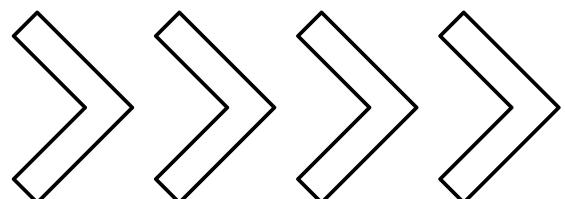


ÁRVORE GERADORA MÍNIMA APLICADA À ANÁLISE DE REDES CRIMINOSAS



Alunos: Wagner Silva e Wagner Lucena
Professor: Leonardo Nogueira
Disciplina: PAA

Introdução

AGM e a Complexidade Criminal

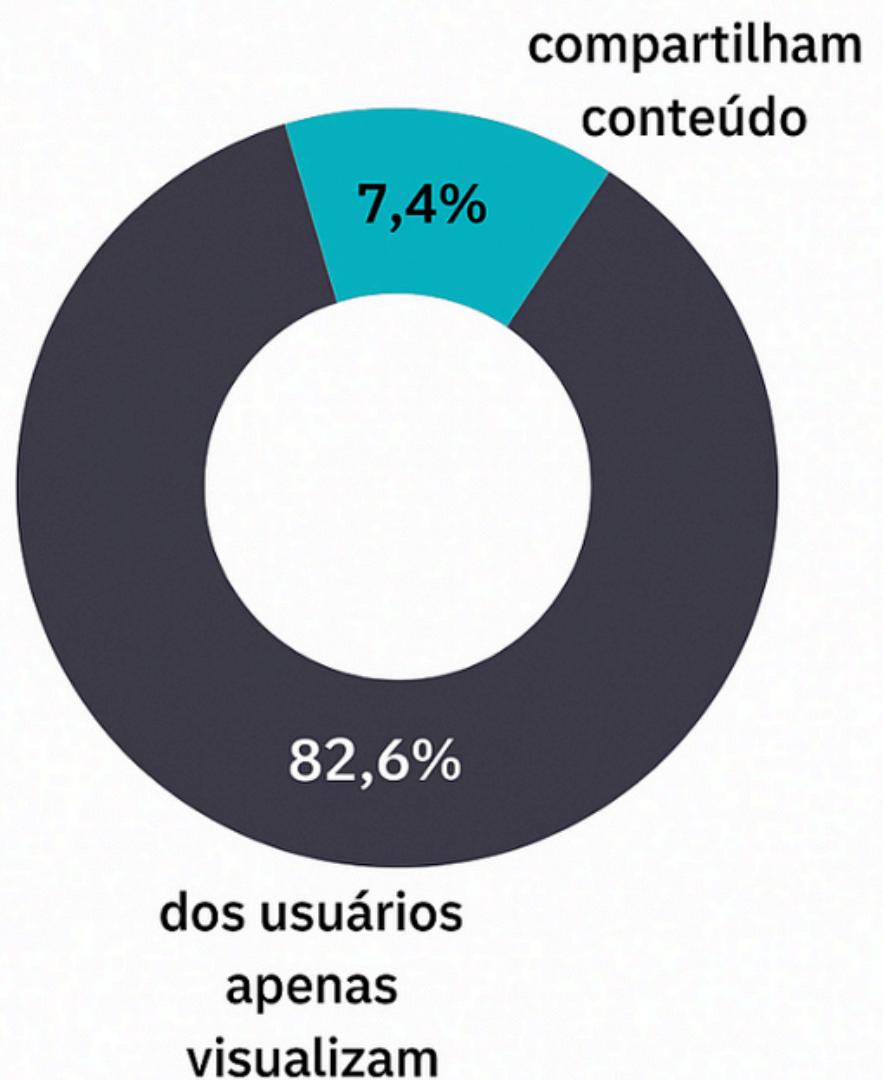
- Redes criminosas são densas e complexas.
- Difícil identificar quem realmente sustenta o sistema.
- AGM aplicada para reduzir ruído e destacar alvos estratégicos.

Problema

Operação Darknet



- Investigação sobre fórum de pornografia infantil na Tor
- Rede com 10.407 usuários em 11 países
- Objetivo: identificar o núcleo ativo e minimizar o ruído.

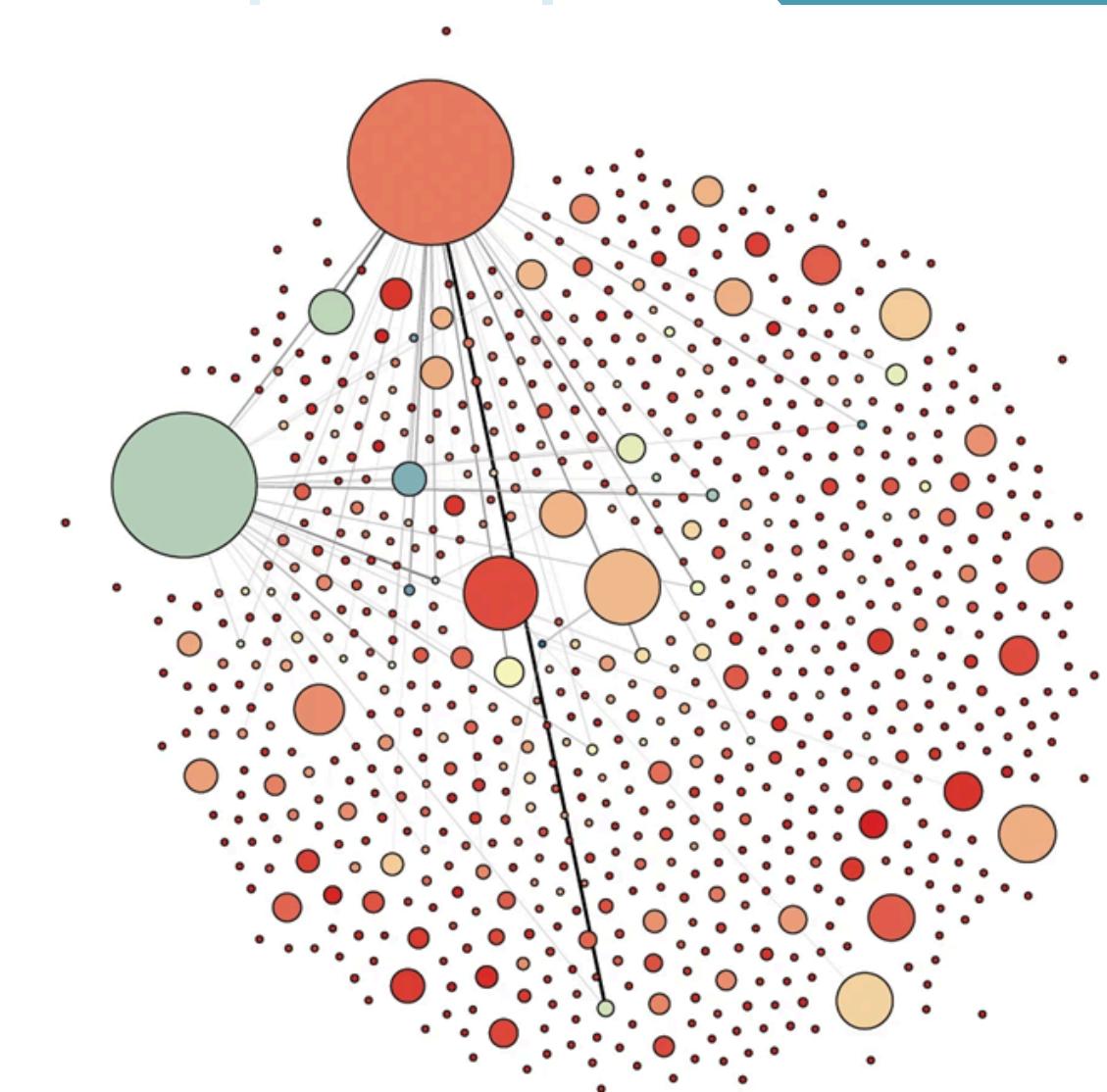


Problema

Operação Darknet



- Rede densa e resiliente com 766 criminosos ativos.
- A operação derrubou 60% das visualizações ao prender alvos estratégicos.



Conceitos

Árvore Geradora Mínima (AGM)

- **Entrada:** Grafo conexo, não direcionado, ponderado
 $G = (V, E)$, com pesos reais nas arestas $w : E \rightarrow \mathbb{R}$.
- **Objetivo:** Encontrar um subconjunto de arestas $T \subseteq E$, tal que:
 (V, T) forma uma árvore (conecta todos os vértices sem ciclos),
O custo total $w(T) = \sum_{(u,v) \in T} w(u, v)$ seja **mínimo**.

Conceitos

Árvore Geradora Mínima (AGM)

- Se G é conexo, então ele sempre possui uma AGM.
- Uma AGM de um grafo com $|V| = n$ vértices tem exatamente $n-1$ arestas.
- A AGM é única se os pesos das arestas são todos distintos.

Algoritmo

Modelagem do Problema para AGM

- Rede original é dirigida; AGM exige grafo não-dirigido.
- Foco no núcleo de 766 usuários que compartilhavam os conteúdos.
- Pesos: inverso da força da ligação.

Exemplo de pesos (simples):

Ligaçāo	Peso (custo)
Frequentam o mesmo fórum	3
Comunicação direta	2
Troca de arquivos ilegais	1

Algoritmo

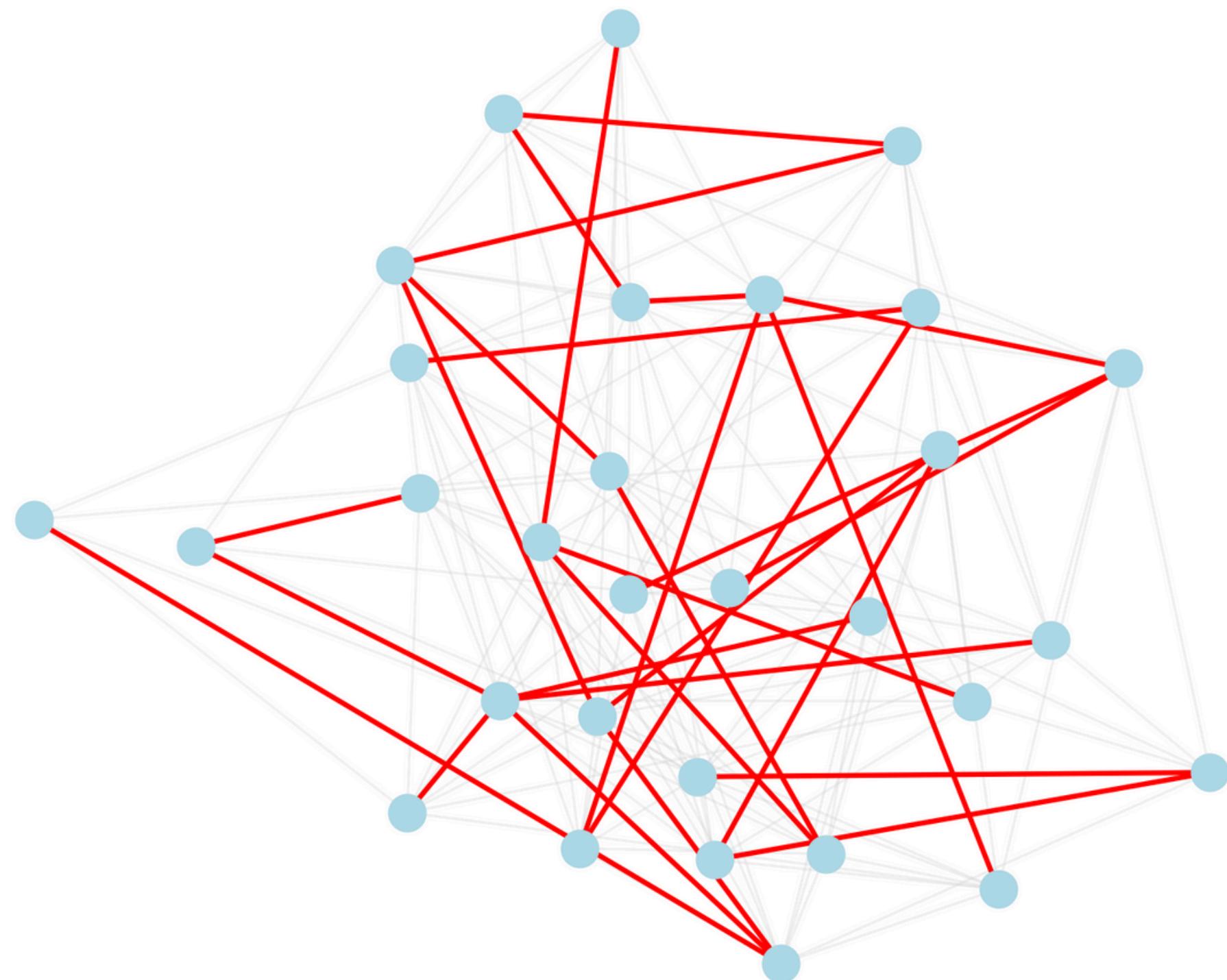
Modelagem do Problema para AGM

- Mantém todos conectados só pelos vínculos mais fortes.
- Remove redundâncias, facilita a leitura e a continuidade da investigação com foco nos principais alvos.
- Elimina redundâncias → deixa só os vínculos mínimos.
- Permanecem os visualizadores mais engajados.
- Justificativa: destacar conexões estratégicas entre criminosos.

Algoritmo

Modelagem do Problema para AGM

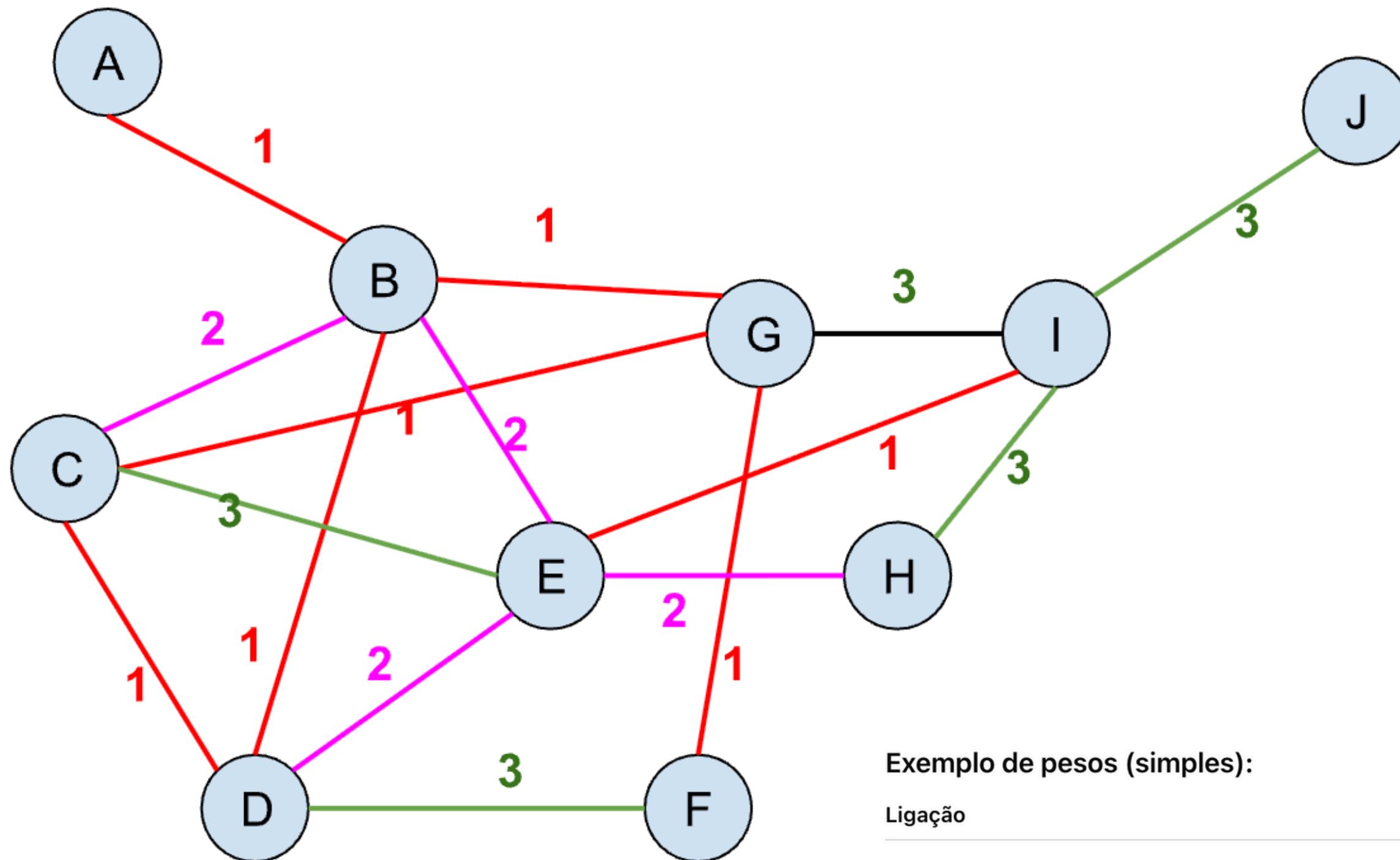
Rede reduzida pela MST (espinha dorsal de conexões fortes)



Algoritmo Kruskal

- É um algoritmo guloso usado para resolver o problema da AGM
- Estratégia: Sempre adiciona a aresta de menor peso que conecta dois componentes distintos, desde que não formem ciclos.
- Usa a estrutura de Disjoint-Set (Union-Find) para manter os componentes disjuntos.
- Complexidade: $O(E \log V)$.

Algoritmo Kruskal



Exemplo de pesos (simples):

Ligações

Frequentam o mesmo fórum

Comunicação direta

Troca de arquivos ilegais

Peso (custo)

3

2

1

Algoritmo Kruskal

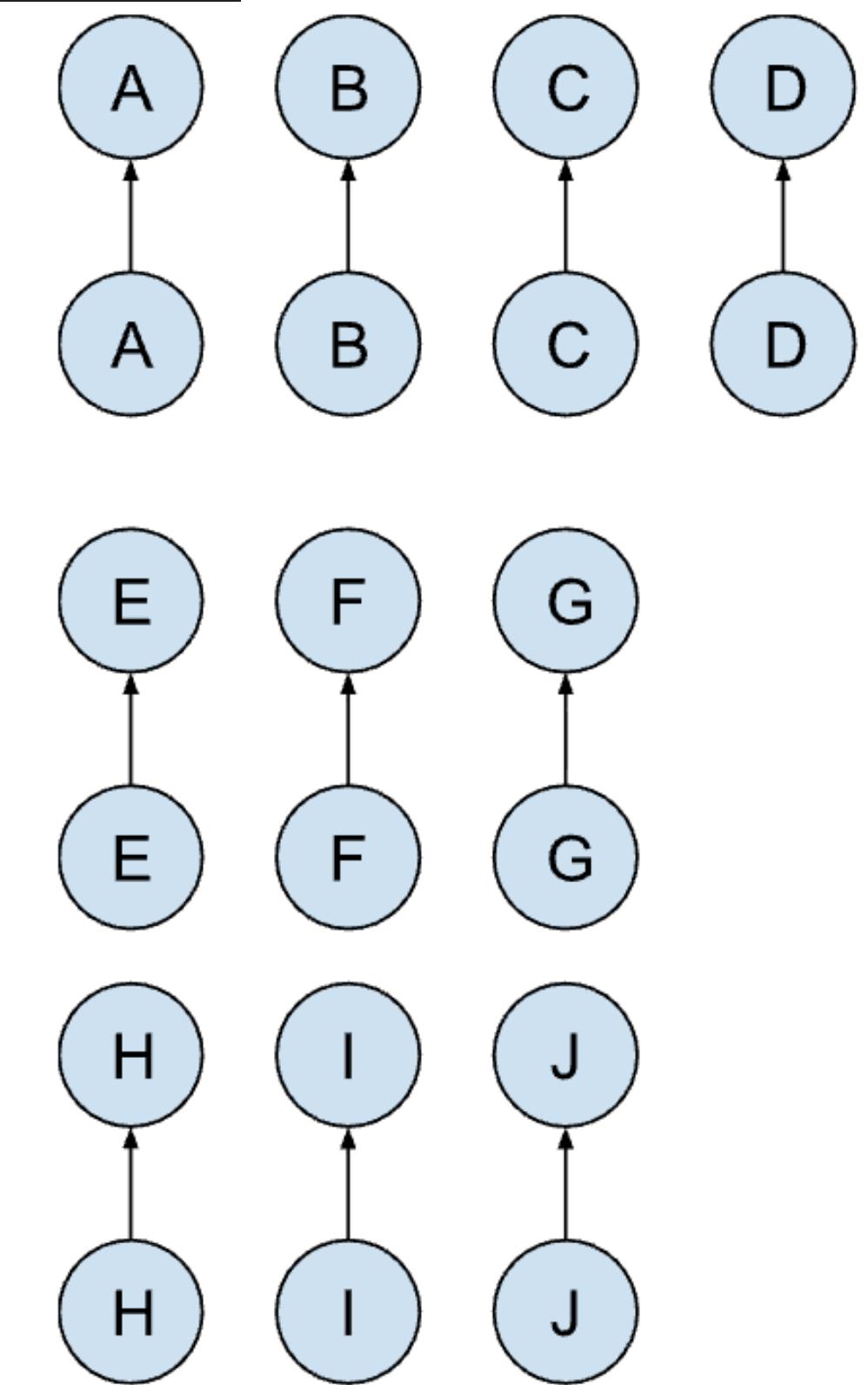
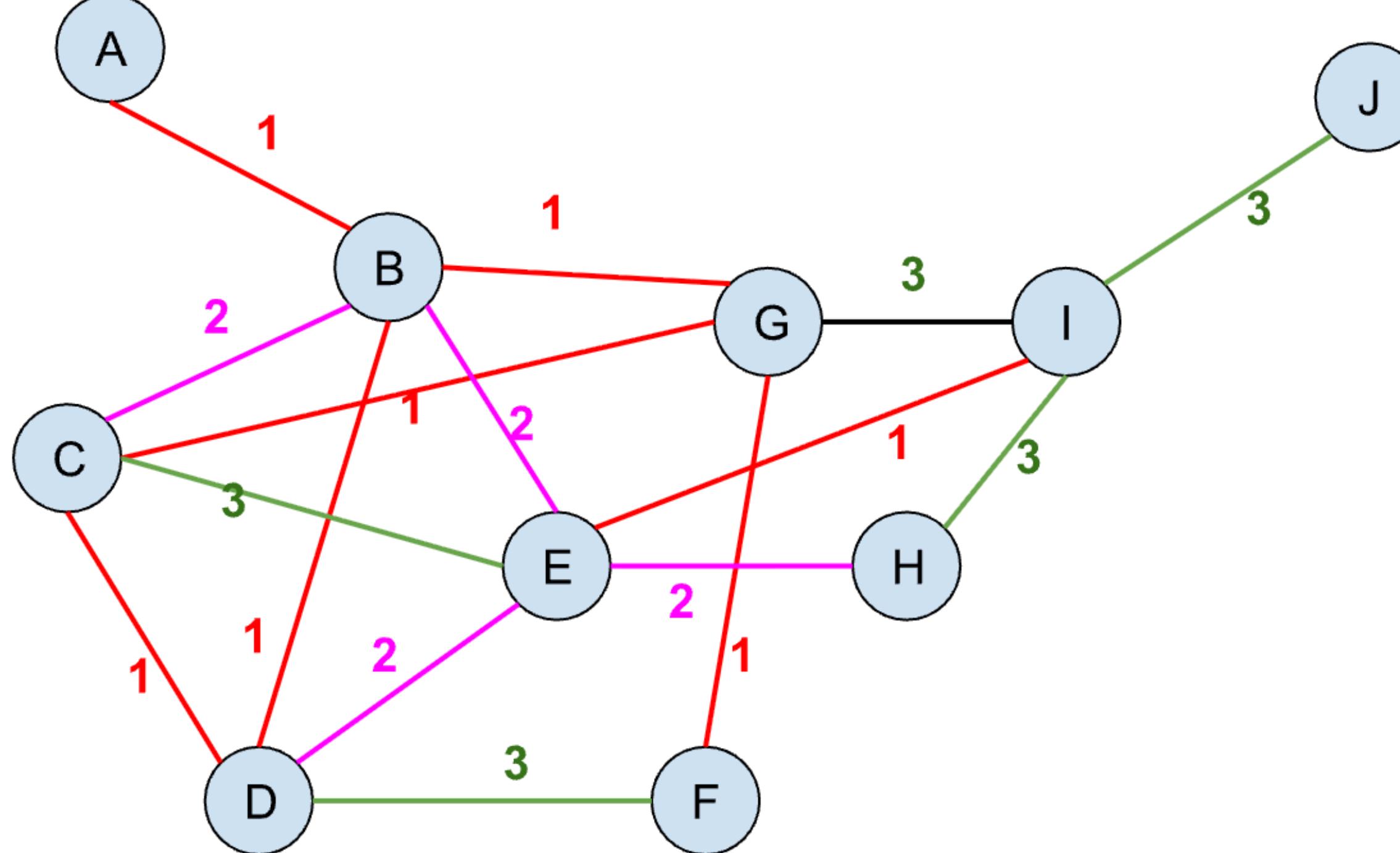
```
Vertex = str
Edge = Tuple[Vertex, Vertex]
Graph = Tuple[Set[Vertex], List[Edge]]
WeightFunction = Dict[Edge, float]

def kruskal(g: Graph, w: WeightFunction) -> List[Edge]:
```

- **MAKE-SET(v)**: Cria um conjunto inicial para cada vértice v com um único elemento v.

```
parent = {v: v for v in g[0]}
rank = {v: 0 for v in g[0]}
```

```
parent = {v: v for v in g[0]}  
rank = {v: 0 for v in g[0]}
```



Algoritmo Kruskal

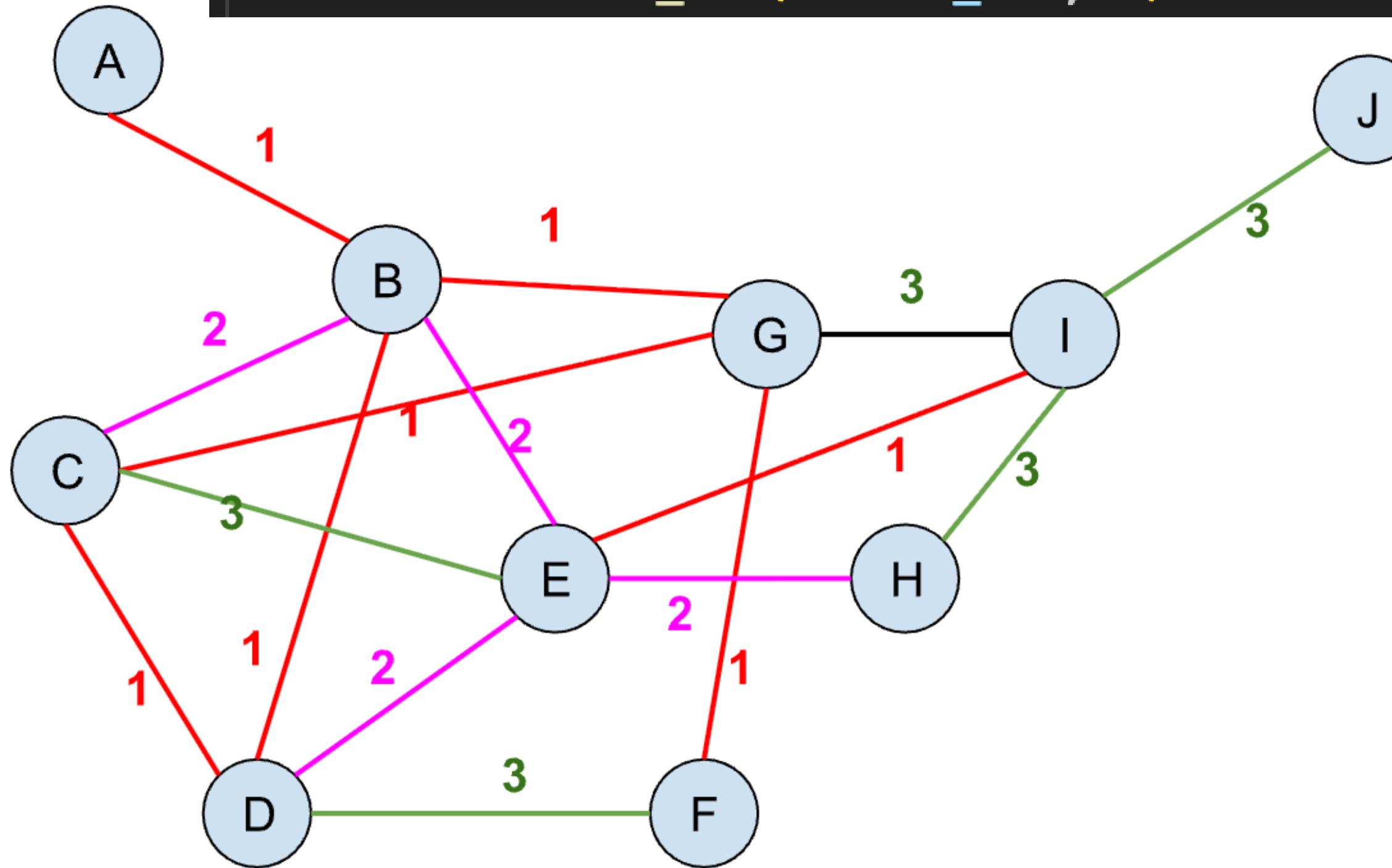
```
Vertex = str
Edge = Tuple[Vertex, Vertex]
Graph = Tuple[Set[Vertex], List[Edge]]
WeightFunction = Dict[Edge, float]

def kruskal(g: Graph, w: WeightFunction) -> List[Edge]:
```

- **FIND-SET(u)**: Verificar se dois vértices pertencem ao mesmo conjunto (árvore) comparando seus representantes (raizes).

```
sorted_edges = sorted(g[1], key=lambda edge: w[edge])
for edge in sorted_edges:
    u, v = edge
    rootU = find_set(vertex_set, u)
    rootV = find_set(vertex_set, v)
```

```
sorted_edges = sorted(g[1], key=lambda edge: w[edge])
for edge in sorted_edges:
    u, v = edge
    rootU = find_set(vertex_set, u)
    rootV = find_set(vertex_set, v)
```



(A,B,1)
(B,D,1)
(B,G,1)
(C,D,1)
(C,G,1)
(E,I,1)
(F,G,1)

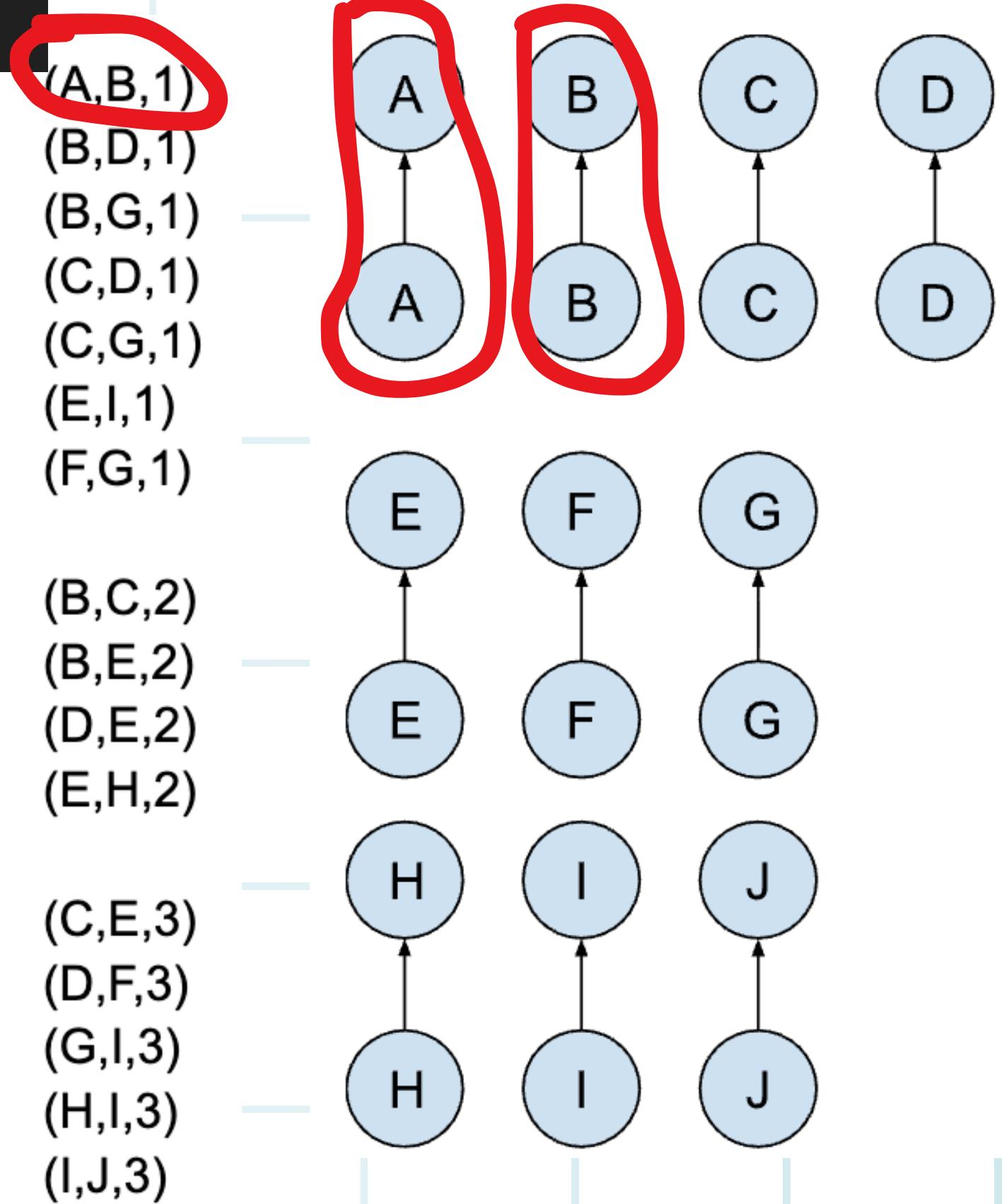
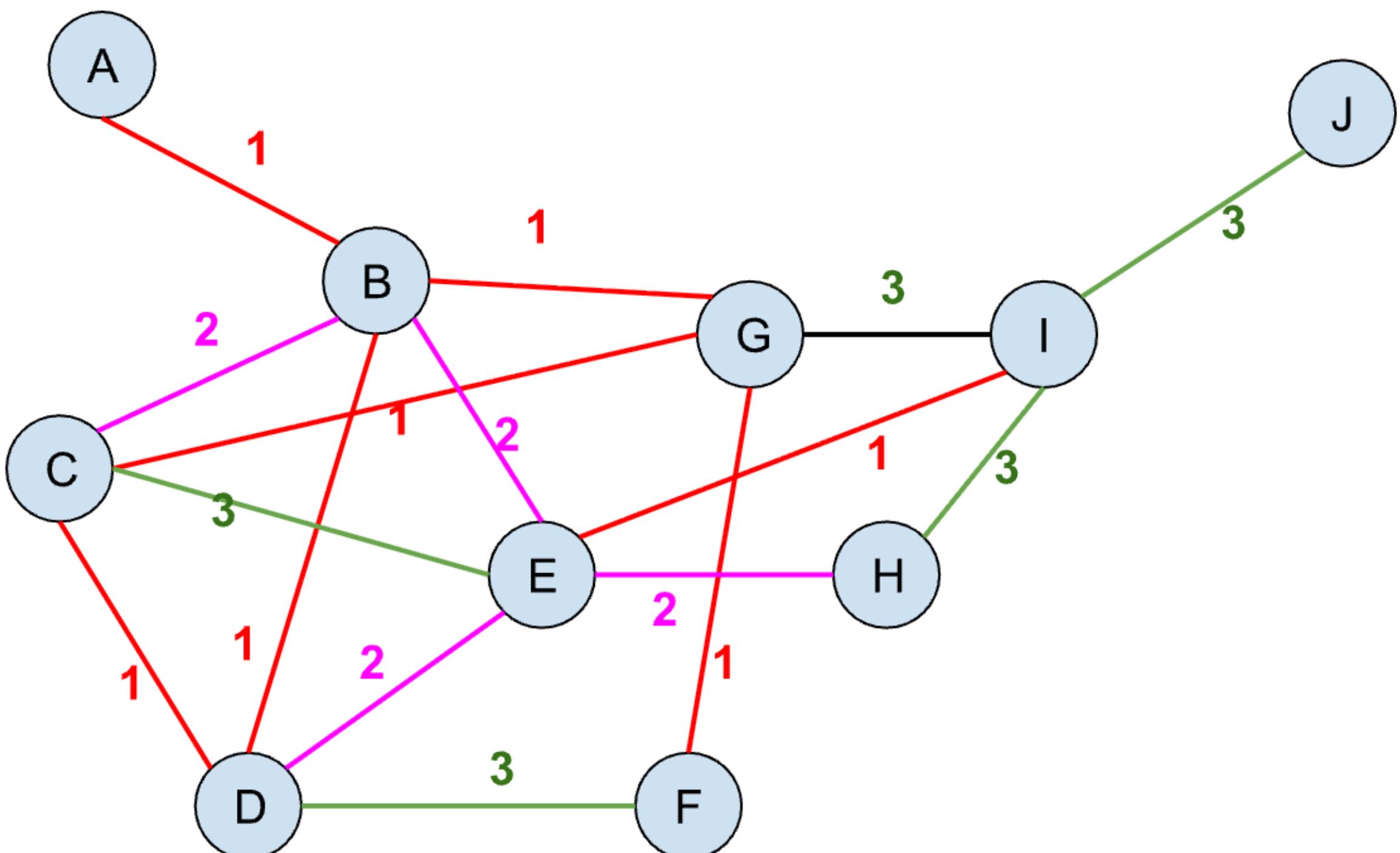
(B,C,2)
(B,E,2)
(D,E,2)
(E,H,2)

(C,E,3)
(D,F,3)
(G,I,3)
(H,I,3)
(I,J,3)

```

sorted_edges = sorted(g[1], key=lambda edge: w[edge])
for edge in sorted_edges:
    u, v = edge
    rootU = find_set(vertex_set, u)
    rootV = find_set(vertex_set, v)

```



Algoritmo Kruskal

```
Vertex = str
Edge = Tuple[Vertex, Vertex]
Graph = Tuple[Set[Vertex], List[Edge]]
WeightFunction = Dict[Edge, float]

def kruskal(g: Graph, w: WeightFunction) -> List[Edge]:
```

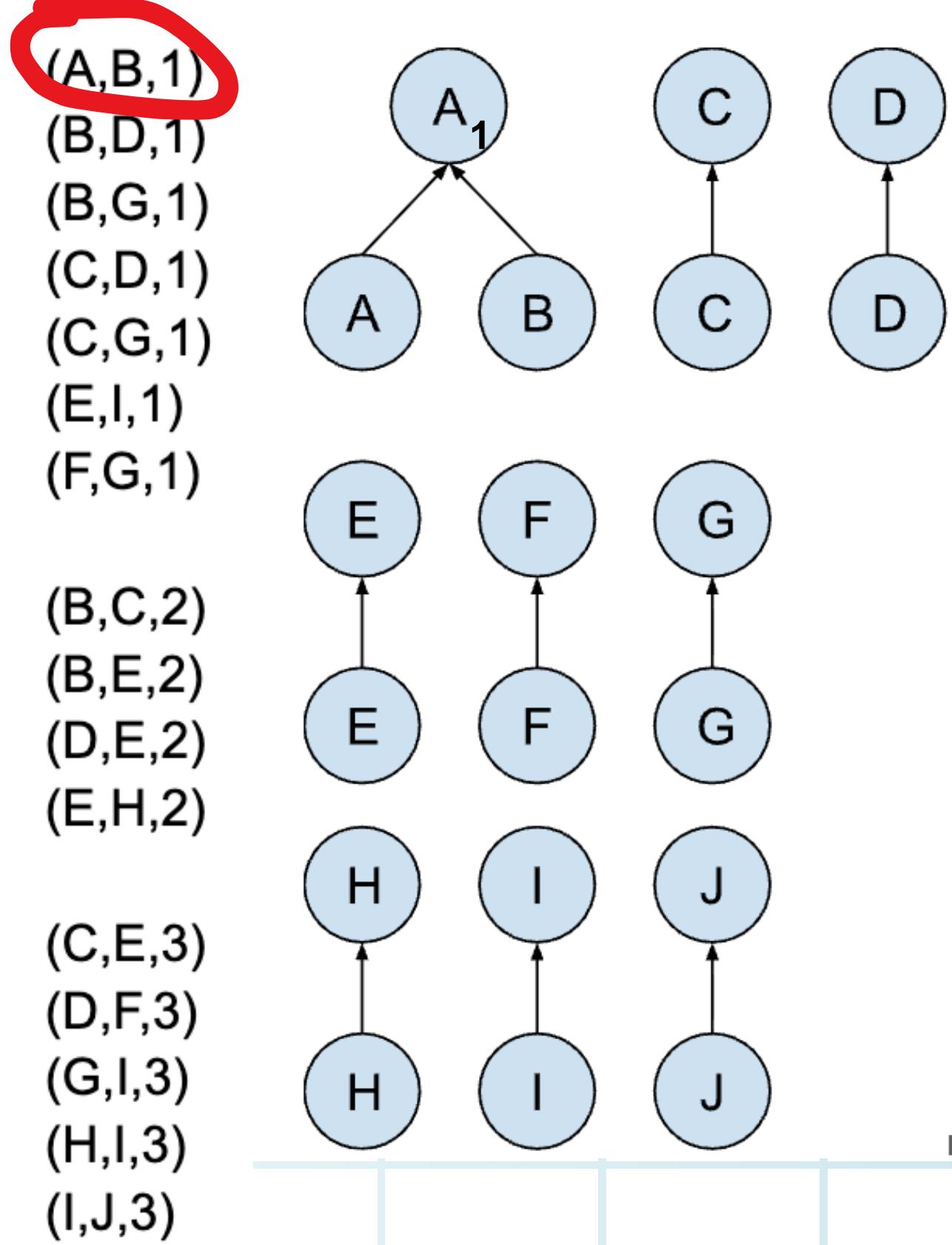
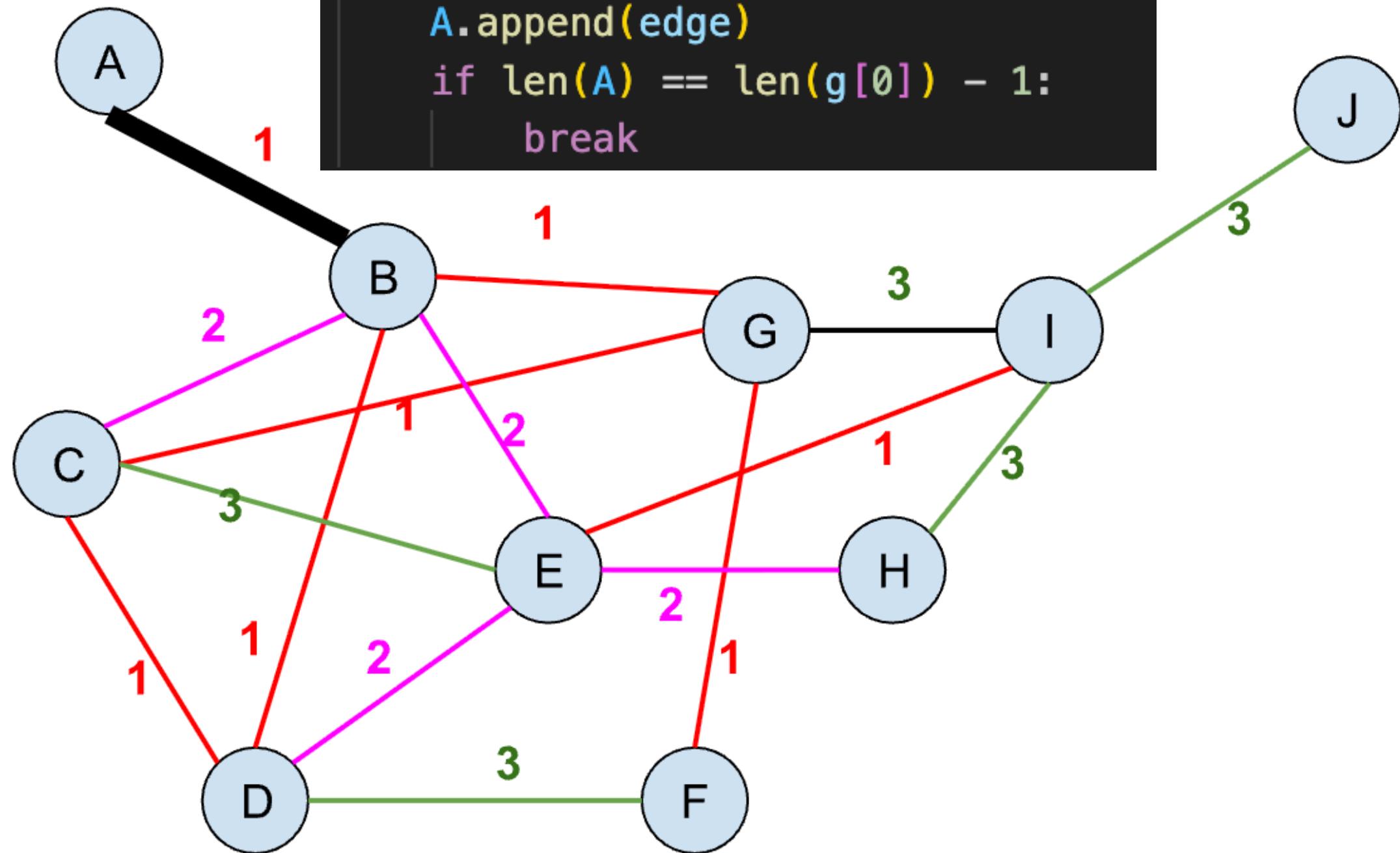
- **UNION(u,v)**: Une os conjuntos de u e v em uma única árvore, apenas se pertencerem a componentes distintas, evitando a formação de ciclos.

```
if rootU != rootV:  
    if rank[rootU] < rank[rootV]:  
        vertex_set[rootU] = rootV  
    elif rank[rootU] > rank[rootV]:  
        vertex_set[rootV] = rootU  
    else:  
        vertex_set[rootV] = rootU  
        rank[rootU] += 1  
    A.append(edge)  
    if len(A) == len(g[0]) - 1:  
        break
```

```

if rootU != rootV:
    if rank[rootU] < rank[rootV]:
        vertex_set[rootU] = rootV
    elif rank[rootU] > rank[rootV]:
        vertex_set[rootV] = rootU
    else:
        vertex_set[rootV] = rootU
        rank[rootU] += 1
        A.append(edge)
        if len(A) == len(g[0]) - 1:
            break

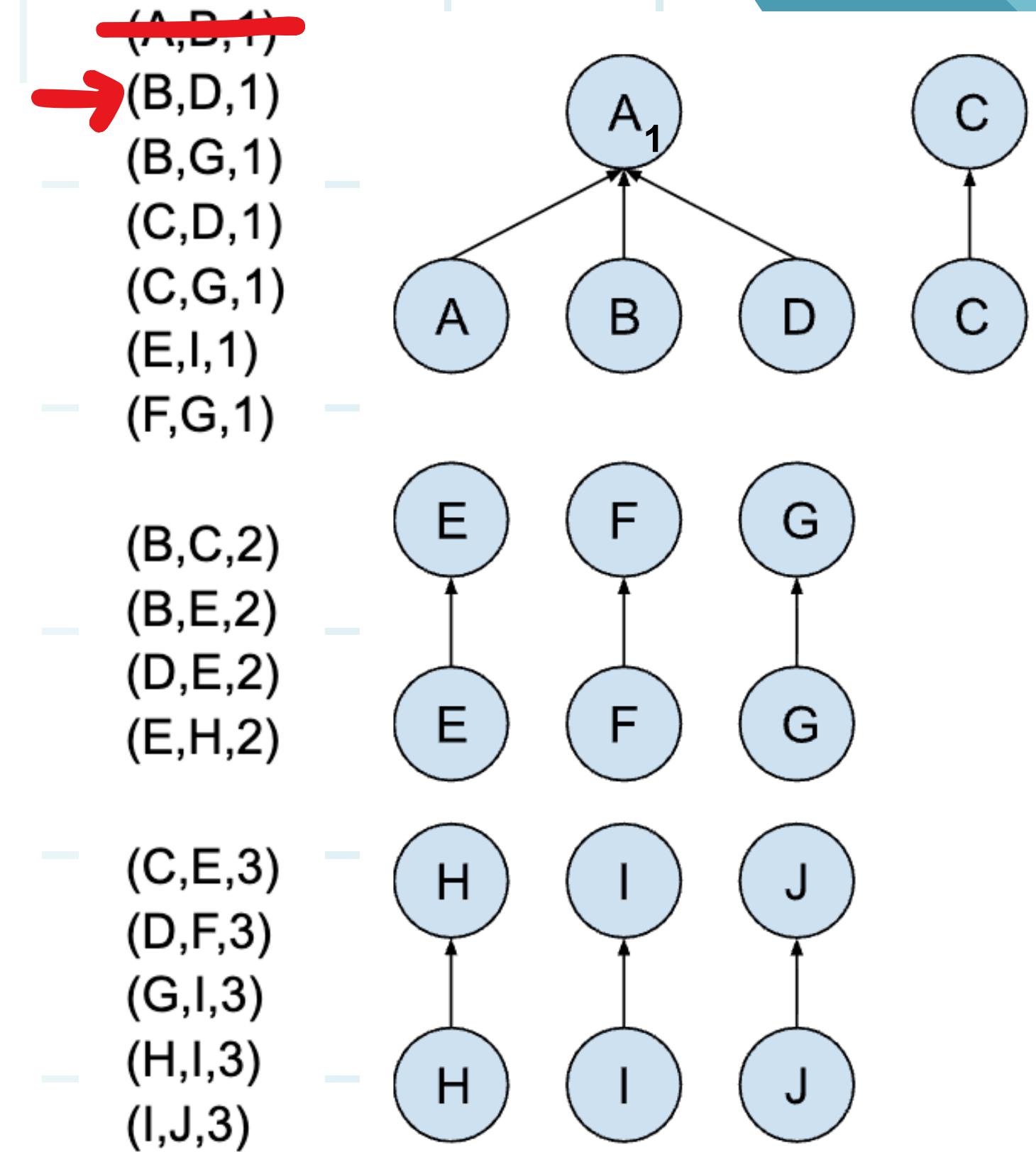
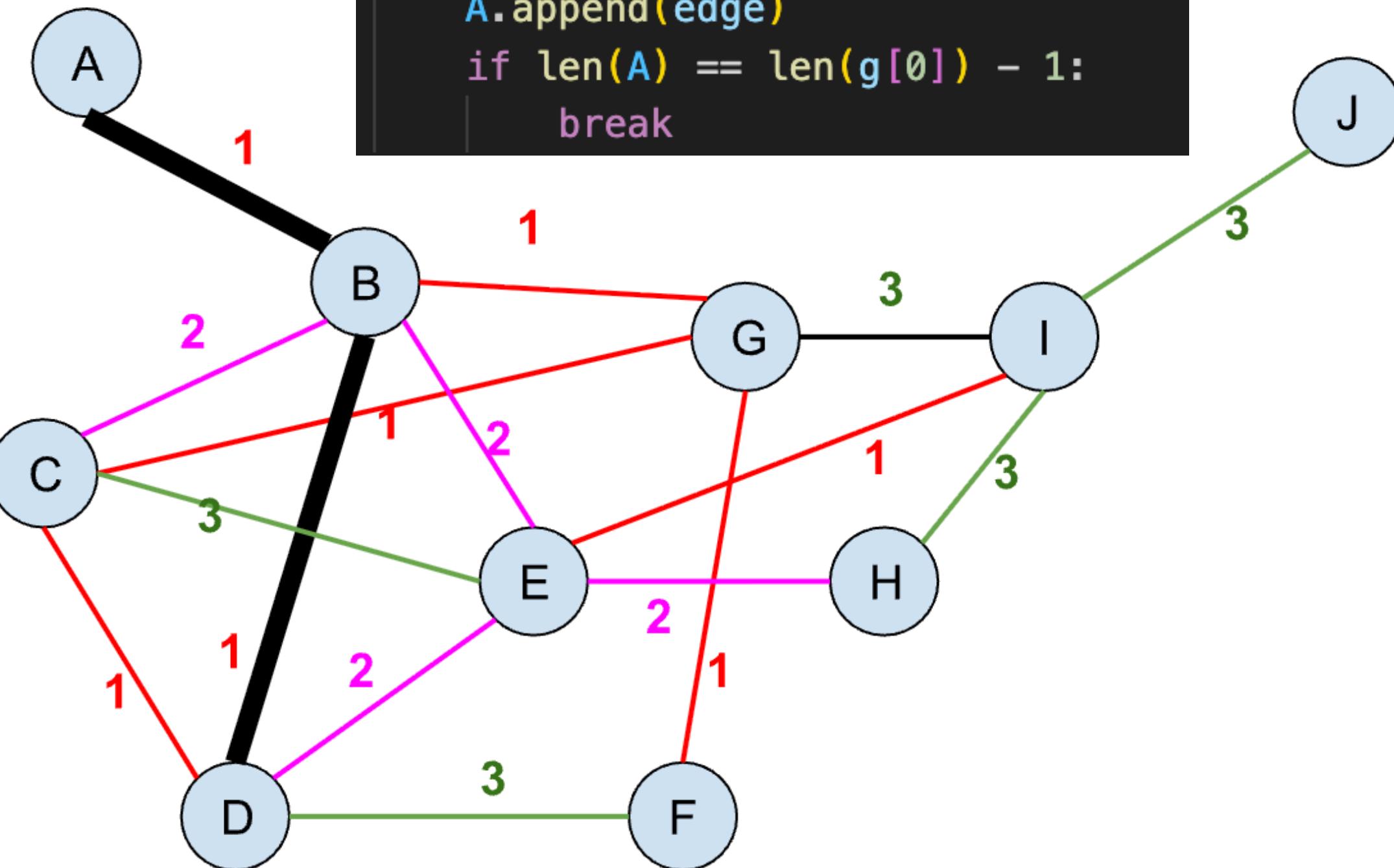
```



```

if rootU != rootV:
    if rank[rootU] < rank[rootV]:
        vertex_set[rootU] = rootV
    elif rank[rootU] > rank[rootV]:
        vertex_set[rootV] = rootU
    else:
        vertex_set[rootV] = rootU
        rank[rootU] += 1
    A.append(edge)
    if len(A) == len(g[0]) - 1:
        break

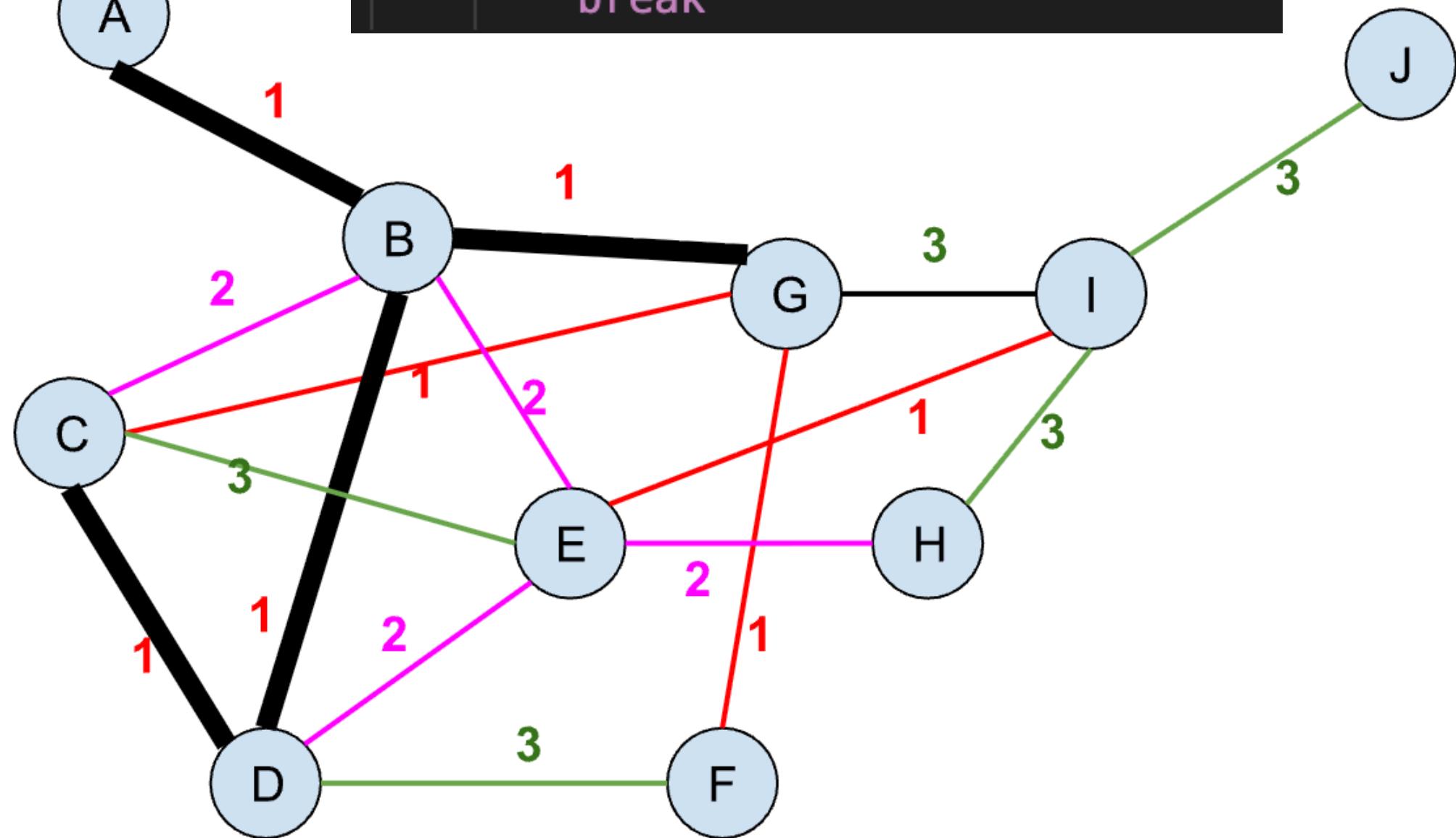
```



```

if rootU != rootV:
    if rank[rootU] < rank[rootV]:
        vertex_set[rootU] = rootV
    elif rank[rootU] > rank[rootV]:
        vertex_set[rootV] = rootU
    else:
        vertex_set[rootV] = rootU
        rank[rootU] += 1
    A.append(edge)
    if len(A) == len(g[0]) - 1:
        break

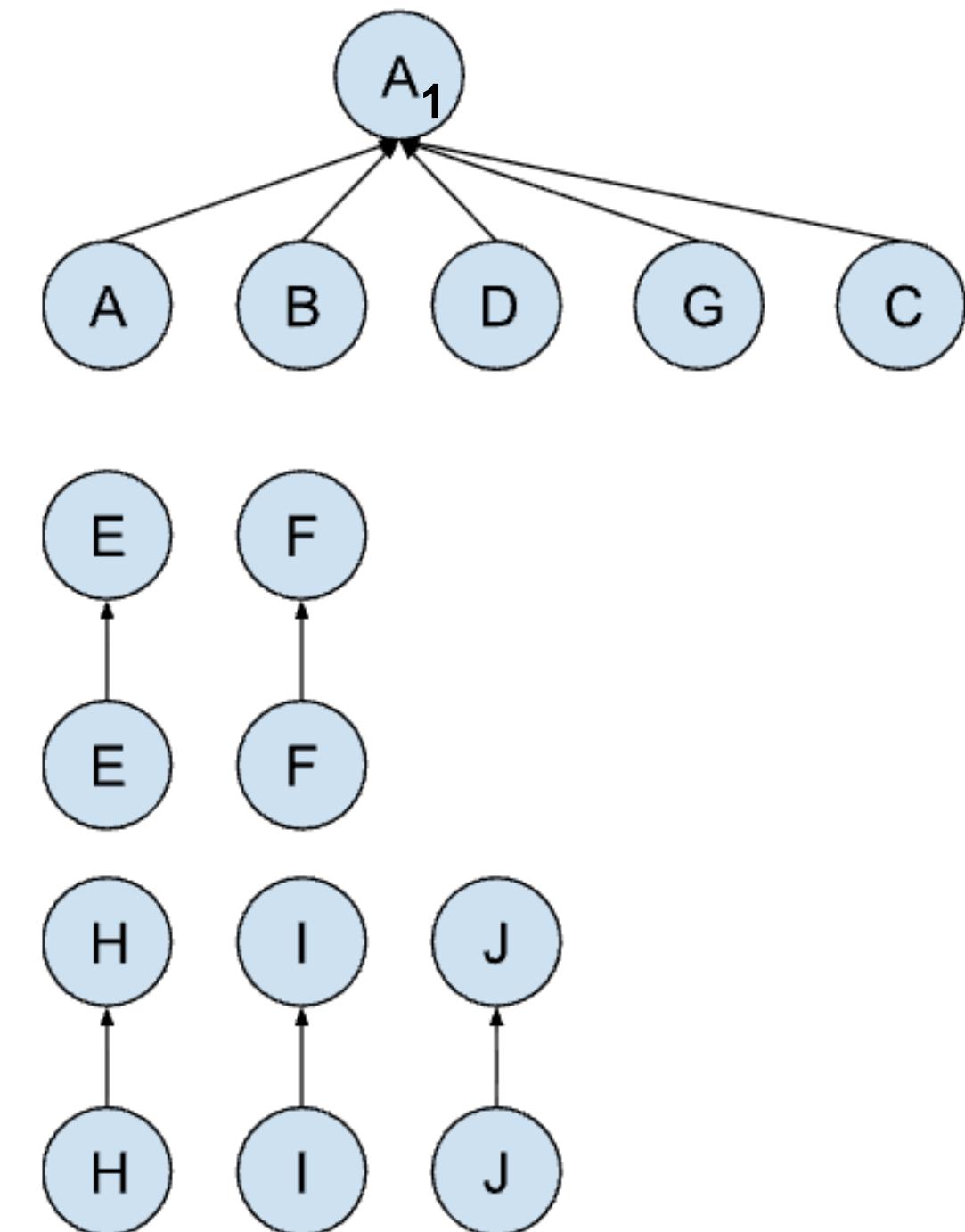
```



~~(A,B,1)~~
~~(B,D,1)~~
~~(B,C,1)~~
→ (C,D,1)
(C,G,1)
(E,I,1)
(F,G,1)

(B,C,2)
(B,E,2)
(D,E,2)
(E,H,2)

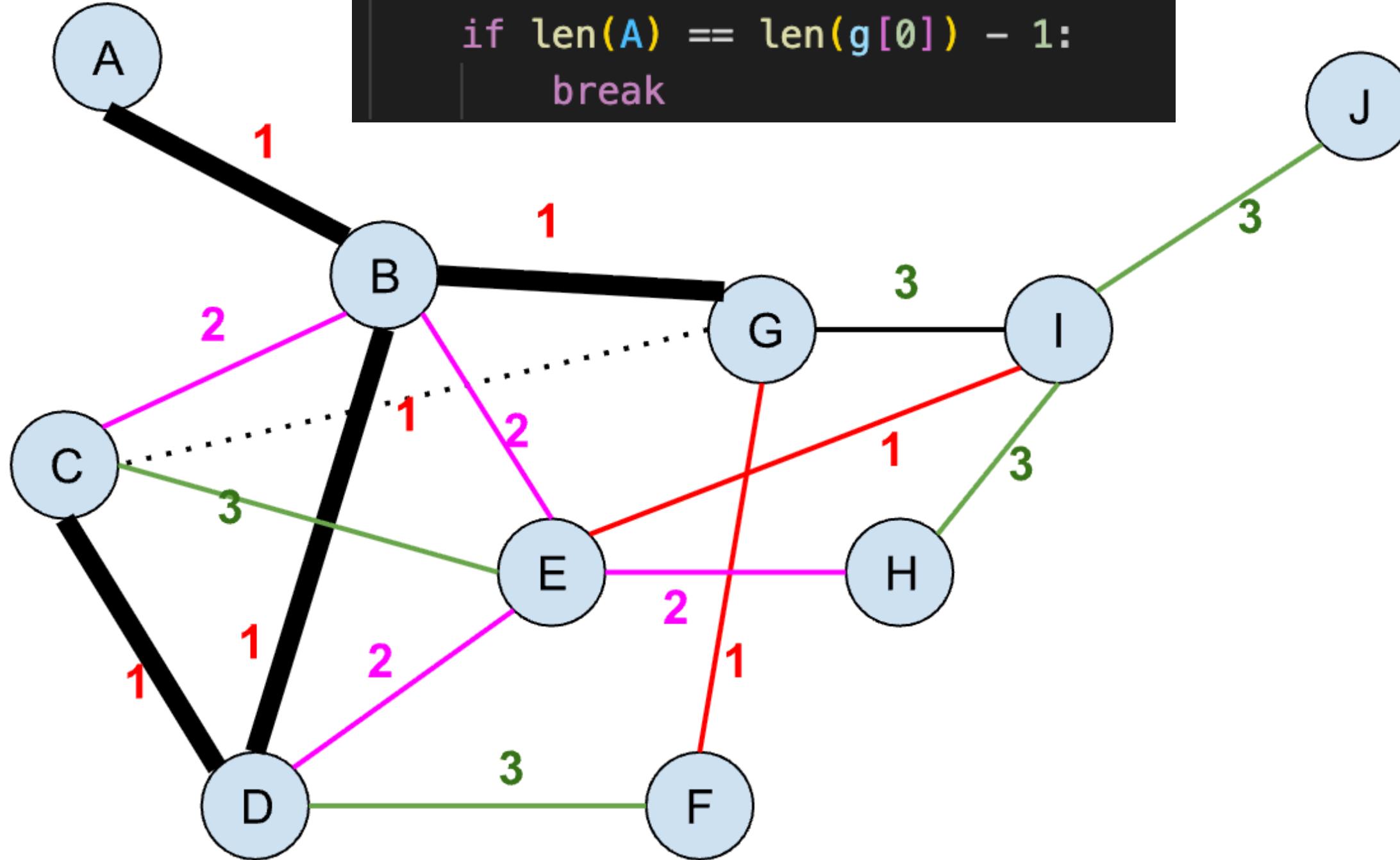
(C,E,3)
(D,F,3)
(G,I,3)
(H,I,3)
(I,J,3)



```

if rootU != rootV:
    if rank[rootU] < rank[rootV]:
        vertex_set[rootU] = rootV
    elif rank[rootU] > rank[rootV]:
        vertex_set[rootV] = rootU
    else:
        vertex_set[rootV] = rootU
        rank[rootU] += 1
A.append(edge)
if len(A) == len(g[0]) - 1:
    break

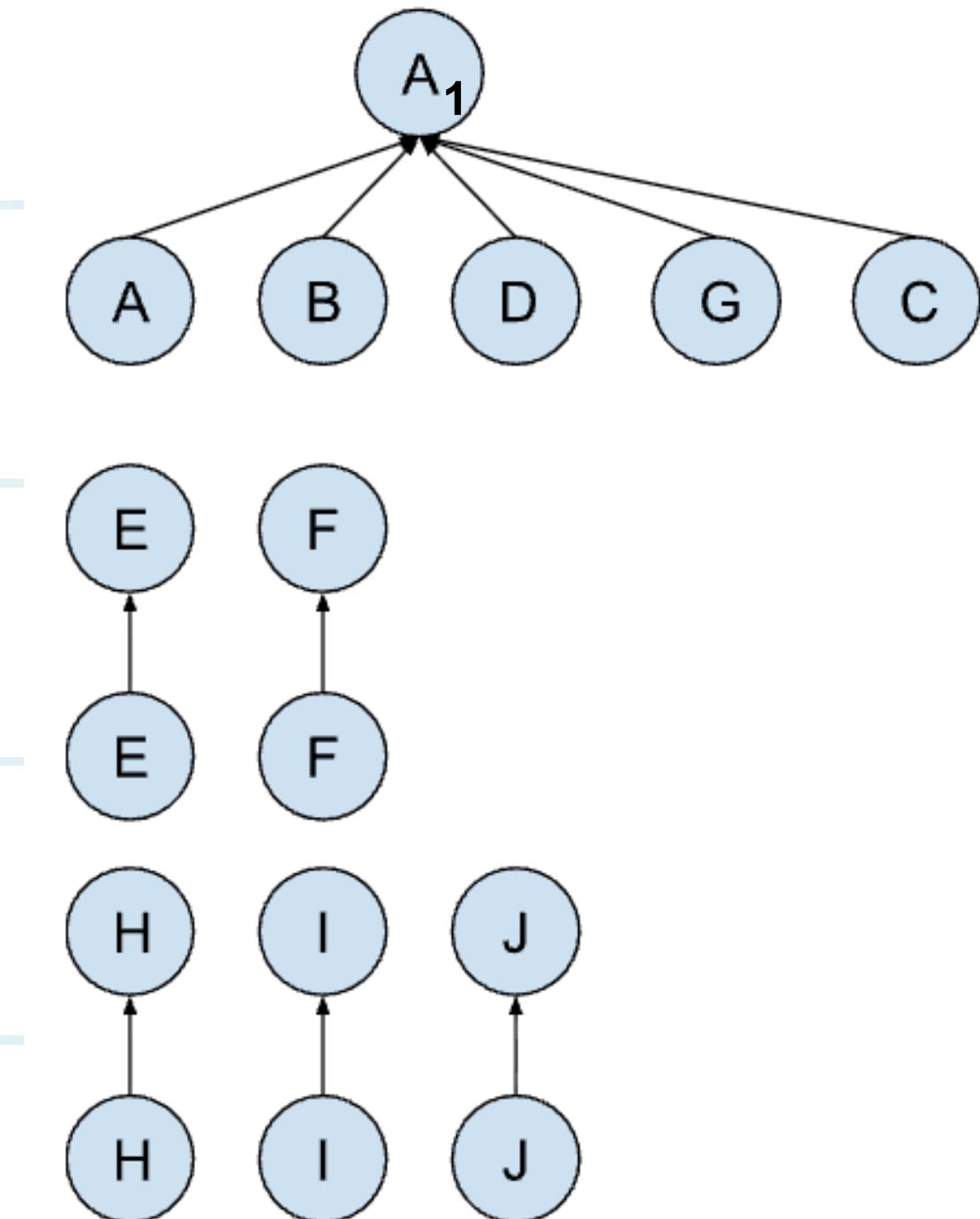
```



~~(A,B,1)~~
~~(B,D,1)~~
~~(B,C,1)~~
~~(C,D,1)~~
~~(C,G,1)~~
~~(E,I,1)~~
~~(F,G,1)~~

~~(B,C,2)~~
~~(B,E,2)~~
~~(D,E,2)~~
~~(E,H,2)~~

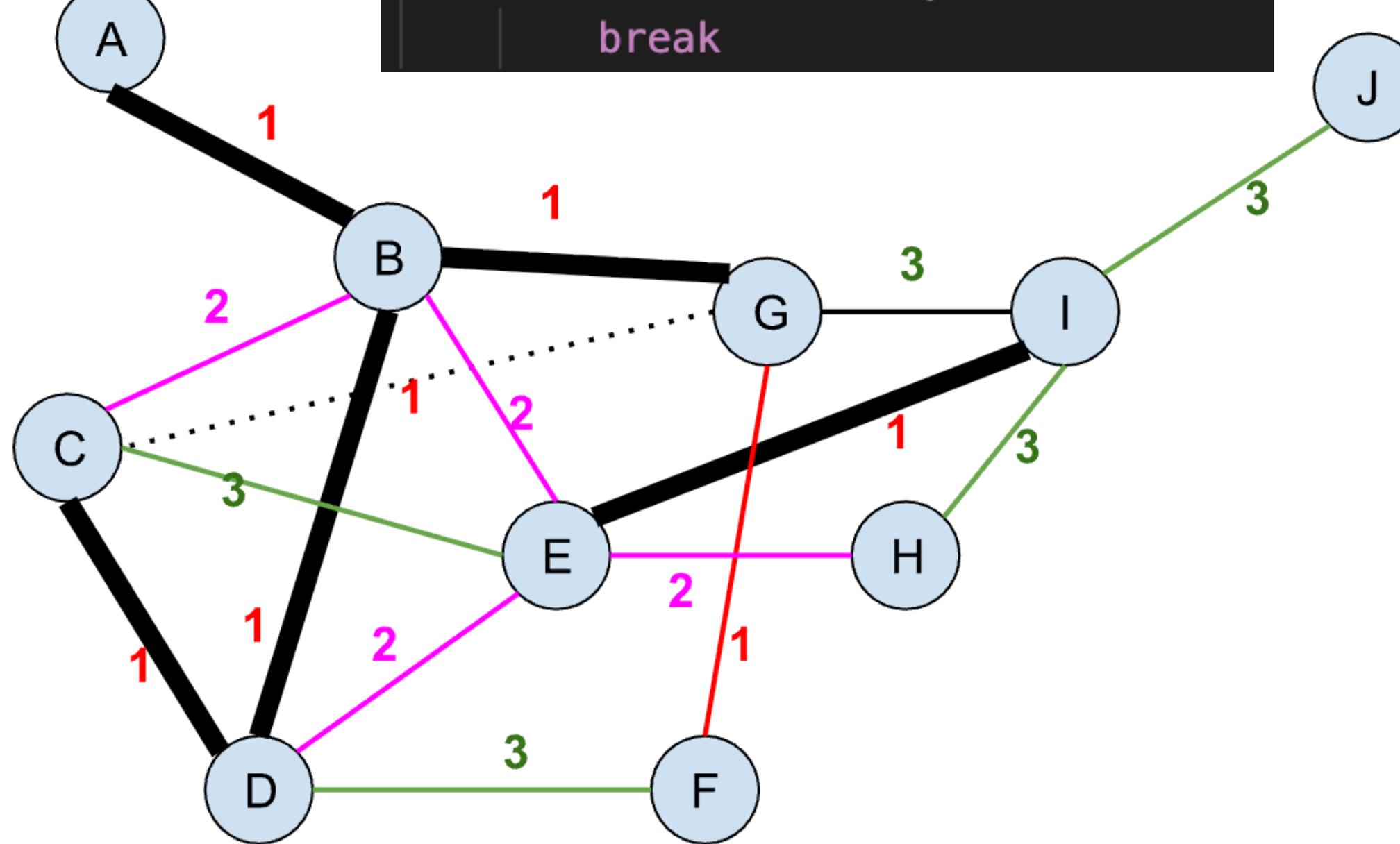
~~(C,E,3)~~
~~(D,F,3)~~
~~(G,I,3)~~
~~(H,I,3)~~
~~(I,J,3)~~



```

if rootU != rootV:
    if rank[rootU] < rank[rootV]:
        vertex_set[rootU] = rootV
    elif rank[rootU] > rank[rootV]:
        vertex_set[rootV] = rootU
    else:
        vertex_set[rootV] = rootU
        rank[rootU] += 1
    A.append(edge)
    if len(A) == len(g[0]) - 1:
        break

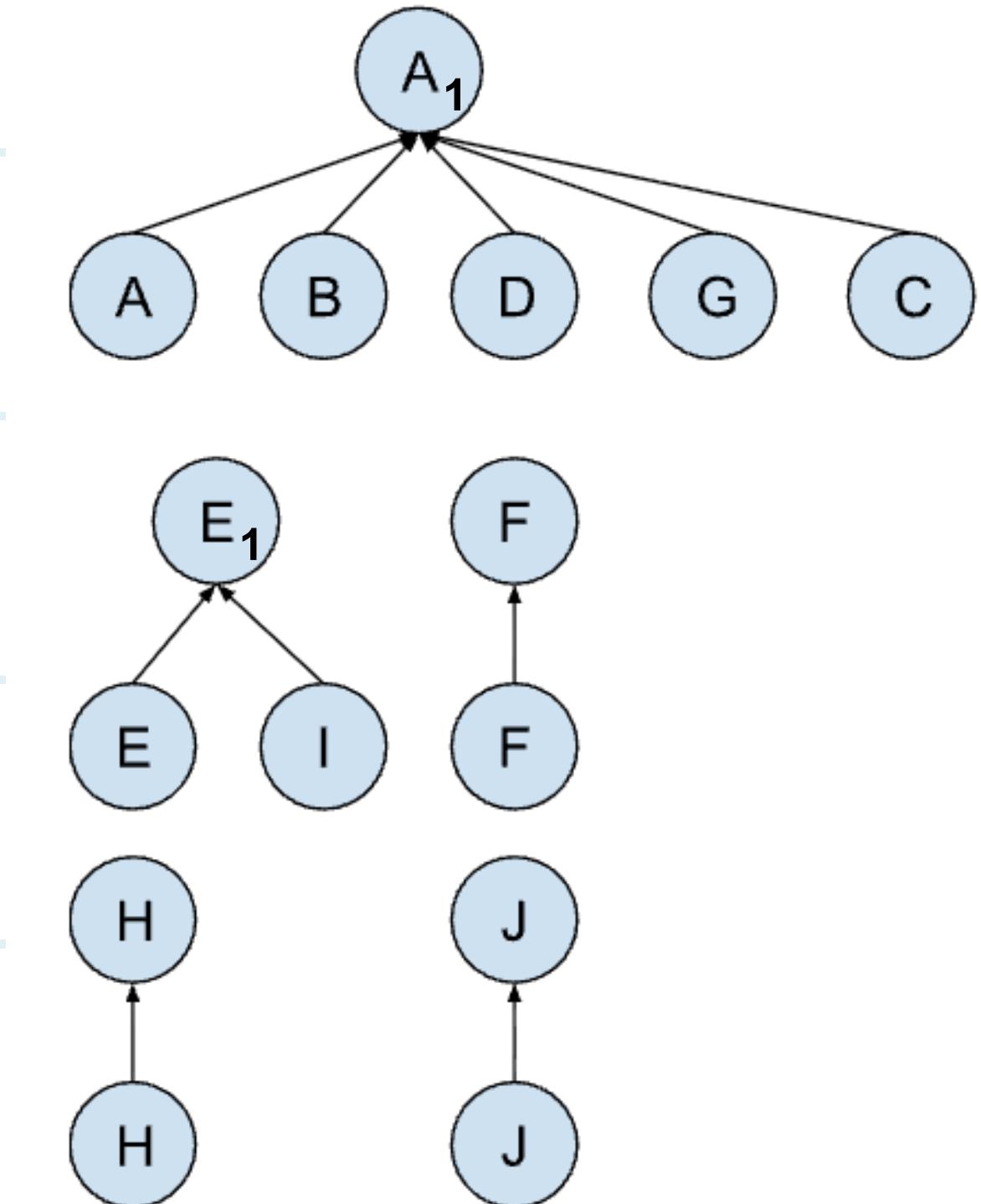
```



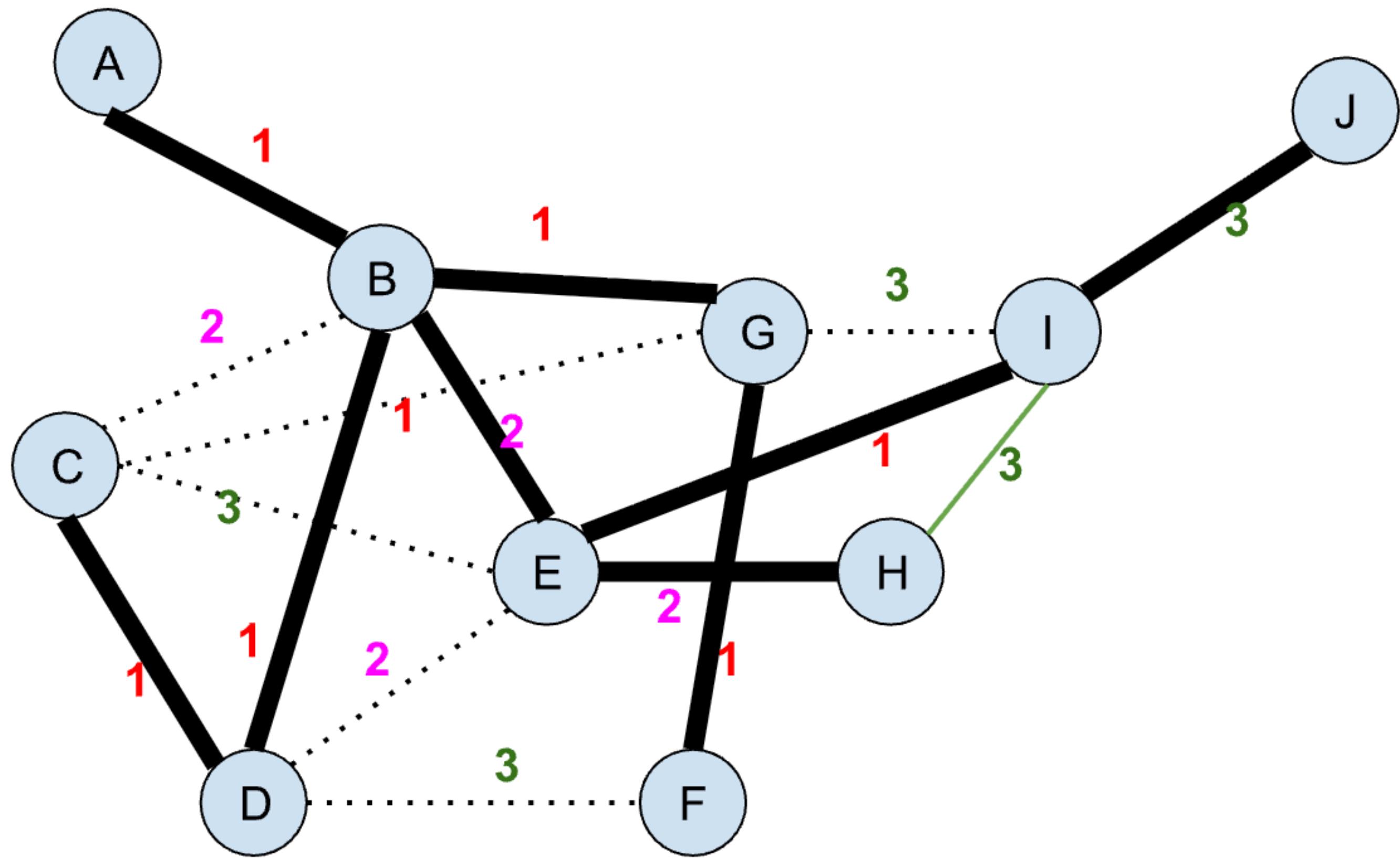
~~(A,B,1)~~
~~(B,D,1)~~
~~(B,C,1)~~
~~(C,D,1)~~
~~(C,G,1)~~
~~(E,I,1)~~
(F,G,1)

(B,C,2)
(B,E,2)
(D,E,2)
(E,H,2)

(C,E,3)
(D,F,3)
(G,I,3)
(H,I,3)
(I,J,3)



```
A.append(edge)
if len(A) == len(g[0]) - 1:
    break
```



Resultados Esperados

Aplicação da AGM

- AGM favoreceria a estratégia de inanição e alto impacto investigativo
- AGM revelaria a estrutura essencial da rede criminosa
- Apoiaria decisões estratégicas com foco em conexões críticas.
- Viabilizaria desmonte do núcleo com menos intervenções.
- Permitiria derrubar o núcleo com eficiência, reduzindo custo e aumentando impacto.