ALGORITMOS: AYUDANTÍA 3

Ayudante: Yerko Ortiz

Objetivo de la ayudantía: Reforzar entendimiento en listas, colas y pilas.

Diseño, análisis e implementación

Para los siguientes enunciados, diseñe e implemente una solución, analice y describa la complejidad de su algoritmo en términos de O(f(n)).

Revertir una lista enlazada

Dada una lista enlazada, diseñe e implemente un algoritmo que invierta la dirección de los punteros y deje de cabeza al que antes era el último nodo en la lista.
Entrada

- Una linea con un entero n que representa la cantidad de nodos en la lista.
- La linea siguiente contiene n enteros x_i que corresponden a los valores en la lista, donde x_0 es el valor de la cabeza, x_1 el valor del siguiente, x_2 del siguiente y así sucesivamente.

Dominio

- $1 \leq n \leq 10^6$
- $1 \leq x_i \leq 10^6$

Salida

- Imprimir el recorrido lista invertida

Caso de prueba
Entrada
5
1
2
3
4
5
Salida
5 4 3 2 1

Crear una cola con dos pilas

Implemente el TDA cola(FIFO) con las operaciones dequeue(), front(), enqueue(int x), con la restricción de que debe usar dos pilas(LIFO) para la implementación.
Para testear su implementación se realizarán q consultas, donde cada consulta puede ser de cierto tipo:

- 1: Encolar el entero x en la cola.
- 2: Imprimir el primer elemento en la cola.
- 3: Desencolar al primer elemento en la cola.

Entrada

- La primera linea contiene un entero q, que representa el número de consultas a realizar.
- Cada linea i subsiguiente representa una de las q consultas, representadas por un entero que denota el tipo de consulta, para el caso de encolar, la linea además de empezar con un 1, seguido de un espacio contendrá un entero x que corresponde al elemento a encolar.

Dominio

- $1 \leq q \leq 10^5$
- $1 \leq tipo \leq 3$
- $0 \leq x \leq 10^9$

Salida

- Para toda consulta del tipo 3, imprimir el valor del primer elemento en la cola.

Caso de prueba
Entrada:

10
1 42
2
1 14
3
1 28
3
1 60
1 78
2
2
Salida 14
14

"Simplicity is a great virtue but it requires hard work to achieve it and education to appreciate it. And to make matters worse: complexity sells better."

Edsger Dijkstra

Paréntesis balanceado

Sean los siguientes caracteres, {, }, [,], (,).
Dada una expresión formada con dichos caracteres se dice que está balanceada si y solo si, el símbolo de la parte izquierda de la expresión (posiciones 0 hasta (n / 2) - 1) es el contrario al que esté en su posición simétrica (de n/2 hasta n - 1) siempre y cuando cierren la expresión.
Entrada:

- Un String s, que contiene la expresión.

Dominio

- $1 \leq |s| \leq 10^3$
- $s_i \in \{(, \{, [,], \},)\}$

Salida

- Imprimir YES si la expresión está balanceada, imprimir NO para el caso complementario.

Casos de prueba
Entrada 1

({[]})
Salida 1
YES

Entrada2
)}{(
Salida 2
NO

Entrada 3
([{}])
Salida 3
NO

El pergamino sagrado

Un maestro ninja pasó sus conocimientos del pergamino sagrado a sus discípulos, estos a su vez cuando se convirtieron en maestros, pasaron sus conocimientos a sus discípulos, continuando este ciclo por la posteridad hasta estos días; los ninjas que son discípulos del mismo maestro se les considera hermanos. Es sabido que todos los maestros enseñan de forma distinta y que a su vez todos los alumnos aprenden de distinta forma, es por esto que mientras las generaciones aumentan, el conocimiento del pergamino va cambiando hasta llegar al punto en que la enseñanza de un maestro, es un conocimiento absolutamente distinto al del pergamino original.
A usted como ninja de la programación, se le delega la tarea de crear un árbol genealógico que abarque a todos los ninjas que han obtenido parte del conocimiento del pergamino, donde para realizar esta tarea debe diseñar e implementar el TDA(tipo de dato abstracto) ArbolNinja tomando en cuenta las siguientes indicaciones:

Atributos de la clase Ninja:

- Ninja primerDiscipulo;
- Ninja nextHermano;
- int id;
- String[] nombre;

El ArbolNinja debe tener los siguientes métodos:

- **void crearArbolNinja(String[] nombreMaestro)**: inicializa el TDA con un ninja en su interior (el maestro inicial) con id = 0 y el nombre que usted estime conveniente.
- **void insertarDiscipulo(String[] nombreDiscipulo, int idMaestro)**: busca el id del maestro en el TDA y luego inserta al ninja al final de la lista de discípulos (asignándole un id).
- **int contarGeneracion(int idNinja)**: busca al ninja con idNinja en el TDA para luego retornar el número de generaciones de enseñanza que han pasado hasta ese ninja.
- **void imprimirArbol()**: que imprime los nombres de todos los ninjas por generación, imprime el nombre del primer maestro, luego sus discípulos, luego los discípulos de esos discípulos y así sucesivamente.
- **Método bonus(usted decide si hacerlo):**
  int minimoAncestroComun(int idNinja1, int idNinja2): dado el id de dos ninjas, este método los busca y calcula el ancestro mínimo(maestro más cercano) que tienen en común.

Hint: para la asignación de id use un contador, así evita que existan dos ninjas con el mismo id en el TDA.