Análise e Técnicas de Algoritmos
Período 2002.2

Aula 12: Algoritmos de Grafos

24/02/2003

Algoritmos de Grafos

Neste capítulo vamos discutir alguns problemas comuns da teoria dos grafos. Muitos dos problemas da teoria dos grafos podem ser utilizados na prática.

Definições

- Um grafo $G = (V, E)$ consiste de um conjunto de *vértices*, V, e um conjunto de *arestas*, E. Cada aresta é um par (v, w), onde $v, w \in V$. Arestas são também chamadas de *arcos*.
- Se o par que define uma aresta é ordenado, então o grafo é dito *dirigido*. Grafos dirigidos são também conhecidos como *digrafos*.
- Um vértice w é *adjacente* a v se e somente se $(v, w) \in E$. Em um grafo não dirigido com arco (v, w), w é adjacente a v e v é adjacente a w.
- Um arco pode ter um terceiro componente que corresponde a um *peso* ou *custo*. Um grafo cujos arcos possuem este elemento é dito *grafo ponderado*.
- Um *caminho* em um grafo é uma seqüência de vértices $w_1, w_2, ..., w_n$, em que $(w_i, w_{i+1}) \in E, \forall 1 \leq i < n$.
- O *tamanho* de um caminho é o número de arcos no caminho que é igual a $n - 1$.
- É possível ter um caminho que parte de um vértice para ele mesmo. Se o caminho não contém nenhum arco, corresponde a um caminho de tamanho 0.
- Um caminho é dito *simples* se todos os seus vértices são distintos, com exceção de primeiro que pode ser igual ao último.
- Um *ciclo* em um grafo dirigido é um caminho de tamanho pelo menos 1, onde $w_1 = w_n$. O ciclo é simples se o caminho é simples.
- No caso de um grafo não dirigido, os arcos devem ser distintos, ou seja, o caminho u, v, u não é considerado um ciclo pois (u, v) e (v, u) representam o mesmo arco.
- Um grafo dirigido é *acíclico* se ele não contém ciclos. Um grafo acíclico dirigido pode ser referenciado como *DAG*.
- Um grafo não dirigido é dito ser *conectado* se existe um caminho entre quaisquer dois vértices.
- Um grafo dirigido com essa propriedade é dito *fortemente conectado*. Se um grafo dirigido não é fortemente conectado e o correspondente grafo não dirigido é conectado, dizemos que o grafo dirigido é *fracamente conectado*.

- Um grafo *completo* é aquele onde existe um arco entre qualquer par de vértices.

Exemplo:

Um exemplo de uma situação real que pode ser modelada por um grafo é o sistema de aeroportos.

- Cada aeroporto é um vértice.
- Dois vértices são conectados por um arco se existir um vôo sem parada entre os aeroportos representados pelos vértices.
- O arco pode possuir um peso, representando tempo, distância, ou custo do vôo.
- Poderíamos considerar o grafo como sendo um grafo dirigido (os pesos dos arcos poderiam ser diferentes em cada sentido, por exemplo).
- Talvez fosse interessante fazer com que o grafo fosse fortemente conectado. Logo, seria possível voar de um aeroporto para qualquer um outro.

Representação de Grafos

Existem duas formas de representar um grafo $G = (V, E)$:

1. Coleção de listas de adjacência.
2. Matriz de adjacência.

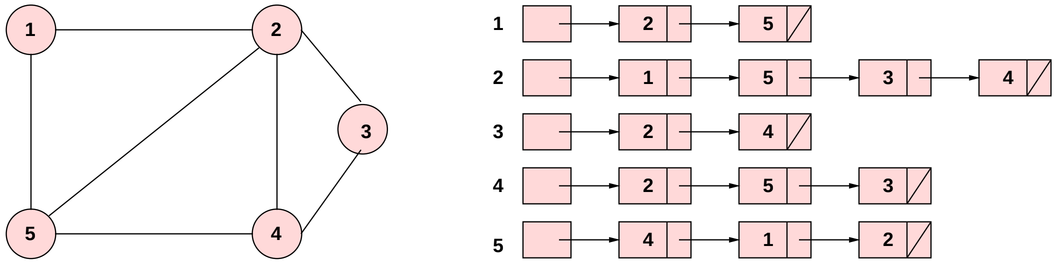
A representação com listas de adjacência é mais utilizada poque provê uma forma compacta de representar *grafos esparsos* , isto é, grafos onde $|E|$ é bem menos do que $|V|^2$.

A representação com matriz de adjacência é indicada no caso de *grafos densos*, onde $|E|$ é perto de $|V|^2$.

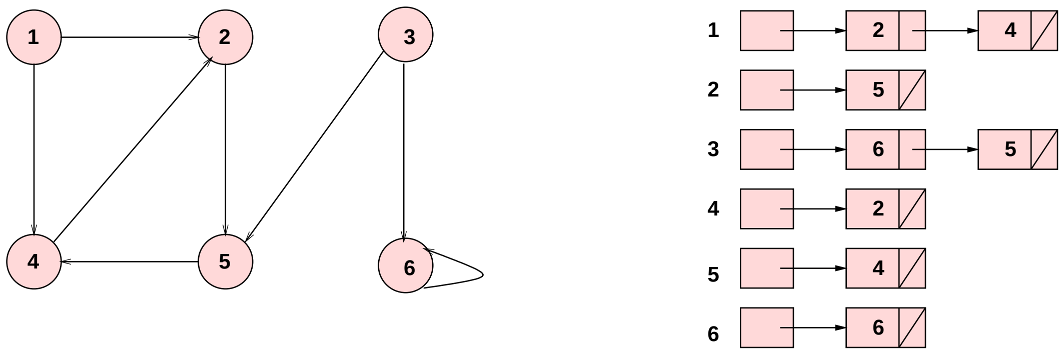
Listas de Adjacência

A representação com listas de adjacência de um grafo $G = (V, E)$ consiste de um array Adj de $|V|$ listas, um para cada vértice de V. Para cada $u \in V$, a lista de adjacência $Adj[u]$ contém um ponteiro para todos os vértices v, onde existe um arco $(u, v) \in E$. Logo, $Adj[u]$ consiste de todos os vértices de G que são adjacentes a u. Esses vértices são armazenados em ordem arbitrária.

A figura abaixo mostra um grafo não dirigido e sua representação com listas de adjacência.



Se G é um grafo dirigido, a soma dos tamanhos de todas as listas de adjacências é $|E|$. A figura abaixo mostra um grafo dirigido e sua representação com listas de adjacência.



Em ambos os casos, grafos dirigidos ou não, a representação com listas de adjacÊncia possui uma propriedade desejável que indica que a quantidade de memória requerida é $O(max(V, E)) = O(V + E)$.

As listas de adjacência também pode ser utilizadas no caso de grafos ponderados e outras variantes de grafos.

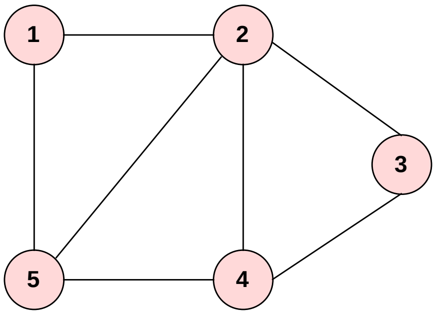
A maior desvantagem desse método de representação é a de não possuir uma forma eficiente de dizer se um determinado arco (u, v) está presente no grafo.

Matriz de Adjacência

A representação por matriz de adjacência de um grafo $G = (V, E)$ requer que os vértices sejam arbitrariamente numerados de $1, 2, ..., |V|$. A matriz de adjacência de um grafo de uma matriz $A = (a_{ij})$, de ordem $|V| \times |V|$, onde

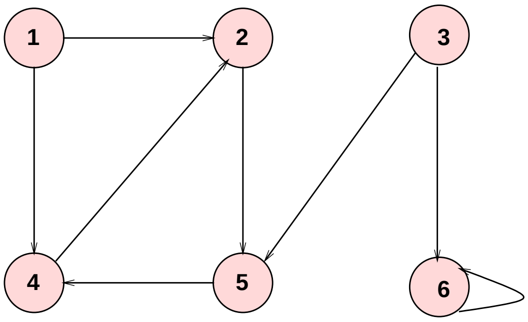
$$a_{ij} = \begin{cases} 1 & \text{se } (i,j) \in E \\ 0 & \text{caso contrário} \end{cases}$$

A figura abaixo mostra um grafo não dirigido e sua representação com matriz de adjacência. Esse tipo de representação requer $O(V^2)$ de memória, independentemente do número de arcos do grafo.



	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	1	1
3	0	1	0	1	0
4	0	1	1	0	1
5	1	1	0	1	0

Da mesma forma, a matriz de adjacência pode ser usada para representar grafos dirigidos.



	1	2	3	4	5	6
1	0	1	0	1	0	0
2	0	0	0	0	1	0
3	0	0	0	0	1	1
4	0	1	0	0	0	0
5	0	0	0	1	0	0
6	0	0	0	0	0	1

Como na representação com listas de adjacência, a matriz de adjacência pode ser usada para representar grafos ponderados. Em vez de usar 1 para indicar a presença do arco, utiliza-se o peso do arco.

Embora a representação com lista de adjacência seja pelo menos tão eficiente quanto a representação com matriz de adjacência, a simplicidade de uma matriz de adjacência a torna preferível no caso de grafos pequenos. No caso de grafos não ponderados, a representação com matriz de adjacência tem a vantagem de requerer apenas um bit por entrada.

Pesquisa em Grafos

Existem dois tipos principais de pesquisa (caminhamento) em grafos:

1. Busca em amplitude (breadth-first search).

2. Busca em profundidade (Depth-first search).

Busca em Amplitude

A busca em amplitude representa um dos mais simples algoritmos de busca em grafo. Apesar de sua simplicidade, esse método funciona como modelo para vários importantes algoritmos de grafos como, por exemplo, o algoritmode caminho mais curto de Dijkstra e o algoritmo da árvore mínima de cobertura de Prim.

Se um grafo $G = (V, E)$ e um vértice inicial s. A busca em amplitude processa os vértices por níveis, começando pelos vértices mais proóximos do vértice inicial s e deixando os vértices mais distantes para depois. Esse tipo de busca é bastante similar ao caminhamento por níveis em árvores.

O algoritmo de busca em amplitude pode ser resumido nos seguintes passos e funciona tanto para grafos dirigidos ou não:

- Distinguir o vértice inicial s.
- Sistematicamente explorar os arcos de G para descobrir todo vértice alcançáveis a partir de s.
- Computar a distância (menor número de arcos) de s para todos os vértices alcançáveis.
- Produz uma *árvore de amplitude* cuja raiz é s e contém todos os vértices alcançáveis.
- Para todo vértice v alcançável a partir de s, o caminho na árvore de amplitude corresponde ao menor caminho de s para v em G.

O algoritmo, portanto, descobre todos os vértices com distância k a partir de s antes de descobrir os vértics com distância $k + 1$.

Para manter controle do processamento dos vértices no algoritmo, é necessário marcar os vértices. Utilizamos um sistema de 3 cores: branca, azul e vermelha.

- Todos os vértices começam com cor branca e podem mudar para cor azul e posteriormente para vermelho.
- Quando um vértice é encontrado pela primeira vez, ele passa a ser colorido.
- É necessário duas cores diferentes do branco para garantir a sistemática da amplitude.
- Se $(u, v) \in E$ e u é vermelho, então o vértice v ou é azul ou vermelho, ou seja, todo vértices adjacente a um vértice vermelho já descoberto.
- Um vértice azul pode ter um vértice adjacente de cor branca.

Como dito anteriormente, a busca em amplitude constroi uma árvore de amplitude, inicialmente contendo apenas a raiz que é o nó de origem. Sempre que um vértice branco v é descoberto a partir de um vértice já descoberto u, ele é adicionado à árvore, juntamente com o arco (u, v). O vértice u é o predecessor de v na árvore. Como cada vértice só é descoberto uma única vez, ele possui apenas um predecessor.

O algoritmo **BFS** abaixo corresponde ao algoritmo de busca por amplitude e assume que o grafo $G = (V, E)$ é representado com listas de adjacência. Para cada vértice no grafo, o algoritmo mantém estruturas auxiliares:

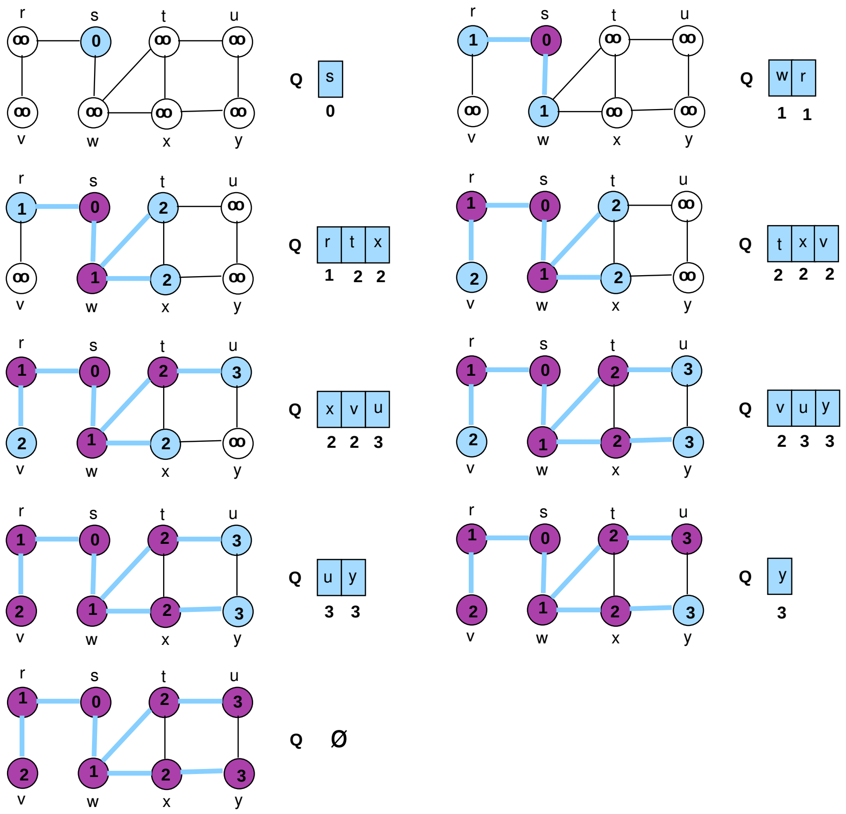
- A variável $cor[u]$ mantém a informação sobre a cor de cada vértice.
- A variável $\pi[u]$ mantém a informação sobre o predecessor de cada vértice. Quando nã existe predecessor $\pi[u] = NIL$.
- $d[u]$ mantém o valor da distância do vértice inicial até u.
- Uma fila Q com política FIFO para gerenciar a lista de vértices de cor azul.

BFS(G, s)

for $\forall u \in V[G] - \{s\}$ **do**
  $cor[u] \leftarrow BRANCO$
  $d[u] \leftarrow \propto$
  $\pi[u] \leftarrow NIL$
$cor[s] \leftarrow AZUL$
$d[s] \leftarrow 0$
$\pi[s] \leftarrow NIL$
$Q \leftarrow \{s\}$
while $Q \neq \emptyset$ **do**
  $u \leftarrow head[Q]$
  for $\forall v \in Adj[u]$ **do**
    if $cor[v] = BRANCO$ **then**
      $cor[v] \leftarrow AZUL$
      $d[v] \leftarrow d[v] + 1$
      $\pi[v] \leftarrow u$
      Enfileira(Q, v)
  Desenfileira(Q)
  $cor[u] = VERMELHO$

Exemplo

O exemplo abaixo mostra a utilização do algoritmo **BFS** em um grafo não dirigido.



Busca em Profundidade

A busca em profundidade é uma generalização do caminhamento em pré-ordem. Como o próprio nome indica, a idéia principal é buscar *verticalmente* sempre que possível. Nesta estratégia, sempre que um novo vértice v é descoberto, ele deve ser explorado por completo. Quando não existe mais nenhum vértice a ser explorado, efetua-se um bactrack para o vértice que propiciou a descoberta de v.

Alguns detalhes do algoritmo de busca em profundidade:

- Da mesma forma que na busca em amplitude, sempre que um vértice v é descoberto, a busca em profundidade também guarda a informação do vértice predecessor.
- Na busca em profundidade, uma floresta de árvores composta pelos vértices predecessores é formada. Isso porque a busca pode ser repetida a partir de origens múltiplas.
- Além de criar a floresta de árvores, o algoritmo de busca em profundidade associa uma marca de tempo em cada vértice. Cada vértice v tem duas marcas de tempo $d[v]$ e $f[v]$ que indicam, respectivamente, quando

v foi primeiramente descoberto (colorido com azul) e quando a lista de adjacência para v foi totalmente examinada (e colorida com vermelho).

- Essas marcas de tempo são utilizadas em alguns algoritmos de grafos e ajudam a verificar o comportamento da busca em profundidade.

DFS(G)

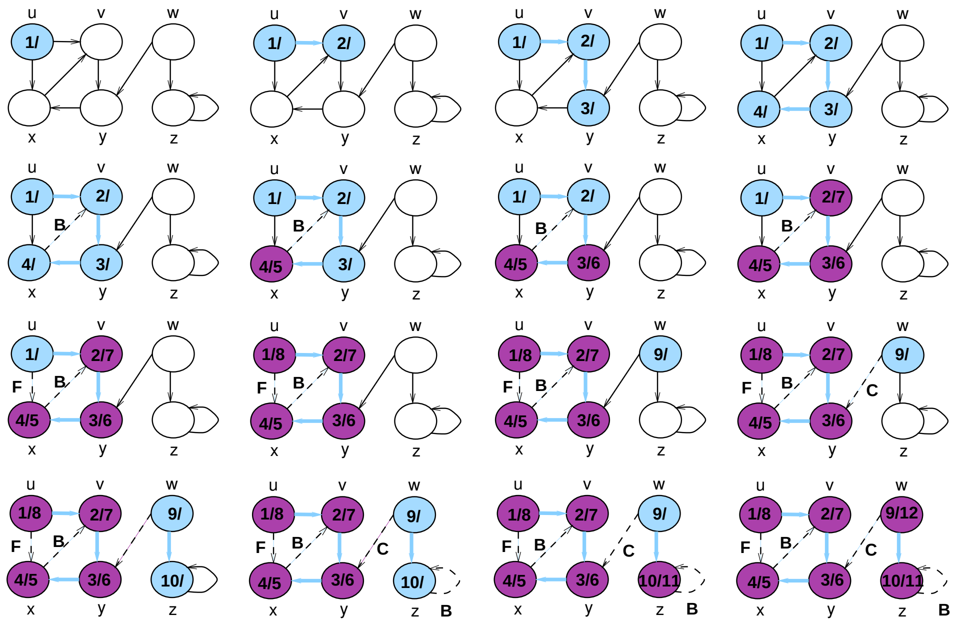
for $\forall u \in V[G]$ **do**
  $cor[u] \leftarrow BRANCO$
  $\pi[u] \leftarrow NIL$
$tempo \leftarrow 0$
for $\forall u \in V[G]$ **do**
  if $cor[u] = BRANCO$ **then**
    VisitaDFS(u)

VisitaDFS(u)

$cor[u] \leftarrow AZUL$
$d[u] \leftarrow tempo \leftarrow tempo + 1$
for $\forall v \in Adj[u]$ **do**
  if $cor[v] = BRANCO$ **then**
    $\pi[v] \leftarrow u$
    VisitaDFS(v)
$cor[u] \leftarrow VERMELHO$
$f[u] \leftarrow tempo \leftarrow tempo + 1$

Exemplo

O exemplo a seguir, mostra o progresso da execução do algoritmo **DFS** sobre um grafo dirigido.



O algoritmo de busca em profundidade pode ser facilmente modificado para classificar os tipos de arcos do grafo G. Podemos distinguir 4 tipos de arcos:

1. Arcos da árvore: arcos que fazem parte da floresta de árvores.
2. Arcos de retorno: arcos que conectam um vértice a um ancestral em uma árvore de profundidade.
3. Arcos para frente: arcos que não fazem parte da árvore que conectam um vértice a um descendente.
4. Arcos de cruzamento: todos os outros arcos.

Essa classificação pode servir para indicar certos aspectos do grafo. Por exemplo, podemos afirmar que um grafo é acíclico se ele não possuir arcos de retorno.

Ordenação Topológica

A ordenação topológica pode ser vista como uma aplicação da técnica de busca em profundidade, que consiste em ordenar o conjunto de vértices de um grafo acíclico dirigido.

Podemos definir uma ordenação topológica de um grafo acíclico dirigido $G = (V, E)$ como uma ordenação linear de todos os vértices que satisfaz a seguinte condição:

Se G contém um arco (u, v), então u aparece antes de v na ordenação.

O seguinte algoritmo ordena topologicamente um grafo acíclico dirigido.

TopologicalSort(G)

Chamar **DFS(G)** para computar os tempos finais de todos os vértices.
Sempre que um vértice se tornar vermelho, inseri-lo no início de uma lista encadeada
Retornar a lista encadeada de vértices.

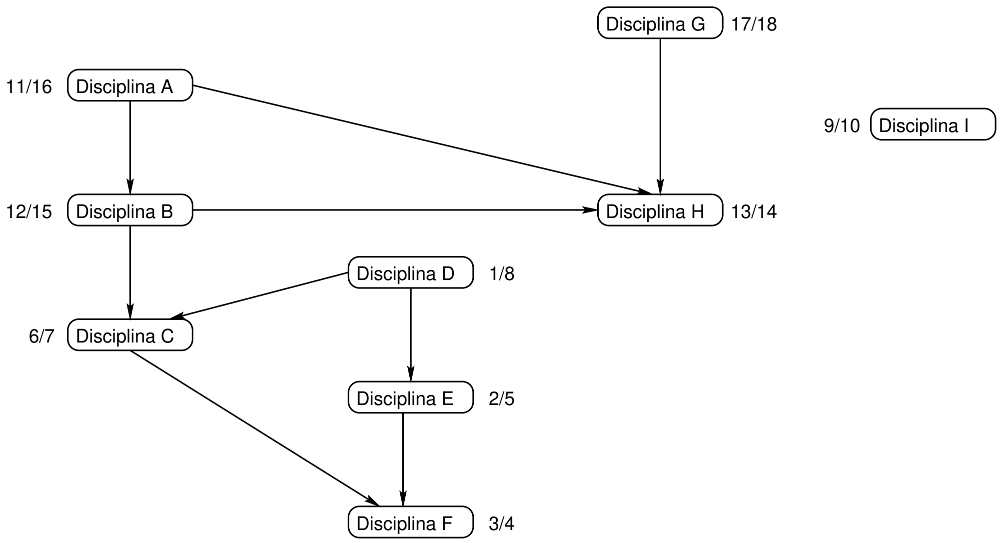
Esse algoritmo é $O(V+E)$ pois, o algoritmo **DFS** é $O(V+E)$ e a inserção na lista encadeada é $O(1)$.

Exemplo

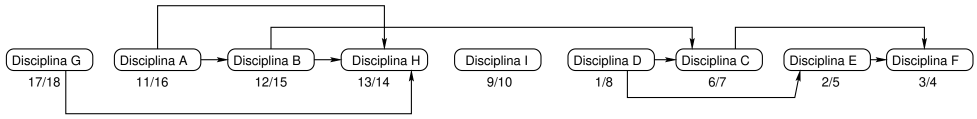
Grafos acíclicos dirigidos são usados em muitas aplicações para indicar precedência entre eventos. Por exemplo, a estrutura de pré-requisitos das disciplinas do Curso de Ciência da Computação na UFPB.

A ordenação topológica das disciplinas seria qualquer seqüência de disciplinas que não violasse a estrutura de pré-requisitos.

Para ilustrar a aplicação apresentada, vamos considerar a figura a seguir. Ela apresenta uma estrutura de pré-requisitos entre 9 disciplinas de um determinado curso. O tempo de inicial e final dos vértices, obtidos através do algoritmo de busca em profundidade estão também indicados.



Uma possível solução para a ordenação topológica do grafo está mostrada a seguir.


Disciplina G
17/18
Disciplina A
11/16
Disciplina B
12/15
Disciplina H
13/14
Disciplina I
9/10
Disciplina D
1/8
Disciplina C
6/7
Disciplina E
2/5
Disciplina F
3/4