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

IIC2133 — Estructuras de Datos y Algoritmos 1'2023

# Interrogación 3

4 de julio 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

La atención al afiliado en la sucursal de una Isapre requiere que, para ser atendidos, éstos deben registrar su RUT y el tipo de atención solicitada (venta de bonos, reembolsos, pago de cotizaciones, cambio de planes, etc.) en un tótem de auto-atención. Éste entrega un número secuencial ascendente, los que son invocados en orden por los ejecutivos de la sucursal al estar disponibles para atender al afiliado.

(a) [2 ptos.] El tótem utiliza la función `new`$(A, b, \texttt{tipo}, \texttt{sec})$ para registrar en $A$ los datos del afiliado $b$ que solicita atención con secuencia `sec` entregada por el tótem. El sistema utiliza la llamada $b =$ `next`$(A)$ para sacar de $A$ al siguiente afiliado $b$ a atender según la secuencia. Escriba el pseudocódigo de ambas funciones e indique explícitamente qué estructura de datos utiliza para representar $A$.
**Solución.**
Utilizando como estructura una cola FIFO implementada como lista ligada, los métodos son

```
new(A, b, tipo, sec):
1    b.tipo ← tipo
2    b.sec ← sec
3    Insert(A, b)


next(A):
1    return Remove(A, 0)
```

donde `Insert`$(A, b)$ agrega $b$ al final de la lista ligada $A$, y `Remove`$(A, 0)$ retorna el primer elemento (cabeza) de la lista ligada $A$ luego de extraerlo.

**Puntajes.**
0.5 por especificar qué estructura utilizar.
0.5 por almacenar información del afiliado
0.5 por pseudocódigo de método para insertar
0.5 por pseudocódigo de método para extraer
*Observación:* También se puede utilizar un MIN-heap donde la medida de prioridad es el atributo `sec`.

(b) [3 ptos.] La gerente de la oficina desea mejorar la atención de los adultos mayores ($> 65$ años), considerando que al registrar al afiliado es posible incluir en $b$ su edad (en meses según $b$.`edad`). Pide entonces promover que los adultos mayores sean atendidos primero, de mayor a menor edad, y luego los demás afiliados según la secuencia entregada por el tótem. Modifique el pseudocódigo de las funciones de (a) e indique explícitamente qué estructura de datos usa para $A$. Puede suponer que no hay dos afiliados con la misma edad en meses. *Pista*: La secuencia del tótem es creciente no negativa.
**Solución.**
En este caso, se utiliza un MIN-heap donde la prioridad se almacena en el atributo `sec`. Para adultos

mayores, este valor se reemplazará por su edad en negativo, de forma que el atributo `sec` por sí solo sirve como prioridad del MIN-heap. Con esto, los métodos quedan como sigue

```
new(A, b, tipo, sec):
    b.tipo ← tipo
    if b.edad < 65 · 12 :
        b.sec ← sec
    else:
        b.sec ← (−1) · (b.edad)
    Insert(A, b)

next(A):
    return Extract(A)
```

donde `Extract`$(A)$ es el método de extracción de raíz en MIN-heap.

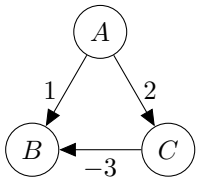
**Puntajes.**
0.5 por especificar qué estructura utilizar.
1.0 por pseudocódigo de método para insertar
0.5 por pseudocódigo de método para extraer
*Observación:* También se pueden usar dos heaps. Lo importante es que se debe utilizar un heap o estructura similar, debido a que la prioridad ya no es solo orden de llegada.

(c) [1 pto.] Al medir el comportamiento de la mejora anterior se observa que el tiempo promedio de espera para ser atendido depende del tipo de atención de quienes llegaron antes, ya que algunas atenciones son rápidas (venta de bonos) y otras lentas (cambio de planes). Un consultor propone aplicar la estrategia SJF (*Short Job First*, Tarea Más Corta Primero) para privilegiar a los que requieren atenciones breves. Así, se puede usar un factor $0 \leq f \leq 1$ donde 1 representa la tarea más breve. Indique (no se requiere el pseudocódigo) qué debería modificar en su respuesta de (b) para incorporar este cambio.
**Solución.**
Se debe actualizar la forma en que se calcula la prioridad en el método `new`. Una estrategia es dividir la prioridad por el factor $f$ de manera que las tareas más largas tienen mayores valores de prioridad luego del cambio.

**Puntajes.**
1.0 por describir qué se debe modificar para lograr la nueva prioridad
*Observación:* También se puede usar un sistema en que se use como prioridad principal el valor $f$ en un MAX-heap, y mantener la estructura original para la prioridad edad-secuencia. Lo importante es identificar el cambio de prioridad.

# 2. Distancias más baratas

(a) [3 ptos.] Demuestre que el algoritmo de Dijkstra no siempre es óptimo cuando el grafo dirigido sobre el que opera tiene costos negativos.
**Solución.**
Consideremos el grafo




El algoritmo de Dijkstra entrega como solución de $A$ hasta $B$ el camino $A, B$ con costo 1, mientras que el óptimo real es el camino $A, C, B$ con costo $-1$.

**Puntajes.**
1.0 por construir contraejemplo
1.0 por calcular camino entregado por Dijkstra para algún par de vértices
1.0 por mostrar el camino óptimo real entre esos dos nodos, logrando una contradicción

(b) [3 ptos.] Sea $G$ un grafo dirigido con costos.

(i) [2 ptos.] Proponga el pseudocódigo de un algoritmo que entregue una lista con los nodos de algún ciclo de costo negativo, en caso que $G$ tenga tal ciclo. En caso contrario, retorna una lista vacía.
**Solución.**
Utilizando como estructura una cola FIFO implementada como lista ligada, los métodos son

```
NegativeCycle(s):
1   for u ∈ V :
2       d[u] ← ∞;  π[u] ← ∅
3   d[s] ← 0
4   for k = 1 ... |V| − 1 :
5       for (u, v) ∈ E :
6           if d[v] > d[u] + cost(u, v) :
7               d[v] ← d[u] + cost(u, v)
8               π[v] ← u
9   for (u, v) ∈ E :
10      if d[v] > d[u] + cost(u, v) :
11          L ← lista con elemento v
12          current ← u
13          while current ≠ v :
14              agregar a L el nodo current
15              current ← π[current]
16  return Lista vacía
```

Notemos que el algoritmo es esencialmente `ValidBellmanFord`, al cual se le cambia su retorno booleano. En caso que exista ciclo negativo (resultado `false` del algoritmo visto en clases), se construye la lista visitando ancestros del nodo que genera el ciclo. En caso contrario, se retorna lista vacía.

Ahora, este método solo indica ciclo negativo desde una fuente. Si existe un ciclo negativo, pero no es alcanzable desde un nodo específico, tenemos que asegurarnos de detectarlo. Para esto, se puede ejecutar este método desde todos los vértices. Es decir

```
NegativeCycleGlobal():
1   for u ∈ V :
2       L ← NegativeCycle(u)
3       if L ≠ ∅ :
4           return L
5   return Lista vacía
```

**Puntajes.**
1.5 por algoritmo desde una fuente
0.5 por llamarlo desde diferentes fuentes

(ii) [1 pto.] Determine la complejidad de su algoritmo.
**Solución.**
El algoritmo de detección desde una sola fuente `NegativeCycle` es $\mathcal{O}(EV)$ al igual que Bellman-Ford. Los $V$ llamados en el método global hacen que esta solución sea $\mathcal{O}(EV^2)$
**Puntajes.**
0.5 por complejidad del primer método
0.5 por complejidad del segundo método

# 3. Árboles de cobertura de costo mínimo

Una estrategia base para asegurar cobertura de comunicaciones es contar con diferentes servicios de conexión. Por ejemplo, el servicio de Tesorería (TGR) puede conectar sus oficinas y puntos de atención mediante: Redes de fibra de proveedores, Redes de telefonía móvil, Red StarLink, Redes Satelitales, Radioaficionados y Banda Local. Naturalmente cada uno de ellos tienen diferentes costos, velocidades y latencias, y pueden estar disponibles o no entre diferentes oficinas y puntos de atención, o perderse al ocurrir una emergencia.

(a) [1 pto.] ¿De qué forma puede TGR decidir la mejor forma de conectar sus oficinas y puntos de atención en función de los servicios de conexión disponibles en cada oficina o punto? Indique la forma de representar el problema y las estrategias conocidas para determinar la solución. *Pista*: Defina primero cómo determinar la "mejor" conexión en cada caso.
**Solución.**
El problema se puede representar como un grafo no dirigdo con costos en las aristas. Los nodos son oficinas y puntos de atención y las aristas son conexiones existentes entre nodos, con una arista por cada tipo de conexión disponible (puede haber más de una arista entre dos nodos). El cost ode cada arista es una función que relaciona costo, velocidad y latencia como un valor que describe el costo real de un tipo de conexión entre nodos. La mejor forma de conectar, basándose en minimizar costo total, es determinar el MST de este grafo con algún algoritmo estudiado como Prim o Kruskal.

**Puntajes.**
0.5 por describir estructura del grafo: si es no-dirigido/dirigido, qué describen los costos y qué son los nodos
0.5 por usar un algoritmo para determinar MST
*Observación:* El grafo puede tener otros detalles, pero lo esencial es representar los nodos y conexiones posibles. El MST es la solución esperada para lograr conectividad.

(b) [3 ptos.] Al ocurrir una emergencia, TGR pierde la conectividad entre dos oficinas a través de uno de los enlaces que es parte de la mejor forma de conectar ya escogida. Proponga una forma de recuperar la conectividad de la "mejor" forma disponible. Justifique por qué es la mejor opción.
**Solución.**
Al perder un enlace el MST del grafo se parte en dos MST independientes. Sabemos por la estrategia de construcción de los MST que ambos MST son óptimos para conectar los nodos que cubren. Luego, si consideramos que lo que tenemos al perder la arista es un "corte" entre los MST (como en Kruskal) si determinamos la siguiente arista de las disponibles que tiene el menor costo y no genera un ciclo podemos recuperar el MST con las aristas disponibles. **Puntajes.**
1.0 por describir el criterio que se usa para decidir qué arista(s) se agrega(n)
1.0 por argumentar que la solución propuesta cumple lo pedido
1.0 por argumentar en qué sentido es la "mejor" solución
*Observación:* Lo importante es resolver el problema. Puede ser estudiando qué arista escoger de las que conectan ambos árboles desconectados y se puede usar el costo como criterio o simplemente escoger una arista disponible. En este último caso se debe argumentar que se privilegia simplicidad por sobre costo.

(c) [2 ptos.] Si la emergencia afecta simultáneamente a múltiples enlaces de la forma de conectar ya escogida, ¿la estrategia que propuso en (b) sigue siendo válida? Argumente su respuesta.
**Solución.**
Sí, es posible generalizar la estrategia anterior cuando el MST del grafo pierde múltiples aristas que eran parte del MST. Podemos aplicar un enfoque similar para reconstruir el MST después de perder múltiples aristas. **Puntajes.**
1.0 por indicar si sirve el enfoque
1.0 por argumentación
*Observación:* Esto depende de la solución propuesta.

# 4. Búsqueda en amplitud

(a) [3 ptos.] En un laboratorio interesa almacenar una gran cantidad de residuos en la menor cantidad de recintos posible. Sabemos que ciertos residuos son incompatibles entre sí y no pueden almacenarse en el mismo recinto por su riesgo de explosiones o reacciones químicas.

(i) [1 pto.] Describa cómo modelar con grafos el escenario de incompatibilidades de residuos, de tal forma que sirva para resolver el inciso (ii). Específicamente, indique qué tipo de grafo utilizará, qué representan los nodos y las aristas y en qué estructura de datos almacenará el grafo.
**Solución.**
Se puede usar un grafo no dirigido, sin costos, de forma que los nodos son residuos y las aristas indican que los nodos conectados son incompatibles. El grafo se puede almacenar como listas de adyacencias o matriz de adyacencias indistintamente.

**Puntajes.**
0.25 por describir nodos
0.25 por describir qué reprsentan las aristas (a quiénes se conectan)
0.25 por describir dirección de las aristas
0.25 por describir qué estructura se usa.

(ii) [2 pto.] Proponga el pseudocódigo de un algoritmo que opere sobre su grafo propuesto en (i) y que retorne un booleano indicando si es posible o no almacenar todos los residuos en 2 recintos. Su algoritmo debe ejecutarse a lo más en tiempo lineal respecto al tamaño del grafo.
**Solución.**
Dado que el grafo conecta a los incompatibles, buscamos ver si el grafo es 2-coloreable (también llamado bipartito). Esto lo podemos hacer modificando BFS para que además de sus atributos, incluya un atributo `flag` para los nodos, el que podrá ser 0 o 1, indicando los dos colores posibles. Cada color representará un recinto. El algoritmo comienza coloreando con 0 y cada vecino será 1. Si el nodo actual es 1, sus vecinos deben ser 0. Cuando se detecta un vecino ya coloreado y que tiene el mismo color, significa que el grafo no es 2-coloreable.

```
2Col(s):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
        u.flag ← 0
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    Q ← cola vacía
    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
                v.flag ← (1 − u.flag)
                π[v] ← u
                Insert(Q, v)
            else:
                if v.flag = u.flag :
                    return false
        u.color ← negro
    return true
```

Notemos que el algoritmo es esencialmente igual a BFS, pero se incluye el `flag` y este se chequea cuando el nodo vecino ya ha sido visitado (no es blanco). Si en ese caso, su flag es igual al nodo actual, significa que no es posible asignarlos en 2 recintos diferentes, representados por el `flag`.

**Puntajes.**
1.0 por incluir color/flag adicional al que usa BFS por defecto, de manera que solo tenga 2 opciones

1.0 detectar correctamente coincidencia de colores o ausencia de esta
*Observaciones*: Pueden no escribir BFS completo y solo indicar qué cambios hacerle.

(b) [3 ptos.] En un grafo no dirigido $G = (V, E)$, la comunidad de radio $k$ de un nodo $u$ es el conjunto de nodos que están a lo más a distancia $k$ de $u$. Por ejemplo, la comunidad de radio 0 de $u$ es $\{u\}$, mientras que la comunidad de radio 1 de $u$ es $\{v \in V \mid \{u, v\} \in E\}$. Proponga el pseudocódigo de un algoritmo que retorne la comunidad de radio $k$ de un nodo $u$ de $G$ en tiempo a lo más lineal en el tamaño del grafo.

**Solución.**
Para resolver este problema, basta con ir acumulando los nodos descubiertos, recordando en qué "nivel" estamos. Comenzamos con nivel 0 y cada vez que visitamos vecinos, aumentamos en 1.

```
Radius(s, k):
    for u ∈ V − {s} :
        u.color ← blanco;  u.δ ← ∞;  π[u] ← ∅
    s.color ← gris;  s.δ ← 0;  π[s] ← ∅
    m ← 0
    L ← lista con nodo s
    Q ← cola vacía
    Insert(Q, s)
    while Q no está vacía :
        u ← Extract(Q)
        if m ≤ k :
            for v ∈ α[u] :
                if v.color = blanco :
                    v.color ← gris;  v.δ ← u.δ + 1
                    π[v] ← u
                    Insert(Q, v)
                    Agregar a L el nodo v
            u.color ← negro
            m ← m + 1
        else:
            Break
    return L
```

El algoritmo es esencialmente igual a BFS, pero se agregan nodos hasta que el número de niveles ($m$) supere la cota $k$ indicada como argumento.

**Puntajes.**
1.0 por variable que cuenta distancia desde la fuente
1.0 agregar nodos la primera vez que se visitan (no repetir)
1.0 Entregar lista en el momento adecuado, cuando se ha llegado al radio $k$
*Observación*: se puede usar también el arreglo $d$ sin requerir una variable nueva. Es más engorroso, pero resuelve el problema en la misma complejidad. Además, pueden no escribir BFS completo y solo indicar qué cambios hacerle.