Kruskal

Árbol: grafo en el que cualquier par de nodos están conectados por exactamente un camino (acíclico)

MST de un grafo G: subgrafo de G que es árbol, cubre todos los nodos y tiene el mínimo costo que cualquier otro árbol de cobertura

$\texttt{Kruskal}(G)$:
    $E \leftarrow E$ ordenada por costo, de menor a mayor
    $T \leftarrow$ lista vacía
    for $e \in E$:
        if *Agregar e a T no forma ciclo* :
            $T \leftarrow T \cup \{e\}$
    return T

$\texttt{Kruskal}(G)$:
    $E \leftarrow E$ ordenada por costo, de menor a mayor
    for $v \in V$:
        $\texttt{MakeSet}(v)$
    $T \leftarrow$ lista vacía
    for $(u, v) \in E$:
        if $\texttt{Find}(u) \neq \texttt{Find}(v)$:
            $T \leftarrow T \cup \{(u, v)\}$
            $\texttt{Union}(u, v)$
    return T

Todos los Nodos parten como **conjuntos disjuntos**

I3 2023-2 P3

[3 ptos.] Se propone el siguiente algoritmo con la intención de obtener un MST de un grafo G. Observe que la línea 2 toma aristas de E en orden arbitrario.

```
    CasiKruskal(G):
1       T ← lista vacía
2       for e ∈ E :
3           if Agregar e a T no forma ciclo :
4               T ← T ∪ {e}
5       return T
```

(i) [1 pto.] Describa qué estructuras de datos utilizar para implementar este algoritmo de forma eficiente. Indique la complejidad de `CasiKruskal` al usar dichas estructuras.

I3 2023-2 P3

[3 ptos.] Se propone el siguiente algoritmo con la intención de obtener un MST de un grafo G. Observe que la línea 2 toma aristas de E en orden arbitrario.

`CasiKruskal`(G):

1 $T \leftarrow$ lista vacía
2 **for** $e \in E$ **:**
3 **if** *Agregar* e *a* T *no forma ciclo* **:**
4 $T \leftarrow T \cup \{e\}$
5 **return** T

(i) [1 pto.] Describa qué estructuras de datos utilizar para implementar este algoritmo de forma eficiente. Indique la complejidad de `CasiKruskal` al usar dichas estructuras.

Solución.

La línea 3 corresponde a la operación más costosa de este algoritmo. Para determinar eficientemente si agregar una arista forma o no un ciclo, podemos usar la estructura de conjunto disjunto para almacenar los conjuntos de nodos que están conectados, tal como en Kruskal.

Complejidad

- Construir V conjuntos singleton $\mathcal{O}(V)$
- Unir conjuntos $V - 1$ veces $\mathcal{O}(V)$
- Buscar $2E$ veces $\mathcal{O}(E\alpha(V)) = \mathcal{O}(E)$

(V +V + E) ∈ O(E) para grafos conexos

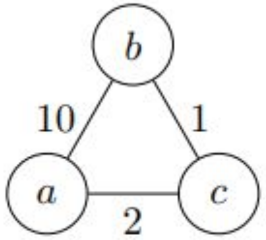
- Recordar que cada llamado a Find y Union se puede considerar O(1)

(ii) [2 ptos.] Demuestre que `CasiKruskal` no retorna un MST para todo grafo G.

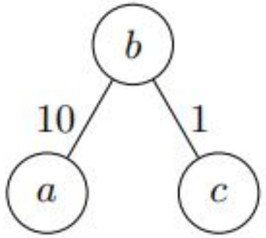
(ii) [2 ptos.] Demuestre que `CasiKruskal` no retorna un MST para todo grafo G.

Solución.

Consideremos el grafo $G = (\{a, b, c\}, \{\{a, b\}, \{b, c\}, \{a, c\}\})$ con los siguientes costos

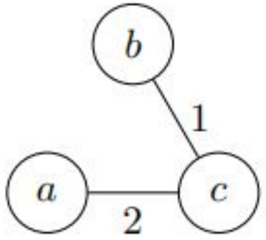


Si el orden de iteración de aristas en la línea 2 del algoritmo es $\{a, b\}, \{b, c\}, \{a, c\}$, el árbol obtenido por `CasiKruskal` es



costo 11

pero sabemos que el único árbol de cobertura de costo mínimo para G es



costo 3

por lo que el algoritmo no entrega un MST para todo grafo.