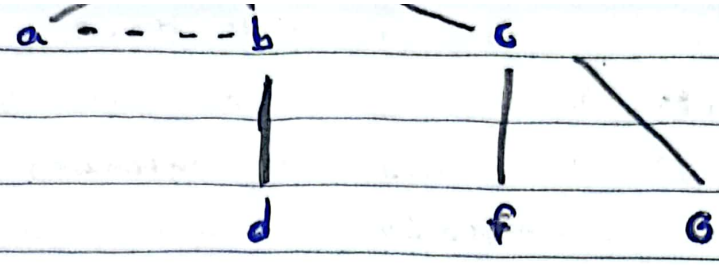


UTB_PD4

Ejercicio 1

1) Los tipos de datos abstractos que modelen este problema son los nodos y aristas. Para solucionar este problema, podemos aplicar el algoritmo de Prim o de Kruskal

2) Algoritmo Kruskal (grafo g)
Inicio
A ← conjunto vacio
aristasordenadas ← ordenarAristasPorPeso (g)
Para cada arista (u, v) en aristas ordenada.
Si noformaCiclo (A, u, v) entonces
Anadir Arista (A, v, v)
Fin Si
Fin Para
Devolver A
Fin

Algoritmo de Prim (Grafo G, Nodo Inicial)
Inicio

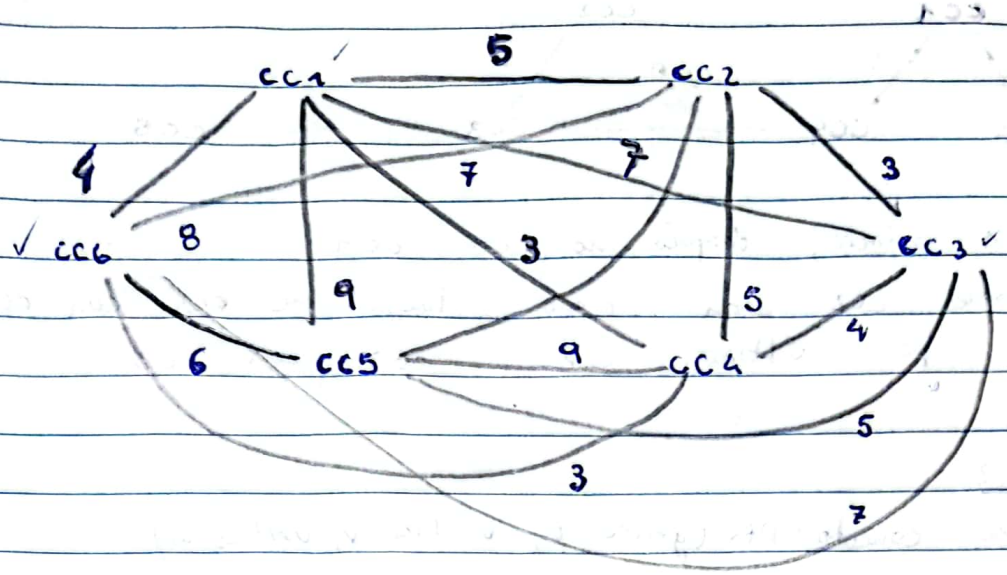
Q ← conjunto de todos los nodos de G
costos ← diccionario con valor infinito para cada nodo
costos [Inicial] ← 0
Predecesores ← diccionario vacío

Mientras Q no esté vacío hacer
u ← extraer Nodo Con Menor Costo (Q, costos)
Para cada vecino v de u Hacer
Si v está en Q y el peso de la arista (u, v) < costos [v] entonces
costos [v] ← Peso de la arista (u, v)
Predecesores [v] ← u
fin Si
fin Para
Fin Mientras

A ← conjunto vacío
Para cada nodo v en predecesores hacer
añadirArista (A, predecesores [v], v)
Fin Para

devolver A
Fin

3) Aplica cada algoritmo



Algoritmo de Prim

1) CC1 — 3 — CC4

Agregar CC4

2) CC1 — 3 — CC4; CC6 — 3 — CC4

Agregar CC6

3) CC1 — 3 — CC4; CC6 — 3 — CC4; CC4 — 4 — CC3

Agregar CC3

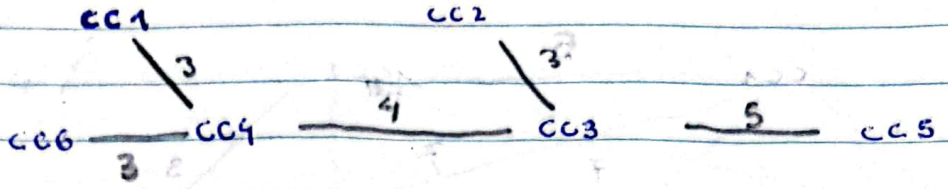
4) CC1 — 3 — CC4; CC6 — 3 — CC4; CC4 — 4 — CC3; CC2 — 3 — CC3; CC3 — 5 — CC5

Agregar CC2 y CC5

costo total 18

Algoritmo de Kruskal

coste total 18



cc1 primero, después une con cc4
después cc2 con cc3, luego une cc3 con cc4
y por último une cc3 con cc5

Ejercicio 2

```
Algoritmo conectadoDFS (grafo G, vértice v, vértice w)
Inicio
    Pila ← CrearPila()
    visitados ← conjuntoVacio()

    apilar (pila, v)
    añadir (visitados, v)

    Mientras no estáVacía (pila) hacer
        v ← desapilar (pila)

        Si v = w entonces
            devolver verdadero
        Fin Si

        Para cada vecino en vecinos (G, v) hacer
            si vecino no en visitados entonces
                apilar (pila, vecino)
                añadir (visitados, vecino)
            Fin Si
        Fin Para
    Fin Mientras
    Devolver Falso
Fin
```

```
Algoritmo conectadoBFS (grafo G, vértice v, vertice w)
Inicio
    cola ← crearcola()
    visitados ← conjuntoVacío()

    encolar (cola, v)
    añadir (visitados, v)

    Mientras no estaVacia(cola) hacer
        u ← desencolar (cola)

        Si u = w entonces
            Devolver verdadero
        Fin si

        Para cada vecino en vecinos (G, u) hacer
            Si vecino no en visitados entonces
                encolar (cola, vecino)
                añadir (visitados, vecino)
            Fin Si
        Fin Para
    Fin Mientras

    Devolver Falso
Fin
```