FACULTAD DE INGENIERIA
Universidad de Buenos Aires

Trabajo Práctico 1

[75.29] Teoria de Algoritmos I
Fecha de Entrega: 07/09/2022

Buceta, Belén	102121	bbuceta@fi.uba.ar

Índice

1. Introducción

El presente trabajo reune la resolución del primer trabajo práctico de la materia.
Más detalles del enunciado a resolver se detallan en:
https://algoritmos-rw.github.io/tda/2022-2c/tp1/.

2. Resolución

2.1. Parte 1: La fiesta del club de amigos

El club de amigos de la república Antillense prepara un ágape en sus instalaciones en la que desea invitar a la máxima cantidad de sus "n" socios. Sin embargo por protocolo cada persona invitada debe cumplir un requisito: Sólo puede asistir si conoce a al menos otras 4 personas invitadas

2.2. 1. Proponga una estrategia greedy óptima para resolver el problema

Con la menor complejidad espacial y temporal posible. Justifique su optimalidad. Justifique que sea greedy.

La estrategia que se propone es la siguiente.
En primer lugar consideraremos que a priori todos los socios del listado que integran el archivo brindado están invitados. O bien esa será nuestra hipótesis para comenzar el algoritmo.
Lo siguiente que haremos será buscar cuál de todos los socios posee la menor cantidad de conocidos, llamaremos *minimo* a la cantidad de socios conocidos del socio devuelto por esta búsqueda. Y una vez obtenido el socio en cuestión, se validará si este *mínimo* es menor a la cantidad mínima necesaria de conocidos para efectivamente poder ser invitado, a la que llamaremos *minimoNecesario*.
En este punto pueden ocurrir dos escenarios. Los detallamos a continuación.

1. El *minimo* es mayor al *minimoNecesario*, de manera que el socio en cuestión y los demás socios, todos ellos, cumplen la condición necesaria para ser invitados (puesto que si el socio en cuestión es el que tiene la menor cantidad de conocidos y aún así cumple la condición, pues mayor aún la cumplirán todos los demás). Finalmente, nuestro algoritmo termina pues se comprueba que los invitados son exactamente los N socios.

2. El *minimo* es menor al *minimoNecesario*, esto significaría que este socio no cumple con la condición necesaria para ser invitado. Por lo tanto es necesario actualizar el listado y "desactivar"la invitación correspondiente a este socio. A su vez, también será necesario disminuir la cantidad de conocidos invitados de los conocidos del socio en cuestión, puesto que éste ya no será invitado y por lo tanto dichos conocidos no lo tendrán como conocido dentro de los invitados.
   Una vez realizado esto, lo que nos interesará es no invitar al resto de invitados que no cumplen la condición, tal como ocurrió con este socio. Entonces lo que haremos será recalcular nuevamente el *minimo* de la lista y verificar nuevamente si cumple o no la condición necesaria para ser invitado. Nuevamente podría suceder cualquiera de las opciones detalladas: o bien la cumple o bien no la cumple. Si fuese la primera opción estaríamos frente a la opción detallada en 1, con la observación de que la cantidad final de socios invitados ya no sería N sino la cantidad restante que posee invitación activa, es decir, que no fue desactivada por no cumplir con la condición. Caso contrario, nos encontraríamos nuevamente en esta opción (2).

Justificación algoritmo greedy: El algoritmo que se describe anteriormente plantea una estrategia de búsqueda siguiendo una heurística que pretende obtener la solución óptima en cada paso (no

invitar al socio que no cumple con la condición requerida) y así llegar a una solución final óptima (todos los invitados cumplen la condición y además son el máximo número posible de invitados que cumplen la condición).
Además en este algoritmo se toman las decisiones en función de la información que está disponible en cada iteración (es decir, según el subproblema actual). La decisión en una iteración de mi algoritmo será hecha en función de la información actual acerca del listado de socios
Es por todo esto que se explica que el algoritmo propuesto es un algoritmo greedy.
Justificación de optimalidad: En primer lugar supondremos que existe una solución Óptima O. A su vez, si todas los socios conocen a más de cuatro invitados entonces todos estarán invitados y ese será el óptimo O.
Vamos a analizar el caso inicial donde el primer paso del algoritmo no invita a un socio cuyo *minimo* no cumple la comdición para ser invitado, de manera tal que este socio no será invitado y por lo tanto no será parte del óptimo. Por consiguiente, en el algoritmo propuesto se procederá a reducir en uno la cantidad de conocidos de todos aquellos conocidos de este socio en cuestión (que no cumple con la condición para ser invitado y por ende no es parte del óptimo). La reducción se da ya que todos esos conocidos ya no conocerán a este socio dentro de los invitados, puesto que este socio ya no será invitado.
Luego se repetirá el proceso buscando al socio que tenga el *minimo*. Y nuevamente, si este socio encontrado no cumple la condición no será invitado y tampoco será parte del óptimo.
Ahora vamos a proceder a pensar en el caso general donde suponemos que i es el socio que removemos de los invitados por no cumplir la condición, es decir no conoce al menos cuatro socios invitados y no es parte del óptimo. Ahora bien, supongamos que el socio i+1 conocía al socio i que fue removido de los invitados, ahora no conoce a la cantidad suficiente de invitados necesarios y resulta ser el próximo socio que posee la cantidad mínima de conocidos. Entonces el socio i+1 no será invitado y tampoco será parte del óptimo.
Como en el caso general el algoritmo prueba que dado el elemento i no está en el óptimo, el próximo socio i+1 tampoco lo estará (y dado que se cumple para el caso inicial y el general), por el principio de inducción se cumple para todos los casos.

2.3. 2. Explique cómo implementar algorítmicamente esa estrategia. Brinde pseudocódigo y estructuras de datos a utilizar.

La implementación del algoritmo propuesto es la siguiente:
Tendremos en primer lugar una lista de n socios invitados o bien potenciales invitados.
Dicha lista contará con el siguiente formato
Sea s un socio perteneciente a la lista de posibles invitados:
s = [Id, Nombre_socio, listaConocidos, cantidadConocidos, estaInvitado = true]
Nótese que las primeras tres informaciones de la lista de s son las proveídas por el archivo de socios. Luego, calculamos la *cantidadDeConocidos* en base a la longitud de la lista de conocidos de s y finalmente le asignamos inicialmente a todos una invitación activa. Utilizaremos la palabra *true* para indicar que el socio está invitado.
Una vez hecho esto, contamos con la *lista[s,..s,s]* en condiciones para efectuar el algoritmo greedy propuesto.
Comenzamos por calcular cuál socio de la lista tiene la menor cantidad de conocidos. Luego verificamos la condición de un ciclo *while*, si la cantidadConocidos del socio con menor cantidad de conocidos es menor a 4 -Condición mínima por enunciado para que un socio pueda ser invitado-.
Aquí se difurcan dos caminos. Podría suceder que inicialmente no se cumpla esta condición, esto querría decir que ningún socio de la lista cuenta con menos de 4 conocidos, por lo tanto, todos ellos son los invitados a la fiesta.
Por otro lado, podría ocurrir que se verifique tal condición y en tal caso, habría al menos un socio que no puede ser invitado. En ese escenario, se procede a disminuir en uno la cantidad de conocidos de todos aquellos conocidos del socio -que no cuenta con el mínimo de conocidos necesario-. Además se cambia a *false* la información de *s en estaInvitado*, puesto que este socio no puede recibir invitación.

Finalmente, debemos descartar que no existan otros socios que tampoco cumplan la condición, por ello, volvemos a calcular al socio de la lista con menor cantidad de conocidos. Y así volver a verificar la condición del ciclo *while* explicado anteriormente.
Una vez que no se cumpla la condición del ciclo *while* estaremos en condiciones de afirmar que todos los socios de la lista cuyo atributo *esteInvitado* continue en *true*, cumplen la condición para ser invitados.
La estructura de datos que utilizaremos será una lista de listas para almacenar a los socios y su información correspondiente.

A continuación un pseudocódigo del algoritmo propuesto.

Algorithm 1 Pseudocodigo del algoritmo greedy propuesto

```
Sea L lista de socios de tamaño n
Sea socioMin = socio de la lista con menor cantidad de conocidos.
while socioMin.conocidos < 4 do                                          ▷
    for cada conocido de socioConConocidosMin do
        if el conocido tiene invitación activa then
            Ir al conocido dentro de L y disminuir en 1 la cantidad de conocidos invitados.
        end if
    end for
    Desactivar la invitación del socioMin.
    socioMin = Recalcular el socio de L con menor cantidad de conocidos.
end while
```

2.4. 3. Analice complejidad temporal y espacial de su propuesta

La complejidad temporal del algoritmo propuesta es de $O(n^2)$.
Dado que contenemos por un lado un ciclo *While* de complejidad $O(n)$. Y en cuyo interior contemos dos ciclos *For* de complejidad $O(n)$, las demás operaciones necesarias del algoritmo son de complejidad $O(1)$.

- Loop While es $O(n)$
- La búsqueda del mínimo es $O(n)$.
- Ciclo For por cada conocido del socio es $O(n)$.
- Las demás operaciones intermedias dentro del while tienen complejidad $O(1)$

A continuación se detalla el pseudocodigo indicando las complejidades:

Algorithm 2 Pseudocodigo del algoritmo greedy propuesto

```
Sea L lista de socios de tamaño n
Sea socioMin = socio de la lista con menor cantidad de conocidos.
while socioMin.conocidos < 4 do                                  ▷ Complejidad O(n)
    for cada conocido de socioConConocidosMin do                 ▷ Complejidad O(n)
        if el conocido tiene invitación activa then              ▷ Complejidad O(1)
            Ir al conocido dentro de L y disminuir en 1 la cantidad de conocidos invitados. ▷
Complejidad O(1)
        end if
    end for
    Desactivar la invitación del socioMin.                       ▷ Complejidad O(1)
    socioMin = Recalcular el socio de L con menor cantidad de conocidos.▷ Complejidad O(n)
end while
```

Las complejidades de las operaciones interiores al *while* se sumen y resultan en $O(n)$.
La complejidad del *while* resultaba de $O(n)$.
Por lo tanto se obtiene que $O(n)$ * $O(n)$ = $O(n^2)$

Por otro lado, el algoritmo posee complejidad espacial $O(1)$ dado que solo se utiliza la estructura de una lista cuyo contenido son los socios.

2.5. Programe su algoritmo y entregue dos ejemplos para su prueba.

El código del algoritmo junto con los dos ejemplos para su prueba se encuentran en el archivo *club.py* provisto.
Ejecutar: Para ejecutar el programa debe abrir una terminal en la ruta donde se encuentre el la carpeta TP1-TDA y correr el comando: py club.py
No debe pasarse por linea de comando el archivo ni el N.
Dentro del archivo club.py se encontrarán distintos casos de prueba. Listos para ejecutarse.

2.6. 5. ¿Su programa mantiene la complejidad espacial y temporal de su algoritmo? Justifique referenciando a la documentación del lenguaje si es necesario.

En el caso de la complejidad espacial, el algoritmo no mantiene la complejidad puesto que se necesitó tratar con más listas para realizar el parseo necesario del archivo de socios provisto. Te-

nemos una lista de listas que almacena a los socios y su información de manera que la complejidad espacial del programa es de $O(n^2)$

Sin embargo, el programa sí mantiene la complejidad temporal $O(n^2)$ dado que no se altera la resolución propuesta.

2.7. 6. Los organizadores desean que cada invitado pueda conocer nuevas personas

Por lo que nos solicitan que adicionemos una nueva restricción a la invitación: Sólo puede asistir si NO conoce al menos otras 4 personas invitadas. Modifique su propuesta para satisfacer esta nueva solución. ¿Impacta en su complejidad?
La modificación que se le aplicaría al algoritmo para cumplir con la nueva condición: Sólo puede asistir si no conoce a al menos otras cuatro personas invitadas, sería agregar una condición al loop *while*, que ya verifica que se cumpla la condición 1 (leáse condición 1. como la condición original del problema), la nueva condición que podemos escribirla de la siguiente forma.
Contamos ya con la información de cuántos conocidos invitados cuenta cada socio. Por otro lado también podemos llevar a cuenta el total de invitados *N*, considerando que inicialmente son N (información que nos llega por parámetro inicialmente). Por lo tanto podríamos escribir la condición como

n-minimo = Total de invitados que el socio no conoce.

Recordemos que *minimo* hace referencia a la cantidad de conocidos que posee el socio con menos conocidos de la lista, es decir el socio con menos conocidos.

La nueva condición a cumplir nos quedaría de la siguiente manera:

(n-mínimo) >= 4

El algoritmo actual debería modificar su condición de ciclo *while* tal que

While(socioMin.conocidos <4 and (n - socioMin.conocidos) >=4)

Por último, otra modificación a realizar es la de actualizar el valor de n dado que se trata de la cantidad total de invitados.

Algorithm 3 Pseudocodigo del algoritmo greedy modificado

```
Sea L lista de socios de tamaño n
Sea n el tamaño de la lista de socios
Sea socioMin = socio de la lista con menor cantidad de conocidos.
while socioMin.conocidos < 4 and (n − socioMin.conocidos >= 4) do        ▷
    for cada conocido de socioConConocidosMin do
        if el conocido tiene invitación activa then
            Ir al conocido dentro de L y disminuir en 1 la cantidad de conocidos invitados.
        end if
    end for
    Desactivar la invitación del socioMin.
    n = n -1
    socioMin = Recalcular el socio de L con menor cantidad de conocidos.
end while
```

Como se observa, la modificación no impacta en la complejidad del algoritmo. Pues las operaciones que se agregan tienen una complejidad $O(1)$.