Pontificia Universidad Católica de Chile
Escuela de Ingeniería
Departamento de Ciencia de la Computación

IIC2133 — Estructuras de Datos y Algoritmos 2'2023

Interrogación 3

14 de diciembre de 2023

Condiciones de entrega. Debe entregar solo 3 de las siguientes 4 preguntas.

Nota. Cada pregunta tiene 6 puntos (+1 punto base). La nota es el promedio de las 3 preguntas entregadas.

1. Colas de prioridad

Un sistema de control de producción utiliza un Stack S (cola LIFO) para manejar la secuencia de proceso de los trabajos que recibe. Así, para agregar un trabajo t a la cola se utiliza `Push`(S, t) y para obtener un trabajo a ejecutar desde la cola se utiliza $t \leftarrow$ `Pop`(S), que retorna `null` si el Stack S está vacío. Los trabajos t tienen entre sus atributos el tiempo restante de ejecución, el que se puede obtener como t`.tiempo`. Se propone mejorar la cantidad de trabajos entregados por día reemplazando el Stack S del sistema por un minHeap H que utilice el tiempo restante de ejecución como prioridad de cada trabajo t.

(a) [2 ptos.] Proponga el pseudocódigo para los siguientes métodos:

(i) [1 pto.] `stack2minHeap`(S), que recibe como entrada el stack S del sistema y retorna un minHeap H representado como arreglo, priorizado por el tiempo restante de ejecución de los trabajos.

Solución.

Usaremos una versión modificada de `SiftDown`

```
SiftDown(H, i):
    if i tiene hijos :
        j ← hijo de i con menor prioridad
        if H[j].tiempo < H[i].tiempo :
            H[j] ⇋ H[i]
            SiftDown(H, j)
```

con lo cual, el algoritmo pedido es

```
stack2minHeap(S):
    A ← S como arreglo
    for i = ⌊n/2⌋ − 1 . . . 0 :        ▷ loop decreciente
        SiftDown(A, i)
```

donde se usa la versión modificada de `SiftDown`.

Puntajes.
0.5 por modificación de `SiftDown`.
0.5 por método pedido.
Observación: cualquier explicación en lugar de pseudocódigo debe ser suficientemente precisa para indicar exactamente los cambios necesarios a los algoritmos vistos en clases para maxHeaps.

(ii) [0,5 pto.] `extract`(H), que retorna el trabajo t de menor tiempo restante de ejecución en H.

Solución.

El método `Extract` es idéntico al visto en clases, pero utilizando `SiftDown` del inciso (i).

Puntajes.
0.5 por especificar el uso de la versión nueva de `SiftDown`.

(iii) [0,5 pto.] `insert`(H, t) que inserta en el minHeap H el trabajo t.

Solución.

Usando la versión modificada de `SiftUp`

```
SiftUp(H, i):
    if i tiene padre :
        j ← ⌊i/2⌋
        if H[j].tiempo > H[i].tiempo :
            H[j] ⇋ H[i]
            SiftUp(H, j)
```

el método `Insert` estudiado en clases cumple lo pedido.

Puntajes.
0.5 por modificación de `SiftUp` para usarse en `Insert`.

(b) [2 ptos.] Para evitar que los trabajos que tienen mayores tiempos restantes de ejecución nunca sean extraídos del minHeap H se propone que, cada vez que se inserta un trabajo t en H, todas las hojas de H reduzcan en un 33 % su tiempo restante de ejecución. Modifique su algoritmo `insert`(H, t) de manera que realice lo indicado, manteniendo la propiedad de Heap de H, representado como arreglo.

Solución.

Proponemos el siguiente pseudocódigo que utiliza la versión modificada de `SiftUp` descrita en el inciso anterior:

```
Insert(H):
    for i = ⌊n/2⌋ − 1 . . . 0 :        ▷ loop decreciente
        H[i].tiempo ← H[i].tiempo · 0,66
    for i = ⌊n/2⌋ − 1 . . . 0 :        ▷ loop decreciente
        SiftUp(A, i)
    i ← primera celda vacía de H
    H[i] ← e
    SiftUp(H, i)
```

el método `Insert` estudiado en clases cumple lo pedido.

Puntajes.
1.0 por modificar las hojas según lo pedido.
1.0 por insertar el nuevo elemento.
Observación: el orden de las dos operaciones puede invertirse, mientras se mantenga la modificación pedida de las hojas originales.

(c) [2 ptos.] Un consultor indica que se logra un mejor desempeño global del sistema si se privilegian para ejecución los trabajos con mayor tiempo restante de ejecución. Proponga un algoritmo en pseudocódigo `minHeap2maxHeap`(H), que retorna un MaxHeap M a partir H. La complejidad de tiempo de su solución debe ser $\mathcal{O}(n)$ para un minHeap con n nodos.

Solución.

Dado que se quiere construir un maxHeap, se puede utilizar la versión de `BuildHeap` estudiada en clase sobre el arreglo que representa el minHeap construido en los incisos anteriores. El resultado es el maxHeap deseado.

Puntajes.
1.0 por utilizar versión de `BuildHeap`.
1.0 por especificar que es la versión estudiada para maxHeaps, i.e. no la del inciso (a).

2. Recorrido de grafos

(a) [3 ptos.] En clases estudiamos una versión recursiva del recorrido DFS de un grafo. Proponga el pseudocódigo de un algoritmo iterativo que implemente DFS en un grafo dirigido $G = (V, E)$, que opere en tiempo $\mathcal{O}(V + E)$. Indique precisamente qué estructura(s) de datos necesita para su solución.

Solución.

Proponemos el siguiente pseudocódigo, que es una modificación del algoritmo para BFS estudiado en clase. Este método reemplaza al `DFS-Visit` recursivo que se estudió en clase:

```
DFS-Visit(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    Q ← stack LIFO vacío
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        for v ∈ α[u] :
            if v.color = blanco :
                v.color ← gris;  v.δ ← u.δ + 1
                π[v] ← u
                Insert(Q, v)
        u.color ← negro
```

donde se utiliza una lista ligada para implementar el stack Q, de forma que el próximo elemento que se extrae es el último que entró. El recorrido DFS es idéntico al estudiado en clase, pero utiliza esta versión iterativa de `DFS-Visit` con igual complejidad.

Puntajes.
1.0 por proponer un algoritmo iterativo.
1.0 por utilizar estructuras de datos acordes a su propuesta.
1.0 por asegurar el recorrido de todo el grafo con DFS usando la versión iterativa.

(b) [3 ptos.] Un algoritmo alternativo para determinar un orden topológico en un grafo dirigido $G = (V, E)$ es extraer un nodo sin aristas que apunten a él, eliminar todas sus aristas, y repetir el proceso hasta que no queden nodos.

(i) [2 ptos.] Proponga el pseudocódigo de este algoritmo de forma que en tiempo $\mathcal{O}(V + E)$ entregue un orden topológico para un grafo acíclico. Especifique qué estructura(s) de datos necesita.

Solución.

```
Algoritmo(V, E):
    L ← lista ligada vacía
    while V ≠ ∅ :
        u ← Extract(Q)
        v ← nodo de V sin aristas incidentes Extraer v de V
        for e ∈ E :
            if v es mencionado en e :
                Extraer arista e de E
        Insertar v en L
    return L
```

Puntajes.
2.0 por algoritmo.

(ii) [1 pto.] Explique cómo este algoritmo puede detectar un ciclo en G. No requiere dar pseudocódigo.

Solución.

Si L no contiene todos los nodos de V, entonces G tiene un ciclo dirigido.

Puntajes.
1.0 por estrategia para deducir presencia de ciclo.

3. Árboles de cobertura

(a) [3 ptos.] Considere un grafo no dirigido G y un árbol de cobertura de costo mínimo T para G. Suponga que se añade un conjunto de k nodos adicionales con ciertas aristas, resultando en un grafo conexo G'. Explique cómo obtener un MST para G' a partir de T e indique la complejidad de su propuesta. No necesita entregar pseudocódigo.

Solución.

Sean E' y V' los conjuntos de aristas y nodos nuevos para agregar a G. Para decidir qué aristas de E' se deben incluir para extender el árbol T, podemos usar el algoritmo de Prim, escogiendo la arista de menor costo que conecta con un nodo no conectado previamente a T. En definitiva, es continuar el

proceso normal del algoritmo de Prim.

La complejidad de este algoritmo es entonces $\mathcal{O}(E' \log(V'))$ si consideramos que la cardinalidad de los conjuntos mencionados es arbitraria.

Puntajes.
2.0 por explicación de cómo extender T.
1.0 por indicar complejidad de la propuesta.

(b) [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.

Solución.

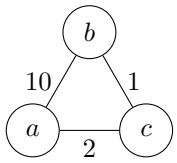
La línea 3 corresponde a la operación más costosa de este algoritmo. Para determinar eficientemente si agregar una arista forma o no un ciclo, podemos usar la estructura de conjunto disjunto para almacenar los conjuntos de nodos que están conectados, tal como en Kruskal.

Puntajes.
1.0 por indicar estructuras adecuadas.

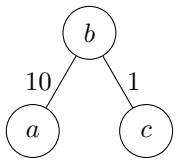
(ii) [2 ptos.] Demuestre que `CasiKruskal` no retorna un MST para todo grafo G.

Solución.

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

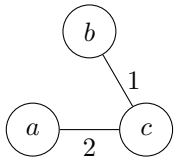


Si el orden de iteración de aristas en la línea 2 del algoritmo es $\{a, b\}, \{b, c\}, \{a, c\}$, el árbol obtenido por `CasiKruskal` es



costo 11

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



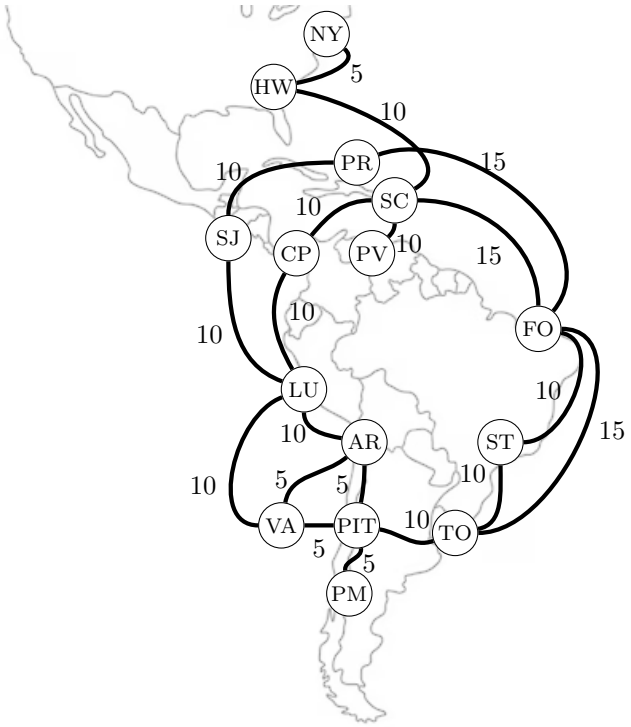
costo 3

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

Puntajes.
1.0 por indicar ejemplo concreto.
1.0 por mostrar óptimo diferente al obtenido.

4. Rutas más baratas

La red de fibra óptica terrestre y submarina que conecta nuestros computadores puede verse como una grafo no dirigido. En este contexto, se pueden utilizar los protocolos OSPF y RIP para determinar las rutas a utilizar. Considere el siguiente fragmento de la red, donde se indican las latencias como costos en aristas.



NY	Brookhaven, NY, EEUU
HW	Hollywood, FL, EEUU
PR	San Juan, PR, EEUU
SC	St. Croix, Virgin Islands, EEUU
PV	Puerto Viejo, Venezuela
CP	Colón, Panamá
SJ	Puerto San José, Guatemala
FO	Fortaleza, Brazil
ST	Santos, Brazil
LU	Lurín, Perú
TO	Las Toninas, Argentina
AR	Arica, Chile
VA	Valparaíso, Chile
PM	Puerto Montt, Chile
PIT	Santiago, Chile

(a) [1 pto.] Los protocolos de ruteo OSPF y RIP utilizan los algoritmos de Dijkstra y Bellman-Ford, respectivamente. Indique si es necesario realizar alguna modificación al grafo mostrado para poder aplicar estos protocolos. Justifique brevemente.

Solución.

La única consideración es notar que cada arista no dirigida puede verse como 2 aristas dirigidas en sentidos opuestos. Sobre este grafo dirigido, se pueden usar ambos algoritmos sin problema.

Puntajes.
1.0 por indicar que se pueden ocupar los algoritmos pues el grafo es dirigido de forma implícita.

(b) [3 ptos.] Utilizando el algoritmo de Dijkstra, determine la ruta más barata desde Santiago hasta Lurín. Para esto, indique el estado del arreglo de costos d y ancestros π al término de cada iteración, hasta obtener el óptimo pedido. Puede usar las etiquetas de los nodos como índices para simplificar su trabajo.

Solución.

Indicamos en la siguiente tabla los pares $d[u], \pi[u]$ para cada nodo u. Las columnas representan iteraciones y las celdas vacías representan $(\varnothing, \infty)$. Las celdas con **- -** indican que el nodo ya se extrajo de la cola de prioridad y sabemos que ya se tiene su óptimo.

	iteración					
nodo	0	1	2	3	4	5
PIT	$\varnothing, 0$	- -	- -	- -	- -	- -
AR		PIT, 5	- -	- -	- -	- -
VA		PIT, 5	PIT, 5	- -	- -	- -
TO		PIT, 10	PIT, 10	PIT, 10	PIT, 10	- -
PM		PIT, 5	PIT, 5	PIT, 5	- -	- -
LU			AR, 15	AR, 15	AR, 15	AR, 15
ST						TO, 20
FO						TO, 25

Dado que Dijkstra conoce el costo óptimo al extraer cada nodo de la cola, sabemos que el óptimo para ir de PIT a LU tiene costo 15 y corresponde al camino

$$\text{PIT}, \ldots, \pi[\pi[\text{LU}]], \pi[\text{LU}], \text{LU} \quad \text{es decir} \quad \text{PIT, AR, LU}$$

Puntajes.
2.0 por utilizar las iteraciones necesarias hasta extraer LU de la cola.
1.0 por obtener el camino óptimo a partir de los ancestros.
Observación. se pedía explícitamente mostrar las iteraciones hasta obtener el óptimo. Esto no se podía realizarse en menos de 5 iteraciones para el grafo dado.

(c) [2 ptos.] Considere que se quiere privilegiar el uso de las aristas (LU,CP) y (SJ,PR) cambiando su peso por -30 y -25 respectivamente. Determine si con este cambio es posible usar el algoritmo de Bellman-Ford para determinar la ruta más barata de Santiago a Lurín. Justifique su respuesta.

Solución.

Al modificar las aristas indicadas, existe un ciclo de costo negativo dado por

$$\text{LU, CP, SC, FO, PR, SJ, LU}$$

que es alcanzable desde PIT. Luego, el algoritmo de Bellman-Ford no se puede utilizar para obtener un camino de costo mínimo desde PIT hasta LU.

Puntajes.
1.0 por indicar la existencia de ciclo de costo negativo.
1.0 por indicar que no se puede usar Bellman-Ford debido a ello.