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

IIC2133 — Estructuras de Datos y Algoritmos
2022 - 1

Pauta Interrogación 3

Pregunta 1

a) La idea es utilizar un grafo, pero donde se dé la posibilidad de que entre dos nodos A y B hayan múltiples aristas que vayan desde A hasta B. Cada una de estas aristas, representa el tiempo y el costo de viaje entre dos lugares usando un medio de transporte específico. Luego si se puede llegar utilizando más de un medio de transporte desde A hasta B, habrá más de una arista que conecta A con B. Es importante que recalquen de alguna forma o que noten que el grafo tiene que ser dirigido.
Para implementarlo en código, se puede utilizar una pequeña variación a listas de adyacencia. Normalmente, en una lista de adyacencia cada nodo guarda una lista de los nodos a los que tiene aristas. Para este caso como pueden haber múltiples aristas entre nodos y cada una con datos distintos, podemos hacer que cada nodo guarde una lista con las aristas que salen de él, donde cada arista tiene los siguientes atributos:

- start: nodo de salida
- end: nodo de llegada
- transportation: medio de transporte que utiliza
- time: tiempo al utilizar esta ruta
- cost: costo

Puntaje:

- (0.5 ptos): por proponer estructura de datos
- (0.5 ptos): por implementación en lenguaje de programación (pueden servir más implementaciones, queda a criterio del ayudante)

Descuentos:

- En caso de usar listas de adyacencia o otra estructura, pero que crear una para cada transporte en vez de una que englobe todos, asignar 0,6 ptos máximo.
- En caso de no especificar que se utilizan grafos dirigidos, asignar máximo 0,8 ptos. Si no menciona los grafos dirigidos, pero la estructura elegida es correcta no descontar.

b) Para calcular la mejor ruta podemos utilizar el algoritmo de Dijkstra, con las variaciones respectivas para la implementación mencionada en a). Además, como nos mencionan que el tiempo es oro, la mejor ruta será la que minimice el tiempo de viaje total. En la pregunta, tambien, se puede ponderar entre tiempo y costo, el alumno puede elegir bajo que criterios ponderar (si asignarle más valor al tiempo o al costo).

Algorithm 1 Dijkstra(s, V, α)

```
d ← Arreglo vacío del largo de V                    // Tiempo de llegada a cada nodo
π ← Arreglo vacío del largo de V                    // Arista por la cual se llegó al nodo
for u ∈ V do
    u.color ← white
    d[u] ← ∞
    π[u] ← null
end for
Q ← cola de prioridades
s.color ← gray
d[s] ← 0
Q.enqueue(s)
while !Q.empty() do
    u ← Q.dequeue()
    for e ∈ α[u] do
        v ← e.end
        if v.color == white or v.color == grey then
            if d[v] > d[u] + e.time then
                d[v] ← d[u] + e.time
                π[v] ← e
            end if
            if v.color == white then
                v.color ← grey
                Q.enqueue(v)
            end if
        end if
    end for
    u.color ← black
end while
return d, π
```

Algorithm 2 short_route(a, b, V, α)

```
d, π ← Dijkstra(a, V, α)
total_time ← d[b]
total_cost ← 0
T ← conjunto vacío                    // Medios de transporte utilizados en la ruta
v ← b
while π[b]! = null do
    e ← π[b]
    total_cost ← total_cost + e.cost
    T ← T ∪ e.transportation
    v ← e.start
end while
return total_time, total_cost, T
```

Después, para obtener el tiempo de viaje se puede revisar el arreglo d utilizado en el algoritmo, y para obtener el costo total y los medios de transporte utilizados se puede revisar las aristas que se usaron para llegar de A a B con el arreglo π (puede diferir dependiendo del pseudocódigo)

Puntaje:

- (0.2 ptos): por proponer Dijkstra
- (0.6 ptos): por pseudocódigo correcto, optimizando la ruta con menor tiempo
- (0.2 ptos): por llevar registro o explicar cómo obtener registro de medios de transporte, tiempo y costo total del viaje

Pregunta 2

A (1 pts)

La estructura de datos para modelas el proceso de alimentación es un heap. Las operaciones necesarias son las de insertar y extraer con sus subrutinas correspondientes. Por mencionar que es un heap (0.4 pts), por cada función (0.15 pts)

```
Extraer(H):
    i <- ultima celda no vacía de H
    best <- H[1]
    H[1] <- H[i]
    best <- Null
    sift down(H, 1)
    return best

sift down(H, i):
    if i tiene hijos:
        i' <- hijo mayor prioridad
    if i H[i'] > H[i]:
        H[i'] <-> H[i]:
        sift down(H, i')


insert(H, e):
    i <- primera celda desocupada de H
    H[i] <- e
    sift up(H, i)


sift up(H, i):
    if i tiene padre:
        i' <- padre de i
    if i H[i'] < H[i]:
        H[i'] <-> H[i]:
        sift up(H, i')
```

En caso de solamente realizar una búsqueda lineal para obtener el mejor cristal el puntaje es de (0.3 pts), dado que el foco de la pregunta es utilizar una estructura de datos apropiada para el problema.

B (1 pts)

Para mejorar la calidad global se tiene que aplicar un heap sobre la linea de producción de esta forma siempre se esta abasteciendo con los mejores cristales dado esa linea, en caso de que se consuma un cristal de una lineal el proceso P_i seguirá esperando a que se produzca el siguiente cristal, dado que tiene que comparar su valor con el resto almacenado en la línea de producción. Esto se suma al heap usado en la linea de alimentación.

Una forma alternativa es generar un heap sobre todas las lineas de producción en vez de tener un heap por linea y ordenarlas por prioridad.

Pregunta 3

a) (0,7 pts) Al observar el árbol podemos ver 2 casos el momento de imaginar un solución:

- La presidente NO está en la solución. Entonces podemos quitarla del grafo y nos quedan subárboles que tienen un nodo raíz y van creciendo. Es claro que el óptimo viene de optimizar cada uno de estos subárboles.
- La presidenta SI está en la solución. Entonces todos los hijos los podemos quitar, sabemos que no estarán en el óptimo por las reglas de la invitación. El óptimo viene de combinar la presidenta con los óptimos de los arboles que nos quedan.

Luego comparamos el caso que nos dio una solución más óptima y esto nos daría la solución.

b) (0,7 pts) Considere N como un nodo del árbol, definamos la siguientes funciones.

obtain_children(N), esta función retorna todos los hijos de un nodo N.

obtain_grandchildren(N), esto retorna una lista de todos los nietos de un nodo N.

finalmente nos queda la función que recibe un nodo y retorna el valor óptimo y una lista con la solución como tupla

Listing 1: Pesudocódigo

```
1   find_optimum(N)
2       if len(obtain_children(N)) == 0:
3           return N.value
4
5       value_1 = N.value
6       for gc in grandchildren(N):
7           value = find_optimum(gc)
8           value_1 += value
9
10      value_2 = 0
11      for c in children(N):
12          value = find_optimum(c)
13          value_2 += value
14
15      if value_1 > value_2:
16          return value_1
17      else:
18          return value_2
```

c) (0,6 pts) Se debe revisar el ejemplo. hay mucho ejemplos válidos.

Pregunta 4

Nota general CP: La idea de esta pregunta era usar DFS y Toposort

General

Se espera que se use DFS y Topological Sort. El alumno tiene que plantear el problema como un problema de grafos.

Parte A

¿Qué representan los nodos y las aristas? ¿Son las aristas direccionales o no direccionales?
Los nodos representan las carreras existentes. Las aristas representan la posibilidad de traspaso entre las carreras que conecta.
Las aristas son direccionales, ya que puedo pasar de una carrera A a una carrera B, pero no necesariamente de la carrera B a la carrera A.
Se espera que el alumno dibuje un grafo representante del problema (extensión del grafo arbitraria)

Parte B

Dado tu grafo de a), ¿qué significa que desde una carrera sea posible traspasarse a otra carrera, posiblemente pasando entre medio por otras carreras? ¿Cómo se puede saber si es así?
Que sea posible pasarse de una carrera a otra quiere decir que existe un camino desde la carrera de origen y la carrera de destino mediante otras carreras.
Para esto, es necesario conocer el orden de las carreras por las que tengo que pasar desde mi carrera de origen para llegar a mi carrera de destino. Se propone Topological sort, que permite ordenar un grafo de dependencias de manera que si requiero pasar por una carrera A antes que una carrera B, entonces este algoritmo entrega un orden lineal de manera que pase por A antes de llegar a B.
Se espera que se explique el funcionamiento de Topological Sort.

Parte C

Dado tu grafo de a), ¿qué significa que desde una carrera sea posible pasarse a cualquiera otra carrera? Y en realidad, ¿qué significa que desde cualquier carrera sea posible traspasarse a cualquiera otra? ¿Cómo se puede saber si es así?
Significa que existe un camino desde la carreara X para llegar a cualquiera de las otras, (ac'a argumentar mejor ns en vdd como explayarlo). La significancia de poder pasarnos de cualquier carrera a cualquier otra significa que el grafo tiene una componente fuertemente conexa, la forma de saber esto es ejecutar algoritmos que nos resuelven la componente fuertemente conexa, como kosaraju, o DFS, el primero nos entrega la componente fuertemente conexa, mientras que con el segundo recorreríamos todas los nodos, llegando al inicial, al guardar los nodos en otra estructura, como un heap o un dict, nos daríamos cuenta que están todas conectadas.