Aula 18 – Busca em Largura e em Profundidade

Notas de Aula de Teoria dos Grafos

Prof[a.]: Patrícia D. L. Machado

UFCG – Unidade Acadêmica de Sistemas e Computação

Sumário

Nesta aula, estudaremos dois algoritmos de busca em árvore: a busca em largura e a busca em profundidade. Ambos retornam uma árvore geradora a partir de um vértice inicial, mas as árvores diferem quanto as propriedades básicas que apresentam. Esta diferença decorre principalmente da estratégia utilizada para visitação dos vértices.

Busca em Largura

Considere o seguinte problema de roteamento em uma rede modelada pelo grafo na Figura 1. Suponha que um sistema rodando na máquina A precisa enviar uma mensagem para um outro sistema em cada uma das outras máquinas.Como o grafo que representa esta rede não é completo, é importante encontrar o ***menor caminho*** (de menor tamanho) entre A e todas as outras máquinas a fim de diminuir o tráfego na rede. Como podemos resolver este problema? Em princípio, para ser possível enviar mensagens para todos, é necessário que o grafo seja ***conectado***!

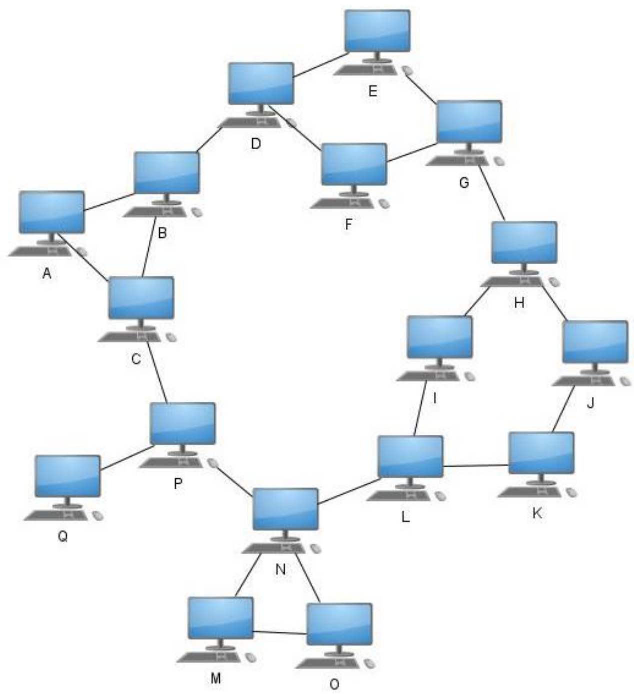


Figura 1

Este problema pode ser resolvida através a construção de uma árvore de busca em largura a partir do vértice A.

Busca em Largura (*Breadth-First Search* – BFS) considera todos os vizinhos de um vértice de acordo com uma ordem de incorporação destes vértices.

Partindo da raiz, **explora todos os nós vizinhos**. Para cada um desses nós vizinhos, explora os seus nós vizinhos inexplorados e assim por diante, até que o alvo da busca seja atingido.

Vértices são mantidos em uma **fila** *Q* que contém todos os vértices a partir dos quais a árvore pode crescer. Fila é estrutura de dados onde o primeiro elemento que entra é o primeiro que sai. A cabeça de uma fila é o primeiro elemento da fila.

Inicialmente a fila é vazia. Quando um novo vértice é adicionado a árvore, este é adicionado à fila. Em cada estágio, a lista de vértices adjacentes ao vértice na cabeça de Q é examinada para que um vizinho seja adicionado à árvore. Se todos os vizinhos já estão na árvore, então o vértice é removido e passamos a considerar os vizinhos do vértice que agora ocupa a cabeça da fila. O algoritmo termina quando *Q* fica vazia.

O algoritmo, abaixo ilustrado, recebe como entrada um grafo conectado *G* e um vértice *r*, o vértice a partir do qual a busca irá iniciar. A saída é uma *r*-árvore em *G* representada como uma função predecessor *p*, uma função nível l, tal que $l(v) \; = \; d_G(r, v)$ (distância entre *r* e *v*), e uma função de tempo ***t.***

1	**Faça** $i := 0$ **e** $Q := \varnothing$
2	$i := i + 1$
3	**Pinte** r
4	$l(r) := 0$ e $t(r) := i$
5	**Adicione** r **a** Q
6	**Enquanto** Q **for não vazia faça**
7	**Considere a cabeça** x **de** Q
8	**Se** x **tem algum vizinho** y **não colorido**
9	$i := i + 1$
10	**Pinte** y
11	$p(y) := x, l(y) = l(x) + 1$ **e** $t(y) = i$
12	**Adicione** y **a** Q
13	**Caso contrário**
14	**Remova** x **de** Q
15	**Retorne** (p, l, t)

Para realizar sua tarefa, o algoritmo utiliza 5 variáveis:

- Q, uma fila que define a ordem de vértices para os quais seus vizinhos serão visitados. Inicialmente, $Q = \varnothing$(Linha 1), mas, em seguida o vértice r é adicionado a Q (Linha 5);
- l, a função nível. Inicialmente, $l(r) = 0$ (Linha 4);
- i, contador para o tempo em que os vértices são adicionados a árvore. Inicialmente $i := 0$ (Linha 1), mas passa a ser 1 (Linha 2) antes da adição do vértice *r* a árvore (Linha 3).
- t, a função tempo que indicará o tempo em que cada vértice foi adicionado a árvore. Inicialmente, $t(r) := 1$ (Linha 4);
- p, função predecessor que armazenará os arcos que definem a árvore. Inicialmente, a função é indefinida para todos os vértices, já que *r* não tem predecessor.

Das linhas 6 a 14, enquanto a fila Q não for vazia (Linha 6), o algoritmo visitará cada um dos vértices, iniciando por *r* que está na cabeça da fila (foi adicionado na Linha 5).

Considerando um vértice x que está na cabeça da fila (Linha 7), um dos vizinhos de x ainda não colorido será escolhido para ser adicionado a árvore. Note que sempre que um vértice for adicionado a árvore, este será "pintado" (Linhas 3 e 10). Em princípio, qualquer vizinho não colorido y pode ser escolhido, mas em uma implementação concreta precisaremos definir como esta escolha será feita porque ela poderá influenciar na árvore resultante (randômica, seguindo alguma ordenação/heurística). Independente da ordem, o resultado do algoritmo será uma árvore BFS, mas há mais de uma possibilidade e a ordem de escolha poderá influenciar sobre qual BFS será produzida. Caso exista algum y, então as Linhas 9 a 12 serão executadas para adicionar y a árvore:

- o contador i é incrementado (Linha 9)
- o vértice y é pintado (Linha 10)

- o predecessor de y é definido com sendo x (porque foi visitado a partir de x), o nível de y é definido como o nível de x + 1 e o tempo de inclusão de y na árvore é definido como i (Linha 11)
- o vértice y é adicionado a fila Q (Linha 12)

Caso não exista um y, então o vértice x é removido de Q, uma vez que todos os seus vizinhos já foram visitados (Linha 14).

Ao final, o algoritmo retorna a funções p, l e t (Linha 15).

Vamos considerar um exemplo de aplicação deste algoritmo, considerando o grafo da Figura 2. Seja **a** o vértice inicial.

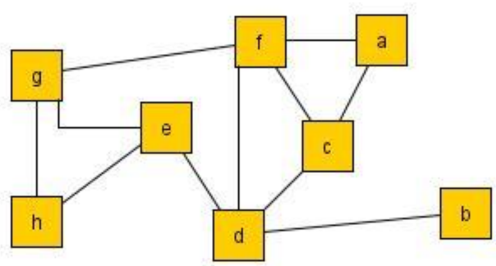


Figura 2

Abaixo, temos os valores das variáveis do algoritmo após a execução das Linhas 1 a 5.

Q	a
l	$a \rightarrow 0$
i	1
t	$a \rightarrow 1$
p	

Vamos considerar a primeira iteração do bloco de repetição das Linhas 6 a 14. A fila Q possui o vértica **a** na cabeça. Os vizinhos de **a** são **f** e **c**, ambos ainda não pintados (visitados). Vamos escolher y como sendo o vértice **f**. O resultado está ilustrado na tabela abaixo.

Q	a, f
l	$a \rightarrow 0, f \rightarrow 1$
i	2
t	$a \rightarrow 1, f \rightarrow 2$
p	$f \rightarrow a$

Na segunda iteração, teremos apenas o vértice **c** como opção de vizinho de a ainda não visitado. O resultado está ilustrado na tabela abaixo.

Q	a, f, c
l	$a \rightarrow 0, f \rightarrow 1, c \rightarrow 1$
i	3
t	$a \rightarrow 1, f \rightarrow 2, c \rightarrow 3$
p	$f \rightarrow a, c \rightarrow a$

Na terceira iteração, o vértice a que ainda está na cabeça da fila Q não possui mais vizinhos não visitados. Assim, ele será removido da Fila. O resultado está ilustrado na tabela abaixo.

Q	f, c
l	$a \rightarrow 0, f \rightarrow 1, c \rightarrow 1$
i	3
t	$a \rightarrow 1, f \rightarrow 2, c \rightarrow 3$
p	$f \rightarrow a, c \rightarrow a$

Na quarta iteração, são considerados os vizinhos não visitados de **f** que é o vértice na cabeça da fila. São eles: **g** e **d**. Considerando que foram escolhidos nesta ordem, o resultado da quarta, quinta e sexta iterações (onde **f** será removido da fila) está ilustrado abaixo.

Q	c, g, d
l	$a \rightarrow 0, f \rightarrow 1, c \rightarrow 1, g \rightarrow 2, d \rightarrow 2$
i	5
t	$a \rightarrow 1, f \rightarrow 2, c \rightarrow 3, g \rightarrow 4, d \rightarrow 5$
p	$f \rightarrow a, c \rightarrow a, g \rightarrow f, d \rightarrow f$

Na sétima iteração, são considerados os vizinhos não visitados de **c** que é o vértice na cabeça da fila. Mas todos os seus vizinhos já foram visitados. Então c é removido da fila. O resultado está ilustrado abaixo.

Q	g, d
l	$a \rightarrow 0, f \rightarrow 1, c \rightarrow 1, g \rightarrow 2, d \rightarrow 2$
i	5
t	$a \rightarrow 1, f \rightarrow 2, c \rightarrow 3, g \rightarrow 4, d \rightarrow 5$
p	$f \rightarrow a, c \rightarrow a, g \rightarrow f, d \rightarrow f$

Na oitava iteração, são considerados os vizinhos não visitados de **g** que é o vértice na cabeça da fila. São eles: **h** e **e**. Considerando que foram escolhidos nesta ordem, o resultado da oitava, nona e décima iterações (onde **g** será removido da fila) está ilustrado abaixo.

Q	$\mathrm{d, h, e}$
l	$\mathrm{a \to 0, f \to 1, c \to 1, g \to 2, d \to 2, h \to 3, e \to 3}$
i	7
t	$\mathrm{a \to 1, f \to 2, c \to 3, g \to 4, d \to 5, h \to 6, e \to 7}$
p	$\mathrm{f \to a, c \to a, g \to f, d \to f, h \to g, e \to g}$

Na undécima iteração, são considerados os vizinhos não visitados de **d** que é o vértice na cabeça da fila. Temos apenas o **b**. Assim o resultado da undécima e duodécima iterações (onde **d** será removido da fila) está ilustrado abaixo.

Q	$\mathrm{h, e}$
l	$\mathrm{a \to 0, f \to 1, c \to 1, g \to 2, d \to 2, h \to 3, e \to 3, b \to 3}$
i	8
t	$\mathrm{a \to 1, f \to 2, c \to 3, g \to 4, d \to 5, h \to 6, e \to 7, b \to 8}$
p	$\mathrm{f \to a, c \to a, g \to f, d \to f, h \to g, e \to g, b \to d}$

Nas iterações finais, o vértice **h** e o vértice **e** são removidos da fila, porque não possuem vizinhos não visitados. Assim, o resultado final da execução do algoritmo está ilustrado na tabela abaixo. A Figura 3 apresenta a árvore BFS produzida. Note que os caminhos nesta árvore representam caminhos de menor tamanho (distância) de **a** para todos os vértices do grafo.

Q	
l	$\mathrm{a \to 0, f \to 1, c \to 1, g \to 2, d \to 2, h \to 3, e \to 3, b \to 3}$
i	8
t	$\mathrm{a \to 1, f \to 2, c \to 3, g \to 4, d \to 5, h \to 6, e \to 7, b \to 8}$
p	$\mathrm{f \to a, c \to a, g \to f, d \to f, h \to g, e \to g, b \to d}$

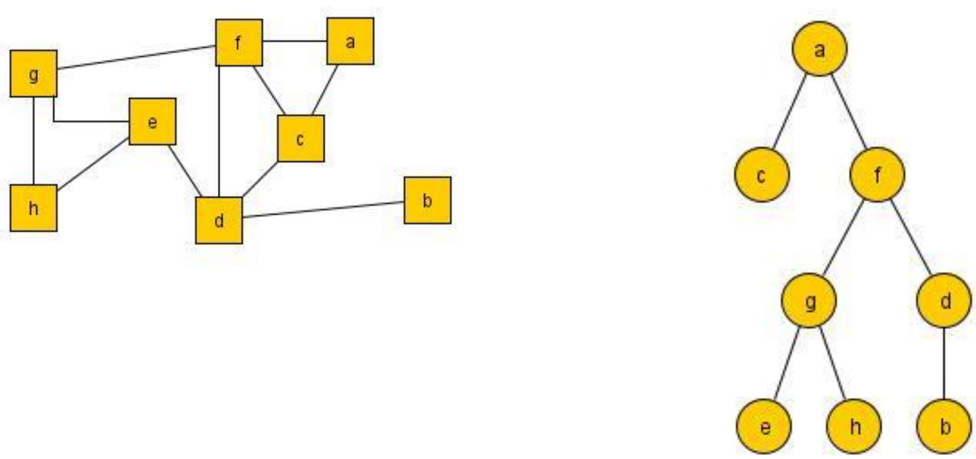


Figura 3

Teorema 1. Seja *T* uma *BFS*-árvore de um grafo conectado *G* com raiz *r*. Então:

a) Para todo vértice *v* de *G*, $l(v) = d_G(r, v)$, isto é, o nível de *v* em *T,* é a distância entre *r* e *v* em *G*

b) Toda aresta de *G* une vértices no mesmo nível ou em níveis consecutivos de *T:* $|l(u) - l(v)| \leq 1$

Podemos observar o que a cláusula a) estabelece na Figura 3. O nível de cada vértice *v* na árvore, corresponde a distância entre *v* e *r* no grafo original. Com relação a cláusula b), podemos observar que vértices vizinhos no grafo original estão:

- no mesmo nível, quando foram visitados como vizinhos de um mesmo vértice (como exemplo **c** e **f**)
- em um nível diferente, mas consecutivo, quando um foi visitado quando o outro estava na cabeça da fila (como exemplo **g** e **f**) ou quando foram visitados quando vértices de um mesmo nível estava na cabeça da fila (**h** e **b**)

Ou seja, dois vértices adjacentes estão em níveis diferentes quando o descendente foi visitado como sendo um vizinho do ancestral. Caso contrário, foram visitados por um ancestral em nível comum. Isto ocorre, devido ao fato de que o algoritmo sempre visita todos os vizinhos de um vizinho e todos os vizinhos destes vizinhos sucessivamente.

Teorema 2. Seja *G* um grafo conectado. Então os valores da função nível retornados pela *BFS* representam as distâncias em *G* a partir da raiz *r*.

Em outras palavras, independente da ordem de visitação dos vizinhos de um vértice, o algoritmo sempre retornará uma árvore BFS que atende a propriedade de representar as distâncias entre *r* e os demais vértices. Considere que no exemplo anterior, o vértice **c** fosse visitado antes de **f**. Poderíamos ter como resultado a árvore na Figura 4 abaixo que também é BFS.

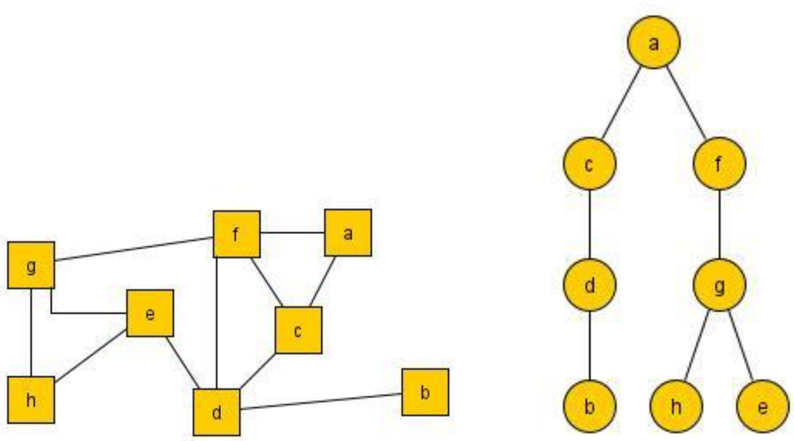


Figura 4

Revisitando o problema apresentado anteriormente de envio de mensagens da máquina A para as demais máquinas da rede considerando o caminho mais curto, usando o algoritmo visto podemos construir a árvore BFS ilustrada na Figura 5.

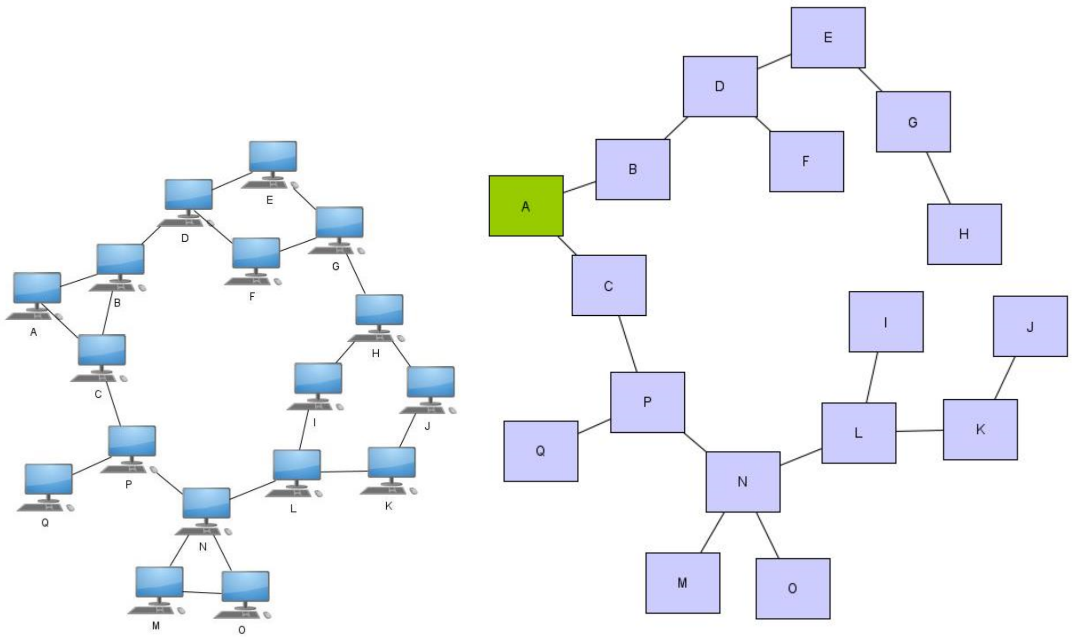


Figura 5

Busca em Profundidade

O algoritmo de busca em profundidade, assim como o algoritmo de busca em largura que estudamos anteriormente, tem como objetivo encontrar uma árvore geradora em um grafo conectado. A principal diferença entre os algoritmos é a ordem de visitação dos vértices. Ao invés de visitar todos os vizinhos de um vértice, o próximo vértice a visitar é sempre um vizinho do vértice que foi adicionado a árvore mais recentemente.

Desta forma, ao invés de usar uma fila, utilizamos uma **pilha** para armazenar os vértices que foram visitados (e que podem ter vizinhos ainda não visitados). Em uma pilha, o primeiro elemento inserido é o último a ser removido. O elemento no topo da pilha é o último elemento inserido (o próximo a ser removido).

Na busca em profundidade, o vértice adicionado a árvore *T* em cada estágio é o vizinho da adição mais recente a *T*. A partir do vértice *x* adicionado mais recentemente, procura-se um vizinho que ainda não está em *T*. Se existir, este é adicionado. Caso contrário, volta-se ao vértice adicionado antes de *x.*

O algoritmo, abaixo ilustrado, recebe como entrada um grafo conectado *G* e um vértice *r*, o vértice a partir do qual a busca irá iniciar. A saída é uma *r*-árvore em *G* representada como uma função predecessor *p* e duas funções tempo:

- $f(v)$ que indica o tempo em que v é incorporado a *T*
- $t(v)$ tempo que indica quando todos os vizinhos de v estão em *T* e o vértice é removido da pilha S.

A árvore resultante é chamada de DFS-árvore.

1	$i := 0\ e\ S := \emptyset$
2	$i := i + 1$
3	**Pinte** r
4	$f(r) := i$
5	**Adicione** r **a** S
6	**Enquanto** S **for não vazia faça**
7	**Considere o vértice** x **do topo de** S
8	$i := i + 1$
9	**Se** x **tem um vizinho** y **não colorido então**
10	**Pinte** y
11	$p(y) := x$ **e** $f(y) := i$
12	**Adicione** y **a** S
13	***C*aso contrário**
14	$t(x) := i$
15	**Remova** x **de** S
16	**Retorne** (p, f, t)

Para realizar sua tarefa o algoritmo utiliza 5 variáveis:

- S, uma pilha que define a ordem de vértices para os quais o próximo vizinho será visitado. Inicialmente, $S = \varnothing$(Linha 1), mas, em seguida o vértice r é adicionado a Q (Linha 5);
- i, contador para o tempo em que os vértices são adicionados a árvore e saem da pilha. Inicialmente $i := 0$ (Linha 1), mas passa a ser 1 (Linha 2) antes da adição do vértice *r* a árvore (Linha 3).
- f, a função tempo que indicará o tempo em que cada vértice foi adicionado a árvore. Inicialmente, $f(r) := 1$ (Linha 4);
- t, a função tempo que indicará o tempo em que cada vértice sai da pilha. Inicialmente, é indefinido para todos os vértices, já que nenhum vértice deixou a pilha ainda;
- p, função predecessor que armazenará os arcos que definem a árvore. Inicialmente, a função é indefinida para todos os vértices, já que *r* não tem predecessor.

Das linhas 6 a 15, enquanto a pilha S não for vazia (Linha 6), o algoritmo visitará cada um dos vértices, iniciando por *r* que está no topo da pilha (foi adicionado na Linha 5).

Considerando um vértice x que está no topo da pilha (Linha 7), um dos vizinhos de x ainda não colorido será escolhido para ser adicionado a árvore (Linha 9). Mas antes, o contador de tempo i é incrementado (Linha 8). Note que sempre que um vértice for adicionado a árvore, este será "pintado" (Linhas 3 e 10). Em princípio, qualquer vizinho não colorido y pode ser escolhido, mas em uma implementação concreta precisaremos definir como esta escolha será feita porque ela poderá influenciar na árvore resultante (randômica, seguindo alguma ordenação/heurística). Independente da ordem, o resultado do algoritmo será uma árvore DFS, mas há mais de uma possibilidade e a ordem de escolha poderá influenciar sobre qual DFS será produzida. Caso exista algum y, então as Linhas 10 a 12 serão executadas para adicionar y a árvore:

- o vértice y é pintado (Linha 10);
- o predecessor de y é definido com sendo x (porque foi visitado a partir de x) e o tempo de inclusão de y na árvore, função f, é definido como i (Linha 11);
- o vértice y é adicionado ao topo da pilha S (Linha 12).

Caso não exista um y, então as Linhas 14 e 15 serão executadas para remover o vértice x da pilha:

- o tempo de saída do vértice da pilha, função t, é definido como i (Linha 14);
- o vértice x que está no topo de S é removido (Linha 15).

Ao final, o algoritmo retorna a funções p, f e t (Linha 16).

Vamos considerar um exemplo de aplicação deste algoritmo, considerando o grafo da Figura 2. Seja **a** o vértice inicial.

Abaixo, temos os valores das variáveis do algoritmo após a execução das Linhas 1 a 5.

S	a
i	1
f	$a \rightarrow 1$
t	
p	

Vamos considerar a primeira iteração do bloco de repetição das Linhas 6 a 15. A pilha S possui o vértica **a** na cabeça. Os vizinhos de **a** são **f** e **c**, ambos ainda não pintados (visitados). Vamos escolher y como sendo o vértice **f**. O resultado está ilustrado na tabela abaixo.

S	f, a
i	2
f	$a \rightarrow 1, f \rightarrow 2$
t	
p	$f \rightarrow a$

Na segunda iteração, a pilha S possui o vértica **f** na cabeça. Os vizinhos de **f** ainda não visitados são **g** e **c**. Vamos escolher y como sendo o vértice **g**. O resultado está ilustrado na tabela abaixo.

S	g, f, a
i	3
f	$a \to 1, f \to 2, g \to 3$
t	
p	$f \to a, g \to f$

Na terceira iteração, a pilha S possui o vértica **g** na cabeça. Os vizinhos de **g** ainda não visitados são **e** e **h**. Vamos escolher y como sendo o vértice **e**. O resultado está ilustrado na tabela abaixo.

S	e, g, f, a
i	4
f	$a \to 1, f \to 2, g \to 3, e \to 4$
t	
p	$f \to a, g \to f, e \to g$

Na quarta e na quinta iteração, vamos escolher o vértice **d** (vizinho de **e**) e o vértice **b** (vizinho de **d**), respectivamente. O resultado está ilustrado na tabela abaixo.

S	b, d, e, g, f, a
i	6
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6$
t	
p	$f \to a, g \to f, e \to g, d \to e, b \to d$

Na sexta iteração, temos o vértice **b** no topo da pilha. Este vértice não possui vizinhos ainda não visitados. Assim, ele será removido da pilha e o seu tempo de remoção da pilha (t) será registrado. O resultado está ilustrado na tabela abaixo.

S	d, e, g, f, a
i	7
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6$
t	$b \to 7$
p	$f \to a, g \to f, e \to g, d \to e, b \to d$

Na sétima iteração, o vértice **d** no topo da pilha ainda possui um vizinho não visitado, o vértice **c**. Vamos escolher y como sendo o vértice **c**. O resultado está ilustrado na tabela abaixo.

S	c, d, e, g, f, a
i	8
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6, c \to 8$
t	$b \to 7$
p	$f \to a, g \to f, e \to g, d \to e, b \to d, c \to d$

Na oitava e nona iteração, os vértices c e d, respectivamente, no topo da pilha, não possuem vizinhos ainda não visitados. Assim, serão removidos da pilha. O resultado está ilustrado na tabela abaixo.

S	e, g, f, a
i	10
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6, c \to 8$
t	$b \to 7, c \to 9, d \to 10$
p	$f \to a, g \to f, e \to g, d \to e, b \to d, c \to d$

Na décima iteração, o vértice **e** no topo da pilha ainda possui um vizinho não visitado, o vértice **h**. Vamos escolher y como sendo o vértice **h**. O resultado está ilustrado na tabela abaixo.

S	h, e, g, f, a
i	11
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6, c \to 8, h \to 11$
t	$b \to 7, c \to 9, d \to 10$
p	$f \to a, g \to f, e \to g, d \to e, b \to d, c \to d, h \to e$

Nas iterações finais, os vértices no topo da pilha não possuem mais vizinhos não visitados. Assim, serão todos removidos da pilha. O resultado está ilustrado na tabela abaixo.

S	
i	16
f	$a \to 1, f \to 2, g \to 3, e \to 4, d \to 5, b \to 6, c \to 8, h \to 11$
t	$b \to 7, c \to 9, d \to 10, h \to 12, e \to 13, g \to 14, f \to 15, a \to 16$
p	$f \to a, g \to f, e \to g, d \to e, b \to d, c \to d, h \to e$

O algoritmo encerra sua execução na Linha 17 retornam as funções com os valores acima definidos. A árvore DFS produzida está ilustrada na Figura 6, juntamente com o grafo original.

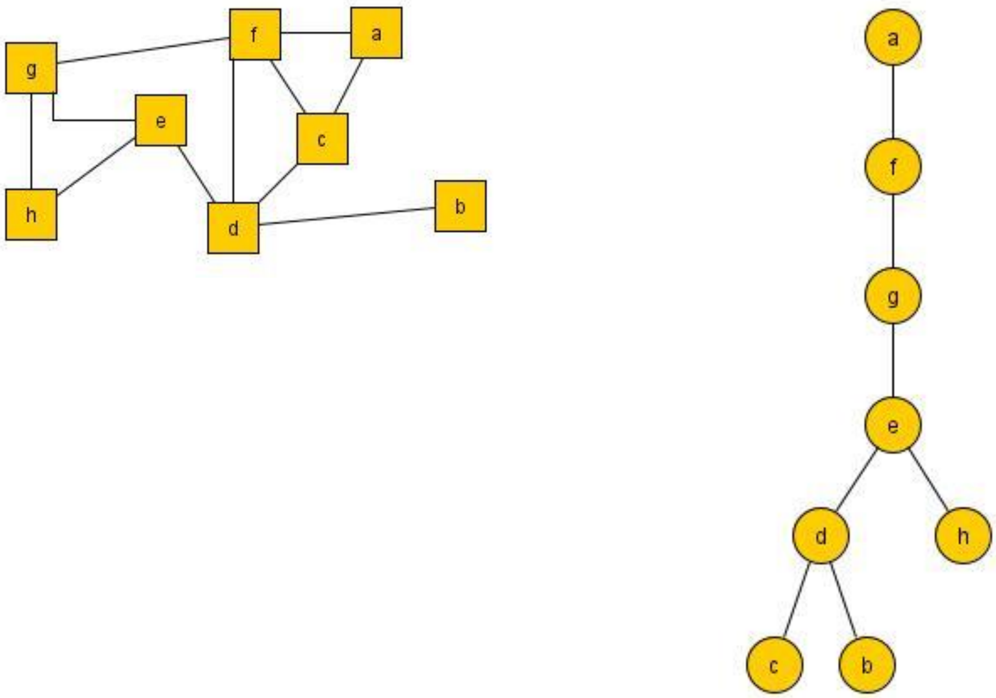


Figura 6

Proposição 1. Sejam u e v dois vértices de *G*, com $f(u) < f(v)$.

a) Se u e v são adjacentes em *G*, então $t(v) < t(u)$

b) u é um ancestral de v em *T* se e somente se $t(v) < t(u)$

A cláusula a) indica que se u e v são adjacentes em G então o tempo de saída da pilha de v, vértice que foi adicionado a árvore depois, é menor que o tempo de saída da pilha de u. Isto ocorre porque v foi incluído na árvore depois de *u*. Por definição, deixará a pilha antes de *u*. Os demais vizinhos de v que forem incluídos a árvore em passos subsequentes, entrarão na pilha antes de *u* sair. Como exemplo, considere os vértices **e** e **d** na demonstração acima do algoritmo, assim como o vértice raiz **a** que é o primeiro a entrar na pilha e o último a sair.

A cláusula b) indica que o vértice u é um ancestral de v em T se s somente se o tempo de saída da pilha de v, vértice que foi adicionado a árvore depois, é menor que o tempo de saída da pilha de u. Podemos observar que esta proposição é válida através da árvore resultante exemplo acima, onde vemos que todo ancestral de qualquer vértice tem um tempo de saída da pilha maior que o de qualquer descendente.

Teorema 3. Seja *T* uma DFS-árvore de um grafo *G*. Então todas as arestas de *G* ligam vértices que estão relacionados em *T*.

A prova para este teorema segue diretamente da Proposição 1. Seja uv uma aresta em G. Suponha que $f(u) < f(v)$. Então $t(v) < t(u)$. Neste caso, u é um ancestral de v e, portanto, estão relacionados em T.

A Figura 7 ilustra outra árvore DFS que poderia ter sido produzida no exemplo anterior se tivéssemos escolhido o vértice c ao invés de f na segunda iteração. Note que, tanto a árvore da Figura 6 quando a da Figura 7 atendem ao Teorema 3.

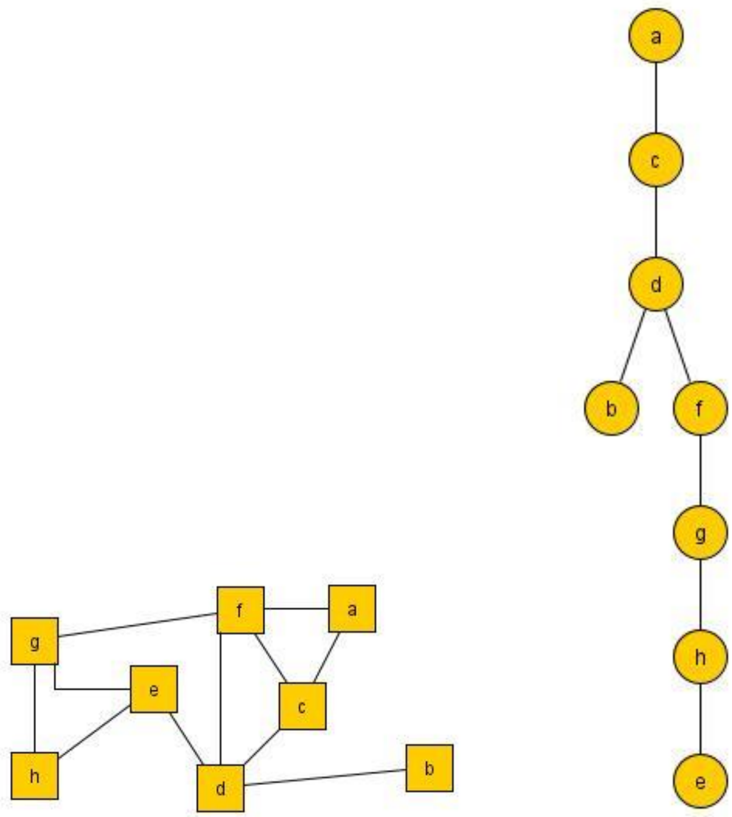


Figura 7

Em uma árvore de busca em profundidade, um vértice *u* é um **vértice de corte** se uma das duas condições a seguir for verdadeira.

1. *u* é a raiz da árvore DFS e possui pelo menos dois filhos.
2. *u* não é raiz da árvore DFS e possui um filho *v*, de modo que nenhum vértice na subárvore com raiz *v* é adjacente no grafo original a um dos ancestrais (na árvore DFS) de *u*.

Considere o grafo e uma árvore DFS apresentados na Figura 8, podemos observar que **c** não é um vértice de corte, pela condição 1. Já o vértice **d**, é um vértice de corte por que pela condição

2, existe um sucessor de **d**, por exemplo, o vértice **a,** tal que este vértice e seus descendentes não são adjacentes aos antecessores de **d**, ou seja, aos vértices **c** e **e**.

Já o vértice **f** não é um vértice de corte pela condição 2 porque, porque seus descentes são adjacentes a pelo menos um antecessor de **f**.

Considere também o grafo da Figura 2 e suas árvores DFS nas Figuras 6 e 7. Note que podemos determinar que **d** é um vértice corte seguindo a condição 2.

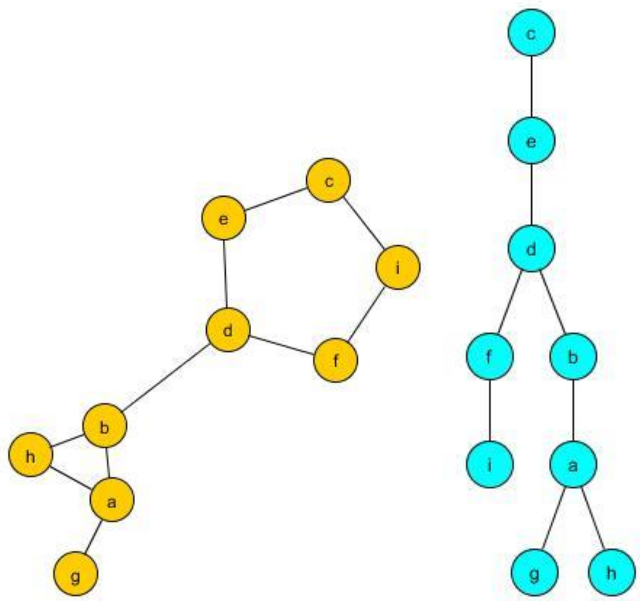
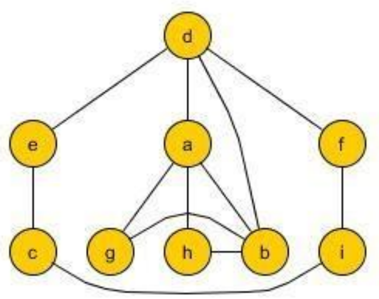


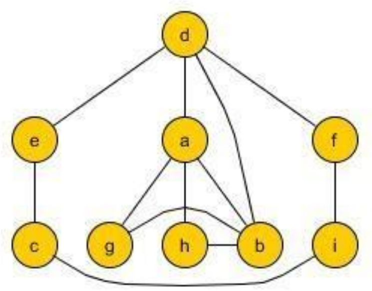
Figura 8

Exercícios Propostos

1. Para o grafo abaixo, encontra uma árvore de busca em largura, considerando o vértice a como raiz.



2. Para o grafo abaixo, encontre uma árvore de busca em profundidade, considerando o vértice a como raiz.



3. Seja T uma BFS-árvore de um grafo conectado G. Mostre que $l(v) = d_T(r,v)$ para todo $v \in V$.

4. Seja T uma BFS-árvore de um grafo conectado G e seja z o último vértice que entra em T. Mostre que T-z é uma BFS-árvore de G-z.

5. Um vértice v de um grafo é central se minimiza a distância máxima entre v e qualquer vértice x. Mostre que toda árvore tem no máximo dois centros e se tiver dois então eles são adjacentes.

Referências

J. A. Bondy and U. S. R. Murty.Graph Theory. Springer, 2008,2010.

- 6.1

https://www.cs.usfca.edu/~galles/visualization/BFS.html

https://www.cs.usfca.edu/~galles/visualization/DFS.html