**Estructuras de Datos y Algoritmos – iic2133**

**Control 5**

22 de mayo, 2019

**Nombre**: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**1**) Se tiene un *stream* de largo indefinido donde cada dato representa una arista no direccional que se agrega a un grafo inicialmente vacío. La idea es que per-manezca acíclico, por lo que si la arista a agregar forma un ciclo se detiene la lectura del *stream* y se retorna . **Explica** **en detalle** cómo llevar a cabo este proceso, determinando de manera eficiente en cada paso si la arista a agregar forma un ciclo en . **Cuidado**: no conoces todos los vértices por adelantado.

**2**) El siguiente es el **algoritmo de Dijkstra** para encontrar las rutas más cortas desde el vérti-ce *s* en un grafo *G* = (*V*, *E* ) direccional, cuyas aristas (*u*,*v*) ∈ *E* tienen costos ω(*u*,*v*) ≥ 0.

|  |  |
| --- | --- |
| dijkstra( *s* ):  **for each** *v* **in** *V*:  d[*v*] = ∞  π[*s*] = null  d[*s*] = 0  *S* = ∅  *q* = Queue(*V* ) | **while** !*q*.empty( ):  *u* = *q*.xMin( )  *S* = *S* ∪ {*u*}  **for each** *v* **in** α[*u*]:  **if** d[*v*] > d[*u*] + ω(*u*,*v*):  d[*v*] = d[*u*] + ω(*u*,*v*)  π[*v*] = *u* |

**a**) En clase vimos la complejidad de este algoritmo cuando *G* es representado mediante sus lis-tas de adyacencias (el arreglo α) y la cola *q* es implementada mediante un *min heap* binario. Determina justificadamente la complejidad del algoritmo de Dijkstra en este caso.

**b**) Demuestra (basta un ejemplo) que cuando se encuentra por primera vez un nodo, más allá de *s*, no necesariamente se lo ha encontrado por la ruta más corta.