Universidad Nacional Autónoma de México

Facultad de Ingeniería
División de Ciencias Básicas

Estructura de Datos y

Algoritmos 1

Profesor(a): Leonardo Ledesma

Domínguez

Semestre 2024-2

Serie 2. Stack & Queue.

Grupo: _10_
Equipo: _16_

Integrantes:

-Velez Romero Oscar Daniel N.L.41

-Mora Celis Ángel Gabriel N.L. 23

Fecha de entrega: _16_ de Marzo de 2024.

Diseño y Análisis de Algoritmos:

1. Llene la siguiente tabla según el tipo de estructura de datos lineal.

Tipo de estructura lineal	Tipo de Memoria	Número y cuáles apuntadores	Casos analizados	Complejidad Algorítmica en O	Nombre Operaciones Básicas	Paradigma
Stack (Pila)	Contigua	1. "T"	1. La pila está llena. 2. La pila está vacía. 3. Caso normal.	Constante	PUSH y POP	LIFO – Last In, First Out. ó, el último en entrar es el primero en salir.
Stack(Pila)	Ligada	1. "X"	1. La pila está llena. 2. La pila está vacía. 3. Caso normal.	Constante	PUSH y POP	LIFO – Last In, First Out. ó, el último en entrar es el primero en salir.
Simple Qeeue (Cola)	Contigua	2. "U y P"	1. La cola está llena. 2. La cola está vacía. 3. Caso normal. **Agregar por U y retirar por P**	Constante	ENCOLAR (PUSH) y DESENCOLAR (POP)	FIFO – First In, First Out ó, el primero en entrar es el primero en salir.
CircularQeeue (Cola Circular)	Ligada	2. " U y P"	1. La cola circular está llena. 2. La cola circular está vacía. 3. Caso normal. **Agregar por U y retirar por P**	Constante	ENCOLAR (PUSH) y DESENCOLAR (POP)	FIFO – First In, First Out ó, el primero en entrar es el primero en salir.
Deque(Cola Doble)	Contigua	2. "U y P"	1. La cola doble está llena. 2. La cola doble está vacía. 3. Caso normal. **Agregar por U y retirar por U, agregar por P y retirar por P**	Constante	ENCOLAR (PUSH) por ambos lados y DESENCOLAR (POP) por ambos lados	FIFO (First In, First Out) y LIFO (Last In, First Out).
Deque(Cola Doble	Ligada	3. "U, P y X"	1. La cola doble está llena. 2. La cola doble está vacía. 3. Caso normal. **Agregar por P y retirar por P**	Constante	ENCOLAR (PUSH) por ambos lados y DESENCOLAR (POP) por ambos lados	FIFO (First In, First Out) y LIFO (Last In, First Out).

2. Algorithm T1. (Sedgewick, Exercise 4.31). A letter means put and an asterisk means get in the following sequence. Give the sequence of values returned by the get operation when this sequence of operations is performed on an initially empty FIFO queue.

```
Función Operaciones_Cola_Palabra(C):
    cola <- Cola()
    resultado <- ""
    Para cada operación en C:
        Si operación es 'E':
            cola.encolar(operación)
        Sino si operación es '*':
            Sí la cola no está vacía:
                elemento <- cola.desencolar()
                resultado <- resultado + elemento
            Sino:
                resultado <- resultado + "No hay elementos en la cola"
      Fin
    Mostrar(resultado)
Fin
```

3. Algoritmo T2. Proponga un algoritmo para usar una ***Stack S*** para determinar si una ***cadena C*** es palíndromo o no, entonces se solicita definir la función booleana ***Palindromo(C)***.

```
Función Palindromo (C) :
        pila -> Stack()
        //Para cada carácter de C:
              pila. Push (carácter)
         //Para cadena invertida:
                mientras pila >= 1:
                    carácter -> pila.pop ()
                          cadena invertida -> cadena invertida + carácter
                              Si C= cadena invertida entonces:
                                    La pila es un palíndromo
                          En caso contrario:
                                    La pila no es un palíndromo
                Fin
   Fin
```

4. Algoritmo T3. Diseñe un algoritmo usando una ***Queue Q*** para simular el comportamiento de un banco que solo tiene un cajero de servicio. Las personas van siendo atendidas conforme van llegando, pero el 25% de las personas tendrán la documentación insuficiente para poder realizar sus trámites, por lo que se regresan a la final de la cola para ser atendidas. Seleccione la cola necesaria para resolver el problema y justifique su respuesta.

JUSTIFICACIÓN:

Para el orden de llegada, como la cola mantiene el orden de llegada de las personas al banco, nos ayudará a resolver lo que pedido que es, "primero en llegar, primero en ser atendido" (FIFO).

Será más fácil regresar personas a la cola cuando le hacen falta documentos necesarios para su

necesidad, simplemente la devolveremos al final de la cola. La operación de encolar (enqueue) es adecuada en una cola.
Para hacer la operaciones nos apoyaremos tanto la operación de encolar como la de desencolar (enqueue y dequeue) tienen complejidad temporal constante en una cola, lo que hace que sea eficiente para simular el proceso de atención en el banco.

ALGORITMO:

Función Simular_Banco():

Q <- Queue()

Para cada persona en el intervalo de tiempo:

Si persona tiene documentación suficiente:

Q.encolar(persona)

Sino:

Q.encolar(persona) # La persona se agrega a la cola

Q.encolar(persona) # La persona con documentación insuficiente se agrega nuevamente a la cola

Mientras haya personas en la cola Q:

persona <- Q.desencolar()

Si persona tiene documentación suficiente:

Atender(persona)

Sino:

Q.encolar(persona) # La persona se agrega nuevamente a la cola para intentar nuevamente después

Mostrar("El banco ha terminado de atender a todos los clientes.")

Fin

5. Algoritmo T4. Proponga un algoritmo que usando dos ***Stack S*1 *y S*2** se cree una ***Queue Q***. Compruebe su funcionamiento usando ejemplos.

```
Función enqueue():
        push(S1, dato)
Fin función
Función Dequeue():
    desde x = 0 hasta x < topeS1; x ++:
         dato = peek (S1)
          push(S2, dato)
```

```
    Fin ciclo
    pop(S2)
   desde x = 0 hasta x < topeS2; x++
        dato = peek (S2)
        push (S1, dato)
   Fin ciclo
Fin
```

6. Algoritmo T5. Un historiador judío que vivió en el siglo I llamado Flavio Josefo relató el asedio de Yodfat, donde él y sus 40 soldados quedaron atrapados en una cueva por soldados romanos. Al elegir el suicidio en lugar de la captura, optaron por un método en serie para suicidarse en el que cada persona que muriera sería asesinada por la siguiente. Josefo afirma que por suerte o posiblemente por la mano de Dios, él y otro hombre permanecieron hasta el final y se rindieron a los romanos en lugar de suicidarse.

   El mecanismo preciso para elegir el orden de las ejecuciones no se describió completamente en el relato de Josefo; sin embargo, en 1612 Claude Gaspar Bachet (el escritor de libros sobre acertijos y trucos matemáticos que formaron la base de casi todos los libros posteriores sobre recreaciones matemáticas) sugirió que los hombres se dispusieron en círculo y luego comenzaron a contar de tres en tres a partir de un hombre seleccionado al azar.

   Escriba un algoritmo llamado ***Josefo*** que solicite al usuario un número de personas en un círculo y un valor n, donde cada enésima persona será asesinada, y luego busca (usando una cola) y muestra las posiciones de esas personas asesinadas, en el orden en que murieron, de una manera similar a la que se muestra en el ejemplo siguiente.

   Sugerencias: observe cuán similar es moverse a través de un círculo a mover a alguien desde el principio de una línea hasta su final. Por supuesto, si una persona en el círculo muere, es más como eliminarla sin volver a agregarla.

Si no queda claro este ejercicio resuelve el ejercicio 7 y luego regresa al 6.

```
Función Josephus(número_de_personas, n):
    cola <- Cola()
    Para i desde 1 hasta número_de_personas:
        cola.encolar(i)
    personas_asesinadas <- Lista()
    contador <- 1
    Mientras la cola no esté vacía:
        persona_actual <- cola.desencolar()
        Si contador == n:
            personas_asesinadas.agregar(persona_actual)
            contador <- 1
        Sino:
            cola.encolar(persona_actual)
            contador <- contador + 1
    Devolver personas_asesinadas
Fin
```

7. Ve el siguiente video en: https://www.youtube.com/watch?v=pkq_6DXycZg

Explique con sus propias palabras la función $S(n)$ y calcule el superviviente de 89 y 12, es decir $S(89)$ y $S(12)$.

S(n) es una función que se encarga de transformar *n* a un número impar, esto debido a que si el valor de n lo transformas a binario, cualquier valor de *n* empezará con uno, y al pasar ese uno al final (en binario) representa que el dato o valor es impar.

Solución para el superviviente 12:

Si *n* no es potencia de dos. Necesariamente n = 2^m + k con m > 0 y 0 < k < 2^m.
(Si k = 0, entonces n = 2^m y si k = 2^m, n = 2^m + 2^m = (2*2)^m = 2^m+1).

Entonces:
n = 12
n = 2^3 + 4 (NOTA: para n = 2^m + k, el individuo que se salvará se encuentra en 2k + 1)
S(n) = 2k + 1 = 2(4) +1 = **9**

Por lo tanto, la posición en la que se debe encontrar el superviviente es 9, siendo S(12) = 9

Solución para el superviviente 89:

Si *n* no es potencia de dos. Necesariamente n = 2^m + k con m > 0 y 0 < k < 2^m.
(Si k = 0, entonces n = 2^m y si k = 2^m, n = 2^m + 2^m = (2*2)^m = 2^m+1).
Entonces:
n = 89
n = 2^6 + 25 (NOTA: para n = 2^m + k, el individuo que se salvará se encuentra en 2k + 1)
S(n) = 2k + 1 = 2(25) +1 = **51**

Por lo tanto, la posición en la que se debe encontrar el superviviente es 51, siendo S(89) = 51

Programación:

1. Tome el programa de Python “from scratch” y programe:

   a. Cola Circular

```
class NodoColaCircular:
    """Clase nodo para una cola circular"""
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None

class ColaCircular:
    """Clase Cola Circular"""
    def __init__(self, capacidad):
        self.capacidad = capacidad
        self.frente = None
        self.final = None
        self.tamanio = 0

    def es_vacia(self):
        return self.tamanio == 0

    def esta_llena(self):
        return self.tamanio == self.capacidad

    def encolar(self, dato):
        """Encola un elemento en la cola circular"""
```

```
        nuevo_nodo = NodoColaCircular(dato)
        if self.esta_llena():
            print("La cola está llena.")
            return
        if self.es_vacia():
            self.frente = nuevo_nodo
        else:
            self.final.siguiente = nuevo_nodo
        self.final = nuevo_nodo
        self.final.siguiente = self.frente
        self.tamanio += 1

    def desencolar(self):
        """Desencola un elemento de la cola circular"""
        if self.es_vacia():
            print("La cola está vacía.")
            return None
        dato = self.frente.dato
        if self.frente == self.final:
            self.frente = None
            self.final = None
        else:
            self.frente = self.frente.siguiente
            self.final.siguiente = self.frente
        self.tamanio -= 1
        return dato

    def __str__(self):
        """Muestra el contenido de la cola circular"""
        if self.es_vacia():
            return "La cola está vacía."
        cola_str = "Frente -> "
        nodo_actual = self.frente
        while nodo_actual:
            cola_str += str(nodo_actual.dato) + " -> "
            nodo_actual = nodo_actual.siguiente
            if nodo_actual == self.frente:
                break
        cola_str += "Final"
        return cola_str
```

b. Cola Doble

```
class NodoColaDoble:
    """Clase nodo para una cola doble"""
    def __init__(self, dato):
        self.dato = dato
        self.siguiente = None
        self.anterior = None
```

```
class ColaDoble:
    """Clase Cola Doble"""
    def __init__(self):
        self.primero = None
        self.ultimo = None
        self.tamanio = 0

    def es_vacia(self):
        return self.tamanio == 0

    def encolar_inicio(self, dato):
        """Encola un elemento al inicio de la cola doble"""
        nuevo_nodo = NodoColaDoble(dato)
        if self.es_vacia():
            self.primero = nuevo_nodo
            self.ultimo = nuevo_nodo
        else:
            nuevo_nodo.siguiente = self.primero
            self.primero.anterior = nuevo_nodo
            self.primero = nuevo_nodo
        self.tamanio += 1

    def encolar_final(self, dato):
        """Encola un elemento al final de la cola doble"""
        nuevo_nodo = NodoColaDoble(dato)
        if self.es_vacia():
            self.primero = nuevo_nodo
            self.ultimo = nuevo_nodo
        else:
            self.ultimo.siguiente = nuevo_nodo
            nuevo_nodo.anterior = self.ultimo
            self.ultimo = nuevo_nodo
        self.tamanio += 1

    def desencolar_inicio(self):
        """Desencola un elemento del inicio de la cola doble"""
        if self.es_vacia():
            print("La cola está vacía.")
            return None
        dato = self.primero.dato
        if self.primero == self.ultimo:
            self.primero = None
            self.ultimo = None
        else:
            self.primero = self.primero.siguiente
            self.primero.anterior = None
```

```
        self.tamanio -= 1
        return dato

    def desencolar_final(self):
        """Desencola un elemento del final de la cola doble"""
        if self.es_vacia():
            print("La cola está vacía.")
            return None
        dato = self.ultimo.dato
        if self.primero == self.ultimo:
            self.primero = None
            self.ultimo = None
        else:
            self.ultimo = self.ultimo.anterior
            self.ultimo.siguiente = None
        self.tamanio -= 1
        return dato

    def __str__(self):
        """Muestra el contenido de la cola doble"""
        if self.es_vacia():
            return "La cola está vacía."
        cola_str = "Primero -> "
        nodo_actual = self.primero
        while nodo_actual:
            cola_str += str(nodo_actual.dato) + " <-> "
            nodo_actual = nodo_actual.siguiente
        cola_str += "Último"
        return cola_str
```

2. Implemente el algoritmo número 7 donde se reciba en línea de comandos el parámetro *n*.

y se regrese el valor *S*(*n*), use memoria dinámica y apuntadores.

```
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#define p printf
#define s scanf
struct Cola {
   int dato;
   struct Cola* siguiente;
};

typedef struct Cola Cola;

Cola* crearCola(int valor) {
   Cola* nuevaCola = (Cola*)malloc(sizeof(Cola));
   nuevaCola->dato = valor;
   nuevaCola->siguiente = NULL; //el apuntador se dirige a NULL
   return nuevaCola;
}
//Para liberar la memoria del dato que ocupa cola
void eliminarCola(Cola* cola) {
   free(cola);
}
//Para poder resolver el problema de Josefo
void josefo(int n) {
   int k = 2 * (n - pow(2, (int)log2(n))) + 1; // Fórmula para encontrar el número que debe ser el último
en sobrevivir
   int posicion_superviviente = (k % n); // Calcular la posición del superviviente

   Cola* inicio = crearCola(1);
   Cola* actual = inicio;

   // Para crear una cola circular ligada
   int i;
   for ( i = 2; i <= n; i++) {
      actual -> siguiente = crearCola(i);
      actual = actual->siguiente;
   }
   actual->siguiente = inicio;

   // Para imprimir la respuesta al problema de Josefo
   p("Orden de supervivientes: ");
   while (actual->siguiente != actual) {
      for (i = 1; i < k; i++) {
         actual = actual->siguiente;
      }
      Cola* eliminado = actual->siguiente;
      p("%d ", eliminado->dato);
      actual->siguiente = eliminado->siguiente;
      eliminarCola(eliminado);
   }
   p("%d\n", actual->dato);//imrpime el valor del último dato en la memoria después de ser eliminados
los pasados

   // Imprimir la posición del superviviente
```

```
    p("La posición del superviviente es: %d\n", posicion_superviviente);

    // Liberar la memoria
    eliminarCola(actual);
}

int main() {

    int n;
    p("\n Ingrese el número de personas participantes (n): ");
    s("%d", &n);

    josefo(n);

    return 0;
}
```