Departamento de Cs. e Ingeniería de la Computación
Universidad Nacional del Sur

Algoritmos y Complejidad

Trabajo Práctico 8

Grafos II

Primer cuatrimestre de 2017

1. Dadas las siguientes funciones

```
inicializar(G: grafo, s: nodo)
    for v in vertices de G:
        d[v] = INFINITO
        padre[v] = NULL
    d[s] = 0

relajar(u: nodo, v: nodo)
    if d[u] + peso(u,v) < d[v]
        d[v] = d[u] + peso(u,v)
        padre[v] = u
```

   a) Dar el pseudocódigo del algoritmo de Dijkstra asumiento que también se cuenta con una implementación de cola de prioridad.

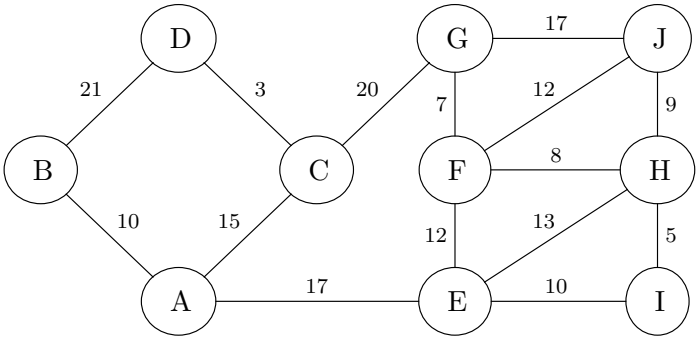
   b) Analizar el tiempo de ejecución del algoritmo para las implementaciones de cola de prioridad como: una búsqueda secuencial, un heap.

   c) Demostrar que durante la ejecución del algoritmo se mantienen la siguientes propiedades:

   - **Cota superior** Para todo nodo u, $d[u] \geq \delta(s, u)$. Donde $\delta(u, v)$ es la mínima distancia de un camino $u \rightsquigarrow_G v$. Utilizar inducción.
   - **Convergencia**: Para todo arco (u, v), si $s \rightsquigarrow_G u \rightarrow v$ es un camino más corto y $d[u] = \delta(s, u)$ en algún momento anterior a relajar el arco (u, v), entonces $d[v] = \delta(s, v)$ en todo momento posterior.

2. Muestre un algoritmo que dado un grafo no dirigido G y dos nodos u y v determine el conjunto de los nodos w tal que existe un camino de costo mínimo $u \rightsquigarrow_G w \rightsquigarrow_G v$.
   Ayuda: Realizar 2 corridas de Dijkstra.

3. Realizar la traza del algoritmo de Dijkstra para el siguiente grafo tomando a A como vértice inicial. En cada paso, indicar cuál es el vértice elegido y actualizar los valores del arreglo $d[]$ mediante la operación de relajación de arcos.



4. *Algoritmo de Bellman-Ford*

   El algoritmo de *Bellman-Ford* propone encontrar caminos más cortos con origen único relajando todos los arcos del grafo $n-1$ veces (donde n es la cantidad de nodos del grafo). Es más ineficiente que el algoritmo de Dijkstra pero es aplicable a grafos más generales, en este caso se permite que algunos arcos del grafo tengan peso negativo. Utilizando `inicializar` y `relajar` definidas en el punto 1.

```
bellman-ford(G,s)
    inicializar(G,s)
    repetir n-1 veces
        for (u,v) in arcos de G
            relajar(u,v)
```

   a) Demostrar por inducción que luego de i repeticiones del bucle `repetir`, todos los caminos de costo mínimo que poseen a lo sumo i arcos están correctamente calculados.

   b) Demostrar que si el grafo no posee ciclos de costo negativo (la suma de los pesos de sus arcos es negativa), el algoritmo computa correctamente el costo de un camino mínimo de s a cada vértice.

   c) Analizar el tiempo de ejecución del algoritmo.

5. Árbol de cubrimiento minimal

   a) Describir la estrategia utilizada por el algoritmo de *Kruskal* y la utilizada por el algoritmo de *Prim*. Aplique ambas estrategias al grafo dado en el ejercicio 3.

   b) Dar un algoritmo que retorne un árbol de cubrimiento minimal, siguiendo la estrategia de *Kruskal* y utilizando un heap para almacenar los arcos del grafo. Analice su tiempo de ejecución.

   c) Un grafo puede tener varios árboles de cubrimiento minimal diferentes. ¿Dónde se refleja esta posibilidad en el algoritmo de *Kruskal*? ¿y en el algoritmo de *Prim*?

   d) Determinar, tanto para el algoritmo de *Kruskal* como para el algoritmo de *Prim* qué sucede si estos algoritmos se utilizan sobre un grafo no conexo.

   e) Sea $G = (N, A)$ un grafo no dirigido conexo tal que el costo asociado a sus arcos es positivo, y sea T un subconjunto de los arcos en A tal que $G' = (N, T)$ es un grafo conexo y la suma de los costos de sus arcos es mínima. Probar que G' es un árbol.

6. Un *circuito de Euler* de un grafo dirigido $G = \langle N, A\rangle$ es un ciclo que pasa exactamente una vez por cada arco de G (pudiendo pasar varias veces por algunos nodos) tal que comienza y termina en un nodo determinado.

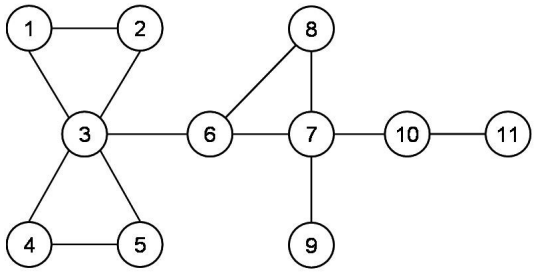
   a) Mostrar que G tiene un circuito de Euler si y solo si el grafo es conexo e incidentes(u) = salientes(u) para todo nodo $u \in N$.

   b) Describir un algoritmo para saber si un grafo G tiene un circuito de Euler.

   c) Enunciar un algoritmo del $O(a)$ para encontrar un circuito de Euler en un grafo G si es que existe.

   d) Se dice que un grafo dirigido contiene un *camino de Euler* si existe un camino que pasa exactamente una vez por cada arco de G. A diferencia de un circuito de Euler, un camino de Euler no necesariamente comienza y termina en un mismo nodo. Realizar un algoritmo para determinar si un grafo G contiene un camino de Euler.

7. Para el algoritmo que encuentra los puntos de articulación de un grafo conexo no dirigido, verificar que en el grafo de la figura que se muestra a continuación se obtienen los mismos puntos de articulación si se comienza el DFS en el nodo 1 o en el nodo 6.



8. Un grafo conexo es *bicoherente* si cada punto de articulación está unido por lo menos por dos arcos con cada componente del grafo restante. Escribir un algoritmo que decida si un grafo es o no bicoherente.

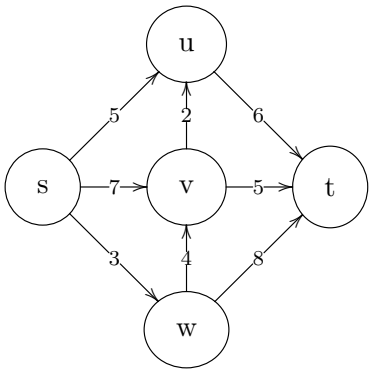
9. Usando el algoritmo de Edmonds-Karp para encontrar el flujo máximo en una red de flujo, encontrar el flujo máximo para la siguiente red, donde s es la fuente y t el sumidero:

	s	a	b	c	d	t
s	0	3	2	0	0	0
a	0	0	1	3	4	0
b	0	0	0	0	2	0
d	0	0	0	2	0	3
c	0	0	0	0	0	2
t	0	0	0	0	0	0

10. Para la red de la figura, donde **s** es el nodo origen (*source*) y **t** es el nodo destino (*sink*).

    a) Calcule su flujo máximo utilizando el algoritmo de Edmonds-Karp. Muestre la secuencia de caminos de aumento producida por el algoritmo, y las redes de flujo y residuales producidas en el proceso.

    b) Muestre cuál es el mínimo corte (Recuerde el teorema *Máximo-Flujo Mínimo-Corte*).



11. Considerando un grafo no dirigido bipartito $G = \langle N, A \rangle$, se define como *"matching"* a un conjunto de arcos $M \subseteq A$, tal que para cualquier nodo v de N se cumple que **a lo sumo** un arco de M incide en v. Un *matching* es máximo si es un *matching* de máxima cardinalidad (ie. no existe otro de cardinalidad mayor).

    Explicar cómo puede utilizarse el método de $Ford - Fulkerson$ para solucionar el problema de encontrar un *matching* de cardinalidad máxima en un grafo bipartito. Dar **un ejemplo** en el que se muestre, paso a paso, (siguiendo la explicación dada) cómo se obtiene el *matching* de cardinalidad máxima.