O exercício pede para demonstrar que, em um grafo não orientado G=(V,E)G = (V, E)G=(V,E), a soma dos graus de todos os vértices é igual ao dobro do número de arestas, ou seja:

### 

### **Solução:**

Este resultado é conhecido como **Teorema da Mão Dupla** ou **Teorema do Aperto de Mãos**. Vamos demonstrar:

1. **Definição de grau de um vértice**: O graud(v) de um vértice v em um grafo não orientado é o número de arestas que incidem sobre ele.
2. **Soma dos graus**: Se somarmos o grau de todos os vértices do grafo G, cada aresta será contada duas vezes (uma vez para cada extremidade da aresta). Portanto, a soma total dos graus de todos os vértices é igual a duas vezes o número total de arestas ∣E(G)∣.
3. **Demonstração formal**:
   * Cada aresta e={u,v} contribui com 1 ao grau do vértice u e 1 ao grau do vértice v.
   * Como cada aresta é contada duas vezes na soma dos graus, temos que: ∑v∈V(G)d(v)=2×∣E(G)∣

**Exemplo:**

Considere o grafo GGG com V={A,B,C} e E={{A,B},{B,C}}:

* d(A)=1
* d(B)=2
* d(C)=1

Soma dos graus: 1+2+1=4

Número de arestas: ∣E(G)∣=2

Portanto, ∑v∈V(G)d(v)=2∣E(G)∣

### 

### **Demonstração:**

**Definição de árvore**:

* Uma **árvore** é um grafo conexo e acíclico.
* Em uma árvore com nnn vértices, existem exatamente n−1n - 1n−1 arestas.

**Definição de caminho**:

* Um **caminho** é um grafo em que existe exatamente uma sequência de vértices v1,v2,…,vkv\_1, v\_2, \dots, v\_kv1​,v2​,…,vk​ conectados por arestas, sem formar ciclos e sem bifurcações.

Agora, provemos que uma árvore com exatamente dois vértices de grau 1 é um caminho.

### **Passo 1: Grau dos vértices em uma árvore**

* Os **vértices de grau 1** em uma árvore são chamados de **folhas**.
* Qualquer árvore com nnn vértices e n−1n - 1n−1 arestas precisa ter pelo menos **uma folha** (vértice de grau 1).

### **Passo 2: Configuração com dois vértices de grau 1**

* Suponha que temos uma árvore com **dois vértices de grau 1**: uuu e vvv.
* Como a árvore é conexa e acíclica, os vértices uuu e vvv precisam estar conectados por uma sequência de vértices u=v1,v2,…,vk=vu = v\_1, v\_2, \dots, v\_k = vu=v1​,v2​,…,vk​=v, formando um **caminho simples**.

### **Passo 3: Prova por contradição**

* Suponha, por contradição, que a árvore **não é um caminho**. Então, a árvore deve conter uma **bifurcação** (um vértice com grau maior que 2).
* Se isso acontecer, haverá mais de duas folhas ou múltiplos ramos, o que viola a condição de que há **exatamente dois vértices de grau 1**.
* Isso contradiz a hipótese inicial.

### **Conclusão**

* Portanto, uma árvore com **exatamente dois vértices de grau 1** deve ser **um caminho**, pois não existe outra forma de conectar os vértices sem introduzir bifurcações ou ciclos.



### **Definições Importantes:**

1. **Árvore**:
   * Um grafo conexo e acíclico.
   * Se T tem nnn vértices, ele terá exatamente n−1 arestas.
2. **Grau de um vértice**:
   * O grau de um vértice vvv, denotado por d(v), é o número de arestas incidentes a v.
3. **∆(T)**:
   * Δ(T) é o **grau máximo** entre todos os vértices da árvore T.
   * 

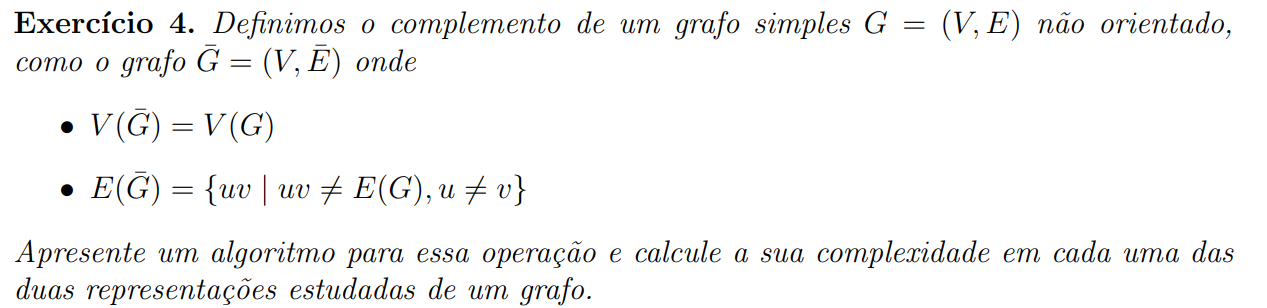
### **Demonstração:**

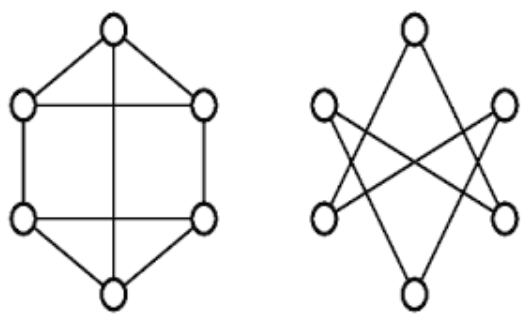
Queremos provar que se Δ(T)≥k, então a árvore T possui pelo menos k vértices.

1. **Árvore com grau máximo Δ(T) ≥ k**:
   * Por definição, existe pelo menos um vértice v∈V tal que d(v)≥k.
   * Esse vértice está conectado a pelo menos **k outros vértices**, pois seu grau é o número de arestas conectadas a ele.
2. **Número mínimo de vértices**:
   * O vértice v contribui com k conexões. Como cada uma dessas conexões é com um vértice diferente (pois não existem loops em uma árvore), temos no mínimo k vértices distintos:
     + O próprio vértice v, com grau d(v)≥k.
     + Os kkk vértices conectados a v.
3. **Conclusão**:
   * Uma árvore T com Δ(T)≥k deve ter pelo menos k vértices, pois um vértice com grau k ou mais necessita de pelo menos k vértices (o próprio vértice e k−1 vizinhos).

### **Exemplo:**

* Considere uma árvore T com Δ(T)=3:
  + O vértice vvv com d(v)=3 está conectado a três outros vértices.
  + Assim, o número mínimo de vértices na árvore é 4 (v + 3 vértices conectados).





#### **Definição do Complemento de um Grafo**

Seja G=(V,E) um grafo simples e não orientado. O **complemento** de G, denotado como G=(V,E), é definido por:

* V(G)=V(G)(os vértices permanecem os mesmos).
* , ou seja, o conjunto de arestas do complemento contém todas as arestas que **não estão presentes em GGG**.

### **Objetivo:**

1. Apresentar um algoritmo para construir o complemento de G.
2. Determinar a complexidade computacional do algoritmo considerando duas representações:
   * **Matriz de Adjacência**
   * **Lista de Adjacência**

### **1. Algoritmo para construir o complemento de um grafo**

#### **Entrada:**

* Um grafo G=(V,E)G = (V, E)G=(V,E), representado por uma matriz ou lista de adjacência.

#### **Saída:**

* O grafo G=(V,E).

#### **Algoritmo (usando matriz de adjacência):**

Seja n=∣V∣, o número de vértices.

1. Crie uma matriz de adjacência M para G, onde M[i][j]=1 se existe uma aresta entre i e j, e 0 caso contrário.
2. Inicialize uma matriz de adjacência M para G com todos os valores iguais a 0.
3. Para cada par (u,v) com 1≤u,v≤n, e u≠v:
   * Se M[u][v]=0, defina M‾[u][v]=1.
   * Caso contrário, M[u][v]=0.
4. Retorne M.

#### **Algoritmo (usando lista de adjacência):**

1. Inicialize uma lista de adjacência L para G.
2. Para cada vértice u∈V, inicialize uma nova lista L[u].
3. Para cada vértice v∈V, com v≠u:
   * Se v∉L[u], adicione v à lista L[u].
4. Retorne L

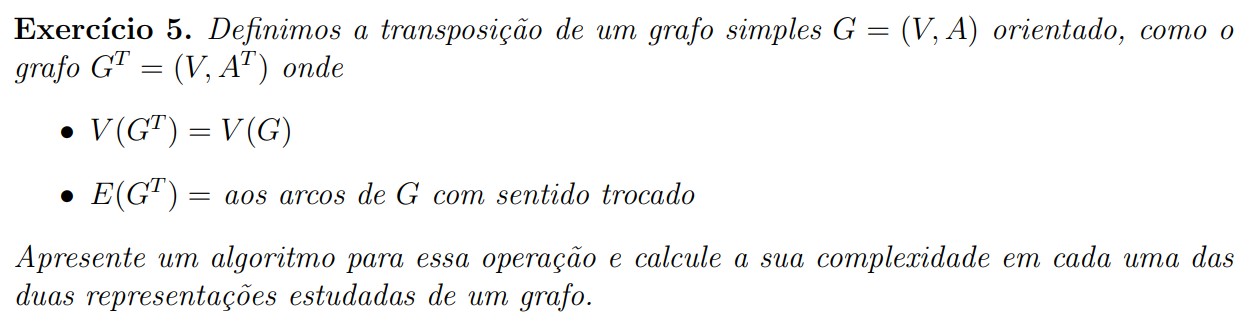
### **2. Complexidade do Algoritmo**

#### **a. Matriz de Adjacência**

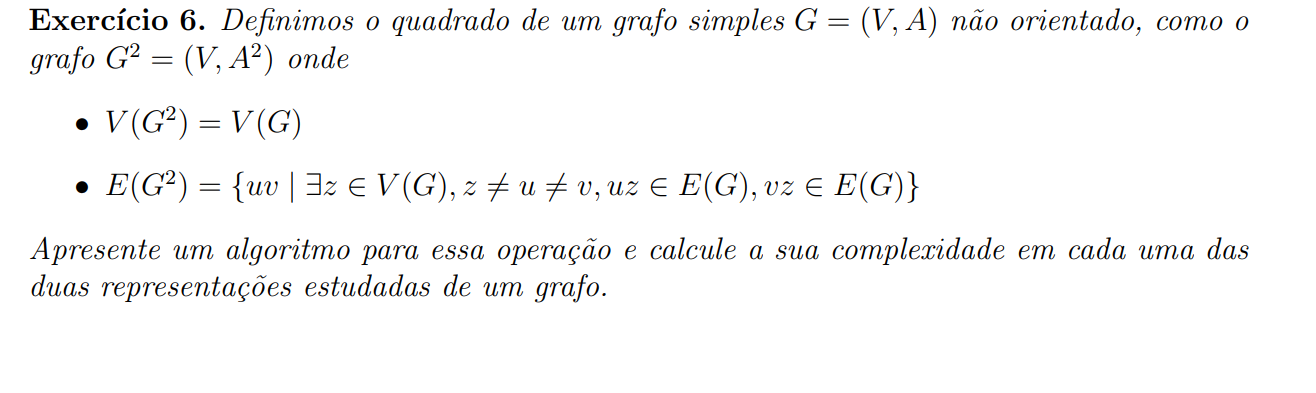
* A matriz de adjacência tem tamanho O(n^2), onde n=∣V∣.
* O algoritmo percorre todos os pares de vértices (u,v), verificando se existe uma aresta em G.
* **Complexidade**: O(n^2).

#### **b. Lista de Adjacência**

* Em uma lista de adjacência, o algoritmo percorre todos os vértices e, para cada vértice, verifica os n−1 possíveis vértices adjacentes.
* Verificar se v∉L[u] pode ser feito em O(deg⁡(u)).
* No pior caso, percorremos todas as n(n−1)/2 arestas ausentes.
* **Complexidade**: O(n2).



Para um grafo orientado, a transposição deste se dá apenas pela inversão do sentido das arestas. Logo, em uma matriz de adjacência, basta transpor a matriz (inverter o índice i pelo j).

O **quadrado de um grafo** G=(V,E), denotado por G^2. é definido como um grafo que possui os mesmos vértices de G, mas as arestas de G^2 conectam dois vértices u e v se:

1. Existe uma aresta uv∈E(G), ou
2. Existe um vértice w tal que uw∈E(G) e wv∈E(G). Em outras palavras, u e v estão a uma distância no máximo 2 em G.

### **Objetivo**

1. Apresentar um algoritmo para calcular G^2.
2. Determinar a complexidade computacional do algoritmo para duas representações:
   * **Matriz de Adjacência**
   * **Lista de Adjacência**

### **1. Algoritmo para calcular G^2**

#### **Entrada:**

* Um grafo G=(V,E), representado por uma matriz ou lista de adjacência.

#### **Saída:**

* O grafo G^2 = (V, E^2).

#### **Algoritmo (Matriz de Adjacência)**

1. Seja n=∣V∣.
2. Inicialize a matriz M de adjacência de G, onde M[u][v]=1 se uv∈E(G), e 0 caso contrário.
3. Crie uma matriz M^2 de tamanho n×n, inicialmente zerada.
4. Para cada par de vértices (u,v):
   * Se u≠v:
     + Se M[u][v]=1, então M^2[u][v]=1.
     + Caso contrário, verifique se existe algum vértice w tal que M[u][w]=1 e M[w][v]=1.
       - Se sim, defina M^2[u][v]=1.
5. Retorne M^2.

#### **Algoritmo (Lista de Adjacência)**

1. Inicialize a lista de adjacência L para G.
2. Crie uma nova lista de adjacência L^2 para G^2.
3. Para cada vértice u∈V:
   * Adicione a L^2[u] todos os vértices v∈L[u] (arestas diretas).
   * Para cada w∈L[u], adicione os vértices v∈L[w] em L^2[u], exceto o próprio u.
   * Garanta que não há duplicatas.
4. Retorne L^2.

### **2. Complexidade**

#### **a. Matriz de Adjacência**

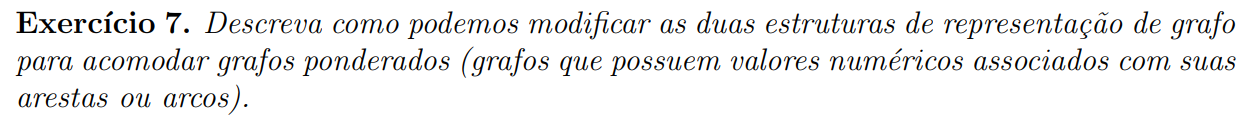
* O algoritmo percorre todos os pares de vértices (u,v) e, para cada par, verifica todos os n vértices intermediários w.
* **Complexidade**: O(n^3), onde nnn é o número de vértices.

#### **b. Lista de Adjacência**

* Para cada vértice uuu, percorremos seus vizinhos w (complexidade O(deg⁡(u)) e, para cada w, percorremos seus vizinhos v (complexidade O(deg⁡(w)).
* No pior caso, para grafos densos, a complexidade também pode atingir O(n^3), mas para grafos esparsos, a complexidade é aproximadamente proporcional ao número total de arestas e graus dos vértices:
  + **Complexidade**: O(n+m⋅Δ(G)), onde m é o número de arestas e Δ(G) é o grau máximo.

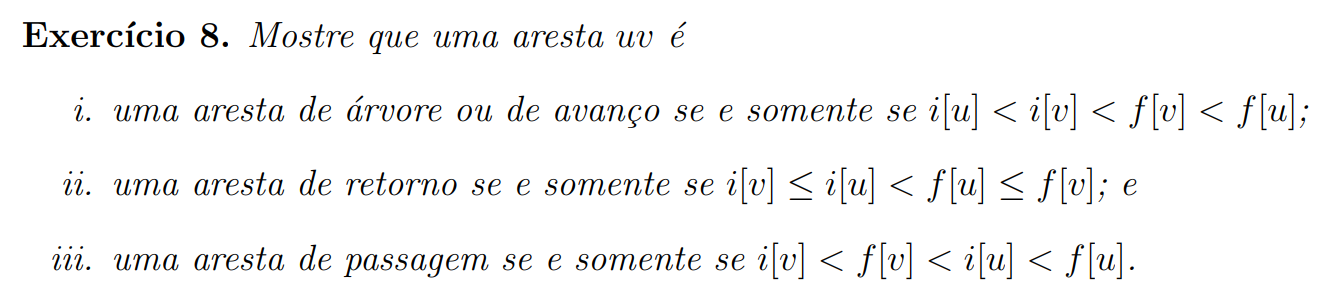
### **Resumo**

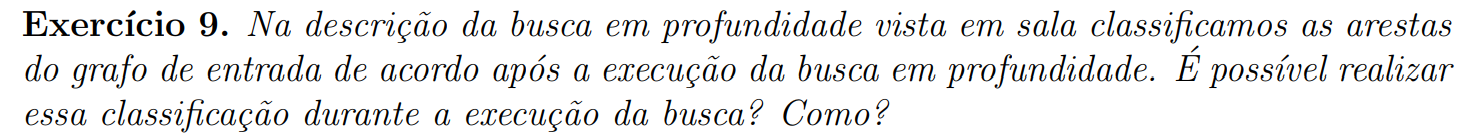
* **Matriz de Adjacência**: O(n^3).
* **Lista de Adjacência**: O(n+m⋅Δ(G)), com pior caso O(n^3).

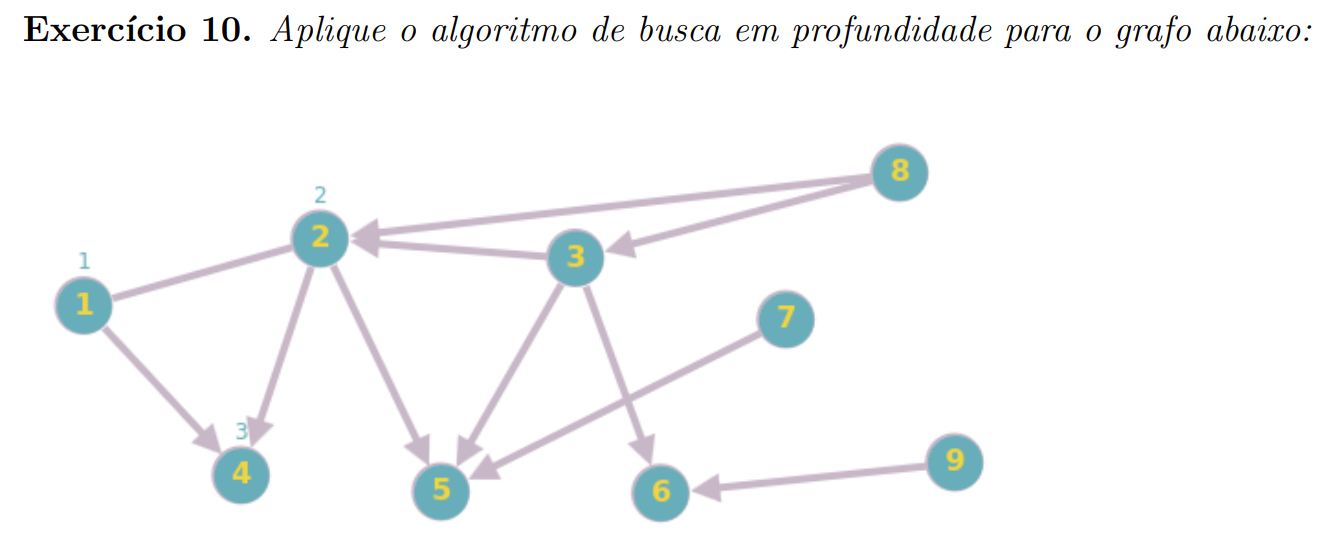


Matriz de adjacência: é mais fácil de se colocar os pesos. Ao invés de possuir o valor 1 quando o vértice i se interliga ao vértice j, e zero quando não se interliga, o valor 1 é substituído pelo peso da aresta.

Lista de adjacência: ao invés de apenas relacionar um único valor atrelado ao vértice v que se interliga ao vértice u, colocar um par de valores, o vértice v e o peso w interligado à u.







Matriz de adjacência

| i/j | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 2 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 3 | 0 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 0 |
| 4 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 5 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 6 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 7 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 8 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 9 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |

Lista de adjacência:

adj[1] = {2,4}

adj[2] = {1,4,5}

adj[3] = {2,5,6}

adj[4] = {}

adj[5] = {}

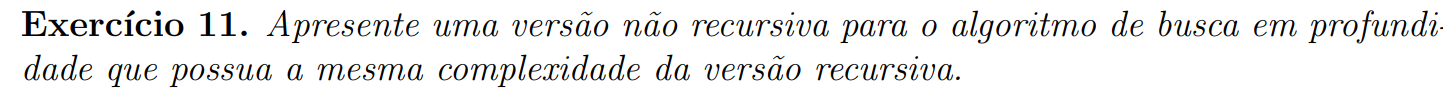
adj[6] = {}

adj[7] = {5}

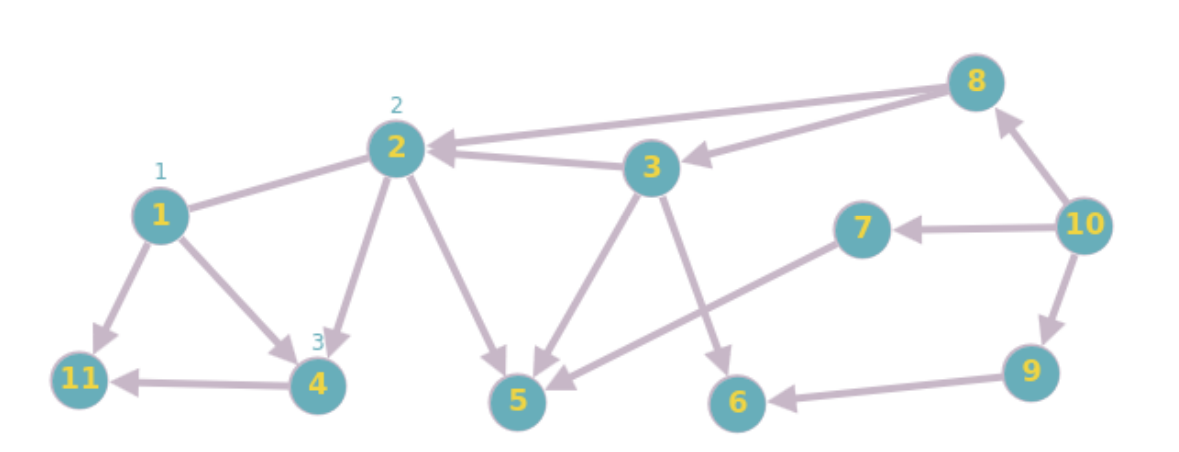
adj[8] = {2,3}

adj[9] = {6}

1. Começa em um vértice inicial.
2. Visita um dos seus vértices adjacentes não visitados.
3. Continua visitando os vértices adjacentes até que não haja mais vértices adjacentes não visitados.
4. Retrocede (volta um passo) e explora outros vértices adjacentes que ainda não foram visitados.
5. O processo se repete até que todos os vértices conectados ao inicial sejam visitados.







adj[1] = {2,4,11}

adj[2] = {1,4,5}

adj[3] = {2,5,6}

adj[4] = {11}

adj[5] = {}

adj[6] = {}

adj[7] = {5}

adj[8] = {2,3}

adj[9] = {6}

adj[10] = {7,8,9}

adj[11] = {}