FACULTAD
DE INGENIERIA
Universidad de Buenos Aires

# Teoría de Algoritmos (95.06)

## Trabajo Práctico 1

Sebastián Mosquera 106744

2do Cuatrimestre 2022

# Indice

# Enunciado

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

Nos solicitan seleccionar el mayor número posible de invitados.

# Estrategia Greedy

## Por qué elegir un algoritmo Greedy

En este problema lo que se busca es maximizar la cantidad de invitados (o minimizar la cantidad de socios no invitables)

Para demostrar que es un problema optimizable mediante un algoritmo Greedy, se demuestra que el mismo cumple con 2 características necesarias: subestructura óptima y elección Greedy.

Veremos que paso a paso se puede descomponer el problema más grande en subproblemas más pequeños, y la solución global contiene a las soluciones locales. A través de una elección heurística el algoritmo ahondará en determinados subproblemas para avanzar hacia una solución óptima global.

## Estrategia

Si bien el objetivo del problema es maximizar la cantidad de invitados, el requisito que se impone en cada socio para que sea “invitable” es conocer otros 4 invitados.
Inicialmente no se puede saber quiénes de los conocidos de cada socio serán finalmente invitados por lo que la condición para ser invitable no se puede determinar directamente. Sin embargo, lo que sí se puede determinar directamente es quien de los socios no puede ser invitado, quien es “no inivitable”.

Definición:

> Se dice que un socio es “no invitable” cuando dentro de sus conocidos, menos de 4 de ellos son invitables.

Por lo tanto, la idea central del algoritmo es ir paso a paso “desinvitando” socios.
Desinvitar un socio significa que este pasa a ser “no invitable”, lo que podría alterar la condición de todos sus conocidos, es necesario entonces que estos vuelvan a ser chequeados.

### Elección Greedy

En cada paso se elimina al primer socio encontrado que tiene menos de 4 conocidos ¨**invitables**¨ y se vuelven a tener en cuenta a sus conocidos

### Subestructura óptima

Cuando **todos** los socios "disponibles" a chequear cuentan con **al menos** 4 conocidos, entonces esto implica que todos son invitables, ya que no se puede eliminar a ninguno y la lista permanece sin cambios.

Por ende, estos son los socios finalmente invitados.

### Optimalidad

- Para demostrar que el algoritmo es óptimo se puede ver que luego de iterar por la lista de socios, además de la lista de conocidos según corresponda, paso a paso se encarga de desinvitar socios que no cumpla la condición fundamental.
- Al finalizar la lista de socios quedará en un estado constante y se retornará una lista de gente que NO tiene menos de 4 invitables, es decir, tiene 4 o más conocidos invitables. Por lo tanto, se cumple el requisito de que todos los invitados conozcan otros 4 invitados.

- Suponiendo O un set óptimo de socios invitados y A un set de socios invitados que se logra mediante el algoritmo, se busca demostrar la igualdad de cardinalidades:

$$|A| = |O|$$

- Como O es óptimo, ninguno de su socios es "no invitable"
- Por como avanza el algoritmo siempre que haya un socio "no invitable", este será desinvitado y sus conocidos vuelven a ser chequeados. Si no hay más **re chequeos** implica que todos los socios "disponibles" tienen al menos 4 conocidos "invitables", y a su vez, estos conocidos invitables deben pertenecer a al conjunto de socios "disponibles" porque ya no habrá más **re chequeos**.
Se concluye que la lista A permanece constante. Y que todos los socios que contiene son "invitables".

## Algoritmo Greedy

### Deconstrucción del problema/enunciado y definiciones

Llamaremos:
- S: conjunto de socios
- C: conjunto de listas de conocidos de cada socio
- Invitable: inicialmente todos los socios son invitables (hasta que se demuestre lo contrario)
- No invitable: el socio tiene menos de 3 conocidos invitables

Hipótesis:
- Cada socio está representado por: número de socio, nombre, y lista de conocidos
- Conocimiento mutuo: si A conoce a B entonces B conoce a A

## Algoritmo

Inicio
- El algoritmo comienza con una pila que contiene a todos los socios.
- Todos los socios son invitables.

Desarrollo
- El algoritmo avanza en la pila (desapilando un socio a la vez) y chequea para este la condición fundamental: si tiene menos de 4 conocidos invitables.
- Para este chequeo se emplea la función: cantidad_conocidos_invitables(socio) la cual recibe un socio, itera por todos los conocidos del socio, para cada uno pregunta si este es invitable, y finalmente retorna la cantidad de conocidos que en ese momento son invitables.

- En el caso de que el chequeo anterior sea **falso**: se avanza con el siguiente socio en la pila.
- En el caso de que el chequeo anterior sea **verdadero**: el socio pasa a ser "no invitable" y se agregan todos sus conocidos a la pila.
- Este último paso es fundamental ya que la actualización en la condición de un socio podría alterar la condición de sus conocidos. Esto parte de la hipótesis de conocimiento mutuo.

Finalización
- El algoritmo finaliza cuando la pila queda vacía, es decir, cuando todos los socios "disponibles" tienen al menos 4 socios, por lo que no se puede desinvitar a ninguno, y la lista de socios invitables permanece inalterable.

## Pseudocodigo

```
Socios = [1, 2, … , n]
pila = crear_pila()
desde i = 0 hasta n
        pila.apilar( socios[i] )

mientras pila != vacia
        socio = pila.desapilar()
        sí conocidos invitables de socio < k
                desinvitar socio
                apilar conocidos del socio

desde i=0 hasta n
        sí socios[i] es_invitable()
                imprimir(socio[i].nombre)
```

Para simplificar el pseudocodigo, se muestra una lista de socios representados cada uno por un número entero. En la implementación real, cada socio es un TDA que contiene su número de socio, su lista de conocidos y su nombre, además de un parámetro booleano que indica si es invitable.

# Complejidad Temporal y espacial

1) Se recorre la lista de socios y se apila uno por uno. Dependiendo de la estructura utilizada para apilar será la complejidad, en nuestro caso se utiliza un array por lo que apilar y desapilar es O(1). Por lo tanto, este recorrido es O(n).

2) Se recorre la pila y en cada iteración se realiza un chequeo al socio. En el caso (peor caso) de que el chequeo de falso: se desinvita al socio poniendo en falso su parámetro de invitado O(1) y se recorren todos sus conocidos y se apilan todos aquellos que no sean no invitables, si tiene m cantidad de conocidos y solo l de ellos son invitables la operación es O(m + l).
   Ahora bien, para que este socio pase a ser desinvitado, debe tener menos de k conocidos invitables, para este problema k=4, por lo tanto a lo sumo se agregaran k-1 socios a la pila, siendo l = 3, 2, 1 o 0.
   Finalmente el chequeo de conocidos sumado a la incorporación en la pila es O(m), siendo m la cantidad de conocidos de un determinado socio.

   El recorrido de la pila, con sus eventuales incorporaciones dependerá mucho del ordenamiento de los socios por su cantidad de conocidos y también del orden de estos últimos.

   Por ejemplo:

   En el caso de que todos los socios tengan exactamente 4 conocidos, la pila se recorrerá una sola vez y en cada paso se recorrerán los 4 conocidos de cada socio.
   O(n*4)

   En el caso de que todos los socios se conozcan entre sí: O(n*n)

   Y en el caso de que el ordenamiento sea desfavorable, es decir, en orden decreciente de conocidos, y que en cada recorrido de la pila se desinvite solo al último socio:

| Socio | Conocidos | Cantidad mi |
|---|---|---|
| 1 | n, n-1, …, 3, 2 | n |
| 2 | n, n-1, …, 3, 1 | n |
| … | | |
| n-1 | n-1, n-2, n-k, n-(k-1) | k |
| n | n-1, n-2, n-(k-1) | k-1 |

   - se recorre n socios, para cada uno se chequean sus m conocidos

$$\sum_{1}^{n} m_i$$

- se desinvita socio n y se apilan k-1.
- se recorren n-1 socios, para cada uno se chequean sus m conocidos

$$(k-1)+\sum_{1}^{n-1} m_i$$

- se repite este procedimiento hasta llegar al socio numero k, a partir de este hasta el socio 1 todos quedan con menos de k conocidos, porque se han eliminado ya el resto de los n-k socios.

- como estamos analizando el caso de que en cada iteración se desinvite un socio, esto implica que: $m_i < n \ \ para\ i > k \ \ y\ k = cte$.

Finalmente se ve que el ciclo es: $O(n^2)$

3) Se recorre la lista de socios, y se imprime aquellos que cumplan la condición de invitados: O(n)

Finalmente la **complejidad temporal** del algoritmo es: $O(k * n^2)$

La complejidad espacial se analiza similarmente, y se llega a que el peor caso posible es que esta esté ordenada por cantidad de socios, pero ahora de forma ascendente y que en cada paso se desinvite un solo socio. Por lo tanto, en la pila se almacenan inicialmente n socios, y paso a paso, se irán agregando k conocidos por cada socio desinvitado:

Finalmente la **complejidad espacial** del algoritmo es: $O(k * n)$

# Programa y Ejemplos

Se realizó una implantación del algoritmo en Python. Para ello se agregaron las estructuras de datos necesarias: Un TDA Pila y un TDA Socio. El algoritmo principal y las funciones que este utiliza se encuentran el archivo “main.py”. El programa esta diseñado para leer un archivo de texto con el mismo formato del enunciado y este imprime por consola los nombres de los socios finalmente invitados

## Instrucciones de compilación

Es necesario utilizar Python versión 3

./python main.py n archivo.txt

Con el ejemplo del enunciado: ./python main.py 9 socios.txt

### Ejemplos

Se incluyen 2 ejemplos que se utilizaron de testeo:

- socios_2.txt
- socios_3.txt

para los cuales la salida es:

```
./python main.py 9 socios_2.txt

1: Alan
2: Barbara
3: Carlos
4: Daniela
5: Esteban
```

```
./python main.py 12 socios_3.txt

1: Alan
2: Barbara
3: Carlos
4: Daniela
5: Esteban
6: Francisca
7: Gabriel
8: Herminia
9: Ivan
10: Julian
11: Karen
12: Lucio
```

## Complejidad Temporal y espacial del programa

Debido a que en el pseudocodigo se tuvo en cuenta las estructuras y los costes de la funciones a utilizar, no se presentan diferencias en la complejidad del programa con respecto al análisis ya realizado, tanto temporal como espacial.

El TDA Pila está implementado con un array, cuyo coste de apilamiento y desapilamiento es O(1), ya que utiliza los métodos append() y pop().
El TDA Socio contiene el parámetro “invitable”, por lo que tanto el chequeo como la desinvitacion son de tiempo constante. Además, se implementa un array para almacenar sus conocidos.

## Nueva Restricción/Propuesta

## Referencias

Documentación de Python: https://docs.python.org/3/