ADA – Tarea 4
Xavier Garzón

Feedback Edges

a)

La siguiente implementación está basada en la descripción del algoritmo de Kruskal dada en Algorithms (Jeff Erickson, pp. 267) usando conjuntos disjuntos. Además, se asume que existen las siguientes operaciones en una estructura de datos dada.

- **MakeSet(*v*):** crea un conjunto que contiene solo el vértice *v*.
- **Find(*v*):** retorna el representante del conjunto donde está contenido el vértice *v*.
- **Union(*u*,*v*):** reemplaza los conjuntos que contienen *u* y *v* con su unión. (Esta operación disminuye el número de conjuntos).

Sea ***V*** conjunto de vértices y ***E*** conjunto de aristas.

```
solve(V,E):
   sort E by decreasing weight
   F ← (V,∅)
   for each vertex v∈V:
      MakeSet(V)
   for i ← i to |E|:
      uv ← ith heavier edge in E
      if Find(u)≠Find(v)
         Union(u,v)
         add uv to F
   return F
```

b)

Para diseñar un algoritmo que calcule este conjunto (*feedback edge*) tomaremos el algoritmo presentado en el punto *"a"* con algunas modificaciones.

FeedBack es el conjunto donde quedarán las aristas que generan los ciclos en el grafo. Las aristas se agregarán a este conjunto en el momento que dos vértices compartan un mismo representante, en otras palabras, generan un ciclo dentro del grafo.

Sea ***V*** conjunto de vértices y ***E*** conjunto de aristas.

```
solve(V,E):
   sort E by decreasing weight
   F ← (V,∅)
   FeedBack ← (V,∅)
   for each vertex v∈V:
      MakeSet(V)
   for i ← i to |E|:
      uv ← ith heavier edge in E
      if Find(u)≠Find(v)
         Union(u,v)
         add uv to F
      else
         add uv to FeedBack
return FeedBack
```

Negative Edges

a)

La siguiente implementación está basada en la descripción del algoritmo de Bellman-Ford explicada por Miguel en clase.

La idea es usar al algoritmo de Bellman-Ford con una pequeña modificación al final del mismo. Si se encuentran ciclos negativos dentro del grafo entonces no se puede determinar un camino óptimo de *s* a *t*. En caso contrario, se retorna el costo de ir de *s* a *t*

```
solve(V,w,s,t):
    ans=None
    dist=[INF for _ in range(V)]
    for i in range(V-1):
        for u in range(V):
            for j in range(len(AdjList[u])):
                v,w=AdjList[u][j]
                dist[v]=min(dist[v], dist[u]+w)

    hasNegativeCycle=False
    for u in range(V):
        for j in range(len(AdjList[u])):
            if (dist[v]>dist[u]+w):  hasNegativeCycle=True
    if  not(hasNegativeCycle): ans=dist[t]
    else: ans= hasNegativeCycle
    return ans
```

b)

El algoritmo presentado en el punto *"a"* sirve igualmente para resolver este problema. La cantidad de aristas negativas en el grafo no afectan la complejidad del algoritmo, pues de la complejidad del mismo está dada por la cantidad de vértices y aristas (sin importar si son positivas o negativas). Complejidad del algoritmo: O(*VE*)