**Universidad Técnica Particular de Loja**



**Carrera:**

Ingeniería en Ciencias de la Computación

**Asignatura:**

Introducción a la programación

**Docente:**

Pedro Daniel Irene Robalino

**Estudiantes:**

* Quizhpe Cristhian
* Vargas Pedro

**Tema Proyecto:**

Sistema de gestión de buses UTPL

1. **Problema**

**TEMA DE PROYECTOS NRO. 2:** Sistema de gestión de buses UTPL

Desarrollar un sistema de gestión que permita monitorear, registrar y optimizar los recorridos de los autobuses de la Universidad Técnica Particular de Loja (UTPL) en toda la ciudad de Loja. El sistema ayudará a administrar eficientemente las paradas, horarios y rutas de los autobuses para mejorar la calidad del servicio ofrecido a los usuarios.

**Características a considerar:**

• **Registro de Paradas:** Crear una estructura de datos (arreglo, lista, etc.) para

almacenar información sobre las paradas de autobús, incluyendo nombre de la

parada, ubicación y horarios.

• **Registro de Horarios:** Desarrollar métodos o funciones para registrar los horarios

de salida y llegada de los autobuses en cada parada. Estos horarios deben poder ser consultados por los usuarios.

• **Gestión de Rutas:** Implementar un mecanismo para definir y administrar las

rutas que seguirán los autobuses. Esto puede lograrse utilizando arreglos u otras

estructuras para representar las rutas.

• **Optimización de Rutas:** Crear funciones que permitan optimizar las rutas de los

autobuses. Esto podría incluir la minimización de tiempos de viaje y la

maximización de la eficiencia en la cobertura de paradas.

• **Interfaz de Usuario:** Desarrollar una interfaz de consola para que los usuarios

puedan acceder a la información sobre las paradas, horarios y rutas de los

autobuses.

1. **Análisis**

|  |  |  |
| --- | --- | --- |
| Entrada | Proceso | Salida |
| Horarios de las líneas de buses.  Rutas y paradas de los Buses.  Ubicación de las paradas. | Creación de un menú para poder ver lo que el usuario quiera saber.  En esta interfaz el usuario podrá elegir si ver los horarios buscar líneas de buses o ver las rutas de todas las líneas. | Horarios  Rutas  Buscador de líneas por paradas  Rutas optimizadas |

* 1. **Datos de entrada**

**Paradas de autobús:**

* Nombre de la parada
* Ubicación (coordenadas o dirección)
* Horarios

**Horarios de autobuses:**

* Hora de cada línea de buses.

**Rutas de los autobuses:**

* Nombre de la ruta
* Lista de paradas en el orden de recorrido

**Solicitudes de usuarios:**

* Consulta de paradas
* Consulta de horarios
* Consulta de rutas disponibles
  1. **Proceso**

**Registro y almacenamiento de datos:**

* Ingreso de paradas con su información
* Registro de horarios de los autobuses
* Definición y almacenamiento de rutas

**Gestión de información:**

* Búsqueda de paradas según nombre
* Consulta
* Visualización de rutas completas

**Optimización de rutas:**

* Ajuste de rutas según demanda y tráfico

**Interfaz de usuario:**

* Implementación de un menú de opciones
* Visualización de información solicitada
* Interacción para consultar paradas, horarios y rutas
  1. **Datos de salida**

**Información de paradas:**

* Lista de paradas con su ubicación y horarios

**Información de horarios:**

* Hora de salida y llegada de cada autobús en cada parada

**Detalles de rutas:**

* Rutas disponibles y paradas en su recorrido
* Por donde pasan

**Interfaz de usuario funcional:**

* Menú de navegación para consultar información
* Resultados claros y organizados para el usuario

1. **Diseño de la Solución - En java** 
   * 1. **Importaciones:**

* **java.io.BufferedReader:** Esta clase se utiliza para leer datos de manera eficiente desde un archivo o una entrada de texto. Bufferiza los datos que lee, lo que significa que lee grandes bloques de datos a la vez y luego los procesa, lo que mejora el rendimiento cuando se manejan archivos grandes. Se utiliza comúnmente junto con FileReader para leer archivos línea por línea.
* **java.io.FileReader:** Es una clase diseñada para leer archivos de texto, uno por uno, carácter por carácter. Aunque es funcional, no es tan eficiente como BufferedReader. FileReader es una opción básica cuando solo necesitas leer un archivo sin la necesidad de procesamiento avanzado, pero se usa mejor cuando se combina con BufferedReader para optimizar la lectura.
* **java.io.IOException:** Esta es una excepción que se lanza cuando ocurre un error de entrada o salida. Por ejemplo, cuando un archivo no se puede leer o escribir debido a problemas con el sistema de archivos (como si el archivo no existe o no se puede acceder). Es una clase fundamental en Java para manejar errores relacionados con operaciones de E/S (entrada/salida).
* **java.util.ArrayList:** ArrayList es una implementación de la interfaz List que usa un arreglo dinámico para almacenar los elementos. Esto significa que su tamaño puede cambiar según se agreguen o eliminen elementos. Es útil cuando necesitas una lista flexible que permite acceder a los elementos de forma rápida (por índice), pero tiene una sobrecarga de rendimientoal insertar o eliminar elementos en posiciones intermedias.
* **java.util.List:** List es una interfaz que define una colección ordenada de elementos. A diferencia de un conjunto (Set), permite elementos duplicados y mantiene el orden de inserción. Las implementaciones más comunes de List son ArrayList y LinkedList. Se utiliza cuando se requiere una lista en la que se puedan almacenar elementos y acceder a ellos mediante un índice.
* **java.util.Scanner:** Nos sirve para poder leer los datos ingresados por el usuario, en el programa para dar valores a variables, arreglos, matrices o listas.
  + 1. **Programa Main:**



* + 1. **Clase de Horarios:**



* + 1. **Clase Optimizar Rutas:**





* + 1. **Clase Rutas y Paradas:**

1. **Estructuras Utilizadas**
   1. **Main:** Convocamos a todas las clases, luego usamos un while true, para hacer un ciclo infinito hasta que el usuario lo decida, y un try-catch, para reconocer si hay algún error, ya que usamos un switch para leer la opción seleccionada.
   2. **Horarios:** Usamos un método para leer el archivo y si no está bien la dirección del archivo se presentará un mensaje de error, una función con array list, para almacenar datos de manera dinámica, con el try ponemos un while para asignar los datos a la lista creada.
   3. **Optimizar Rutas:** Lee un archivo CSV con información de rutas de autobuses, extrae y almacena las paradas, optimiza la ruta minimizando distancias entre ellas y muestra los resultados en formato tabular en la consola; para ello, usa métodos que leen el archivo (leerArchivoCSV), calculan la distancia geográfica entre paradas (calcularDistancia), buscan información de una parada (buscarParada) y organizan una ruta optimizada (optimizarRuta), empleando estructuras como listas (List) para almacenar los datos y conjuntos (Set) para evitar repetir paradas ya visitadas. Usamos 4 funciones y un arreglo para presentar el resultado.
   4. **Buscar Paradas:** Creamos una función y un método, el método para buscar las líneas que correspondan con la información dada, y la función hace que se lea el archivo con toda la información de paradas, para después compararla.
   5. **Paradas y Rutas:** Creamos como en las anteriores una función para leer el archivo y finalmente un método que lo presente en pantalla, eliminando los espacios en blanco para no consumir mucho espacio.