Plan de Proyecto: Mini-Uber en C

1. Idea Central del Proyecto

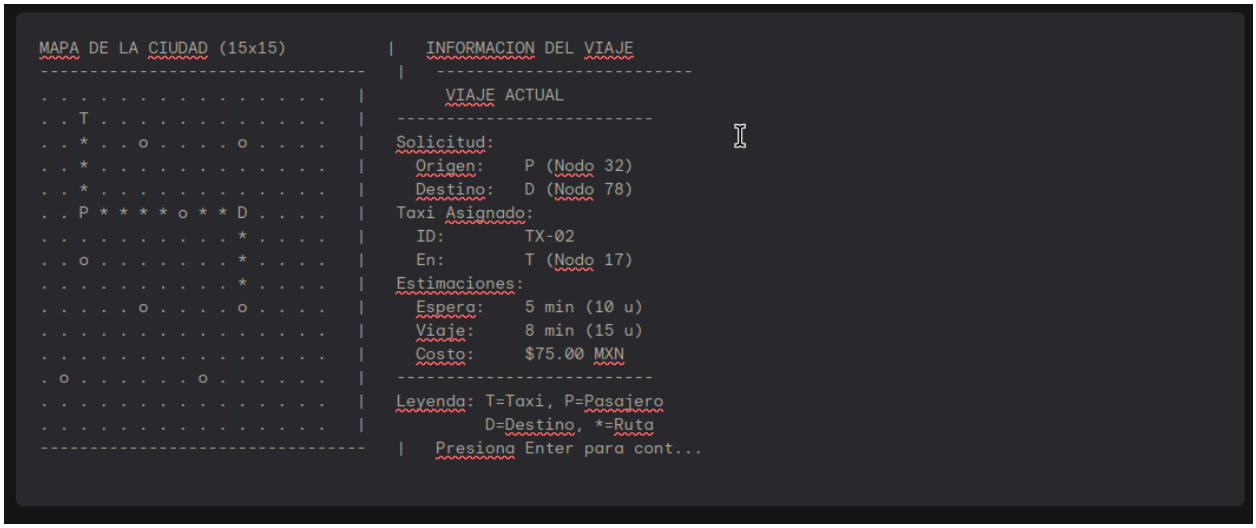
Desarrollar una aplicación en C que simule una versión simplificada de un servicio de taxis tipo Uber. El sistema utilizará un mapa representado como un grafo, donde los nodos son intersecciones y las aristas son calles. Se cargará una cola de solicitudes de viaje y, para cada una, se asignará el taxi más cercano utilizando el algoritmo de Dijkstra. La interacción será por consola, procesando una solicitud cada vez que el usuario presione "Enter" y mostrando información relevante del viaje junto con una representación visual del mapa.

2. Decisiones Clave y Características

- **Lenguaje de Programación:** C.
- **Representación del Mapa:**
    - Grafo no dirigido.
    - Nodos: Hasta 100, representan intersecciones o puntos clave.
    - Aristas: Calles, con peso igual a la distancia.
    - Implementación del Grafo: Matriz de adyacencia (ej. int mapa[100][100]).
- **Taxis:**
    - Número definido de taxis.
    - Ubicaciones iniciales predefinidas.
    - Movimiento: Solo se mueven cuando atienden una solicitud. Después de completar un viaje, permanecen en el nodo destino del pasajero hasta una nueva asignación.
    - Estado: Disponible / Ocupado.
- **Algoritmo Principal:** Dijkstra para calcular la ruta más corta (distancia).
    - Se usará para encontrar el taxi más cercano al origen del pasajero.
    - Se usará para calcular la ruta del pasajero desde su origen hasta su destino.
    - Enfoque de implementación: O(V^2), adecuado para 100 nodos.
- **Entrada de Datos:**
    - Mapa (matriz de adyacencia): Desde un archivo mapa.txt.
    - Taxis (ID, ubicación inicial): Desde un archivo taxis.txt.
    - Solicitudes de Viaje (nodo origen, nodo destino): Desde un archivo solicitudes.txt, cargadas a una cola.
- **Interacción y Salida:**
    - El programa procesará una solicitud de la cola cada vez que el usuario presione "Enter".
    - Se mostrará en consola una representación visual del mapa (cuadrícula de 15x15) y, a su lado, información concisa del viaje.
        - **Mapa Visual:**
            - T: Ubicación actual del taxi asignado.
            - P: Nodo origen del pasajero.
            - D: Nodo destino del pasajero.

        - *: Camino/ruta del taxi al pasajero y del pasajero al destino.
        - o (u otro símbolo): Otros nodos del mapa.
        - .: Espacio vacío o calles no destacadas en la ruta actual.
    - **Información del Viaje:**
        - Origen y destino de la solicitud.
        - ID del taxi asignado y su ubicación.
        - Tiempo de espera estimado (basado en distancia del taxi al pasajero).
        - Duración estimada del viaje (basado en distancia del origen al destino del pasajero).
        - Costo total estimado del servicio.
- **Cálculos:**
    - **Precio:** Basado en la distancia total recorrida por el taxi (distancia para recoger al pasajero + distancia del viaje del pasajero). Se puede definir una tarifa por unidad de distancia.
    - **Tiempo:** Directamente proporcional a la distancia (ej. 1 unidad de distancia = 1 minuto, o alguna otra conversión).

3. Ejemplo de Visualización en Consola (Mockup)



(Nota: Para el mapa, los IDs de nodo (0-99) necesitarán una forma de mapearse a coordenadas (fila, columna) de la cuadrícula 15x15, o bien, la representación del mapa se basará en destacar los nodos clave y sus conexiones en lugar de una cuadrícula geográfica exacta).

4. Plan de Desarrollo por Pasos

Aquí tienes una secuencia sugerida para abordar la programación:

1. **Paso 1: Definición de Estructuras de Datos Fundamentales (en C)**

- Crea las struct para:
  - Mapa: Contendrá el número de nodos y la matriz de adyacencia.
  - Taxi: Incluirá ID, ubicación actual (nodo), y estado (disponible/ocupado).
  - SolicitudViaje: Contendrá nodo origen y nodo destino.
- Define constantes importantes (ej. MAX_NODOS, INFINITO para distancias).

2. **Paso 2: Implementación de la Cola de Solicitudes**
   - Decide cómo implementarás la cola (ej. array simple con índices de cabeza/cola, o lista enlazada).
   - Crea funciones para encolar (enqueue) y desencolar (dequeue) SolicitudViaje.
3. **Paso 3: Carga de Datos Iniciales desde Archivos .txt**
   - Función cargar_mapa(Mapa *mapa, const char* nombre_archivo): Lee mapa.txt y llena la matriz de adyacencia.
   - Función cargar_taxis(Taxi taxis[], int *num_taxis, const char* nombre_archivo): Lee taxis.txt y llena un array de taxis con sus datos iniciales.
   - Función cargar_solicitudes(ColaSolicitudes *cola, const char* nombre_archivo): Lee solicitudes.txt y encola todas las solicitudes.
4. **Paso 4: Implementación del Algoritmo de Dijkstra**
   - Función dijkstra(const Mapa *mapa, int nodo_inicio, int distancias[], int predecesores[]).
     - distancias[]: Array para almacenar la distancia más corta desde nodo_inicio a cada otro nodo.
     - predecesores[]: Array para reconstruir la ruta (almacena el nodo anterior en la ruta más corta).
     - Implementa la lógica O(V^2): encontrar el nodo no visitado más cercano, actualizar distancias a sus vecinos.
5. **Paso 5: Lógica Principal de Simulación y Asignación**
   - Función principal que se ejecuta en bucle (o cada vez que se presiona Enter).
   - Dentro del bucle:
     - Desencola una SolicitudViaje. Si la cola está vacía, termina o espera.
     - Para cada Taxi disponible:
       - Ejecuta dijkstra desde taxi.ubicacion_actual_nodo hasta solicitud.nodo_origen.
       - Guarda la distancia (tiempo de espera para ese taxi).
     - Selecciona el taxi con la menor distancia/tiempo de espera. Si no hay taxis disponibles, maneja el caso (ej. solicitud en espera, mensaje al usuario).
     - Si se asigna un taxi:
       - Marca el taxi como ocupado.
       - Calcula la ruta del pasajero: ejecuta dijkstra desde solicitud.nodo_origen hasta solicitud.nodo_destino.
       - Calcula el costo total (basado en distancia taxi->pasajero + distancia pasajero->destino).
       - Calcula el tiempo total del viaje del pasajero.
       - Prepara los datos para la visualización.

6. **Paso 6: Desarrollo de la Visualización en Consola**
    - Función mostrar_estado_viaje(const Mapa *mapa, const SolicitudViaje *solicitud, const Taxi *taxi_asignado, int dist_taxi_pasajero, int dist_viaje_pasajero, const int ruta_taxi_pasajero[], const int ruta_pasajero_destino[]).
        - Esta función será la más compleja visualmente.
        - **Mapeo de Nodos a Cuadrícula (si aplica):** Si quieres la cuadrícula 15x15, necesitas una lógica para convertir IDs de nodo a coordenadas (fila, columna) o una forma de representar el mapa que destaque los nodos clave.
        - **Dibujo del Mapa:** Itera para imprimir cada fila del mapa. Decide qué carácter mostrar (T, P, D, *, o, .) para cada celda basado en los datos del viaje actual y las rutas.
        - **Panel de Información:** Imprime la información del viaje de forma estructurada al lado del mapa.
        - Usa printf cuidadosamente para alinear el texto.
7. **Paso 7: Integración y Pruebas Exhaustivas**
    - Une todas las partes en tu función main.
    - Crea archivos mapa.txt, taxis.txt y solicitudes.txt con diversos escenarios:
        - Mapa pequeño, mapa más conectado.
        - Pocos taxis, muchos taxis.
        - Solicitudes a nodos cercanos, lejanos, inexistentes (para probar manejo de errores).
        - Casos donde no hay taxis disponibles.
    - Depura y refina la lógica y la visualización.
8. **Paso 8 (Opcional): Mejoras y Funcionalidades Adicionales**
    - Permitir al usuario añadir nuevas solicitudes en tiempo de ejecución.
    - Simular movimiento aleatorio de taxis disponibles (más complejo).
    - Guardar un historial de viajes.
    - Mejorar la interfaz de usuario en consola con más opciones.

Este plan debería darte una buena hoja de ruta. ¡Mucho ánimo con tu proyecto!