Estructuras de datos
Clase teórica 5

UNIVERSIDAD NACIONAL DE COLOMBIA SEDE MEDELLÍN

Contenido

- El juego de Josephus Flavius

Material elaborado por: Julián Moreno

Facultad de Minas, Departamento de Ciencias de la Computación y la Decisión

Evaluaciones del módulo 1

- Examen parcial (24%): Martes 16 de Septiembre, 4:00 pm, auditorio Peter Santamaría. ¿Qué hay que saber?
  - Arquitectura y funcionamiento de las estructuras vistas
  - Eficiencia de las operaciones básicas y el por qué de las mismas
  - Uso en java de dichas estructuras
  - No se rebajará por errores leves de sintaxis
- Taller (6%): Martes 16 de Septiembre, 3:55 pm
- Trabajo práctico (4%):
  - Entrega: Miércoles 17 de Septiembre, 11:55 pm al correo jmoreno1@unal.edu.co
  - Sustentación: Jueves 18 de Septiembre, 2:00 a 6:00 pm, oficina M8A 311. Se lleva a cabo por UN(A) integrante del grupo seleccionado aleatoriamente. Los programas deben ser jugables y entendibles.

El juego de Josephus Flavius

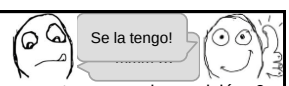
Josephus Flavius fue un famoso historiador judío del siglo I. Durante la guerra judío-romana fue rodeado por el ejército romano y quedó atrapado junto a 40 soldados en una cueva. La leyenda dice que, prefiriendo el suicido a la captura, los soldados decidieron formar un círculo. Luego se numeraron del 1 al 41 (para incluir a Josephus). Comenzando con el soldado 1, contaban 3 soldados a la derecha y ese debía ser asesinado por el compañero de la izquierda. El proceso se repetía a partir de allí hasta que no quedara sino uno en pie (al último no había quien lo matara). Josephus no quería morir y por eso hizo un rápido ejercicio mental, encontró el "puesto seguro" y escapó de la muerte (los romanos al ver la masacre y que era el único sobreviviente lo dejaron con vida).

El juego de Josephus Flavius

Basado en hechos reales …

En su entrevista para una de las más importantes empresas de software del mundo a un estudiante de la nacho le pidieron: "Diseñe un algoritmo para determinar el "puesto seguro" en el juego de Josephus considerando *n* jugadores y un conteo de *k* para eliminar un jugador (*n* y *k* son enteros positivos)".

Solución de nuestro protagonista:



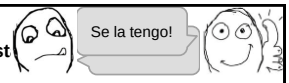
- Creamos un arreglo de *n* números enteros, en la posición 0 ponemos el "1", en la 1 el "2" y así sucesivamente hasta "n".
- Hacemos *posición* = 0
- Hacemos *cantidadJugadores* = *n* y entramos a un *while*(*cantidadJugadores* > 1)
- Hacemos *contador* = 0
- En cada iteración hacemos un *while*(*contador* < k)
- Si el arreglo en *posición+1* es diferente de -1 hacemos *contador*++, ponemos en esa posición -1 y hacemos *cantidadJugadores*--
- Hacemos posición++ (si llegamos a *n* "reseteamos" *posición*)
- Al terminar el while más externo buscamos la única posición que no valga -1 y esa es la solución

Objeción del entrevistador: ¿cuál es la eficiencia del algoritmo en términos de *n* y k?, ¿qué pasaría si son muy grandes?

Análisis de nuestro protagonista:

- Como no se puede simplemente saltar en el arreglo *k* posiciones hacia la derecha pues no se sabe cuantos en el medio han muerto, "matar uno" no es *O(1)*
- Tampoco es *O(k)* por la misma razón
- En el peor de los casos es *O(nk)* y eso es solo para matar uno
- Entonces matar *n-1* tomaría $O(n^2k)$
- Si *n* es por ejemplo un millón y *k* es quinientos mil, eso daría … 500000000000000000 (5E+18)

Solución #2 de nuestro protagonist

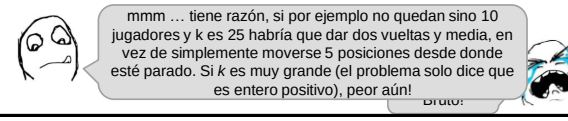


- Creamos una lista enlazada de *n* números enteros, en la posición 0 ponemos el "1", en la 1 el "2" y así sucesivamente hasta "n".
- Hacemos *posición* = 0
- Hacemos *cantidadJugadores* = *n* y entramos a un *while*(*cantidadJugadores* > 1)
- Hacemos *contador* = 0
- En cada iteración hacemos un *while*(*contador* < k)
- Hacemos contador++, nos movemos al elemento siguiente y si llegamos al último lo "reseteamos"
- Borramos el elemento de la lista
- Al terminar el while más externo buscamos el único elemento de la lista que quede

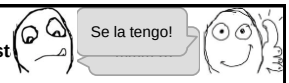
Análisis de nuestro protagonista:

- Como ya se trata de una lista donde solo están jugadores vivos, matar a uno de ellos tomaría *O(k)*
- Entonces matar *n-1* tomaría *O(nk)* … Si, otra vez, *n* es por ejemplo un millón y *k* es quinientos mil, eso daría … 500000000000 (5E+12)

Objeción del entrevistador: ¿No se puede hacer mejor considerando que si en algún momento *k* es mayor que la cantidad de jugadores con vida estaría dando vueltas "innecesarias"?



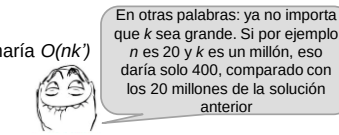
Solución #3 de nuestro protagonist



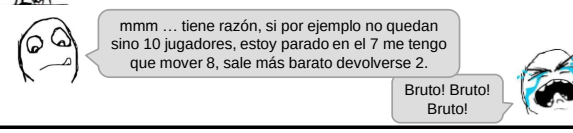
- Creamos una lista enlazada de *n* números enteros, en la posición 0 ponemos el "1", en la 1 el "2" y así sucesivamente hasta "n".
- Hacemos *posición* = 0
- Hacemos *cantidadJugadores* = *n* y entramos a un *while*(*cantidadJugadores* > 1)
- Hacemos *contador* = 0
- En cada iteración hacemos un *while*(*contador* < (*k % cantidadJugadores*))
- Hacemos contador++, nos movemos al elemento siguiente y si llegamos al último lo "reseteamos"
- Borramos el elemento de la lista
- Al terminar el while más externo buscamos el único elemento de la lista que quede

Análisis de nuestro protagonista:

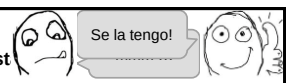
- Usando una lista y moviéndose hacia adelante usando el modulo garantizo que, si en algún momento *k* es mayor a la cantidad de jugadores, habría que moverse, a lo sumo, la cantidad de jugadores menos 1
- Y entonces matar *n-1* tomaría *O(nk')* siendo *k'* = *MIN(n,k)*



Objeción del entrevistador: pero, si lo que se tiene que mover hacia adelante es mayor que la mitad de los jugadores con vida, ¿no es más eficiente ir hacia atrás en vez de hacia adelante?



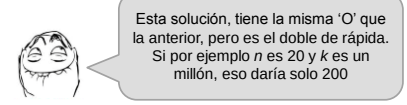
Solución #4 de nuestro protagonist



- Creamos una lista enlazada de *n* números enteros, en la posición 0 ponemos el "1", en la 1 el "2" y así sucesivamente hasta "n".
- Hacemos *posición* = 0
- Hacemos *cantidadJugadores* = *n* y entramos a un *while*(*cantidadJugadores* > 1)
- Si *k % cantidadJugadores* es menor que *cantidadJugadores/2 nos movemos esas posiciones a la derecha,* si llegamos al último lo "reseteamos". En caso contrario nos movemos *cantidadJugadores – k % cantidadJugadores* hacia la izquierda, si llegamos al primero lo "reseteamos"
- Borramos el elemento de la lista
- Al terminar el while más externo buscamos el único elemento de la lista que quede

Análisis de nuestro protagonista:

- Moviéndose hacia adelante o hacia atrás según convenga garantizo que matar a uno de ellos tomará, como máximo, la mitad de los jugadores con vida
- Y entonces matar *n-1* tomaría *O(nk")* siendo *k"*= *MIN(n/2,k)*



Entrevistador: muchas gracias por venir … lo estaremos llamando