contador que mantém o número de filhos de um vértice na árvore DFS.

Saída:

- A função não retorna um valor diretamente, mas atualiza as estruturas de dados articulacoes e pontes com os pontos de articulação e pontes identificados.

Conexão com Grafos:
A identificação de pontos de articulação e pontes é crucial em análise de redes, detecção de vulnerabilidades em grafos conectados e em várias aplicações de redes de comunicação.

Explicação das Variáveis e Chamadas

- **vertice**: Vértice atual sendo processado na DFS.
- **pai**: Vértice de origem na DFS, que precede o vertice atual.
- **tempo**: Contador incremental que marca a ordem de descoberta dos vértices.

- **adjacencia**: Mapa que representa as conexões entre os vértices do grafo.
- **descoberta**: Mapa que armazena o tempo de descoberta de cada vértice.
- **maisBaixo**: Mapa que armazena o menor tempo acessível a partir de cada vértice.
- **visitados**: Conjunto de vértices que já foram visitados.
- **articulacoes**: Conjunto onde serão armazenados os pontos de articulação identificados.
- **pontes**: Vetor onde serão armazenadas as pontes identificadas.
- **filhos**: Contador que registra o número de filhos de um vértice na árvore DFS.

Função imprimirArticulacoes

Propósito:

A função imprimirArticulacoes identifica e imprime os vértices de articulação de um grafo não direcionado em ordem lexicográfica. Um vértice de articulação, ou ponto de articulação, é um vértice cuja remoção aumenta o número de componentes conectados do grafo.

Entrada:

- arestas: Um vetor de arestas, onde cada aresta conecta dois vértices no grafo.
- vertices: Um vetor contendo todos os vértices do grafo.

Saída:

- A função não retorna um valor, mas imprime os vértices de articulação encontrados no grafo em ordem lexicográfica.

Conexão com Grafos:

Identificar pontos de articulação é fundamental para entender a vulnerabilidade de redes e garantir a conectividade em grafos, especialmente em contextos como redes de computadores, circuitos elétricos, ou qualquer sistema onde a conectividade é crucial.

Explicação das Variáveis e Chamadas

- **arestas**: Contém as conexões entre os vértices do grafo.
- **vertices**: Lista de todos os vértices presentes no grafo.
- **adjacencia**: Mapa que armazena a lista de adjacências de cada vértice, representando o grafo.
- **visitados**: Conjunto que mantém o rastreamento dos vértices já visitados durante a DFS.
- **descoberta**: Mapa que armazena o tempo de descoberta de cada vértice durante a DFS.

- **maisBaixo**: Mapa que armazena o menor tempo de descoberta acessível a partir de cada vértice.
- **articulacoes**: Conjunto que armazena os vértices de articulação identificados.
- **pontes**: Vetor que armazenaria as pontes identificadas, mas não é utilizado diretamente nesta função.
- **tempo**: Contador que mantém o tempo de descoberta durante a DFS.
- **filhos**: Variável usada para contar o número de filhos de um vértice na árvore DFS.

Função contarArestasPonte

Propósito:

A função contarArestasPonte tem como objetivo identificar e contar o número de arestas ponte em um grafo não direcionado. Uma aresta ponte é aquela que, ao ser removida, aumenta o número de componentes conectados do grafo.

Entrada:

- arestas: Um vetor de arestas, onde cada aresta conecta dois vértices no grafo.
- vertices: Um vetor contendo todos os vértices do grafo.

Saída:

- Um inteiro representando o número total de arestas ponte identificadas no grafo.

Conexão com Grafos:

Identificar arestas ponte é crucial em análise de redes para entender a robustez e vulnerabilidade do grafo. A remoção de uma ponte pode desconectar partes significativas do grafo, afetando a conectividade e funcionalidade da rede.

Explicação das Variáveis e Chamadas

- **arestas**: Vetor contendo as arestas do grafo, representando as conexões entre vértices.
- **vertices**: Lista de todos os vértices presentes no grafo.
- **adjacencia**: Mapa que armazena a lista de adjacências para cada vértice, representando o grafo.
- **visitados**: Conjunto que mantém o rastreamento dos vértices já visitados durante a DFS.
- **descoberta**: Mapa que armazena o tempo de descoberta de cada vértice durante a DFS.
- **maisBaixo**: Mapa que armazena o menor tempo de descoberta acessível a partir de cada vértice.

- **articulacoes**: Conjunto para armazenar vértices de articulação identificados, mas não utilizado diretamente nesta função.
- **pontes**: Vetor que armazena as arestas identificadas como pontes.
- **tempo**: Contador que mantém o tempo de descoberta durante a DFS.
- **filhos**: Variável usada para contar o número de filhos de um vértice na árvore DFS.

Função dfsImprimirArvore

Propósito:

A função dfsImprimirArvore realiza uma busca em profundidade (DFS) em um grafo e gera uma árvore de busca a partir do vértice de partida. Durante a busca, a função coleta os identificadores (idsArvore) das arestas que pertencem à árvore de busca, garantindo que as arestas sejam exploradas de maneira ordenada (lexicograficamente).

Entrada:

- vertice: O vértice inicial a partir do qual a DFS começa.
- adjacencia: Um mapa que representa a lista de adjacências do grafo, ou seja, as conexões entre os vértices.
- visitados: Um conjunto que mantém o rastreamento dos vértices que já foram visitados durante a DFS.
- idsArvore: Um vetor que armazena os identificadores das arestas que compõem a árvore de busca.
- idAresta: Um mapa que associa pares de vértices (representando uma aresta) a um identificador de aresta específico.

Saída:

- A função não retorna valores diretamente, mas preenche o vetor idsArvore com os identificadores das arestas que pertencem à árvore de busca gerada pela DFS.

Conexão com Grafos:

Gerar uma árvore de busca em profundidade a partir de um grafo é uma operação fundamental para analisar a estrutura do grafo. Identificar as arestas que pertencem à árvore de busca permite entender a conectividade e as relações entre os vértices.

Explicação das Variáveis e Chamadas

- **vertice**: O vértice atual sendo explorado na busca em profundidade.
- **adjacencia**: Mapa que contém as listas de adjacência para cada vértice, representando as conexões do grafo.

- **visitados**: Conjunto que rastreia quais vértices já foram explorados para evitar ciclos na DFS.
- **idsArvore**: Vetor que armazena os identificadores das arestas que fazem parte da árvore de busca.
- **idAresta**: Mapa que mapeia pares de vértices para um identificador de aresta, usado para identificar as arestas durante a DFS.

Função imprimirArvoreEmProfundidade

Propósito:
A função imprimirArvoreEmProfundidade realiza a busca em profundidade (DFS) em um grafo, partindo de um vértice raiz especificado, e imprime a sequência de identificadores de arestas que compõem a árvore gerada por essa busca. Caso o grafo seja desconexo, a função lida com isso adequadamente, considerando a raiz fornecida e verificando se o grafo é conexo.

Entrada:

- arestas: Um vetor contendo as arestas do grafo, onde cada aresta possui um vértice de origem, um vértice de destino, e um identificador único.
- raiz: O vértice de onde a DFS começa e onde a árvore de busca será gerada.
- direcionado: Um booleano que indica se o grafo é direcionado ou não.

Saída:

- A função não retorna valores diretamente, mas imprime a sequência de identificadores de arestas que compõem a árvore gerada pela DFS. Caso a raiz não exista no grafo ou o grafo seja desconexo e a raiz não seja "0", a função imprime -1.

Conexão com Grafos:
A construção e visualização da árvore de busca em profundidade a partir de um vértice raiz específico são essenciais para entender a conectividade e a estrutura do grafo. A função também verifica se o grafo é conexo, o que é crucial para determinar a completude da árvore gerada.

Explicação das Variáveis e Chamadas

- **arestas**: Contém as arestas do grafo, cada uma com origem, destino, e identificador.
- **raiz**: O vértice inicial de onde a DFS começa.
- **direcionado**: Indica se o grafo é direcionado. Isso pode afetar como as arestas são tratadas na DFS.
- **adjacencia**: Mapa que armazena as listas de adjacência do grafo para cada vértice.

- **idAresta**: Mapa que associa pares de vértices (representando uma aresta) a um identificador de aresta específico.
- **visitados**: Conjunto que rastreia quais vértices já foram explorados na DFS.
- **idsArvore**: Vetor que armazena os identificadores das arestas que fazem parte da árvore de busca.
- **vertices**: Vetor que armazena todos os vértices do grafo para auxiliar na verificação de conectividade.
- **conexo**: Booleano que armazena o resultado da verificação de conectividade do grafo.

Função bfsImprimirArvore

Propósito:

A função bfsImprimirArvore realiza uma busca em largura (BFS) em um grafo, a partir de um vértice raiz, e gera uma árvore de busca em largura. Durante esse processo, a função também armazena os identificadores das arestas que compõem essa árvore, permitindo que a sequência seja impressa posteriormente.

Entrada:

- raiz: O vértice inicial de onde a BFS começa.
- adjacencia: Um mapa que representa a lista de adjacência do grafo, onde a chave é um vértice e o valor é um vetor de vértices adjacentes.
- pais: Um mapa que registra o "pai" de cada vértice na árvore de busca, ou seja, o vértice a partir do qual cada vértice foi descoberto durante a BFS.
- idsArvore: Um vetor onde serão armazenados os identificadores das arestas que compõem a árvore de busca em largura.
- idAresta: Um mapa que associa pares de vértices a identificadores de arestas únicos, permitindo identificar a aresta específica entre dois vértices.

Saída:

- A função não retorna valores diretamente, mas popula o vetor idsArvore com os identificadores das arestas que fazem parte da árvore de busca em largura gerada.

Conexão com Grafos:

A busca em largura (BFS) é uma técnica fundamental em teoria dos grafos, especialmente útil para explorar grafos de forma nível por nível, garantindo que todos os vértices em um mesmo nível de profundidade em relação à raiz sejam visitados antes de passar para o próximo nível. Essa função constrói a árvore de busca em largura, que é uma representação importante das relações entre os vértices no contexto da BFS.

Explicação das Variáveis e Chamadas

- **raiz**: Vértice de onde a BFS inicia.
- **adjacencia**: Mapa que armazena as listas de adjacência do grafo para cada vértice.
- **pais**: Mapa que rastreia de onde cada vértice foi descoberto na árvore de busca.
- **idsArvore**: Vetor que armazena os identificadores das arestas que compõem a árvore de busca em largura.
- **idAresta**: Mapa que associa pares de vértices a identificadores de arestas únicos.
- **visitados**: Conjunto que rastreia quais vértices já foram explorados na BFS.
- **fila**: Fila que gerencia a ordem de visita dos vértices durante a BFS, garantindo que a exploração seja feita em largura.

Função imprimirArvoreEmLargura

Propósito:

A função imprimirArvoreEmLargura é responsável por gerar e imprimir a árvore de busca em largura (BFS) de um grafo a partir de um vértice raiz específico. Ela utiliza a função auxiliar bfsImprimirArvore para realizar a BFS e identificar as arestas que fazem parte da árvore gerada. O propósito é visualizar a estrutura da árvore de busca em largura, que pode ser útil para diversas análises em grafos.

Entrada:

- arestas: Um vetor de estruturas Aresta que representam as arestas do grafo, cada uma contendo a origem, o destino e um identificador.
- raiz: O vértice a partir do qual a BFS será iniciada.
- direcionado: Um booleano que indica se o grafo é direcionado (true) ou não (false).

Saída:

- A função imprime os identificadores das arestas que compõem a árvore de busca em largura. Se o grafo for desconexo e a raiz não for "0", imprime -1 para indicar que a árvore não foi gerada a partir dessa raiz.

Conexão com Grafos:

A BFS é uma técnica fundamental na teoria dos grafos, usada para explorar os vértices de um grafo de maneira nivelada, ou seja, primeiro todos os vértices de uma mesma distância da raiz são visitados, antes de passar para os próximos níveis. A função imprimirArvoreEmLargura permite visualizar essa exploração como uma árvore, o que é útil em várias aplicações como busca de caminhos mínimos em grafos não ponderados, análise de conectividade, entre outros.

Explicação das Variáveis e Chamadas

- **arestas**: Vetor contendo as arestas do grafo, cada uma com seu vértice de origem, destino e um identificador único.
- **raiz**: Vértice inicial a partir do qual a BFS será realizada.
- **direcionado**: Booleano que determina se o grafo é direcionado ou não.
- **adjacencia**: Mapa que armazena as listas de adjacência do grafo para cada vértice.
- **idAresta**: Mapa que associa pares de vértices a identificadores de arestas únicos.
- **pais**: Mapa que armazena o vértice "pai" de cada vértice na árvore gerada pela BFS.
- **idsArvore**: Vetor que armazena os identificadores das arestas que compõem a árvore de busca em largura.
- **vertices**: Vetor que contém todos os vértices únicos do grafo.
- **conexo**: Booleano que indica se o grafo é conexo ou não, determinado pela função ehConexo.

Função encontrar

Propósito:
A função encontrar é usada para encontrar o representante (ou líder) do conjunto ao qual um determinado vértice pertence em uma estrutura de conjuntos disjuntos (ou union-find). Ela implementa a técnica de compressão de caminho para otimizar o processo de busca, atualizando a estrutura de dados de modo que todos os vértices de um conjunto apontem diretamente para o representante do conjunto.

Entrada:

- vertice: O vértice para o qual se deseja encontrar o representante do conjunto.
- pais: Um vetor que armazena o pai de cada vértice na estrutura de conjuntos disjuntos.

Saída:

- Retorna o representante do conjunto ao qual o vértice pertence.

Conexão com Grafos:
A função encontrar é uma parte crucial das operações de conjuntos disjuntos, frequentemente usadas em algoritmos de grafos como o algoritmo de Kruskal para encontrar árvores geradoras mínimas. A compressão de caminho melhora a eficiência dessas operações, tornando a busca de conjuntos mais rápida, o que é essencial para lidar com grandes grafos.

Explicação das Variáveis e Chamadas

- **vertice**: O vértice cujo representante do conjunto está sendo buscado.

- **pais**: Vetor que armazena o pai de cada vértice. Cada elemento pais[i] é o pai do vértice i. Inicialmente, cada vértice é seu próprio pai.

Função unir

Propósito:
A função unir é usada para unir dois conjuntos distintos na estrutura de conjuntos disjuntos (ou union-find). Ela combina os conjuntos de dois vértices em um único conjunto, utilizando a técnica de união por rank para manter a árvore de busca balanceada, o que ajuda a manter a eficiência das operações.

Entrada:

- vertice1: O primeiro vértice cujo conjunto será unido com o conjunto do segundo vértice.
- vertice2: O segundo vértice cujo conjunto será unido com o conjunto do primeiro vértice.
- pais: Vetor que armazena o pai de cada vértice na estrutura de conjuntos disjuntos.
- rank: Vetor que armazena a profundidade da árvore para cada raiz, ajudando a decidir qual árvore deve se tornar a nova raiz.

Saída:

- Não retorna valor. Modifica a estrutura dos conjuntos disjuntos ao unir dois conjuntos.

Conexão com Grafos:
A função unir é utilizada em algoritmos de grafos que requerem a união de componentes, como o algoritmo de Kruskal para encontrar a árvore geradora mínima. O uso do rank ajuda a manter as operações eficientes ao evitar árvores desbalanceadas.

Explicação das Variáveis e Chamadas

- **vertice1** e **vertice2**: Vértices cujos conjuntos precisam ser unidos.
- **pais**: Vetor que representa a estrutura dos conjuntos disjuntos. Cada elemento pais[i] é o pai do vértice i.
- **rank**: Vetor que armazena a profundidade (ou rank) da árvore de cada raiz. Usado para manter a árvore balanceada.

Função calcularMST

Propósito:
A função calcularMST calcula o valor total da Árvore Geradora Mínima (MST) de um grafo usando o algoritmo de Kruskal. O MST é uma árvore que conecta todos os vértices do grafo com o menor custo total de arestas possível.

Entrada:

- arestas: Vetor contendo as arestas do grafo, cada uma com um peso associado.
- vertices: Vetor contendo todos os vértices do grafo.

Saída:

- Retorna o valor total das arestas que formam a Árvore Geradora Mínima.

Conexão com Grafos:
O algoritmo de Kruskal é utilizado para encontrar o MST de um grafo ponderado. A função calcularMST implementa esse algoritmo ao ordenar as arestas por peso e usar a estrutura de conjuntos disjuntos para evitar ciclos e construir a árvore mínima.

Explicação das Variáveis e Chamadas

- **arestas**: Vetor de arestas do grafo, onde cada aresta tem um peso associado.
- **vertices**: Vetor de vértices do grafo.
- **arestasOrdenadas**: Cópia do vetor arestas, ordenada em ordem crescente de peso.
- **indice**: Mapeia cada vértice para um índice único.
- **pais**: Vetor que representa a estrutura de conjuntos disjuntos, armazenando o pai de cada vértice.
- **rank**: Vetor que armazena a profundidade da árvore para cada raiz, utilizado para manter a estrutura balanceada.
- **valorTotal**: Armazena o valor total das arestas incluídas na MST.

Função dfsTopologico

Propósito:
A função dfsTopologico realiza uma busca em profundidade (DFS) em um grafo direcionado para gerar uma ordenação topológica dos vértices. A ordenação topológica é uma disposição linear dos vértices de um grafo direcionado acíclico (DAG) onde, para cada aresta u→vu \to vu→v, o vértice uuu aparece antes de vvv na ordenação.

Entrada:

- vertice: O vértice atual a partir do qual a DFS começa.
- adjacencia: Mapa que representa a lista de adjacências do grafo.
- visitado: Mapa que marca os vértices já visitados durante a DFS.
- topoStack: Pilha que armazena a ordenação topológica dos vértices.

Saída:

- Atualiza a pilha topoStack com a ordenação topológica dos vértices.

Conexão com Grafos:
A ordenação topológica é uma técnica fundamental para muitos problemas em grafos direcionados acíclicos, como a ordenação de tarefas e a resolução de dependências.

Explicação das Variáveis e Chamadas

- **vertice**: O vértice a partir do qual a DFS é iniciada.
- **adjacencia**: Estrutura que armazena as arestas do grafo. Cada chave é um vértice, e o valor é uma lista dos vértices adjacentes.
- **visitado**: Mapa que indica se um vértice foi visitado durante a DFS.
- **topoStack**: Pilha que armazena os vértices na ordem inversa à topológica.

Função imprimirOrdenacaoTopologica

Propósito:
A função imprimirOrdenacaoTopologica realiza a ordenação topológica de um grafo direcionado e imprime a sequência dos vértices conforme a ordenação.

Entrada:

- vertices: Lista de todos os vértices do grafo.
- arestas: Lista de todas as arestas do grafo.
- direcionado: Booleano indicando se o grafo é direcionado.

Saída:

- Imprime a ordenação topológica dos vértices se o grafo for direcionado; caso contrário, imprime -1.

Conexão com Grafos:
A ordenação topológica é usada para resolver problemas de dependência em grafos direcionados acíclicos (DAGs), como a ordenação de tarefas ou a resolução de conflitos.

Explicação das Variáveis e Chamadas

- **vertices**: Vetor contendo todos os vértices do grafo.
- **arestas**: Vetor contendo todas as arestas do grafo.
- **direcionado**: Indica se o grafo é direcionado, determinando se a ordenação topológica deve ser realizada.

Função dijkstra

Propósito:
A função dijkstra encontra o caminho mínimo entre dois vértices em um grafo ponderado usando o algoritmo de Dijkstra.

Entrada:

- arestas: Lista de arestas do grafo, cada uma com origem, destino e peso.

- vertices: Lista de todos os vértices do grafo.
- origem: Vértice de onde começa a busca do caminho mínimo.
- destino: Vértice para o qual se quer encontrar o caminho mínimo.

Saída:

- Retorna a distância mínima entre o vértice origem e o vértice destino. Se não houver caminho, retorna -1.

Conexão com Grafos:

O algoritmo de Dijkstra é usado para encontrar o caminho mais curto em grafos ponderados e não negativos, o que é útil em muitos contextos, como roteamento e análise de redes.

Explicação das Variáveis e Chamadas

- **arestas**: Vetor contendo as arestas do grafo, cada uma com origem, destino e peso.
- **vertices**: Vetor contendo todos os vértices do grafo.
- **origem**: Vértice inicial para o cálculo do caminho mínimo.
- **destino**: Vértice final para o cálculo do caminho mínimo.
- **adjacencia**: Mapa que representa o grafo com listas de adjacência e pesos das arestas.
- **distancia**: Mapa que armazena a menor distância conhecida de cada vértice a partir da origem.
- **filaPrioridade**: Fila de prioridade (min-heap) usada para explorar os vértices na ordem da menor distância conhecida.

Função calcularCaminhoMinimo

Propósito:

A função calcularCaminhoMinimo usa o algoritmo de Dijkstra para calcular e exibir o caminho mínimo entre o primeiro e o último vértice em uma lista de vértices de um grafo.

Entrada:

- arestas: Lista de arestas do grafo, cada uma com origem, destino e peso.
- vertices: Lista de todos os vértices do grafo.

Saída:

- Imprime a distância mínima entre o primeiro vértice (origem) e o último vértice (destino). Se não houver caminho, imprime -1.

Conexão com Grafos:
Esta função utiliza o algoritmo de Dijkstra, que é um método fundamental para encontrar o caminho mais curto entre dois vértices em um grafo ponderado.

Explicação das Variáveis e Chamadas

- **arestas**: Vetor contendo as arestas do grafo, cada uma com origem, destino e peso.
- **vertices**: Vetor contendo todos os vértices do grafo.
- **origem**: O primeiro vértice na lista de vértices, definido como o ponto de partida para o cálculo do caminho.
- **destino**: O último vértice na lista de vértices, definido como o ponto de chegada para o cálculo do caminho.

Função bfsFluxoMaximo

Propósito:
A função bfsFluxoMaximo realiza uma busca em largura (BFS) para encontrar um caminho aumentante em um grafo de fluxo. Esse caminho é usado para maximizar o fluxo em redes de transporte, como em algoritmos de fluxo máximo.

Entrada:

- grafico: Mapa que representa o grafo, onde cada vértice mapeia seus vizinhos e as capacidades das arestas.
- origem: O vértice de origem do fluxo.
- destino: O vértice de destino do fluxo.
- predecessor: Mapa que armazena o predecessor de cada vértice no caminho encontrado.
- capacidadeResidual: Mapa que armazena a capacidade residual de cada aresta.

Saída:

- Retorna true se encontrar um caminho aumentante do origem ao destino; caso contrário, retorna false.

Conexão com Grafos:
Esta função é parte essencial do algoritmo de fluxo máximo, como o algoritmo de Ford-Fulkerson, e é utilizada para encontrar caminhos em grafos de fluxo onde ainda há capacidade disponível para adicionar mais fluxo.

Explicação das Variáveis e Chamadas

- **grafico**: Representa o grafo de fluxo, mapeando cada vértice para seus vizinhos e capacidades.
- **origem**: Vértice inicial do fluxo.
- **destino**: Vértice final do fluxo.
- **predecessor**: Mapa usado para reconstruir o caminho aumentante encontrado.
- **capacidadeResidual**: Mapa que armazena a capacidade residual das arestas após cada fluxo ser adicionado.

Função fluxoMaximo

Propósito:

A função fluxoMaximo calcula o fluxo máximo em uma rede de fluxo usando o algoritmo de Ford-Fulkerson. Esse algoritmo usa o método de busca em largura (BFS) para encontrar caminhos aumentantes e ajustar as capacidades residuais até que não haja mais caminhos aumentantes disponíveis.

Entrada:

- arestas: Vetor contendo as arestas do grafo, cada uma com origem, destino e peso.
- vertices: Vetor contendo todos os vértices do grafo.

Saída:

- Retorna o valor do fluxo máximo possível do vértice de origem até o vértice de destino.

Conexão com Grafos:

A função é aplicada em grafos de fluxo para determinar o máximo fluxo que pode ser transportado de um vértice fonte (origem) para um vértice de sumidouro (destino), o que é um problema clássico em teoria de grafos e otimização.

Explicação das Variáveis e Chamadas

- **grafico**: Mapa que representa o grafo com as capacidades das arestas.
- **capacidadeResidual**: Mapa que representa as capacidades residuais das arestas, que são atualizadas ao longo do algoritmo.
- **origem**: Vértice inicial do fluxo (primeiro vértice da lista).
- **destino**: Vértice final do fluxo (último vértice da lista).
- **fluxoMaximo**: Armazena o valor total do fluxo máximo encontrado.
- **predecessor**: Mapa usado para reconstruir o caminho aumentante.

Função entradaDados

Propósito:
A função entradaDados coleta informações do usuário para configurar um grafo e, em seguida, executa uma série de operações com base nas opções fornecidas. Ela lida com a entrada dos dados do grafo, como o número de vértices, arestas e tipo de grafo, e também processa comandos específicos para análise e manipulação do grafo.

Entrada:

- Os dados do grafo (número de vértices e arestas, tipo de grafo, e detalhes das arestas) são lidos do console.
- As opções de comando são lidas e processadas.

Saída:

- Executa e imprime o resultado das operações com base nas opções selecionadas, como verificar conexidade, calcular componentes, encontrar caminhos, etc.

Conexão com Grafos:
Esta função é um ponto central para a interação do usuário com o sistema de análise de grafos, permitindo a execução de várias operações e análises em grafos direcionados ou não direcionados.

Explicação das Variáveis e Chamadas

- **input**: Armazena a linha de entrada do usuário para a leitura de dados.
- **ss**: Utilizado para separar a entrada em números e opções.
- **options**: Vetor que armazena as opções fornecidas pelo usuário para execução.
- **option**: Armazena cada opção individualmente.
- **numVertices**: Número de vértices do grafo.
- **numArestas**: Número de arestas do grafo.
- **tipoGrafo**: Tipo do grafo (direcionado ou não direcionado).
- **direcionado**: Booleano que indica se o grafo é direcionado.
- **arestas**: Vetor que armazena as arestas do grafo.
- **verticesSet**: Conjunto para garantir que não haja vértices duplicados.
- **vertices**: Vetor final de vértices, convertendo o conjunto de vértices.
- **predecessor**: Mapa utilizado nas funções de fluxo máximo e outros algoritmos que precisam reconstruir o caminho.

Função boasVindas

Propósito:
A função boasVindas exibe uma mensagem de boas-vindas para os usuários do programa, apresentando informações sobre o trabalho e os autores.

Entrada:

- Nenhuma entrada é necessária para esta função.

Saída:

- Exibe uma mensagem de boas-vindas no console com informações sobre o trabalho e os autores.

Conexão com Grafos:
Embora não esteja diretamente relacionada à análise de grafos, a função boasVindas serve como uma introdução ao programa de análise de grafos, preparando o usuário para a interação com o software.

Função menu

Propósito:
A função menu exibe um menu de opções para o usuário, permitindo a seleção de diferentes análises e operações sobre grafos.

Entrada:

- Nenhuma entrada é necessária para esta função.

Saída:

- Imprime no console uma lista de opções disponíveis para o usuário escolher.

Conexão com Grafos:
O menu oferece várias funcionalidades relacionadas a grafos, como verificação de propriedades, listagem de características e configurações específicas, todas relevantes para a análise e manipulação de grafos.

Explicação das Variáveis e Chamadas

- **cout**: Usado para exibir o menu no console, formatando as opções e suas descrições para facilitar a leitura.