Grafos

Definições, implementação e algoritmos

Prof. Marcelo de Souza
marcelo.desouza@udesc.br

Algoritmos e Estruturas de Dados
Bacharelado em Engenharia de Software
Universidade do Estado de Santa Catarina

Material de consulta

Leitura obrigatória:

- Capítulo 1 de Goldbarg and Goldbarg (2012) – Conceitos básicos.
- Capítulo 3 de Kleinberg and Tardos (2006) – Grafos.

Leitura complementar:

- Capítulo 14 de Goodrich et al. (2014) – Algoritmos em grafos.
- Capítulo 15 de Preiss (2001) – Grafos e algoritmos em grafos.

Definições básicas

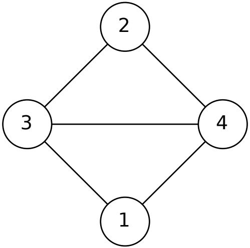
O que é um grafo?

- Grafos modelam relacionamentos entre pares de objetos.
- Notação: $G = (V, E)$
  - V é o conjunto de vértices/nodos (objetos).
  - E é o conjunto de arestas/arcos (relacionamentos).
    - Uma aresta possui dois vértices terminais: $e = \{u, v\}$ com $u, v \in V$.
    - Os vértices u e v são ditos vizinhos.
  - Tamanho do grafo: $n = |V|$ e $m = |E|$.

Definições básicas

O que é um grafo?

- Grafos modelam relacionamentos entre pares de objetos.
- Notação: $G = (V, E)$
  - V é o conjunto de vértices/nodos (objetos).
  - E é o conjunto de arestas/arcos (relacionamentos).
    - Uma aresta possui dois vértices terminais: $e = \{u, v\}$ com $u, v \in V$.
    - Os vértices u e v são ditos vizinhos.
  - Tamanho do grafo: $n = |V|$ e $m = |E|$.
- Representação visual:



- $V = \{1, 2, 3, 4\}$
- $E = \{\{1, 3\}, \{1, 4\}, \{2, 3\}, \{2, 4\}, \{3, 4\}\}$
- $n = 4$ e $m = 5$

Definições básicas

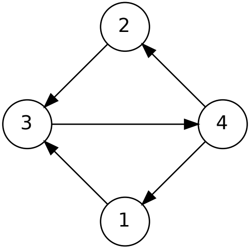
Grafos dirigidos

- Grafos podem conter direção nas suas arestas.
- Neste caso, chamamos de **grafo dirigido** ou **grafo direcionado**.
- Abreviamos como digrafo (do inglês *digraph* – *directed graph*).
- Cada aresta $e \in E$ é um par direcionado $e = (u, v)$.
    - $e = (u, v) \neq e' = (v, u)$.
    - **Convenção:** *arestas* são não-direcionadas e *arcos* são direcionados.

Definições básicas

Grafos dirigidos

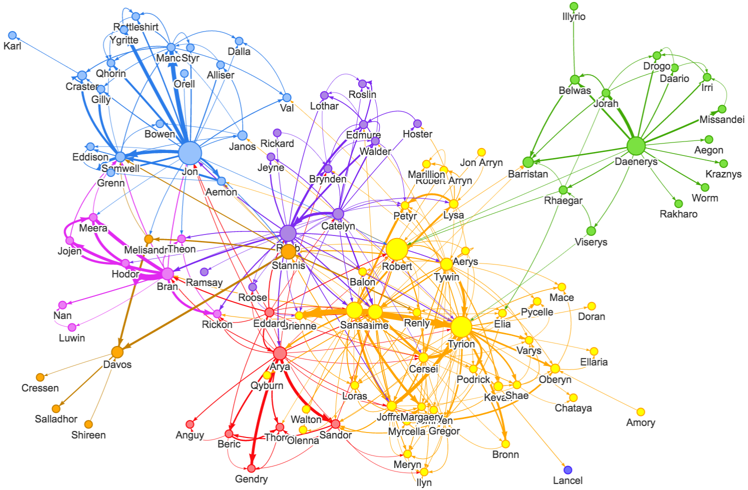
- Grafos podem conter direção nas suas arestas.
- Neste caso, chamamos de **grafo dirigido** ou **grafo direcionado**.
- Abreviamos como digrafo (do inglês *digraph* – *directed graph*).
- Cada aresta $e \in E$ é um par direcionado $e = (u, v)$.
  - $e = (u, v) \neq e' = (v, u)$.
  - **Convenção:** *arestas* são não-direcionadas e *arcos* são direcionados.
- Representação visual:



- $V = \{1, 2, 3, 4\}$
- $E = \{(1, 3), (2, 3), (3, 4), (4, 1), (4, 2)\}$
- $n = 4$ e $m = 5$

Definições básicas

Exemplo – Rede social de *Game of Thrones*



Definições básicas

Outros exemplos

Grafo	Nodos	Arestas
Comunicação	computador	cabo de fibra ótica
Circuito eletrônico	componentes	fio
Transporte	interceção de ruas	vias
Linhas aéreas	aeroportos	rotas aéreas
Jogo	posições em um tabuleiro	movimentos possíveis
Internet	páginas	links
Dependências	cursos	pré-requisitos

Representação de grafos

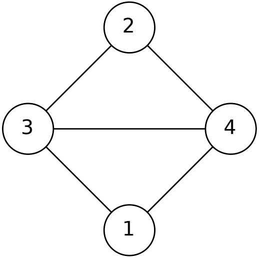
Matriz de adjacências

- Matriz $n \times n$ em que $A_{uv} = 1$ se (u, v) é uma aresta.
- Complexidade de espaço: $\Theta(n^2)$.
- Checar se (u, v) é uma aresta: $\Theta(1)$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(n)$.
- Recuperar todas as arestas: $\Theta(n^2)$.

Representação de grafos

Matriz de adjacências

- Matriz $n \times n$ em que $A_{uv} = 1$ se (u, v) é uma aresta.
- Complexidade de espaço: $\Theta(n^2)$.
- Checar se (u, v) é uma aresta: $\Theta(1)$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(n)$.
- Recuperar todas as arestas: $\Theta(n^2)$.

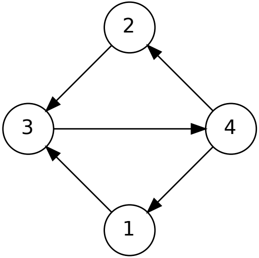


	1	**2**	**3**	**4**
1	0	0	1	1
2	0	0	1	1
3	1	1	0	1
4	1	1	1	0

Representação de grafos

Matriz de adjacências

- Matriz $n \times n$ em que $A_{uv} = 1$ se (u, v) é uma aresta.
- Complexidade de espaço: $\Theta(n^2)$.
- Checar se (u, v) é uma aresta: $\Theta(1)$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(n)$.
- Recuperar todas as arestas: $\Theta(n^2)$.



	1	**2**	**3**	**4**
1	0	0	1	0
2	0	0	1	0
3	0	0	0	1
4	1	1	0	0

Representação de grafos

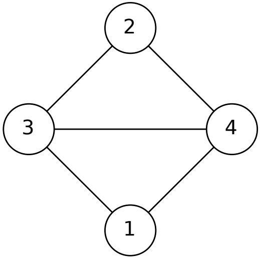
Listas de adjacências

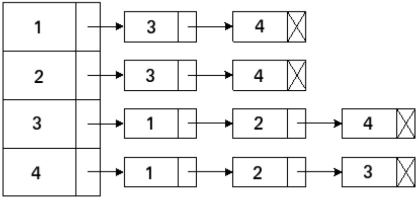
- Vetor indexado pelos nodos com listas de vizinhos.
- Complexidade de espaço: $\Theta(m+n)$.
- Checar se (u,v) é uma aresta: $\Theta(grau(u))$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(grau(v))$.
  - O grau de um vértice é o número de vizinhos que ele possui.
- Recuperar todas as arestas: $\Theta(m+n)$.

Representação de grafos

Listas de adjacências

- Vetor indexado pelos nodos com listas de vizinhos.
- Complexidade de espaço: $\Theta(m+n)$.
- Checar se (u,v) é uma aresta: $\Theta(grau(u))$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(grau(v))$.
  - O grau de um vértice é o número de vizinhos que ele possui.
- Recuperar todas as arestas: $\Theta(m+n)$.

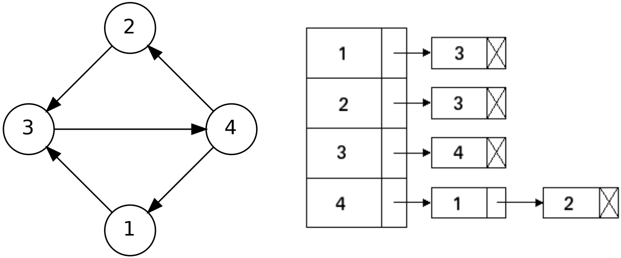




Representação de grafos

Listas de adjacências

- Vetor indexado pelos nodos com listas de vizinhos.
- Complexidade de espaço: $\Theta(m + n)$.
- Checar se (u, v) é uma aresta: $\Theta(grau(u))$.
- Recuperar todos os vizinhos de um vértice v: $\Theta(grau(v))$.
  - O grau de um vértice é o número de vizinhos que ele possui.
- Recuperar todas as arestas: $\Theta(m + n)$.



Representação de grafos

Implementação

- A lista de adjacências é uma melhor representação para grafos.
- Qual estrutura básica utilizar? **Vetores** ou **listas encadeadas**?
- Vetores: são ideais para implementação de grafos.
  - Permitem o acesso aleatório à estrutura.
  - Não demandam tempo adicional para caminhamento na estrutura.
- Listas: são ideais para estruturas dinâmicas.
  - Permitem a inserção e remoção de elementos em tempo constante.
  - Não demandam tempo adicional para modificações na estrutura.
- Dica geral:
  - Para grafos estáticos ou com pouca variação: **vetores**.
  - Para grafos dinâmicos e com muita variação: **encadeamento**.

Exercício

Implementação de grafos

1. Implemente uma classe grafo para modelar essa estrutura de dados. Utilize a representação por listas de adjacências e implemente versões utilizando vetores e listas encadeadas. Utilize as classes criadas para armazenar grafos de um domínio específico à sua escolha. Implemente rotinas para calcular o grau de um vértice, sua lista de vizinhos e o tamanho do grafo.

Caminhos e conectividade de grafos

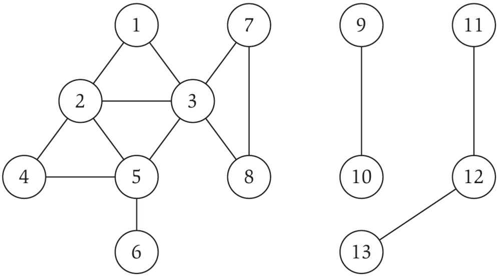
- Um **caminho** em um grafo não direcionado $G = (V, E)$ é uma sequência de vértices $v_1, v_2, \ldots, v_k$, onde cada par consecutivo v_{i-1}, v_i está ligado por uma aresta em E.
  - O mesmo conceito se aplica a grafos direcionados (caminho direcionado).
- Um caminho é **simples** se todos os seus vértices são diferentes.

Caminhos e conectividade de grafos

- Um **caminho** em um grafo não direcionado $G = (V, E)$ é uma sequência de vértices $v_1, v_2, \ldots, v_k$, onde cada par consecutivo v_{i-1}, v_i está ligado por uma aresta em E.
  - O mesmo conceito se aplica a grafos direcionados (caminho direcionado).
- Um caminho é **simples** se todos os seus vértices são diferentes.
- Um grafo é **conectado** (ou conexo) se para todo par de vértices u e v, existe um caminho conectando u a v.
  - Um grafo direcionado é **fortemente conectado** se para todo par de vértices u e v, existe caminhos conectando u a v e v a u.

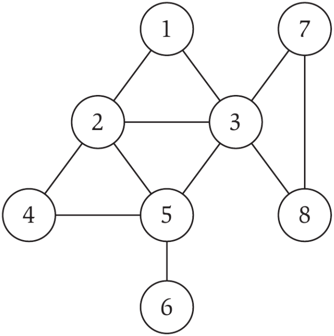
Caminhos e conectividade de grafos

- Um **caminho** em um grafo não direcionado $G = (V, E)$ é uma sequência de vértices $v_1, v_2, \ldots, v_k$, onde cada par consecutivo v_{i-1}, v_i está ligado por uma aresta em E.
  - O mesmo conceito se aplica a grafos direcionados (caminho direcionado).
- Um caminho é **simples** se todos os seus vértices são diferentes.
- Um grafo é **conectado** (ou conexo) se para todo par de vértices u e v, existe um caminho conectando u a v.
  - Um grafo direcionado é **fortemente conectado** se para todo par de vértices u e v, existe caminhos conectando u a v e v a u.



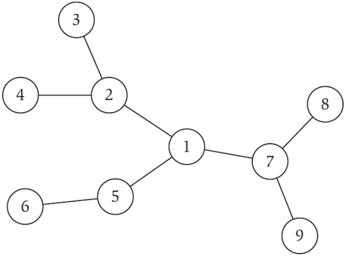
Ciclos em grafos

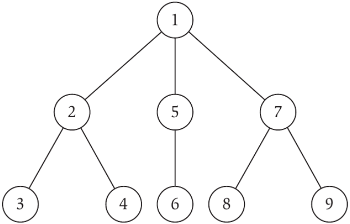
- Um **ciclo** é um caminho $v_1, v_2, \ldots, v_k$, em que $v_1 = v_k$, $k > 2$ e todos os $k - 1$ primeiros vértices são distintos.
  - O caminho $1, 2, 4, 5, 3, 1$ é um ciclo no grafo abaixo.
  - O caminho $3, 7, 8, 3$ é um ciclo no grafo abaixo.



Árvores

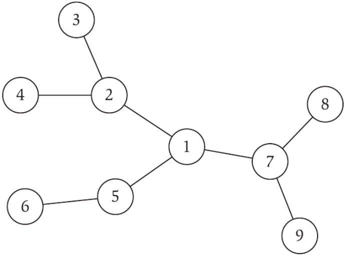
- Um grafo é uma **árvore** se ele é conectado e não contém ciclos.
  - Ou seja, uma árvore é um grafo conectado sem ciclos.
- Qualquer vértice pode ser escolhido como raiz da árvore para uma organização gráfica comum de árvores.

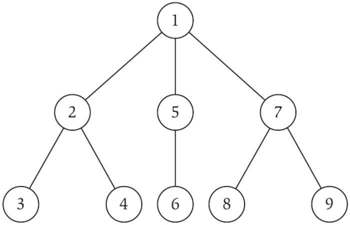




Árvores

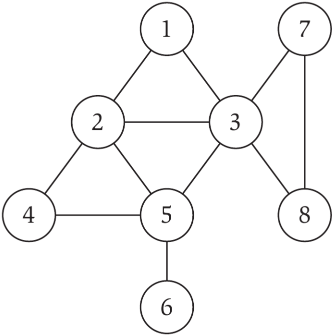
- Teorema: seja G um grafo não-direcionado com n vértices. Quaisquer duas afirmações verdadeiras implica na terceira.
  1. G é conectado.
  2. G não contém ciclos.
  3. G possui $n - 1$ arestas.





Buscas – percorrendo um grafo

- Busca: visitação sistemática dos vértices do grafo.
  - Buscar um vértice determinado.
  - Procedimento para resolução de outros problemas relacionados.
- Estratégias básicas (ordem de visitação):
  - Busca em largura (*breadth-first search* – BFS).
  - Busca em profundidade (*depth-first search* – DFS).
- Dado um vértice inicial, explora o grafo.



Busca em largura

Estratégia

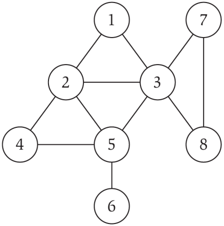
- Na **busca em largura**, o grafo é explorado por camadas a partir do vértice de início.
    1. Visita o vértice atual.
    2. Inclui seus vizinhos ainda não visitados e na lista para exploração.
    3. Seleciona o vértice de menor camada e vai para o passo 1.

- Seja L_i o conjunto de vértices da camada i.
- Estes vértices estão a uma distância mínima de i do vértice de início.
- Todo vértice em L_{i-1} é visitado antes dos vértices em L_i.
- Só existe um caminho de u a v se e somente se v estiver em uma das camadas da busca a partir de u (e vice-versa).

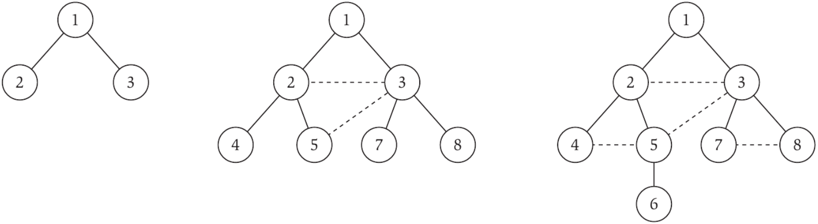
Busca em largura

Funcionamento

- Seja o grafo abaixo.



- Execução da busca em largura (camadas de visitação):



Busca em profundidade

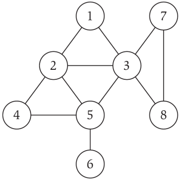
Estratégia

- Na **busca em profundidade**, o grafo é explorado indo o mais profundo possível nas camadas, antes de explorar vértices da mesma camada.
  1. Visita o vértice atual.
  2. Inclui seus vizinhos ainda não visitados e na lista para exploração.
  3. Seleciona o vértice de **maior** camada e vai para o passo 1.
- O algoritmo caminha para mais longe na busca, antes de visitar vizinhos mais próximos.
- Só existe um caminho de u a v se e somente se v estiver em uma das camadas da busca a partir de u (e vice-versa).

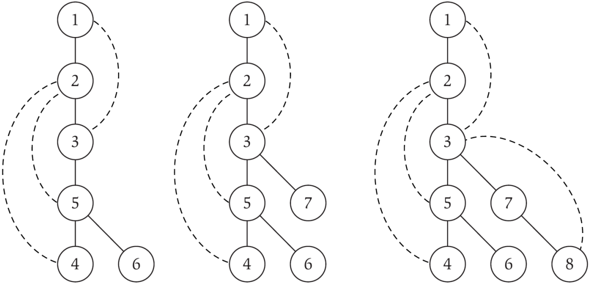
Busca em profundidade

Funcionamento

- Seja o grafo abaixo.



- Execução da busca em profundidade (ordem de visitação):



Estratégia básica de busca

Detalhes de implementação

Algorithm: `search(Vertex s)`

$A \leftarrow \{s\}$
while $A \neq \emptyset$ **do**
    $v \leftarrow$ extract vertex from A
    Visit vertex v
    Let $N(v)$ the set of unvisited neighbors of v
    $A \leftarrow A \cup N(v)$

Estratégia básica de busca

Detalhes de implementação

Algorithm: `search(Vertex s)`

$A \leftarrow \{s\}$
while $A \neq \emptyset$ **do**
    $v \leftarrow$ extract vertex from A
    Visit vertex v
    Let $N(v)$ the set of unvisited neighbors of v
    $A \leftarrow A \cup N(v)$

- Estruturas de dados básicas:
  - Para implementar a busca em largura, A é uma **fila**.
  - Para implementar uma busca em profundidade, A é uma **pilha**.
  - Para outras estratégias, A pode ser uma **fila de prioridades**.

Estratégia básica de busca

Detalhes de implementação

Algorithm: `search(Vertex s)`

$A \leftarrow \{s\}$
while $A \neq \emptyset$ **do**
  $v \leftarrow$ extract vertex from A
  Visit vertex v
  Let $N(v)$ the set of unvisited neighbors of v
  $A \leftarrow A \cup N(v)$

- Complexidade:
  - A recuperação de vizinhos não visitados está limitada ao grau do nodo.
  - O somatório dos graus de cada nodo é $2m$.
  - Logo, a complexidade do laço é $O(m)$.
  - É necessário um esforço inicial, para criar um vetor `discovered` para armazenar a visitação de cada vértice, demandando $O(n)$.
  - Logo, as buscas estudadas executam em tempo $O(m + n)$.

Buscas em grafos

Aplicações

- Encontrar um caminho entre um par de vértices.
- Detectar ciclos em um grafo.
- Resolver jogos de quebra-cabeça.
- Descobrir links entre páginas em um buscador.
- Encontrar amigos próximos em uma rede social.
- Broadcast em redes.

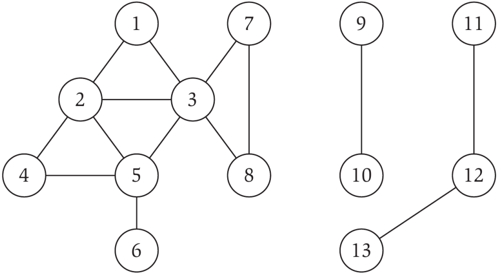
Exercício

Buscas em grafos

1. Implemente os algoritmos de busca em largura e profundidade para o grafo criado no exercício anterior. Considere que o grafo é conectado e utilize o algoritmo para as seguintes tarefas:
   1.1 Encontrar um determinado vértice no grafo.
   1.2 Retornar o menor caminho entre um par de vértices.
   1.3 Imprimir toda a estrutura do grafo.

Componente conexo

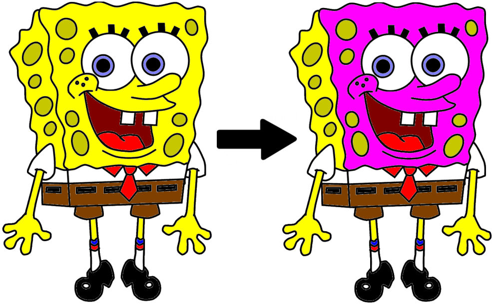
- Dizemos que C é o **componente conexo** de um vértice s, se e somente se, todos os vértices alcançáveis a partir de s estão em C.
- Ou seja, é o conjunto de vértices que estão conectados a s.
- Exemplos:
  - O componente conexo de 12 é $\{11, 12, 13\}$.
  - O componente conexo de 5 é $\{1, 2, 3, 4, 5, 6, 7, 8\}$.



Componente conexo

Aplicações

- Flood fill: um desenho é representado por um grafo de píxels. Ao clicar em um píxel para pintura, toda a área da mesma cor é pintada. Ou seja, todo o componente conexo de uma determinada cor.



Componente conexo

Algoritmo

Algorithm: `connected-component(Vertex s)`

Let C be the set of nodes to which s has a path
$C \leftarrow \{s\}$
while *there is an edge* (u, v) *where* $u \in C$ *and* $v \notin C$ **do**
    Add v to C

- Uma boa estratégia é utilizar as buscas estudadas para obter o componente conexo de um vértice inicial.
  - Busca largura/profundidade definirão a ordem de inserção dos vértices.
- Propriedade: se um nodo v está conectado a s, ele estará no conjunto C produzido pelo algoritmo.
  - As buscas exploram todos os vértices alcançáveis a partir do vértice inicial.

Encontrando todos os componentes conexos

- Executar a busca (largura, por exemplo) repetidamente:
  1. Executa a busca a partir do vértice s, obtendo seu componente conexo.
  2. Repete o passo anterior, iniciando por um vértice não marcado.
- Ao final, o algoritmo descobre todos os componentes conexos.

Encontrando todos os componentes conexos

- Executar a busca (largura, por exemplo) repetidamente:
  1. Executa a busca a partir do vértice s, obtendo seu componente conexo.
  2. Repete o passo anterior, iniciando por um vértice não marcado.
- Ao final, o algoritmo descobre todos os componentes conexos.
- **Complexidade:**
  - O algoritmo de busca executa em tempo $O(m+n)$.
  - Porém, ele se limita ao componente conexo em questão.
  - Na soma de todos os componentes conexos, o algoritmo demanda tempo total de $O(m+n)$.

Encontrando todos os componentes conexos

- Executar a busca (largura, por exemplo) repetidamente:
  1. Executa a busca a partir do vértice s, obtendo seu componente conexo.
  2. Repete o passo anterior, iniciando por um vértice não marcado.
- Ao final, o algoritmo descobre todos os componentes conexos.
- **Complexidade:**
  - O algoritmo de busca executa em tempo $O(m + n)$.
  - Porém, ele se limita ao componente conexo em questão.
  - Na soma de todos os componentes conexos, o algoritmo demanda tempo total de $O(m + n)$.
- Como podemos adaptar esse algoritmo para obter o maior componente conexo?

Exercício

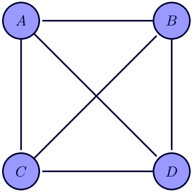
Componente conexo

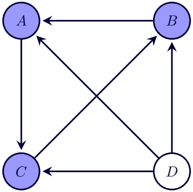
1. Crie um grafo para modelar a rede interna de telefonia de uma empresa. Os departamentos da empresa são os vértices e as arestas são as conexões cabeadas existentes na rede de telefonia. Neste sentido, os departamentos da empresa possuem ligações entre si. Esse grafo deve ser desconectado, representando grupos de departamentos que estão isolados na rede interna. Dado um grafo dessa natureza, crie um algoritmo que receba um departamento e retorne seu componente conexo. Crie também um algoritmo para determinar o maior componente conexo da empresa.

Operações em grafos dirigidos

Buscas

- Em **grafos não-dirigidos** uma busca é capaz de encontrar o componente conexo a partir de um vértice s.
- Em **grafos dirigidos** uma busca encontra apenas o conjunto de vértices para os quais s tem um caminho.
- Exemplos:



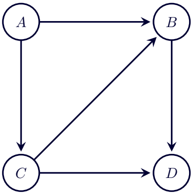


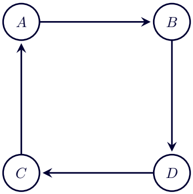
- No primeiro grafo, todos os vértices são visitados pela busca a partir de A. No segundo grafo, apenas aqueles com caminho desde A.

Operações em grafos dirigidos

Conexidade

- Em **grafos não-dirigidos**, o grafo é conexo se entre todo par de vértices existe um caminho.
- Em **grafos dirigidos**:
  - Fracamente conexo: entre todo par de vértices existe uma ligação.
    - Dica: ignorando as direções dos arcos, verifica se o grafo não-direcionado subjacente é conexo.
  - Fortemente conexo: entre todo par de vértices $\{u, v\}$ existe um caminho de u para v e um caminho de v para u.
    - Ou seja, os vértices tem que ser mutuamente alcançáveis.
- Exemplos – (1) fracamente conexo e (2) fortemente conexo:





Operações em grafos dirigidos

Conexidade

- Descobrir se G é fracamente conexo:
  - Desconsidera a direção dos arcos (grafo não-direcionado subjacente) e aplica uma busca em largura.
  - Complexidade $O(m + n)$.
- Descobrir se G é fortemente conexo:
  1. Aplica uma busca em largura a partir de s, verificando se s pode atingir todos os demais vértices.
  2. Considera o grafo reverso G^{rev}, onde a direção de cada arco é invertida e aplica uma busca em largura a partir de s, verificando os vértices que podem atingir s.
  3. Caso s atinja todos, e todos atinjam s, o grafo é fortemente conexo, caso contrário não.

  - Argumento da prova: se todos os vértices chegam a s e s chega a todos, é garantido que existe caminho entre qualquer par de vértices $\{u, v\}$. Sai de u até s e de s até v.
  - Complexidade $O(m + n)$.

Exercício

Conexidade em grafos dirigidos

1. Crie um programa que recebe uma descrição de um grafo direcionado (seus vértices e arcos) e retorne as seguintes informações sobre ele:
    - Número de vértices e de arcos.
    - Grau de cada vértice.
    - Se o grafo é fracamente conexo.
    - Se o grafo é fortemente conexo.

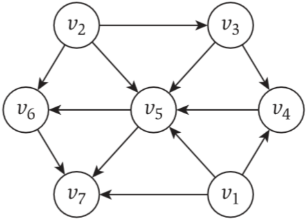
Grafos do tipo DAG

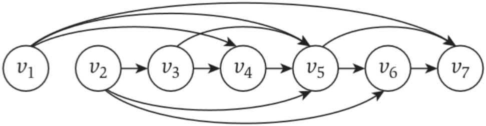
- **DAG:** *Directed Acyclic Graph*.
  - Grafo direcionado acíclico (que não contém ciclos).
- Exemplo: relações de precedência.
  - Processos em uma linha de produção.
  - Matriz de pré-requisitos de disciplinas.

$$
\begin{tikzcd}
 & v_2 \arrow[rr] \arrow[dl] \arrow[dr] & & v_3 \arrow[dl] \arrow[dr] & \\
v_6 \arrow[dr] & & v_5 \arrow[ll] \arrow[dl] & & v_4 \arrow[ll] \\
 & v_7 & & v_1 \arrow[ll] \arrow[ul] \arrow[ur] &
\end{tikzcd}
$$

Ordenação topológica

- **Problema:** dado um DAG, encontra uma ordem válida de processamento dos vértices, respeitando as precedências.
- **Ordem topológica:** uma ordenação dos vértices $v_1, v_2, \ldots, v_n$ tal que para todo arco (v_i, v_j), $i < j$.
  - Ou seja, todos os arcos apontam para frente na ordenação.
  - Exemplos: qual a ordem para cursar as disciplinas?





Ordenação topológica
Propriedades

1. G possui uma ordenação topológica se e somente se G é um DAG.
2. Em todo DAG G, existe um vértice v sem arcos incidentes.

Ordenação topológica

Propriedades

1. G possui uma ordenação topológica se e somente se G é um DAG.
2. Em todo DAG G, existe um vértice v sem arcos incidentes.

Detalhes

- Com relação à propriedade (1), existindo um ciclo, não é possível ordenar topologicamente seus vértices. Não existindo ciclo, é possível definir um primeiro vértice para a ordenação topológica e repetir o processo para os vértices restantes.
- Com relação à propriedade (2), caso todos os vértices tenham ao menos um arco incidente, necessariamente o grafo possui ao menos um ciclo e, consequentemente, não é um DAG.

Exercício: tente criar um DAGs que não respeitem as condições acima.

Ordenação topológica

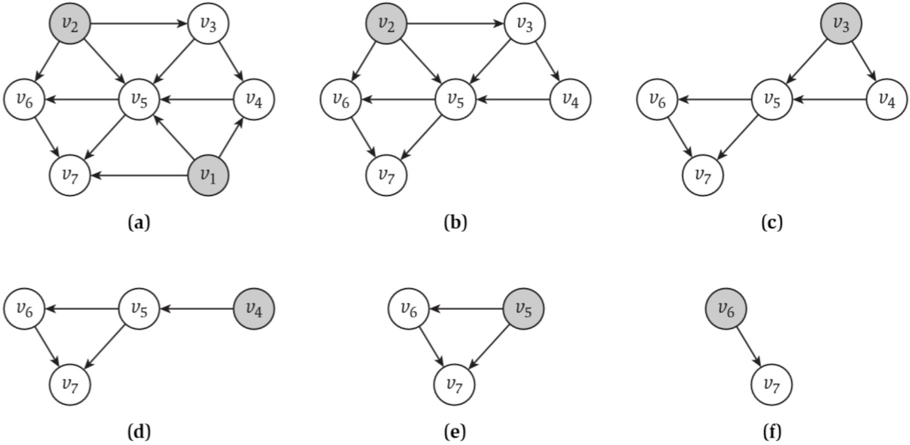
Algoritmo

Algorithm: `topological-order(DAG G)`

Find a vertex v with no incoming arcs and order it first
Delete v from G
Recursively compute a topological ordering of $G \setminus \{v\}$
Append the result after v and return the topological order.

Ordenação topológica

Funcionamento



Ordenação topológica

Complexidade

- Encontrar o vértice v demanda $O(n)$.
- O algoritmo executa n iterações, totalizando $O(n^2)$.
- Para melhorar, fazemos a busca mais eficiente. Definimos um vértice ativo se ele ainda não foi deletados e mantemos:
  1. Número de arcos incidentes para cada vértice a partir de vértices ativos.
  2. Vértices que não possuem arcos incidentes a partir de vértices ativos.
- No início, todos os vértices são ativos e a criação de (1) e (2) é feita com uma passagem sobre os vértices.
- A cada deleção do vértice v, percorremos todos os vértices w para os quais v possui arco, decrementando o valor em (1). Caso atinja zero, o vértice é incluído em (2).
- Logo, determinar um vértice sem arcos incidentes é feito em tempo constante.
- **Complexidade final:** $O(n + m)$.

Exercício

Ordenação topológica

1. Crie um grafo com as disciplinas da matriz curricular do curso de Engenharia de Software, com as devidas relações de pré-requisitos. Implemente o algoritmo para determinação de ordenação topológica e calcule uma possível ordenação para cursar as disciplinas.

Referências

Goldbarg, M. and Goldbarg, E. (2012). *Grafos: Conceitos, algoritmos e aplicações*. Elsevier.

Goodrich, M. T., Tamassia, R., and Goldwasser, M. H. (2014). *Data structures and algorithms in Java*. John Wiley & Sons, 6th edition.

Kleinberg, J. and Tardos, É. (2006). *Algorithm Design*. Pearson Education India.

Preiss, B. R. (2001). *Estruturas de dados e algoritmos: padrões de projetos orientados a objetos com Java*. Campus.