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

IIC2133 — Estructuras de Datos y Algoritmos 2'2022

Interrogación 3

7 de diciembre de 2022

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

Puntajes y nota. Cada pregunta tiene 6 puntos más un punto base. La nota de la interrogación será el promedio de las notas de las 3 preguntas entregadas.

Uso de algoritmos. De ser necesario, en sus diseños puede utilizar llamados a cualquiera de los algoritmos vistos en clase. No debe demostrar la correctitud o complejidad de estos llamados.

1. Algoritmos codiciosos

Dado un intervalo cerrado $[s, f]$ considere un conjunto de intervalos cerrados no necesariamente disjuntos $S = \{[s_i, f_i] \mid i = 1, \ldots, n\}$ tales que $s \leq s_i < f_i \leq f$ para todo i. El conjunto $\Omega \subseteq S$ es un cubrimiento de $[s, f]$ si la unión de sus elementos contiene a $[s, f]$ y diremos que es un cubrimiento óptimo si es el cubrimiento más pequeño (en cantidad de intervalos) para el S dado.

(a) [2 ptos.] Muestre un intervalo $[s, f]$ y un conjunto S que sirvan como contraejemplo para la siguiente estrategia codiciosa que no es óptima: *el siguiente intervalo de S que se escoge es $[s_i, f_i]$ tal que se solapa con el último intervalo escogido y s_i es el menor posible.* Se espera que compare el óptimo verdadero con la solución encontrada por la estrategia propuesta.

Solución. Sea el intervalo $[s, f] = [0, 3]$ y el conjunto de intervalos $S = \{[0, 2], [1, 2], [2, 3]\}$.

- La estrategia propuesta escoge los siguientes intervalos en orden
  - $[0, 2]$, pues tiene el menor s_i posible
  - $[1, 2]$, pues de los dos que se solapan con el escogido antes, es el que empieza primero
  - $[2, 3]$, pues es el que queda y llega al extremo del intervalo

  Es decir, $|\Omega| = 3$ con esta estrategia. Notamos que la unión de estos intervalos cubre $[0, 3]$ por completo, por lo que es un cubrimiento válido.
- Por inspección notamos que un cubrimiento más pequeño es $\Omega^* = \{[0, 2], [2, 3]\}$ que tiene $|\Omega^*| = 2$, por lo que Ω no es óptimo.

Asignación de puntajes.
1 por definición de $[s, f]$ y S 0.5 por obtener el cubrimiento dada la estrategia indicada 0.5 por mostrar un cubrimiento más pequeño
Observación. Se admiten casos que operen diferente. Lo importante es que haya una discrepancia. En la prueba se corrigió el enunciado, pero no a tiempo. Se espera una comprensión global del problema solamente.

(b) [3 ptos.] Proponga el pseudocódigo de un algoritmo codicioso que efectivamente encuentre un cubrimiento óptimo para $[s, f]$ y S cualesquiera. No necesita demostrar la correctitud de su estrategia codiciosa. *Hint:* escoja el intervalo que se solapa con el último escogido y que tiene mayor f_i.

Solución.

input : límite inferior s, límite superior f, conjunto de intervalos S

`Greedy`(s, f, S)**:**

```
1  S ← S ordenado por f_i
2  Ω ← ∅
3  t ← s
4  while t < f :
5      f_i ← máx{f_j | s_j ≤ t ≤ f_j}
6      Ω ← Ω ∪ {i}
7      t ← f_i
8  return S
```

Asignación de puntajes.
1 por ordenar de alguna forma los intervalos (o equivalente)
1 por la implementación de la estrategia que decide según extremo superior
1 por construcción del cubrimiento
Observación. Se admiten soluciones diferentes y el puntaje es a criterio del corrector. Lo esencial es la estrategia codiciosa. Los detalles pueden variar y no deben perjudicar la nota si se ve que hay una comprensión del problema.

(c) [1 pto.] Determine la complejidad de su algoritmo.

Solución.

Si tomamos como parámetro de complejidad el número de intervalos n en S, entonces

- Ordenar S toma $\mathcal{O}(n \log(n))$ con algún algoritmo conocido (e.g. Quicksort)
- El loop **while** itera $\mathcal{O}(n)$ veces en el peor caso
- La deducción del máximo que se solapa toma en el peor caso $\mathcal{O}(n)$
- Con esto, el loop completo aporta $\mathcal{O}(n^2)$

El algoritmo mostrado toma $\mathcal{O}(n^2)$.

Asignación de puntajes.
0.2 por ordenación
0.2 por loop
0.2 por deducción del máximo
0.4 por concluir
Observación. Depende solo del algoritmo propuesto. Si no es correcto, de todas formas este inciso tiene puntaje si la complejidad es la que corresponde a su propuesta.

2. Programación dinámica

(a) Como usted bien sabe, la serie de Fibonacci es una secuencia de números enteros donde el siguiente entero de la serie es la suma de los dos anteriores, la que se define según

$$f_i = \begin{cases} 0 & \text{si } i = 0 \\ 1 & \text{si } i = 1 \\ f_{i-1} + f_{i-2} & \text{si } i \geq 2 \end{cases}$$

(i) [3 ptos.] Proponga en pseudo código una función `Fibonacci`(n), utilizando programación dinámica, que permita generar de forma eficiente un arreglo con **todos** los números de la secuencia de Fibonacci hasta un número n dado. **Solución.**

input : natural $n \geq 2$

`Fibonacci`(n)**:**

```
1  M ← arreglo vacío con n celdas
2  M[0] ← 0
3  M[1] ← 1
4  for j = 2...n :
5      M[j] ← M[j − 1] + M[j − 2]
6  return M
```

Asignación de puntajes.
1 por inicialización del arreglo
2 por implementar la definición recursiva usando llamados almacenados

(ii) [1 pto.] Justifique por qué su solución anterior es una aplicación de programación dinámica. **Solución.**

Como el problema tiene subestructura óptima y los valores más pequeños se almacenan, cuando son usados para calcular un valor mayor se evita hacer cálculos repetidos. Esta es la definición de programación dinámica. En particular, en la línea 5 del algoritmo se implementa. **Asignación de puntajes.**
0.5 por mencionar conceptos como subestructura óptima y cálculos solapados o similar
0.5 por indicar dónde su algoritmo implementa esta idea

(b) [2 ptos.] Sabemos que el problema de "dar vuelto" es posible abordarlo utilizando programación dinámica. ¿Qué condiciones cumple dicho problema que permite ser abordado de dicha forma?

Solución.

El problema exhibe subestructura óptima, es decir, una instancia puede descomponerse en problemas más pequeños. Al resolver los más pequeños, sus óptimos se pueden usar para generar el óptimo de la solución principal. Adicionalmente, estos subproblemas se solapan, es decir, una estrategia recursiva clásica calcula varias veces los mismos subproblemas para resolver un problema dado. Esto permite usar una estrategia de programación dinámica para (1) resolver el problema con subproblemas y (2) guardar resultados de instancias pequeñas para no computar varias veces el mismo subproblema.

Asignación de puntajes.
1 por subestructura óptima
1 por solapamiento y reutilización de soluciones a subproblemas

3. Heaps

Como parte de la búsqueda de una Inteligencia Artificial General (AGI) un grupo de aficionados propone utilizar los miles (n) de modelos especializados de AI existentes (narrow IA) en paralelo, enviándoles la misma "pregunta" al mismo tiempo, y elegir como "respuesta" aquella con mayor confiabilidad (la respuesta de cada modelo viene acompañada de un valor entre 0 y 1 que indica la confiabilidad de la respuesta, siendo 1 el 100 % de confianza). Actualmente han logrado construir el software necesario para enviar la pregunta a los modelos especializados y almacenar las respuestas en un arreglo $A[0 \ldots n-1]$, en orden de llegada para los primeros 3 segundos.

(a) [2 ptos.] Proponga un algoritmo en pseudo código para la función `Answer`(A) que permite obtener de la manera más eficiente la respuesta de mayor confianza desde el arreglo A sin eliminarla.

Solución. Suponemos que los datos se almacenan con atributos `respuesta` y `confiabilidad`. Además, luego de recibir los datos iniciales, se aplica `BuildHeap`(A) para transformar al arreglo A en un heap binario, donde se usa el atributo `confiabilidad` como prioridad. Con estos supuestos,

input : Arreglo A que representa un heap binario

`Answer`(A)**:**
1 **return** $A[0]$.*respuesta*

Asignación de puntajes.
1 por supuestos iniciales
1 por algoritmo
Observación. hay libertad en el abordaje de la pregunta. Lo importante es que se construye el heap en alguna etapa previa, dado que los datos en A solo están ordenados por tiempo de llegada, no por confiabilidad.

(b) [2 ptos.] Un caso de uso del sistema es que el usuario pueda pedir la "siguiente mejor respuesta" al sistema, eliminándola del arreglo. Proponga el pseudo código para la función `nextAnswer`(A).

Solución. Dado el enunciado, puede interpretarse como *entregar el elemento más prioritario* o *entregar el segundo más prioritario*. Ambos se consideran correctos para efectos de la corrección. La implementación de ambos casos es

input : Arreglo A que representa un heap binario

`nextAnswer`(A)**:**
1 **return** `Extract`(A).*respuesta*

`nextAnswer`(A)**:**
2 **if** $A[1]$.*confiabilidad* $\geq A[2]$.*confiabilidad* **:**
3 **return** `Extract`$(A, 1)$.*respuesta*
4 **return** `Extract`$(A, 2)$.*respuesta*

En el segundo caso, se asume que `Extract`(A, i) opera de la misma forma que `Extract` visto en clase, pero extrayendo la posición indicada y reestableciendo la propiedad de heap.

Asignación de puntajes.
2 puntos por algoritmo
Observación. No es necesario dar detalles de la implementación de los métodos de heaps en caso de haberlos modificado. Basta con mencionar en qué son diferentes. Errores en esas implementaciones no perjudican la nota. Lo importante es comprender el uso de `Extract`.

(c) [2 ptos.] Dado que algunos modelos especializados son más lentos en responder se decide permitir que el arreglo A pueda recibir respuestas "tardías" (después de los 3 segundos iniciales) para ser consideradas en los algoritmos anteriores (una vez que dichas respuestas están disponibles). Proponga el pseudo código para la función `lateAnswer`$(A,$ `respuesta`$,$ `confiabilidad`$)$ que permite incorporar de manera eficiente una respuesta nueva al arreglo A una vez que este ya fue "consultado" por los algoritmos anteriores.

Solución.

input : Arreglo A que representa un heap binario, respuesta y confiabilidad

`lateAnswer`$(A, respuesta, confiabilidad)$**:**
1 $i \leftarrow$ primera celda vacía de A
2 $A[i]$.`respuesta` $\leftarrow$ *respuesta*
3 $A[i]$.`confiabilidad` $\leftarrow$ *confiabilidad*
4 `SiftUp`(A, i)

Asignación de puntajes.
2 por algoritmo

Observación. Tal como en las otras preguntas, lo importante es comprender las operaciones esenciales de heaps.

4. Algoritmos en grafos

En los siguientes incisos se espera que modifique o invoque como subrutinas algoritmos conocidos.

(a) [2 ptos.] Considere un grafo dirigido $G = (V, E)$ que en lugar de aristas con pesos tiene nodos con pesos. El problema de rutas más cortas desde una fuente se define análogamente, donde el costo de un camino es la suma de los costos de sus nodos. Proponga cómo modificar el grafo para poder utilizar el algoritmo de Dijkstra para resolver dicho problema con la misma complejidad vista en clase.

Solución.

Dado un grafo con pesos en sus nodos, podemos modificar el grafo incluyendo pesos en sus aristas de acuerdo al nodo de partida:

```
CreateWeights(V, E):
1    for v ∈ V :
2        for (x, y) ∈ E :
3            if v = x :
4                cost(x, y) ← cost(v)
```

Luego de ejecutar este algoritmo, las aristas de salida de los nodos tienen como costo el de su nodo inicial. Luego, usamos Disjktra como siempre para encontrar rutas más cortas. Si se interpreta el problema como el de ecncontrar el costo del camino, entonces basta sumar al costo encontrado por Dijkstra el costo del nodo fuente.

Asignación de puntajes.
1 por modificar grafo
1 por mencionar el uso de Dijkstra sobre el grafo nuevo
Observación. Si el problema que se resuelve es el de *encontrar ruta más corta*, no es necesario el comentario del costo del extremo. Si se resuelve *encontrar el costo óptimo*, se incluye el costo del extremo. Importante: también funciona considerar como costo el del nodo de llegada. En tal caso, el costo del camino total se obtiene añadiento el costo del nodo final.

(b) [2 ptos.] El diámetro de un árbol no dirigido conexo T se define como el largo del camino más largo en T. Suponga que existe un único camino de largo máximo en T con extremos u y v. Si x es un nodo cualquiera, se puede demostrar que el nodo más lejano a x es u o v. Usando este resultado, proponga un algoritmo que determine el diámetro de un árbol T.

Solución.

La propiedad descrita permite plantear el siguiente algoritmo

```
Diameter(V, E):
1    x ← cualquier nodo de V
2    d ← BFS(x)
3    u ← nodo que tiene máximo d[·]
4    d ← BFS(u)
5    D ← máx{d[v] | v ∈ V}
6    return D
```

donde se usa una versión modificada de `BFS` para que retorne el arreglo con distancias desde la fuente, así como el nodo asociado a cada distancia.

Asignación de puntajes.
1 por uso de BFS o Dijkstra

1 por obtener el diámetro usando distancia entre los nodos más lejanos entre sí
Observación. Pueden usar Dijkstra también. Con BFS es suficiente.

(c) [2 ptos.] Un multigrafo G es un grafo en el cual pueden haber más de una arista entre dos vértices. Considere un multigrafo G donde cada arista tiene un costo w_i y una etiqueta l_i. Proponga un algoritmo para encontrar un árbol de cobertura de costo mínimo para G y reportar las etiquetas de sus aristas.

Solución.

Si se almacenan las aristas de forma que cada una tiene sus nodos, costo y además etiqueta, se puede usar Kruskal o Prim para resolver el problema. No es necesario realmente modificar el algoritmo si las aristas se guardan juntas.

Asignación de puntajes.
1 por uso de Kruskal o Prim
1 por mencionar cambios/cambios no necesarios (depende de su propuesta).