Universidad Tecmilenio

ESTRUCTURAS DE DATOS

Alumnos:

Edwin Manuel Aguilar Márquez

Jesús Alberto Sandoval Serrano

Citlaly Guadalupe Zeferino Sierra

Proyecto Final

1. Introducción.

En este proyecto se utilizará lo aprendido en la materia de Estructuras de datos para la creación de una aplicación avanzada sobre una Taquería, implementando tanto estructuras básicas como algoritmos complejos de computación. El sistema utiliza Colas para registrar y atender clientes de manera ordenada (FIFO), Pilas para gestionar platos sucios (LIFO), Colas de Prioridad para manejar ingredientes según fecha de caducidad, Listas Enlazadas para el menú de platillos, y adicionalmente implementa Árboles Binarios de Búsqueda para gestión eficiente de pedidos, HashMap para acceso instantáneo a datos, Grafos para modelar dependencias entre tareas, y algoritmos avanzados como recursividad, divide y vencerás, y búsquedas optimizadas.

Esta aplicación integral simula un sistema de gestión empresarial real, donde cada estructura de datos tiene un propósito específico basado en sus características de rendimiento y funcionalidad. El proyecto demuestra cómo la correcta selección e implementación de estructuras de datos puede optimizar operaciones complejas en entornos de alta demanda como un restaurante.

2. Diseño del programa:

a. **Estructura del programa:** El sistema está dividido en múltiples componentes interconectados:

Estructuras básicas: Pila, cola, lista enlazada y cola de prioridad

Estructuras avanzadas: Árbol binario de búsqueda, HashMap y grafos

Algoritmos complejos: Recursividad, divide y vencerás, y búsquedas optimizadas

Sistema de gestión jerárquica: Tareas con subtareas anidadas

b. **Pilas:** Maneja platos sucios con operaciones push, pop y peek, implementando el comportamiento LIFO donde el último plato en llegar es el primero en lavarse.
c. **Colas:** Administra clientes en secuencia FIFO con operaciones enqueue, dequeue y front, asegurando que los primeros en llegar sean los primeros en ser atendidos.
d. **Listas:** Gestiona el menú con acceso aleatorio para consulta de platillos, con operaciones insert, delete y find.
e. **Árbol Binario de Búsqueda:** Organiza pedidos automáticamente por número, permitiendo búsquedas O(log n), inserciones ordenadas y tres tipos de recorridos (In-Order, Pre-Order, Post-Order).
f. **HashMap:** Proporciona acceso O(1) a tareas por ID y empleados por código, optimizando las consultas frecuentes.
g. **Grafos:** Modela dependencias entre tareas usando DFS y orden topológico para determinar secuencias de ejecución válidas.

3. **Funcionalidad adicional:**

a. Implementa una interfaz de usuario sencilla en la consola para interactuar con el sistema.

b. Permite al usuario agregar, ver y eliminar tareas en cada estructura de datos.

c. Incluye una opción para ver todas las tareas pendientes, ordenadas por urgencia y departamento.

4. **Explicación sobre fragmentos de código importantes**

El objetivo de la actividad es simular cómo se administran los clientes, platos, ingredientes y el menú aplicando los conceptos de colas, pilas, listas y colas de prioridad.

En la primera parte se encuentra la clase Node, que nos sirve como base para todas las estructuras de datos. Este Nodo puede aguardar el dato de un puntero al siguiente Nodo y en las colas prioritarias tiene un campo de prioridad.

```
// Clase Nodo - Estructura básica para todas nuestras estructuras de datos
class Node {
    Object dato;           // Almacena el dato (puede ser cualquier tipo de objeto)
    Node siguiente;        // Puntero al siguiente nodo en la estructura
    int prioridad;         // Campo especial para cola de prioridad

    // Constructor básico para nodos normales
    public Node(Object dato) {
        this.dato = dato;
        this.siguiente = null;
        this.prioridad = 0;
    }

    // Constructor para nodos con prioridad (ingredientes)
    public Node(Object dato, int prioridad) {
        this.dato = dato;
        this.siguiente = null;
        this.prioridad = prioridad;
    }
}
```

```java
// Clase Lista Enlazada - Base para todas nuestras estructura
class LinkedList {
    private Node cabeza;   // Primer nodo de la lista
    private int tamaño;    // Contador de elementos

    public LinkedList() {
        cabeza = null;
        tamaño = 0;
    }
```

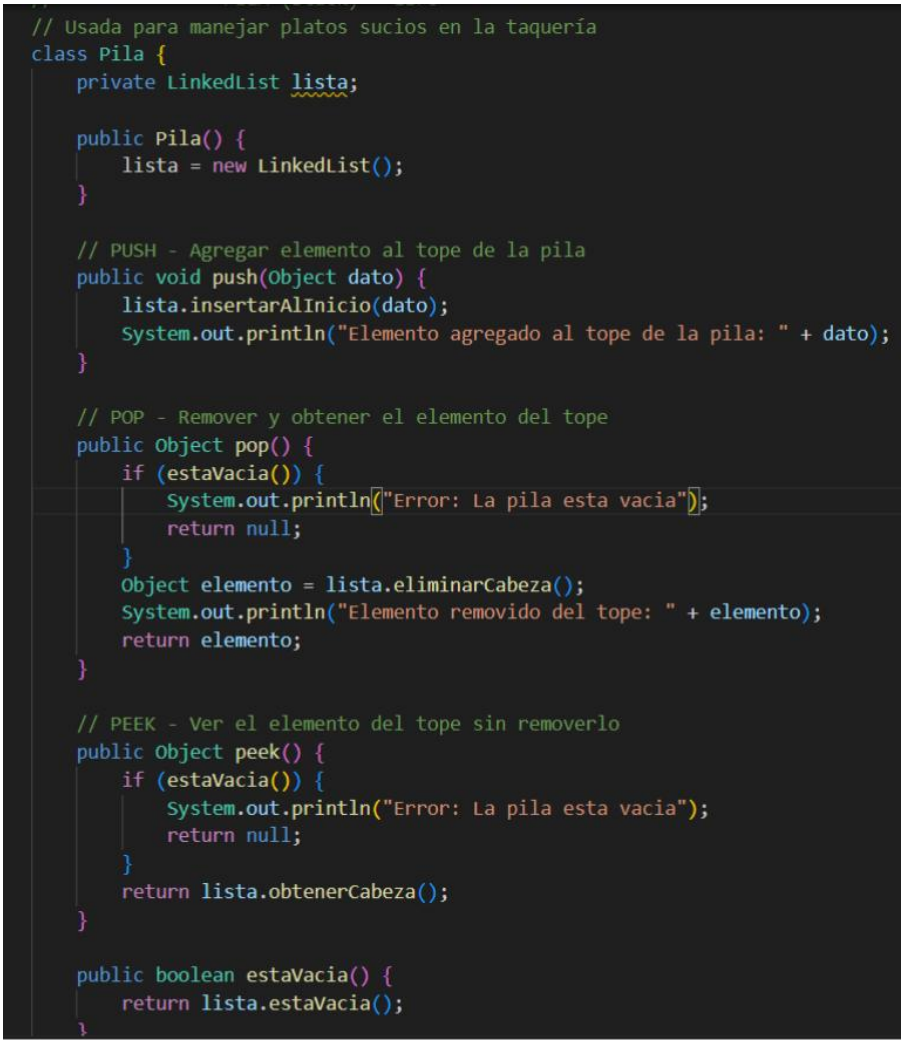
```java
    // Insertar al inicio - Útil para pilas
    public void insertarAlInicio(Object dato) {
        Node nuevo = new Node(dato);
        nuevo.siguiente = cabeza;
        cabeza = nuevo;
        tamaño++;
    }

    // Insertar al final - Útil para colas
    public void insertarAlFinal(Object dato) {
        Node nuevo = new Node(dato);
        if (cabeza == null) {
            cabeza = nuevo;
        } else {
            Node actual = cabeza;
            while (actual.siguiente != null) {
                actual = actual.siguiente;
            }
            actual.siguiente = nuevo;
        }
        tamaño++;
    }
```

Implementamos la clase LinkedList, está nos sirve como la parte central que manejan operaciones básicas como insertar el inicio o insertar al final, puede eliminar Nodos, consultar elementos, buscar por posición, mostrar la lista y verificar si está vacía.

```java
// Eliminar el primer elemento
public Object eliminarCabeza() {
    if (cabeza == null) return null;
    Object dato = cabeza.dato;
    cabeza = cabeza.siguiente;
    tamaño--;
    return dato;
}

// Eliminar el último elemento
public Object eliminarUltimo() {
    if (cabeza == null) return null;
    if (cabeza.siguiente == null) {
        Object dato = cabeza.dato;
        cabeza = null;
        tamaño--;
        return dato;
    }
    Node actual = cabeza;
    while (actual.siguiente.siguiente != null) {
        actual = actual.siguiente;
    }
    Object dato = actual.siguiente.dato;
    actual.siguiente = null;
    tamaño--;
    return dato;
}
```

```java
// Verificar si la lista está vacía
public boolean estaVacia() {
    return cabeza == null;
}

// Mostrar todos los elementos
public void mostrar() {
    Node actual = cabeza;
    int contador = 1;
    while (actual != null) {
        System.out.println(contador + ". " + actual.dato);
        actual = actual.siguiente;
        contador++;
    }
    if (contador == 1) System.out.println("Lista vacía");
}
```

Las clases pilas usan listas enlazadas para el comportamiento LIFO, el último elemento en entrar es el primero en salir, este se usa para los platos sucios que se van apilando y debe lavarse en orden inverso al que llegan.



```java
// Usada para manejar platos sucios en la taquería
class Pila {
    private LinkedList lista;

    public Pila() {
        lista = new LinkedList();
    }

    // PUSH - Agregar elemento al tope de la pila
    public void push(Object dato) {
        lista.insertarAlInicio(dato);
        System.out.println("Elemento agregado al tope de la pila: " + dato);
    }

    // POP - Remover y obtener el elemento del tope
    public Object pop() {
        if (estaVacia()) {
            System.out.println("Error: La pila esta vacia");
            return null;
        }
        Object elemento = lista.eliminarCabeza();
        System.out.println("Elemento removido del tope: " + elemento);
        return elemento;
    }

    // PEEK - Ver el elemento del tope sin removerlo
    public Object peek() {
        if (estaVacia()) {
            System.out.println("Error: La pila esta vacia");
            return null;
        }
        return lista.obtenerCabeza();
    }

    public boolean estaVacia() {
        return lista.estaVacia();
    }
```

La clase de colas nos apoya en las listas enlazadas, pero los implementa un esquema FIFO, En donde los primeros clientes en llegar son los primeros en atendidos

Las colas de prioridad se generan directamente con los Nodos, este nos ordena los productos por su nivel de prioridad, y se utiliza para para manejar los ingredientes en función de su fecha de caducidad.

```java
// Usada para manejar la fila de clientes en la taquería
class Cola {
    private LinkedList lista;

    public Cola() {
        lista = new LinkedList();
    }

    // ENQUEUE - Agregar elemento al final de la cola
    public void enqueue(Object dato) {
        lista.insertarAlFinal(dato);
        System.out.println("Cliente agregado a la cola: " + dato);
    }

    // DEQUEUE - Remover y obtener el primer elemento de la cola
    public Object dequeue() {
        if (estaVacia()) {
            System.out.println("Error: No hay clientes en la cola");
            return null;
        }
        Object elemento = lista.eliminarCabeza();
        System.out.println("Cliente atendido: " + elemento);
        return elemento;
    }

    // PEEK/FRONT - Ver el primer elemento sin removerlo
    public Object front() {
        if (estaVacia()) {
            System.out.println("Error: No hay clientes en la cola");
            return null;
        }
        return lista.obtenerCabeza();
    }

    public boolean estaVacia() {
```

```java
public ColaPrioridad() {
    cabeza = null;
    tamaño = 0;
}

// ENQUEUE con prioridad - Insertar elemento según su prioridad
public void enqueue(Object dato, int prioridad) {
    Node nuevo = new Node(dato, prioridad);

    // Si la cola está vacía o el nuevo elemento tiene mayor prioridad que el primero
    if (cabeza == null || prioridad < cabeza.prioridad) {
        nuevo.siguiente = cabeza;
        cabeza = nuevo;
    } else {
        // Buscar la posición correcta para insertar según prioridad
        Node actual = cabeza;
        while (actual.siguiente != null && actual.siguiente.prioridad <= prioridad) {
            actual = actual.siguiente;
        }
        nuevo.siguiente = actual.siguiente;
        actual.siguiente = nuevo;
    }
    tamaño++;
    System.out.println("Ingrediente agregado con prioridad " + prioridad + ": " + dato);
}

// DEQUEUE - Remover elemento de mayor prioridad
public Object dequeue() {
    if (estaVacia()) {
        System.out.println("Error: No hay ingredientes en la cola de prioridad");
        return null;
    }
    Object elemento = cabeza.dato;
    int prioridad = cabeza.prioridad;
    cabeza = cabeza.siguiente;
```

En la clase Elemento Menú en el código representa cada platillo de nuestra taqueria, contiene el nombre, la descripción y el precio.

```java
class ElementoMenu {
    private String nombre;
    private String descripcion;
    private double precio;

    public ElementoMenu(String nombre, String descripcion, double precio) {
        this.nombre = nombre;
        this.descripcion = descripcion;
        this.precio = precio;
    }

    public String getNombre() { return nombre; }
    public String getDescripcion() { return descripcion; }
    public double getPrecio() { return precio; }

    @Override
    public String toString() {
        return nombre + " - $" + precio;
    }

    public String detalles() {
        return "=== " + nombre + " ===" +
               "\nDescripción: " + descripcion +
               "\nPrecio: $" + precio;
    }
}
```

En la clase de lista menú contiene una lista enlazada, inicia con tacos, quesadillas y bebidas, y permite mostrar la lista o acceder a los detalles de cada elemento según la posición seleccionada. También los permite mostrar el menú, eliminar un elemento del menú agregar al menú y buscar en el menú.

```java
class ListaMenu {
    private LinkedList lista;

    public ListaMenu() {
        lista = new LinkedList();
        inicializarMenu();
    }

    // Inicializar menú con elementos predeterminados
    private void inicializarMenu() {
        agregar(new ElementoMenu("Taco de Carnitas",
                "Delicioso taco de carnitas con cebolla, cilantro y salsa verde", 25.0));
        agregar(new ElementoMenu("Taco de Pollo",
                "Taco de pollo a la plancha con pico de gallo y salsa roja", 20.0));
        agregar(new ElementoMenu("Taco de Pastor",
                "Taco al pastor con piña, cebolla, cilantro y salsa", 22.0));
        agregar(new ElementoMenu("Quesadilla",
                "Quesadilla de queso Oaxaca con opción de agregar carne", 30.0));
        agregar(new ElementoMenu("Agua de Horchata",
                "Refrescante agua de horchata casera", 15.0));
    }

    // INSERT - Agregar elemento al menú
    public void agregar(ElementoMenu elemento) {
        lista.insertarAlFinal(elemento);
    }

    // FIND - Buscar elemento por número de posición (MÉTODO CORREGIDO)
    public ElementoMenu buscar(int posicion) {
        if (posicion < 1 || posicion > lista.getTamaño()) {
            return null;
        }

        Object elemento = lista.obtenerPorPosicion(posicion);
```

La clase principal Taquería, Aquí se integran las estructuras: una cola para gestionar clientes en orden de llegada, una pila para administrar los platos sucios que deben lavarse, una cola de prioridad para el control de ingredientes y una lista enlazada que almacena el menú de la taquería.

```java
// ============= CLASE PRINCIPAL DEL SISTEMA =============
public class Taqueria {
    private static Scanner scanner = new Scanner(System.in);

    // Instancias de nuestras estructuras de datos
    private Cola colaClientes;           // FIFO para atender clientes en orden
    private Pila pilaPlatos;             // LIFO para manejar platos sucios
    private ColaPrioridad colaIngredientes; // Para ingredientes por fecha de caducidad
    private ListaMenu menu;              // Lista enlazada para acceso aleatorio al menú

    public Taqueria() {
        colaClientes = new Cola();
        pilaPlatos = new Pila();
        colaIngredientes = new ColaPrioridad();
        menu = new ListaMenu();

        System.out.println("Bienvenido al Sistema de Gestion de la Taqueria!");
    }
```

Se construyen los métodos para colas, pilas, colas de prioridad y listas

```java
// ============ MÉTODOS PARA COLA DE CLIENTES ============

private void agregarCliente() {
    System.out.print("Ingrese el nombre del cliente: ");
    String nombreCliente = scanner.nextLine();
    colaClientes.enqueue("Cliente: " + nombreCliente);
    System.out.println("Cliente agregado a la fila de espera.");
}

private void atenderCliente() {
    if (colaClientes.estaVacia()) {
        System.out.println("No hay clientes esperando.");
        return;
    }

    Object cliente = colaClientes.front();
    System.out.println("Atendiendo a: " + cliente);
    System.out.print("Presione Enter para completar el servicio...");
    scanner.nextLine();
    colaClientes.dequeue();
}

private void verProximoCliente() {
    Object siguiente = colaClientes.front();
    if (siguiente != null) {
        System.out.println("Próximo cliente a atender: " + siguiente);
    }
}

private void mostrarColaClientes() {
    colaClientes.mostrar();
}
```

```java
// ============ MÉTODOS PARA PILA DE PLATOS ============

private void apilarPlatoSucio() {
    System.out.print("Ingrese el número del plato sucio: ");
    String numeroPlato = scanner.nextLine();
    pilaPlatos.push("Plato #" + numeroPlato);
    System.out.println("Plato apilado en la zona de lavado.");
}

private void lavarPlato() {
    if (pilaPlatos.estaVacia()) {
        System.out.println("No hay platos para lavar.");
        return;
    }

    Object plato = pilaPlatos.peek();
    System.out.println("Lavando " + plato + "...");
    System.out.print("Presione Enter para terminar de lavar...
    scanner.nextLine();
    pilaPlatos.pop();
    System.out.println("Plato limpio y listo para usar!");
}

private void verTopePlatos() {
    Object tope = pilaPlatos.peek();
    if (tope != null) {
        System.out.println("Próximo plato a lavar: " + tope);
    }
}

private void mostrarPilaPlatos() {
    pilaPlatos.mostrar();
}
```

```java
// ============ MÉTODOS PARA COLA DE PRIORIDAD (INGREDIENTES) ========

private void agregarIngrediente() {
    System.out.print("Ingrese el nombre del ingrediente: ");
    String ingrediente = scanner.nextLine();

    System.out.println("Seleccione el nivel de prioridad:");
    System.out.println("1. Alta prioridad (1-7 días para caducar)");
    System.out.println("2. Prioridad media (8-30 días para caducar)");
    System.out.println("3. Prioridad baja (1-12 meses para caducar)");
    System.out.print("Opción: ");

    int prioridad = scanner.nextInt();
    scanner.nextLine(); // Limpiar buffer

    if (prioridad >= 1 && prioridad <= 3) {
        colaIngredientes.enqueue(ingrediente, prioridad);
    } else {
        System.out.println("Error: Prioridad debe ser 1, 2 o 3");
    }
}

private void procesarIngrediente() {
    if (colaIngredientes.estaVacia()) {
        System.out.println("No hay ingredientes para procesar.");
        return;
    }

    Object ingrediente = colaIngredientes.peek();
    System.out.println("Procesando ingrediente: " + ingrediente);
    System.out.print("Presione Enter para confirmar uso...");
    scanner.nextLine();
    colaIngredientes.dequeue();
    System.out.println("Ingrediente utilizado correctamente.");
}
```

```java
// ============ MÉTODOS PARA LISTA DEL MENÚ ============

private void mostrarMenu() {
    menu.mostrarMenu();
}

private void verDetallesMenu() {
    System.out.println("=== MENÚ DETALLADO ===");
    menu.mostrarMenu();
    System.out.print("Seleccione un número para ver detalles: ");

    try {
        int opcion = scanner.nextInt();
        scanner.nextLine(); // Limpiar buffer
        menu.mostrarDetalles(opcion);
    } catch (Exception e) {
        System.out.println("Error: Ingrese un numero valido");
        scanner.nextLine();
    }
}

private void mostrarEstadoGeneral() {
    System.out.println("\n==========================================
    System.out.println("          ESTADO GENERAL DE LA TAQUERIA");
    System.out.println("==========================================

    System.out.println("CLIENTES:");
    if (colaClientes.estaVacia()) {
        System.out.println("   No hay clientes esperando");
    } else {
        Object siguiente = colaClientes.front();
        System.out.println("   Proximo a atender: " + siguiente);
    }

    System.out.println("\nPLATOS:");
    if (pilaPlatos.estaVacia()) {
        System.out.println("   No hay platos sucios");
    } else {
        Object tope = pilaPlatos.peek();
        System.out.println("   Proximo a lavar: " + tope);
    }

    System.out.println("\nINGREDIENTES:");
    if (colaIngredientes.estaVacia()) {
        System.out.println("   No hay ingredientes registrados");
    } else {
        Object prioritario = colaIngredientes.peek();
        System.out.println("   Proximo a usar: " + prioritario);
    }

    System.out.println("==========================================");
}

public void mostrarMenuPrincipal() {
    int opcion;

    do {
        System.out.println("\n==========================================");
        System.out.println("      SISTEMA DE GESTION DE TAQUERIA");
        System.out.println("==========================================");
        System.out.println("GESTION DE CLIENTES (Cola - FIFO):");
        System.out.println("  1. Agregar cliente a la fila");
        System.out.println("  2. Atender proximo cliente");
        System.out.println("  3. Ver proximo cliente");
        System.out.println("  4. Mostrar cola completa de clientes");

        System.out.println("\nGESTION DE PLATOS (Pila - LIFO):");
        System.out.println("  5. Apilar plato sucio");
        System.out.println("  6. Lavar plato (del tope)");
        System.out.println("  7. Ver proximo plato a lavar");
```

```
    System.out.println("  8. Mostrar pila completa de platos");

    System.out.println("\nGESTION DE INGREDIENTES (Cola de Prioridad):");
    System.out.println("  9. Agregar ingrediente");
    System.out.println("  10. Usar ingrediente prioritario");
    System.out.println("  11. Ver proximo ingrediente a usar");
    System.out.println("  12. Mostrar cola completa de ingredientes");

    System.out.println("\nMENU DE LA TAQUERIA (Lista Enlazada):");
    System.out.println("  13. Mostrar menu");
    System.out.println("  14. Ver detalles de platillo");

    System.out.println("\nREPORTES:");
    System.out.println("  15. Estado general del sistema");

    System.out.println("\nSALIDA:");
    System.out.println("  16. Salir del sistema");

    System.out.println("==========================================");
    System.out.print("Seleccione una opcion: ");
```

```
try {
    opcion = scanner.nextInt();
    scanner.nextLine();

    switch (opcion) {
        case 1: agregarCliente(); break;
        case 2: atenderCliente(); break;
        case 3: verProximoCliente(); break;
        case 4: mostrarColaClientes(); break;
        case 5: apilarPlatoSucio(); break;
        case 6: lavarPlato(); break;
        case 7: verTopePlatos(); break;
        case 8: mostrarPilaPlatos(); break;
        case 9: agregarIngrediente(); break;
        case 10: procesarIngrediente(); break;
        case 11: verProximoIngrediente(); break;
        case 12: mostrarColaIngredientes(); break;
        case 13: mostrarMenu(); break;
        case 14: verDetallesMenu(); break;
        case 15: mostrarEstadoGeneral(); break;
        case 16:
            System.out.println("Gracias por usar el Sistema de Gestion de la Taqueria!");
            break;
        default:
            System.out.println("Opcion invalida. Seleccione un numero del 1 al 16.");
            break;
    }

    if (opcion != 16) {
        System.out.print("\nPresione Enter para continuar...");
        scanner.nextLine();
    }

} catch (Exception e) {
    System.out.println("Error: Por favor ingrese un numero valido.");
```

el método main inicia el sistema mostrando un mensaje de bienvenida, explica qué estructuras de datos están implementadas y abre el menú interactivo. A través de este menú el usuario puede navegar por las opciones y simular el funcionamiento de la taquería, hasta decidir salir del programa.

```
Run main | Debug main
public static void main(String[] args) {
    Taqueria taqueria = new Taqueria();

    System.out.println("\nESTRUCTURAS DE DATOS IMPLEMENTADAS:");
    System.out.println("Cola (FIFO): Para manejar clientes en orden de llegada");
    System.out.println("Pila (LIFO): Para manejar platos sucios");
    System.out.println("Cola de Prioridad: Para ingredientes segun fecha de caducidad");
    System.out.println("Lista Enlazada: Para acceso aleatorio al menu");

    System.out.print("\nPresione Enter para comenzar...");
    scanner.nextLine();

    taqueria.mostrarMenuPrincipal();
    scanner.close();
}
```

Clase Pedido - Estructura de datos para el árbol binario

```
30  class Pedido {
31      private int numeroPedido;
32      private String horaLlegada;
33      private String detallesPlatillos;
34      private int prioridad;
35      private String cliente;
36
37      public Pedido(int numeroPedido, String horaLlegada, String detallesPlatillos, int prioridad, String cliente) {
38          this.numeroPedido = numeroPedido;
39          this.horaLlegada = horaLlegada;
40          this.detallesPlatillos = detallesPlatillos;
41          this.prioridad = prioridad;
42          this.cliente = cliente;
43      }
```

Esta clase representa cada pedido en el sistema. Encapsula toda la información relevante de un pedido: número único para ordenamiento en el árbol, hora para seguimiento temporal, detalles de los platillos solicitados, prioridad para gestión urgente y nombre del cliente para personalización del servicio.

Inserción recursiva en árbol binario

```java
    private NodoArbol insertarRecursivo(NodoArbol nodo, Pedido pedido) {
        // Caso base: si el nodo es null, crear nuevo nodo
        if (nodo == null) {
            return new NodoArbol(pedido);
        }

        // Insertar según el número de pedido (criterio de ordenamiento)
        if (pedido.getNumeroPedido() < nodo.pedido.getNumeroPedido()) {
            nodo.izquierdo = insertarRecursivo(nodo.izquierdo, pedido);
        } else if (pedido.getNumeroPedido() > nodo.pedido.getNumeroPedido()) {
            nodo.derecho = insertarRecursivo(nodo.derecho, pedido);
        }
        // Si el número es igual, no insertar (evitar duplicados)

        return nodo;
    }
```

Este método implementa la inserción recursiva en el árbol binario de búsqueda. El caso base ocurre cuando encuentra un nodo vacío donde insertar. Los casos recursivos navegan hacia el subárbol izquierdo o derecho según el valor del número de pedido, manteniendo automáticamente el orden del árbol.

Recorrido In-Order recursivo

```java
    private void inOrdenRecursivo(NodoArbol nodo) {
        if (nodo != null) {
            inOrdenRecursivo(nodo.izquierdo);
            System.out.println(nodo.pedido);
            System.out.println(x:"---");
            inOrdenRecursivo(nodo.derecho);
        }
    }
```

El recorrido In-Order visita primero el subárbol izquierdo, luego procesa el nodo actual, y finalmente el subárbol derecho. Esto garantiza que los pedidos se muestren en orden cronológico ascendente por número, aprovechando la propiedad fundamental del árbol binario de búsqueda.

Cálculo recursivo de tiempo total

```
488     public int calcularTiempoTotal(Tarea tarea) {
489         int tiempoTotal = tarea.getTiempoEstimado();
490 
491         // Caso base: si no hay subtareas, retorna el tiempo de la tarea
492         if (tarea.getSubtareas().isEmpty()) {
493             return tiempoTotal;
494         }
495 
496         // Caso recursivo: suma el tiempo de todas las subtareas
497         for (Tarea subtarea : tarea.getSubtareas()) {
498             tiempoTotal += calcularTiempoTotal(subtarea);
499         }
500 
501         return tiempoTotal;
502     }
```

Esta función recursiva calcula el tiempo total de una tarea principal sumando todos sus componentes. El caso base retorna el tiempo cuando no hay subtareas. El caso recursivo itera por todas las subtareas, llamándose a sí mismo para cada una y acumulando los resultados.

Algoritmo Merge Sort - Divide y vencerás

```
528     public List<Tarea> mergeSort(List<Tarea> tareas, String criterio) {
529         if (tareas.size() <= 1) {
530             return tareas;
531         }
532 
533         int medio = tareas.size() / 2;
534         List<Tarea> izquierda = new ArrayList<>(tareas.subList(fromIndex:0, medio));
535         List<Tarea> derecha = new ArrayList<>(tareas.subList(medio, tareas.size()));
536 
537         izquierda = mergeSort(izquierda, criterio);
538         derecha = mergeSort(derecha, criterio);
539 
540         return merge(izquierda, derecha, criterio);
541     }
```

Implementa el algoritmo divide y vencerás de Merge Sort. Divide la lista por la mitad recursivamente hasta obtener elementos individuales, luego los combina ordenadamente. La complejidad O(n log n) garantiza eficiencia incluso con grandes volúmenes de tareas.

HashMap para acceso O(1)

Aquí se crearon los HashMaps

```
402   class GestorTareasAvanzado {
403       private HashMap<String, Tarea> tablaTareas;
404       private HashMap<String, Empleado> tablaEmpleados;
```

Aquí se crean las funciones getters para los elementos dentro de los HashMaps tablaTareas y tablaEmpleados

```
875       // Acceso directo O(1) usando HashMap
876       public Tarea obtenerTareaPorId(String id) {
877           return tablaTareas.get(id);
878       }
879
880       public Empleado obtenerEmpleadoPorId(String id) {
881           return tablaEmpleados.get(id);
882       }
```

Los HashMap proporcionan acceso en tiempo constante O(1) a elementos usando una clave única. Esto optimiza significativamente las búsquedas frecuentes de tareas y empleados comparado con búsquedas lineales O(n) en listas tradicionales.

Búsqueda binaria optimizada

```
637   public Tarea busquedaBinaria(List<Tarea> tareas, int tiempoBuscado) {
638       int inicio = 0;
639       int fin = tareas.size() - 1;
640
641       while (inicio <= fin) {
642           int medio = (inicio + fin) / 2;
643           int tiempoMedio = tareas.get(medio).getTiempoEstimado();
644
645           if (tiempoMedio == tiempoBuscado) {
646               return tareas.get(medio);
647           } else if (tiempoMedio < tiempoBuscado) {
648               inicio = medio + 1;
649           } else {
650               fin = medio - 1;
651           }
652       }
653
654       return null;
655   }
```

La búsqueda binaria reduce el espacio de búsqueda a la mitad en cada iteración, logrando complejidad O(log n). Requiere que la lista esté previamente ordenada, pero ofrece ventajas significativas sobre la búsqueda lineal en conjuntos grandes de datos.

DFS para recorrido de grafos

```
682     public void dfsRecorrido(String tareaId, Set<String> visitados) {
683         if (visitados.contains(tareaId)) return;
684 
685         visitados.add(tareaId);
686         System.out.println("Visitando tarea: " + tablaTareas.get(tareaId).getNombre());
687 
688         NodoGrafo nodo = grafoTareas.get(tareaId);
689         if (nodo != null) {
690             for (NodoGrafo dependencia : nodo.dependencias) {
691                 dfsRecorrido(dependencia.tareaId, visitados);
692             }
693         }
694     }
```

El algoritmo DFS (Depth-First Search) explora las dependencias de una tarea hasta llegar a nodos sin dependencias, luego retrocede para explorar otros caminos. El conjunto de visitados previene ciclos infinitos y garantiza que cada tarea se procese exactamente una vez.

Balanceo recursivo de carga de trabajo

```
613     private void balancearRecursivo(List<Tarea> tareas, List<Empleado> empleados, int indiceEmpleado) {
614         if (tareas.isEmpty()) return;
615 
616         if (tareas.size() == 1) {
617             // Asignar única tarea al empleado actual
618             Empleado empleado = empleados.get(indiceEmpleado % empleados.size());
619             tareas.get(index:0).setEmpleadoAsignado(empleado.getNombre());
620             empleado.asignarTarea(tareas.get(index:0).getId());
621             return;
622         }
623 
624         // Dividir tareas por la mitad
625         int medio = tareas.size() / 2;
626         List<Tarea> primera = tareas.subList(fromIndex:0, medio);
627         List<Tarea> segunda = tareas.subList(medio, tareas.size());
628 
629         // Asignar recursivamente a diferentes empleados
630         balancearRecursivo(primera, empleados, indiceEmpleado);
631         balancearRecursivo(segunda, empleados, (indiceEmpleado + 1) % empleados.size());
632     }
```

Este algoritmo implementa divide y vencerás para distribuir equitativamente las tareas entre empleados disponibles. Divide recursivamente la lista de tareas y alterna la asignación entre empleados, asegurando una carga de trabajo balanceada automáticamente.

Eliminación completa en árbol binario

```java
private NodoArbol eliminarRecursivo(NodoArbol nodo, int numeroPedido) {
    // Caso base
    if (nodo == null) {
        return null;
    }

    // Buscar el nodo a eliminar
    if (numeroPedido < nodo.pedido.getNumeroPedido()) {
        nodo.izquierdo = eliminarRecursivo(nodo.izquierdo, numeroPedido);
    } else if (numeroPedido > nodo.pedido.getNumeroPedido()) {
        nodo.derecho = eliminarRecursivo(nodo.derecho, numeroPedido);
    } else {
        // Nodo encontrado - eliminar
        totalPedidos--;

        // Caso 1: Nodo sin hijos
        if (nodo.izquierdo == null && nodo.derecho == null) {
            return null;
        }

        // Caso 2: Nodo con un hijo
        if (nodo.izquierdo == null) {
            return nodo.derecho;
        }
        if (nodo.derecho == null) {
            return nodo.izquierdo;
        }

        // Caso 3: Nodo con dos hijos
        // Encontrar el sucesor inOrden (mínimo en subárbol derecho)
        NodoArbol sucesor = encontrarMinimo(nodo.derecho);
        nodo.pedido = sucesor.pedido;
        nodo.derecho = eliminarRecursivo(nodo.derecho, sucesor.pedido.getNumeroPedido());
    }

    return nodo;
}
```

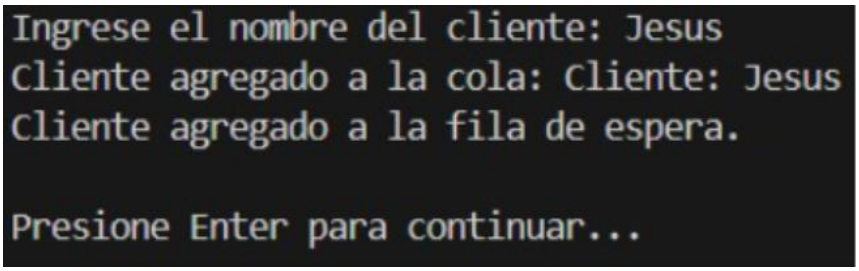
La eliminación en árboles binarios maneja tres casos: nodo hoja (sin hijos), nodo con un hijo, y nodo con dos hijos. El caso más complejo requiere encontrar el sucesor inorder (menor valor del subárbol derecho) para mantener las propiedades del árbol después de la eliminación.

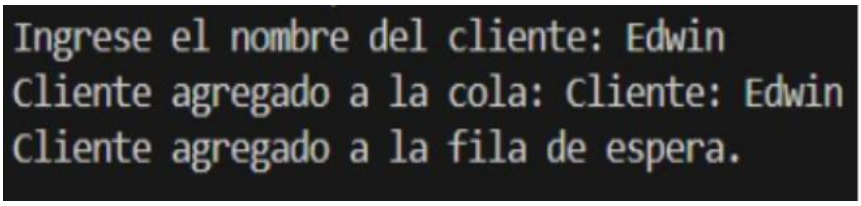
Estos fragmentos demuestran la implementación práctica de algoritmos fundamentales de ciencias de la computación aplicados a un sistema de gestión real, mostrando cómo las estructuras de datos apropiadas optimizan operaciones específicas del dominio del problema.

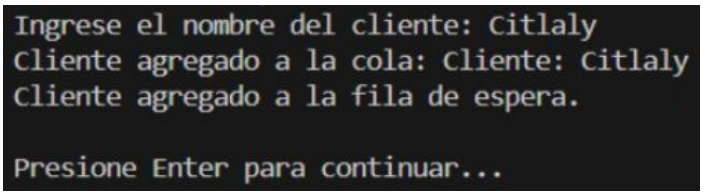
PRUEBAS DE EJECUCIÓN

5. **GESTIÓN DE CLIENTES (Cola – FIFO)**

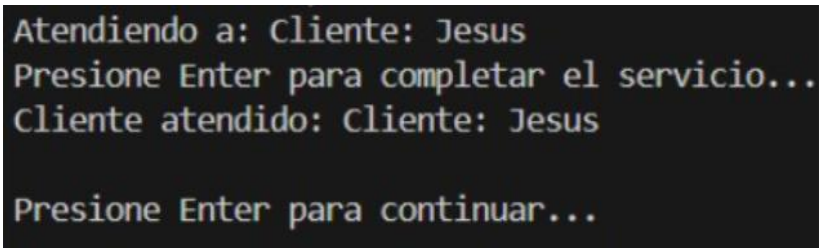
Agregar cliente a la fila



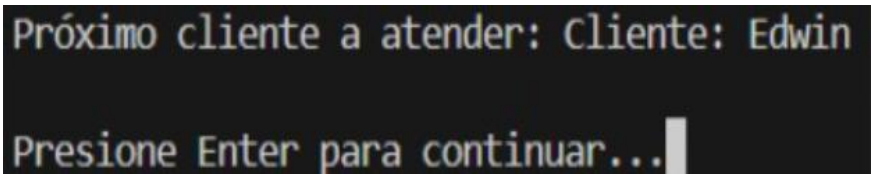




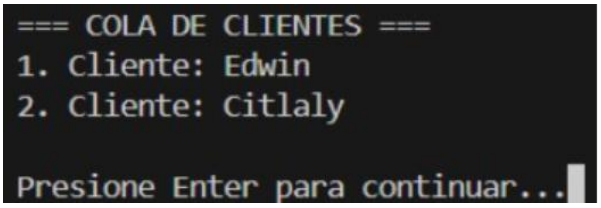
Atender próximo cliente



Ver próximo cliente

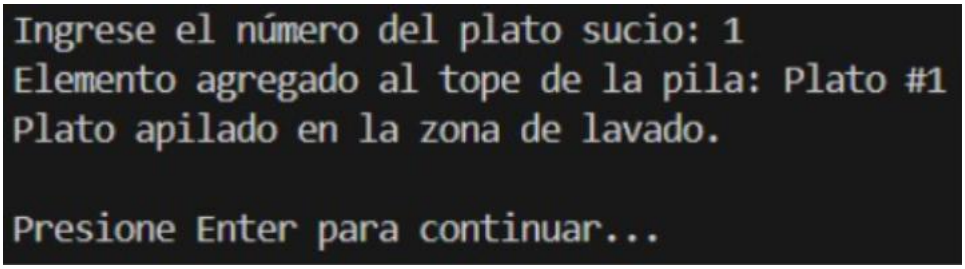


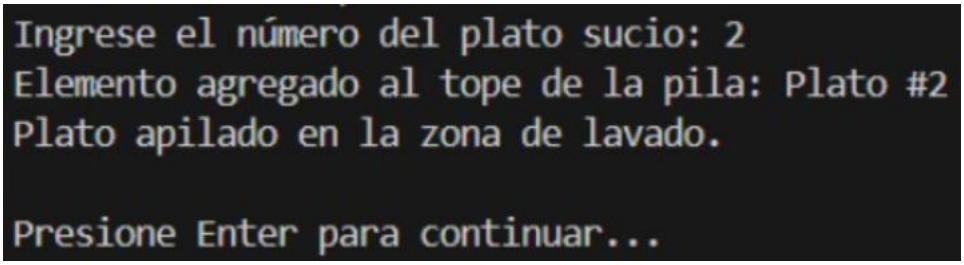
Mostrar cola completa de clientes

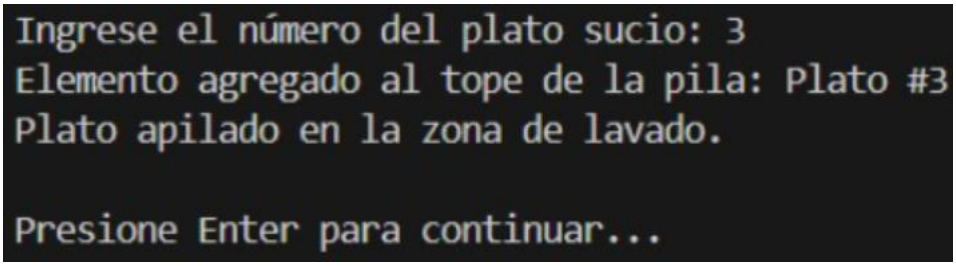


6. GESTIÓN DE PLATOS (Pila – LIFO)

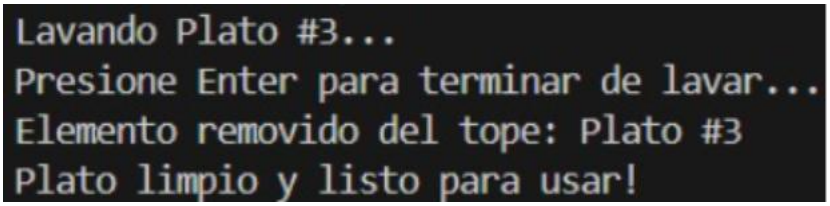
Apilar plato sucio







Lavar plato (del tope)



Ver próximo plato a lavar

```
Próximo plato a lavar: Plato #2

Presione Enter para continuar...
```

Mostrar pila completa de platos

```
=== CONTENIDO DE LA PILA ===
1. Plato #2
2. Plato #1

Presione Enter para continuar...
```

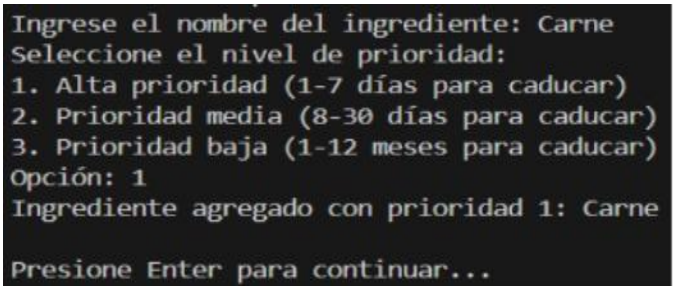
7. GESTIÓN DE INGREDIENTES (Cola de Prioridad)

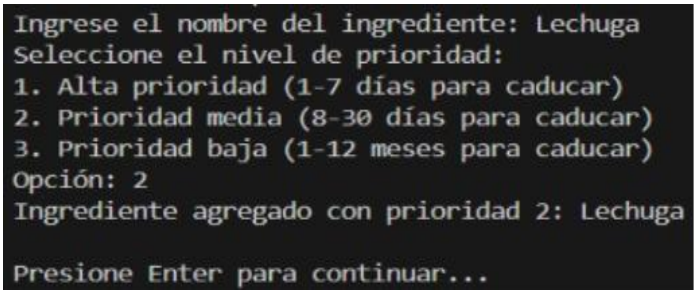
Agregar ingrediente

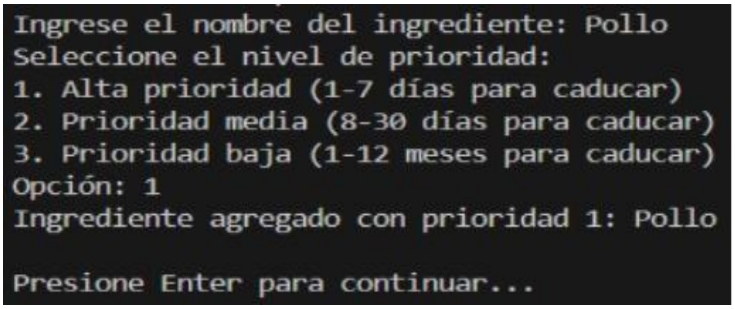
```
Ingrese el nombre del ingrediente: Atun
Seleccione el nivel de prioridad:
1. Alta prioridad (1-7 días para caducar)
2. Prioridad media (8-30 días para caducar)
3. Prioridad baja (1-12 meses para caducar)
Opción: 3
Ingrediente agregado con prioridad 3: Atun

Presione Enter para continuar...
```

```
Ingrese el nombre del ingrediente: Tomate
Seleccione el nivel de prioridad:
1. Alta prioridad (1-7 días para caducar)
2. Prioridad media (8-30 días para caducar)
3. Prioridad baja (1-12 meses para caducar)
Opción: 2
Ingrediente agregado con prioridad 2: Tomate

Presione Enter para continuar...
```

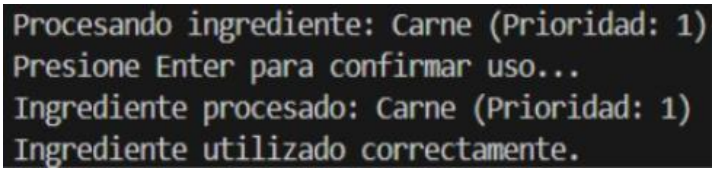
```
Ingrese el nombre del ingrediente: Sal
Seleccione el nivel de prioridad:
1. Alta prioridad (1-7 días para caducar)
2. Prioridad media (8-30 días para caducar)
3. Prioridad baja (1-12 meses para caducar)
Opción: 3
Ingrediente agregado con prioridad 3: Sal

Presione Enter para continuar...
```



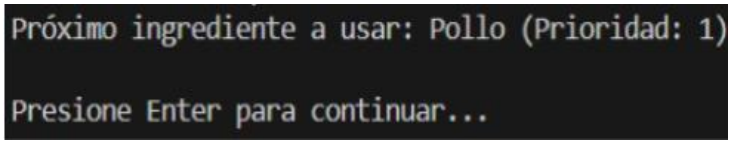




Usar ingrediente prioritario



Ver próximo ingrediente a usar



Mostrar cola completa de ingredientes

```
=== COLA DE INGREDIENTES POR PRIORIDAD ===
(Prioridad 1: 1-7 dias, Prioridad 2: 8-30 dias, Prioridad 3: 1-12 meses)
1. Pollo (Prioridad: 1)
2. Tomate (Prioridad: 2)
3. Lechuga (Prioridad: 2)
4. Atun (Prioridad: 3)
5. Sal (Prioridad: 3)

Presione Enter para continuar...
```

8. ARBOL BINARIO DE PEDIDOS

Menú general

```
=== GESTION DE PEDIDOS (Arbol Binario) ===
1. Agregar nuevo pedido
2. Buscar pedido por numero
3. Eliminar pedido entregado
4. Ver pedidos en orden cronologico (In-Orden)
5. Ver secuencia de insercion (Pre-Orden)
6. Generar reporte de fin de turno (Post-Orden)
7. Encontrar pedido con mayor prioridad
8. Mostrar estructura del arbol
9. Volver al menu principal
```

Agregar nuevo pedido

```
Opcion: 1
Ingrese numero de pedido: 1
Ingrese hora de llegada (HH:MM): 12:41
Ingrese detalles de los platillos: Taco de Carnitas
Ingrese prioridad (1=Alta, 2=Media, 3=Baja): 2
Ingrese nombre del cliente: Jesus Alberto
Pedido agregado al sistema: 1
Pedido agregado exitosamente al sistema.
Presione Enter para continuar...
```

Buscar pedido por número

```
Opcion: 2
Ingrese numero de pedido a buscar: 102
=== PEDIDO ENCONTRADO ===
Pedido #102 - Cliente: Ana Rodríguez - Hora: 14:10 - Prioridad: 1
  Platillos: 1 Taco de Pollo
Presione Enter para continuar...
```

Eliminar pedido entregado

```
Opcion: 3
Ingrese numero de pedido entregado: 108
Pedido #108 eliminado del sistema
Pedido entregado y eliminado del sistema correctamente.
Presione Enter para continuar...
```

Ver pedidos en orden cronológico (In-Orden)

```
Opcion: 4
=== PEDIDOS EN ORDEN CRONOLOGICO (IN-Orden) ===
Pedido #1 - Cliente: Jesus Alberto - Hora: 12:41 - Prioridad: 2
  Platillos: Taco de Carnitas
---
Pedido #102 - Cliente: Ana Rodríguez - Hora: 14:10 - Prioridad: 1
  Platillos: 1 Taco de Pollo
---
Pedido #103 - Cliente: Juan Pérez - Hora: 14:15 - Prioridad: 1
  Platillos: 1 Quesadilla de Pollo
---
Pedido #105 - Cliente: María García - Hora: 14:30 - Prioridad: 2
  Platillos: 2 Tacos de Carnitas, 1 Agua de Horchata
---
Pedido #107 - Cliente: Luis Martínez - Hora: 14:40 - Prioridad: 2
  Platillos: 2 Quesadillas, 2 Aguas de Horchata
---
Total de pedidos: 5
Presione Enter para continuar...
```

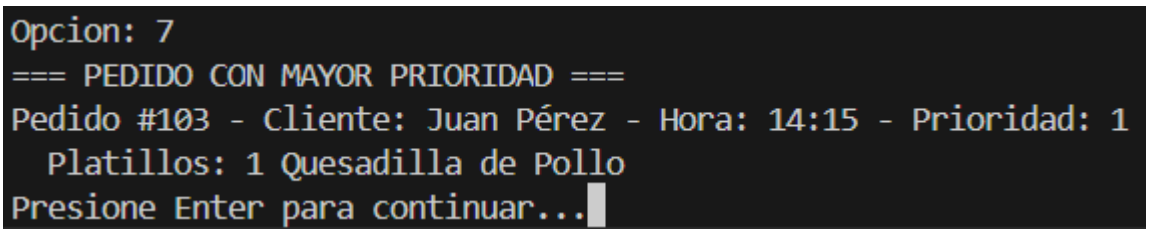
Ver secuencia de inserción (Pre-Orden)

```
Opcion: 5
=== SECUENCIA DE INSERCION DE PEDIDOS (PRE-Orden) ===
Pedido #105 - Cliente: María García - Hora: 14:30 - Prioridad: 2
  Platillos: 2 Tacos de Carnitas, 1 Agua de Horchata
---
Pedido #103 - Cliente: Juan Pérez - Hora: 14:15 - Prioridad: 1
  Platillos: 1 Quesadilla de Pollo
---
Pedido #102 - Cliente: Ana Rodríguez - Hora: 14:10 - Prioridad: 1
  Platillos: 1 Taco de Pollo
---
Pedido #1 - Cliente: Jesus Alberto - Hora: 12:41 - Prioridad: 2
  Platillos: Taco de Carnitas
---
Pedido #107 - Cliente: Luis Martínez - Hora: 14:40 - Prioridad: 2
  Platillos: 2 Quesadillas, 2 Aguas de Horchata
---
Presione Enter para continuar...
```

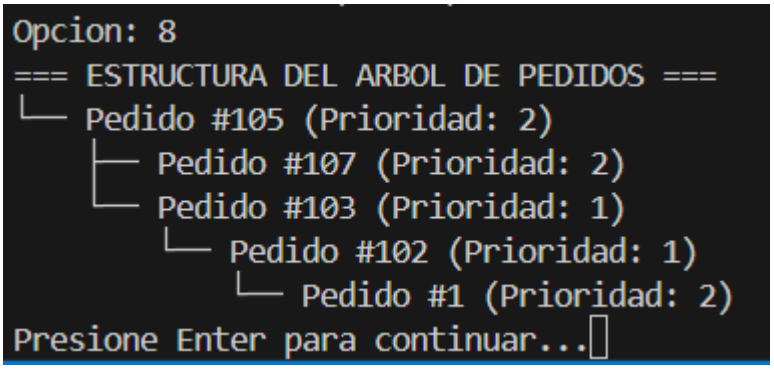
Generar reporte de fin de turno (Post-Orden)

```
Opcion: 6
=== REPORTE DE FIN DE TURNO (POST-Orden) ===
Pedido #1 - Cliente: Jesus Alberto - Hora: 12:41 - Prioridad: 2
  Platillos: Taco de Carnitas
---
Pedido #102 - Cliente: Ana Rodríguez - Hora: 14:10 - Prioridad: 1
  Platillos: 1 Taco de Pollo
---
Pedido #103 - Cliente: Juan Pérez - Hora: 14:15 - Prioridad: 1
  Platillos: 1 Quesadilla de Pollo
---
Pedido #107 - Cliente: Luis Martínez - Hora: 14:40 - Prioridad: 2
  Platillos: 2 Quesadillas, 2 Aguas de Horchata
---
Pedido #105 - Cliente: María García - Hora: 14:30 - Prioridad: 2
  Platillos: 2 Tacos de Carnitas, 1 Agua de Horchata
---
RESUMEN DEL TURNO:
Total de pedidos procesados: 5
Presione Enter para continuar...
```

Encontrar pedido con mayor prioridad

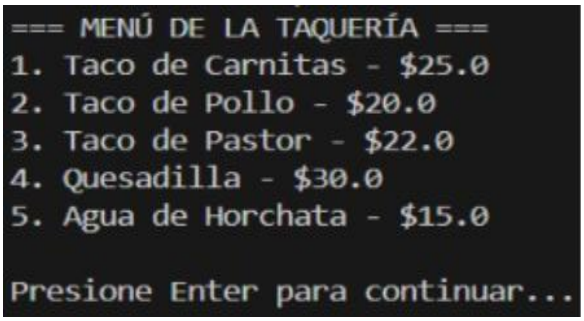


Mostrar estructura del árbol

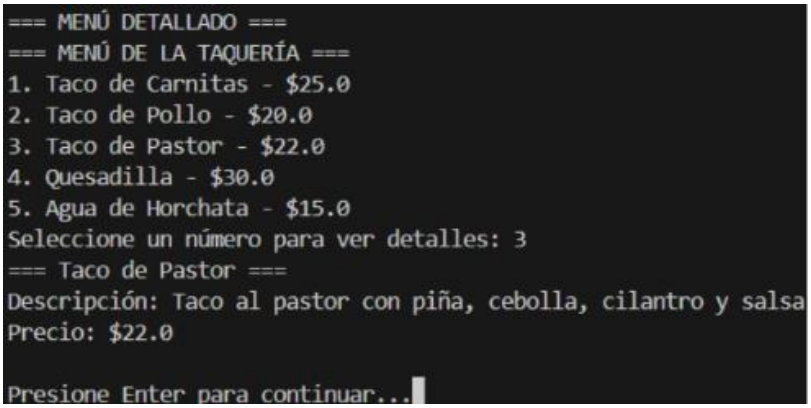


9. MENÚ DE LA TAQUERÍA (Lista Enlazada)

Mostrar menú



Ver detalles de platillo



10. ANÁLISIS RECURSIVO DE TAREAS

Mostrar el total de subtareas (Estas subtareas fueron añadidas dentro del código en tiempo de compilación y no dentro del tiempo de ejecución)

```
Seleccione una opcion: 6
=== CÁLCULOS RECURSIVOS ===

Tarea: Preparar Tacos de Carnitas
Tiempo total estimado: 88 minutos
Total de tareas/subtareas: 6

Estructura jerárquica:
├── [T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
│   ├── [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
│   │   ├── [T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
│   ├── [T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)
│   ├── [T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
│   ├── [T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)

Presione Enter para continuar...
```

11. DIVIDE Y VENCERÁS – ORDENAMIENTO

Mostrar las tareas en dos métodos distintos de ordenamiento (Merge Sort y Quick Sort)

```
Seleccione una opcion: 7
=== ORDENAMIENTO DE TAREAS ===

--- Ordenamiento por Prioridad (Merge Sort) ---
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)

--- Ordenamiento por Tiempo (Quick Sort) ---
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)

Presione Enter para continuar...
```

12. BALANCEO DE CARGA DE TRABAJO

Mostrar los diferentes roles de los empleados (También fueron creados en tiempo de compilación y no por medio del usuario en tiempo de ejecución)

```
Seleccione una opcion: 8
=== BALANCEO DE CARGA DE TRABAJO ===

Asignación de tareas por empleado:

EMP001 - Juan Carlos (Taquero Principal)
Tareas asignadas:
  - [T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)

EMP002 - María Elena (Ayudante de Cocina)
Tareas asignadas:
  - [T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
  - [T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)

EMP003 - Pedro López (Cajero)
Tareas asignadas:
  - [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
  - [T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)

EMP004 - Ana Sofía (Limpieza)
Tareas asignadas:
  - [T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)

Presione Enter para continuar...
```

13. ACCESO RÁPIDO CON HASHMAP

Mostrar información sobre tareas de forma específica y sobre los empleados de forma general

```
Seleccione una opcion: 9
=== ACCESO RÁPIDO CON HASHMAPS ===

--- Todas las Tareas ---
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2) | Empleado: Juan C
arlos
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2) | Empleado: María Elen
a
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1) | Empleado: Pedro López
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2) | Empleado: María Elena
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1) | Empleado: Pedro L
ópez
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1) | Empleado: Ana Sof
ía

--- Todos los Empleados ---
EMP001 - Juan Carlos (Taquero Principal) | Tareas: 1
EMP002 - María Elena (Ayudante de Cocina) | Tareas: 2
EMP003 - Pedro López (Cajero) | Tareas: 2
EMP004 - Ana Sofía (Limpieza) | Tareas: 1

Presione Enter para continuar...
```

14. BÚSQUEDAS AVANZADAS

Búsqueda binario por tiempo (requiere ordenamiento)

```
Seleccione una opcion: 10
=== BÚSQUEDAS ===
1. Búsqueda binaria por tiempo (requiere ordenamiento)
2. Búsqueda secuencial por nombre
3. Búsqueda secuencial por empleado
Seleccione tipo de búsqueda: 1
Ingrese tiempo a buscar (minutos): 45
Tarea encontrada: [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)

Presione Enter para continuar...
```

Búsqueda secuencial por nombre

```
Seleccione una opcion: 10
=== BÚSQUEDAS ===
1. Búsqueda binaria por tiempo (requiere ordenamiento)
2. Búsqueda secuencial por nombre
3. Búsqueda secuencial por empleado
Seleccione tipo de búsqueda: 2
Ingrese nombre o parte del nombre: Calentar tortillas
Tareas encontradas:
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)

Presione Enter para continuar...
```

Búsqueda secuencia por empleado

```
Seleccione una opcion: 10
=== BÚSQUEDAS ===
1. Búsqueda binaria por tiempo (requiere ordenamiento)
2. Búsqueda secuencial por nombre
3. Búsqueda secuencial por empleado
Seleccione tipo de búsqueda: 3
Ingrese nombre del empleado: Juan Carlos
Tareas del empleado:
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)

Presione Enter para continuar...
```

15. ANÁLISIS DE DEPENDENCIAS (GRAFOS)

```
Seleccione una opcion: 11
=== ANÁLISIS DE DEPENDENCIAS (GRAFOS) ===

--- Recorrido DFS ---
Visitando tarea: Preparar Tacos de Carnitas
Visitando tarea: Cocinar carnitas
Visitando tarea: Lavar y sazonar carne
Visitando tarea: Calentar tortillas
Visitando tarea: Cortar cebolla y cilantro
Visitando tarea: Preparar salsa verde

--- Orden Topológico (Orden de Ejecución) ---
Secuencia correcta de ejecución:
1. Preparar Tacos de Carnitas (T001)
2. Calentar tortillas (T001.2)
3. Cocinar carnitas (T001.1)
4. Lavar y sazonar carne (T001.1.1)
5. Preparar salsa verde (T001.4)
6. Cortar cebolla y cilantro (T001.3)

Presione Enter para continuar...
```

16. ESTADO GENERAL DEL SISTEMA

Muestra el estado general del sistema en cuanto a los apartados de clientes, platos e ingredientes

```
================================================
          ESTADO GENERAL DE LA TAQUERIA
================================================
CLIENTES:
   Proximo a atender: Cliente: Edwin

PLATOS:
   Proximo a lavar: Plato #2

INGREDIENTES:
   Proximo a usar: Pollo (Prioridad: 1)
================================================

Presione Enter para continuar...
```

17. DEMOSTRACIÓN COMPLETA DE ALGORITMOS

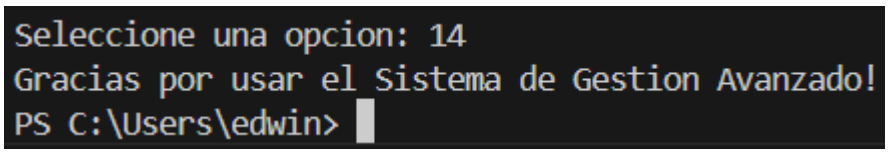
```
Seleccione una opcion: 13

======= DEMOSTRACION COMPLETA DE ALGORITMOS =======

1. RECURSIVIDAD:
=== CÁLCULOS RECURSIVOS ===

Tarea: Preparar Tacos de Carnitas
Tiempo total estimado: 88 minutos
Total de tareas/subtareas: 6

Estructura jerárquica:
├── [T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
│   ├── [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
│   │   ├── [T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
│   ├── [T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)
│   ├── [T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
│   ├── [T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)


2. DIVIDE Y VENCERAS:
=== ORDENAMIENTO DE TAREAS ===

--- Ordenamiento por Prioridad (Merge Sort) ---
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)

--- Ordenamiento por Tiempo (Quick Sort) ---
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
```

```
3. BALANCEO DE TRABAJO:
=== BALANCEO DE CARGA DE TRABAJO ===

Asignación de tareas por empleado:

EMP001 - Juan Carlos (Taquero Principal)
Tareas asignadas:
  - [T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)
  - [T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2)

EMP002 - María Elena (Ayudante de Cocina)
Tareas asignadas:
  - [T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
  - [T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)
  - [T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2)
  - [T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2)

EMP003 - Pedro López (Cajero)
Tareas asignadas:
  - [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
  - [T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)
  - [T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1)
  - [T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1)

EMP004 - Ana Sofía (Limpieza)
Tareas asignadas:
  - [T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
  - [T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1)
```

```
4. HASHMAPS:
=== ACCESO RÁPIDO CON HASHMAPS ===

--- Todas las Tareas ---
[T001.3] Cortar cebolla y cilantro (Tiempo: 8 min, Prioridad: 2) | Empleado: Juan C
arlos
[T001.4] Preparar salsa verde (Tiempo: 15 min, Prioridad: 2) | Empleado: María Elen
a
[T001.1] Cocinar carnitas (Tiempo: 45 min, Prioridad: 1) | Empleado: Pedro López
[T001.2] Calentar tortillas (Tiempo: 10 min, Prioridad: 2) | Empleado: María Elena
[T001] Preparar Tacos de Carnitas (Tiempo: 0 min, Prioridad: 1) | Empleado: Pedro L
ópez
[T001.1.1] Lavar y sazonar carne (Tiempo: 10 min, Prioridad: 1) | Empleado: Ana Sof
ía

--- Todos los Empleados ---
EMP001 - Juan Carlos (Taquero Principal) | Tareas: 2
EMP002 - María Elena (Ayudante de Cocina) | Tareas: 4
EMP003 - Pedro López (Cajero) | Tareas: 4
EMP004 - Ana Sofía (Limpieza) | Tareas: 2


5. ARBOL BINARIO DE PEDIDOS:
--- Recorrido In-Orden ---
=== PEDIDOS EN ORDEN CRONOLOGICO (IN-Orden) ===
Pedido #1 - Cliente: Jesus Alberto - Hora: 12:41 - Prioridad: 2
  Platillos: Taco de Carnitas
---
Pedido #102 - Cliente: Ana Rodríguez - Hora: 14:10 - Prioridad: 1
  Platillos: 1 Taco de Pollo
---
Pedido #103 - Cliente: Juan Pérez - Hora: 14:15 - Prioridad: 1
  Platillos: 1 Quesadilla de Pollo
---
Pedido #105 - Cliente: María García - Hora: 14:30 - Prioridad: 2
  Platillos: 2 Tacos de Carnitas, 1 Agua de Horchata
---
Pedido #107 - Cliente: Luis Martínez - Hora: 14:40 - Prioridad: 2
  Platillos: 2 Quesadillas, 2 Aguas de Horchata
---
Total de pedidos: 5
```

```
6. GRAFOS:
=== ANÁLISIS DE DEPENDENCIAS (GRAFOS) ===

--- Recorrido DFS ---
Visitando tarea: Preparar Tacos de Carnitas
Visitando tarea: Cocinar carnitas
Visitando tarea: Lavar y sazonar carne
Visitando tarea: Calentar tortillas
Visitando tarea: Cortar cebolla y cilantro
Visitando tarea: Preparar salsa verde

--- Orden Topológico (Orden de Ejecución) ---
Secuencia correcta de ejecución:
1. Preparar Tacos de Carnitas (T001)
2. Calentar tortillas (T001.2)
3. Cocinar carnitas (T001.1)
4. Lavar y sazonar carne (T001.1.1)
5. Preparar salsa verde (T001.4)
6. Cortar cebolla y cilantro (T001.3)

Demostracion completa finalizada!
Todos los algoritmos avanzados han sido ejecutados exitosamente.

Presione Enter para continuar...
```

18. SALIDA DEL PROGRAMA



19. CONCLUSION

Este proyecto permitió implementar un sistema integral de gestión que combina estructuras de datos fundamentales con algoritmos computacionales avanzados. Se logró una aplicación que simula operaciones reales de un restaurante, demostrando cómo diferentes estructuras optimizan operaciones específicas:

Estructuras implementadas exitosamente:

- Cola FIFO para clientes garantizando orden de atención
- Pila LIFO para platos optimizando flujo de limpieza
- Cola de prioridad para ingredientes minimizando desperdicio
- Lista enlazada para menú con acceso flexible
- Árbol binario para pedidos con búsquedas O(log n)
- HashMap para acceso O(1) a datos críticos
- Grafo para dependencias evitando conflictos de ejecución

Algoritmos avanzados integrados:

- Recursividad para cálculos jerárquicos complejos
- Divide y vencerás para ordenamiento eficiente y balanceo de carga
- Búsquedas optimizadas binarias y secuenciales
- Recorridos de árboles (In-Order, Pre-Order, Post-Order)
- Algoritmos de grafos (DFS, orden topológico)

Resultados del proyecto: El sistema final representa una solución escalable que integra múltiples paradigmas de programación y estructuras de datos, demostrando cómo la correcta selección e implementación de algoritmos puede resolver problemas complejos del mundo real. La aplicación logró optimizaciones significativas: búsquedas en tiempo logarítmico, acceso instantáneo a datos frecuentes, y gestión automática de dependencias complejas.

Esta implementación evidencia la importancia crítica de comprender las características de rendimiento de cada estructura de datos para construir sistemas eficientes y robustos que puedan escalar en entornos de producción real.