

# < OREON - UVA 1208 >

EN UN FUTURO APOCALÍPTICO LA HUMANIDAD SE VE REDUCIDA A CIUDADES.

CADA CIUDAD PRODUCE UN MATERIAL ÚNICO, NECESARIO PARA MANTENER LO QUE QUEDA DE LA HUMANIDAD CON VIDA.

TODAS LAS CIUDADES SE ENCUENTRAN CONECTADAS ENTRE SI POR TÚNELES.

CUANDO LOS TÚNELES SE VEN BAJO ATAQUE, SE DETERMINA LA CANTIDAD DE PERSONAL NECESARIA PARA DEFENDER CADA UNO DE ELLOS.

SE DEBE ENCONTRAR EL MÍNIMO PERSONAL REQUERIDO PARA MANTENER TODAS LAS CIUDADES CONECTADAS.

## PLANTEO TEORICO

El problema se reduce a:

“Teniendo un grafo conexo pesado no dirigido, encontrar las aristas necesarias para que el grafo siga estando conectado, de forma que la suma de los pesos de dichas aristas sea mínima”

o en otras palabras encontrar el MST del grafo dado.

# SOLUCIONES APLICABLES

## REALIZAR PERMUTACIONES

- Realizar todas las permutaciones posibles de las aristas, descartar las que tengan mas/menos de  $V - 1$  aristas.
- Seleccionar los grafos conexos resultantes.
- Elegir el grafo cuya sumatoria de pesos sea menor.

## ALGORITMO DE KRUSKAL

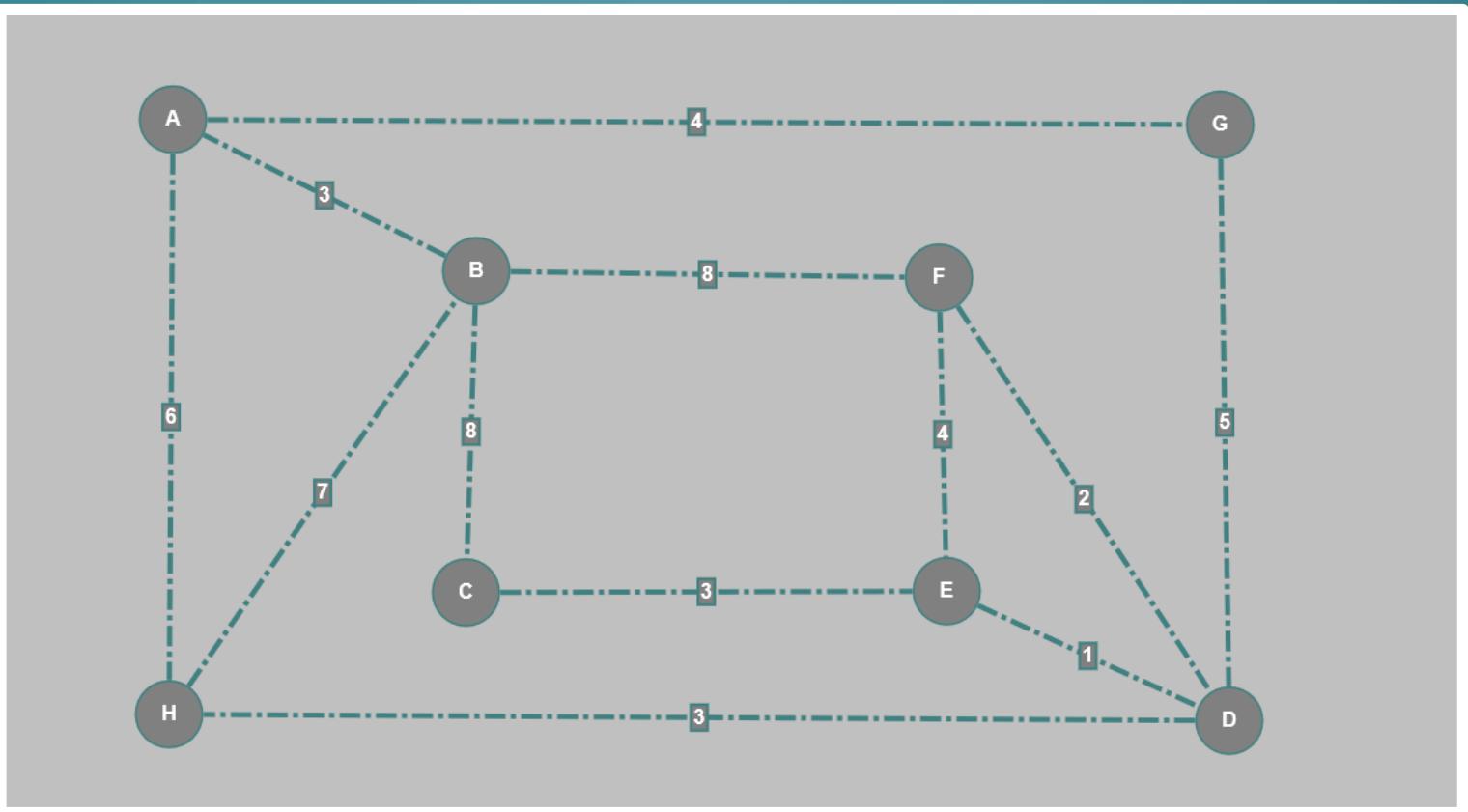
- Seleccionar la arista menos pesada que no cause un ciclo y agregarla al MST.
- Repetir el paso anterior hasta tener  $V - 1$  aristas.
- Se puede utilizar una lista de aristas ordenada por pesos para reducir el tiempo.

# ALGORITMO DE KRUSKAL

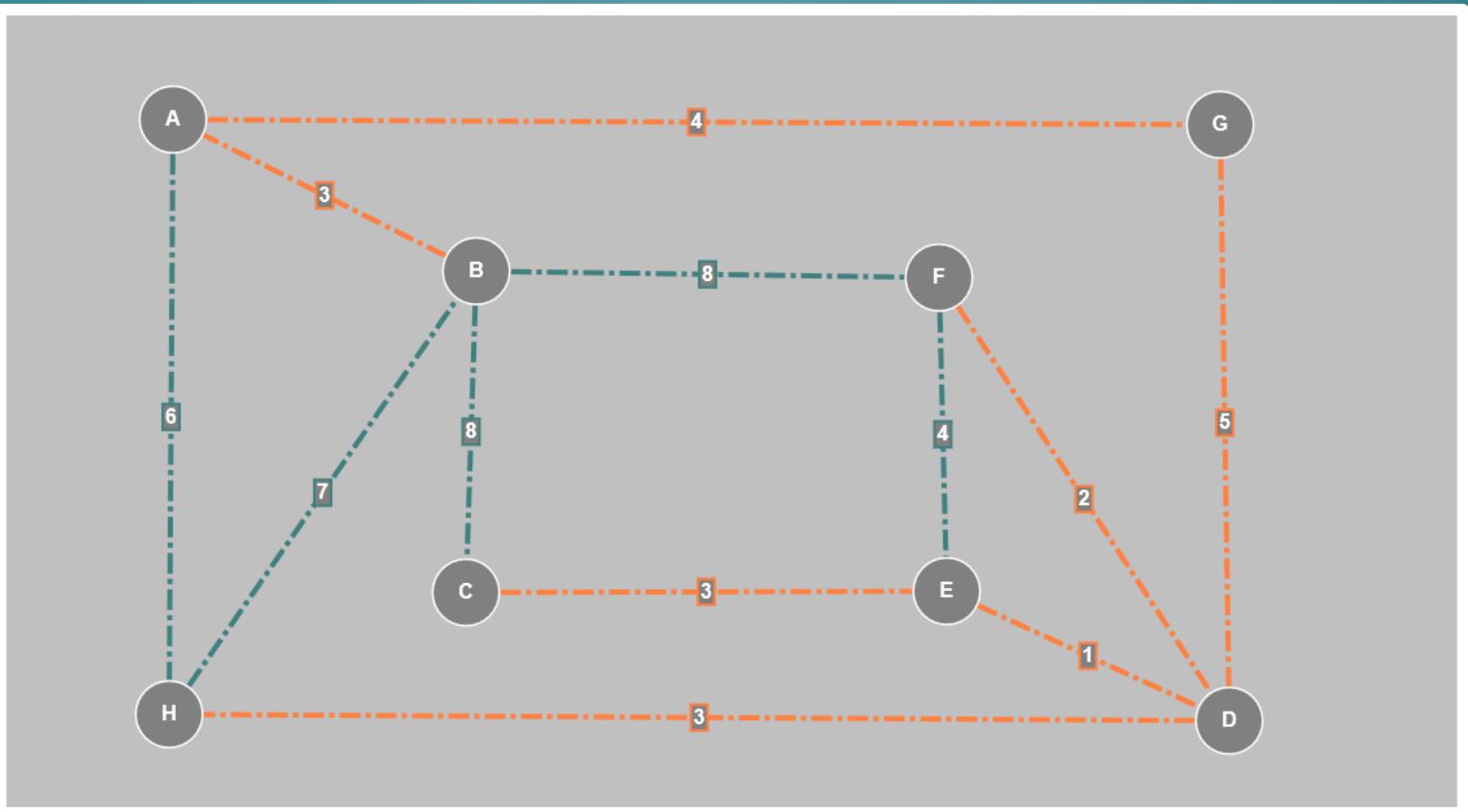
- El algoritmo de Kruskal es una forma de obtener el MST de un grafo no dirigido, pesado y conexo.
- Primero ordena las  $E$  aristas de forma no decreciente, y luego trata de insertarlas en el MST en orden, verificando que no formen un ciclo.
- El big O es igual a la suma del coste de ordenar las aristas, y el coste de insertar cada arista (que incluye el coste de la verificación).

$$O(E \log E + E(x \approx 1)) = O(E \log E) = O(E \log V^2) = O(E \log V)$$

# ALGORITMO DE KRUSKAL (DEMO)



# ALGORITMO DE KRUSKAL (DEMO)



# CONTEMPLACIONES

- No habrá caminos con coste negativo.
- Todos los caminos tendrán un coste  $n$ , con  $n > 0$  y  $n \neq \text{infinity}$ .
- La cantidad de casos a probar será finita, positiva y no nula.
- No puede haber casos donde existan solo 1 o 2 asentamientos.
- Ninguna ciudad tendrá mas de 1 camino hacia una ciudad determinada.
- El máximo de ciudades permitidas es 26 (cantidad de letras del alfabeto inglés).
- De la condición anterior, el máximo de aristas del MST será 25, y el máximo número de aristas del grafo inicial será 325 (grafo completo).

# CÓDIGO

```
iter = int(stdin.readline())

for i in range(1, iter + 1):
    number_of_vertex = int(stdin.readline())

    graph = []

    ## Proceed to adapt input graph to a (x, y, v) format, where x and y are vertex and v is the weight of the connection.
    for x in range(number_of_vertex):
        aux = [int(z) for z in stdin.readline().split(' ', ' ')]

        # print(aux)

        x += 1
        y = 0
        v = 0

        for conn in aux:
            # print(conn)

            y += 1

            if conn != 0:
                v = conn
                graph.append([x, y, v])

    # print( prettyPrint(graph))
    MSTGraph = KruskalMST(number_of_vertex, graph)
    prettyPrint(i, MSTGraph)
```

```
def prettyPrint(currentIteration, MSTGraph):
    global to_letter

    output = f"Case {currentIteration}:\n"

    for elem in MSTGraph:
        output += f"{to_letter[elem[0]]}-{to_letter[elem[1]]} {elem[2]}\n"

    stdout.write(output)
```

# CÓDIGO

```
def KruskalMST(V, graph):
    i = 0
    j = 0
    MST_graph = []

    # Sorting non-decreasing as was found online.
    aux_graph = sorted(graph, key=lambda item: item[2])

    # print(aux_graph)

    parent = []
    rank = []

    # Using V+1 to avoid bugs, pos 0 of list is unused
    for node in range(V+1):
        parent.append(node)
        rank.append(0)

    # Edges = V-1
    while j < (V - 1):
        try:
            u, v, w = aux_graph[i]
        except:
            break

        i = i + 1
        x = find(parent, u)
        y = find(parent, v)

        if x != y:
            j = j + 1
            MST_graph.append([u, v, w])
            union(parent, rank, x, y)

    return MST_graph

def find(parent, i):
    if parent[i] != i:
        parent[i] = find(parent, parent[i])

    return parent[i]

def union(parent, rank, x, y):
    if rank[x] < rank[y]:
        parent[x] = y

    elif rank[x] > rank[y]:
        parent[y] = x

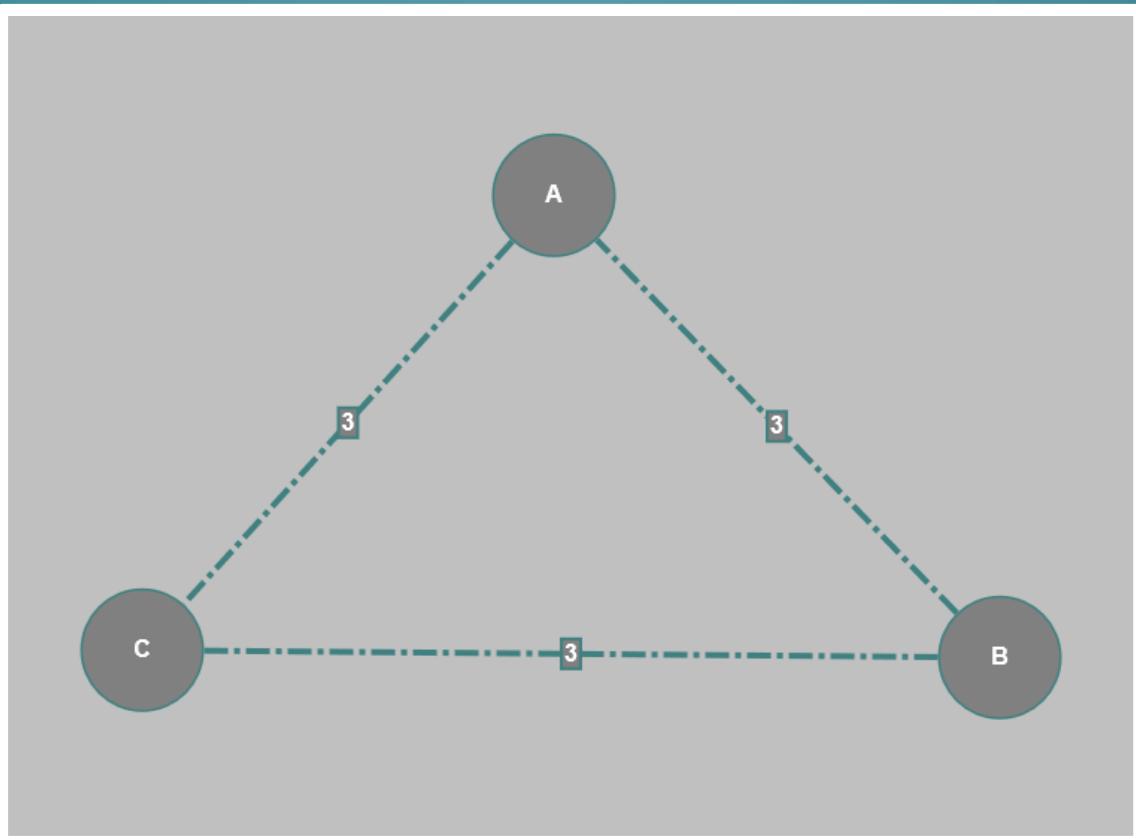
    else:
        parent[y] = x
        rank[x] += 1
```

# CASOS DE PRUEBA

- CASO 1: Mismo peso en todas las aristas del grafo
- CASO 2: Loop con mismo peso dentro del grafo
- CASO 3: Grafo completo
- CASO 4: Grafo con vértices con caminos a si mismos
- CASO 5: Grafo grande

# CASOS DE PRUEBA

CASO 1



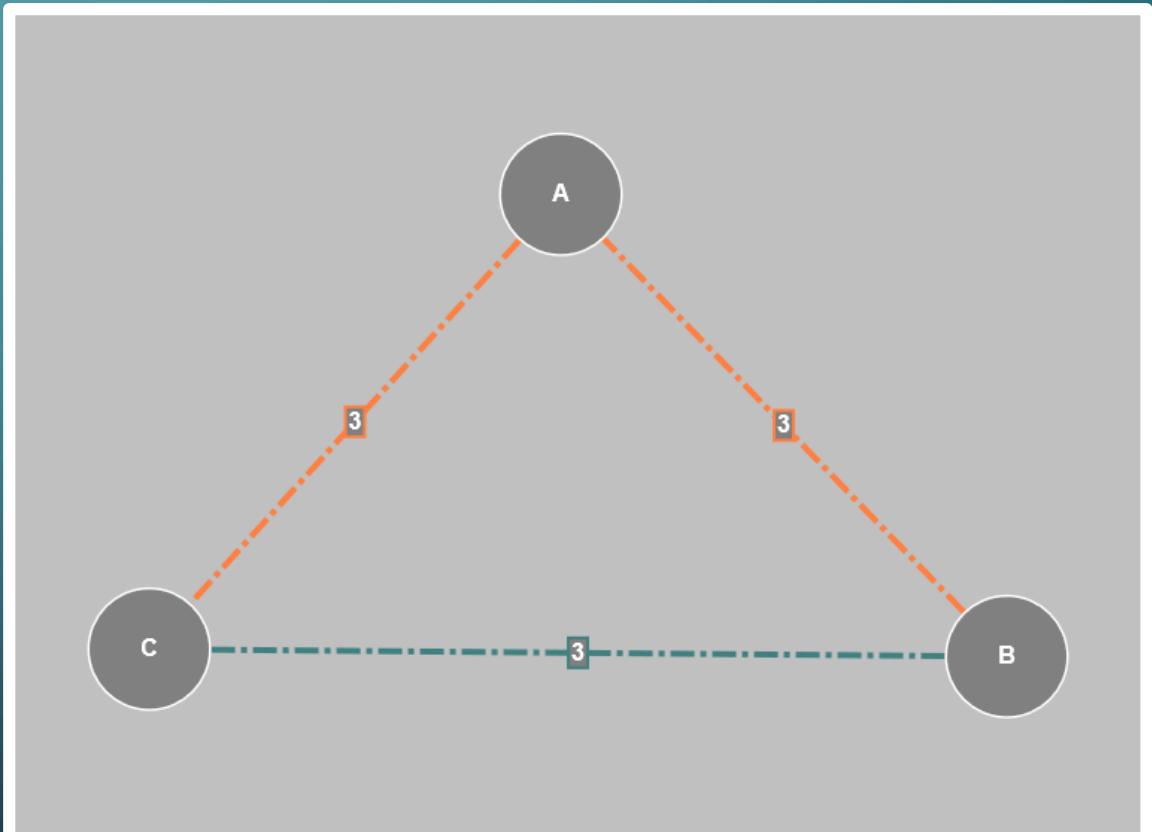
# CASOS DE PRUEBA

CASO 1

Case 1:

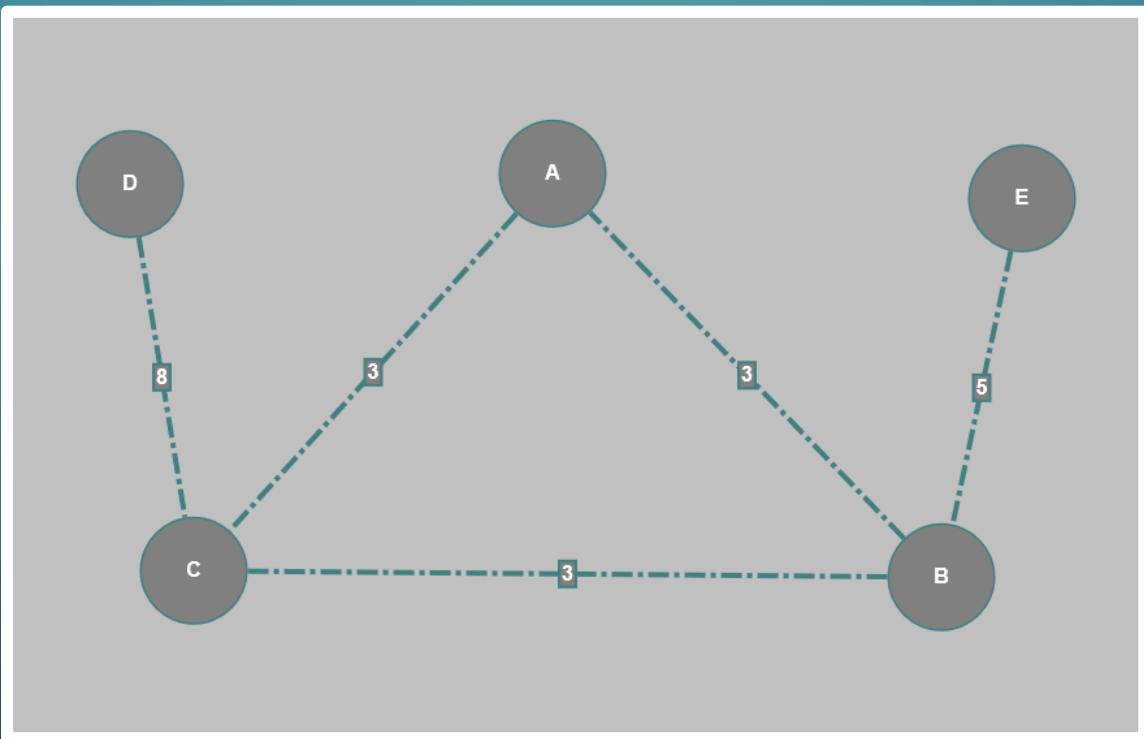
A-B 3

A-C 3



# CASOS DE PRUEBA

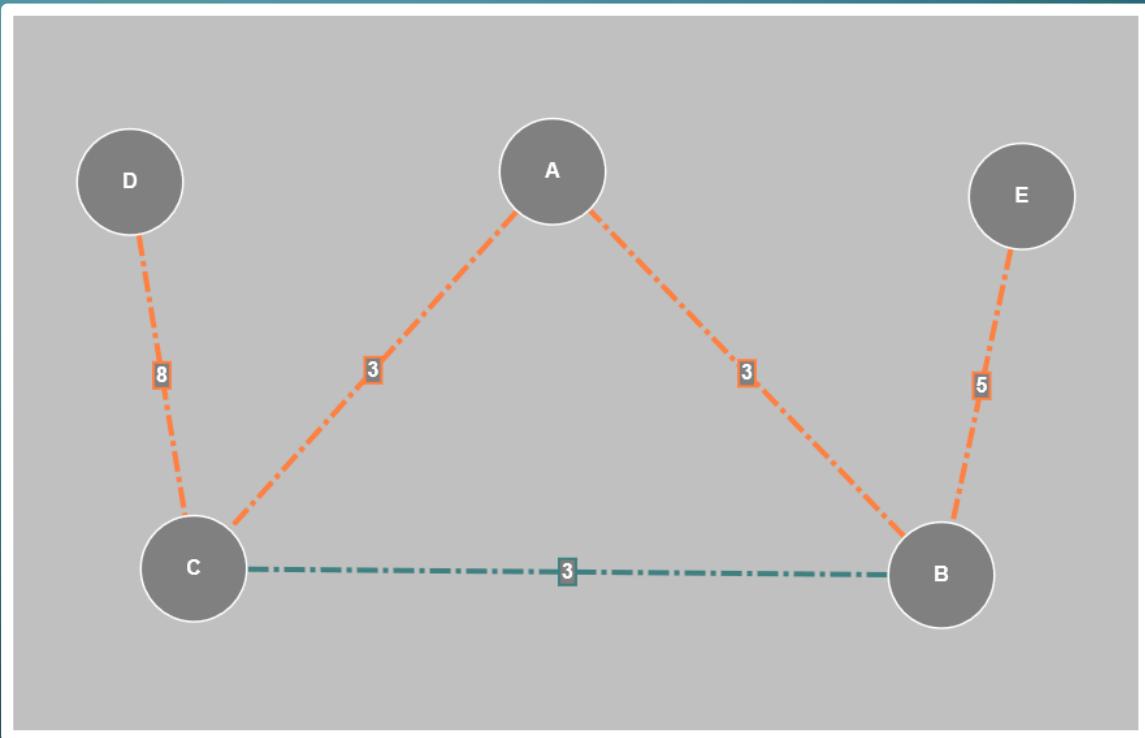
CASO 2



# CASOS DE PRUEBA

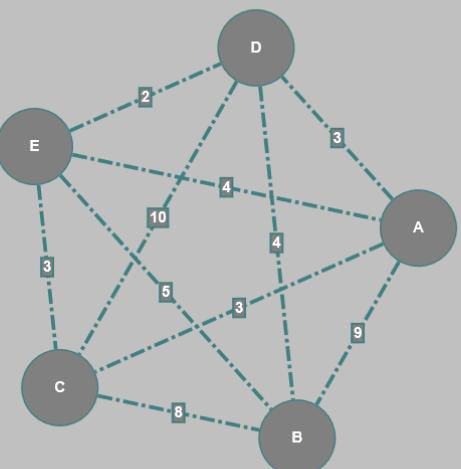
## CASO 2

Case 2:  
A-B 3  
A-C 3  
B-E 5  
C-D 8



# CASOS DE PRUEBA

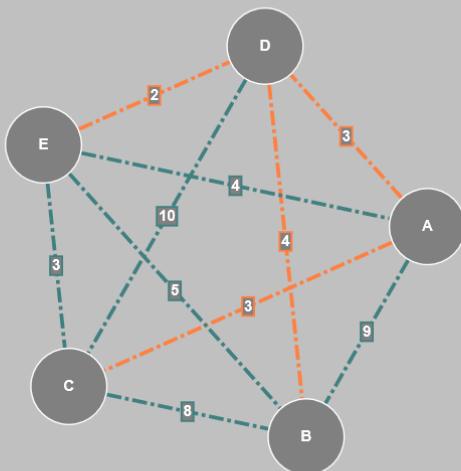
CASO 3



# CASOS DE PRUEBA

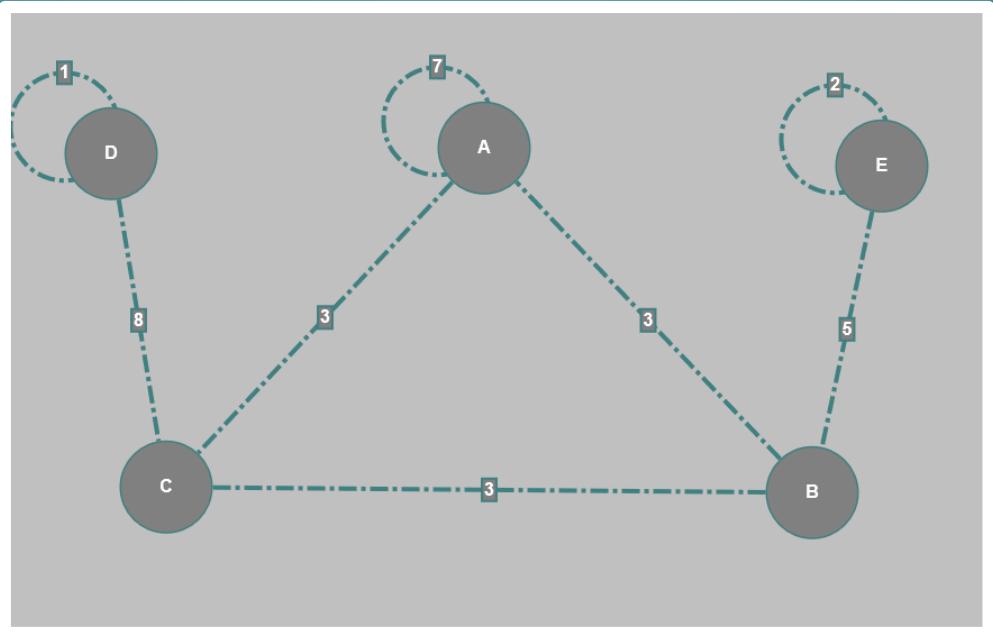
## CASO 3

Case 3:  
D-E 2  
A-C 3  
A-D 3  
B-D 4



# CASOS DE PRUEBA

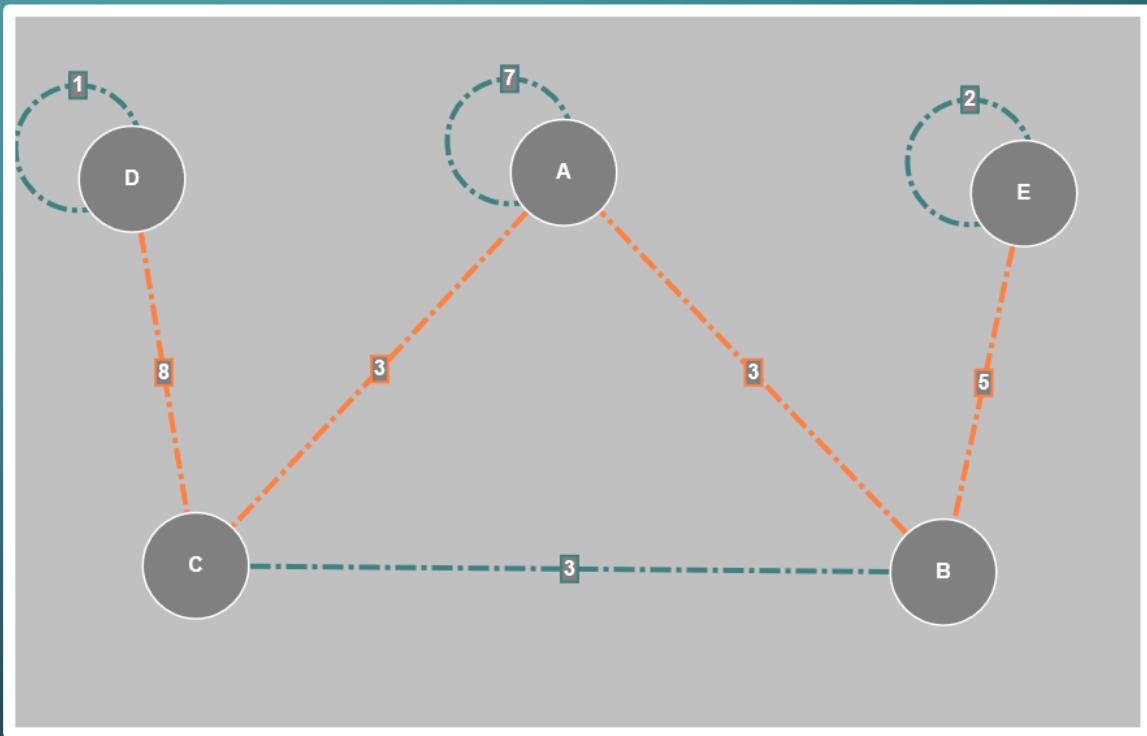
CASO 4



# CASOS DE PRUEBA

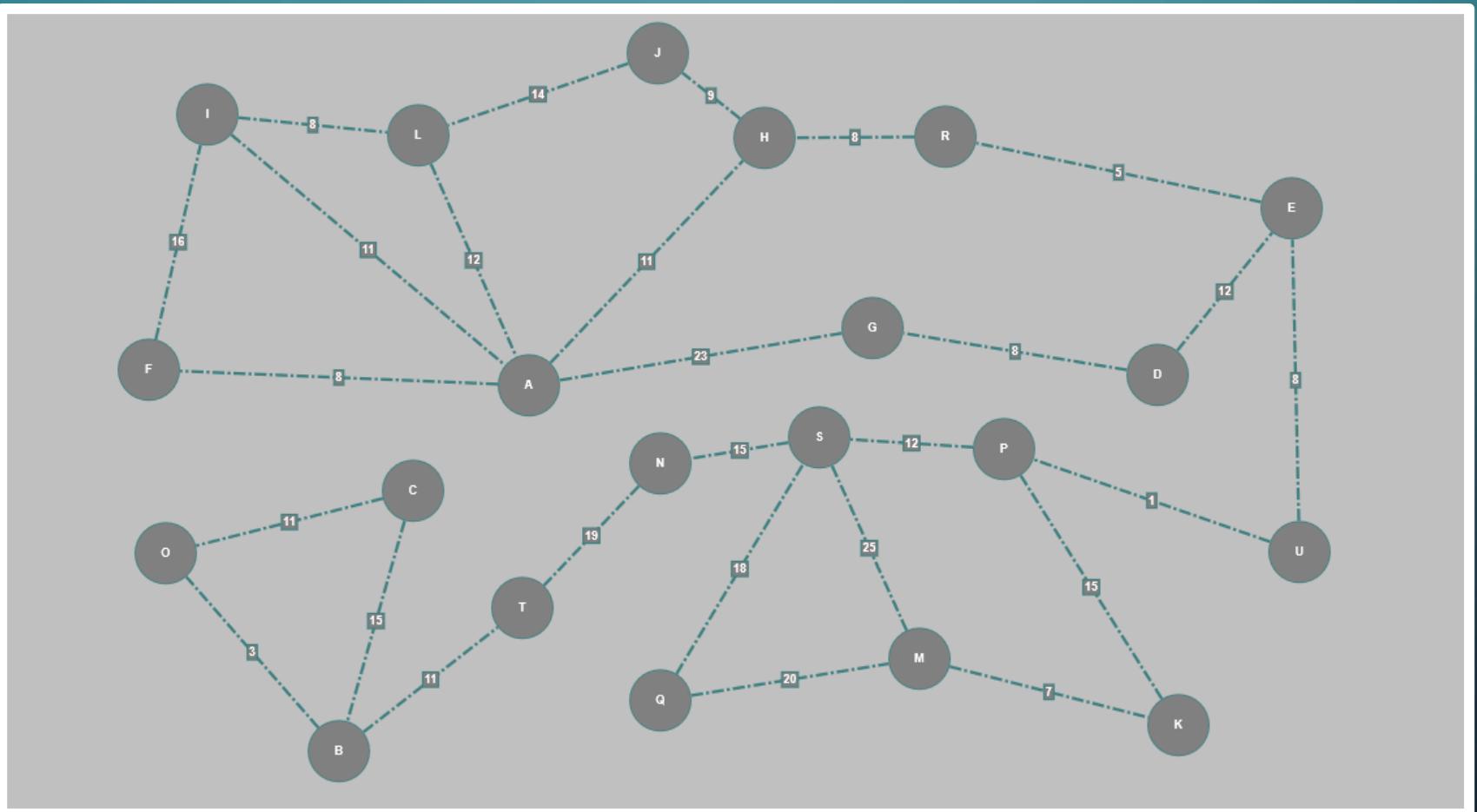
CASO 4

Case 5:  
A-B 3  
A-C 3  
B-E 5  
C-D 8



# CASOS DE PRUEBA

CASO 5

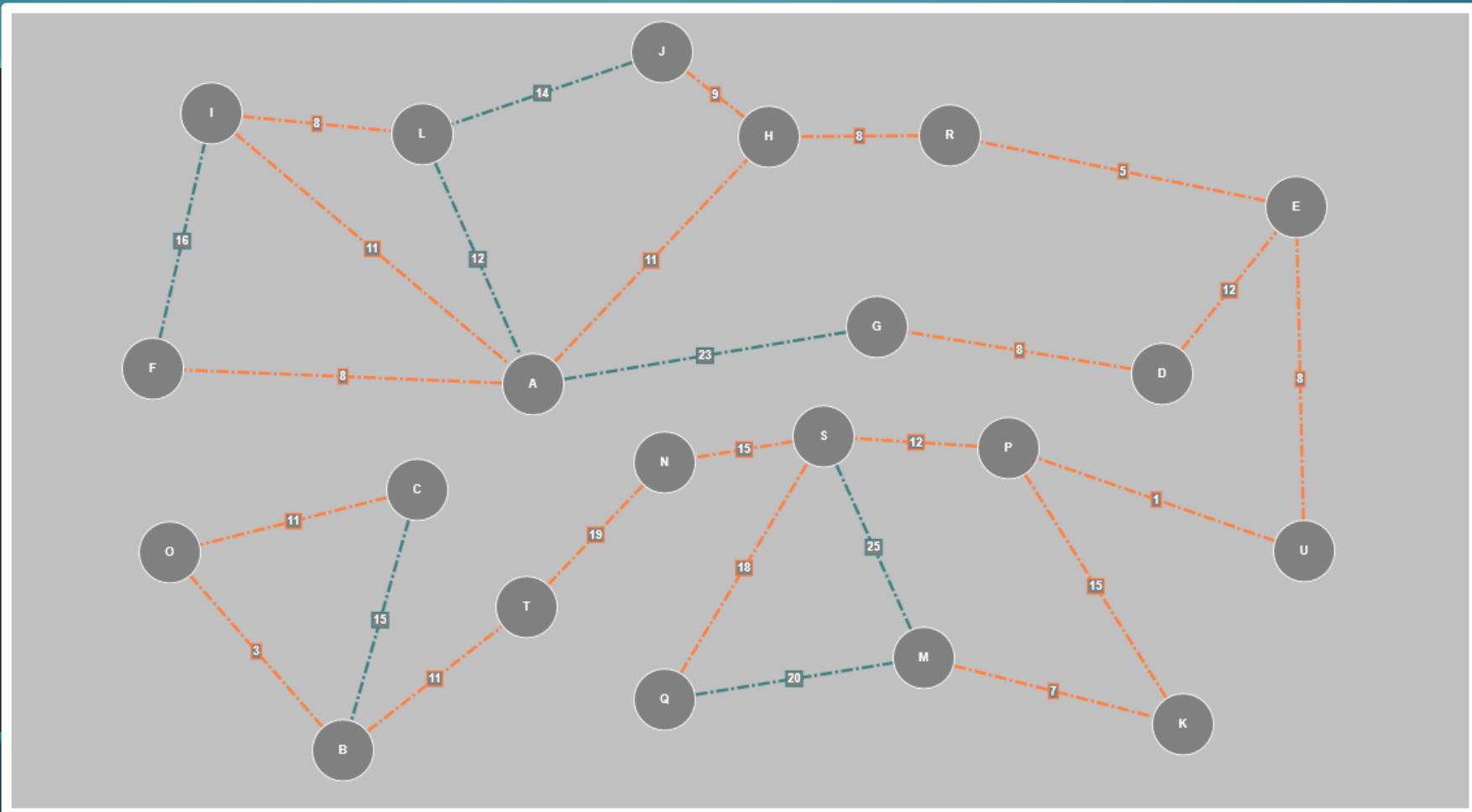


# CASOS DE PRUEBA

CASO 5

Case 4:

P-U 1  
B-O 3  
E-R 5  
K-M 7  
A-F 8  
D-G 8  
E-U 8  
H-R 8  
I-L 8  
H-J 9  
A-H 11  
A-I 11  
B-T 11  
C-O 11  
D-E 12  
P-S 12  
K-P 15  
N-S 15  
Q-S 18  
N-T 19



## REPOSITORIO PUBLICO

[github.com/Kass-UNLP/Oreon](https://github.com/Kass-UNLP/Oreon)