Respostas da lista de exercícios 01 - Grafos: conceitos e buscas

1. Responda verdadeiro ou falso. Justifique cada caso mediante prova ou contraexemplo:

(a) Um grafo G é uma árvore se e somente se G for acíclico maximal (ou seja, G é acíclico e a adição de qualquer aresta cria um ciclo).

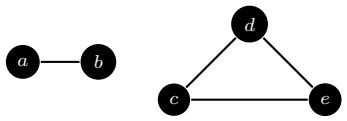
Resposta. Verdadeiro.

$\Rightarrow$ Por definição, toda árvore é acíclica e, como visto em aula, a adição de uma aresta a qualquer árvore cria um ciclo, logo toda árvore é um grafo acíclico maximal. Portanto, se G é uma árvore, então G é acíclico maximal.

$\Leftarrow$ Seja G um grafo acíclico maximal e u, v dois vértices quaisquer de G. Se a aresta (u, v) estiver em G, então os vértices são adjacentes. Caso contrário, a adição dessa aresta em G cria um ciclo, logo existe um caminho entre u e v em G. Assim, entre todo par de vértices de G há um caminho, portanto G é conexo e, como também é acíclico, temos que G é uma árvore.

(b) Se em um grafo G todos os vértices tem grau no conjunto $\{1, 2\}$ e há exatamente dois vértices com grau 1, então G é exatamente um caminho.

Resposta. Falso. Considere o grafo $G = (\{a, b, c, d, e\}, \{(a, b), (c, d), (c, e), (d, e)\})$ (na figura embaixo). Todos os vértices de G tem grau 1 ou 2 e há exatamente dois vértices de grau 1 (a e b). Contudo G não é um caminho.



(c) Um grafo bipartido com n vértices tem no máximo $\frac{n^2}{4}$ arestas.

Resposta. Verdadeiro. Se o grafo é bipartido, então o conjunto de vértices pode ser particionado em dois conjuntos: um com p vértices e outro com q vértices, onde $p+q = n$. Como não pode haver arestas com os dois extremos no mesmo conjunto, o máximo de arestas é quando cada vértice de um conjunto tem como adjacentes todos os do outro. Isto é, quando $p \cdot q$ tem valor máximo. Observe que:

$$4p \cdot q = 2p \cdot q + 2p \cdot q = p^2 + 2p \cdot q + q^2 - (p^2 - 2p \cdot q - q^2) = (p+q)^2 - (p-q)^2 = n^2 - (p-q)^2.$$

Logo, $p \cdot q = \frac{n^2-(p-q)^2}{4} \leq \frac{n^2}{4}$, pois $(p-q)^2 \geq 0$ para quaisquer valores de p e q. Portanto, um grafo bipartido não tem mais que $\frac{n^2}{4}$ arestas.

(d) Se d é o grau mínimo de um grafo G, então em G há um caminho de comprimento maior ou igual que d.

Resposta. Verdadeiro. Considere um caminho maximal $P = (v_0, v_1, \ldots, v_k)$ de G. Como d é o menor grau de qualquer vértice em G, temos que $d(v_k) \geq d$. Logo, v_k, tem pelo menos d vizinhos e todos devem pertencer ao caminho P, caso contrário, P não seria maximal. Portanto, P contém pelo menos $d+1$ vértices (v_k mais pelo menos d vizinhos), sendo seu comprimento pelo menos d.

2. Um *k-cubo* é um grafo cujos vértices são todas as sequências $b_1 b_2 \dots b_k$ em que cada b_i pertence a $\{0, 1\}$. Em um k-cubo, dois vértices são adjacentes se diferem em exatamente uma posição. Quantas arestas tem o k-cubo? Prove que um k-cubo é bipartido.

Resposta. Cada vértice do k-cubo tem exatamente k vizinhos (cada vizinho se obtém trocando um dos k bits do vértice). O k-cubo tem 2^k vértices (todas as combinações de k bits). Logo, a soma dos graus dos vértices do k-cubo é $2^k k$ e pelo lema de aperto de mãos o número de arestas resulta a metade desse valor: $2^{k-1}k$.

Se um vértice u do k-cubo tem um número par de 1's, então qualquer adjacente dele deve ter um número ímpar de 1's (pois somente são adjacentes vértices com um bit diferente). Portanto, vértices com número par de 1's não são adjacentes no k-cubo. Um raciocínio análogo implica que vértices com número ímpar de 1's não são adjacentes no k-cubo. Assim, podemos particionar os vértices do k-cubo em dois conjuntos, um com aqueles que tem número par de 1's e outro com os que tem número ímpar de 1's, garantindo que não há arestas do k-cubo com os dois extremos no mesmo conjunto. Concluindo que o k-cubo é bipartido.

3. Uma *grade p-por-q* é um grafo cujo conjunto de vértices é o produto cartesiano $\{1, 2 \dots, p\} \times \{1, 2, \dots, q\}$ e dois vértices (i, j) e (i', j') são adjacentes se: $|i - i'| = 1$ e $j = j'$, ou $i = i'$ e $|j - j'| = 1$. Quantas arestas tem a grade p-por-q? Mostre que a grade p-por-q é um grafo bipartido.

Resposta. Analisemos o grau dos vértices em uma grade p-por-q, com $p, q \geq 2$:

(i) Se $2 \leq i \leq p-1$ e $2 \leq j \leq q-1$, então o vértice (i, j) tem exatamente 4 vizinhos.

(ii) Se $i \in \{1, p\}$ e $2 \leq j \leq q-1$, então o vértice (i, j) tem exatamente 3 vizinhos.

(iii) Se $2 \leq i \leq p-1$ e $j \in \{1, q\}$, então o vértice (i, j) tem exatamente 3 vizinhos.

(iv) Se $i \in \{1, p\}$ e $j \in \{1, q\}$, então o vértice (i, j) tem exatamente 2 vizinhos.

Há $(p-2)(q-2)$ vértices no caso (i), $2(q-2)$ no (ii), $2(p-2)$ no (iii) e 4 no (iv). Somando os graus de todos os vértices, temos: $4(p-2)(q-2)+3(2(q-2))+3(2(p-2))+2(4) = 4pq-2p-2q = 2(2pq-p-q)$. Como a soma dos graus dos vértices é o dobro do número de arestas, temos $2pq-p-q$ arestas.

Se $p = 1$ e $q \geq 1$, a grade 1-por-q é um caminho de q vértices e $q - 1 = 2pq - p - q$ arestas. Se $p \geq 1$ e $q = 1$, a grade p-por-1 é um caminho de p vértices e $p - 1 = 2pq - p - q$ arestas. Assim, qualquer seja o caso, temos que a grade p-por-q possui $2pq - p - q$ arestas.

Para provar que p-por-q é bipartido, considere os conjuntos de vértices:

- $U = \{(i, j) | 1 \leq i \leq p \text{ e } 1 \leq j \leq q \text{ e } i(mod\ 2) = j(mod\ 2)\}$.
- $W = \{(i, j) | 1 \leq i \leq p \text{ e } 1 \leq j \leq q \text{ e } i(mod\ 2) \neq j(mod\ 2)\}$

Isto é, U contém os vértices em que i e j tem a mesma paridade (os dois são pares ou os dois são ímpares); enquanto W contém aqueles em que i e j possuem paridade diferente (um é par e o outro ímpar). Trivialmente, (U, W) é uma partição dos vértices da p-por-q grade. Precisamos provar que nenhuma aresta tem os dois extremos em U ou em W.

Note que, um vértice (i, j) só pode ser adjacente de (i, j') ou de (i', j) com $|j-j'| = |i-i'| = 1$. Como $|j-j'| = |i-i'| = 1$, temos que a paridade de i' é diferente da paridade de i e a paridade de j' é diferente da paridade de j. Assim, se $(i, j) \in U$, então i e j tem a mesma paridade, enquanto i e j' tem paridade diferente, assim como i' e j, pelo que qualquer adjacente $(i, j'), (i', j) \in W$. Um argumento análogo nos mostra que se $(i, j) \in W$, então os adjacentes estariam em U. Desta forma, concluímos que uma grade p-por-q é um grafo bipartido.

4. O *diâmetro* de um grafo G é a maior distância entres dois vértices de G.

(a) Seja G um grafo simples de diâmetro maior ou igual que três. Mostre que $\overline{G}$ tem diâmetro no máximo três.

Resposta. Sejam u e v dois vértices de $G = (V, E)$, tais que a distância entre eles em G é igual ao diâmetro de G. Como $dist_G(u, v) \geq 3$, temos as seguintes propriedade:

(i) $(u, v) \notin E$, portanto $(u, v) \in \overline{E}$ (isto é, a aresta entre u e v está no complemento).

(ii) Para qualquer vértice w, $(u, w) \notin E$ ou $(v, w) \notin E$ (isto é, w não pode ser adjacente em G de u e v ao mesmo tempo, caso contrário a distância entre u e v seria 2). Isto implica que, todo vértice w é adjacente em $\overline{G}$ de u ou de v.

Sejam x e y dois vértices quaisquer. Pela propriedade (ii) x é adjacente de u ou v em $\overline{G}$, o mesmo vale para y. Se os dois forem adjacentes do mesmo vértice em $\{u, v\}$, então $dist_{\overline{G}}(x, y) \leq 2$. Senão, sem perda de generalidade, suponha que x é adjacente de u em $\overline{G}$ e y de v, pela propriedade (i) a aresta (u, v) está em $\overline{G}$ e temos o passeio (x, u, v, y) entre x e y em $\overline{G}$, logo $dist_{\overline{G}}(x, y) \leq 3$.

(b) Seja T uma árvore com pelo menos uma aresta de diâmetro d. Suponha que após a remoção de uma aresta, obtemos duas componentes conexas T_1 e T_2 com diâmetro d_1 e d_2, respectivamente. Mostre que $d \geq (d_1 + d_2)/2 + 1$.

Resposta. Denote por P_1 um caminho em T_1 com comprimento d_1 e por P_2 um caminho em T_2 com comprimento d_2. Sejam $u \in P_1$ e $v \in P_2$ dois vértices, tais que a distância em T seja mínima (isto é, o vértice de P_1 e o de P_2 mais perto um do outro em T). Denote por P_1^u o subcaminho de P_1 entre u e o extremo mais longe. Analogamente, denote por P_2^v o subcaminho de P_2 entre v e o extremo mais longe. Finalmente, denote por P_{uv} o caminho em T entre u e v. Note que, $P = P_1^u P_{uv} P_2^v$ é um caminho de T com comprimento igual à soma dos comprimentos de P_1^u, de P_{uv} e de P_2^v. Como, P_1^u é o subcaminho de P_1 entre u e o extremo mais longe, seu comprimento é pelo menos $\frac{d_1}{2}$, analogamente o comprimento de P_2^v é pelo menos $\frac{d_2}{2}$ e, como $u \in T_1$ e $v \in T_2$, temos que o comprimento de P_{uv} é maior ou igual que 1. Portanto, o comprimento de P é, maior ou igual que $\frac{d_1}{2} + \frac{d_2}{2} + 1 = \frac{(d_1+d_2)}{2} + 1$. Como T é uma árvore, exite um único caminho entre qualquer par de vértices, logo a distância entre os extremos e P é $\frac{(d_1+d_2)}{2} + 1$ e o diâmetro tem que ser pelo menos esse valor.

5. Dê um algoritmo que determine se um determinado grafo não direcionado $G = (V, E)$ contém um ciclo. Seu algoritmo deve ser executado em $O(V)$, independente de $|E|$.

Resposta. Primeiro, observe que um grafo não direcionado tem ciclos se e somente se durante a execução do DFS encontramos uma aresta de retrocesso. Segue a prova:

$\Rightarrow$ Considere um grafo com ciclos e seja C um ciclo do grafo. Denote por u o primeiro vértice de C a ser descoberto durante o DFS e v o segundo. Como v é alcançável por u, temos que no momento em que v é descoberto, o processamento de u não terminou (u é cinza). Assim, v é descendente de u. Um argumento similar, nos mostra que todo vértice de $C \setminus \{u, v\}$ é descendente de v (e também de u). Como u tem dois adjacentes em C, um deles deve ser diferente de v e portanto ser descendente de v, denote esse vértice por w. Durante o processamento de w, a aresta (w, u) é analisada, que leva ao vértice u (ainda cinza). Como u não é o pai de w (pois na árvore de busca v é um ancestral mais perto de w que u), a aresta (w, u) é de retrocesso.

$\Leftarrow$ Se (u, v) é uma aresta de retrocesso analisada durante o chamado de u, então v é ancestral de u (e não é o pai de u), portanto existe um caminho P de v até u com comprimento pelo menos 2 e adicionando (u, v) a esse caminho, temos um ciclo.

Pelo resultado anterior, basta encontrar uma aresta de retrocesso durante o DFS para saber se o grafo contém ciclos. Portanto um algoritmo para tal função, só precisa retornar no momento em

que tal aresta é encontrada. Observe que em grafos não direcionados, não há arestas de cruzamento ou avanço (visto em aula), assim todas as arestas são da árvore de busca ou de retrocesso. Como a árvore de busca tem no máximo $|V| - 1$ arestas, temos que o algoritmo analisa no pior caso $O(V)$ arestas.

6. Uma outra maneira de obter uma ordenação topológica em um grafo direcionado $G = (V, E)$ é repetidamente encontrar um vértice com grau de entrada 0, incluí-lo na solução e depois remover do grafo esse vértice com todas as arestas de saída. Explique como implementar essa ideia de forma que o algoritmo execute em tempo $O(V + E)$.

Resposta. Para implementar essa ideia em tempo $O(V + E)$, considere o seguinte processo:

1. Atribua a cada vértice $u \in V$ um valor $d^-[u]$ que seja igual ao grau de entrada. Isto pode ser feito em tem $O(V+E)$: inicializar todos os $d^-[u] = 0$ em tempo $O(V)$ e por cada aresta (u, v) do grafo fazer $d^-[u] \leftarrow d^-[u] + 1$ em tempo $O(E)$.

2. Inicializar uma fila Q vazia e percorrer os vértices do grafo inserindo em Q aqueles com $d^-[u] = 0$, isto consome tempo $O(V)$.

3. Percorrer a fila em ordem (não é necessário desenfileirar), por cada vértice u, percorrer cada aresta (u, v) e fazer $d^-[v] \leftarrow d^-[v]-1$, em caso de $d^-[v] = 0$, adicionar v no final da fila Q. Isto consome tempo $O(V + E)$, pois cada vértice é enfileirado exatamente uma vez e suas arestas percorridas uma única vez.

4. Retornar a fila Q.

7. Um ponto de articulação de um grafo G é um vértice cuja remoção provoca o aumento de componentes conexas. Projete um algoritmo para encontrar pontos de articulação em tempo $O(V + E)$.

Resposta. Para solucionar este problema, adicionaremos uma informação adicional durante a execução do DFS: cada vértice u terá associado um valor $h[u]$ que indica quão alto na sua árvore de DFS ele alcança (isto é qual dos antecessores, distintos do pai, tem o menor tempo de descoberta que ele ou algum descente dele chega).

Se u é um ponto de articulação e ele for raiz numa árvore do DFS, então ele precisa ter pelo menos dois filhos. Se tiver só um filho, esse filho alcança todos os vértices que u alcança e não precisa passar por u, portanto a remoção de u não aumentaria o número de componentes. Se tiver mais de um filho, então o primeiro filho só alcança o segundo através de u (caso contrário o segundo filho seria descendente do primeiro), portanto a remoção de u desconectaria o primeiro filho do segundo aumentando o número de componentes.

Se u é um ponto de articulação e não for raiz numa árvore do DFS, então existe algum filho v de u tal que $h[v] \geq d[u]$ (o mais alto que o filho alcança é o próprio u). Note que, a remoção de u não tem como desconectar seus ancestrais e se todos os filhos de u alcançam acima de u, significa que eles tem caminhos (que não passam por u) até algum ancestral de u, portanto poderíamos remover u sem aumentar o número de componentes do grafo. Se existe um filho v de u tal que $h[v] \geq d[u]$, então qualquer caminho entre v e um ancestral de u passa por u, portanto ao remover u desconectamos seu v dos ancestrais de u aumentando o número de componentes.

Pelo anteriormente exposto, basta modificarmos o DFS contando os filhos (no caso que seja raiz) e adicionando o calculo do valor $h[u]$. Para isto, no início do processamento de um vértice u, podemos colocar seu $h[u] \leftarrow d[u]$ (que alcança tão alto quanto sua descoberta) e o número de filhos igual a zero. Por cada adjacente descoberto v (diferente do pai), fazemos $h[u] \leftarrow \min\{h[u], d[v]\}$, isto é, u alcança esse ancestral ou um mais alto. Por cada adjacente não descoberto v, aumentamos o número de filhos em 1 e fazemos $h[u] \leftarrow \min\{h[u], h[v]\}$ após o chamado DFS(v), ou seja, u alcança o mesmo que o filho ou mais alto.

```
    input    : u vértice sendo visitado; p pai de u (NIL quando u é raiz)
 1  begin
 2      cor[u] ← cinza
 3      tempo ← tempo + 1
 4      d[u] ← tempo
 5      h[u] ← d[u]
 6      filhos ← 0
 7      for v ∈ Adj[u] do
 8          if cor[v] = branco then
 9              DFS(v, u)
10              h[u] ← min{h[u], h[v]}
11              if h[v] ≥ d[u] then
12                  PONTO_DE_ARTICULACAO(u)
13              end
14              ← filhos + 1
15          else
16              if v ≠ p then
17                  h[u] ← min{h[u], d[v]}
18              end
19          end
20      end
21      if p = NIL e filhos > 1 then
22          PONTO_DE_ARTICULACAO(u)
23      end
24      cor[u] ← preto
25      tempo ← tempo + 1
26      f[u] ← tempo
27  end
```

Algorithm 1: DFS modificado para pontos de articulação.

As modificações propostas mantém o DFS com o mesmo custo $O(V + E)$ visto em aula.

8. Projete um algoritmo que recebe um digrafo acíclico G e dois vértices s e t e devolve o número de caminhos diferentes em G de s até t. Seu algoritmo deve executar em tempo $O(V + E)$.

Resposta. A quantidade de caminhos diferentes de um vértice u qualquer até t, pode ser obtida somando, para cada vizinho v de u a quantidade de caminhos de v até t (note que por cada caminho de v até t há um caminho de u até t usando a aresta (u, v)). Assim, se denotarmos por $q[u]$ a quantidade de caminhos de um vértice u até t temos que:

$$q[u] = \begin{cases} 1 & \text{se } u = t \\ \sum_{v \in Adj[u]} q[v] & \text{se } u \neq t \end{cases}$$

Como o digrafo é acíclico, o valor $q[u]$ pode ser calculado para cada vértice u seguindo a ordem reversa de uma ordenação topológica. Isto é, começando pelo último vértice da ordenação, depois seu predecessor, depois seu predecessor, e assim até chegar no primeiro vértice da ordenação. Como em uma ordenação topológica não há arcos de um vértice para qualquer dos que o precedem, o algoritmo em questão garante que ao computar $q[u]$ o valor de cada adjacente v de u teria sido computado previamente. Obter uma ordenação topológica pode ser feito em $O(V + E)$, assim como percorrer os vértices da ordenação calculando o vetor q para cada um deles.

9. Um *ralo universal* em um digrafo $G = (V, E)$ é um sorvedouro com grau de entrada $|V| - 1$ (no qual chega um arco desde cada vértice do grafo). Dada a matriz de adjacências de um digrafo G, projete um algoritmo $O(V)$ para determinar se G possui um ralo universal.

Resposta. Para solucionar este problema consideramos os vértices numerados $\{1, 2, \ldots, |V|\}$. A ideia consiste em primeiro selecionar um vértice candidato a ralo universal e depois testar se tal vértice é de fato um ralo universal.

Mantemos dois vértices, um que é o candidato (r) e outro que usamos para verificar (v). Inicializamos o candidato com o primeiro vértice ($r \leftarrow 1$) e o verificador com o segundo ($v \leftarrow 2$). Enquanto o verificador não chega no final ($v < |V|$), testamos se existe o arco de (v, r). Se existir, note que o verificador não pode ser ralo (pois tem um arco saindo), assim aumentamos o verificador para o próximo vértice ($v \leftarrow v + 1$). Se não existir, então o candidato atual não pode ser o ralo (pois está faltando o arco de um vértice para ele), assim fazemos que o novo candidato

seja o próprio verificador ($r \leftarrow v$) e fazemos que verificador seja o próximo vértice ($v \leftarrow v + 1$). Note que, nesse segundo caso, os verificadores e candidatos anteriores já haviam sido descartados, portanto é desnecessários considerá-los. Caso G possua um ralo universal, o único vértice que pode ser esse ralo é o candidato que o algoritmo encontra, pois todos os outros foram descartados: por ter um arco saindo deles ou por estar faltando um arco entrando de algum vértice. Nesta ideia, em cada iteração o verificador aumenta em 1, portanto há $|V|$ iterações com custo $O(1)$ (testar a existência de uma aresta na matriz e atualizar o verificador e o candidato quando necessário). Logo, o custo de selecionar um candidato a ralo universal é $O(V)$. Segue o algoritmo de seleção:

```
   input  : Grafo G = (V, E), representação em matriz de adjacência
   output : r candidato a ralo universal de G.
 1 begin
 2 |   r ← 1
 3 |   v ← 2
 4 |   while v ≤ |V| do
 5 |   |   if (v, r) ∈ E then
 6 |   |   |   v ← v + 1
 7 |   |   else
 8 |   |   |   r ← v
 9 |   |   |   v ← v + 1
10 |   |   end
11 |   end
12 |   return r
13 end
```

Algorithm 2: Encontra candidato.

Para testar se o candidato selecionado é de fato um ralo universal, basta percorrer na matriz sua linha para ver que não tem nenhum arco saindo e sua coluna para verificar que tem $|V| - 1$ arcos chegando. Percorrer uma linha e uma coluna para verificar que haja ou não arcos pode ser feito em $O(V)$. Segue o algoritmo verificador:

```
   input  : Grafo G = (V, E), representação em matriz de adjacência; r candidato a ralo universal de G.
   output : TRUE se r for um ralo universal, FALSE senão.
 1 begin
 2 |   for v = 1 até |V| do
 3 |   |   if v ≠ r e (v, r) ∉ E then
 4 |   |   |   return FALSE
 5 |   |   end
 6 |   |   if (r, v) ∈ E then
 7 |   |   |   return FALSE
 8 |   |   end
 9 |   end
10 |   return TRUE
11 end
```

Algorithm 3: Testa candidato.

10. Seja T uma árvore produzida pelo *BFS* de um grafo G. Embora os conceitos de arcos de avanço, cruzamento e retrocesso tenham sido definidos no contexto da busca *DFS*, eles fazem sentido em relação a qualquer árvore ou floresta radicada de G.

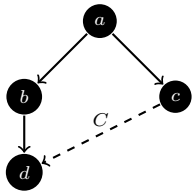
(a) Mostre que G não tem arcos de avanço em relação a T.

Resposta. Suponha que (u, v) seja um arco de avanço. Isso significa que v é descendente de u em T. Portanto, no momento em que u vai ser processado, v não havia sido descoberto ainda (v é branco). Durante o processamento de u todos seus adjacentes não descobertos (brancos) são adicionados na fila como filhos de u em T. Logo, v teria sido adicionado como filho de u e a aresta (u, v) seria aresta da árvore e não de avanço.

(b) Mostre que, se G for não-dirigido, então G não tem arcos de retorno.

Resposta. Mesmo texto que o item anterior, só trocar (u, v) por (v, u) e "avanço" por "retorno".

(c) G pode ter arcos de cruzamento em relação a T? E se G for não-dirigido?



Resposta. Sim, pode haver arcos de cruzamento, considere o seguinte exemplo:

Suponha um BFS a partir de a que enfileira b e c. Depois a partir de b enfileira d. Temos que o arco (c, d) é de cruzamento.

Se G não for dirigido, o mesmo exemplo vale, só precisa remover a orientação dos arcos.